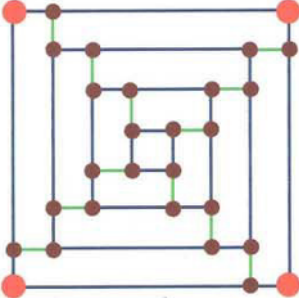
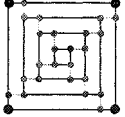


PLANAR GRAPH DRAWING



Takao Nishizeki
Md. Saidur Rahman



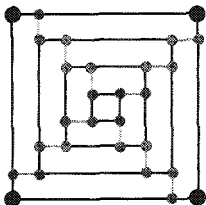
PLANAR GRAPH
DRAWING

LECTURE NOTES SERIES ON COMPUTING

Editor-in-Chief: D T Lee (*Academia Sinica, Taiwan*)

Published

- Vol. 1: Computing in Euclidean Geometry
Eds. D-Z Du & F Hwang
- Vol. 2: Algorithmic Aspects of VLSI Layout
Eds. D T Lee & M Sarrafzadeh
- Vol. 3: String Searching Algorithms
G A Stephen
- Vol. 4: Computing in Euclidean Geometry (Second Edition)
Eds. D-Z Du & F Hwang
- Vol. 5: Proceedings of the Conference on
Parallel Symbolic Computation — PASCO '94
Ed. H Hong
- Vol. 6: VLSI Physical Design Automation: Theory and Practice
S M Sait & H Youssef
- Vol. 7: Algorithms: Design Techniques and Analysis
Ed. M H Alsuwaiyel
- Vol. 8: Computer Mathematics
Proceedings of the Fourth Asian Symposium (ASCM 2000)
Eds. X-S Gao & D Wang
- Vol. 9: Computer Mathematics
Proceedings of the Fifth Asian Symposium (ASCM 2001)
Eds. K Yokoyama & K Shirayanagi
- Vol. 10: Computer Mathematics
Proceedings of the Sixth Asian Symposium (ASCM 2003)
Eds. Z Li & W Sit
- Vol. 11: Geometric Computation
Eds. F Chen & D Wang

PLANAR  GRAPH
DRAWING

Takao Nishizeki

Tohoku University, Japan

Md. Saidur Rahman

Bangladesh University of Engineering and Technology, Bangladesh

TEAM LING - LIVE, HELP, LEARN, NOT COST, and GENUINE !
 World Scientific

NEW JERSEY • LONDON • SINGAPORE • BEIJING • SHANGHAI • HONG KONG • TAIPEI • CHENNAI

Published by

World Scientific Publishing Co. Pte. Ltd.

5 Toh Tuck Link, Singapore 596224

USA office: 27 Warren Street, Suite 401–402, Hackensack, NJ 07601

UK office: 57 Shelton Street, Covent Garden, London WC2H 9HE

British Library Cataloguing-in-Publication Data

A catalogue record for this book is available from the British Library.

PLANAR GRAPH DRAWING

Lecture Notes Series on Computing — Vol. 12

Copyright © 2004 by World Scientific Publishing Co. Pte. Ltd.

All rights reserved. This book, or parts thereof, may not be reproduced in any form or by any means, electronic or mechanical, including photocopying, recording or any information storage and retrieval system now known or to be invented, without written permission from the Publisher.

For photocopying of material in this volume, please pay a copying fee through the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, USA. In this case permission to photocopy is not required from the publisher.

ISBN 981-256-033-5

TEAM LING - LIVE, INFORMATIVE, NON-COST AND GENUINE !

Printed by FuIsland Offset Printing (S) Pte Ltd, Singapore

Preface

Motivation

This book deals with theories and algorithms for drawing planar graphs. Graph drawing has appeared as a lively area in computer science due to its applications in almost all branches of science and technology. Many researchers have concentrated their attention on drawing planar graphs for the following reasons:

- drawings of planar graphs have no edge crossings, and look nice;
- drawings of planar graphs have practical applications in VLSI floorplanning and routing, architectural floorplanning, displaying RNA structures in bioinformatics, etc.; and
- algorithms for drawing planar graphs can be successfully used for drawing a nonplanar graph by transforming it into a similar planar graph.

During the last two decades numerous results have been published on drawing planar graphs. For example, in 1990 it was shown that every planar graph of n vertices has a straight-line drawing on a grid of area $O(n^2)$. This result solved the open question for about four decades whether a planar graph has a straight line drawing on a grid of a polynomial area. Many algorithms have been developed to produce drawings of planar graphs with different styles to fulfill different application needs. While developing these algorithms, many elegant theories on the properties of planar graphs have been discovered, which have applications in solving problems on planar graphs other than graph drawing problems. For example, Schnyder introduced a “realizer” to produce straight line drawings of planar graphs, but later a realizer is used to solve the “independent spanning tree problem”

TEAM LING - LIVE, INFORMATIVE, NON-COST AND GENUINE !

of a certain class of planar graphs. A “canonical ordering” which was introduced by de Fraysseix *et al.* is later used to solve a “graph partitioning problem.” On the other hand, many established graph theoretic results have been successfully used to solve graph drawing problems. For example, the problem of orthogonal drawings of plane graphs with the minimum number of bends is solved by reducing the problem to a network flow problem.

Recently, it appeared to us that a systematic and organized book containing these many results on planar graph drawings can help students and researchers of computer science to apply the results in appropriate areas. For example, we observed that people working with VLSI floorplanning by rectangular dual did not notice Thomassen’s result on rectangular drawings of plane graphs. In our opinion the theory and algorithms are complementary to each other in the research of planar graph drawings. We have thus tried to include in the book most of the important theorems and algorithms that are currently known for planar graph drawing. Furthermore, we have tried to provide constructive proofs for theorems, from which algorithms immediately follow.

Organization of the Book

This book is organized as follows.

Chapter 1 is the introduction of graph drawing. It introduces different drawing styles of planar graphs, and presents properties of graph drawing and some applications of graph drawing.

Chapter 2 deals with graph theoretic fundamentals.

Chapter 3 provides algorithmic fundamentals.

Chapter 4 describes straight line drawings of planar graphs on an integer grid. We present both the famous results of de Fraysseix *et al.* and Schnyder on straight-line drawings of planar graphs in this chapter.

Chapter 5 focuses on convex drawings of planar graphs. In this chapter we present the results of Tutte, Thomassen and Chiba *et al.* on characterization of planar graphs with convex drawings. We also include recent results on convex grid drawings by Kant and Chrobak, and Miura *et al.*

Chapter 6 deals with rectangular drawings of planar graphs. In this chapter we present a technique of Miura *et al.* for reducing a rectangular drawing problem to a matching problem. We present Thomassen’s result on rectangular drawings of plane graphs, and describe a generalization of

Thomassen's result given by Rahman *et al.* We also present a necessary and sufficient condition for a planar graph to have a rectangular drawing. Several algorithms for rectangular drawings are included in this chapter.

Chapter 7 deals with box-rectangular drawings of plane graphs. In this chapter we present a necessary and sufficient condition for a plane graph to have a box-rectangular drawing, and present a linear algorithm for finding a box-rectangular drawing of a plane graph.

Chapter 8 discusses orthogonal drawings of plane graphs. In this chapter we present the results of Tamassia for solving the problem of finding a bend-minimum orthogonal drawing of a plane graph by reducing the problem to a network flow problem. We explain a linear algorithm for finding a bend-minimum orthogonal drawing of a triconnected cubic plane graph. In this chapter we also include a necessary and sufficient condition for a plane graph to have a no-bend orthogonal drawing.

Chapter 9 deals with octagonal drawings of plane graphs with prescribed face areas. In this chapter we show that every "good slicing graph" has an octagonal drawing where each face is drawn as a rectilinear polygon of at most eight corners and the area of each inner face is equal to a prescribed value. We also present a linear algorithm for finding such a drawing.

Appendix A presents planarity testing and embedding algorithms.

Use of the book

This book is suitable for use in advanced undergraduate and graduate level courses on Algorithms, Graph Theory, Graph Drawing, Information Visualization, and Computational Geometry. This book will serve as a good reference book for the researchers in the field of graph drawing. In this book many fundamental graph drawing algorithms are described with illustrations, which are helpful for software developers, particularly in the area of information visualization, VLSI design and CAD.

Acknowledgments

It is a pleasure to record our gratitude to those to whom we are indebted, directly or indirectly, in writing this book. A book as this one owes a great deal, of course, to many previous researchers and writers. Without trying to be complete, we would like to mention the books of Nishizeki and Chiba, Di Battista *et al.*, and the tutorial edited by Kaufmann and Wagner. We

TEAM LING - LIVE, INFORMATIVE, NON-COST and GENUINE !

also acknowledge T. C. Biedl, M. Chrobak, H. de Fraysseix, A. Garg, G. Kant, R. Tamassia, C. Thomassen, J. Pach, T. H. Payne, R. Pollack, W. Schnyder and W. T. Tutte; some of their results are covered in this book.

A substantial part of this book is based on a series of the authors' own investigations. We wish to thank the coauthors of our joint papers: Norishige Chiba, Shubhashis Ghosh, Hiroki Haga, Kazuyuki Miura, Shin-ichi Nakano, and Mahmuda Naznin.

This book is based on our graph drawing research project supported by Japan Society for the Promotion of Science (JSPS) and Tohoku University. We also acknowledge Bangladesh University of Engineering and Technology (BUET) for providing the second author necessary leave to write this book. We thankfully mention the names of our colleagues Yasuhito Asano and Xiao Zhou at Tohoku university and Md. Shamsul Alam, Mohammad Kaykobad and Md. Abul Kashem Mia at BUET for their encouraging comments on the book. The second author wishes to thank his parents for supporting him throughout his life and for encouraging him to stay in a foreign country for the sake of writing this book.

We must thank the series editor D. T. Lee for his positive decision for publishing the book from World Scientific Publishing Co. We also thank Yubing Zhai and Steven Patt of World Scientific Publishing Co. for their helpful cooperation.

Finally, we would like to thank our wives Yuko Nishizeki and Mossa. Anisa Khatun for their patience and constant support.

Takao Nishizeki
Md. Saidur Rahman

Contents

<i>Preface</i>	v
1. Graph Drawing	1
1.1 Introduction	1
1.2 Historical Background	2
1.3 Drawing Styles	3
1.3.1 Planar Drawing	4
1.3.2 Polyline Drawing	5
1.3.3 Straight Line Drawing	5
1.3.4 Convex Drawing	6
1.3.5 Orthogonal Drawing	6
1.3.6 Box-Orthogonal Drawing	7
1.3.7 Rectangular Drawing	8
1.3.8 Box-Rectangular Drawing	8
1.3.9 Grid Drawing	8
1.3.10 Visibility Drawing	9
1.4 Properties of Drawings	10
1.5 Applications of Graph Drawing	11
1.5.1 Floorplanning	12
1.5.2 VLSI Layout	13
1.5.3 Software Engineering	14
1.5.4 Simulating Molecular Structures	15
1.6 Scope of This Book	15
2. Graph Theoretic Foundations	19
2.1 Basic Terminology	19

TEAM LING - LIVE, INFORMATIVE, NON-COST and GENUINE !

2.1.1	Graphs and Multigraphs	19
2.1.2	Subgraphs	20
2.1.3	Paths and Cycles	21
2.1.4	Chains	21
2.1.5	Connectivity	22
2.1.6	Trees and Forests	22
2.1.7	Complete Graphs	23
2.1.8	Bipartite Graphs	24
2.1.9	Subdivisions	24
2.2	Planar Graphs	24
2.2.1	Plane Graphs	26
2.2.2	Euler's Formula	29
2.2.3	Dual Graph	30
2.3	Bibliographic Notes	31
3.	Algorithmic Foundations	33
3.1	What is an Algorithm?	33
3.2	Machine Model and Complexity	34
3.2.1	The $O(\)$ notation	34
3.2.2	Polynomial Algorithms	35
3.2.3	NP-Complete Problems	35
3.3	Data Structures and Graph Representation	36
3.4	Exploring a Graph	38
3.4.1	Depth-First Search	38
3.4.2	Breadth-First Search	39
3.5	Data Structures for Plane Graphs	42
3.6	Bibliographic Notes	44
4.	Straight Line Drawing	45
4.1	Introduction	45
4.2	Shift Method	46
4.2.1	Canonical Ordering	46
4.2.2	Shift Algorithm	50
4.2.3	Linear-Time Implementation	54
4.3	Realizer Method	58
4.3.1	Barycentric Representation	58
4.3.2	Schnyder Labeling	62
4.3.3	Realizer	66

4.3.4	Drawing Algorithm with Realizer	69
4.4	Compact Grid Drawing	72
4.4.1	Four-Canonical Ordering	74
4.4.2	Algorithm Four-Connected-Draw	77
4.4.3	Drawing G'	79
4.5	Bibliographic Notes	87
5.	Convex Drawing	89
5.1	Introduction	89
5.2	Convex Drawing	90
5.3	Convex Testing	94
5.3.1	Definitions	95
5.3.2	Condition II	98
5.3.3	Testing Algorithm	101
5.4	Convex Grid Drawings of 3-Connected Plane Graphs	105
5.4.1	Canonical Decomposition	105
5.4.2	Algorithm for Convex Grid Drawing	110
5.5	Convex Grid Drawings of 4-Connected Plane Graphs	117
5.5.1	Four-Canonical Decomposition	117
5.5.2	Algorithm	119
5.5.2.1	How to Compute x -Coordinates	119
5.5.2.2	How to Compute y -Coordinates	123
5.6	Bibliographic Notes	127
6.	Rectangular Drawing	129
6.1	Introduction	129
6.2	Rectangular Drawing and Matching	130
6.3	Linear Algorithm for Rectangular Drawings of Plane Graphs	135
6.3.1	Thomassen's Theorem	135
6.3.2	Sufficiency	137
6.3.3	Rectangular Drawing Algorithm	152
6.3.4	Rectangular Grid Drawing	156
6.4	Rectangular Drawings without Designated Corners	159
6.5	Rectangular Drawings of Planar Graphs	161
6.5.1	Case for a Subdivision of a Planar 3-connected Cubic Graph	163
6.5.2	The Other Case	169
6.6	Bibliographic Notes	173

7.	Box-Rectangular Drawing	175
7.1	Introduction	175
7.2	Preliminaries	175
7.3	Box-Rectangular Drawings with Designated Corner Boxes .	178
7.4	Box-Rectangular Drawings without Designated Corners . .	182
7.4.1	Box-Rectangular Drawings of G with $\Delta \leq 3$	183
7.4.2	Box-Rectangular Drawings of G with $\Delta \geq 4$	193
7.5	Bibliographic Notes	195
8.	Orthogonal Drawing	197
8.1	Introduction	197
8.2	Orthogonal Drawing and Network Flow	198
8.2.1	Orthogonal Representation	198
8.2.2	Flow Network	201
8.2.3	Finding Bend-Optimal Drawing	202
8.3	Linear Algorithm for Bend-Optimal Drawing	208
8.3.1	Genealogical Tree	211
8.3.2	Assignment and Labeling	213
8.3.3	Feasible Orthogonal Drawing	217
8.3.4	Algorithm	224
8.4	Orthogonal Grid Drawing	227
8.5	Orthogonal Drawings without Bends	229
8.6	Bibliographic Notes	231
9.	Octagonal Drawing	233
9.1	Introduction	233
9.2	Good Slicing Graphs	235
9.3	Octagonal Drawing	238
9.3.1	Algorithm Octagonal-Draw	239
9.3.2	Embedding a Slicing Path	243
9.3.3	Correctness and Time Complexity	249
9.4	Bibliographic Notes	250
	Appendix A Planar Embedding	253
A.1	Introduction	253
A.2	Planarity Testing	254
A.2.1	st -Numbering	255
A.2.2	Bush Form and PQ -Tree	259

A.2.3 Planarity Testing Algorithm	263
A.3 Finding Planar Embedding	266
A.3.1 Algorithm for Extending A_u into Adj	267
A.3.2 Algorithm for Constructing A_u	271
A.4 Bibliographic Notes	277
<i>Bibliography</i>	281
<i>Index</i>	291

This page intentionally left blank

Chapter 1

Graph Drawing

1.1 Introduction

A graph consists of a set of vertices and a set of edges, each joining two vertices. A drawing of a graph can be thought of as a diagram consisting of a collection of objects corresponding to the vertices of the graph together with some line segments corresponding to the edges connecting the objects. People are using diagrams from ancient time to represent abstract things like ideas, concepts, etc. as well as concrete things like maps, structures of machines, etc. A diagram of a computer network is depicted in Fig. 1.1, where each component of the network is drawn by a small circle and a connection between a pair of components is drawn by a straight line segment. We can consider this diagram as a drawing of a graph which represents information regarding interconnections of the computer network. The vertices of the graph represent components of the network and are drawn as small circles in the diagram, while the edges of the graph represent interconnection relationship among the components and are drawn by straight line segments. A graph may be used to represent any information, like interconnection information of a computer network, which can be modeled as objects and relationship between those objects. A drawing of a graph is a sort of visualization of information represented by the graph.

We now consider another example. The graph in Fig. 1.2(a) represents eight components and their interconnections in an electronic circuit, and Fig. 1.2(b) depicts a drawing of the graph. Although the graph in Fig. 1.2(a) correctly represents the circuit, the representation is messy and hard to trace the circuit for understanding and troubleshooting. Furthermore, in this representation one cannot lay the circuit on a single layered PCB (Printed Circuit Board) because of edge crossings. On the other hand,

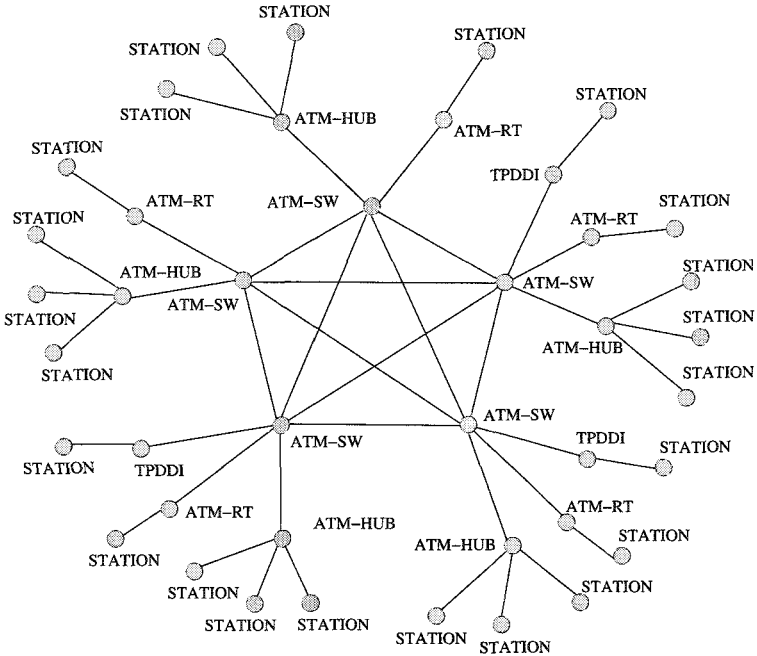


Fig. 1.1 A diagram of a computer network.

the drawing of the graph in Fig. 1.2(b) looks better and it is easily traceable. Furthermore one can use the drawing to lay the circuit on a single layered PCB, since it has no edge crossing. Thus the objective of graph drawing is to obtain a nice representation of a graph such that the structure of the graph is easily understandable, and moreover the drawing should satisfy some criteria that arises from the application point of view.

1.2 Historical Background

The origin of graph drawing is not well known. Although Euler (1707-1783) is credited with originating graph theory in 1736 [BW76], graph drawings were in limited use during centuries before Euler's time. A known example of ancient graph drawings is a family tree that decorated the atria of patrician roman villas [KMBW02].

The industrial need for graph drawing algorithms arose in the late 1960's when a large number of elements in complex circuit designs made hand-

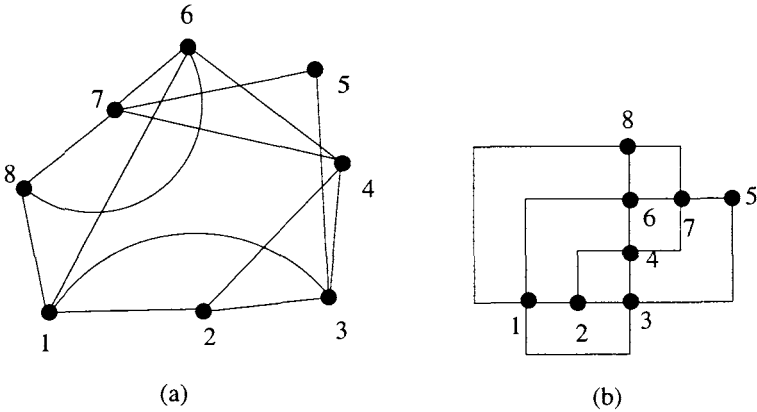


Fig. 1.2 An example of graph drawing in circuit schematics.

drawing too complicated [Bie97, Kan93, Rah99, Sug02]. Algorithms were developed to aid circuit design; an overview can be found in the book of Lengauer [Len90]. The field of graph drawing with the objective of producing aesthetically pleasing pictures became of interest in the late 1980's for presenting information of engineering and production process [CON85, TDB88].

The field of graph drawing has been flourished very much in the last decade. Recent progress in computational geometry, topological graph theory, and order theory has considerably affected the evolution of this field, and has widened the range of issues being investigated. A comprehensive bibliography on graph drawing algorithms [DETT94] cites more than 300 papers written before 1993. From 1993, an international symposium on graph drawing is being held annually in different countries and the proceedings of the symposium are published by Springer-Verlag in the LNCS series [TT95, Bra96, Nor97, Dib97, Whi98, Kra99, Mar01, GK02, Lio04]. Several special issues of journals dedicated to graph drawing have been recently assembled [CE95, DT96, DT98, DM99, LW00, Kau02].

1.3 Drawing Styles

In this section we introduce some important drawing styles and related terminologies.

TEAM LING - LIVE, INFORMATIVE, NON-COST AND GENUINE !

Various graphic standards are used for drawing graphs. Usually, vertices are represented by symbols such as points or boxes, and edges are represented by simple open Jordan curves connecting the symbols that represent the associated vertices. From now on, we assume that vertices are represented by points if not specified. We now introduce the following drawing styles.

1.3.1 Planar Drawing

A drawing of a graph is *planar* if no two edges intersect in the drawing. Figure 1.3 depicts a planar drawing and a non-planar drawing of the same graph. It is preferable to find a planar drawing of a graph if the graph has such a drawing. Unfortunately not all graphs admit planar drawings. A graph which admits a planar drawing is called a *planar graph*.

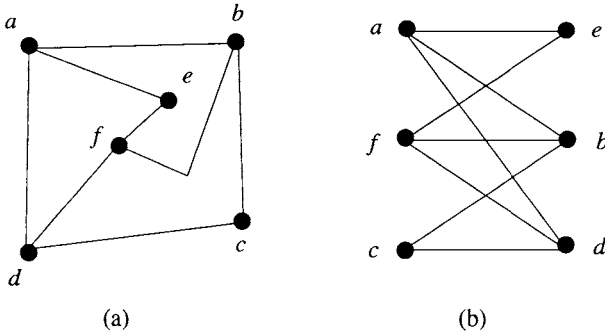


Fig. 1.3 (a) A planar drawing, and (b) a non-planar drawing of the same graph.

If one wants to find a planar drawing of a given graph, first he/she needs to test whether the given graph is planar or not. If the graph is planar, then he/she needs to find a planar embedding of the graph, which is a data structure representing adjacency lists: in each list the edges incident to a vertex are ordered, all clockwise or all counterclockwise, according to the planar embedding. Kuratowski [Kur30] gave the first complete characterization of planar graphs. (See Theorem 2.2.1.) Unfortunately the characterization does not lead to an efficient algorithm for planarity testing. Linear-time algorithms for this problem have been developed by Hopcroft and Tarjan [HT74], and Booth and Lueker [BL76]. Chiba *et al.* [CNAO85] and Mehlhorn and Mutzel [MM96] gave linear-time algorithms for finding a planar embedding of a planar graph. Shih and Hsu [SH99] gave a simple

linear-time algorithm which performs planarity testing and finds a planar embedding of a planar graph simultaneously. For the interested reader, algorithms for planarity testing and embeddings are given in Appendix A. A planar graph with a fixed planar embedding is called a *plane graph*.

1.3.2 Polyline Drawing

A *polyline drawing* is a drawing of a graph in which each edge of the graph is represented by a polygonal chain. A polyline drawing of a graph is shown in Fig. 1.4. A point at which an edge changes its direction in a polyline drawing is called a *bend*. Polyline drawings provide great flexibility since they can approximate drawings with curved edges. However, it may be difficult to follow edges with more than two or three bends by the eye. Several interesting results on polyline drawings can be found in [BSM02, DDLW03, GM98].

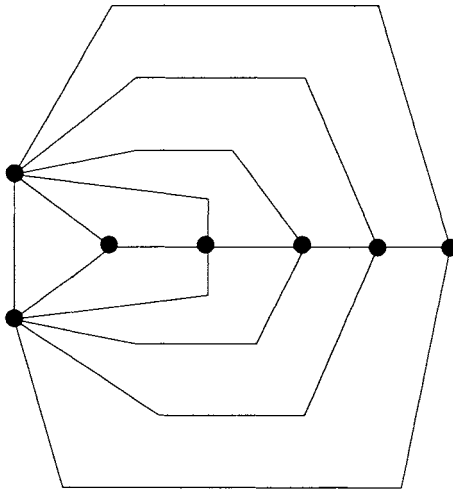


Fig. 1.4 A polyline drawing of a graph.

1.3.3 Straight Line Drawing

A *straight line drawing* is a drawing of a graph in which each edge of the graph is drawn as a straight line segment, as illustrated in Fig. 1.5. A straight line drawing is a special case of a polyline drawing, where edges

are drawn without bend.

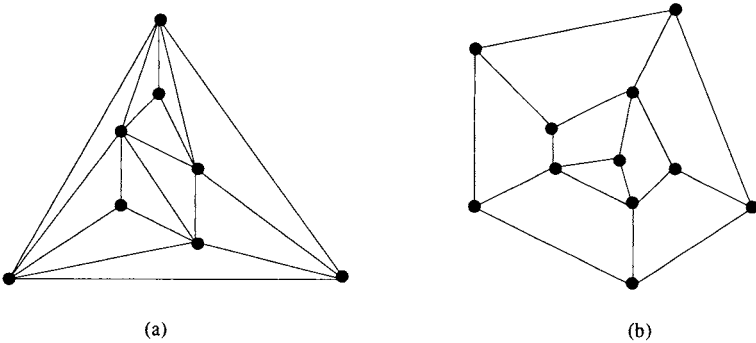


Fig. 1.5 (a) A straight line drawing, and (b) a convex drawing.

Wagner [Wag36], Fáry [Far48] and Stein [Ste51] independently proved that every planar graph has a straight line drawing. Many works have been done on straight line drawings of planar graphs [DETT94].

1.3.4 Convex Drawing

A straight line drawing of a plane graph G is called a *convex drawing* if the boundaries of all faces of G are drawn as convex polygons, as illustrated in Fig. 1.5(b). Although not every plane graph has a convex drawing, every 3-connected plane graph has such a drawing [Tut60]. Several algorithms are known for finding a convex drawing of a plane graph [CK97, CON85, CYN84, Kan96].

1.3.5 Orthogonal Drawing

An *orthogonal drawing* is a drawing of a plane graph in which each edge is drawn as a chain of horizontal and vertical line segments, as illustrated in Fig. 1.6(a). Orthogonal drawings have attracted much attention due to their numerous applications in circuit layouts, database diagrams, entity-relationship diagrams, etc. Many results have been published in recent years on both planar orthogonal drawings [Bie96a, Bie96b, Kan96, RNN99, RNN03, Sto84, Tam87, TTV91] and non-planar orthogonal drawings [BK98, PT95, PT97]. An orthogonal drawing is called an *octagonal drawing* if the outer cycle is drawn as a rectangle and each inner face is drawn as a rectilinear polygon of at most eight corners [RMN04].

1.3.6 Box-Orthogonal Drawing

Conventionally, each vertex in an orthogonal drawing is drawn as a point, as illustrated in Fig. 1.6(a). Clearly a graph having a vertex of degree five or more has no orthogonal drawing, because at most four edges can be incident to a vertex in an orthogonal drawing. A *box-orthogonal drawing* of a graph is a drawing such that each vertex is drawn as a (possibly degenerate) rectangle, called a *box*, and each edge is drawn as a sequence of alternate horizontal and vertical line segments, as illustrated in Fig. 1.6(b). Every plane graph has a box-orthogonal drawing. Several results are known for box-orthogonal drawings [BK97, FKK97, PT00].

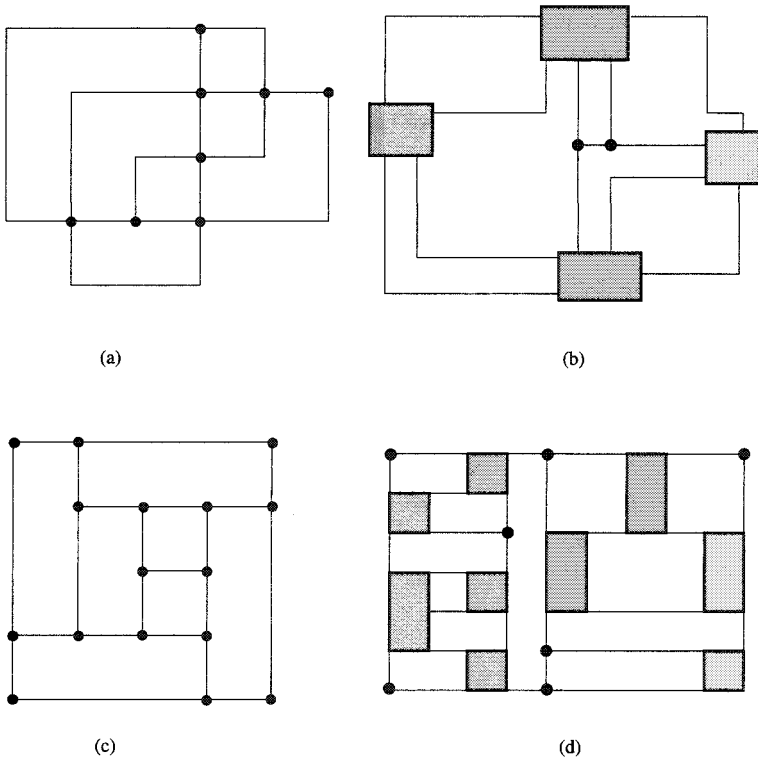


Fig. 1.6 (a) An orthogonal drawing, (b) a box-orthogonal drawing, (c) a rectangular drawing, and (d) a box-rectangular drawing.

1.3.7 Rectangular Drawing

A *rectangular drawing* of a plane graph G is a drawing of G in which each vertex is drawn as a point, each edge is drawn as a horizontal or vertical line segment without edge-crossings, and each face is drawn as a rectangle, as illustrated in Fig. 1.6(c). Not every plane graph has a rectangular drawing. Thomassen [Tho84] and Rahman *et al.* [RNN02] established necessary and sufficient conditions for a plane graph of the maximum degree three to have a rectangular drawing. Linear-time algorithms for finding rectangular drawings of such plane graphs are also known [BS88, RNN98, RNN02]. Recently Miura *et al.* reduced the problem of finding a rectangular drawing of a plane graph of the maximum degree four to a perfect matching problem [MHN04].

A planar graph G is said to have a rectangular drawing if at least one of the plane embeddings of G has a rectangular drawing. Recently Rahman *et al.* [RNG04] gave a linear time algorithm to examine whether a planar graph of the maximum degree three has a rectangular drawing and to find a rectangular drawing if it exists.

1.3.8 Box-Rectangular Drawing

A *box-rectangular drawing* of a plane graph G is a drawing of G on the plane such that each vertex is drawn as a (possibly degenerate) rectangle, called a *box*, and the contour of each face is drawn as a rectangle, as illustrated in Fig. 1.6(d). If G has multiple edges or a vertex of degree five or more, then G has no rectangular drawing but may have a box-rectangular drawing. However, not every plane graph has a box-rectangular drawing. Rahman *et al.* [RNN00] gave a necessary and sufficient condition for a plane graph to have a box-rectangular drawing. Linear-time algorithms are also known for finding a box-rectangular drawing of a plane graph if it exists [He01, RNN00].

1.3.9 Grid Drawing

A drawing of a graph in which vertices and bends are located at grid points of an integer grid as illustrated in Fig. 1.7 is called a *grid drawing*. Grid drawing approach overcomes the following problems in graph drawing with real number arithmetic.

- (i) When the embedding has to be drawn on a raster device, real vertex

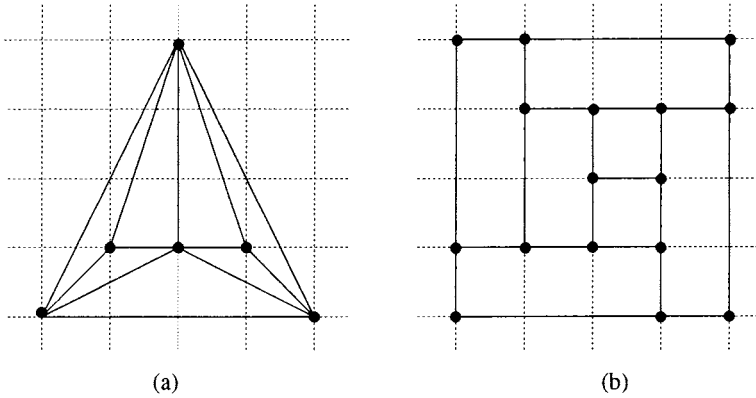


Fig. 1.7 (a) A straight line grid drawing, and (b) a rectangular grid drawing.

coordinates have to be mapped to integer grid points, and there is no guarantee that a correct embedding will be obtained after rounding.

- (ii) Many vertices may be concentrated in a small region of the drawing. Thus the embedding may be messy, and line intersections may not be detected.
- (iii) One cannot compare area requirement for two or more different drawings using real number arithmetic, since any drawing can be fitted in any small area using magnification.

The *size* of an integer grid required for a grid drawing is measured by the size of the smallest rectangle on the grid which encloses the drawing. The *width* W of the grid is the width of the rectangle and the *height* H of the grid is the height of the rectangle. The grid size is usually described as $W \times H$. The grid size is sometimes described by the *half perimeter* $W + H$ or the *area* $W \cdot H$ of the grid.

It is a very challenging problem to draw a plane graph on a grid of the minimum size. In recent years, several works are devoted to this field [CN98, FPP90, Sch90]; for example, every plane graph of n vertices has a straight line grid drawing on a grid of size $W \times H \leq (n - 1) \times (n - 1)$.

1.3.10 Visibility Drawing

A *visibility drawing* of a plane graph G is a drawing of G where each vertex is drawn as a horizontal line segment and each edge is drawn as a vertical line segment. The vertical line segment representing an edge must connect

points on the horizontal line segments representing the end vertices [Kan97]. Figure 1.8(b) depicts a visibility drawing of the plane graph G in Fig. 1.8(a).

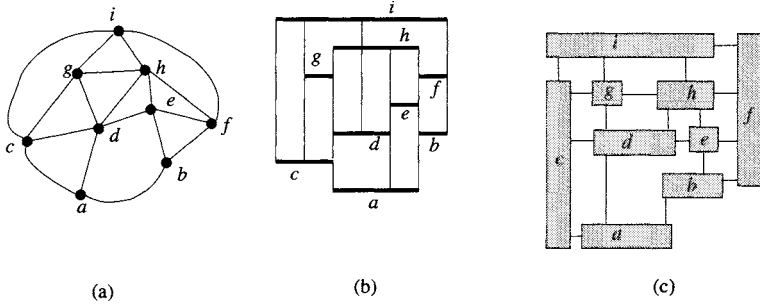


Fig. 1.8 (a) A plane graph G , (b) a visibility drawing of G , and (c) a 2-visibility drawing of G .

A *2-visibility drawing* is a generalization of a visibility drawing where vertices are drawn as boxes and edges are drawn as either a horizontal line segment or a vertical line segment [FKK97]. Figure 1.8(c) depicts a 2-visibility drawing of the plane graph G in Fig. 1.8(a).

1.4 Properties of Drawings

There are infinitely many drawings of a graph. When drawing a graph, we would like to consider a variety of properties. For example, if a graph corresponds to a VLSI circuit, then we may be interested in a planar orthogonal drawing of the graph such that the number of bends in the drawing is as small as possible, because bends increase the manufacturing cost of a VLSI chip. To avoid wasting valuable space in the chip, it is important to keep the area of the drawing small. Even if we are motivated to obtain only a nice drawing, we cannot precisely define a nice drawing, and hence we consider some properties of graph drawings [Bie97]. In this section we introduce some properties of graph drawings which we generally consider.

Area. A drawing is useless if it is unreadable. If the used area of the drawing is large, then we have to use many pages, or we must decrease resolution, so either way the drawing becomes unreadable. Therefore one major objective is to ensure a small area. Small drawing area is also preferable in application domains like VLSI floorplanning.

Aspect Ratio. *Aspect ratio* is defined as the ratio of the length of the longest side to the length of the shortest side of the smallest rectangle which encloses the drawing.

Bends. At a bend, the polyline drawing of an edge changes direction, and hence a bend on an edge increases the difficulties of following the course of the edge. For this reason, both the total number of bends and the number of bends per edge should be kept small.

Crossings. Every crossing of edges bears the potential of confusion, and therefore the number of crossings should be kept small.

Shape of Faces. If every face has a regular shape in a drawing, the drawing looks nice. For VLSI floorplanning, it is desirable that each face is drawn as a rectangle.

Symmetry. Symmetry is an important aesthetic criteria in graph drawing. A symmetry of a two-dimensional figure is an isometry of the plane that fixes the figure [HE03]. There are two types of two-dimensional symmetry, *rotational symmetry* and *reflectional symmetry*. Rotational symmetry is a rotation about a point and reflectional symmetry is a reflection in an axis.

Angular Resolution. *Angular resolution* is measured by the smallest angle between adjacent edges in a drawing. Higher angular resolution is desirable for displaying a drawing on a raster device.

For most of the cases above, it is hard to achieve optimum. Garey and Johnson showed that minimizing the number of crossings is NP-complete [GJ83]. Kramer and van Leeuwen [KL84] proved that it is NP-complete to examine whether a graph can be embedded in a grid of prescribed size, and Formann and Wagner pointed out some corrections to the proof [FW91]. Garg and Tamassia showed that the problem of determining the minimum number of bends for orthogonal drawings is NP-complete [GT01]. The problem of determining whether a given graph can be drawn symmetrically is also NP-complete [Lub81, Man91].

1.5 Applications of Graph Drawing

Graph drawings have applications in almost every branch of science and technology [JM04, KW01, Sug02]. In Section 1.1 we have seen two applications of graph drawings in computer networks and circuit schematics. In

this section we will investigate a few more applications of graph drawings.

1.5.1 Floorplanning

Graph drawings have applications in VLSI floorplanning as well as architectural floorplanning. In a VLSI floorplanning problem, an input is a plane graph F as illustrated in Fig. 1.9(a); F represents the functional entities of a chip, called *modules*, and interconnections among the modules; each vertex of F represents a module, and an edge between two vertices of F represents the interconnections between the two corresponding modules. An output of the problem for the input graph F is a partition of a rectangular chip area into smaller rectangles as illustrated in Fig. 1.9(d); each module is assigned to a smaller rectangle, and furthermore, if two modules have interconnections, then their corresponding rectangles must be adjacent, that is, must have a common boundary. A similar problem may arise in architectural floorplanning also. When building a house, the owner may have some preference; for example, a bed room should be adjacent to a reading room. The owner's choice of room adjacencies can be easily modeled by a plane graph F , as illustrated in Fig. 1.9(a); each vertex represents a room and an edge between two vertices represents the desired adjacency between the corresponding rooms.

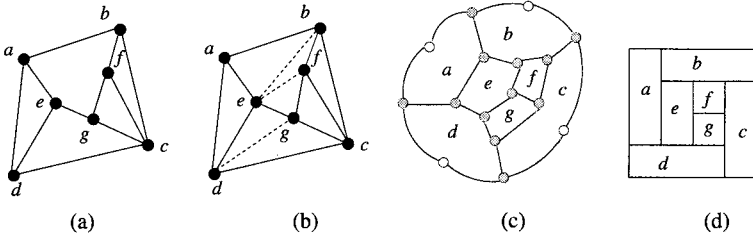


Fig. 1.9 (a) Graph F , (b) triangulated graph F' , (c) dual-like graph G , and (d) rectangular drawing of G .

A rectangular drawing of a plane graph may provide a suitable solution of the floorplanning problem described above. First, obtain a plane graph F' by triangulating all inner faces of F as illustrated in Fig. 1.9(b), where dotted lines indicate new edges added to F . Then obtain a dual-like graph G of F' as illustrated in Fig. 1.9(c), where the four vertices of degree 2 drawn by white circles correspond to the four corners of the rectangular area. Finally, by finding a rectangular drawing of the plane graph G , obtain

a possible floorplan for F as illustrated in Fig. 1.9(d).

In the floorplan above, two rectangles are always adjacent if the modules corresponding to them have interconnections in F . However, two rectangles may be adjacent even if the modules corresponding to them have no interconnections. For example, module e and module f have no interconnection in F , but their corresponding rectangles are adjacent in the floorplan in Fig. 1.9(d). Such unwanted adjacencies are not desirable in some other floorplanning problems. In floorplanning of a MultiChip Module (MCM), two chips generating excessive heat should not be adjacent, or two chips operating on high frequency should not be adjacent to avoid malfunctioning due to their interference [She95]. Unwanted adjacencies may cause a dangerous situation in some architectural floorplanning, too [FW74]. For example, in a chemical industry, a processing unit that deals with poisonous chemicals should not be adjacent to a cafeteria.

We can avoid the unwanted adjacencies if we obtain a floorplan for F by using a box-rectangular drawing instead of a rectangular drawing, as follows. First, without triangulating the inner faces of F , find a dual-like graph G of F as illustrated in Fig. 1.10(b). Then, by finding a box-rectangular drawing of G , obtain a possible floorplan for F as illustrated in Fig. 1.10(c). In Fig. 1.10(c) rectangles e and f are not adjacent although there is a dead space corresponding to a vertex of G drawn by a rectangular box. Such a dead space to separate two rectangles in floorplanning is desirable for dissipating excessive heat in an MCM or for ensuring safety in a chemical industry.

1.5.2 VLSI Layout

In this section we consider another VLSI layout problem. Modules and their *interconnections* of a VLSI circuit are given as a graph as illustrated in Fig. 1.11(a); a vertex of the graph represents a module of the VLSI circuit and an edge represents an interconnection between two modules. Edges appear around a vertex in the same order as they will appear in the actual layout. Pin positions of the modules are shown in Fig. 1.11(b). We want to find a layout of the circuit. One can find a layout of the circuit as illustrated in Fig. 1.11(c) by finding a box-orthogonal drawing of the graph in Fig. 1.11(a) [SAR01].

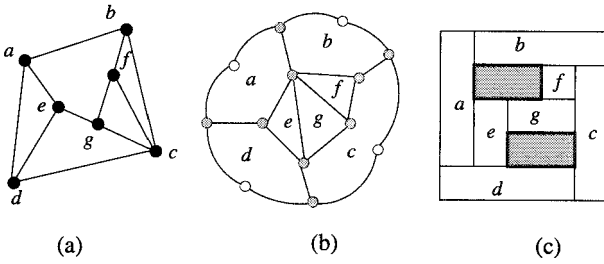


Fig. 1.10 (a) F , (b) G , and (c) box-rectangular drawing of G .

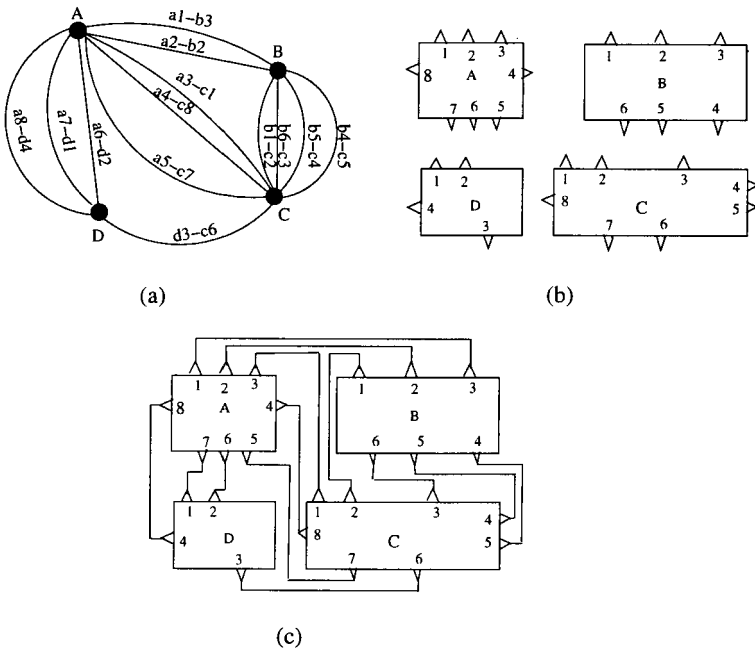


Fig. 1.11 (a) Interconnections graph, (b) pin positions of modules, and (c) a VLSI floorplan with detailed routing.

1.5.3 Software Engineering

Graph drawings have applications in visualization of large scale object-oriented software systems to support maintenance and re-engineering processes in an appropriately automated way for large programs [Sug02].

tices in such graphs represent structure entities like classes or packages. They are visualized by simple geometric objects (as spheres or cubes) with geometric properties (as color or size) representing software metric values. Relations are displayed as straight lines colored according to their relation type (method usage, inheritance).

1.5.4 *Simulating Molecular Structures*

A common strategy in drug design and pharmacophore identification is to evaluate a large set of molecular structures by comparing their 2D structure drawings. Using suitable graph representations of molecules and drawing such graphs, one can simplify the chemist's task since similarities and differences between drugs are revealed in the drawing. Boissonnate *et al.* [BCF01] gave a heuristic algorithm to compute 2D structure drawings of molecules which have effective applications in drug design. A promising application of graph drawing is found in RNA genomics for characterizing and analyzing RNA structures, where RNA structures are represented by planar graphs [GPS03].

1.6 Scope of This Book

Due to numerous practical applications of graph drawing, researchers have worked on graph drawing with various view-points, and as a result, the field of graph drawing has become very wide. Together with established results on graph drawing, such as spring model, barycentric method, flow model, etc. [DETT99, Sug02], the field has been enriched with many recent results on 3D drawing, dynamic drawing, proximity drawing, map labeling, etc [KW01]. Different classes of graphs such as planar graphs, hierarchical graphs, clustered graphs, interval graphs, etc. are investigated to produce graph drawings having some desired properties. Therefore it would be difficult to cover all results of graph drawing in a single book.

In our opinion, the concepts of drawing planar graphs are basic building blocks of the field of graph drawing. A drawing of a planar graph looks nice and algorithms for drawing planar graphs can be successfully used for drawing a nonplanar graph by transforming the nonplanar graph into a similar planar graph. Furthermore, drawings of planar graphs have practical applications in VLSI floorplanning and routing, architectural floorplanning, displaying RNA structures in bioinformatics, etc. We thus devote

TEAM LING - LIVE, INFORMATIVE, NON-COST and GENUINE !

this book to the theories and algorithms for drawing planar graphs. Particularly, straight line drawings, convex drawings, rectangular drawings, box-rectangular drawings, orthogonal drawings and octagonal drawings of planar graphs are in the scope of this book.

Exercise

1. Find a straight line drawing of the plane graph in Fig. 1.12 on a grid, and determine the grid size of your drawing. Does your drawing take the minimum grid size?

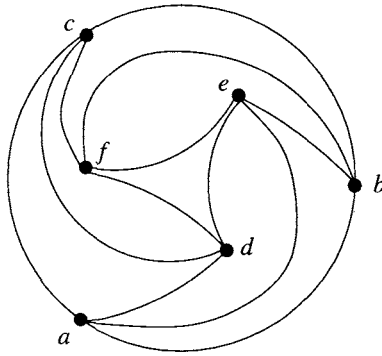


Fig. 1.12 Graph for Exercise 1.

2. Find rectangular drawings of the graphs in Fig. 1.13.
3. The graphs in Fig. 1.14 have no rectangular drawings. Explain the reasons.
4. Find a box-rectangular drawing of the graph in Fig. 1.15.
5. The plane graphs in Fig. 1.16 have no convex drawing. Why?
6. Establish relationship among an orthogonal drawing, a box-orthogonal drawing, a rectangular drawing and a box-rectangular drawing.
7. Find an orthogonal drawing of the plane graph in Fig. 1.17. Count the number of bends in your drawing. Is the number minimum for an orthogonal drawing of the plane graph?
8. Find a visibility drawing and a 2-visibility drawing of the plane graph in Fig. 1.12.
9. Mention some similarities and differences between a box-rectangular drawing and a 2-visibility drawing of a plane graph.

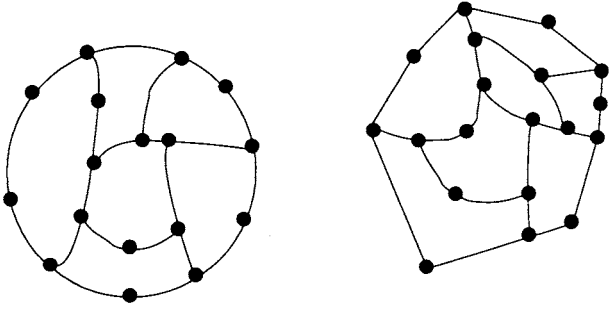


Fig. 1.13 Graphs for Exercise 2.

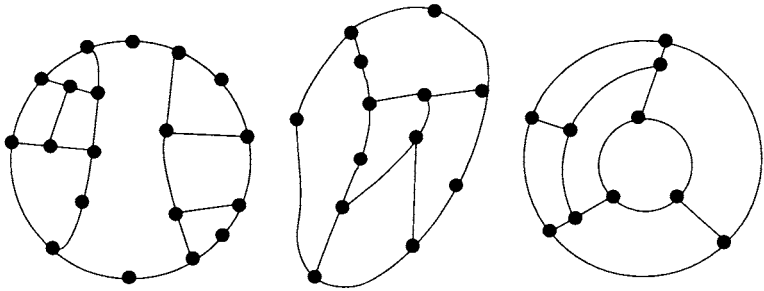


Fig. 1.14 Graphs for Exercise 3.

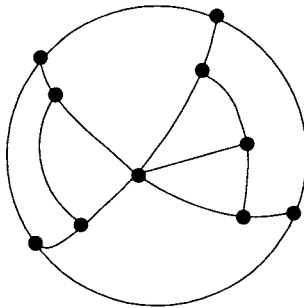


Fig. 1.15 Graph for Exercise 4.

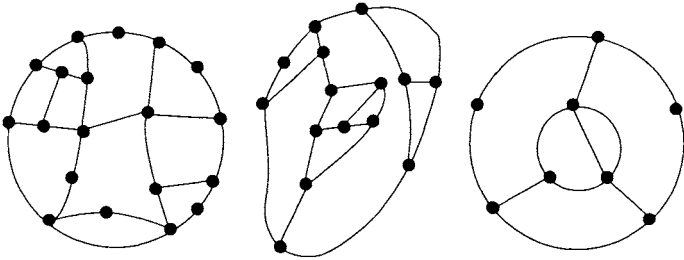


Fig. 1.16 Graphs for Exercise 5.

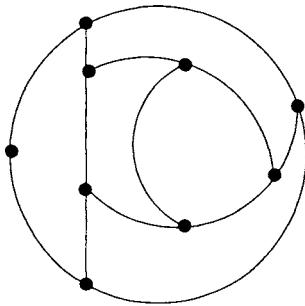


Fig. 1.17 Graph for Exercise 7.

Chapter 2

Graph Theoretic Foundations

2.1 Basic Terminology

In this section we give some definitions of standard graph-theoretical terms used throughout the remainder of this book.

2.1.1 *Graphs and Multigraphs*

A *graph* G is a tuple (V, E) which consists of a finite set V of *vertices* and a finite set E of *edges*; each edge is an unordered pair of vertices. We often denote the set of vertices of G by $V(G)$ and the set of edges by $E(G)$. Figure 2.1 depicts a graph G , where each vertex in $V(G) = \{v_1, v_2, \dots, v_6\}$ is drawn by a small black circle and each edge in $E(G) = \{e_1, e_2, \dots, e_9\}$ is drawn by a line segment. The number of vertices of G is denoted by n , that is, $n = |V|$, and the number of edges of G is denoted by m , that is, $m = |E|$. Thus $n = 6$ and $m = 9$ for the graph in Fig. 2.1.

If a graph G has no “multiple edges” or “loops,” then G is called a *simple graph*. *Multiple edges* join the same pair of vertices, while a *loop* joins a vertex itself. A graph in which loops and multiple edges are allowed is called a *multigraph*. In the remainder of the book we call a simple graph a *graph* if there is no possibility of confusion.

We denote an edge joining vertices u and v of G by (u, v) or simply by uv . If $uv \in E$, then two vertices u and v are said to be *adjacent* in G ; edge uv is then said to be *incident* to vertices u and v ; u is a *neighbor* of v . The *degree* $d(v, G)$ of a vertex v in G is the number of edges incident to v in G . We often write $d(v)$ for simplicity. In the graph in Fig. 2.1 vertices v_1 and v_2 are adjacent, and $d(v_1) = 3$ since three edges e_1, e_5 and e_6 are incident to v_1 . We denote by Δ the maximum of the degrees of all vertices in G and

TEAM LING - LIVE, INFORMATIVE, NON-COST AND GENUINE !

call it the *maximum degree* of graph G .

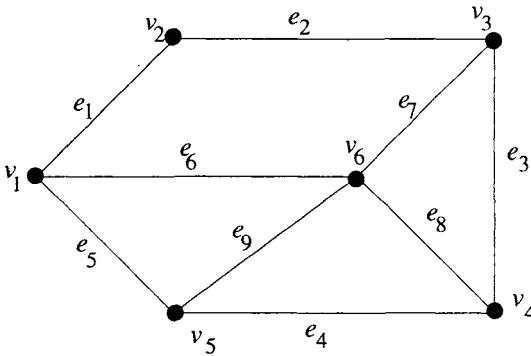


Fig. 2.1 A graph with six vertices and nine edges.

2.1.2 Subgraphs

A *subgraph* of a graph $G = (V, E)$ is a graph $G' = (V', E')$ such that $V' \subseteq V$ and $E' \subseteq E$. If G' contains all the edges of G that join vertices in V' , then G' is called the *subgraph induced by V'* . Figure 2.2 depicts a subgraph of G in Fig. 2.1 induced by $\{v_1, v_2, v_5, v_6\}$. If $V' = V$, then G' is called a *spanning subgraph* of G .

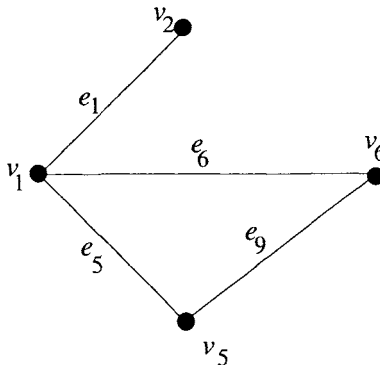


Fig. 2.2 A vertex-induced subgraph.

We often construct new graphs from old ones by deleting some vertices or edges. If v is a vertex of a given graph $G = (V, E)$, then $G - v$ is the

subgraph of G obtained by deleting the vertex v and all the edges incident to v . More generally, if V' is a subset of V , then $G - V'$ is the subgraph of G obtained by deleting the vertices in V' and all the edges incident to them. Thus $G - V'$ is a subgraph of G induced by $V - V'$. Similarly, if e is an edge of G , then $G - e$ is the subgraph of G obtained by deleting the edge e . More generally, if $E' \subseteq E$, then $G - E'$ is the subgraph of G obtained from G by deleting all the edges in E' .

2.1.3 Paths and Cycles

A *walk*, $v_0, e_1, v_1, \dots, v_{l-1}, e_l, v_l$, in a graph G is an alternating sequence of vertices and edges of G , beginning and ending with a vertex, in which each edge is incident to two vertices immediately preceding and following it. If the vertices v_0, v_1, \dots, v_l are distinct (except possibly v_0, v_l), then the walk is called a *path* and usually denoted either by the sequence of vertices v_0, v_1, \dots, v_l or by the sequence of edges e_1, e_2, \dots, e_l . The length of the path is l , one less than the number of vertices on the path. A path or walk is closed if $v_0 = v_l$. A closed path containing at least one edge is called a *cycle*.

2.1.4 Chains

Let $P = v_0, v_1, v_2, \dots, v_{l+1}$, $l \geq 1$, be a path of a graph G such that $d(v_0) \geq 3, d(v_1) = d(v_2) = \dots = d(v_l) = 2$, and $d(v_{l+1}) \geq 3$. Then we call the subpath $P' = v_1, v_2, \dots, v_l$ of P a *chain* of G , and we call vertices v_0 and v_{l+1} the *supports* of the chain P' . Two chains of G are *adjacent* if they have a common support. Figure 2.3 illustrates a plane graph with five chains P_1, P_2, P_3, P_4 and P_5 ; only P_2 and P_3 are adjacent.

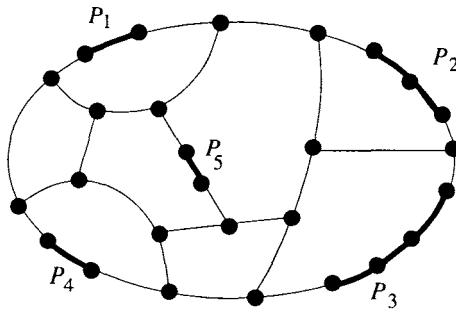


Fig. 2.3 A plane graph with five chains P_1, P_2, P_3, P_4 and P_5 drawn by thick lines.

2.1.5 Connectivity

A graph G is *connected* if for any two distinct vertices u and v there is a path between u and v in G . A graph which is not connected is called a *disconnected graph*. A (*connected*) *component* of a graph is a maximal connected subgraph. The graph in Fig. 2.4(a) is connected since there is a path for any two distinct vertices of the graph. On the other hand the graph in Fig. 2.4(b) is disconnected since there is no path between v_1 and v_5 . The graph in Fig. 2.4(b) has two connected components indicated by dotted lines.

The *connectivity* $\kappa(G)$ of a graph G is the minimum number of vertices whose removal results in a disconnected graph or a single-vertex graph K_1 . We say that G is *k -connected* if $\kappa(G) \geq k$. We call a set of vertices in a connected graph G a *separator* or a *vertex-cut* if the removal of the vertices in the set results in a disconnected or single-vertex graph. If a vertex-cut contains exactly one vertex, then we call the vertex a *cut vertex*. If a vertex-cut contains exactly two vertices, then we call the two vertices a *separation pair*.

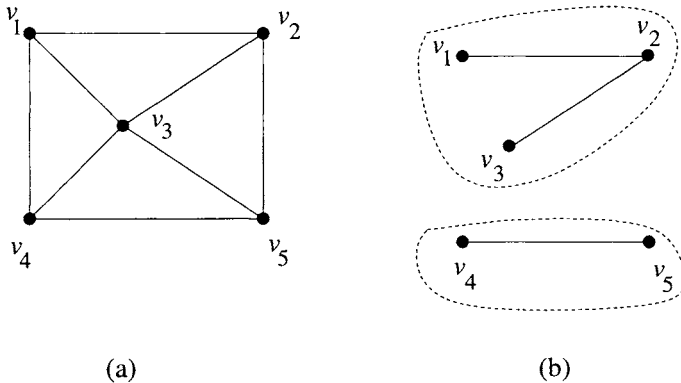


Fig. 2.4 (a) A connected graph, and (b) a disconnected graph with two connected components.

2.1.6 Trees and Forests

A *tree* is a connected graph without any cycle. Figure 2.5 is an example of a tree. The vertices in a tree are usually called *nodes*. A *rooted tree* is a tree in which one of the nodes is distinguished from the others. The

distinguished node is called the *root* of the tree. The root of a tree is usually drawn at the top. In Fig. 2.5, the root is v_1 . If a rooted tree is regarded as a directed graph in which each edge is directed from top to bottom, then every node u other than the root is connected by an edge from some other node p , called the *parent* of u . We also call u a *child* of node p . We draw the parent of a node above that node. For example, in Fig. 2.5, v_1 is the parent of v_2, v_3 and v_4 , while v_2 is the parent of v_5 and v_6 ; v_2, v_3 and v_4 are the children of v_1 , while v_5 and v_6 are the children of v_2 . A *leaf* is a node of a tree that has no children. An *internal node* is a node that has one or more children. Thus every node of a tree is either a leaf or an internal node. In Fig. 2.5, the leaves are v_4, v_5, v_6 and v_7 , and the nodes v_1, v_2 and v_3 are internal nodes.

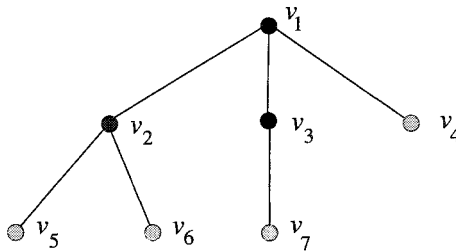


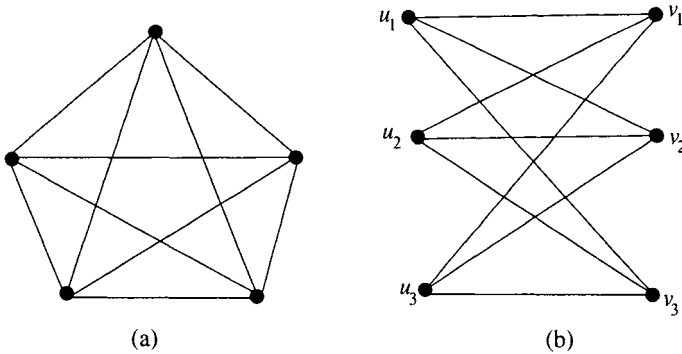
Fig. 2.5 A tree.

The parent-child relationship can be extended naturally to ancestors and descendants. Suppose that u_1, u_2, \dots, u_l is a sequence of nodes in a tree such that u_1 is the parent of u_2 , which is a parent of u_3 , and so on. Then node u_1 is called an *ancestor* of u_l and node u_l a *descendant* of u_1 . The root is an ancestor of every node in a tree and every node is a descendant of the root. In Fig. 2.5, all nodes are descendants of root v_1 , and v_1 is an ancestor of all nodes.

A graph without a cycle is called a *forest*. Each connected component of a forest is a tree. A spanning subgraph of a graph G is called a *tree* of G if it is a tree. A tree of a connected graph G is called a *spanning tree* of G .

2.1.7 Complete Graphs

A graph in which every pair of distinct vertices are adjacent is called a *complete graph*. A complete graph of n vertices is denoted by K_n . K_5 is depicted in Fig. 2.6(a).

Fig. 2.6 (a) K_5 and (b) $K_{3,3}$.

2.1.8 Bipartite Graphs

Suppose that the vertex set V of a graph G can be partitioned into two disjoint sets V_1 and V_2 in such a way that every edge of G joins a vertex of V_1 to a vertex of V_2 ; G is then called a *bipartite graph*. If every vertex of V_1 is joined to every vertex of V_2 , then G is called a *complete bipartite graph* and is denoted by $K_{s,r}$ where $s = |V_1|$ and $r = |V_2|$. Figure 2.6(b) depicts a complete bipartite graph $K_{3,3}$ with partite sets $V_1 = \{u_1, u_2, u_3\}$ and $V_2 = \{v_1, v_2, v_3\}$.

2.1.9 Subdivisions

Subdividing an edge (u, v) of a graph G is the operation of deleting the edge (u, v) and adding a path $u(= w_0), w_1, w_2, \dots, w_k, v(= w_{k+1})$ through new vertices $w_1, w_2, \dots, w_k, k \geq 1$, of degree two. A graph G is said to be a *subdivision* of a graph G' if G is obtained from G' by subdividing some of the edges of G' . Figure 2.7 depicts subdivisions of K_5 and $K_{3,3}$.

2.2 Planar Graphs

A graph is *planar* if it can be embedded in the plane so that no two edges intersect geometrically except at a vertex to which they are both incident. Note that a planar graph may have an exponential number of embeddings. Figure 2.8 depicts four planar embeddings of the same planar graph.

One of the most beautiful theorems in graph theory is Kuratowski's, which gives a characterization of planar graphs in terms of "forbidden

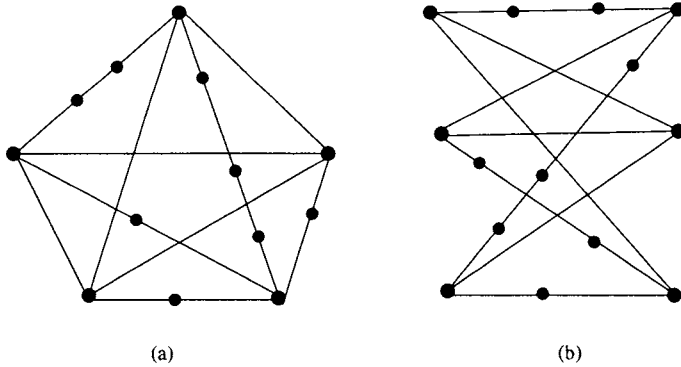


Fig. 2.7 Subdivisions of (a) K_5 and (b) $K_{3,3}$.

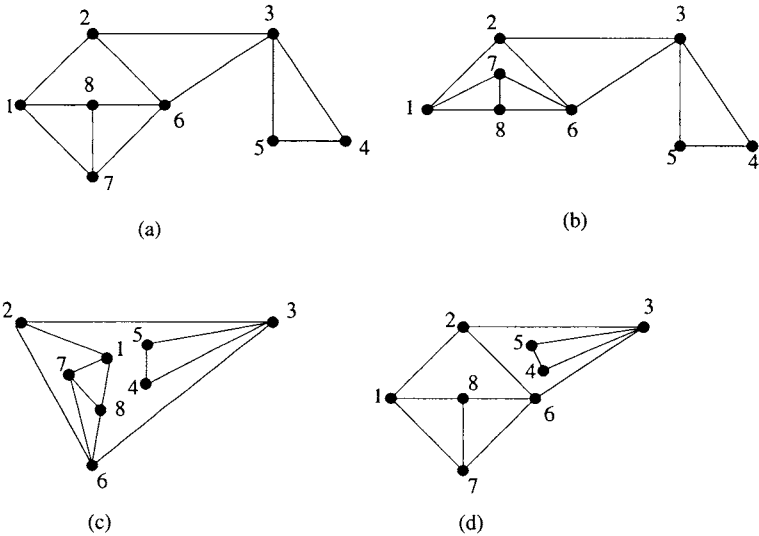


Fig. 2.8 Four planar embeddings of the same graph.

graphs,” as follows.

Theorem 2.2.1 (Kuratowski 1930) *A graph is planar if and only if it contains neither a subdivision of K_5 nor a subdivision of $K_{3,3}$.*

2.2.1 Plane Graphs

A *plane graph* G is a planar graph with a fixed embedding in the plane. A plane graph divides the plane into connected regions called *faces*. The unbounded region is called the *outer face* of G . The boundary of a face of a connected plane graph G is a closed walk in general, and is a cycle if G is 2-connected and has at least three vertices. The boundary of the outer face of G is called the *outer boundary* of G and denoted by $C_o(G)$. If $C_o(G)$ is a cycle, then $C_o(G)$ is called the *outer cycle* of G . We call a vertex v of G an *outer vertex* of G if v is on $C_o(G)$; otherwise v is an *inner vertex* of G . Similarly we define an *outer edge* and an *inner edge* of G .

Let C be a cycle of a graph G . A graph of a single edge, not in C , joining two vertices in C is called a C -*component* of G . A graph which consists of a connected component of $G - V(C)$ and all edges joining vertices in that component and vertices in C is also called a C -*component*. The outer cycle $C_o(G)$ of the plane graph G in Fig. 2.9(a) is drawn by thick lines, and the $C_o(G)$ -components J_1 , J_2 and J_3 of G are depicted in Fig. 2.9(b).

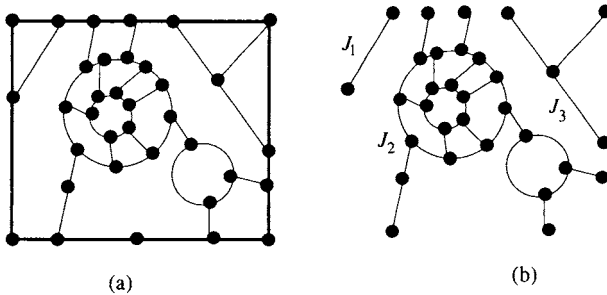


Fig. 2.9 (a) Plane graph G , and (b) $C_o(G)$ -components.

For a cycle C in a plane graph G , we denote by $G(C)$ the plane subgraph of G inside C (including C). The subgraph shaded in Fig. 2.10 is $G(C)$ for a cycle C . An edge which is incident to exactly one vertex of C and located outside of C is called a *leg* of C , and the vertex of C to which the leg is incident is called a *leg-vertex* of C . A cycle C in G is called a *k-legged cycle* of G if C has exactly k legs. A k -legged cycle C is *minimal* if $G(C)$ does not contain any other k -legged cycle of G . The cycle C in Fig. 2.10 is a 4-legged cycle with the legs e_1, e_2, e_3 and e_4 .

We say that cycles C and C' in a plane graph G are *independent* if $G(C)$ and $G(C')$ have no common vertex. A set S of cycles is *independent* if any

pair of cycles in \mathcal{S} are independent.

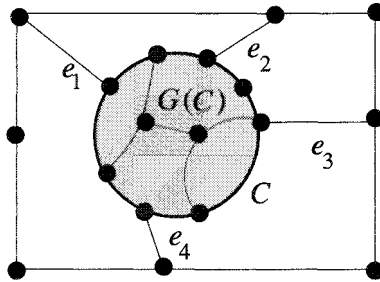


Fig. 2.10 Illustration of $G(C)$ and legs of a cycle C .

As mentioned before, a planar graph may have an exponential number of embeddings in the plane. We shall now define an equivalence relation among these embeddings. Two embeddings of a planar graph are *equivalent* when the boundary of a face in one embedding always corresponds to the boundary of a face in the other. We say that the plane embedding of a graph is *unique* when the embeddings are all equivalent. If G is a disconnected plane graph, one can obtain a new nonequivalent embedding simply by replacing a connected component within another face. Similarly, if G has a cut vertex v , one may obtain a new nonequivalent embedding by replacing a component of $G - v$ (together with the edges joining v and vertices in the component) in another face incident to v . Thus we shall assume that G is 2-connected if the embedding is unique. Whitney [Whi33] proved that the embedding of a 3-connected planar graph is unique. Before proving the result, we need some definitions.

If G has a separation pair $\{x, y\}$, then we often split G into two graphs G_1 and G_2 , called *split graphs*. Let $G'_1 = (V_1, E'_1)$ and $G'_2 = (V_2, E'_2)$ be two subgraphs satisfying the following conditions (a) and (b):

- (a) $V = V_1 \cup V_2, V_1 \cap V_2 = \{x, y\}$;
- (b) $E = E'_1 \cup E'_2, E'_1 \cap E'_2 = \emptyset, |E'_1| \geq 2, |E'_2| \geq 2$.

Define a split graph G_1 to be the graph obtained from G'_1 by adding a new edge (x, y) if it does not exist; similarly define a split graph G_2 . (See Fig. 2.11.)

Theorem 2.2.2 *The embedding of a 2-connected planar graph G is unique if and only if G is a subdivision of a 3-connected graph.*

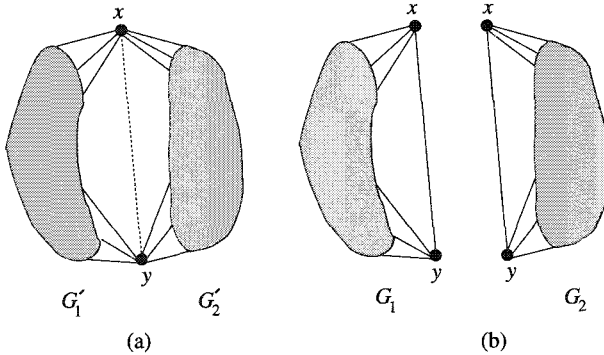


Fig. 2.11 (a) A graph G with a separation pair $\{x, y\}$ where edge (x, y) may not exist, and (b) split graphs G_1 and G_2 .

Proof. Necessity: Suppose that a 2-connected planar graph G is not a subdivision of a 3-connected graph. Then there is a separation pair $\{x, y\}$ having split graphs G_1 and G_2 such that both G'_1 and G'_2 are not paths. (See Fig. 2.11.) Consider a plane embedding of G in which both x and y are outer vertices. Then a new embedding of G is obtained by a reflection or twist of G'_1 or G'_2 . The boundary of the outer face in the original embedding is no longer a face boundary in the new embedding. Thus the embedding of G is not unique.

Sufficiency: Suppose that the embedding of a 2-connected planar graph G is not unique. Thus, according to the definition, the original embedding $\Gamma(G)$ of G has a face F whose facial cycle C in $\Gamma(G)$ is no longer a facial cycle in another embedding $\Gamma'(G)$ of G . Clearly G has two C -components J and J' ; one in the interior and the other in the exterior of C in $\Gamma'(G)$. One may assume that C is the boundary of the outer face of $\Gamma(G)$. Let x_1, x_2, \dots, x_k be the vertices of C contained in J occurring in cyclic order. One may assume that all the vertices of C contained in J' are in the subpath of C joining x_1 and x_2 and containing no other x_i , as illustrated in Fig. 2.12. Then $\{x_1, x_2\}$ is a separation pair of G , for which both G'_1 and G'_2 are not paths. Therefore G is not a subdivision of a 3-connected graph. \square

Theorem 2.2.2 immediately implies that every 3-connected planar graph has a unique plane embedding.

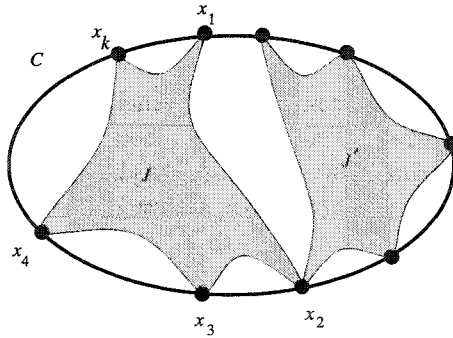


Fig. 2.12 Illustration for the proof of Theorem 2.2.2.

2.2.2 Euler's Formula

There is a simple formula relating the numbers of vertices, edges and faces in a connected plane graph. It is known as *Euler's formula* because Euler established it for those plane graphs defined by the vertices and edges of polyhedra. In this section we discuss Euler's formula and its immediate consequences.

Theorem 2.2.3 (Euler 1750) *Let G be a connected plane graph, and let n , m , and f denote respectively the numbers of vertices, edges and faces of G . Then $n - m + f = 2$.*

Proof. We employ an induction on m , the result being obvious for $m = 0$ or 1. Assume that $m \geq 2$ and the result is true for all connected plane graphs having fewer than m edges, and suppose that G has m edges. Consider first the case G is a tree. Then G has a vertex v of degree one. The connected plane graph $G - v$ has $n - 1$ vertices, $m - 1$ edges and $f (= 1)$ faces, so by the inductive hypothesis, $(n - 1) - (m - 1) + f = 2$, which implies that $n - m + f = 2$. Consider next the case when G is not a tree. Then G has an edge e on a cycle. In this case the connected plane graph $G - e$ has n vertices, $m - 1$ edges, and $f - 1$ faces, so that the desired formula immediately follows from the inductive hypothesis. \square

A *maximal planar graph* is one to which no edge can be added without losing planarity. Thus in any embedding of a maximal planar graph G with $n \geq 3$, the boundary of every face of G is a triangle, and hence the embedding is often called a *triangulated plane graph*. Although a general graph may have up to $n(n - 1)/2$ edges, it is not true for planar graphs.

Corollary 2.2.4 *If G is a planar graph with $n(\geq 3)$ vertices and m edges, then $m \leq 3n - 6$. Moreover the equality holds if G is maximal planar.*

Proof. We can assume without loss of generality that G is a maximal planar graph; otherwise add new edges without increasing n so that the resulting graph is maximal planar. Consider a plane embedding of G . Every face is bounded by exactly three edges, and each edge is on the boundaries of two faces. Therefore, counting up the edges around each face, we have $3f = 2m$. Applying Theorem 2.2.3, we obtain $m = 3n - 6$. \square

2.2.3 Dual Graph

For a plane graph G , we often construct another graph G^* called the (*geometric*) *dual* of G as follows. A vertex v_i^* is placed in each face F_i of G ; these are the vertices of G^* . Corresponding to each edge e of G we draw an edge e^* which crosses e (but no other edge of G) and joins the vertices v_i^* which lie in the faces F_i adjoining e ; these are the edges of G^* . The edge e^* of G^* is called the *dual edge* of e of G . The construction is illustrated in Fig. 2.13; the vertices v_i^* are represented by small white circles, and the edges e^* of G^* by dotted lines. G^* is not necessarily a simple graph even if G is simple. Clearly the dual G^* of a plane graph G is also a plane graph. One can easily observe the following lemma.

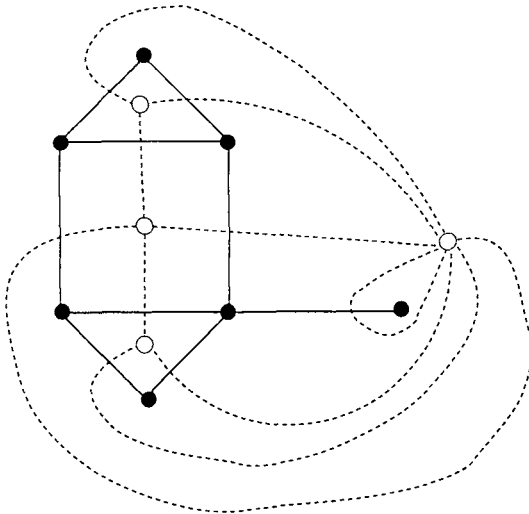


Fig. 2.13 A plane graph G and its dual graph G^* .

Lemma 2.2.5 *Let G be a connected plane graph with n vertices, m edges, and f faces, and let the dual G^* have n^* vertices, m^* edges and f^* faces, then $n^* = f$, $m^* = m$, and $f^* = n$.*

Clearly the dual of the dual of a plane graph G is the original graph G . However a planar graph may give rise to two or more geometric duals since the plane embedding is not necessarily unique.

2.3 Bibliographic Notes

Graph theory is an important mathematical tool which has applications in a wide variety of subjects, and there are several good books on graph theory. Readers interested in graph theory may find the books of Harary [Har72], West [Wes96] and Wilson [Wil96] very useful. The book of Nishizeki and Chiba [NC88] is devoted to the theories and algorithms for planar graphs.

Exercise

1. Show that $m \leq 2n - 4$ for a planar bipartite graph of n vertices and m edges.
2. Show that every planar graph contains a vertex of degree at most five.
3. Suppose G is a maximal planar graph. Prove that G^* is a cubic graph, that is, every vertex of G^* has degree three.
4. Prove that a set of edges in a connected plane graph G forms a spanning tree of G if and only if the duals of the remaining edges form a spanning tree of G^* .
5. Let G be a subdivision of a planar 3-connected cubic graph, and let Γ be an arbitrary plane embedding of the planar graph G . Then prove the following (a) – (c).
 - (a) If C is a 2-legged cycle in Γ , then the legs e_1 and e_2 and the leg-vertices v_1 and v_2 of C are on the outer cycle $C_o(\Gamma)$, and the set of all vertices not in $\Gamma(C)$ induces a chain of G on $C_o(\Gamma)$ with supports v_1 and v_2 .
 - (b) For any chain P on $C_o(\Gamma)$, the outer cycle of the plane graph $\Gamma - V(P)$ is a 2-legged cycle in Γ .
 - (c) Any pair of 2-legged cycles in Γ are not independent.

This page intentionally left blank

Chapter 3

Algorithmic Foundations

3.1 What is an Algorithm?

Consider a computational problem on graphs, such as the planarity testing problem: given a graph, is it planar? If a given graph is small, one may perform the test by hand. However the problem would not be solved without the help of fast computers if a given graph has hundreds or thousands of vertices. Thus we need an “algorithm”: a precise method usable by a computer to solve a problem. An algorithm must solve any instances of a problem and terminate after a finite number of operations.

Direct application of Kuratowski’s theorem (Theorem 2.2.1) provides the following procedure for the planarity testing problem:

Systematically generate all (edge-induced) subgraph J of a given graph G , and check whether each of the J ’s is a subdivision of K_5 or $K_{3,3}$. If there is at least one such J , G is nonplanar; otherwise G is planar.

Clearly a graph G of m edges has 2^m subgraphs, and it is easy to check whether each of them is a subdivision of K_5 or $K_{3,3}$. Thus the procedure above can test the planarity for any given graph and terminate within finite time. That is, it is indeed an algorithm, although it is not efficient.

Let us consider the following classical question: is there a computational problem for which there is no algorithm? A. M. Turing [Tur36] introduced a mathematical model of computers, called a *Turing machine*, and showed that such an unsolvable problem does exist. A typical example is the so-called halting problem: given a computer program with its input, will it ever halt? Turing proved that there is no algorithm that solves correctly all instances of this problem within finite time. Thus there exist problems for which there is no algorithm. However, all of the problems discussed in this book have algorithms.

3.2 Machine Model and Complexity

The planarity testing algorithm mentioned in Section 3.1 would require at least 2^m steps (elementary instructions). Therefore the solution by this algorithm of a modestly sized graph, say having 100 edges, would require more than one hundred centuries, even under the most optimistic assumptions about the speed of computer in the future. Thus that algorithm is completely useless in practice. In contrast efficient algorithms for the problem are known. (See Appendix A.)

In order to study the efficiency of algorithms, we need a model of computation. The earliest and simplest one is the Turing machine. The model was very useful in high-level theoretical studies of computations, such as the existence of an unsolvable problem. However the model is not realistic enough to allow accurate analysis of practical algorithms. In this book we use the so-called *random-access machine* (RAM) model, introduced by Cook and Reckhow [CR76]. The RAM is an abstraction of a general-purpose computer in which each memory cell has a unique address and can store a single integer or real number, and it sequentially performs an access to any cell, an arithmetic operation, or a Boolean operation, following a finite sequence of instructions, called a program. We assume that each of the operations requires *unit time*, and each of the memory cells uses *unit space*. That is, our model is the *unit-cost* RAM.

The most widely accepted complexity measure for an algorithm are the running time and storage space. The *running time* is the number of operations it performs before producing the final answer, while the *storage space* is the number of memory cells it uses. The number of operations or cells required by an algorithm is not the same for all problem instances. For example, in the planarity testing, the number of operations required to test the planarity of a graph with n vertices and m edges may vary considerably with graphs, even if n and m are kept constant. Thus, we consider all inputs of a given size together, and we define the complexity of the algorithm for that input size to be the worst case behavior of the algorithm on any of these inputs. Then the running time (or storage space) is a function of size n of the input.

3.2.1 The $O(\)$ notation

In analyzing the complexity of an algorithm, we are often interested only in the “asymptotic behavior,” that is, the behavior of the algorithm when

applied to very large inputs. To deal with such a property of functions we shall use the following notations for asymptotic running time. Let $f(n)$ and $g(n)$ are the functions from the positive integers to the positive reals, then we write $f(n) = O(g(n))$ if there exists positive constants c_1 and c_2 such that $f(n) \leq c_1 g(n) + c_2$ for all n . Thus the running time of an algorithm may be bounded from above by phrasing like “takes time $O(n^2)$.”

3.2.2 Polynomial Algorithms

An algorithm is said to be *polynomially bounded* (or simply *polynomial*) if its complexity is bounded by a polynomial of the size n of a problem instance. Examples of such complexities are $O(n)$, $O(n \log n)$, $O(n^{100})$, etc. The remaining complexities are usually referred as *exponential* or *nonpolynomial*. Examples of such complexity are $O(2^n)$, $O(n!)$, etc.

When the running time of an algorithm is bounded by $O(n)$, we call it a *linear-time* algorithm or simply a *linear* algorithm.

3.2.3 NP-Complete Problems

There are a number of interesting computational problems for which it has not been proved whether there is a polynomial-time algorithm or not. Most of them are “NP-complete,” which we will briefly explain in this section.

The state of algorithms consists of the current values of all the variables and the location of the current instruction to be executed. A *deterministic algorithm* is one for which each state, upon execution of the instruction, uniquely determines at most one of the next states. All computers, which exist now, run deterministically. A problem Q is in the class P if there exists a deterministic polynomial-time algorithm which solves Q .

In contrast, a *nondeterministic algorithm* is one for which a state may determine many next states simultaneously. We may regard a nondeterministic algorithm as having the capability of branching off into many copies of itself, one for the each next state. Thus, while a deterministic algorithm must explore a set of alternatives one at a time, a nondeterministic algorithm examines all alternatives at the same time. A problem Q is in the class NP if there exists a nondeterministic polynomial-time algorithm which solves Q . Clearly $P \subseteq NP$.

Among the problems in NP are those that are hardest in the sense that if one can be solved in polynomial-time then so can every problem in NP . These are called *NP-complete* problems. The class of NP-complete

problems has the very interesting properties.

- (a) No NP-complete problem can be solved by any known polynomial algorithm.
- (b) If there is a polynomial algorithm for any NP-complete problem, then there are polynomial algorithms for all NP-complete problems.

Sometimes we may be able to show that, if problem Q is solvable in polynomial time, all problems in NP are so, but we are unable to argue that $Q \in \text{NP}$. So Q does not qualify to be called NP-complete. Yet, undoubtedly Q is as hard as any problem in NP. Such a problem Q is called *NP-hard*.

3.3 Data Structures and Graph Representation

In this section we first give a brief account of basic data structures, and then show two methods to represent a graph in a computer. A vector or a set of variables is usually stored as a (1-dimensional) *array*, and a matrix is stored as a *2-dimensional* array. The main feature of an array is its index capability. The indices should uniquely determine the location of each entry, and accessing an entry is done in a constant amount of time.

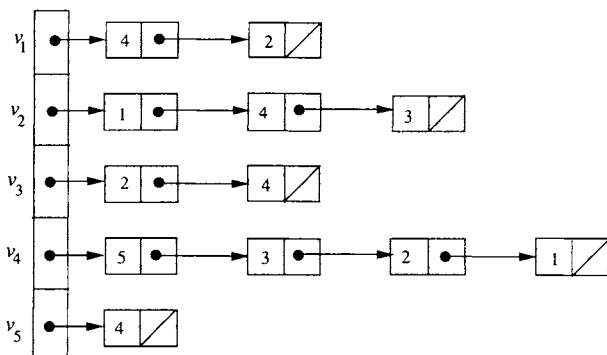
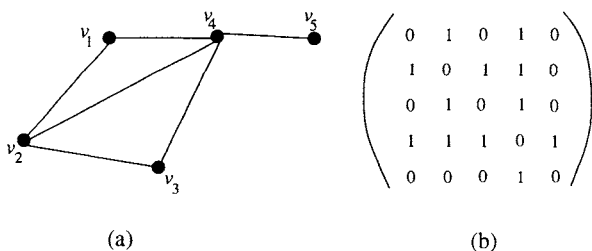
A *list* is a data structure which consists of homogeneous records which are linked together in a linear fashion. Each record contains one or more items of data and one or more of pointers. Figure 3.1(c) shows five singly linked lists; each record has a single forward pointer indicating the address of the memory cell of the next record. In a *doubly linked list* each record has forward and backward pointers, and we can delete a record or insert a record without being given the location of the previous record.

A stack and a queue are two special types of lists. A *stack* is a list in which we are only permitted to insert and delete elements at one end, called the *top* of the stack. Thus a stack functions in a last-in, first-out manner with respect to insertion and deletion. A *queue* is a list in which we are only permitted to insert at one end, called the *tail* of the queue, and delete from the other end, called the *head* of the queue. That is, a queue functions in a first-in, first-out manner. Both a stack and a queue are realized by a list with pointers indicating the current locations of the top, head or tail.

We measure the complexity of an algorithm as a function of the size of the input of an algorithm. What is the size of the input of a graph problem? To represent a graph by a computer, we must encode it as a sequence of

symbols over some fixed alphabet such as bits or typewriter symbols. The size is the length of the sequence.

A graph may be represented in many ways. For example we can associate with a graph $G = (V, E)$ its $n \times n$ adjacency matrix $A = [a_{ij}]$ such that $a_{ij} = 1$ if $(v_i, v_j) \in E$, and $a_{ij} = 0$ otherwise. Figure 3.1(a) illustrates a graph and Fig. 3.1(b) illustrates its adjacency matrix. If G is a simple graph, then the main diagonal of A is all zeros and A is symmetric. An adjacency matrix uses n^2 space to represent a graph of n vertices. It is not economical when a graph is *sparse*, that is, the number m of edges is far less than $n(n - 1)/2$. In this case simply listing the edges one by one would be much more efficient. If A is stored in a computer as a 2-dimensional array, then only one step is required for the statement “Is $(v_i, v_j) \in E$?” or “Erase the edge (v_i, v_j) .” However, scanning all the neighbors of a vertex v requires n steps even if degree $d(v)$ is far less than n .



(c)

Fig. 3.1 (a) Graph, (b) adjacency matrix, and (c) adjacency lists.

Another useful way of representing a graph is by its *adjacency lists*. For each vertex $v \in V$, we record the set $N(v)$ of v 's neighbors. The sets are stored in a computer as lists $Adj(v)$, as illustrated in Fig. 3.1(c). The space requirement for the adjacency lists is

$$O\left(\sum_{v \in V} (1 + d(v))\right) = O(n + m).$$

Thus the representation is much more economical than the adjacency matrix if a graph is sparse. Scanning the adjacency list $Adj(v)$ can be done in $d(v)$ steps, but the statement “Is $(v, w) \in E$?” requires $d(v)$ steps.

We can assume that $m \geq n/2$ (for instance, if our graph has no isolated vertices). Therefore m is a reasonable approximation to the size of a graph. For a planar graph we usually use n as the approximated size of an input graph since $m < 3n$ by Corollary 2.2.4. Thus we analyze the complexity of algorithms using n (for planar graphs) or n and m (for general graphs) as parameters.

A graph algorithm is said to be *linear* if it runs in time $O(n + m)$ on a graph of n vertices and m edges. This is usually the best that one could expect for a graph algorithm.

3.4 Exploring a Graph

When designing algorithms on graphs, we often need a method for exploring the vertices and edges of a graph. Most important are the depth-first search (DFS) and breadth-first search (BFS). In both methods, each edge is traversed exactly once in the forward and reverse directions, and all vertices are visited. Thus both run in linear time. The choice of which method to use must depend on the problem.

3.4.1 Depth-First Search

Consider visiting the vertices of a graph in the following way. We select and visit a starting vertex v . Then we select any edge (v, w) incident to v and visit w . In general, suppose x is the most recently visited vertex. The search is continued by selecting some unexplored edge (x, y) incident to x . If y has been previously visited, we find another new edge incident to x . If y has not been previously visited, then we visit y and begin the search anew starting at y . After completing the search through all paths beginning at

y , the search returns to x , the vertex from which y was first reached. The process of selecting unexplored edges incident to x is continued until the list of these edges is exhausted. This method is called the *depth-first search* since we continue searching deeper direction as long as possible.

A depth-first search of an undirected graph $G = (V, E)$ partitions E into two sets T and B ; T comprises a spanning forest of G in general; in particular, if G is connected, then T comprises a spanning tree of G . The edge (x, y) is placed into T if vertex y was visited for the first time from x . The edges in T are called *tree edges*. The remaining edges, called *back edges*, are placed into B . The algorithm is given below.

Algorithm Depth-First-Search

begin

Set $T = \emptyset$;

for each vertex v in V **do** mark v “new”;

while there exists a vertex v in V marked “new” **do**

 Search(v)

end.

procedure Search(v);

begin

 Mark v “old”;

for each vertex w in adjacency list $Adj(v)$ **do**

if w is marked “new” **then**

begin

 Add (v, w) to T ;

 Search(w);

end

end;

For example, the depth-first search starting with vertex v_1 of graph $G = (V, E)$ in Fig. 3.1(a) partitions E into a spanning tree T and a set B of back edges; T and B are drawn by solid and dotted lines in Fig. 3.2, respectively.

3.4.2 Breadth-First Search

In implementing the breadth-first search (BFS) we choose an arbitrary vertex and put it on a queue of vertices to be visited. We repeatedly delete the vertex x at the head of the queue Q , and scan the adjacency list $Adj(x)$

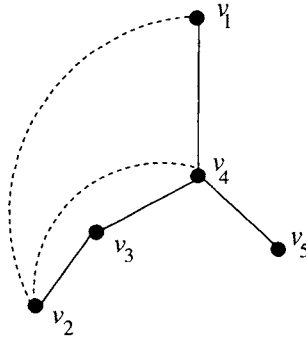


Fig. 3.2 DFS: tree edges and back edges.

with inserting to Q all neighbors of x which have never been inserted to Q . For simplicity we may assume that a given graph $G = (V, E)$ is connected; otherwise repeatedly apply BFS to each component. BFS partitions the edges E of a connected graph into a spanning tree T and a set B of back edges. The algorithm is given below.

Algorithm Breadth-First-Search

begin

Set $T = \emptyset$;

Set $Q =$ empty queue;

for each vertex $v \in V$ **do** mark v “not reached”;

Choose an arbitrary vertex $r \in V$ as a starting vertex and
add r to queue Q ;

{ r is a root of tree T }

Mark r “reached”;

Set $\text{LEVEL}(r)=1$;

while Q is nonempty **do**

begin

Let x be the head of Q ;

Set $Q = Q - x$;

Scan(x);

end

end.

procedure Scan(x)

begin

for each $y \in \text{Adj}(x)$ **do**

TEAM LING - LIVE, INFORMATIVE, NON-COST AND GENUINE !

```

if  $y$  is marked “not reached” then
  begin
    Add edge  $(x, y)$  to  $T$ ;
    Set  $\text{LEVEL}(y) = \text{LEVEL}(x) + 1$ ;
    Add  $y$  to  $Q$ ;
    Mark  $y$  “reached”;
  end
end.

```

Thus the search scans the adjacency lists of vertices in a first reached, first scanned manner; in this sense it is called the breadth-first search. The breadth-first search tree T and the vertex levels have the following properties:

- Every edge of G , whether tree or back edge, joins two vertices whose levels differ by at most one;
- The level of v equals the length (i.e. number of edges) of the shortest path from the root r to v in G .

As well as DFS, BFS explores each edge exactly twice, in the forward and backward directions, and hence runs in linear time.

The breadth-first search starting with vertex v_1 of a graph $G = (V, E)$ in Fig. 3.1(a) partitions E into a spanning tree T and a set B of back edges. T and B are drawn by solid and dotted lines in Fig. 3.3, where the numbers next to vertices are levels.

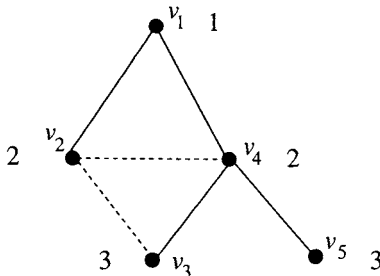


Fig. 3.3 BFS: tree edges and back edges.

3.5 Data Structures for Plane Graphs

In Section 3.3 we have seen two data structures to represent a graph: an adjacency matrix representation and a list representation. The two plane graphs in Figs. 3.1(a) and 3.4(a) have the same adjacency matrix representation as illustrated in Fig. 3.1(b). This implies that the two graphs in Fig. 3.1(a) and 3.4(a) are the same graph. However, they are two different plane graphs although the adjacency list depicted in Fig. 3.1(c) represents both the graphs. Now the question is how can we represent a plane graph, i.e., a particular embedding of a planar graph? One can observe that if we preserve the ordering of the edges incident to a vertex in a plane embedding of a graph then that representation truly represents the embedding. The adjacency list representations in Figs. 3.1(c) and 3.4(b) represent the plane graphs in Figs. 3.1(a) and 3.4(a), respectively, since they preserve the clockwise ordering of the edges incident to each vertex.

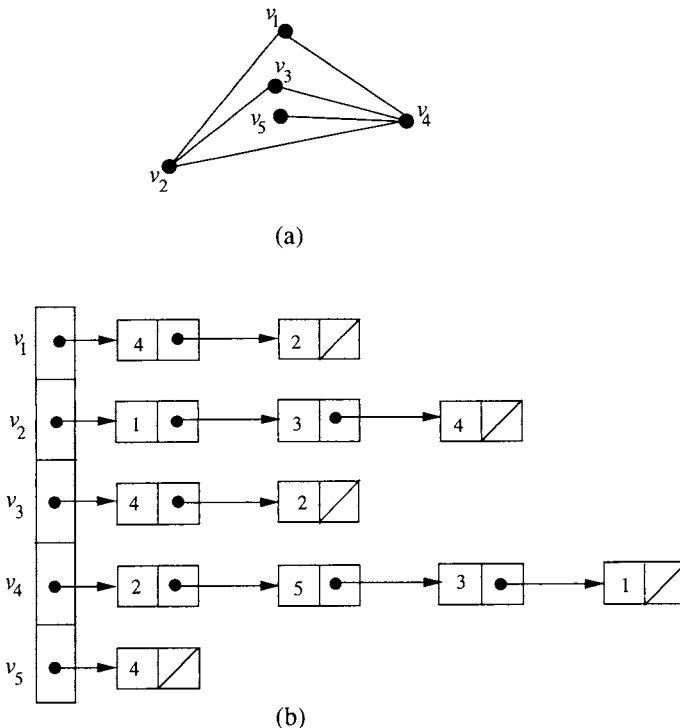


Fig. 3.4 (a) Plane graph, and (b) adjacency lists.

In many drawing algorithms of plane graphs we often need to traverse the faces of a plane graph efficiently. Assume that we want to clockwise traverse a face F in a plane graph G starting from a vertex v_1 , as illustrated in Fig. 3.5(a). Starting from v_1 we first traverse an edge (v_1, v_4) and reach at vertex v_4 . We then need to traverse edge (v_4, v_2) . An efficient algorithm needs to find the edge (v_4, v_2) in constant time. Clearly the edge (v_1, v_4) follows edge (v_4, v_2) in the clockwise ordering of edges incident to vertex v_4 . In other words, the edge (v_4, v_2) is counterclockwise next to edge (v_1, v_4) in the adjacent list of v_4 . However, we cannot find it in constant time using the data structure in Fig. 3.1(c). We can find the edge (v_4, v_2) in

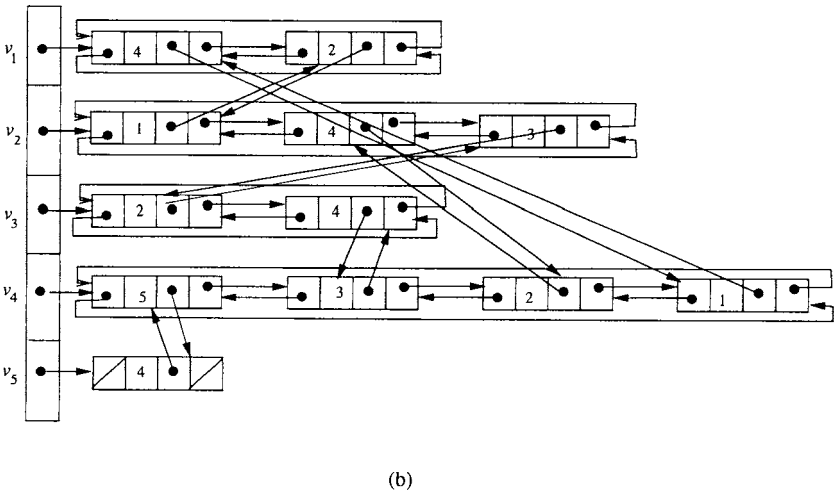
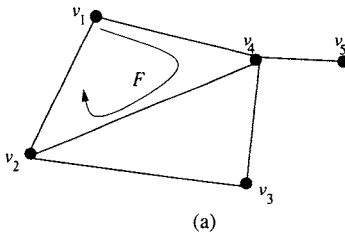


Fig. 3.5 Illustration of a data structure for traversing a face in a plane graph efficiently.

constant time if we have a data structure of G as illustrated in Fig. 3.5(b). In this representation both the clockwise and counterclockwise ordering of edges incident to a vertex is preserved using a doubly circular linked list of

neighbors of the vertices; traversing the list forward and backward we get clockwise and counterclockwise ordering, respectively. The two entries for an edge in the representation are also linked so that one of them can be accessed from the other directly. Using the data structure in Fig. 3.5(b) we can find the edge (v_4, v_2) as follows. From Entry 4 in the adjacency list of v_1 go to Entry 1 in the adjacency list of v_4 directly using the link between them. Then we can find v_2 , since Entry 2 is counterclockwise next to Entry 1 in the adjacency list of v_4 .

3.6 Bibliographic Notes

There are several good books on data structures and algorithms. Just for examples we mention the book of Aho *et al.* [AHU74], Cormen *et al.* [CLR90], and Sedgewick and Flajolet [SF96]. The recent book on algorithm design by Goodrich and Tamassia [GT02] presents various algorithms with beautiful illustrations.

Exercise

1. Using DFS, develop an algorithm to determine whether a given graph is connected or not.
2. Design an algorithm to find all connected components of a given graph.
3. How can you find a cycle in a given graph using DFS if the graph has a cycle?
4. Let s be a designated vertex in a connected graph $G = (V, E)$. Design an $O(n + m)$ time algorithm to find a path between s and v with the minimum number of edges for all vertices $v \in V$.
5. Design an algorithm to traverse all faces in a plane graph in linear time.

Chapter 4

Straight Line Drawing

4.1 Introduction

A *straight line drawing* of a plane graph is a drawing in which each edge is drawn as a straight line segment without edge-crossings, as illustrated in Fig. 1.5. Wagner [Wag36], Fáry [Far48] and Stein [Ste51] independently proved that every planar graph G has a straight line drawing. Their proofs immediately yield polynomial-time algorithms to find a straight line drawing of a given plane graph. However, the area of a rectangle enclosing a drawing on an integer grid obtained by these algorithms is not bounded by any polynomial in the number n of vertices in G . In fact, it remained as an open problem for long time to obtain a drawing of area bounded by a polynomial. In 1990, de Fraysseix *et al.* [FPP90] and Schnyder [Sch90] showed by two different methods that every planar graph of $n \geq 3$ vertices has a straight line drawing on an integer grid of size $(2n - 4) \times (n - 2)$ and $(n - 2) \times (n - 2)$, respectively. The two methods can be implemented as linear-time algorithms, and are well known as the “shift method” and the “realizer method,” respectively. We present the shift method in Section 4.2 and the realizer method in Section 4.3.

A natural question arises: what is the minimum size of a grid required for a straight line drawing? de Fraysseix *et al.* showed that, for each $n \geq 3$, there exists a plane graph of n vertices, for example nested triangles, which needs a grid of size at least $\lfloor 2(n - 1)/3 \rfloor \times \lfloor 2(n - 1)/3 \rfloor$ for any grid drawing [CN98, FPP90]. It has been conjectured that every plane graph of n vertices has a grid drawing on a $\lceil 2n/3 \rceil \times \lceil 2n/3 \rceil$ grid, but it is still an open problem. On the other hand, a restricted class of graphs has a more compact grid drawing. For example, if G is a 4-connected plane graph, then G has a more compact grid drawing. We consider such a drawing in Section 4.4.

4.2 Shift Method

In this section we describe a constructive proof for the theorem by de Fraysseix *et al.* [FPP90] that every plane graph G of $n \geq 3$ vertices has a straight line grid drawing of size $(2n - 4) \times (n - 2)$, and present a linear-time implementation of an algorithm for finding such a drawing [CP95]. If G is not triangulated, then we obtain a triangulated plane graph G' from G by adding dummy edges to G . From a straight line grid drawing of G' we can immediately obtain a straight line grid drawing of G by deleting the dummy edges. Therefore it is sufficient to prove that a triangulated plane graph G of n vertices has a straight line grid drawing of size $(2n - 4) \times (n - 2)$. To construct such a drawing, de Fraysseix *et al.* introduced an ordering of vertices called a “canonical ordering” and installed vertices one by one in the drawing according to the ordering.

In Section 4.2.1 we present a canonical ordering, and in Section 4.2.2 we present the algorithm of de Fraysseix *et al.* We present a linear time implementation of the algorithm in Section 4.2.3.

4.2.1 Canonical Ordering

For a cycle C in a graph, an edge joining two non-consecutive vertices in C is called a *chord* of C . For a 2-connected plane graph G , we denote by $C_o(G)$ the *outer cycle* of G , that is, the boundary of the outer face of G . A vertex on $C_o(G)$ is called an *outer vertex* and an edge on $C_o(G)$ is called an *outer edge*. A plane graph is *internally triangulated* if every inner face is a triangle.

Let $G = (V, E)$ be a triangulated plane graph of $n \geq 3$ vertices, as illustrated in Fig. 4.1. Since G is triangulated, there are exactly three vertices on $C_o(G)$. One may assume that these three vertices, denoted by v_1, v_2 and v_n , appear on $C_o(G)$ counterclockwise in this order. Let $\pi = (v_1, v_2, \dots, v_n)$ be an ordering of all vertices in G . For each integer $k, 3 \leq k \leq n$, we denote by G_k the plane subgraph of G induced by the k vertices v_1, v_2, \dots, v_k . Then $G_n = G$. We call π a *canonical ordering* of G if the following conditions (co1)–(co3) hold for each index $k, 3 \leq k \leq n$:

- (co1) G_k is 2-connected and internally triangulated;
- (co2) (v_1, v_2) is an outer edge of G_k ; and
- (co3) if $k + 1 \leq n$, then vertex v_{k+1} is located in the outer face of G_k , and all neighbors of v_{k+1} in G_k appear on $C_o(G_k)$ consecutively.

An example of a canonical ordering is illustrated for a triangulated plane graph of $n = 16$ vertices in Fig. 4.1.

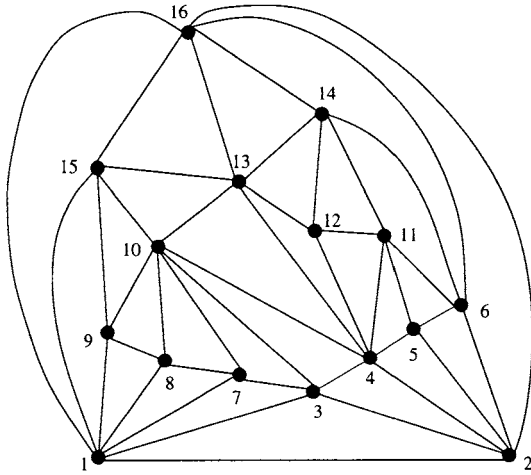


Fig. 4.1 A canonical ordering of a triangulated plane graph of $n = 16$ vertices.

We now have the following lemma.

Lemma 4.2.1 *Every triangulated plane graph G has a canonical ordering.*

Proof. Obviously G has a canonical ordering if $n = 3$. One may thus assume that $n \geq 4$. Since $G = G_n$, clearly (co1)–(co3) hold for $k = n$. We then choose the $n - 3$ inner vertices $v_{n-1}, v_{n-2}, \dots, v_3$ in this order, and show that (co1)–(co3) hold for $k = n - 1, n - 2, \dots, 3$.

Assume for inductive hypothesis that the vertices $v_n, v_{n-1}, \dots, v_{k+1}$, $k + 1 \geq 4$, have been appropriately chosen, and that (co1)–(co3) hold for k . If one can choose as v_k a vertex $w \neq v_1, v_2$ on the cycle $C_o(G_k)$ which is not an end of a chord of $C_o(G_k)$, as illustrated in Fig. 4.2(a), then clearly (co1)–(co3) hold for $k - 1$ since $G_{k-1} = G_k - v_k$. Thus it suffices to show that there is such a vertex w .

Let $C_o(G_k) = w_1, w_2, \dots, w_t$, where $w_1 = v_1$ and $w_t = v_2$. If $C_o(G_k)$ has no chord, then any of the vertices w_2, w_3, \dots, w_{t-1} is such a vertex w . One may thus assume that $C_o(G_k)$ has a chord. Then G_k has a “minimal” chord (w_p, w_q) , $p + 2 \leq q$, such that none of the vertices $w_{p+1}, w_{p+2}, \dots, w_{q-1}$ is an end of a chord, as illustrated in Fig. 4.2(b) where chords are drawn by thick lines. Then any of the ver-

tices $w_{p+1}, w_{p+2}, \dots, w_{q-1}$ is such a vertex w . □

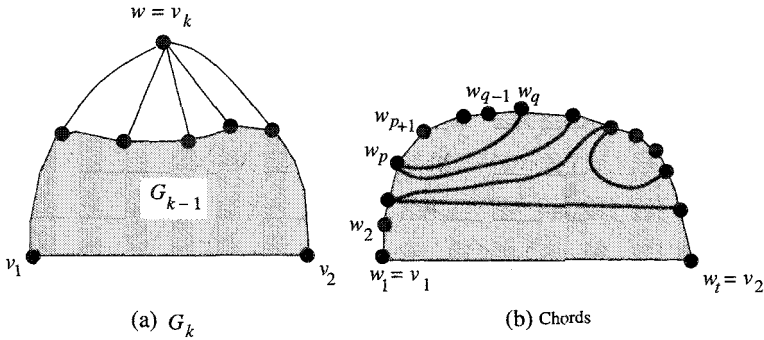


Fig. 4.2 Graph G_k and chords.

The following algorithm computes a canonical ordering of a triangulated plane graph $G = (V, E)$. For each vertex v , we keep the following variables:

- $mark(v) = true$ if v has been added to the ordering, and *false* otherwise;
- $out(v) = true$ if v is an outer vertex of a current plane graph, and *false* otherwise; and
- $chords(v) =$ the number of chords of the outer cycle whose end vertex is v .

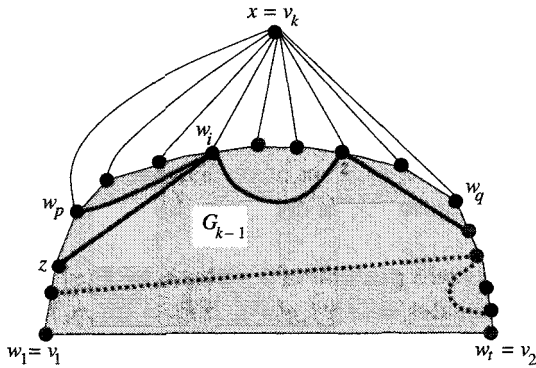
The algorithm is as follows.

Algorithm Canonical-Ordering(G)

begin

- 1 Let v_1, v_2 and v_n be the vertices appearing on the outer cycle counterclockwise in this order;
- 2 Set $chords(x) = 0$, $out(x) = false$, and $mark(x) = false$ for all vertices $x \in V$;
- 3 Set $out(v_1) = true$, $out(v_2) = true$, and $out(v_n) = true$;
- 4 **for** $k = n$ **down to** 3 **do**
- 5 **begin**
- 6 Choose any vertex x such that $mark(x) = false$, $out(x) = true$, $chords(x) = 0$, and $x \neq v_1, v_2$;
- 7 Set $v_k = x$ and $mark(x) = true$;
- 8 Let $C_o(G_{k-1}) = w_1, w_2, \dots, w_t$, where $w_1 = v_1$ and

$w_t = v_2$;
 8 Let w_p, w_{p+1}, \dots, w_q be the neighbors of v_k which
 have $mark(w_i) = false$;
 {They are consecutive on $C_o(G_{k-1})$, as illustrated in
 Fig. 4.3.}
 9 For each vertex $w_i, p < i < q$, set $out(w_i) = true$, and
 update the variable $chords$ for w_i and its neighbors.
 {The details will be given in the proof of Lemma 4.2.2.}
 end
 end.

Fig. 4.3 G_k .

Lemma 4.2.2 *Algorithm Canonical-Ordering(G) computes a canonical ordering of a triangulated plane graph G in time $O(n)$.*

Proof. By the proof of Lemma 4.2.1, there always exists a vertex x satisfying the conditions in Line 5. This provides the correctness of Algorithm **Canonical-Ordering**.

To find the vertex x in Line 5 in time $O(1)$, we maintain a list containing all vertices satisfying the conditions.

Line 9 is implemented as follows. If $q = p + 1$, then we decrease $chords(w_p)$ and $chords(w_q)$ by one, because the edge (w_p, w_q) on the current outer cycle $C_o(G_{k-1})$ was a chord (w_p, w_q) on the previous outer cycle $C_o(G_k)$. If $q > p + 1$, then we update variable $chords$ for each vertex $w_i, p < i < q$, and each neighbor z of w_i . For each vertex $w_i, p < i < q$, we inspect its neighbors z . If $out(z) = true$ and $z \neq w_{i-1}, w_{i+1}$, then

(w_i, z) is a chord of $C_o(G_{k-1})$ and hence we increase $\text{chords}(w_i)$ by one. If $\text{out}(z) = \text{true}$, $z \neq w_{i-1}, w_{i+1}$ and $z \neq w_{p+1}, w_{p+2}, \dots, w_{q-1}$, then we increase $\text{chords}(z)$ by one. (In Fig. 4.3 new chords on $C_o(G_{k-1})$ are drawn by thick solid lines, and chords on $C_o(G_k)$ are drawn by thick dotted lines.)

The update for w_i and its neighbors z above can be done in time $O(d(w_i))$, where $d(w_i)$ is the degree of w_i in G . Since this is done only once for every vertex w_i in G , the total running time of Line 9 is $O(n)$. Note that $\sum_{w_i \in V} d(w_i) = 2m \leq 2 \cdot 3n = O(n)$ by Corollary 2.2.4. Clearly all other lines can be done in time $O(n)$. Hence **Canonical-Ordering** takes time $O(n)$. \square

4.2.2 Shift Algorithm

In this section we describe the shift algorithm given by de Fraysseix *et al.* [FPP90]. The algorithm embeds G , one vertex at a time in a canonical order $\pi = (v_1, v_2, \dots, v_n)$ at each stage, adjusting the current partial embedding. With each vertex v_i , a set of vertices need to be moved whenever the position of v_i is adjusted. We denote by $L(v_i)$ the set of such vertices. Note that $v_i \in L(v_i)$.

We denote the current position of a vertex v by $P(v)$; $P(v)$ is expressed by its x - and y -coordinates as $(x(v), y(v))$. If $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ are two grid points whose Manhattan distance is even, then the straight line with slope $+1$ through P_1 and the straight line with slope -1 through P_2 intersects at a grid point, which is denoted by $\mu(P_1, P_2)$. Clearly

$$\mu(P_1, P_2) = \left(\frac{1}{2}(x_1 - y_1 + x_2 + y_2), \frac{1}{2}(-x_1 + y_1 + x_2 + y_2) \right). \quad (4.1)$$

We are now ready to describe the drawing algorithm.

First we draw G_3 by a triangle as follows. Set $P(v_1) = (0, 0)$, $P(v_2) = (2, 0)$, $P(v_3) = (1, 1)$, and $L(v_i) = \{v_i\}$ for $i = 1, 2, 3$. (See Fig. 4.4(a).)

Assume that $k - 1 \geq 3$ and we have embedded G_{k-1} in such a way that the following conditions hold:

- (e1) $P(v_1) = (0, 0)$ and $P(v_2) = (2k - 6, 0)$;
- (e2) $x(w_1) < x(w_2) < \dots < x(w_t)$, where $C_o(G_{k-1}) = w_1, w_2, \dots, w_t$, $w_1 = v_1$ and $w_t = v_2$; and
- (e3) each edge (w_i, w_{i+1}) on $C_o(G_{k-1})$ is drawn by a straight line having slope either $+1$ or -1 , as illustrated in Fig. 4.5(a).

We now explain how to install v_k to a drawing of G_{k-1} . Let

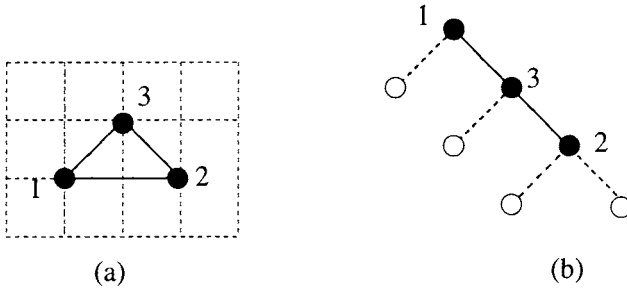


Fig. 4.4 (a) Graph G_3 , and (b) binary tree T for G_3 .

w_p, w_{p+1}, \dots, w_q be the neighbors of v_k on $C_o(G_{k-1})$, as illustrated in Fig. 4.5(a). We say that the vertex v_k covers the vertices $w_{p+1}, w_{p+2}, \dots, w_{q-1}$. By (e3) the Manhattan distance between w_p and w_q is even, and hence $\mu(w_p, w_q)$ is a grid point. However, if we installed v_k at $\mu(w_p, w_q)$, then the straight line segment $w_p v_k$ would overlap with $w_p w_{p+1}$, because $w_p w_{p+1}$ may have slope $+1$ as illustrated in Fig. 4.5(a). We thus shift vertices $w_1 (= v_1), w_2, \dots, w_p$ together with some inner vertices to the left by one, as illustrated in Fig. 4.5(b). Similarly we shift vertices $w_q, w_{q+1}, \dots, w_t (= v_2)$ together with some inner vertices to the right by one. We then install v_k at the grid point $\mu(w_p, w_q)$ for the new positions of w_p and w_q . More precisely, we execute the following Steps 1–4.

Step 1: for each $v \in \bigcup_{i=1}^p L(w_i)$ do $x(v) = x(v) - 1$;

Step 2: for each $v \in \bigcup_{i=q}^t L(w_i)$ do $x(v) = x(v) + 1$;

Step 3: $P(v_k) = \mu(w_p, w_q)$

Step 4: $L(v_k) = \{v_k\} \cup (\bigcup_{i=p+1}^{q-1} L(w_i))$

Figure 4.5(a) depicts a drawing of G_{k-1} , and Fig. 4.5(b) depicts a drawing of G_k obtained by Steps 1–4. The Manhattan distance between w_p and w_q was even in the drawing of G_{k-1} . Vertex w_p is moved to the left by one by Step 1, and w_q is moved to the right by one by Step 2. Therefore the Manhattan distance between w_p and w_q is even in the drawing of G_k , and hence $\mu(w_p, w_q)$ is a grid point as in Fig. 4.5(b). Vertices w_1, w_2, \dots, w_p are moved to the left by one, and w_q, w_{q+1}, \dots, w_t are moved to the right by one. However, the positions of all vertices $w_{p+1}, w_{p+2}, \dots, w_{q-1}$ are unchanged by Steps 1 and 2; they are indi-

cated by double circles in Fig. 4.5. Therefore the slopes of all edges $(w_{p+1}, w_{p+2}), (w_{p+2}, w_{p+3}), \dots, (w_{q-2}, w_{q-1})$ have absolute value 1; these edges are drawn by thick solid lines in Fig. 4.5. The slopes of edges (w_p, w_{p+1}) and (w_{q-1}, w_q) have absolute values smaller than 1 in the drawing of G_k as illustrated in Fig. 4.5(b). Thus all the vertices w_p, w_{p+1}, \dots, w_q are visible from the point $\mu(w_p, w_q)$, and hence one can draw all edges $(v_k, w_p), (v_k, w_{p+1}), \dots, (v_k, w_q)$ by straight line segments without edge crossings as illustrated in Fig. 4.5(b).

Clearly $P(v_1) = (-1, 0)$ in Fig. 4.5(b). Replace Steps 1 and 2 above by the following Steps 1' and 2' to make $P(v_1) = (0, 0)$ by translating the drawing in Fig. 4.5(b) to the right by one.

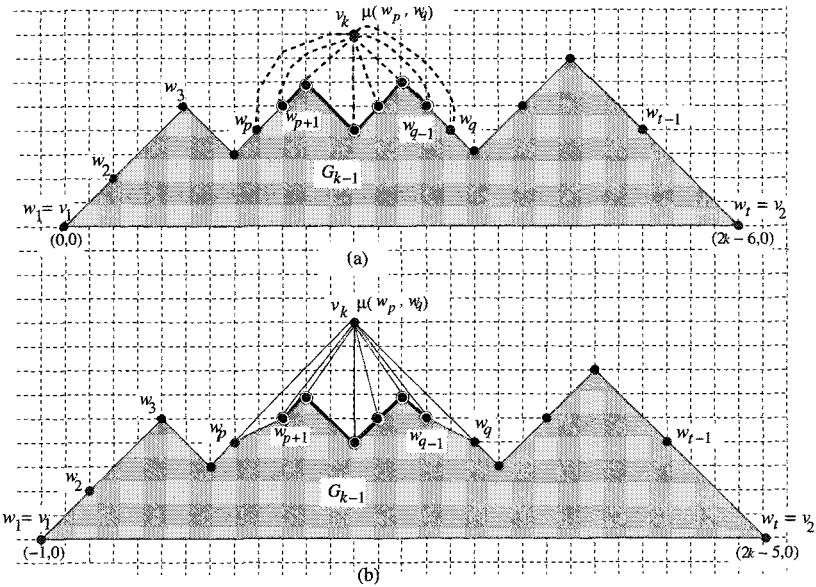


Fig. 4.5 (a) G_k before Shift, and (b) G_k after Shift.

Step 1': for each $v \in \bigcup_{i=p+1}^{q-1} L(w_i)$ do $x(v) = x(v) + 1$;

Step 2': for each $v \in \bigcup_{i=q}^t L(w_i)$ do $x(v) = x(v) + 2$.

Then (e1), (e2) and (e3) hold for G_k .

Figure 4.6(a) illustrates $L(w_i)$ for all outer vertices w_i of G_{15} for the graph G in Fig. 4.1.

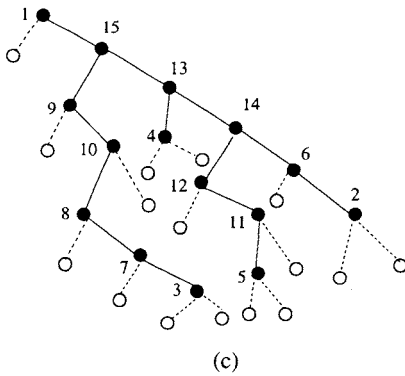
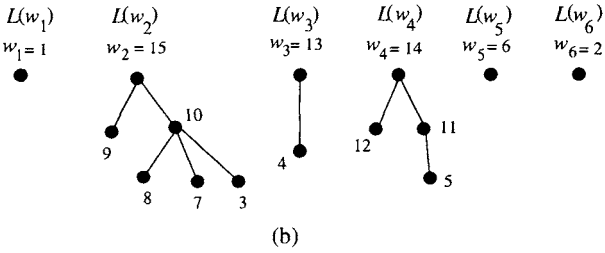
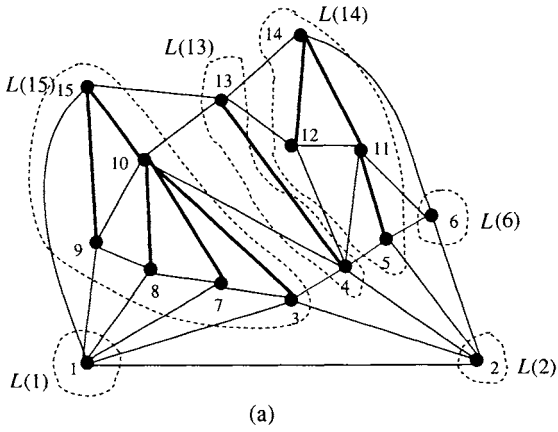


Fig. 4.6 (a) Graph G_{15} for G in Fig. 4.1, (b) forest F , and (c) binary tree T .

The following lemma ensures that G_{k-1} , $3 \leq k \leq n$, remains to be a straight line grid drawing after Steps 1' and 2' are executed and hence G_k is a straight line grid drawing.

Lemma 4.2.3 *Let G_k , $3 \leq k \leq n$, be straight line grid embedded as described above. Let $C_o(G_k)$ contain t' vertices, and let $\delta_1 \leq \delta_2 \leq \dots \leq \delta_{t'}$ be any non-decreasing sequence of t' non-negative integers. If, for each i , we shift the vertices in $L(w_i)$ by δ_i to the right, then we again obtain a straight line grid embedding of G_k .*

Proof. The proof is by induction on k . For G_3 the lemma is obvious. So suppose that it holds for G_{k-1} , $k \geq 4$. Let $C_o(G_{k-1}) = w_1, w_2, \dots, w_t$ as in the algorithm. We are about to add v_k to the drawing of G_{k-1} by Steps 1', 2', 3 and 4. Clearly $C_o(G_k) = w_1, w_2, \dots, w_p, v_k, w_q, w_{q+1}, \dots, w_t$. Let $\delta_1 \leq \delta_2 \leq \dots \leq \delta_p \leq \delta \leq \delta_q \leq \dots \leq \delta_t$ be a fixed sequence of t' non-negative integers for which we translate each $L(w_i)$ by δ_i and $L(v_k)$ by δ for G_k . We show that G_k remains straight line grid embedded.

Take a sequence $\delta'_1 \leq \delta'_2 \leq \dots \leq \delta'_t$ for G_{k-1} as follows:

$$\delta'_i = \begin{cases} \delta_i & \text{if } 1 \leq i \leq p; \\ \delta + 1 & \text{if } p + 1 \leq i \leq q - 1; \\ \delta_i + 2 & \text{if } q \leq i \leq t. \end{cases}$$

Then $\delta'_1, \delta'_2, \dots, \delta'_t$ is a non-decreasing sequence of t non-negative integers. By induction, if we translate by δ'_i the sets $L(w_i)$ for the drawing of G_{k-1} before executing Steps 3 and 4 for G_{k-1} , then G_{k-1} remains straight line grid embedded. Thus G_k is straight line grid embedded, because v_k is translated by δ , $L(v_k) = \{v_k\} \cup (\cup_{i=p+1}^{q-1} L(w_i))$, and hence v_k moves rigidly with w_p, w_{p+1}, \dots, w_q . \square

So, in the end we have a straight line embedding of $G = G_n$ such that $P(v_1) = (0, 0)$ and $P(v_2) = (2n - 4, 0)$. By (e3), $P(v_n) = (n - 2, n - 2)$. Therefore, the whole graph G is drawn in a $(2n - 4) \times (n - 2)$ grid, as illustrated in Fig. 4.7.

It is easy to implement the drawing algorithm described above in time $O(n^2)$.

4.2.3 Linear-Time Implementation

In this section we describe a linear-time implementation of the straight line drawing algorithm in Section 4.2.2 [CP95].

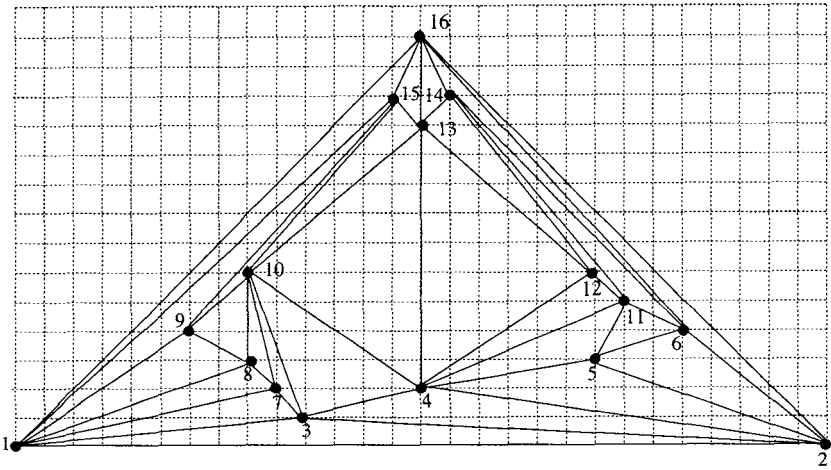


Fig. 4.7 Grid drawing of the plane graph in Fig 4.1.

We assume that G is already triangulated and embedded in the plane, and that a canonical ordering $\pi = (v_1, v_2, \dots, v_n)$ of G is given.

We view the family of sets $L(w_1), L(w_2), \dots, L(w_t)$ for the outer vertices w_1, w_2, \dots, w_t of graph G_k as a forest F in G_k consisting of trees $L(w_1), L(w_2), \dots, L(w_t)$ rooted at the vertices w_1, w_2, \dots, w_t . For G_{15} in Fig. 4.6(a) the forest F is drawn by thick solid lines in Fig. 4.6(a) and is depicted also in Fig. 4.6(b). The children of root w_i of a tree $L(w_i)$ are the vertices that w_i covers, that is, its neighbors that leave the outer cycle when w_i is installed. The forest F is represented by a binary tree T as illustrated in Fig. 4.6(c). The root of T is $w_1 (= v_1)$. The w_1 's right child is w_2 , the w_2 's right child is w_3 , and so on. The set $L(w_i)$ consists of w_i and all nodes in the w_i 's left subtree in T . Thus, the subtree of T rooted at w_i consists of the vertices in $\cup_{j \geq i} L(w_j)$. In the left subtree of T rooted at w_i , the left child of w_i is the w_i 's leftmost child in tree $L(w_i)$ (if any), the left child's right child in T is its next sibling to the right in tree $L(w_i)$ (if any), the left child's right child's right child in T is its next next sibling to the right in tree $L(w_i)$ (if any), and so on. For G_3 in Fig. 4.4(a), T is illustrated in Fig. 4.4(b).

Since v_k is embedded at a point $\mu(w_p, w_q)$, by Eq. (4.1) we have

$$x(v_k) = \frac{1}{2} \{x(w_q) + x(w_p) + y(w_q) - y(w_p)\}, \quad (4.2)$$

$$y(v_k) = \frac{1}{2} \{ [x(w_q) - x(w_p)] + y(w_q) + y(w_p) \} \quad (4.3)$$

and hence

$$x(v_k) - x(w_p) = \frac{1}{2} \{ [x(w_q) - x(w_p)] + y(w_q) - y(w_p) \}. \quad (4.4)$$

The crucial observation is that, when we embed v_k , it is not necessary to know the exact position of w_p and w_q . If we know only their y -coordinates and their relative x -coordinates, that is, $x(w_q) - x(w_p)$, then by Eq. (4.3) we can compute $y(v_k)$ and by Eq. (4.4) we can compute the x -coordinate of v_k relative to w_p , that is, $x(v_k) - x(w_p)$.

For each vertex $v \neq v_1$, the x -offset of v is defined as $\Delta x(v) = x(v) - x(w)$, where w is the parent of v in T . More generally, if w is an ancestor of v , then the x -offset between w and v is $\Delta x(w, v) = x(v) - x(w)$.

With each vertex v we store the following information:

- $left(v)$ = the left child of v in T ;
- $right(v)$ = the right child of v in T ;
- $\Delta x(v)$ = the x -offset of v from its parent in T ; and
- $y(v)$ = the y -coordinate of v .

The algorithm consists of two phases. In the first phase, we add new vertices one by one, and each time we add a vertex we compute its x -offset and y -coordinate, and update the x -offsets of one or two other vertices. In the second phase, we traverse the tree T and compute the final x -coordinates by accumulating offsets.

The first phase is implemented as follows. First we initialize the values stored at v_1, v_2 , and v_3 as follows (see Fig. 4.4):

- $\Delta x(v_1) = 0$; $y(v_1) = 0$; $right(v_1) = v_3$; $left(v_1) = nil$;
- $\Delta x(v_3) = 1$; $y(v_3) = 1$; $right(v_3) = v_2$; $left(v_3) = nil$; and
- $\Delta x(v_2) = 1$; $y(v_2) = 0$; $right(v_2) = nil$; $left(v_2) = nil$.

We then embed the other vertices, one by one, as follows.

- 1 **for** $k = 4$ to n **do**
- 2 **begin**
- 2 Let w_1, w_2, \dots, w_t be the outer cycle $C_o(G_{k-1})$ of G_{k-1} ;
{See Fig. 4.8(a).}
- 3 Let w_p, w_{p+1}, \dots, w_q be the neighbors of v_k on $C_o(G_{k-1})$;

```

4      Increase offset of  $w_{p+1}$  and  $w_q$  by one; {cf. Steps 1' and 2'.}
5      Calculate  $\Delta x(w_p, w_q)$  as
           $\Delta x(w_p, w_q) = \Delta x(w_{p+1}) + \Delta x(w_{p+2}) + \dots + \Delta x(w_q)$ ;
6      Calculate  $\Delta x(v_k)$  as
           $\Delta x(v_k) = \frac{1}{2} \{ \Delta x(w_p, w_q) + y(w_q) - y(w_p) \}$ ;
          {cf. Eq. (4.4).}
7      Calculate  $y(v_k)$  as  $y(v_k) = \frac{1}{2} \{ \Delta x(w_p, w_q) + y(w_q) + y(w_p) \}$ ;
          {cf. Eq. (4.3).}
8      Calculate  $\Delta x(w_q)$  as  $\Delta x(w_q) = \Delta x(w_p, w_q) - \Delta x(v_k)$ ;
9      if  $p + 1 \neq q$  then
10         Calculate  $\Delta x(w_{p+1})$  as  $\Delta x(w_{p+1}) = \Delta x(w_{p+1}) - \Delta x(v_k)$ ;
11         Set  $right(w_p) = v_k$  and  $right(v_k) = w_q$ ;
12         if  $p + 1 \neq q$  then
13             Set  $left(v_k) = w_{p+1}$  and  $right(w_{q-1}) = nil$ 
14         else
15             Set  $left(v_k) = nil$ ;
16     end

```

Figure 4.8 illustrates the construction of T for G_k from T for G_{k-1} by the algorithm above.

In the second phase, we compute the x -coordinate $x(v_i)$ for each vertex v_i in G . Let Q be the path from the root v_1 to v_i in tree T . Then $x(v_i) = \sum \{ \Delta(x) \mid \text{vertex } x \text{ is on } Q \}$. One can compute $x(v_i)$ for all vertices v_i by invoking **Accumulate-Offset**($v_1, 0$); procedure **Accumulate-Offset** is as follows.

```

procedure Accumulate-Offset( $v$ :vertex;  $\delta$ :integer);
begin
    if  $v \neq nil$  then begin
        Set  $\Delta x(v) = \Delta x(v) + \delta$ ;
        Accumulate-Offset( $left(v)$ ;  $\Delta x(v)$ );
        Accumulate-Offset( $right(v)$ ;  $\Delta x(v)$ );
    end
end;

```

Clearly $x(v_i) = \Delta x(v_i)$ for each vertex v_i in G .

The first phase takes linear time, since adding a vertex v_k takes at most time $O(d(v_k))$. The second phase, that is, **Accumulate-Offset**, takes time proportional to the number nodes in T . Thus the algorithm takes linear time in total.

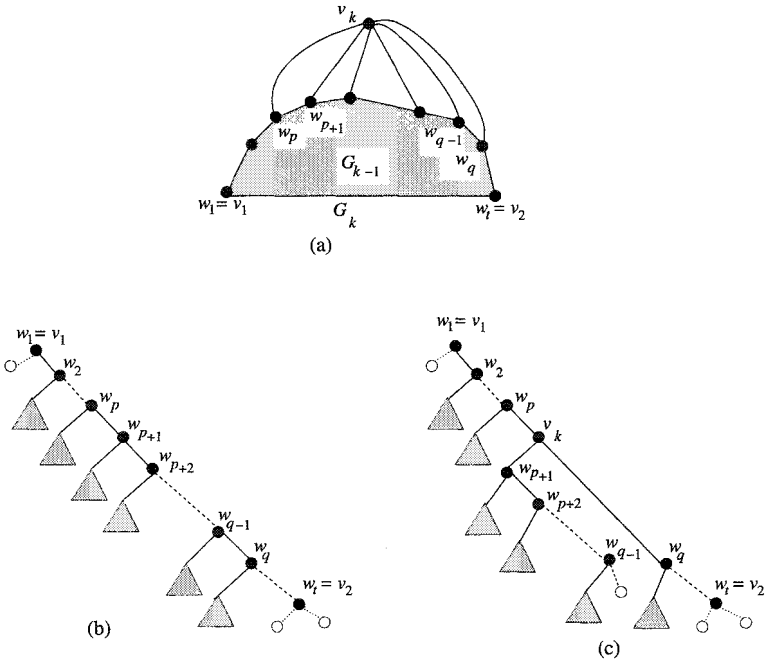


Fig. 4.8 (a) Graph G_k , (b) tree T for G_{k-1} , and (c) tree T for G_k .

4.3 Realizer Method

In this section we describe the realizer method by Schnyder for finding a straight line grid drawing of a plane graph on an $(n - 2) \times (n - 2)$ grid.

Schnyder used a barycentric representation of a plane graph, which is obtained from a “realizer” of a triangulated plane graph G . A realizer is a kind of partition of the inner edges of G into three sets, each inducing a tree in G . Defining a labeling of a triangulated plane graph, he showed that every triangulated plane graph has a realizer. We present a barycentric representation in Section 5.3.1, a labeling in Section 5.3.2, and a realizer in Section 5.3.3, and a drawing algorithm in Section 5.3.4.

4.3.1 Barycentric Representation

A *barycentric representation* of a graph G is an injective function $f : v \in V(G) \rightarrow (v_1, v_2, v_3) \in \mathbf{R}^3$ satisfying the following two conditions:

- (1) $v_1 + v_2 + v_3 = 1$ for all vertices v ; and

(2) for each edge (x, y) and each vertex $z \notin \{x, y\}$, there is some index $k \in \{1, 2, 3\}$ such that $x_k < z_k$ and $y_k < z_k$.

One can view v_1, v_2 and v_3 as barycentric coordinates of vertex v . Condition (1) means that each vertex v of G is mapped to a point on the plane spanned by three points $(1,0,0)$, $(0,1,0)$ and $(0,0,1)$ as illustrated in Fig. 4.9(a). Condition (2) means that, for each edge (x, y) , G has no vertex $z \notin \{x, y\}$ such that $\max\{x_k, y_k\} \geq z_k$ for each index $k \in \{1, 2, 3\}$, and hence there is no vertex z in the triangle on the plane shaded in Fig. 4.9(b).

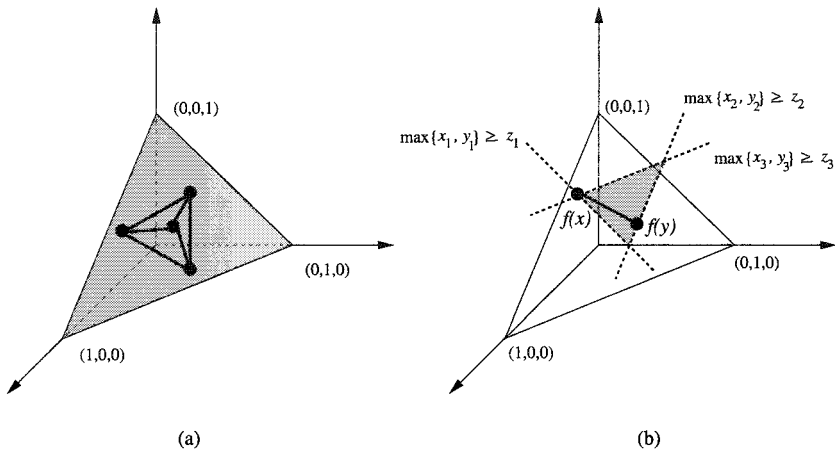


Fig. 4.9 Illustration for Conditions (1) and (2).

The following lemma holds for a barycentric representation of a graph.

Lemma 4.3.1 *A barycentric representation f of a graph G is a planar straight line drawing of G in the plane spanned by three points $(1,0,0)$, $(0,1,0)$ and $(0,0,1)$.*

Proof. Let (x, y) be an edge in G , let z be a vertex such that $z \notin \{x, y\}$. Then by Condition (2), $x_k < z_k$ and $y_k < z_k$ for some index $k \in \{1, 2, 3\}$. Therefore point $f(z)$ does not lie on the straight line segment $f(x)f(y)$ between points $f(x)$ and $f(y)$; otherwise, $f(z) = cf(x) + (1 - c)f(y)$ for some number c , $0 \leq c \leq 1$, but then

$$z_k = cx_k + (1 - c)y_k < cz_k + (1 - c)z_k = z_k,$$

a contradiction.

Let (x, y) and (u, v) be any two edges in G such that the four ends x, y, u and v are pairwise distinct. By Condition (2) there exist indices $i, j, k, l \in \{1, 2, 3\}$ such that

$$u_i, v_i < x_i, \quad (4.5)$$

$$u_j, v_j < y_j, \quad (4.6)$$

$$x_k, y_k < u_k, \quad (4.7)$$

and

$$x_l, y_l < v_l. \quad (4.8)$$

Then $i \neq k, l$ by Eqs. (4.5), (4.7) and (4.8), and $j \neq k, l$ by Eqs. (4.6), (4.7) and (4.8). Thus $i = j$ or $k = l$; otherwise, the four indices i, j, k and l would be distinct with each other although $i, j, k, l \in \{1, 2, 3\}$. One may hence assume that $i = j = 1$. Then by Eqs. (4.5) and (4.6) the two straight line segments $f(x)f(y)$ and $f(u)f(v)$ are separated by a straight line parallel to the line segment $(0,1,0)(0,0,1)$. Hence the two line segments $f(x)f(y)$ and $f(u)f(v)$ do not intersect. \square

Lemma 4.3.1 implies that only a planar graph can have a barycentric representation.

Using a barycentric representation, one can obtain a straight line grid drawing of a plane graph on a $(2n - 5) \times (2n - 5)$ grid [Sch90]. However, using a *weak barycentric representation* defined below, one can obtain a straight line grid drawing on an $(n - 2) \times (n - 2)$ grid. For ordered pairs (a, b) and (c, d) of real numbers, we write $(a, b) <_{lex} (c, d)$ if either $a < c$ or $a = c$ and $b < d$. A *weak barycentric representation* of a graph G is an injective function $v \in V(G) \rightarrow (v_1, v_2, v_3) \in \mathbf{R}^3$ satisfying the following two conditions:

- (1) $v_1 + v_2 + v_3 = 1$ for all vertices v ; and
- (2) for each edge (x, y) and each vertex $z \notin \{x, y\}$, there is some index $k \in \{1, 2, 3\}$ such that $(x_k, x_{k+1}) <_{lex} (z_k, z_{k+1})$ and $(y_k, y_{k+1}) <_{lex} (z_k, x_{k+1})$, where indices are computed as *mod 3*.

Similarly to Lemma 4.3.1 one can prove the following lemma for a weak barycentric representation.

Lemma 4.3.2 *A weak barycentric representation f of a graph G is a planar straight line drawing of G in the plane spanned by three points $(1,0,0)$, $(0,1,0)$ and $(0,0,1)$.*

Proof. Let (x, y) be an edge in G , let z be a vertex in G such that $z \notin \{x, y\}$. Then by Condition (2) above, $(x_k, x_{k+1}) <_{lex} (z_k, z_{k+1})$ and $(y_k, y_{k+1}) <_{lex} (z_k, z_{k+1})$ for some index $k \in \{1, 2, 3\}$. Therefore the point $f(z)$ does not lie on the straight line segment $f(x)f(y)$; otherwise, $f(z) = cf(x) + (1 - c)f(y)$ for some number c , $0 \leq c \leq 1$, but then either

$$z_k = cx_k + (1 - c)y_k < cz_k + (1 - c)z_k = z_k$$

or

$$z_{k+1} = cx_{k+1} + (1 - c)y_{k+1} < cz_{k+1} + (1 - c)z_{k+1} = z_{k+1},$$

a contradiction.

Let (x, y) and (u, v) be any two disjoint edges in G . By Condition (2) there exist indices $i, j, k, l \in \{1, 2, 3\}$ such that

$$(u_i, u_{i+1}), (v_i, v_{i+1}) <_{lex} (x_i, x_{i+1}), \tag{4.9}$$

$$(u_j, u_{j+1}), (v_j, v_{j+1}) <_{lex} (y_j, y_{j+1}), \tag{4.10}$$

$$(x_k, x_{k+1}), (y_k, y_{k+1}) <_{lex} (u_k, u_{k+1}), \tag{4.11}$$

and

$$(x_l, x_{l+1}), (y_l, y_{l+1}) <_{lex} (v_l, v_{l+1}). \tag{4.12}$$

Then $i \neq k, l$ and $j \neq k, l$ and hence either $i = j$ or $k = l$. One may thus assume that $i = j = 1$. Then by Eqs. (4.9) and (4.10) one can observe that the two straight line segments $f(x)f(y)$ and $f(u)f(v)$ are separated by a straight line parallel to $(0,1,0)(0,0,1)$, $(1,0,0)(0,0,1)$, $f(x)f(y)$ or $f(u)f(v)$, as illustrated in Fig. 4.10 where the half plane which can contain points $f(u)$ and $f(v)$ is shaded. Hence the two line segments $f(x)f(y)$ and $f(u)f(v)$ do not intersect. \square

The following Lemma 4.3.3 is an immediate consequence of Lemma 4.3.2.

Lemma 4.3.3 *Let $v \in V(G) \rightarrow (v_1, v_2, v_3) \in \mathbf{R}^3$ be a weak barycentric representation of a graph G , and let α, β and γ be any non-collinear points.*

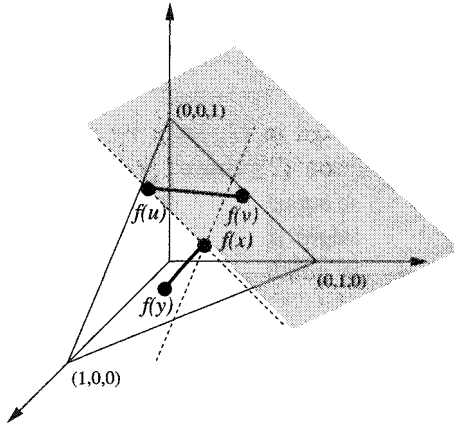


Fig. 4.10 Lines $f(x)f(y)$ and $f(u)f(v)$ are separated by a line.

Then the mapping $f : v \in V(G) \rightarrow v_1\alpha + v_2\beta + v_3\gamma \in \mathbf{R}^3$ is a planar straight line drawing of G in the plane spanned by α, β and γ .

4.3.2 Schnyder Labeling

The *angles* of a triangulated plane graph G are the angles of all inner facial triangles of G . A *Schnyder labeling* of G is a labeling of all angles of G with labels 1, 2, 3 satisfying the following conditions (a) and (b):

- (a) Each inner facial triangle of G has an angle labeled 1, an angle labeled 2 and an angle labeled 3. The corresponding three vertices of the triangle appear in counterclockwise order, as illustrated in Fig. 4.11(a).
- (b) The labels of the angles at each inner vertex of G form, in counterclockwise order, a nonempty interval of 1's followed by a nonempty interval of 2's followed by a nonempty interval of 3's, as illustrated in Fig. 4.11(b).

A Schnyder labeling of a plane graph G is illustrated in Fig 4.12. We will show later in Lemma 4.3.6 that, in a Schnyder labeling, all angles at one of the three outer vertices have label 1, all angles at another have label 2, all angles at the other have label 3, and these three outer vertices appear around the outer face in counterclockwise order. The following theorem holds for the labeling.

Theorem 4.3.4 *Every triangulated plane graph has a Schnyder labeling.*

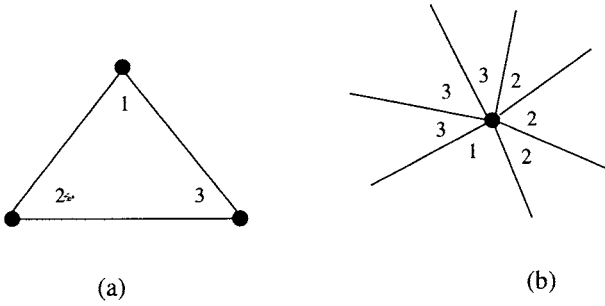


Fig. 4.11 Conditions (a) and (b) of Schnyder labeling.

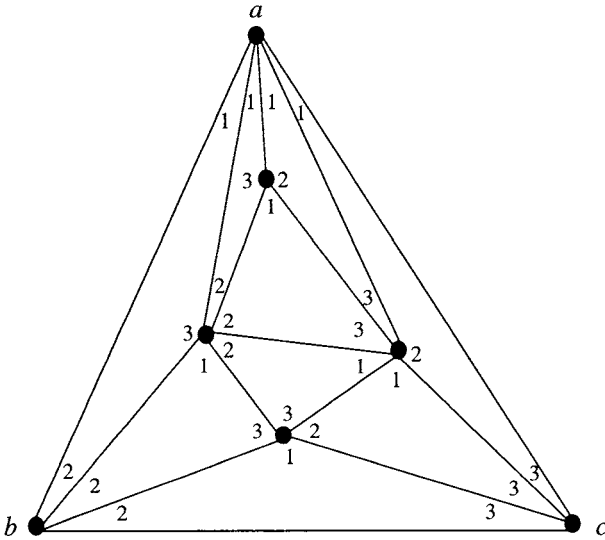


Fig. 4.12 Illustration of a Schnyder labeling.

To present a proof of Theorem 4.3.4 we need some preparation. We first review the method of *edge contraction*. For a vertex x of a graph G , we denote by $N(x)$ the set of neighbors of x in G . For an edge (x, y) of G , we denote by $G/(x, y)$ a (simple) graph obtained from G by contracting edge (x, y) , that is, by removing the vertex y and all the edges incident to y and by inserting an edge (x, z) for each vertex $z \in N(y) - N(x)$, as illustrated in Fig. 4.13. We say that an edge (x, y) is *contractible* if x and y have exactly two common neighbors, as illustrated in Fig. 4.13(a). Let G be a

triangulated plane graph of at least four vertices, then $G/(x, y)$ remains to be a triangulated plane graph if and only if (x, y) is a contractible edge of G . The edge (x, y) in Fig. 4.13(a) is contractible since $N(x) \cap N(y) = \{u, v\}$. On the other hand, the edge (x, y) in Fig. 4.14(a) is not contractible since $N(x) \cap N(y) = \{u, v, z\}$, and hence a quadrilateral face would appear in $G/(x, y)$ around the third common neighbor z as illustrated in Fig. 4.14(b). A *separating triangle* of a triangulated plane graph G is a triangle in G whose interior and exterior contain at least one vertex. The triangle x, y, z is a separating triangle in Fig. 4.14(a). Clearly, an edge (x, y) of a triangulated plane graph is contractible if and only if there is no separating triangle containing x and y . We now have the following lemma [Kam76].

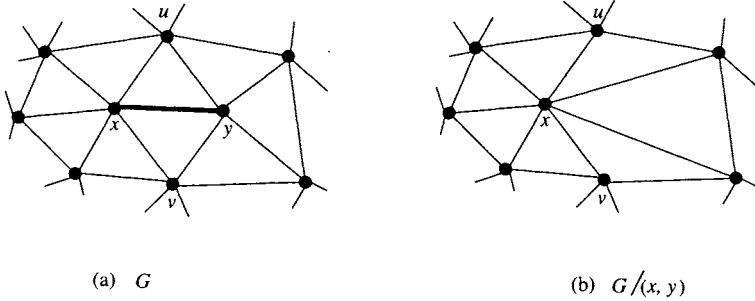


Fig. 4.13 Contraction of a contractible edge (x, y) .

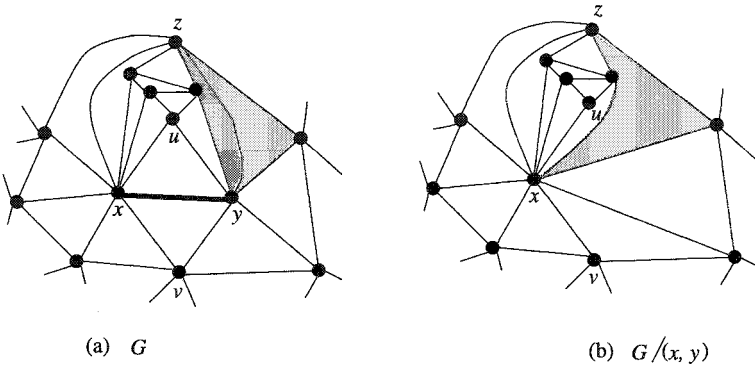


Fig. 4.14 Contraction of a non-contractible edge (x, y) .

Lemma 4.3.5 *Let G be a triangulated plane graph of $n \geq 4$ vertices, and let $a, b,$ and c be the outer vertices of G . Then the outer vertex a has a neighbor $x \neq b, c$ such that edge (a, x) is contractible.*

Proof. The proof is by induction on the number of vertices in G . If there are exactly four vertices in G , then G must be K_4 and we can choose the inner vertex as x . Clearly edge (a, x) is contractible, since a and x have exactly two common neighbors b and c . We may thus assume that G has five or more vertices. Consider first the case where G has no separating triangle containing a . Then edge (a, x) is contractible for any neighbor $x \neq b, c$ of a . Consider next the other case where G has a separating triangle T containing a . Then, by applying induction to the triangulated plane subgraph inside T , we get a vertex x adjacent to a inside T such that edge (a, x) is contractible in the subgraph and hence in G . \square

We are now ready to give a proof of Theorem 4.3.4.

Proof of Theorem 4.3.4 Let G be a triangulated plane graph, and let a be an outer vertex of G . We prove by induction on the number n of vertices of G that G has a Schnyder labeling in which all angles at a have label 1. The case $n = 3$ is trivial, and hence let $n \geq 4$ and assume that our claim is true for all triangulated plane graphs having less than n vertices. Lemma 4.3.5 implies that there exists an inner vertex x adjacent to a such that edge (a, x) is contractible and hence the graph $G/(a, x)$ obtained from G by contracting edge (a, x) is a triangulated plane graph. By induction hypothesis, $G/(a, x)$ has a Schnyder labeling in which all angles at a have label 1, as illustrated in Fig. 4.15(b). This labeling can be extended to a Schnyder labeling of G in which all angles at a have label 1, as illustrated in Fig. 4.15(a). \square

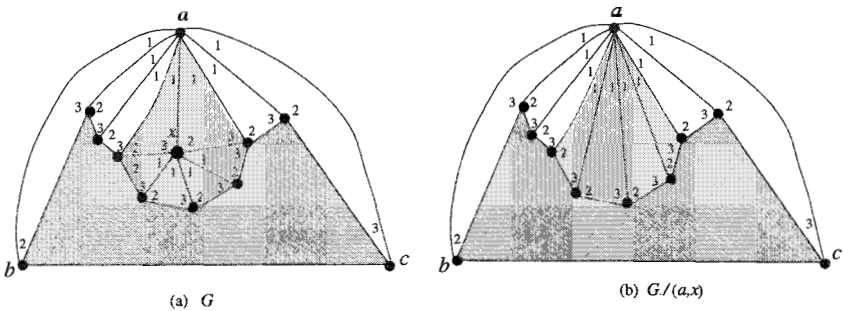


Fig. 4.15 Extension of Schnyder labeling.

Based on the proof of Theorem 4.3.4, one can construct a Schnyder labeling in time $O(n)$ as follows. First consider an initial triangle of three outer vertices a, b and c in counterclockwise order, where the angles at a, b and c are labeled by 1, 2 and 3, respectively. Then the vertices of G are ordered in an *expansion sequence* x_1, x_2, \dots, x_n where $x_1 = b, x_2 = c$ and $x_n = a$, and x_3, x_4, \dots, x_{n-1} describes an order in which the inner vertices are successively inserted inside, that is, edges $(a, x_{n-1}), (a, x_{n-2}), \dots, (a, x_3)$ are successively contracted when one obtains the initial triangle from G . (Indeed a canonical ordering in Section 4.2.1 can serve as an expansion sequence.) Then the desired labeling is obtained by successive expansions leading from the triangle a, b, c to the whole graph G , as described in the proof of Theorem 4.3.4.

4.3.3 Realizer

Consider a Schnyder labeling of a triangulated plane graph G . Each inner edge of G belongs to two facial triangles. The conditions (a) and (b) of a Schnyder labeling imply that each inner edge has two distinct labels, say j and k , at one end, and have the third label i repeated twice at the other end. We call this distinguished label i the *label of the edge*, and orient this edge from the end with distinct labels to the end with the identical labels. (See Figs. 4.12, 4.16 and 4.17.) We thus give orientation to all inner edges according to a Schnyder labeling. Using the orientation of edges above, we can prove the following lemma.

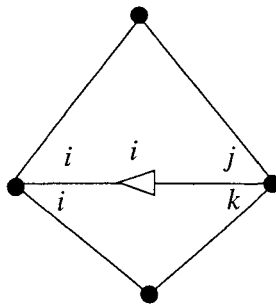


Fig. 4.16 Orientation of an edge.

Lemma 4.3.6 *In a Schnyder labeling of a triangulated plane graph G , all angles at one of the three outer vertices have label 1, all angles at another*

have label 2, all angles at the other have label 3, and these three outer vertices appear around the outer face in counterclockwise order.

Proof. Since G is triangulated, by Corollary 2.2.4 G has $3n - 6$ edges and hence has $3n - 9$ inner edges. By Condition (b) of a Schnyder labeling, each of the $n - 3$ inner vertices has exactly three outgoing edges as illustrated in Fig. 4.18, and hence the number of edges outgoing from all inner vertices of G is $3(n - 3)$. Therefore, no inner edge is outgoing from an outer vertex of G , and hence all inner edges incident on an outer vertex v of G are incoming to v , as illustrated in Fig. 4.17. Thus, according to the orientation of the edges, all angles at v have the same label in the Schnyder labeling. Therefore, by Condition (a) of a Schnyder labeling, all angles at one of the three outer vertices have label 1, all angles at another have label 2, all angles at the other have label 3, and the three vertices with labels 1, 2, 3 appear around the outer face in counterclockwise order. \square

We are now ready to define a realizer. A *realizer* of a triangulated plane graph G is a partition of the inner edges of G into three sets of oriented edges of trees T_1, T_2 and T_3 such that for each inner vertex v

- (a) v has an outgoing edge in each of T_1, T_2 and T_3 ; and
- (b) the counterclockwise order of the edges incident on v is as follows: leaving in T_1 , entering in T_3 , leaving in T_2 , entering in T_1 , leaving in T_3 , entering in T_2 , as illustrated in Fig. 4.18.

Note that v may have indegree zero in T_1, T_2 or T_3 .

For the orientation of edges induced by a Schnyder labeling of G , let T_i , $1 \leq i \leq 3$, consist of all oriented inner edges having label i . Then Condition (b) of a Schnyder labeling implies that T_1, T_2 and T_3 satisfy Conditions (a) and (b) of a realizer. Therefore, by Lemma 4.3.6 each T_i , $1 \leq i \leq 3$, is a tree containing all inner vertices and exactly one outer vertex, and all edges of T_i are oriented toward this outer vertex. The outer vertices belonging to T_1, T_2 and T_3 are distinct and appear in counterclockwise order. Thus T_1, T_2 and T_3 is a realizer of G . Figure 4.17 illustrates a realizer of a triangulated plane graph corresponding to the Schnyder labeling in Fig. 4.12. We call the outer vertex belonging to T_i the *root* r_i of T_i . Note that this is the outer vertex all of whose angles have the label i . We call this outer vertex the *root* of T_i even when G has only three vertices, in which $E(T_1) = E(T_2) = E(T_3) = \emptyset$. Thus a Schnyder labeling induces a realizer. In fact a Schnyder labeling and a realizer are equivalent notions. We thus have the following theorem from Theorem 4.3.4.

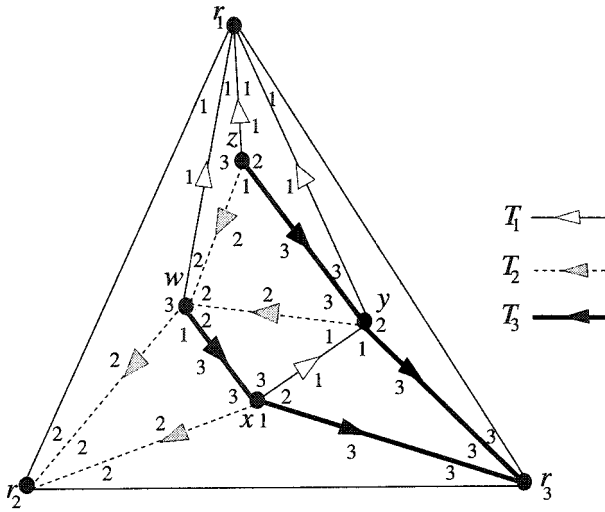


Fig. 4.17 Edge-orientation and realizer.

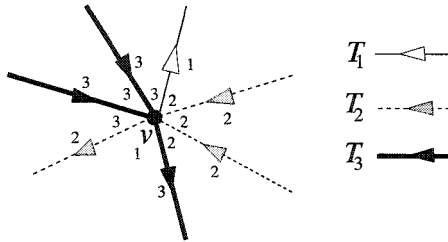


Fig. 4.18 Edge orientations at a vertex v .

Theorem 4.3.7 *Every triangulated plane graph has a realizer.*

Let G be a labeled triangulated plane graph with realizer T_1, T_2 and T_3 . For an inner vertex v of G , we define the i -path $P_i(v)$ starting at v to be the path in T_i from v to the root r_i of T_i . For $i \neq j$, $P_i(v)$ and $P_j(v)$ share only the vertex v (Exercise 5). Therefore, $P_1(v)$, $P_2(v)$ and $P_3(v)$ divide G into three regions $R_1(v)$, $R_2(v)$ and $R_3(v)$, where $R_i(v)$ denotes the closed region opposite to the root r_i of T_i , as illustrated in Figs. 4.19 and 4.20. We denote also by $R_i(v)$ the set of all vertices in region $R_i(v)$. $R_i(v)$ includes all vertices on paths $P_{i+1}(v)$ and $P_{i-1}(v)$, where indices are computed as *modulo 3*. We now have the following lemma.

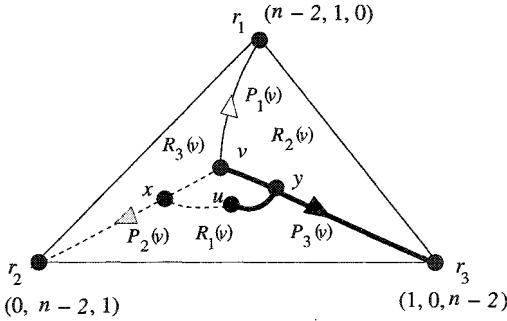


Fig. 4.19 Regions $R_1(v)$, $R_2(v)$ and $R_3(v)$.

Lemma 4.3.8 *Let u and v be any two distinct inner vertices of a labeled triangulated plane graph G , and let $1 \leq i \leq 3$. If $u \in R_i(v)$, then $R_i(u) \subset R_i(v)$ where the inclusion is proper.*

Proof. It suffices to prove the lemma for $i = 1$, since the proof for the other case is similar. We thus assume that $u \in R_1(v)$, as illustrated in Fig. 4.19. We consider only the case where u does not lie on the boundary of $R_1(v)$; the other case is similar. Let x be the first vertex of $P_2(u)$ that belongs to the boundary of $R_1(v)$. Condition (b) of a realizer implies that $x \notin P_3(v)$. Thus $x \in (P_2(v) - \{v\})$. Similarly the first vertex y of $P_3(u)$ belonging to the boundary of $R_1(v)$ must lie on $P_3(v) - \{v\}$. Hence $R_1(u) \subset R_1(v)$. This inclusion is proper since $v \in (R_1(v) - R_1(u))$. \square

4.3.4 Drawing Algorithm with Realizer

Let G be a labeled triangulated plane graph of n vertices with realizer T_1, T_2 and T_3 . We denote also by $P_i(v)$ the set of all vertices in path $P_i(v)$. For an inner vertex v of G , let $n_i(v)$ be the number of all vertices in region $R_i(v)$ that are not in the path $P_{i-1}(v)$, i.e. $n_i(v) = |R_i(v) - P_{i-1}(v)|$. Then $n_1(v) + n_2(v) + n_3(v) = n - 1$ since v is counted by none of $n_1(v)$, $n_2(v)$ and $n_3(v)$. Furthermore $1 \leq n_1(v), n_2(v), n_3(v) \leq n - 3$ since $n_i(v)$ counts r_{i+1} but does not count any of the three vertices v, r_i and r_{i-1} . For example, in Fig. 4.17, we have $(n_1(x), n_2(x), n_3(x)) = (1, 1, 4)$. This definition is extended to the outer vertices of G by setting $n_i(r_i) = n - 2, n_{i+1}(r_i) = 1$ and $n_{i+2}(r_i) = 0$ for the root r_i of T_i . Then, for each vertex v , $n_1(v) + n_2(v) + n_3(v) = n - 1$ and $0 \leq n_1(v), n_2(v), n_3(v) \leq n - 2$. We now show that the following lemma holds.

Lemma 4.3.9 *Let u and v be distinct vertices of a labeled triangulated plane graph G , let v be an inner vertex, and let $1 \leq i \leq 3$. If $u \in R_i(v)$, then $(n_i(u), n_{i+1}(u)) <_{lex} (n_i(v), n_{i+1}(v))$.*

Proof. We first claim that for any k , $1 \leq k \leq 3$, the following hold: if $u \in (R_k(v) - P_{k-1}(v))$, then $n_k(u) < n_k(v)$. Consider first the case where u is an outer vertex. Then u is the root r_{k+1} of T_{k+1} , and hence $n_k(u) = 0$. Since v is an inner vertex, $1 \leq n_k(v)$ and hence $n_k(u) < n_k(v)$. Consider next the case where u is an inner vertex. In this case by Lemma 4.3.8 $R_k(u) \subset R_k(v)$. Path $P_{k-1}(u)$ is in region $R_k(v)$. Thus we have $(R_k(u) - P_{k-1}(u)) \subset (R_k(v) - P_{k-1}(v))$ where the inclusion is proper since $u \notin (R_k(v) - P_{k-1}(v))$ but $u \in (R_k(u) - P_{k-1}(u))$. Hence $n_k(u) < n_k(v)$.

Suppose now that $u \in R_i(v)$. Lemma 4.3.8 together with $n_i(r_{i+1}) = 0$ and $n_i(r_{i-1}) = 1$ imply that $n_i(u) \leq n_i(v)$. Consider first the case where $u \notin P_{i-1}(v)$. Then by the claim above with $k = i$ we have $n_i(u) < n_i(v)$ and hence $(n_i(u), n_{i+1}(u)) <_{lex} (n_i(v), n_{i+1}(v))$. Consider next the case where $u \in P_{i-1}(v)$. Then $u \in (R_{i+1}(v) - P_i(v))$ as illustrated in Fig. 4.20. Therefore by the claim above with $k = i + 1$ we have $n_{i+1}(u) < n_{i+1}(v)$ and hence $(n_i(u), n_{i+1}(u)) <_{lex} (n_i(v), n_{i+1}(v))$. \square

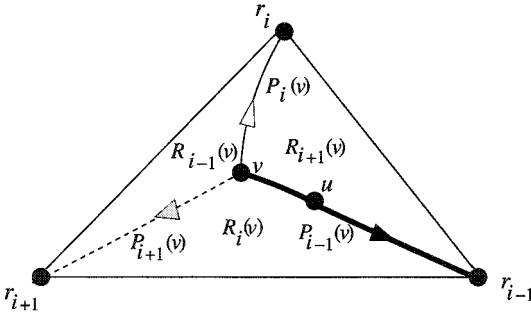


Fig. 4.20 $P_i, P_{i+1}, P_{i-1}, R_i, R_{i+1}$ and R_{i-1} .

We now have the following lemma.

Lemma 4.3.10 *The function $f : v \in V(G) \rightarrow \frac{1}{n-1}(n_1(v), n_2(v), n_3(v)) \in \mathbf{R}^3$ is a weak barycentric representation of G .*

Proof. Since $n_1(v) + n_2(v) + n_3(v) = n - 1$, Condition (1) of a weak barycentric representation is satisfied. There remains to verify Condition (2). Let (x, y) be an edge, and let z be a vertex other than x and y .

Consider first the case where z is an outer vertex. Then z is the root of T_k for some $k \in \{1, 2, 3\}$, and hence $n_k(z) = n - 2$ and $n_k(x), n_k(y) < n_k(z)$. Thus $(x_k, x_{k+1}) <_{lex} (z_k, z_{k+1})$ and $(y_k, y_{k+1}) <_{lex} (z_k, z_{k+1})$. Consider next the case where z is an inner vertex. Then edge (x, y) is contained in $R_k(z)$ for some $k \in \{1, 2, 3\}$, and hence $x, y \in R_k(z)$. Therefore by Lemma 4.3.9 $(n_k(x), n_{k+1}(x)) <_{lex} (n_k(z), n_{k+1}(z))$ and $(n_k(y), n_{k+1}(y)) <_{lex} (n_k(z), n_{k+1}(z))$. \square

Hence, taking $\alpha = (n - 1, 0, 0)$, $\beta = (0, n - 1, 0)$ and $\gamma = (0, 0, 0)$ in Lemma 4.3.3, we get a straight line grid drawing of G on the 2D plane. We now formally describe the algorithm.

Algorithm Realizer-Drawing(G)

begin

 Compute a Schnyder labeling of G ;

 Construct a realizer T_1, T_2, T_3 of G ;

for each vertex $v \in V$ **do**

begin

 Calculate $n_1(v)$ and $n_2(v)$;

 Install v at a grid point $(n_1(v), n_2(v))$ on the 2D plane;

end

end

Note that root r_1 is installed at $(n - 2, 1)$, r_2 at $(0, n - 2)$ and r_3 at $(1, 0)$. Figure 4.21 illustrates a straight line drawing of the graph in Fig. 4.17 on the $(n - 2) \times (n - 2)$ grid obtained by Algorithm **Realizer-Drawing**.

Since $0 \leq n_1(v), n_2(v), n_3(v) \leq n - 2$, the size of the grid is $(n - 2) \times (n - 2)$, and hence we have the following theorem.

Theorem 4.3.11 *The mapping $v \in V(G) \rightarrow (n_1(v), n_2(v)) \in \mathbf{R}^2$ is a straight line drawing of G on an $(n - 2) \times (n - 2)$ grid.*

To make the time complexity of Algorithm **Realizer-Drawing** linear we need to calculate $n_1(v)$ and $n_2(v)$ for all inner vertices v in linear time in total. This can be done as follows.

Let $|R_i(v)|$, $1 \leq i \leq 3$, be the number of vertices in region $R_i(v)$, and let $|P_i(v)|$ be the number of vertices on path $P_i(v)$, then $n_i(v) = |R_i(v)| - |P_{i-1}(v)|$. Let $t_i(v)$ be the number of vertices in the subtree of T_i that is rooted at inner vertex v , and let $t_i(r_{i+1}) = t_i(r_{i-1}) = 1$. Traversing the trees T_1, T_2 , and T_3 by a constant number of times, we can calculate $|P_i(v)|$ and $t_i(v)$ for all vertices v and all indices i . We then calculate $|R_i(v)|$

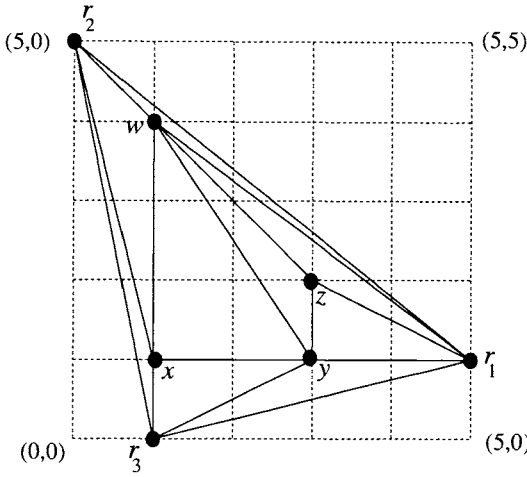


Fig. 4.21 A straight line drawing obtained by Algorithm **Realizer-Drawing**.

for all vertices v and all indices i , as follows. One can observe that, for each inner vertex $u \in R_i(v)$, the path $P_i(u)$ must intersect $P_{i+1}(v)$ or $P_{i-1}(v)$. Thus u belongs to the subtree of T_i rooted at some vertex x on $P_{i+1}(v)$ or $P_{i-1}(v)$. Furthermore, each of these subtrees of T_i is entirely contained in $R_i(v)$. (See Fig. 4.22.) Then the following expression holds:

$$|R_i(v)| = \sum_{x \in P_{i+1}(v)} t_i(x) + \sum_{x \in P_{i-1}(v)} t_i(x) - t_i(v).$$

Thus $|R_i(v)|$ can be computed by traversing T_1, T_2 , and T_3 by a constant number of times. Hence $n_1(v), n_2(v)$, and $n_3(v)$ for all inner vertices v can be computed in linear time in total.

4.4 Compact Grid Drawing

We have seen two algorithms in the previous two sections to find a straight line drawing on grids of sizes $(2n - 4) \times (n - 2)$ and $(n - 2) \times (n - 2)$, respectively. A natural question arises: what is the minimum size of a grid required for a straight line drawing? de Fraysseix *et al.* showed that, for each $n \geq 3$, there exists a plane graph of n vertices, for example nested triangles, which needs a grid of size at least $\lfloor 2(n - 1)/3 \rfloor \times \lfloor 2(n - 1)/3 \rfloor$ for any grid drawing [CN98, FPP90]. It has been conjectured that every plane

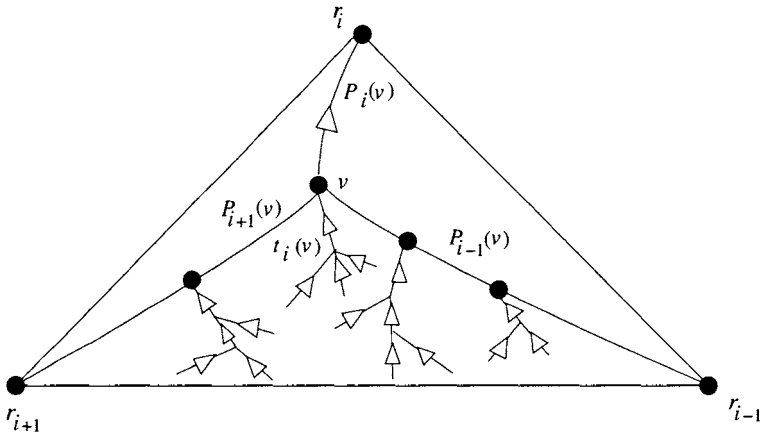


Fig. 4.22 Subtrees of T_i .

graph of n vertices has a grid drawing on a $\lceil 2n/3 \rceil \times \lceil 2n/3 \rceil$ grid, but it is still an open problem. On the other hand, a restricted class of graphs has a more compact grid drawing. For example, if G is a 4-connected plane graph, then G has a more compact grid drawing [He97, MNN01]. In this section we consider compact straight line grid drawings of 4-connected plane graphs.

Let G be a 4-connected plane graph having at least four outer vertices. Miura *et al.* [MNN01] gave a very simple algorithm which finds a grid drawing of G on a $W \times H$ grid such that $W = \lceil n/2 \rceil - 1$ and $H = \lceil n/2 \rceil$ in linear time. Their result is indeed best possible, because there exist an infinite number of 4-connected plane graphs, for example the nested quadrangles depicted in Fig. 4.23, which need grids of size at least $W = \lceil n/2 \rceil - 1$ and $H = \lceil n/2 \rceil$ for any grid drawing.

In this section we present the algorithm of Miura *et al.* [MNN01]. The outline of the algorithm is as follows. One may assume without loss of generality that a given plane graph G is internally triangulated as illustrated in Fig. 4.24(a). First, the algorithm finds a “4-canonical ordering” of G [KH97]. Using the ordering, the algorithm then divides G into two subgraphs G' and G'' , each of which has about $n/2$ vertices as illustrated in Fig. 4.24(b) where G' and G'' are shaded. Next, the algorithm draws the plane subgraph G' in an isosceles right triangle whose base has length $W' = n/2 - 1$ and whose height is $H' = W'/2$, as illustrated in Fig. 4.24(c). Similarly, the algorithm draws G'' in the same triangle with its upside

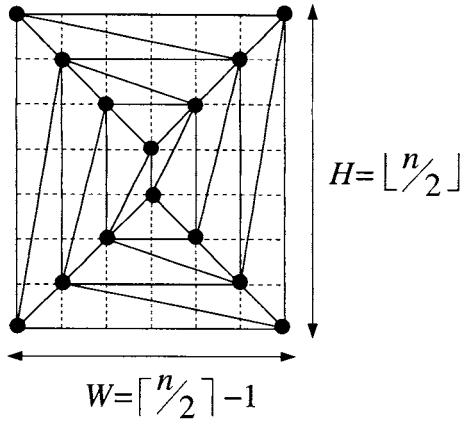


Fig. 4.23 Nested quadrangles attaining the bounds.

down. In Fig. 4.24(c) the two triangles are drawn by thick dotted lines. The algorithm places the two triangles so that their vertices opposite to their bases are separated by distance 1. Finally, the algorithm combines the drawings of G' and G'' to obtain a grid drawing of G , as illustrated in Fig. 4.24(d). The drawing of G has sizes $W = W' = n/2 - 1$ and $H = 2H' + 1 = W' + 1 = n/2$.

4.4.1 Four-Canonical Ordering

In this section we give a definition of a 4-canonical ordering of a plane graph G [KH97]. The 4-canonical ordering is a generalization of the “canonical ordering” [FPP90] described in Section 4.2.1. Let $\pi = (v_1, v_2, \dots, v_n)$ be an ordering of set V . Figure 4.25(a) illustrates an ordering of the graph G in Fig. 4.24(a). Let G_k , $1 \leq k \leq n$, be the plane subgraph of G induced by the vertices in $\{v_1, v_2, \dots, v_k\}$, and let $\overline{G_{k-1}}$ be the plane subgraph of G induced by the vertices in $\{v_k, v_{k+1}, \dots, v_n\}$. Thus $\overline{G_{k-1}} = G - \{v_1, v_2, \dots, v_{k-1}\}$ and $G = G_n = \overline{G_0}$. In Fig. 4.25(b), G_k is darkly shaded, while $\overline{G_{k-1}}$ is lightly shaded. We say that π is a *4-canonical ordering* of G if the following three conditions are satisfied:

- (co1) (v_1, v_2) and (v_{n-1}, v_n) are edges on $C_o(G)$;
- (co2) for each k , $3 \leq k \leq n - 2$, v_k is on both $C_o(G_k)$ and $C_o(\overline{G_{k-1}})$; and
- (co3) for each k , $3 \leq k \leq n - 2$, both G_k and $\overline{G_{k-1}}$ are 2-connected.

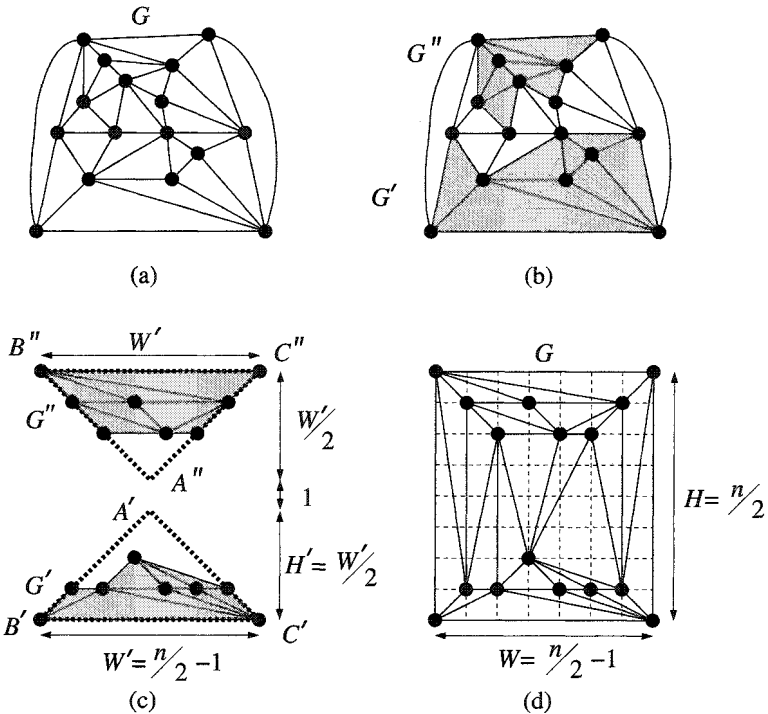


Fig. 4.24 Drawing process of algorithm.

Although the definition of a 4-canonical ordering above is slightly different from that in [KH97], they are effectively equivalent with each other. We have the following lemma.

Lemma 4.4.1 *Let G be a 4-connected internally triangulated plane graph having at least four outer vertices. Then G has a 4-canonical ordering π , and π can be found in linear time.*

Proof. We first show that G has a 4-canonical ordering. Since there are four or more outer vertices, we can arbitrarily choose four distinct outer vertices v_1, v_2, v_n and v_{n-1} so that (v_1, v_2) and (v_n, v_{n-1}) are outer edges of G and v_1, v_2, v_n and v_{n-1} appear on $C_o(G)$ counterclockwise in this order. Thus (co1) holds. We choose as v_3 the third vertex of the inner triangular face containing v_1 and v_2 . Then G_3 is a triangle and hence is 2-connected. Since G is 4-connected, one can know that $\overline{G_2}$ is 2-connected, $v_3 \neq v_n, v_{n-1}$, and v_3 is on both $C_o(G_3)$ and $C_o(\overline{G_2})$. Thus (co2) and (co3) hold for $k = 3$. Furthermore, v_3 is not an end of a chord of the cycle $C_o(\overline{G_2})$; otherwise,

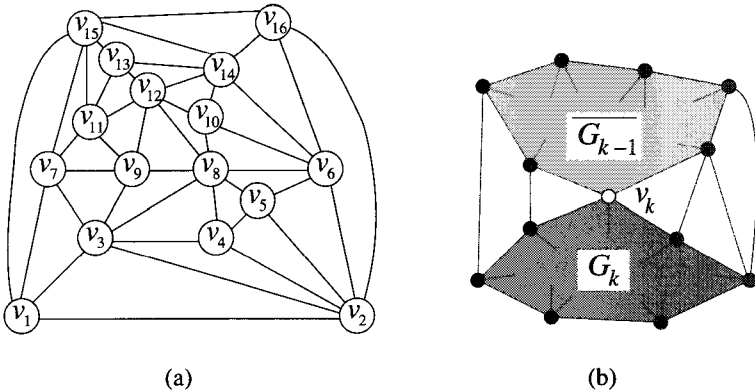


Fig. 4.25 (a) A 4-canonical ordering of a 4-connected plane graph of $n = 16$ vertices, and (b) an illustration for the condition (co2).

v_3 , the other end of the chord and either v_1 or v_2 would form a separator of G , contrary to the 4-connectivity of G .

Assume for inductive hypothesis that $3 \leq k \leq n - 3$, the vertices v_1, v_2, \dots, v_k have been appropriately chosen, (co2) and (co3) hold for k , and v_k is not an end of a chord of the cycle $C_o(\overline{G_{k-1}})$.

Suppose that there is a vertex $w \neq v_n, v_{n-1}$ on the cycle $C_o(\overline{G_k})$ which has two or more neighbors in G_k and is not an end of a chord of $C_o(\overline{G_k})$, as illustrated in Fig. 4.26. Then choose w as v_{k+1} . Clearly $w = v_{k+1}$ is on both $C_o(G_{k+1})$ and $C_o(\overline{G_k})$, and v_{k+1} is not an end of a chord of $C_o(\overline{G_k})$. G_{k+1} is biconnected since G_k is biconnected and v_{k+1} has two or more neighbors in G_k . $\overline{G_k}$ is also biconnected because $\overline{G_{k-1}}$ is biconnected, $\overline{G_k} = \overline{G_{k-1}} - v_k$, and v_k is not an end of a chord of $C_o(\overline{G_{k-1}})$. Thus (co2) and (co3) hold for $k + 1$. Hence it suffices to show that there is such a vertex w .

Let $C_o(\overline{G_k}) = w_1, w_2, \dots, w_t$, where $t \geq 3$, $w_1 = v_n$ and $w_t = v_{n-1}$, as illustrated in Fig. 4.27. We first consider the case where the cycle $C_o(\overline{G_k})$ has no chord. Then there is a vertex $w \in \{w_2, w_3, \dots, w_{t-1}\}$ which has two or more neighbors in G_k ; otherwise, all vertices w_1, w_2, \dots, w_t would have a common neighbor y in G_k and hence $\{w_1, w_t, y\}$ would be a separator of G as illustrated in Fig. 4.27(a). We next consider the other case. Then $C_o(\overline{G_k})$ has a "minimal" chord (w_p, w_q) , $p + 2 \leq q$, such that none of the vertices $w_{p+1}, w_{p+2}, \dots, w_{q-1}$ is an end of a chord of $C_o(\overline{G_k})$. There is a vertex $w \in \{w_{p+1}, w_{p+2}, \dots, w_{q-1}\}$ which has two or more neighbors in G_k ; otherwise, all vertices w_p, w_{p+1}, \dots, w_q would have a common neighbor y

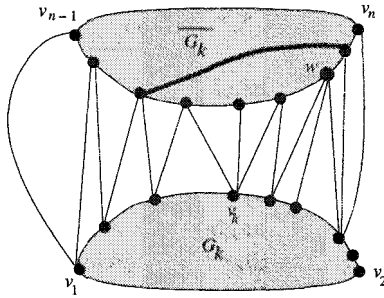


Fig. 4.26 Choosing w as v_{k+1} .

in G_k and hence $\{w_p, w_q, y\}$ would be a separator of G as illustrated in Fig. 4.27(b).

We thus have proved that there exists a 4-canonical ordering.

One can implement a linear algorithm to find a 4-canonical ordering of G based on the proof, using a data structure similar to one for Algorithm **Canonical-Ordering** in Section 4.2.1. \square

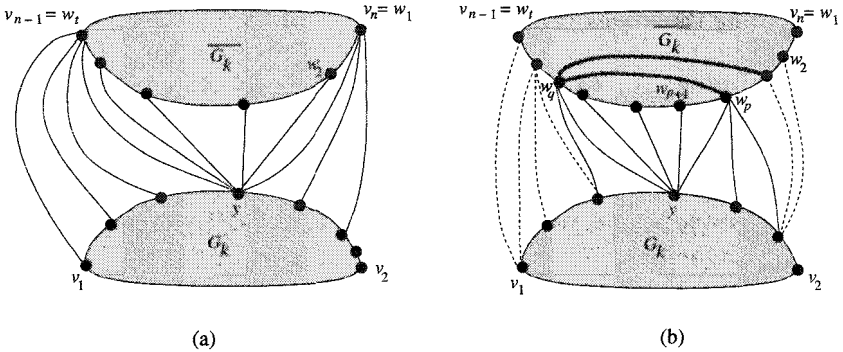


Fig. 4.27 Illustration for the proof of Lemma 4.4.1.

4.4.2 Algorithm Four-Connected-Draw

In this section we formally present the algorithm of Miura *et al.* in [MNN01] as Algorithm **Four-Connected-Draw**.

Procedure Four-Connected-Draw(G)

begin

TEAM LING - LIVE, INFORMATIVE, NON-COST AND GENUINE !

- 1 Find a 4-canonical ordering $\pi = (v_1, v_2, \dots, v_n)$ of a given 4-connected plane graph $G = (V, E)$;
- 2 Divide G into two subgraphs G' and G'' where $n' = \lceil n/2 \rceil$, $G' = G_{n'}$, and $G'' = \overline{G_{n'}}$;
- 3 Draw G' in an isosceles right triangle $\Delta' = A'B'C'$ whose base $B'C'$ has length $W' = \lceil n/2 \rceil - 1$ and whose height is $H' = W'/2$;
- 4 Draw G'' in a congruent triangle $\Delta'' = A''B''C''$ with its upside down;
- 5 Place the two triangles so that their vertices A' and A'' opposite to their bases are separated by distance 1 and have the same x-coordinate;
- 6 Draw every edge of G joining a vertex in G' and a vertex in G'' by a straight line segment;

end.

We say that a curve in the plane is *x-monotone* if the intersection of the curve and any vertical line is a single point when it is nonempty. We then have the following lemma for the drawing of G' , the proof of which will be given later in Section 4.4.3.

Lemma 4.4.2 *One can find in linear time a grid drawing of G' satisfying the following conditions (a), (b) and (c):*

- (a) *the drawing is in an isosceles right triangle $\Delta' = A'B'C'$ whose base $B'C'$ has length $W' = \lceil n/2 \rceil - 1$ and whose height is $H' = W'/2$;*
- (b) *the absolute value of the slope of every edge on $C_o(G')$ is at most 1; and*
- (c) *the drawing of the path going clockwise on $C_o(G')$ from v_1 to v_2 is x-monotone.*

If $\pi = (v_1, v_2, \dots, v_n)$ is a 4-canonical ordering, then the reversed ordering $\pi' = (v_n, v_{n-1}, \dots, v_1)$ is also a 4-canonical ordering. Therefore G'' has a grid drawing in a triangle Δ'' congruent with Δ' . Hence we have the following theorem.

Theorem 4.4.3 *Algorithm Four-Connected-Draw finds in linear time a grid drawing of a given 4-connected plane graph G on a $W \times H$ grid such that $W = \lceil n/2 \rceil - 1$ and $H = W + 1 = \lceil n/2 \rceil$ if G has four or more outer vertices.*

Proof. If step 6 in **Four-Connected-Draw** does not introduce any edge-intersection, then algorithm **Four-Connected-Draw** correctly finds a grid drawing of G and clearly the size of a drawing of G satisfies $W = W'$

and $H = H' + H' + 1$. (See Fig. 4.24.) By Lemma 4.4.2(a) $W' = \lceil n/2 \rceil - 1$ and $H' = W'/2$. Therefore $W = \lceil n/2 \rceil - 1$ and $H = W + 1 = \lceil n/2 \rceil$. Thus we shall show that step 6 does not introduce any edge-intersection.

An oblique side of each isosceles right triangle has slope $+1$ and the other oblique side has slope -1 . The two vertices A' and A'' are separated by distance 1 and have the same x-coordinate: if A' has a coordinate (x, y) , then A'' has a coordinate $(x, y + 1)$. Therefore the absolute value of the slope of any straight line connecting a point in Δ' and a point in Δ'' is greater than the slope $H/W = 1 + 1/W (> 1)$ of a diagonal of the $W \times H$ rectangle. Thus, the absolute value of the slope of any edge connecting a vertex on $C_o(G')$ and a vertex on $C_o(G'')$ is greater than 1. (See Fig. 4.28.) On the other hand, by Lemma 4.4.2(b) the absolute value of the slope of every edge on $C_o(G')$ or $C_o(G'')$ is less than or equal to 1. Furthermore, by Lemma 4.4.2(c) both the drawing of the path from v_1 to v_2 on $C_o(G')$ and the drawing of the path from v_{n-1} to v_n on $C_o(G'')$ are x-monotone. Therefore, the straight line drawing of any edge of G connecting a vertex on $C_o(G')$ and a vertex on $C_o(G'')$ does not intersect the drawings of G' and G'' . Furthermore the drawings of all these edges do not intersect with each other since G is a plane graph and the drawings of the two paths above are x-monotone. Thus step 6 does not introduce any edge-intersection.

By Lemma 4.4.1 one can execute steps 1 and 2 of procedure **Four-Connected-Draw** in linear time. By Lemma 4.4.2 one can execute steps 3 and 4 in linear time. Clearly one can execute steps 5 and 6 in linear time. Thus **Four-Connected-Draw** runs in linear time. \square

4.4.3 Drawing G'

In this section, we show how to find a drawing of G' satisfying the conditions (a), (b) and (c) in Lemma 4.4.2. It suffices to decide only the coordinates of all vertices of G' , because one can immediately find a straight line drawing from the coordinates.

We first define some terms. Let $\pi = (v_1, v_2, \dots, v_n)$ be a 4-canonical ordering of G . For any vertices $v_i, v_j \in V$, we write $v_i < v_j$ if $1 \leq i < j \leq n$, and write $v_i \preceq v_j$ if $1 \leq i \leq j \leq n$. We will later show that the following lemma holds.

Lemma 4.4.4 *If (u, v) is an edge in G' and $u \preceq v$, then the y-coordinates of vertices u and v satisfy $y(u) \leq y(v)$.*

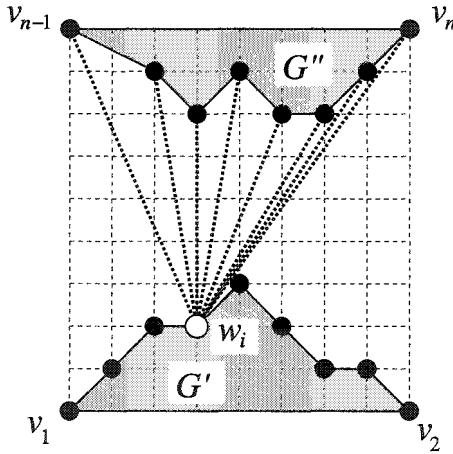


Fig. 4.28 Illustration for the proof of Theorem 4.4.3.

We say that a vertex u in a graph G is a *smaller neighbor* of v if u is a neighbor of v and u is smaller than v , that is $u < v$. Similarly, we say that u is a *larger neighbor* of v if u is a neighbor of v and $u > v$. The smallest one among the neighbors of vertex v is called the *smallest neighbor* of v , and is denoted by $w_s(v)$. We often denote $w_s(v)$ simply by w_s . The definition of a 4-canonical ordering implies that each vertex v_k , $3 \leq k \leq n-2$, has at least two smaller neighbors and at least two larger neighbors. Let $3 \leq k \leq n$, and let $C_o(G_{k-1}) = w_1, w_2, \dots, w_t$, where $w_1 = v_1$ and $w_t = v_2$. Since G is internally triangulated, all the smaller neighbors of v_k consecutively appear on $C_o(G_{k-1})$. Thus one may assume that they are w_p, w_{p+1}, \dots, w_q for some indices p and q , $1 \leq p < q \leq t$, as illustrated in Fig. 4.29. Then the following lemma holds.

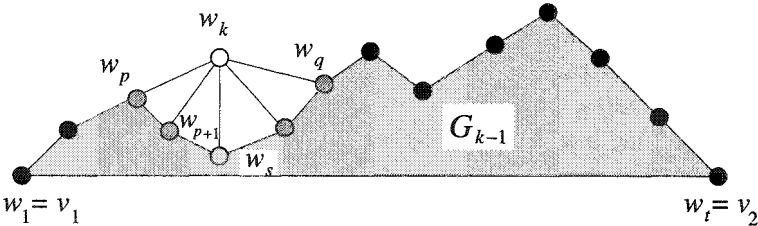


Fig. 4.29 Graph G_k .

Lemma 4.4.5 Let $\pi = (v_1, v_2, \dots, v_n)$ be a 4-canonical ordering of G ,

TEAM LING - LIVE, INFORMATIVE, NON-COST AND GENUINE !

and let w_p, w_{p+1}, \dots, w_q be the smaller neighbors of v_k , $3 \leq k \leq n$. Then the following (a) and (b) hold:

- (a) there is no index r such that $p < r < q$ and $w_{r-1} \prec w_r \succ w_{r+1}$; and
- (b) let $w_s = w_s(v_k)$, then $w_p \succeq w_{p+1} \succeq \dots \succeq w_s \preceq \dots \preceq w_q$ and $y(w_p) \geq y(w_{p+1}) \geq \dots \geq y(w_s) \leq \dots \leq y(w_q)$. (See Fig. 4.29.)

Proof. (a) Assume for a contradiction that there is an index r such that $p < r < q$ and $w_{r-1} \prec w_r \succ w_{r+1}$, as illustrated in Fig. 4.30. Let $w_r = v_i$ for an index i , $1 \leq i \leq k-1$. Since v_k is adjacent to w_{r-1}, w_r and w_{r+1} in G_k , $w_r = v_i$ is neither on $C_o(G_k)$ nor on $C_o(G)$ and hence $3 \leq i \leq n-2$. Therefore by the condition (co3) of the 4-canonical ordering, w_r has at least two larger neighbors. Let v_j be the largest one among all the w_r 's neighbors other than v_k . Then $w_r = v_i \prec v_j \neq v_k$. Clearly vertex v_j is either in the triangular face v_k, w_r, w_{r-1} of graph G_k or in the triangular face v_k, w_{r+1}, w_r . Since $w_r \prec v_j$ and $w_{r-1} \prec w_r \succ w_{r+1}$, we have $v_j \neq w_{r-1}, w_{r+1}$. Therefore v_j must be in the proper inside of one of the two faces above. Thus one may assume that v_j is in the proper inside of the face v_k, w_r, w_{r-1} as illustrated in Fig. 4.30. Since v_j is not on $C_o(G)$, we have $3 \leq j \leq n-2$. Since v_j is not in G_{k-1} , we have $v_{k-1} \prec v_j$ and hence $v_k \prec v_j$. Therefore v_k is contained in G_j , but v_j is not on $C_o(G_j)$, contrary to the condition (co2) of the 4-canonical ordering.

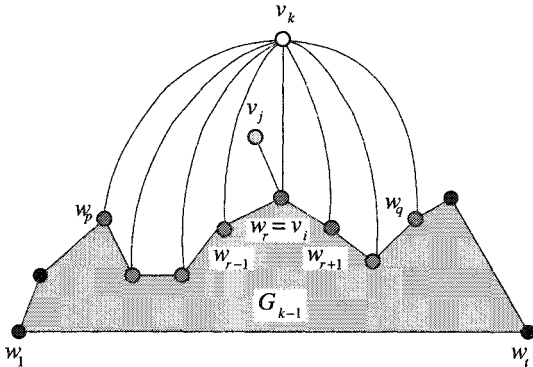


Fig. 4.30 Graph G_k and vertex v_j .

(b) Since $w_s \preceq w_q$, by (a) we have $w_s \preceq w_{s+1} \preceq \dots \preceq w_q$. Therefore by Lemma 4.4.4 we have $y(w_s) \leq y(w_{s+1}) \leq \dots \leq y(w_q)$. Similarly we have $y(w_p) \geq y(w_{p+1}) \geq \dots \geq y(w_s)$. \square

We are now ready to show how to find a drawing of G' . First, we put vertices v_1, v_2, v_3 on grid points $(0, 0), (2, 0)$ and $(1, 1)$ so that G_3 is drawn as an isosceles right triangle, as illustrated in Fig. 4.4(a). Clearly the conditions (b) and (c) in Lemma 4.4.2 hold for G_3 . Next, for each k , $4 \leq k \leq \lceil n/2 \rceil$, we decide the x-coordinate $x(v_k)$ and the y-coordinate $y(v_k)$ of v_k so that the conditions (b) and (c) in Lemma 4.4.2 hold for G_k . One may assume that the conditions hold for G_{k-1} . Let $C_o(G_{k-1}) = w_1, w_2, \dots, w_t$, and let w_p, w_{p+1}, \dots, w_q be the smaller neighbors of v_k . Since the condition (c) of Lemma 4.4.2 holds for G_{k-1} , the drawing of the path w_p, w_{p+1}, \dots, w_q is x-monotone. Furthermore, by Lemma 4.4.5(b), we have $y(w_p) \geq y(w_{p+1}) \geq \dots \geq y(w_s) \leq \dots \leq y(w_q)$, as illustrated in Figs. 4.29, 4.31 and 4.32.

We always shift a drawing of G_{k-1} to the x-direction before adding vertex v_k , as illustrated in Fig. 4.31. We have to determine which vertices of G_{k-1} must be shifted to the x-direction. Thus we will maintain a set $L(v_k)$ for each vertex v_k , $1 \leq k \leq \lceil n/2 \rceil$. This set will contain vertices covered by v_k that need to be shifted whenever v_k is shifted. Initially, we set $L(v_k) = \{v_k\}$ for $k = 1, 2, 3$. For k , $4 \leq k \leq \lceil n/2 \rceil$, we set $L(v_k) = \{v_k\} \cup (\bigcup_{i=p+1}^{q-1} L(w_i))$. Thus all vertices in $L(v_k)$ except v_k are inner vertices of G_k . The *shift operation* on a vertex w_j , denoted by $Shift(w_j)$, is achieved by increasing the x-coordinate of each vertex $u \in \bigcup_{i=j}^t L(w_i)$ by 1.

We then show how to decide $y(v_k)$ and $x(v_k)$. Let y_{max} be the maximum value of y-coordinates of w_p, w_{p+1}, \dots, w_q , then either $y_{max} = y(w_p)$ or $y_{max} = y(w_q)$. There are the following six cases as illustrated in Fig. 4.32:

- (i) $y(w_p) < y(w_q) = y_{max}$;
- (ii) $y_{max} = y(w_p) > y(w_q)$;
- (iii) $y(w_p) = y(w_q) = y_{max}$, $p < s < q$ and $y(w_{p+1}) \neq y_{max}$;
- (iv) $y(w_p) = y(w_q) = y_{max}$, $p < s < q$ and $y(w_{p+1}) = y_{max}$;
- (v) $y(w_p) = y(w_q) = y_{max}$ and $s = p$; and
- (vi) $y(w_p) = y(w_q) = y_{max}$ and $s = q$.

We first consider the three cases (i), (iii) and (v). In these cases $y_{max} = y(w_q)$. We decide $y(v_k)$ and $x(v_k)$ as follows. We first execute $Shift(w_{s+1})$, that is, we increase by 1 the x-coordinates of all vertices $w_{s+1}, w_{s+2}, \dots, w_t$ and all vertices covered by them, as illustrated in Figs. 4.31(a), (c) and (e).

We then decide

$$y(v_k) = \begin{cases} y_{max} & \text{if } y(w_{q-1}) < y_{max}; \\ y_{max} + 1 & \text{if } y(w_{q-1}) = y_{max}, \end{cases}$$

and

$$x(v_k) = x(w_s) + y(v_k) - y(w_s).$$

We denote the slope of a straight line segment uv by $slope(uv)$. Then clearly we have

$$slope(w_s v_k) = \frac{y(v_k) - y(w_s)}{x(v_k) - x(w_s)} = 1.$$

Since $y(v_k) \geq y(w_p) \geq y(w_s)$ and $x(w_p) \leq x(w_s) < x(v_k)$, we have

$$0 \leq slope(w_p v_k) = \frac{y(v_k) - y(w_p)}{x(v_k) - x(w_p)} \leq slope(w_s v_k) = 1$$

as illustrated in Figs. 4.31(a), (c) and (e).

If $y(w_{q-1}) < y_{max}$, then $y(v_k) = y_{max} = y(w_q)$ and hence $slope(v_k w_q) = 0$ as illustrated in Fig. 4.31(a). On the other hand, if $y(w_{q-1}) = y_{max}$, then $y(v_k) = y(w_q) + 1$, $x(v_k) \leq x(w_q) - 1$ and hence we have

$$-1 \leq slope(v_k w_q) = \frac{y(w_q) - y(v_k)}{x(w_q) - x(v_k)} < 0$$

as illustrated in Figs. 4.31(c) and (e).

The absolute slope of each straight line segment on $C_o(G_k)$ except $w_p v_k$ and $v_k w_q$ is equal to its absolute slope on $C_o(G_{k-1})$, and hence is at most 1.

Thus the condition (b) in Lemma 4.4.2 holds for G_k .

One can easily observe that the condition (c) in Lemma 4.4.2 holds for G_k .

We next consider the remaining three cases (ii), (iv) and (vi). In these cases we decide $y(v_k)$ and $x(v_k)$ in a mirror image way of the cases (i), (iii) and (v) above. That is, we execute *Shift*(w_s), and decide

$$y(v_k) = \begin{cases} y_{max} & \text{if } y(w_{p+1}) < y_{max}; \\ y_{max} + 1 & \text{if } y(w_{p+1}) = y_{max}, \end{cases}$$

and

$$x(v_k) = x(w_s) - (y(v_k) - y(w_s)).$$

(See Figs. 4.31(b), (d) and (f).) Then, similarly as in Cases (i), (iii) and (v) above, the conditions (b) and (c) hold for G_k .

Since we decide the y-coordinate as above, Lemma 4.4.4 clearly holds.

We call the algorithm described above for finding a drawing of G' Algorithm **Draw-Triangle**.

We are now ready to prove Lemma 4.4.2.

Proof of Lemma 4.4.2 As shown above, the conditions (b) and (c) hold for a drawing obtained by Algorithm **Draw-Triangle**. Therefore the absolute value of the slope of every edge on $C_o(G')$ is at most 1, and the drawing of the path going clockwise on $C_o(G')$ from v_1 to v_2 is x-monotone.

The drawing of G_3 has width 2. We execute the shift operation once when we add a vertex v_k , $4 \leq k \leq n' = \lceil n/2 \rceil$, to the drawing of G_{k-1} . Therefore the width W' of the drawing of G' is $W' = 2 + (n' - 3) = \lceil n/2 \rceil - 1$. Since the conditions (b) and (c) hold, the height is at most $W'/2$. Therefore G' is drawn in an isosceles right triangle $\Delta' = A'B'C'$ whose base $B'C'$ has length $W' = \lceil n/2 \rceil - 1$ and whose height is $H' = W'/2$. Thus the condition (a) holds.

We then show that the drawing of G' obtained by Algorithm **Draw-Triangle** is a grid drawing. The algorithm puts each vertex v_k , $4 \leq k \leq \lceil n/2 \rceil$, on a grid point. Clearly each edge (v_k, w_j) , $p \leq j \leq q$, does not intersect any edge of G_{k-1} . Furthermore, similarly to the proof of Lemma 4.2.3, one can easily prove by induction on k that any number of executions of the shift operation for G_{k-1} introduce no edge-intersection in G_{k-1} . Thus Algorithm **Draw-Triangle** obtains a grid drawing of G' .

All operations in Algorithm **Draw-Triangle** except the shift operation can be executed total in time $O(n)$. A simple implement of the shift operation takes time $O(n)$, and the algorithm executes the shift operation at most $\lceil n/2 \rceil$ times. Therefore a straightforward implementation would take time $O(n^2)$. However, using the data structure described in Section 4.2.3 for representing the sets $L(w_i)$, $1 \leq i \leq t$, one can implement the shift operation so that the total time required by the operation is $O(n)$.

Thus Algorithm **Draw-Triangle** finds a drawing of G' in time $O(n)$. \square

By modifying step 5 of algorithm **Four-Connected-Draw**, one can slightly improve the bound $H = \lceil n/2 \rceil$ in Theorem 4.4.3 to a bound $H = \lfloor n/2 \rfloor$. The modification can be done on the following observation. One

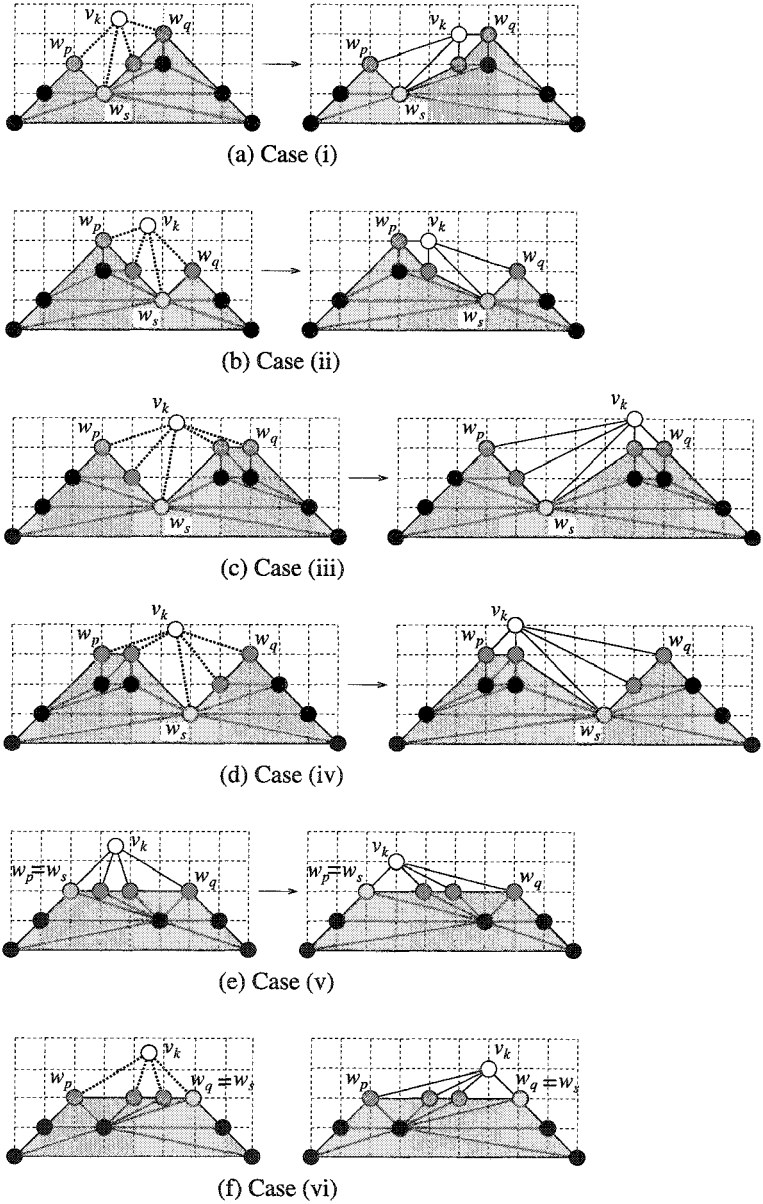


Fig. 4.31 How to put v_k .

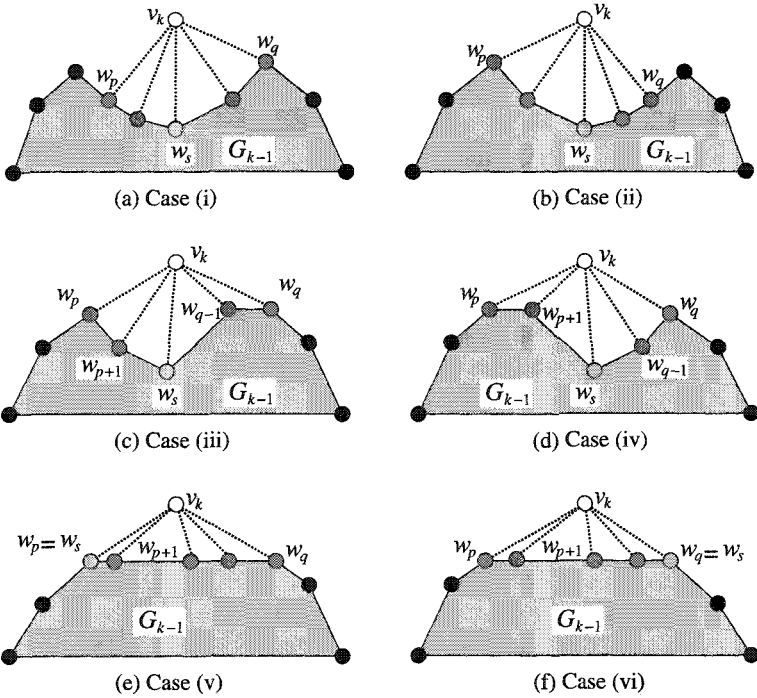


Fig. 4.32 Illustration of Cases (i)–(iv).

may assume without loss of generality that n is an odd integer, because $\lceil n/2 \rceil = \lfloor n/2 \rfloor$ if n is an even integer. We furthermore assume that $n = 3 \pmod{4}$; the argument for the case $n = 1 \pmod{4}$ is similar to one for the case $n = 3 \pmod{4}$. Since G' has $n' = \lceil n/2 \rceil$ vertices, G' has a grid drawing in an isosceles right triangle $\Delta' = A'B'C'$ whose base $B'C'$ has length $W' = n' - 1 = \lceil n/2 \rceil - 1$ and whose height is $H' = W'/2$. On the other hand, since G'' has $n'' = \lfloor n/2 \rfloor = n' - 1$ vertices, G'' has a grid drawing in an isosceles right triangle $\Delta'' = A''B''C''$ which is smaller than Δ' ; the base $B''C''$ of Δ'' has length $W'' = n'' - 1 = \lfloor n/2 \rfloor - 1$, and the height of Δ'' is $H'' = W''/2$. Since $n = 3 \pmod{4}$, A' is not a grid point, but A'' is a grid point as illustrated in Fig. 4.33. Therefore G' has no vertex on A' , but G'' may have a vertex on A'' . Fixing the position of the triangle Δ'' either as in Fig. 4.33(a) or as in Fig. 4.33(b), one can improve the bound $H = \lceil n/2 \rceil$ to a bound $H = H' + H'' + 1/2 = \lfloor n/2 \rfloor$. See [MNN01] for the detail.

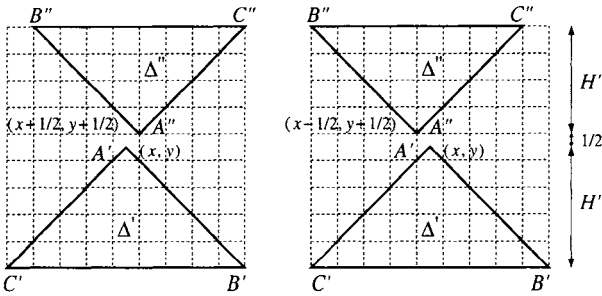


Fig. 4.33 Arrangement of triangles Δ' and Δ'' .

4.5 Bibliographic Notes

Kao *et al.* [KFHR94] efficiently parallelized the construction of realizers of triangulated planar graphs and gave an optimal parallel algorithm for straight line grid drawings of planar graphs. Di Battista *et al.* [DTV99] extended the definition of realizers to 3-connected planar graphs and gave efficient output-sensitive algorithms for performing “ k -path queries,” and also gave an algorithm for finding convex grid drawings of 3-connected planar graphs. Miura *et al.* [MTNN99] extended the definition of realizers to four-connected plane graphs and used it to solve the “independent spanning tree problem.” Generalizing a canonical ordering and a realizer, Chiang *et al.* introduced the concept of an “orderly spanning tree” for maximal planar graphs, which has many applications in graph encoding and graph drawing [CLL01]. Miura *et al.* showed that a Schnyder labeling, a realizer, a canonical decomposition, an orderly spanning tree and an “outer triangular” convex grid drawing are notions equivalent with each other for plane graphs such that each vertex has degree three or more [MAN04].

Exercise

1. Write a program to implement the algorithm of de Fraysseix *et al.* so that it takes time $O(n^2)$.
2. Using the shift method, design an algorithm to find a straight line grid drawing of a plane graph of n vertices on an $(n - 1) \times (n - 1)$ grid.
3. Design and implement a linear algorithm to construct a Schnyder labeling of a triangulated plane graph.
4. Let G be a labeled triangulated plane graph with realizer T_1, T_2, T_3 . For

an inner vertex v of G , the i -path $P_i(v)$ is defined as a path in T_i from v to the root of T_i . Show that for $i, j \in \{1, 2, 3\}$, $i \neq j$, $P_i(v)$ and $P_j(v)$ have v as only the common vertex.

5. Design and implement a linear algorithm for computing a 4-canonical ordering.

Chapter 5

Convex Drawing

5.1 Introduction

Some planar graphs can be drawn in such a way that each edge is drawn as a straight line segment and each face is drawn as a convex polygon, as illustrated in Figs. 1.5(b) and 5.1(b). Such a drawing is called a *convex drawing*. The drawings in Figs. 5.1(d) and (f) are not convex drawings. Although not every planar graph has a convex drawing, Tutte showed that every 3-connected planar graph has a convex drawing, and obtained a necessary and sufficient condition for a plane graph to have a convex drawing [Tut60]. Furthermore, he gave a “barycentric mapping” method for finding a convex drawing of a plane graph, which requires solving a system of $O(n)$ linear equations [Tut63]. The system of equations can be solved either in $O(n^3)$ time and $O(n^2)$ space using the ordinary Gaussian elimination method, or in $O(n^{1.5})$ time and $O(n \log n)$ space using the sparse Gaussian elimination method [LRT79]. Thus the barycentric mapping method leads to an $O(n^{1.5})$ time convex drawing algorithm for plane graphs.

In Sections 5.2 and 5.3 we present two linear algorithms for the convex drawing problem of planar graphs: drawing and testing algorithms [CYN84]. The former finds a convex drawing of a given *plane* graph G if there is; it extends a given convex polygonal drawing of the outer cycle of G into a convex drawing of G . The latter algorithm tests the possibility for a given *planar* graph. That is, it examines whether a given planar graph has a *plane embedding* which has a convex drawing.

A convex drawing is called a *convex grid drawing* if it is a grid drawing. Every 3-connected plane graph has a convex grid drawing on an $(n - 2) \times (n - 2)$ grid, and there is an algorithm to find such a grid drawing in linear time [CK97]. In Section 5.3 we describe the algorithm.

One may expect that the size of an integer grid required by a convex grid drawing will be smaller than $(n - 2) \times (n - 2)$ for 4-connected plane graphs. In Section 5.5 we present an algorithm which finds in linear time a convex grid drawing of any given 4-connected plane graph G on an integer grid such that $W + H \leq n - 1$ if G has four or more outer vertices [MNN00]. Since $W + H \leq n - 1$, the area of the grid satisfies $W \cdot H \leq n^2/4$.

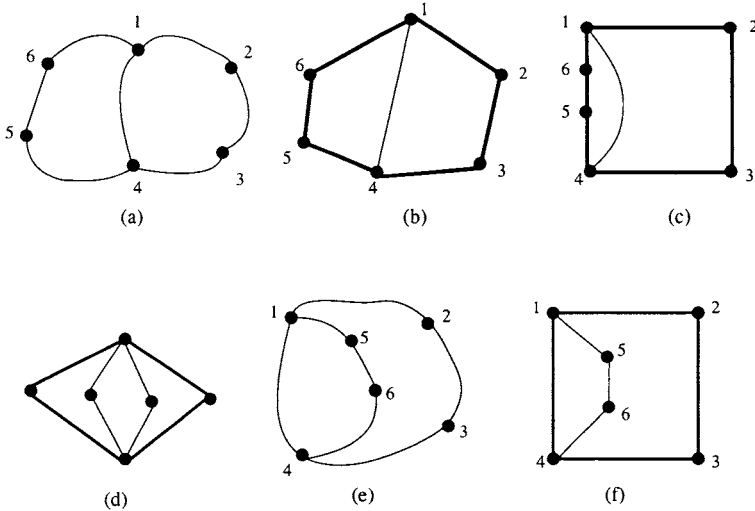


Fig. 5.1 Plane graphs and drawings.

5.2 Convex Drawing

In this section we present a linear algorithm for finding a convex drawing of a plane graph G [CYN84]. We first define terms and illustrative examples, then present a necessary and sufficient condition for G to have a convex drawing, and finally give the algorithm.

One may assume that a plane graph G is 2-connected. A *convex drawing* of G is a straight line drawing of G such that all the face boundaries are drawn as convex polygons. A convex drawing of the plane graph in Fig. 5.1(a) is depicted in Fig. 5.1(b). Not every 2-connected plane graph has a convex drawing. For example, the 2-connected plane graph depicted in Fig. 5.1(d) has no convex drawing. In a convex drawing of a plane graph G the outer cycle $C_o(G)$ is also drawn as a convex polygon. The polygonal

drawing C_o^* of $C_o(G)$, called an *outer convex polygon*, plays a crucial role in finding a convex drawing of G . The plane graph G in Fig. 5.1(a) admits a convex drawing if an outer convex polygon C_o^* has all vertices 1, 2, 3, 4, 5, and 6 of $C_o(G)$ as the *apices* (i.e., geometric vertices) of C_o^* , as illustrated in Fig. 5.1(b). However, if C_o^* has only apices 1, 2, 3 and 4, then G does not admit a convex drawing as depicted in Fig. 5.1(c). We say that an outer convex polygon C_o^* is *extendible* if there exists a convex drawing of G in which $C_o(G)$ is drawn as C_o^* . Thus the outer convex polygon drawn by thick lines in Fig. 5.1(b) is extendible, while that in Fig. 5.1(c) is not.

Tutte established a necessary and sufficient condition for an outer convex polygon to be extendible [Tut60]. The following Theorem obtained by Thomassen [Tho80] is slightly more general than this result.

Theorem 5.2.1 *Let G be a 2-connected plane graph, and let C_o^* be an outer convex polygon of G . Let C_o^* be a k -gon, $k \geq 3$, and let P_1, P_2, \dots, P_k be the paths in $C_o(G)$, each corresponding to a side of the polygon C_o^* , as illustrated in Fig. 5.2(a). Then C_o^* is extendible if and only if the following Condition I holds.*

Condition I

- (a) *For each inner vertex v with $d(v) \geq 3$, there exist three paths disjoint except v , each joining v and an outer vertex;*
- (b) *$G - V(C_o(G))$ has no connected component H such that all the outer vertices adjacent to vertices in H lie on a single path P_i , and no two outer vertices in each path P_i are joined by an inner edge; and*
- (c) *any cycle containing no outer edge has at least three vertices of degree ≥ 3 .*

Proof. The situations violating Condition I(a), (b) or (c) are illustrated in Fig. 5.2. It is impossible to draw G so that both of the two faces marked by \times in Fig. 5.2(a) are convex polygons. It is similar for Figs. 5.2(b) and (c). One can thus easily observe the necessity of Condition I. On the other hand, the sufficiency is implied by the algorithm below, which always finds a convex drawing of a plane graph G if G and C_o^* satisfy Condition I. \square

Suppose that a 2-connected plane graph G and an outer convex polygon C_o^* satisfy Condition I. The convex drawing algorithm extends C_o^* into a convex drawing of G in linear time. For simplicity, we assume that every inner vertex has degree three or more in G . Otherwise, replace each maximal induced path not on $C_o(G)$ by a single edge joining its ends (the resulting

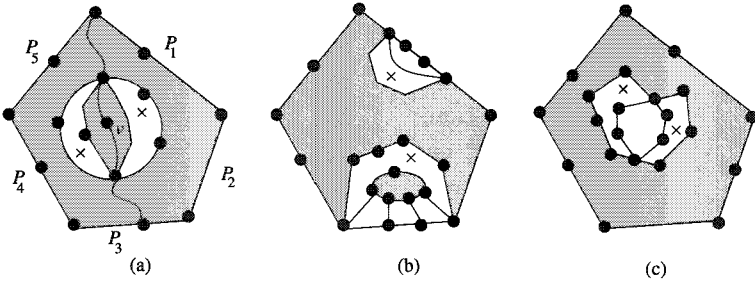


Fig. 5.2 G and C_o^* violating Condition I(a), (b) or (c).

simple graph G' satisfies Condition I); then find a convex drawing of G' ; and finally subdivide each edge substituting a maximal induced path.

The outline of the drawing algorithm is as follows. We reduce the convex drawing of G to those of several subgraphs of G as follows: delete from G an arbitrary apex v of the outer convex polygon C_o^* together with the edges incident to v ; divide the resulting graph $G' = G - v$ into blocks B_1, B_2, \dots, B_p , $p \geq 1$, as illustrated Fig. 5.3; determine an outer convex polygon C_i^* of each block B_i so that B_i with C_i^* satisfies Condition I; and recursively apply the algorithm to each block B_i with C_i^* to determine the position of inner vertices of B_i . The detail of the algorithm is as follows.

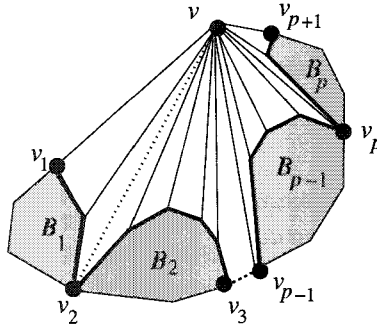


Fig. 5.3 Reduction of the convex drawing of G into subproblems.

Algorithm Convex-Drawing(G, C_o^*)

Step 1. Assume that G has at least four vertices and is not a single cycle;

TEAM LING - LIVE, INFORMATIVE, NON-COST AND GENUINE !

otherwise, a convex drawing of G has been obtained. Choose an arbitrary apex v of C_o^* , and let $G' := G - v$. Divide the plane graph G' into blocks B_i , $1 \leq i \leq p$. Let v_1 and v_{p+1} be outer vertices adjacent to v , and let v_i , $2 \leq i \leq p$, be the cut vertex of G' such that $v_i = V(B_{i-1}) \cap V(B_i)$, as illustrated Fig. 5.3.

{Every vertex v_i , $1 \leq i \leq p+1$, is an outer vertex of G since G and C_o^* satisfy Condition I(a) and every inner vertex of G has degree three or more.}

Step 2. Find a convex drawing of each block B_i , $1 \leq i \leq p$, by applying the following procedures.

Step 2.1 Determine an outer convex polygon C_i^* of B_i as follows.

{Since the positions of the vertices in $C_o(G)$ have been already determined on C_o^* , it remains to determine the positions of the vertices in $C_o(B_i) - C_o(G)$.}

Locate the vertices in $C_o(B_i) - C_o(G)$ in the interior of the triangle v, v_i, v_{i+1} in such a way that all the vertices adjacent to v are apices of C_i^* and the others are not apices of C_i^* .

Step 2.2 Recursively call the procedure **Convex-Drawing**(B_i, C_i^*) to extend C_i^* to a convex drawing of B_i .

{Note that B_i and C_i^* satisfy Condition I and every inner vertex of B_i has degree three or more.}

We have the following result on the algorithm.

Theorem 5.2.2 *Let G be a 2-connected plane graph, and let C_o^* be an extendible outer convex polygon of G . Then Algorithm **Convex-Drawing** extends C_o^* into a convex drawing of G in linear time.*

Proof. We first prove that Algorithm **Convex-Drawing** correctly finds a convex drawing of G . Each inner face F containing the apex v is a convex polygon in a drawing obtained by the algorithm; if either F contains none of v_2, v_3, \dots, v_p or F contains an edge (v, v_i) , $2 \leq i \leq p$, then F is drawn as a triangle; if F contains v_i , $2 \leq i \leq p$, but (v, v_i) is not an edge of G , then F is drawn as a convex quadrilateral obtained from two triangles by patching them along their side vv_i , which is drawn by a dotted line in Fig. 5.3. Therefore, in order to prove inductively the correctness of the algorithm, it suffices to observe that every block B_i with C_i^* satisfies Condition I. If B_i violated Condition I(a) or (c) then G would do the same one, while if B_i violated (b) then G would do either (a), (b) or (c).

We then prove the claim on time complexity. A path joining vertices x

and y is called an x - y path. Let P be the v_1 - v_{p+1} path in the outer cycle $C_o(G')$ of $G' = G - v$ which newly appears on the outer cycle. P is drawn by thick lines in Fig. 5.3. Traversing P , one can easily

- (1) find the vertices v_i , $1 \leq i \leq p + 1$, which appear on both $C_o(G)$ and $C_o(G')$,
- (2) obtain the outer cycle $C_o(B_i)$ of B_i as the union of the v_i - v_{i+1} path on P and the v_{i+1} - v_i path on $C_o(G)$, and
- (3) decide the positions of the vertices of $C_o(B_i)$ as specified in Step 2.1.

Thus we can implement the algorithm so that the required time, exclusive of recursive calls to itself, is proportional to the number of traversed edges in P , that is, the edges that newly appear on the outer cycle. Since every edge appears on the outer cycle at most once, the number of edges traversed during an execution of **Convex-Drawing** is at most m in total, where m is the number of edges in G . Thus **Convex-Drawing** runs in linear time. \square

We say that an outer convex polygon C_o^* of a plane graph G is *strict* if every vertex of $C_o(G)$ is an apex of the polygon C_o^* . The outer convex polygon in Fig. 5.1(b) is strict, while that in Fig. 5.1(c) is not. Theorem 5.2.1 implies that if there is an extendible outer convex polygon of G then any outer strict convex polygon of G is extendible, too.

5.3 Convex Testing

In this section, we present a linear algorithm to examine whether a *planar* graph has a plane embedding with a convex drawing [CYN84].

We call a cycle C of a planar graph G a *facial cycle* if there exist a plane embedding of G in which C is embedded as a boundary of a face. Thus if F is a facial cycle of G , then G has a plane embedding in which F is embedded as an outer face. A planar graph may have a convex drawing for a particular embedding, but may not have a convex drawing for another embedding. For example, although the two embeddings in Figs. 5.1(a) and (e) are of the same planar graph, the embedding in Fig. 5.1(a) has a convex drawing as illustrated in Fig. 5.1(b), while the embedding in Fig. 5.1(e) has no convex drawing as illustrated in Fig. 5.1(f). We say that a facial cycle F of a planar graph G is *extendible* if there is a plane embedding of G whose outer cycle is F and which has a convex drawing for some outer convex polygon F^* , say for an outer *strict* convex polygon. Thus the facial

cycle 1,2,3,4,5,6 of the graph in Fig. 5.1(a) is extendible, while 1,2,3,4 is not. It is rather easy to design a linear algorithm which only examines whether an outer convex polygon F^* of a plane graph is extendible, that is, satisfies Condition I. However, a planar graph may have an exponential number of facial cycles, so it is impractical to test all the facial cycles of a graph one by one through such an algorithm. We thus modify Condition I in Lemma 5.2.1 into a form suitable for our purpose, which is represented in terms of “3-connected components.” One may easily notice that the existence of a convex drawing of a graph G heavily depends on the structure of “3-connected components” of G .

This section is organized as follows. In Section 5.3.1 we give definitions of 3-connected components and separation pairs. In Section 5.3.2 we express Condition I in terms of 3-connected components. Section 5.3.3 gives a linear convex testing algorithm.

5.3.1 Definitions

In this subsection a “graph” means the so-called multigraph, so some terms defined in Chapter 2 shall be redefined.

A pair $\{x, y\}$ of vertices of a 2-connected graph $G = (V, E)$ is called a *separation pair* if there exists two subgraphs $G'_1 = (V_1, E'_1)$ and $G'_2 = (V_2, E'_2)$ satisfying the following conditions (a) and (b):

- (a) $V = V_1 \cup V_2, V_1 \cap V_2 = \{x, y\}$; and
- (b) $E = E'_1 \cup E'_2, E'_1 \cap E'_2 = \emptyset, |E'_1| \geq 2, |E'_2| \geq 2$.

The graph G in Fig. 5.4(a) has six separation pairs $\{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 7\}, \{3, 6\}$, and $\{4, 5\}$.

For a separation pair $\{x, y\}$ of G , $G_1 = (V_1, E'_1 + (x, y))$ and $G_2 = (V_2, E'_2 + (x, y))$ are called the *split graphs* of G . The new edges (x, y) added to G_1 and G_2 are called the *virtual edges*. Even if G has no multiple edges, G_1 and G_2 may have. Dividing a graph G into two split graphs G_1 and G_2 are called *splitting*. Reassembling the two split graphs G_1 and G_2 into G is called *merging*. Merging is the inverse of splitting. Suppose that a graph G is split, the split graphs are split, and so on, until no more splits are possible, as illustrated in Fig. 5.4(b) where virtual edges are drawn by dotted lines. The graphs constructed in this way are called the *split components* of G . The split components are of three types: triple bonds (i.e. a set of three multiple edges), triangles, and 3-connected graphs. The *3-connected components* of G are obtained from the split components of G by merging triple

bonds into a bond and triangles into a ring, as far as possible, where a *bond* is a set of multiple edges, and a *ring* is a cycle. The graph in Fig. 5.4(a) is decomposed into seven 3-connected components H_1, H_2, \dots, H_7 as depicted in Fig. 5.4(c), where H_1, H_2 and H_6 are 3-connected graphs, H_3, H_5 and H_7 are rings, and H_4 is a bond. The split components of G are not necessarily unique, but the 3-connected components of G are unique [HT73].

A separation pair $\{x, y\}$ is *prime* if x and y are the end vertices of a virtual edge contained in a 3-connected component. As known from Fig. 5.4(c), $\{1, 2\}, \{1, 3\}, \{2, 3\}$ and $\{4, 5\}$ are prime separation pairs, while $\{2, 7\}$ and $\{3, 6\}$ are not.

Suppose that $\{x, y\}$ is a prime separation pair of a graph G and that G is split at $\{x, y\}$, the split graphs are split, and so on, until no more splits are possible at $\{x, y\}$, as illustrated in Fig. 5.4(d). A graph constructed in this way is called an $\{x, y\}$ -*split component* of G if it has at least one real (i.e. non-virtual) edge. In Fig. 5.4(d), I_1, I_2, I_4 and I_5 are the $\{2, 3\}$ -split components, while I_3 is not.

In some cases it can be easily known only from the $\{x, y\}$ -split components for a single separation pair $\{x, y\}$ that a graph G has no convex drawing. A prime separation pair $\{x, y\}$ of G is called a *forbidden separation pair* if there are either

- (a) at least four $\{x, y\}$ -split components, as illustrated in Fig. 5.5(a), or
- (b) exactly three $\{x, y\}$ -split components each of which is neither a ring nor bond, as illustrated in Fig. 5.5(b).

Note that an $\{x, y\}$ -split component corresponds to an edge (x, y) if it is a bond, and to a subdivision of an edge (x, y) if it is a ring. The graph in Fig. 5.4(a) has exactly one forbidden separation pair $\{2, 3\}$. One can easily know that if a planar graph G has a forbidden separation pair then any plane embedding of G has no convex drawing, that is, G has no extendible facial cycle.

On the other hand, the converse of the fact above is not true. In order to be more precise, we need one more term. A prime separation pair $\{x, y\}$ is called a *critical separation pair* if there are either

- (i) exactly three $\{x, y\}$ -split components including a bond or a ring, as illustrated in Figs. 5.6(a) and (b), or
- (ii) exactly two $\{x, y\}$ -split components each of which is neither a bond nor a ring, as illustrated in Fig. 5.6(c).

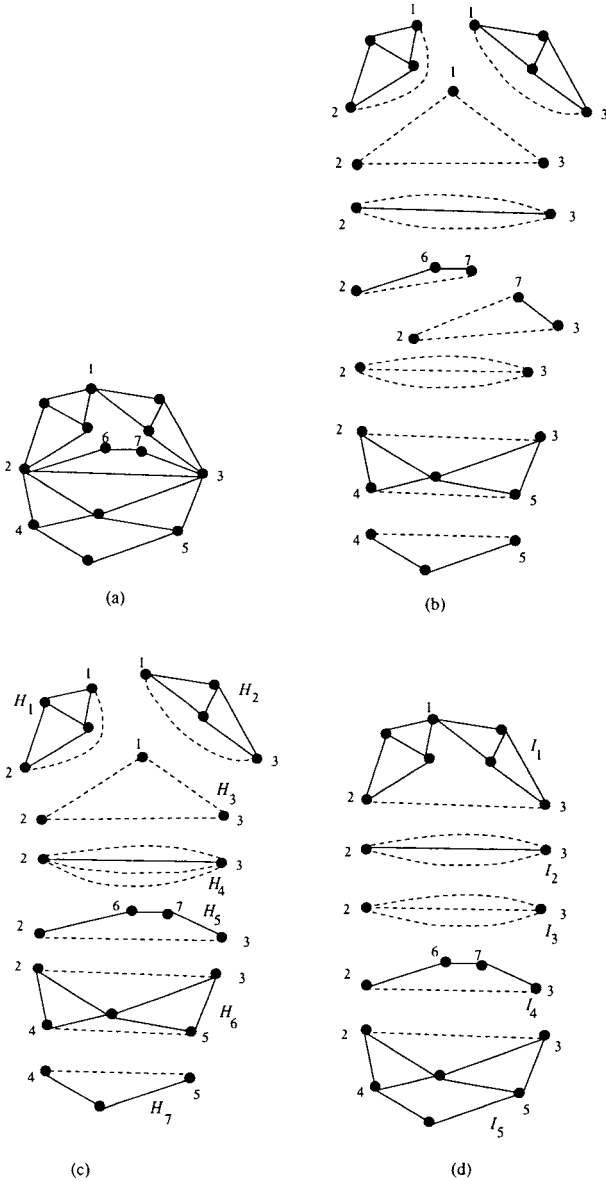


Fig. 5.4 (a) A 2-connected graph, (b) split components, (c) 3-connected components, and (d) $\{2,3\}$ -split components with one exception I_3 .

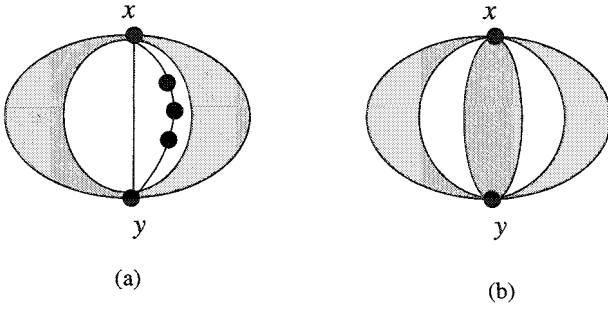


Fig. 5.5 Planar graphs with forbidden separation pairs $\{x, y\}$.

In the graph of Fig. 5.4(a), prime separation pairs $\{1, 2\}$ and $\{1, 3\}$ are critical, but $\{4, 5\}$ is neither forbidden nor critical. When a planar graph G has no forbidden separation pair, two cases occur: if G has no critical separation pair either, then G is a subdivision of a 3-connected graph, and so every facial cycle of G is extendible; otherwise, that is, if G has critical separation pairs, then a facial cycle F of G may or may not be extendible, depending on the interaction of F and critical separation pairs. The detailed criterion, called Condition II, will be given in Section 5.3.2.

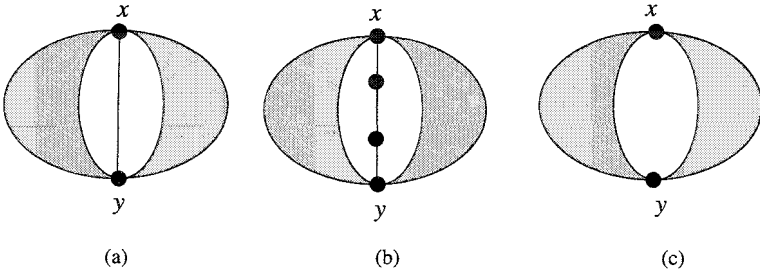


Fig. 5.6 Planar graphs with critical separation pairs $\{x, y\}$.

5.3.2 Condition II

We now give a condition suitable for the testing algorithm, which is equivalent to Condition I under a restriction that the outer convex polygon C_o^* is strict, that is, every vertex of $C_o(G)$ is an apex of C_o^* [CYN84].

Theorem 5.3.1 *Let $G = (V, E)$ be a 2-connected plane graph with the*

outer facial cycle $F = C_o(G)$, and let C_o^* be an outer strict convex polygon of G . Then C_o^* is extendible if and only if G and F satisfy the following Condition II.

Condition II.

- (a) G has no forbidden separation pair;
- (b) For each critical separation pair $\{x, y\}$ of G , there is at most one $\{x, y\}$ -split component having no edge of F , and, if any, it is either a bond if $(x, y) \in E$ or a ring otherwise.

Proof. We shall show that Condition II is equivalent to Condition I under the restriction that C_o^* is strict.

Condition I implies Condition II: Suppose that Condition I holds, and let $\{x, y\}$ be a prime separation pair of G .

Let I_1, I_2, \dots, I_k be the $\{x, y\}$ -split components having no edge of F . Then we claim that $k = 0$ or 1 and that if $k = 1$ then I_1 is either a bond or a ring. First suppose that one of the components I_1, I_2, \dots, I_k , say I_1 , is neither a bond nor a ring. Then I_1 has a vertex $v (\neq x, y)$ of degree three or more. Since I_1 has no outer edge, all vertices in I_1 other than x and y are inner vertices of G . Therefore G has no three paths disjoint except v , each joining v and an outer vertex, since such a path must contain either x or y . This contradicts Condition I(a). Thus every component $I_i, 1 \leq i \leq k$, must be either a bond or a ring. Next suppose that $k \geq 2$. Then I_1 together with I_2 forms a cycle which has no outer edge of G and has exactly two vertices x and y of degree three or more in G , contrary to Condition I(c). We have thus verified the claim.

Since at most two $\{x, y\}$ -split components contain outer edges, the claim above implies that there are at most three $\{x, y\}$ -split components and one of them is a bond or a ring if there are three. Thus $\{x, y\}$ is not a forbidden separation pair. Hence Condition II(a) holds.

Let $\{x, y\}$ be a critical separation pair. If $k = 0$ as illustrated in Fig. 5.6(c), then Condition II(b) holds for $\{x, y\}$. We may hence assume that $k = 1$ as illustrated in Figs. 5.6(a) and (b). Consider first the case $(x, y) \notin E$ as illustrated in Fig. 5.6(b). Then the $\{x, y\}$ -split component I_1 having no outer edge is not a bond. Therefore, the claim above implies that I_1 must be a ring, and hence Condition II(b) holds for this case. Consider next the case $(x, y) \in E$, as illustrated in Fig. 5.6(a). If the $\{x, y\}$ -split component I_1 having no outer edge was a ring as illustrated in Fig. 5.7, then (x, y) would be an outer edge and the connected component $I_1 - \{x, y\}$ of

$G - V(C_o(G))$ would be adjacent only with the outer vertices x and y in a side of C_o^* , contradicting Condition I(b). Note that edge (x, y) is a side of the outer strict convex polygon C_o^* as illustrated in Fig. 5.7(a), where C_o^* is drawn by thick lines. Thus I_1 must be a bond, and hence Condition II(b) holds for this case.

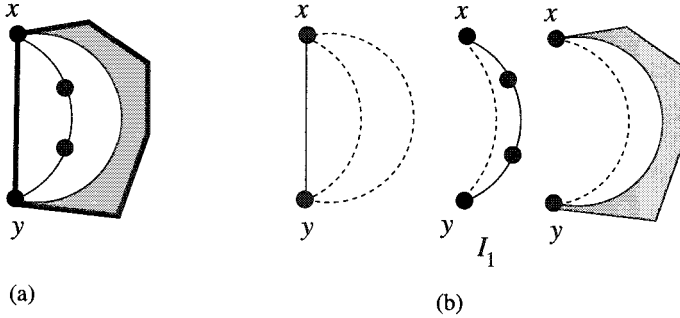


Fig. 5.7 (a) A plane graph G with a strict outer convex polygon C_o^* violating Condition II(b), and (b) $\{x, y\}$ -split components.

Condition II implies Condition I: Suppose for a contradiction that Condition II holds but Condition I does not hold.

First suppose that G has a vertex v of degree three or more, violating Condition I(a). Then, using Menger's theorem [Wes96], one can easily show that there exists a prime separation pair $\{x, y\}$ such that an $\{x, y\}$ -split component I_i contains v and has no outer edge. Since G is supposed to satisfy II(a), $\{x, y\}$ is not a forbidden separation pair. Thus $\{x, y\}$ is a critical separation pair. Since I_i contains a vertex v of degree three or more, I_i is neither a bond nor a ring, contradicting Condition II(b).

Next suppose that Condition I(b) is violated. Then $G - V(C_o(G))$ has a connected component H such that only the end-vertices x and y of an edge (x, y) on $C_o(G)$ are adjacent with vertices in H , because G has no multiple edges and C_o^* is strict. Clearly $\{x, y\}$ is a critical separation pair, and the $\{x, y\}$ -split component containing H has no outer edge and is not a bond. This contradicts Condition II(b).

Finally suppose that there exists a cycle C in G violating Condition I(c). Since G is 2-connected, C has exactly two vertices x and y of degree three or more. Clearly $\{x, y\}$ is a critical separation pair. If $(x, y) \in E$, then an $\{x, y\}$ -split component having no outer edge is a ring. Otherwise, there are two $\{x, y\}$ -split components having no outer edge. Either case

contradicts Condition II(b). \square

It should be noted that Condition II does not depend on the drawing C_o^* of $F = C_o(G)$ at all, and hence Theorem 5.3.1 leads to the following theorem.

Theorem 5.3.2 *A facial cycle F of a 2-connected planar graph G is extendible if and only if G and F satisfy Condition II.*

5.3.3 Testing Algorithm

Condition II is more suitable for testing than Condition I. In this subsection, we show that the convex testing of a planar graph G , i.e., examining whether G has a facial cycle F satisfying Condition II, can be reduced to planarity testing of a certain graph constructed from G .

Theorem 5.3.2 immediately yields the following corollaries.

Corollary 5.3.3 *A 2-connected planar graph G has no convex drawing for any facial cycle of G if G has a forbidden separation pair.*

Corollary 5.3.4 *A 2-connected planar graph G has a convex drawing for any facial cycle of G if G has neither a forbidden separation pair nor a critical separation pair and hence G is a subdivision of a 3-connected graph.*

Corollary 5.3.5 *Every 3-connected planar graph G has a convex drawing for any facial cycle of G .*

Corollary 5.3.6 *If a facial cycle F of a 2-connected planar graph G is extendible, then F contains every vertex of critical separation pairs of G .*

Proof. Since F is extendible, G and F satisfy Condition II. Suppose for a contradiction that $\{x, y\}$ is a critical separation pair of G and that F does not contain x . Then exactly one of the $\{x, y\}$ -split components contains all the edges of F . On the other hand, Condition II implies that there exists exactly one $\{x, y\}$ -split component containing no edges of F and it must be a bond or a ring. Thus there are exactly two $\{x, y\}$ -split components, one of which is a bond or a ring. Then $\{x, y\}$ could not be critical, a contradiction. \square

We will show later in Theorem 5.3.9 that the converse of Corollary 5.3.6 is also true in a certain sense. Before presenting Theorem 5.3.9 we need the following two lemmas.

Lemma 5.3.7 *Let G be a 2-connected planar graph, and let $\{x, y\}$ be a prime separation pair of G . If a facial cycle F of G contains both x and y , then exactly two of the $\{x, y\}$ -split components contain edges of F .*

Proof. Since F is a cycle, at most two of the $\{x, y\}$ -split components contain edges of F . Furthermore, since the facial cycle F contains both x and y , not all the edges of F are contained in a single $\{x, y\}$ -split component. Thus exactly two of the $\{x, y\}$ -split components contain edges of F . \square

Lemma 5.3.8 *Suppose that a 2-connected planar graph G has no forbidden separation pair and has exactly one critical separation pair. Then G has a convex drawing for some outer facial cycle.*

Proof. Let $\{x, y\}$ be the critical separation pair of G . By the definition of a critical separation pair, one can observe that G has one of the seven plane embeddings in Fig. 5.8; a shaded part corresponds to an $\{x, y\}$ -split component which is neither a bond nor a ring. In each case, one can easily verify that G and the outer facial cycle satisfy Condition II(b). \square

Thus we may concentrate on a graph having two or more critical separation pairs. We are now ready to present Theorem 5.3.9 which plays a crucial role in the testing algorithm.

Theorem 5.3.9 *Suppose that a 2-connected planar graph G has no forbidden separation pair and has two or more critical separation pairs as illustrated in Fig. 5.9(a), where all vertices in critical separation pairs are drawn by white circles. Apply the following operation to every critical separation pair $\{x, y\}$ of G :*

- (a) if $(x, y) \in E$, then delete edge (x, y) from G ;
- (b) if $(x, y) \notin E$ and exactly one $\{x, y\}$ -split component is a ring, then delete the x - y path in the component from G .

Let G_1 be the resulting planar graph, as illustrated in Fig. 5.9(b). Then F is an extendible facial cycle of G if and only if F is a facial cycle of G_1 and contains all the vertices of critical separation pairs of G .

Proof. *Necessity.* Assume that F is the extendible outer facial cycle of a plane embedding of G , as illustrated in Fig. 5.9(a). Then, since the embedding and F satisfy Condition II, all the deleted edges or paths are not on F , and hence F remains to be the outer cycle of the plane subgraph G_1 of G , as illustrated in Fig. 5.9(b). Moreover F contains all the vertices of critical separation pairs by Corollary 5.3.6.

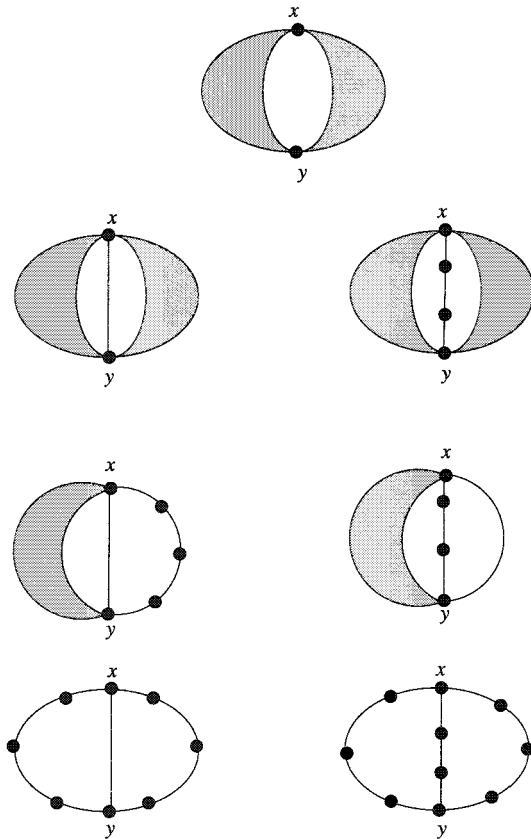


Fig. 5.8 Seven types of a plane graph having exactly one critical separation pair $\{x, y\}$.

Sufficiency. Assume that F is a facial cycle of G_1 which contains all the vertices of critical separation pairs of G , as illustrated in Fig. 5.9(b). Clearly F is also a facial cycle of G , as illustrated in Fig. 5.9(a). Since G has no forbidden separation pair, Condition II(a) holds. It thus suffices to show that every critical separation pair $\{x, y\}$ of G satisfies Condition II(b).

By Lemma 5.3.7, exactly two of the $\{x, y\}$ -split components, say I_1 and I_2 , contain edges of F . Therefore at most one of the $\{x, y\}$ -split components contains no edges of F .

Suppose for a contradiction that there is such a component I_3 which is neither a bond nor a ring. Then I_3 contains no vertex of critical separation pairs other than x and y ; if I_3 contained a vertex $z (\neq x, y)$ of a critical

separation pair, then I_3 would contain an edge of F since F contains all the vertices of critical separation pairs, a contradiction. Since G has at least two critical separation pairs, there is a critical separation pair $\{u, v\}$, different from $\{x, y\}$, and vertex u or v is not contained in I_3 . Hence I_1 or I_2 , say I_1 , is neither a bond nor a ring. The other component I_2 is a bond or a ring; otherwise, $\{x, y\}$ would be a forbidden separation pair. Then the edge (x, y) or the x - y path in I_2 should have been deleted in G_1 , and hence F could not be a facial cycle of G_1 , a contradiction. \square

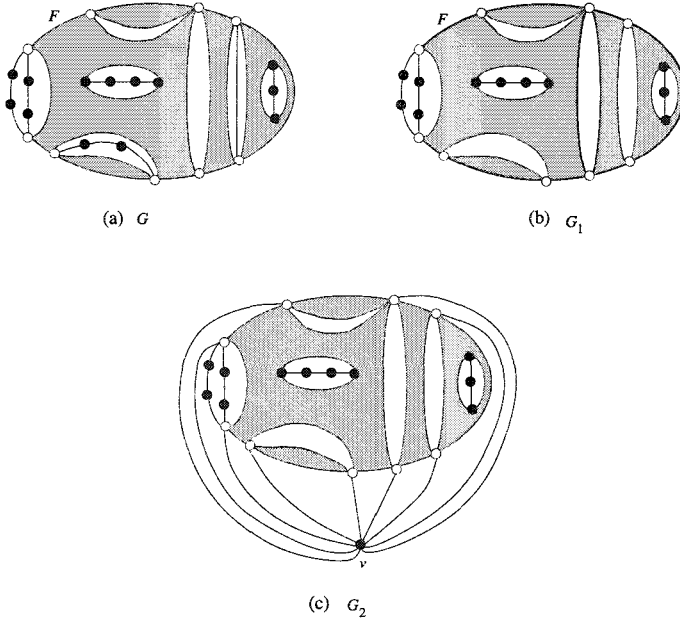


Fig. 5.9 Graphs (a) G , (b) G_1 , and (c) G_2 .

We define one more term before presenting two corollaries which follow immediately from Theorem 5.3.9. Let v be a vertex of a 2-connected plane graph G_2 , and let $G_1 = G_2 - v$ be 2-connected. Then the v -cycle of G_2 is the cycle of the plane subgraph G_1 of G_2 which bounds the face of G_1 in which v lay.

Corollary 5.3.10 *Suppose that a 2-connected planar graph G has no forbidden separation pair and has two or more critical separation pairs. Let G_1 be the graph defined in Theorem 5.3.9. Let G_2 be the graph obtained from G_1 by adding a new vertex v and joining v to all vertices of critical*

separation pairs of G , as illustrated in Fig. 5.9(c). Then F is an extendible facial cycle of G if and only if

- (a) G_2 is planar; and
- (b) F is the v -cycle of a plane embedding of G_2 .

Corollary 5.3.11 *Suppose that a 2-connected planar graph G has no forbidden separation pair and has two or more critical separation pairs. Let G_2 be the graph defined in Corollary 5.3.10. Then there is a plane embedding of G which has a convex drawing if and only if G_2 is planar.*

Combining Lemma 5.3.8 and Corollaries 5.3.3, 5.3.4, 5.3.10, and 5.3.11, one can easily devise a linear testing algorithm (Exercise 2). It is also easy to find all extendible facial cycles of a given planar graph (Exercise 3) [CYN84].

5.4 Convex Grid Drawings of 3-Connected Plane Graphs

In this section we describe a linear algorithm for finding a convex grid drawing of a 3-connected plane graph on an $(n-2) \times (n-2)$ grid [CK97]. The algorithm is based on a “canonical decomposition” of a 3-connected plane graph, which is a generalization of a canonical ordering described in Section 4.2.1. In Section 5.4.1 we present a canonical decomposition, and in Section 5.4.2 we present the algorithm.

5.4.1 Canonical Decomposition

We say that a plane graph G is *internally 3-connected* if G is 2-connected and, for any separation pair $\{u, v\}$ of G , u and v are outer vertices and each connected component of $G - \{u, v\}$ contains an outer vertex. In other words, G is internally 3-connected if and only if it can be extended to a 3-connected graph by adding a vertex in an outer face and connecting it to all outer vertices. If a 2-connected plane graph G is not internally 3-connected, then G has a separation pair $\{u, v\}$ of one of the three types illustrated in Fig. 5.10, where the split component I contains a vertex other than u and v . If an internally 3-connected plane graph G is not 3-connected, then G has a separation pair of outer vertices and hence G has a “chord-path” when G is not a single cycle.

We now define a “chord-path.” Let G be a 2-connected plane graph, and let w_1, w_2, \dots, w_t be the vertices appearing clockwise on the outer cycle

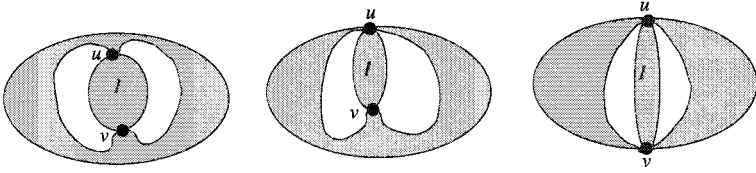


Fig. 5.10 Biconnected plane graphs which are not internally 3-connected.

$C_o(G)$ in this order, as illustrated in Fig. 5.11. We call a path P in G a *chord-path* of the cycle $C_o(G)$ if P satisfies the following (i)–(iv):

- (i) P connects two outer vertices w_p and w_q , $p < q$;
- (ii) $\{w_p, w_q\}$ is a separation pair of G ;
- (iii) P lies on an inner face; and
- (iv) P does not pass through any outer edge and any outer vertex other than the ends w_p and w_q .

The plane graph G in Fig. 5.11 has six chord-paths P_1, P_2, \dots, P_6 drawn by thick lines. A chord-path P is *minimal* if none of $w_{p+1}, w_{p+2}, \dots, w_{q-1}$ is an end of a chord-path. Thus the definition of a minimal chord-path depends on which vertex is considered as the starting vertex w_1 of $C_o(G)$. P_1, P_2, P_3 and P_6 in Fig. 5.11 are minimal, while P_4 and P_5 are not minimal.

Let $\{v_1, v_2, \dots, v_p\}$, $p \geq 3$, be a set of three or more outer vertices consecutive on $C_o(G)$ such that $d(v_1) \geq 3$, $d(v_2) = d(v_3) = \dots = d(v_{p-1}) = 2$, and $d(v_p) \geq 3$. Then we call the set $\{v_2, v_3, \dots, v_{p-1}\}$ an *outer chain* of G . The graph in Fig. 5.11 has two outer chains $\{w_4, w_5\}$ and $\{w_8\}$.

We are now ready to define a canonical decomposition. Let $G = (V, E)$ be a 3-connected plane graph of $n \geq 4$ vertices like one in Fig. 5.13. For an ordered partition $\Pi = (U_1, U_2, \dots, U_l)$ of set V , we denote by G_k , $1 \leq k \leq l$, the subgraph of G induced by $U_1 \cup U_2 \cup \dots \cup U_k$, while we denote by \overline{G}_k , $0 \leq k \leq l-1$, the subgraph of G induced by $U_{k+1} \cup U_{k+2} \cup \dots \cup U_l$. Clearly $G_k = G - U_{k+1} \cup U_{k+2} \cup \dots \cup U_l$, and $G = G_l = \overline{G}_0$. Let (v_1, v_2) be an outer edge of G . We then say that Π is a *canonical decomposition* of G (for an outer edge (v_1, v_2)) if Π satisfies the following conditions (cd1)–(cd3).

- (cd1) U_1 is the set of all vertices on the inner face U containing edge (v_1, v_2) , and U_l is a singleton set containing an outer vertex $v_n \notin \{v_1, v_2\}$.
- (cd2) For each index k , $1 \leq k \leq l$, G_k is internally 3-connected.
- (cd3) For each index k , $2 \leq k \leq l$, all vertices in U_k are outer vertices of G_k and the following conditions hold:

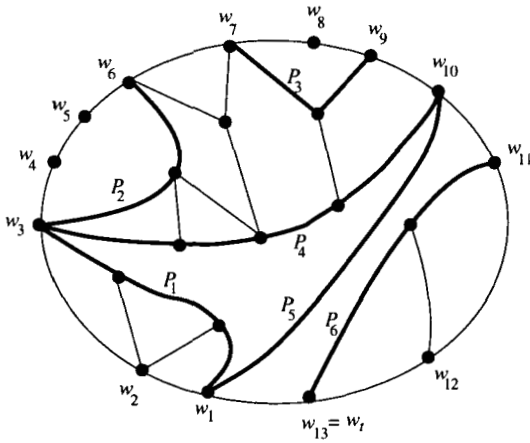


Fig. 5.11 A plane graph with chord-paths P_1, P_2, \dots, P_6 .

- (a) if $|U_k| = 1$, then the vertex in U_k has two or more neighbors in G_{k-1} and has at least one neighbor in $\overline{G_k}$ when $k < l$, as illustrated in Fig. 5.12(a); and
- (b) If $|U_k| \geq 2$, then U_k is an outer chain of G_k , and each vertex in U_k has at least one neighbor in $\overline{G_k}$, as illustrated in Fig. 5.12(b).

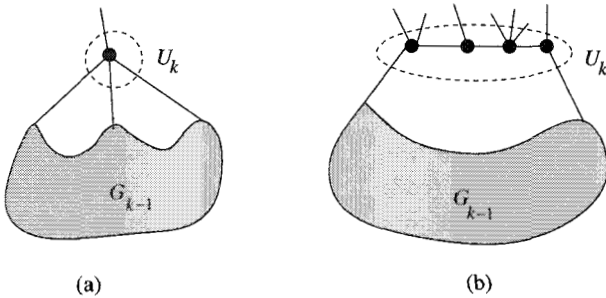


Fig. 5.12 G_k with some edges joining U_k and $\overline{G_k}$.

Figure 5.13 illustrates a canonical decomposition $\Pi = (U_1, U_2, \dots, U_8)$ of a 3-connected plane graph of $n = 15$ vertices. We now have the following lemma on a canonical decomposition.

Lemma 5.4.1 *Every 3-connected plane graph G of $n \geq 4$ vertices has a canonical decomposition Π , and Π can be found in linear time.*

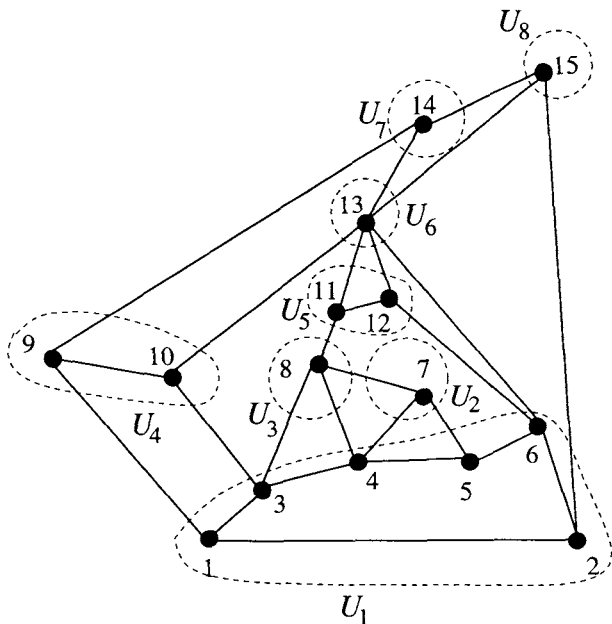


Fig. 5.13 A canonical decomposition of a 3-connected plane graph.

Proof. We first show that G has a canonical decomposition. Let U_1 be the set of all vertices on the inner face containing edge (v_1, v_2) . Since G is 3-connected and $n \geq 4$, there is an outer vertex $v_n \notin U_1$. We choose the singleton set $\{v_n\}$ as U_l . Thus (cd1) holds. Since $G_l = G$, (cd2) holds for $k = l$. Since G is 3-connected and v_n is on $C_o(G)$, $G_{l-1} = G - v_n$ is internally 3-connected and hence (cd2) holds for $k = l - 1$. Since v_n has degree three or more in G , (cd3) holds for $k = l$. If $V = U_1 \cup U_l$, then simply setting $l = 2$ completes the proof. One may thus assume that $V \supset U_1 \cup U_l$ and hence $l \geq 3$. We choose $U_{l-1}, U_{l-2}, \dots, U_2$ in this order and show that (cd2) and (cd3) hold.

Assume for inductive hypothesis that $l \geq i + 1 \geq 3$ and the sets $U_l, U_{l-1}, \dots, U_{i+1}$ have been appropriately chosen so that

- (1) (cd2) holds for each index $k, l \geq k \geq i$, and
- (2) (cd3) holds for each index $k, l \geq k \geq i + 1$.

We then show that there is a set U_i of outer vertices of G_i such that

- (1) (cd2) holds for the index $k = i - 1$, and
- (2) (cd3) holds for the index $k = i$.

Let w_1, w_2, \dots, w_t be the outer vertices of G_i appearing clockwise on

$C_o(G_i)$ in this order, where $w_1 = v_1$ and $w_t = v_2$. There are the following two cases to consider.

Case 1: G_i is 3-connected.

Since G_i is 3-connected and a vertex in U_{i+1} has a neighbor in G_i , there is an outer vertex $w \notin U_1$ of G_i which has a neighbor in $\overline{G_i}$. We choose the singleton set $\{w\}$ as U_i . Since G_i is 3-connected and w is an outer vertex of G_i , $G_{i-1} = G_i - w$ is internally 3-connected and w has three or more neighbors in G_{i-1} . Thus (cd2) holds for $k = i - 1$, and (cd3) holds for $k = i$.

Case 2: Otherwise.

Since $i \geq 2$, G_i is not a single cycle. G_i is internally 3-connected, but is not 3-connected. Therefore there is a chord-path for $C_o(G_i)$. Let P be a minimal chord-path for $C_o(G)$, and let w_p and w_q be the two ends of P such that $p < q$. Then $q \geq p + 2$ since G_i is internally 3-connected and $\{w_p, w_q\}$ is a separation pair of G_i . We now have the following two subcases.

Subcase 2a: $\{w_{p+1}, w_{p+2}, \dots, w_{q-1}\}$ is an outer chain of G_i

In this case we choose $\{w_{p+1}, w_{p+2}, \dots, w_{q-1}\}$ as U_i . Since U_i is an outer chain and P is a minimal chord-path, one can observe that $U_i \cap U_1 \neq \emptyset$. Since G is 3-connected and each vertex $w \in U_i$ has degree two in G_i , each vertex $w \in U_i$ has a neighbor in $\overline{G_i}$ and hence (cd3) holds for $k = i$.

We now claim that G_{i-1} is internally 3-connected and hence (cd2) holds for $k = i - 1$. Assume for a contradiction that G_{i-1} is not internally 3-connected. Then G_{i-1} has either a cut vertex v or a separation pair $\{u, v\}$ having one of the three types illustrated in Fig. 5.10.

Consider first the case where G_{i-1} has a cut vertex v . Then v must be an outer vertex of G_i and $v \neq w_p, w_q$; otherwise, G_i would not be internally 3-connected. Then the minimal chord-path P above must pass through v as illustrated in Fig 5.14, contrary to the Condition (iv) of the definition of a chord-path.

Consider next the case where G_{i-1} has a separation pair $\{u, v\}$ having one of the three types illustrated in Fig. 5.10. Then $\{u, v\}$ would be a separation pair of G_i having one of the three types illustrated in Fig. 5.10, and hence G_i would not be internally 3-connected, a contradiction.

Subcase 2b: Otherwise.

In this case, any vertex $w \in \{w_{p+1}, w_{p+2}, \dots, w_{q-1}\}$ has degree three or more in G_i ; otherwise, P would not be minimal. At least one vertex $w \in \{w_{p+1}, w_{p+2}, \dots, w_{q-1}\}$ has a neighbor in $\overline{G_i}$; otherwise, $\{w_p, w_q\}$ would be a separating pair of G and hence G would not be 3-connected. We choose the singleton set $\{w\}$ as U_i . Then clearly $U_i \cap U_1 = \emptyset$, and (cd3)

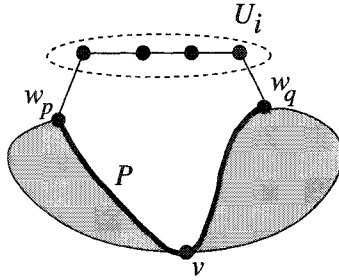


Fig. 5.14 G_i in which P passes through an outer vertex v .

holds for $k = i$. Since w is not an end of a chord-path of $C_o(G_i)$ and G_i is internally 3-connected, $G_{i-1} = G_i - w$ is internally 3-connected and hence (cd2) holds for $k = i - 1$.

Thus we have proved that there exists a canonical decomposition.

One can implement an algorithm for finding a canonical decomposition, based on the proof above. It maintains a data structure to keep the outer chains and minimal chord-paths of $C_o(G_k)$. The algorithm traverses every face at most a constant number of times, and runs in linear time. \square

5.4.2 Algorithm for Convex Grid Drawing

In this section we describe a linear algorithm for finding a convex grid drawing of a 3-connected plane graph [CK97].

Let G be a 3-connected plane graph, and let $\Pi = (U_1, U_2, \dots, U_l)$ be a canonical decomposition of G . The algorithm will add to a drawing the vertices in set U_k , one by one, in the order U_1, U_2, \dots, U_l , adjusting the drawing at every step. Before giving the detail of the algorithm we need some preparation.

We say that a vertex $v \in U_k$, $1 \leq k \leq l$, has rank k . Let $2 \leq k \leq l$, and let $C_o(G_{k-1}) = w_1, w_2, \dots, w_t$, where $w_1 = v_1$ and $w_t = v_2$. The definition of a canonical decomposition implies that there is a pair of indices a and b , $1 \leq a < b \leq t$, such that each of w_a and w_b has a neighbor in $\overline{G_{k-1}}$ but any vertex w_i , $a < i < b$, has no neighbor in $\overline{G_{k-1}}$ and is an inner vertex of G , as illustrated in Fig. 5.15. (See also Fig. 5.12.) Then the path w_a, w_{a+1}, \dots, w_b is a part of an inner facial cycle F of G ; F also contains two edges connecting w_a and w_b with $\overline{G_{k-1}}$, plus possibly some edges in $\overline{G_{k-1}}$. Let c , $a \leq c < b$, be an index such that w_c has the smallest rank among the vertices $w_a, w_{a+1}, \dots, w_{b-1}$. If there are two or more vertices

with the smallest rank, then let w_c be the leftmost one, that is, let c be the smallest index of these vertices. Intuitively, the algorithm will work in such a way that, for any such pair of indices a and b , either the vertex w_c or w_{c+1} will have the smallest y -coordinate among the vertices on the face F . (See Fig. 5.15.) We denote the index c for a and b by $\mu_k^+(a)$ and $\mu_k^-(b)$. Thus $c = \mu_k^+(a) = \mu_k^-(b)$. The superscript $+$ indicates $a \leq c$, while the superscript $-$ indicates $c < b$. We often omit the subscript k for simplicity. Note that if $b = a + 1$ then $a = \mu_k^+(a) = \mu_k^-(b)$.

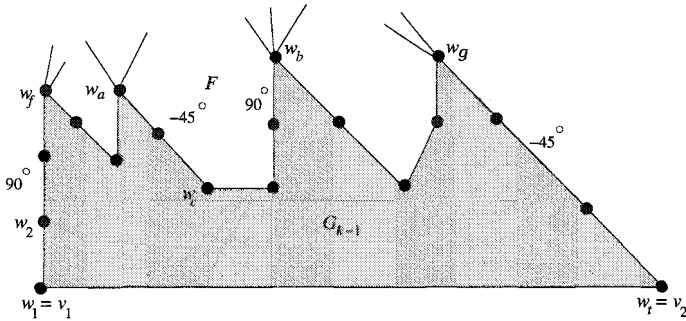


Fig. 5.15 G_{k-1} with some edges connecting G_{k-1} and $\overline{G_{k-1}}$.

We denote the current position of a vertex v by $P(v)$; $P(v)$ is expressed by its x - and y -coordinates as $(x(v), y(v))$. With each vertex v , a set of vertices need to be moved whenever the position of v is adjusted. We denote by $L(v)$ the set of such vertices.

We are now ready to describe the drawing algorithm.

First we draw $C_o(G_1) = w_1, w_2, \dots, w_t$ as follows. Set $P(w_1) = (0, 0)$, $P(w_t) = (t - 1, 0)$ and $P(w_i) = (i - 1, 0)$ for all indices $i = 2, 3, \dots, t - 1$, as illustrated in Fig. 5.16(a) for the graph in Fig. 5.13. Also set $L(w_i) = \{w_i\}$ for each index $i = 1, 2, \dots, t$.

Then, for each index $k = 2, 3, \dots, l$, we do the following. Let $C_o(G_{k-1}) = w_1, w_2, \dots, w_t$ be the outer cycle of G_{k-1} where $w_1 = v_1$ and $w_t = v_2$. Let $U_k = \{u_1, u_2, \dots, u_r\}$. U_k is either a singleton set or an outer chain of G_k , but in the algorithm we will treat both cases uniformly.

Let w_p and w_q be the leftmost and rightmost neighbors of U_k in G_{k-1} as illustrated in Figs. 5.17 and 5.18. Let $\alpha = \mu^+(p)$, and let $\beta = \mu^-(q)$. If U_k is an outer chain, then all vertices $w_{p+1}, w_{p+2}, \dots, w_{q-1}$ belong to the same inner face of G_k and none of them has a neighbor in $\overline{G_k}$ and hence $\alpha = \beta$, as illustrated in Fig. 5.18(a). If U_k is a singleton set of a vertex u_1

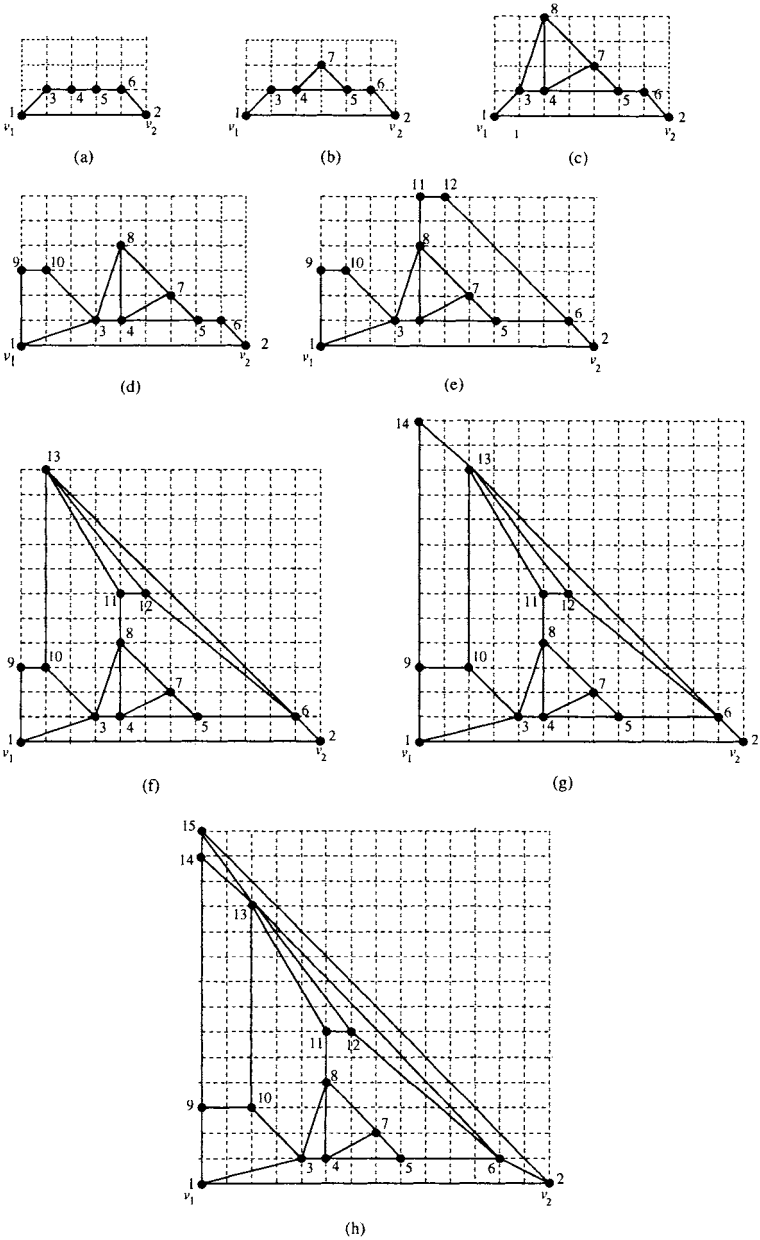


Fig. 5.16 Drawing process of the plane graph in Fig. 5.13.

having three or more neighbors in G_{k-1} , then at least one of the vertices $w_{p+1}, w_{p+2}, \dots, w_{q-1}$ has a neighbor in $\overline{G_k}$ and hence $\alpha < \beta$; in fact, w_α and w_β will belong to two different inner faces of G_k , to the first and last faces among those that are created when adding u_1 to G_{k-1} , as illustrated in Fig. 5.17. We thus execute the following steps.

Update:

$$\text{Set } L(w_p) = \bigcup_{i=p}^{\alpha} L(w_i);$$

$$\text{Set } L(w_q) = \bigcup_{i=\beta+1}^q L(w_i);$$

$$\text{Set } L(u_1) = \{u_1\} \cup \left(\bigcup_{i=\alpha+1}^{\beta} L(w_i) \right);$$

$$\text{Set } L(u_i) = \{u_i\} \text{ for each index } i, 2 \leq i \leq r;$$

Shift: For each vertex $v \in \bigcup_{i=q}^t L(w_i)$, set $x(v) = x(v) + r$;

Install U_k : Let ϵ be 0 if w_p has no neighbor in $\overline{G_k}$ and 1 otherwise. For each $i = 1, 2, \dots, r$, we set $x(u_i) = x(w_p) + i - 1 + \epsilon$, and set $y(u_i) = y(w_q) + x(w_q) - x(w_p) - r + 1 - \epsilon$. In other words, we draw U_k horizontally in such a way that the slope of the segment $u_r w_q$ is -45° . Vertex u_1 is placed above w_p if w_p has no neighbor in $\overline{G_k}$, and at the next x -coordinate otherwise. Note that in the last equation we use the new updated value of $x(w_q)$.

We call the algorithm above Algorithm **Convex-Grid-Drawing**. Figure 5.16 illustrates the execution of Algorithm **Convex-Grid-Drawing** for the plane graph in Fig. 5.13. The linear-time implementation of **Convex-Grid-Drawing** can be achieved by using a data structure presented in Section 4.2.3.

We now verify the correctness of Algorithm **Convex-Grid-Drawing**. Let $2 \leq k \leq l$, and let $C_o(G_{k-1}) = w_1, w_2, \dots, w_t$, where $w_1 = v_1$ and $w_t = v_2$. Then by induction on k it can be proved that in the drawing of G_{k-1} , $P(v_1) = (0, 0)$, $P(v_2) = (|\bigcup_{i=1}^{k-1} U_i| - 1, 0)$, and any line segment $w_i w_{i+1}$, $1 \leq i \leq t - 1$, has slope in $\{-45^\circ, 0^\circ\} \cup [45^\circ, 90^\circ]$ as illustrated in Fig. 5.15, where $[45^\circ, 90^\circ]$ denotes the set of all angles θ , $45^\circ \leq \theta \leq 90^\circ$. (Exercise 6.) More specifically, the following properties (a)–(c) hold on the

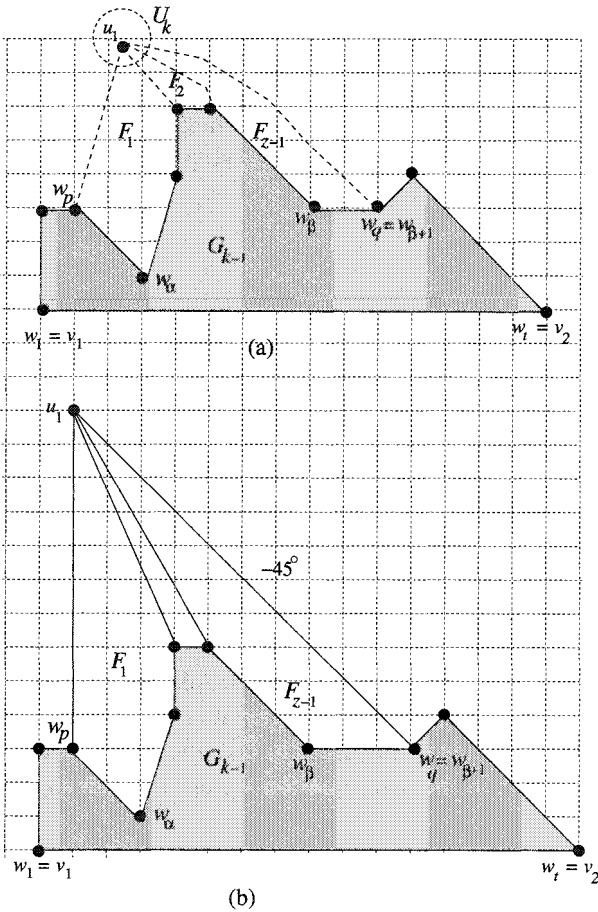


Fig. 5.17 (a) G_k before Shift, and (b) G_k after Shift and Install with $\epsilon = 0$.

slopes of line segments on $C_o(G_{k-1})$ in the drawing of G_{k-1} [CK97]:

(a) Let w_f , $1 \leq f \leq t$, be the first vertex on $C_o(G_{k-1})$ which has a neighbor in $\overline{G_{k-1}}$, then the slope of each line segment on the path w_1, w_2, \dots, w_f is 90° .

(b) Let w_g , $1 \leq g \leq t$, be the last vertex on $C_o(G_{k-1})$ which has a neighbor in $\overline{G_{k-1}}$, then the slope of each line segment on the path w_g, w_{g+1}, \dots, w_t is -45° .

(c) For any triple of indices a, b and c as defined earlier and illustrated in Fig 5.15, each of the first $c - a$ line segments on the path w_a, w_{a+1}, \dots, w_b has slope -45° , while each of the last $b - c - 1$ segments has slope 90° . The

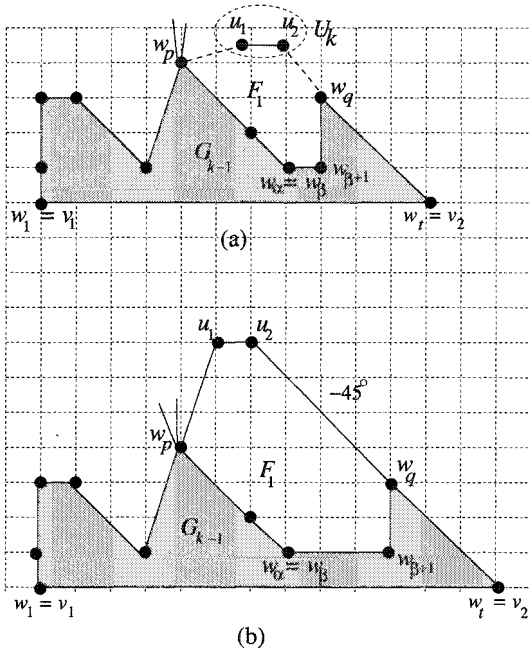


Fig. 5.18 (a) G_k before Shift, and (b) G_k after Shift and Install with $\epsilon = 1$.

remaining line segment $w_c w_{c+1}$ has slope in $\{-45^\circ, 0^\circ\} \cup [45^\circ, 90^\circ]$, and the slope is not 90° if $c = a$.

One can observe that, after the shift operation while adding U_k , all neighbors of U_k on $C_o(G_{k-1})$ are visible from the vertices in U_k . Hence the edges joining vertices in U_k and vertices in $C_o(G_{k-1})$ do not intersect themselves or edges on $C_o(G_{k-1})$. Thus adding U_k does not destroy the planarity. One can observe also that the newly created inner faces are convex polygons.

What remains to show is that we do not destroy the planarity and convexity when we apply the shift operation. We will prove this in the following lemma, which is similar to Lemma 4.2.3 for straight line drawings [CK97]. We call a drawing of a plane graph *internally convex* if all inner faces are drawn as convex polygons.

Lemma 5.4.2 (a) Each graph G_k , $1 \leq k \leq l$, is straight-line embedded and internally convex.

(b) Suppose that $C_o(G_k) = w'_1, w'_2, \dots, w'_{t'}$, $w'_1 = v_1$ and $w'_{t'} = v_2$, and that s is any index, $1 \leq s \leq t'$, and δ is any nonnegative integer. If we shift

all vertices in $\cup_{i=s}^{t'} L(w'_i)$ by δ to the right, then G_k remains straight-line embedded and internally convex.

Proof. The proof is by induction on k . For G_1 the lemma is obvious. So suppose that it holds for G_{k-1} , $k \geq 2$. Let $C_o(G_{k-1}) = w_1, w_2, \dots, w_t$, $w_1 = v_1$, and $w_t = v_2$ as in the algorithm. We are about to add U_k to the drawing of G_{k-1} . Let w_p and w_q be the leftmost and rightmost neighbors of U_k in $C_o(G_{k-1})$.

Let $U_k = \{u_1, u_2, \dots, u_r\}$, $r \geq 1$. Then the outer cycle of G_k is $C_o(G_k) = w'_1, w'_2, \dots, w'_{t'}$ where

$$t' = t + \xi,$$

$$\xi = r - q + p + 1,$$

and

$$w'_i = \begin{cases} w_i & \text{if } 1 \leq i \leq p; \\ u_{i-p} & \text{if } p+1 \leq i \leq p+r; \\ w_{i-\xi} & \text{if } p+r+1 \leq i \leq t'. \end{cases}$$

If $s \geq p+r+2$, then U_k does not move, and the lemma follows directly by induction. If $s \leq p$, then the lemma also follows from the inductive assumption, since U_k shifts rigidly with the rest of the graph.

Let us assume now that U_k is a singleton, that is, $U_k = \{u_1\}$. (The proof when U_k is a chain of two or more vertices is similar and is left to the reader.) Then it suffices to consider the two cases $s = p+1$ and $s = p+2$: $w'_s = u_1$ and $w'_s = w_q$. Let u_1 have exactly z neighbors in G_{k-1} , $z \geq 2$, and let F_1, F_2, \dots, F_{z-1} be the faces created when adding u_1 , as illustrated in Fig 5.17.

If $s = p+1$, that is, $w'_s = u_1$, then we apply the induction assumption to G_{k-1} with $s' = \mu_{k-1}^+(p) + 1 = \alpha + 1$. The straight-line embedding and internal convexity are preserved in G_{k-1} by induction. All faces F_2, F_3, \dots, F_{z-1} are shifted rigidly with G_{k-1} , and hence only F_1 will be deformed. However, in F_1 we will only stretch the edges (w_p, u_1) and $(w_{s'-1}, w_{s'}) (= (w_\alpha, w_{\alpha+1}))$, and by the choice of s' this will not destroy the convexity of F_1 .

If $s = p+2$, that is, $w'_s = w_q$, then the proof is similar: we apply the inductive assumption to G_{k-1} with $s'' = \mu_{k-1}^-(q) + 1$. In this case only F_{z-1} will be deformed, but by the choice of s'' the convexity of F_{z-1} will be preserved. \square

The conditions (a) and (b) on the slopes of line segments on $C_o(G_k)$ imply that the outer cycle $C_o(G)$ is drawn as an isosceles right triangle in the final drawing of G , as illustrated in Fig. 5.16(h). Clearly $x(v_1) = 0, x(v_2) = n - 1$ and $y(v_n) = n - 1$, and hence Algorithm Convex-Grid-Drawing produces a convex drawing of a 3-connected plane graph G on an $(n - 1) \times (n - 1)$ grid.

Using a canonical decomposition such that v_n is adjacent to v_2 and fixing the position of v_n carefully, one can obtain a convex grid drawing on an $(n - 2) \times (n - 2)$ grid. The modification of the algorithm is left as Exercise 4.

5.5 Convex Grid Drawings of 4-Connected Plane Graphs

In this section we present an algorithm for finding a convex grid drawing of a 4-connected plane graph G with four or more outer vertices [MNN00]. A “4-canonical decomposition” of a plane graph G [NRN97] plays a crucial role in the algorithm; it is a generalization of a canonical decomposition described in Section 5.4.

In Section 5.5.1 we present a 4-canonical decomposition and in Section 5.5.2 we present the algorithm.

5.5.1 Four-Canonical Decomposition

A 4-canonical decomposition $\Pi = (U_1, U_2, \dots, U_l)$ is illustrated in Fig. 5.19 for a 4-connected plane graph of $n = 21$ vertices. We call an ordered partition $\Pi = (U_1, U_2, \dots, U_l)$ of set V a *4-canonical decomposition* of a plane graph $G = (V, E)$ if the following three conditions are satisfied.

- (c1) U_1 consists of the two ends of an edge on $C_o(G)$, and U_l consists of the two ends of another edge on $C_o(G)$;
- (c2) For each index $k, 2 \leq k \leq l - 1$, both G_k and $\overline{G_{k-1}}$ are 2-connected (in Fig. 5.20 G_k is darkly shaded, and $\overline{G_{k-1}}$ is lightly shaded); and
- (c3) For each index $k, 2 \leq k \leq l - 1$, one of the following three conditions holds (the vertices in U_k are drawn by black circles in Fig. 5.20):
 - (a) U_k is a singleton set of a vertex u on $C_o(G_k)$ such that $d(u, G_k) \geq 2$ and $d(u, \overline{G_{k-1}}) \geq 2$ (see Fig. 5.20(a)).
 - (b) U_k is a set of two or more consecutive vertices on $C_o(G_k)$ such

- that $d(u, G_k) = 2$ and $d(u, \overline{G_{k-1}}) \geq 3$ for each vertex $u \in U_k$ (see Fig. 5.20(b)).
- (c) U_k is a set of two or more consecutive vertices on $C_o(G_k)$ such that $d(u, G_k) \geq 3$ and $d(u, \overline{G_{k-1}}) = 2$ for each vertex $u \in U_k$ (see Fig. 5.20(c)).

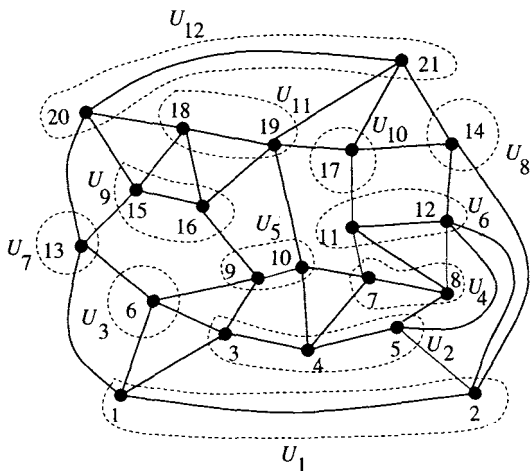


Fig. 5.19 A 4-canonical decomposition of a 4-connected plane graph having $n = 21$ vertices.

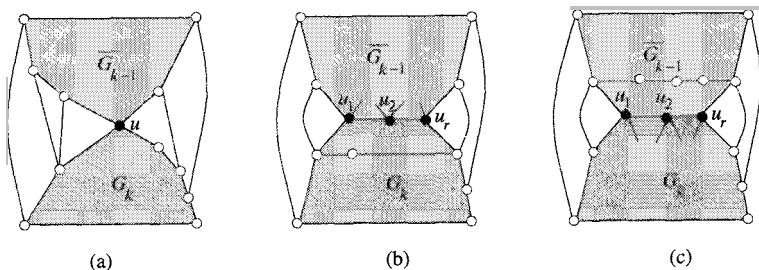


Fig. 5.20 Illustration for the three conditions (a)–(c) of (co3).

Similarly as Lemma 5.4.1 one can prove the following lemma on a 4-canonical decomposition [MNN00, NRN97].

Lemma 5.5.1 *Let G be a 4-connected plane graph having at least four outer vertices. Then G has a 4-canonical decomposition Π , and Π can be*

TEAM LING - LIVE, INFORMATIVE, NON-COST AND GENUINE !

found in linear time.

By the condition (c3), one may assume that all the vertices in $U_k, 1 \leq k \leq l$, consecutively appear clockwise on $C_o(G_k)$. We number all vertices of G by $1, 2, \dots, n$ so that they appear in U_1, U_2, \dots, U_l in this order, as illustrated in Fig. 5.19. We call each vertex in G by the number $i, 1 \leq i \leq n$. Thus one can define an order $<$ among the vertices in G .

5.5.2 Algorithm

In this section we describe an algorithm **4-Convex-Draw** which finds a convex drawing of a 4-connected plane graph G having four or more outer vertices in a grid with $W + H \leq n - 1$ [MNN00].

We first define some terms which are used in the algorithm. A *lower neighbor* of a vertex u is a neighbor of u which is smaller than u . An *upper neighbor* of u is a neighbor of u which is larger than u . Every upper neighbor v of u satisfies $y(v) \geq y(u)$ in the drawing. The number of lower neighbors of u is denoted by $d_{low}(u, G)$, and the number of upper neighbors of u is denoted by $d_{up}(u, G)$. Every vertex u except vertex 1 satisfies $d_{low}(u, G) \geq 1$, and every vertex u except vertex n satisfies $d_{up}(u, G) \geq 1$. For a vertex $u, 1 \leq u \leq n - 1$, we denote by $w^*(u)$ the largest neighbor of u . The *in-degree* of a vertex u in a directed graph D is denoted by $d_{in}(u, D)$, while the *out-degree* of u is denoted by $d_{out}(u, D)$.

We are now ready to present the algorithm. The algorithm decides only the integer coordinates of the vertices $1, 2, \dots, n$ of G effectively in this order. One can immediately find a (straight line) grid drawing of G from the coordinates. We first show how to compute the x -coordinates of all vertices, and then show how to compute the y -coordinates.

5.5.2.1 How to Compute x -Coordinates

We first show how to compute the x -coordinates of vertices. The algorithm puts vertices on the same vertical grid line as many as possible to reduce the width W of a drawing. Suppose that vertex i has been put on a grid point. (See Fig. 5.21.) If possible, the algorithm puts an upper neighbor j of i on the same vertical grid line as i , that is, it decides $x(j) = x(i)$ and hence $y(j) > y(i)$ of course. The algorithm tries to choose as j the largest neighbor $w^*(i)$ of i (this is crucial for making every face a convex polygon). However, it is impossible for a case where $w^*(i)$ has been already put on the same vertical grid line as a vertex $i' (< i)$, which was put on a grid

point before i , that is, $w^*(i') = w^*(i)$. Thus, if there exist upper neighbors of i which have not been put on the same vertical grid line as any vertex $i' (< i)$, then the algorithm puts the largest one j among them on the same vertical grid line as i . If there do not exist such an upper neighbor of i , then the algorithm does not put any vertex ($> i$) on the same vertical grid line as i . In this way, the following procedure **Construct- F** constructs a directed forest $F = (V, E_F)$. All vertices in each component of F have the same x -coordinate; if there is a directed edge (i, j) in F , then $x(j) = x(i)$ and $y(j) > y(i)$.

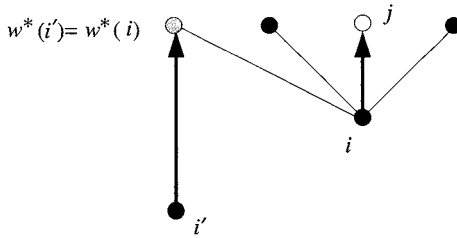


Fig. 5.21 Selection of an i 's upper neighbor j above i .

Procedure Construct- F

begin $\{F = (V, E_F)\}$

- 1 $E_F := \emptyset$; {the initial forest $F = (V, \emptyset)$ consists of isolated vertices}
- 2 **for** $i := 1$ **to** n **do**
 - 3 **if** vertex i has upper neighbors j such that $d_{in}(j, F) = 0$ **then**
 - Let j be the largest one among them, and add a directed edge (i, j) to the directed graph F , that is, $E_F := E_F \cup \{(i, j)\}$;
- end.**

Since $d_{in}(i, F), d_{out}(i, F) \leq 1$ for each vertex i , $1 \leq i \leq n$, F is a forest and each component of F is a directed path. Clearly $d_{in}(1, F) = d_{in}(2, F) = 0$ and $d_{out}(n-1, F) = d_{out}(n, F) = 0$. Figure 5.22(b) illustrates the directed forest F of the graph G in Fig. 5.22(a). Both the path 1, 13, 20 going clockwise on $C_o(G)$ from 1 to $n-1 = 20$ and the path 2, 14, 21 going counterclockwise on $C_o(G)$ from 2 to $n = 21$ are directed paths in F , and hence these two paths are put on vertical grid lines as depicted in Fig. 5.23(g). Each of the other paths in F is put on a vertical grid line, too.

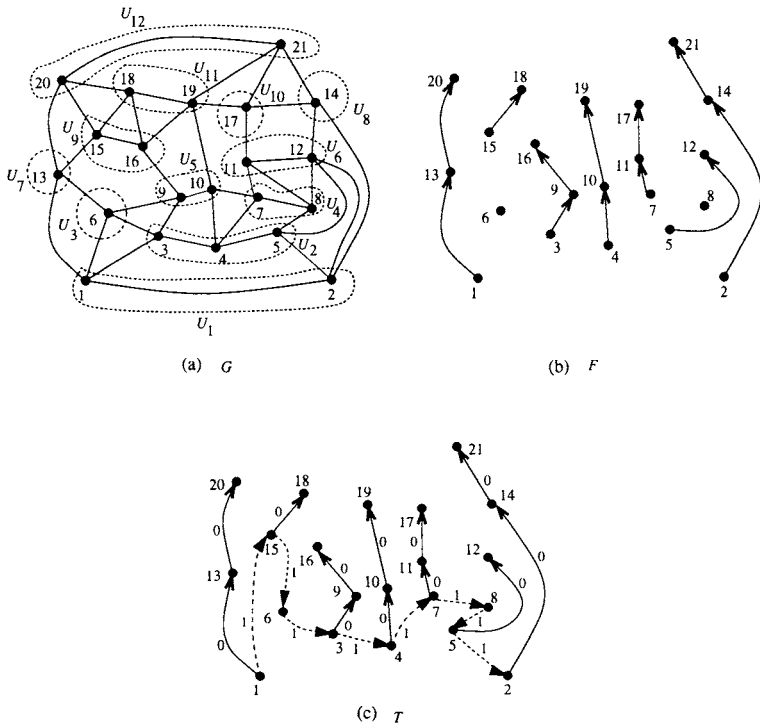


Fig. 5.22 (a) Plane graph G , (b) forest F , and (c) tree T .

We then show how to arrange the paths in F from left to right. The algorithm decides a total order among all starting vertices of paths in F . For this purpose, using the following procedure **Total-Order**, the algorithm finds a directed path P going from vertex 1 to vertex 2 passing through all starting vertices of F . In Fig. 5.22(c), the directed path P is drawn by dotted lines.

Procedure Total-Order

begin

- 1 Let P be the path directly going from vertex 1 to vertex 2;
- 2 **for** $i := 3$ **to** n **do**
 - if** $d_{in}(i, F) = 0$ **then** $\{i$ is a starting vertex of a path in $F\}$
 - begin**
 - 3 Let j be the first lower neighbor of i in the i 's adjacency list in which the i 's neighbors appear counterclockwise around i ,

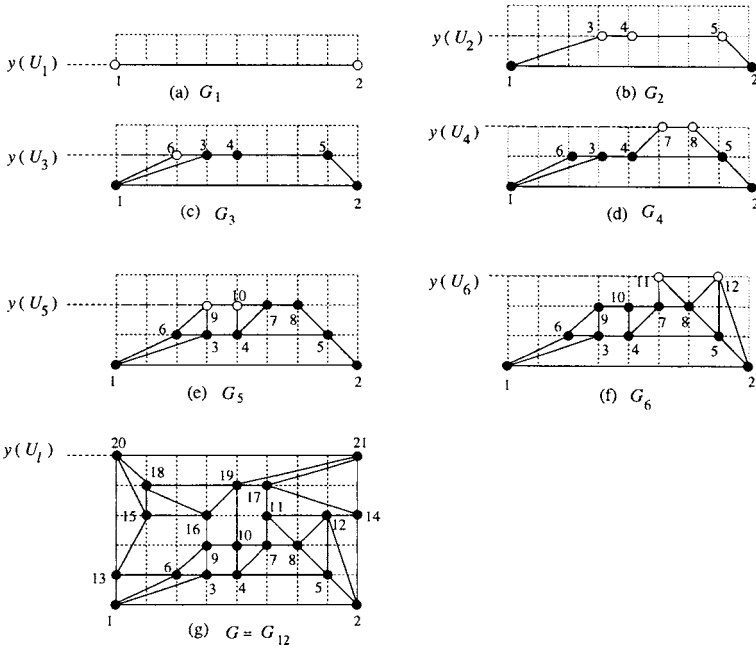


Fig. 5.23 Some of the drawing processes.

- and the first element of which is $w^*(i)$; {See Fig. 5.24(a).}
- 4 Let j' be the starting vertex of the path in F containing vertex j ; $\{2 \neq j' < i.\}$
- 5 Let k be the successor of j' in path P ;
 {The path starting from vertex k in F has been put next to the right of the path starting from vertex j' as illustrated in Fig. 5.24(a).}
- 6 Insert i in P between j' and k ;
 {The path starting from i in F is put between the path starting from j' and the path starting from k as illustrated in Fig. 5.24(b).}
- end
- end.

The algorithm constructs a weighted tree T rooted at vertex 1 by adding the path P to the forest F ; every edge of F has weight 0, and every edge of P has weight 1 in T , as illustrated in Fig. 5.22(c). Then the x -coordinate

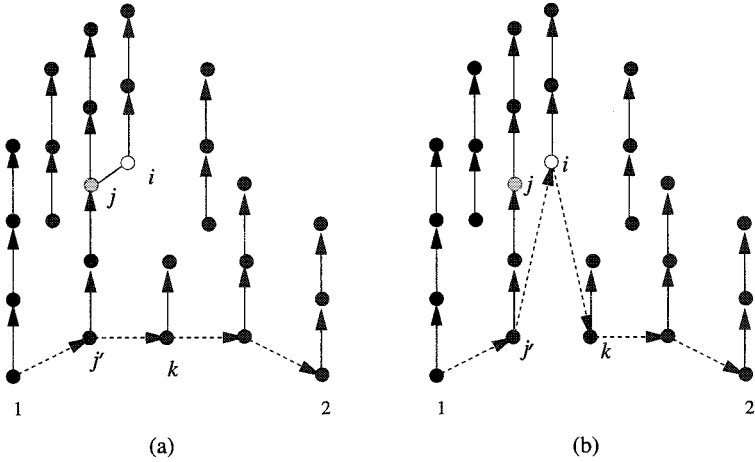


Fig. 5.24 Illustration of Total-Order.

$x(i)$ of each vertex i , $1 \leq i \leq n$, is equal to the length of the path from root 1 to i in T . Thus $x(1) = 0$, and the width $W = x(2)$ of the drawing is equal to the number of paths in F except one starting from vertex 1.

5.5.2.2 How to Compute y -Coordinates

We now show how to compute y -coordinates. For each index k , $1 \leq k \leq l$, y -coordinates of all vertices in $U_k = \{u_1, u_2, \dots, u_r\}$ are decided as the same integer, which is denoted by $y(U_k)$. Thus the path u_1, u_2, \dots, u_r on $C_o(G_k)$ is drawn as a horizontal line segment connecting points $(x(u_1), y(U_k))$ and $(x(u_r), y(U_k))$. (See Fig. 5.23.) Furthermore, the algorithm decides the y -coordinates $y(U_1), y(U_2), \dots, y(U_l)$ in this order.

The algorithm first decides the y -coordinate $y(U_1)$ of $U_1 = \{1, 2\}$ as $y(U_1) = 0$. Thus it draws $G_1 = K_2$ as a horizontal line segment connecting points $(x(1), 0)$ and $(x(2), 0)$, as illustrated in Fig. 5.23(a). Since $y(U_1) = 0$, $H = y(U_l)$.

Suppose that $y(U_1), y(U_2), \dots, y(U_{k-1})$, $k \geq 2$, have already been decided, that is, G_{k-1} has already been drawn, and one is now going to decide $y(U_k)$ and obtain a drawing of G_k by adding the vertices in U_k to the drawing of G_{k-1} . Let $C_o(G_{k-1}) = w_1, w_2, \dots, w_t$, where $w_1 = 1$ and $w_t = 2$. Let $C_o(G_k) = w_1, w_2, \dots, w_p, u_1, u_2, \dots, u_r, w_q, \dots, w_t$, where $1 \leq p < q \leq t$. Let y_{max} be the maximum value of y -coordinates of vertices w_p, w_{p+1}, \dots, w_q ; all these vertices were on $C_o(G_{k-1})$, but they are not

on $C_o(G_k)$ except w_p and w_q . (See Fig. 5.25.) Clearly one must decide $y(U_k) \geq y_{max}$ to obtain a plane drawing of G_k . The algorithm decides $y(U_k)$ to be either y_{max} or $y_{max} + 1$ so that the height H of the drawing becomes as small as possible. There are the following six cases.

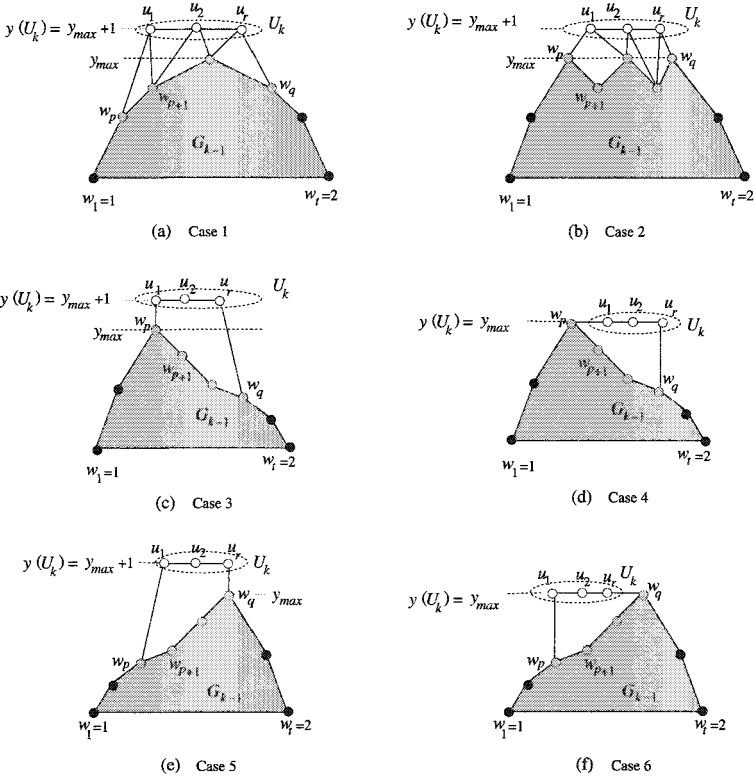


Fig. 5.25 Illustration for the six cases.

Case 1: $y_{max} > y(w_p), y(w_q)$. (See Fig. 5.25(a).)

In this case, if one decided $y(U_k) = y_{max}$, then G_k could not be a plane drawing. Therefore the algorithm decides $y(U_k) = y_{max} + 1$.

Case 2: $y_{max} = y(w_p) = y(w_q)$. (See Fig. 5.25(b).)

In this case, if one decided $y(U_k) = y_{max}$, then G_k might not be a plane drawing. Therefore the algorithm decides $y(U_k) = y_{max} + 1$.

Case 3: $y_{max} = y(w_p) > y(w_q)$, and F has a directed edge (w_p, u_1) , that is, $x(w_p) = x(u_1)$. (See Fig. 5.25(c).)

In this case, if one decided $y(U_k) = y_{max}$, then vertices w_p and u_1 would

overlap each other. Therefore the algorithm decides $y(U_k) = y_{max} + 1$.

Case 4: $y_{max} = y(w_p) > y(w_q)$, and F does not have a directed edge (w_p, u_1) , that is, $x(w_p) < x(u_1)$. (See Fig. 5.25(d).)

In this case, the algorithm decides $y(U_k) = y_{max}$.

Case 5: $y_{max} = y(w_q) > y(w_p)$, and F has a directed edge (w_q, u_r) , that is, $x(w_q) = x(u_r)$. (See Fig. 5.25(e).)

In this case, if one decided $y(U_k) = y_{max}$, then vertices w_q and u_r would overlap each other. Therefore the algorithm decides $y(U_k) = y_{max} + 1$.

Case 6: $y_{max} = y(w_q) > y(w_p)$, and F does not have a directed edge (w_q, u_r) , that is, $x(u_r) < x(w_q)$. (See Fig. 5.25(f).)

In this case, the algorithm decides $y(U_k) = y_{max}$.

We now have the following theorem on time complexity of the algorithm.

Theorem 5.5.2 *Algorithm 4-Convex-Draw takes linear time.*

Proof. By Lemma 5.5.1, a 4-canonical decomposition can be found in linear time. Clearly the forest F and the rooted tree T can be found in linear time, and the x -coordinates of vertices can be found from T in linear time. Furthermore, the y -coordinates can be found in linear time as above. Thus Algorithm 4-Convex-Draw takes linear time. \square

We next show the correctness of the algorithm. The algorithm finds the drawing of $G_1, G_2, \dots, G_l (= G)$ in this order, as illustrated in Fig. 5.23. Thus, assuming that the drawing of G_{k-1} , $k \geq 2$, is internally convex, we shall show that the drawing of G_k is internally convex. However, it is difficult to show that the drawing of G_k is internally convex for the case where either $k = l$ or U_k , $2 \leq k \leq l - 1$, satisfies the condition (c3)(c). Therefore, subdividing all such sets U_k , we obtain a refined partition Π' of V from $\Pi = (U_1, U_2, \dots, U_l)$ as follows. For each $U_k = \{u_1, u_2, \dots, u_{r_k}\}$ such that either $k = l$ or U_k satisfies the condition (c3)(c), replace U_k in Π with singleton sets $\{u_1\}, \{u_2\}, \dots, \{u_{r_k}\}$. We call the resulting partition $\Pi' = (U_1, U_2^1, U_2^2, \dots, U_2^{r_2}, U_3^1, U_3^2, \dots, U_3^{r_3}, \dots, U_l^1, U_l^2)$ of V a *refined decomposition* of G . If either $k = l$ or U_k satisfies the condition (c3)(c), then $U_k = U_k^1 \cup U_k^2 \cup \dots \cup U_k^{r_k}$, $r_k = |U_k|$ and $|U_k^i| = 1$ for each i , $1 \leq i \leq r_k$. Otherwise, $r_k = 1$ and $U_k = U_k^1$.

For each k , $2 \leq k \leq l$, and for each i , $1 \leq i \leq r_k$, we

denote by G_k^i the plane subgraph of G induced by the vertices in $U_1 \cup U_2 \cup \dots \cup U_{k-1} \cup U_k^1 \cup U_k^2 \cup \dots \cup U_k^i$. Moreover, for each k , $2 \leq k \leq l$, and for each i , $0 \leq i \leq r_k - 1$, we denote by \overline{G}_k^i the plane subgraph of G induced by the vertices in $U_k^{i+1} \cup U_k^{i+2} \cup \dots \cup U_k^{r_k} \cup U_{k+1} \cup \dots \cup U_l$. For notational convenience, let $G_k^0 = G_{k-1}$ and $\overline{G}_k^{r_k} = \overline{G}_k$.

Let $k \geq 2$ and $U_k^i = \{u_1, u_2, \dots, u_h\}$. By the definition of a refined decomposition, vertices u_1, u_2, \dots, u_h consecutively appear clockwise on $C_o(G_k^i)$ in this order, as illustrated in Fig. 5.26. Let $C_o(G_k^{i-1}) = w_1, w_2, \dots, w_t$, where $w_1 = 1$ and $w_t = 2$. Let $C_o(G_k^i) = w_1, w_2, \dots, w_p, u_1, u_2, \dots, u_h, w_q, \dots, w_t$, where $1 \leq p < q \leq t$.

The following lemma holds for the drawing of G_k^i [MNN00].

Lemma 5.5.3 For each k , $2 \leq k \leq l$, and each i , $0 \leq i \leq r_k$, the following (i)–(iii) hold:

- (i) the path going clockwise on $C_o(G_k^{i-1})$ from vertex $w_1 = 1$ to vertex $w_t = 2$ is x -monotone, that is, $x(w_1) \leq x(w_2) \leq \dots \leq x(w_t)$ (such a path is drawn by thick solid lines in Fig. 5.26);
- (ii) the path going clockwise on $C_o(G_k^{i-1})$ from w_p to w_q is “unipole” that is, there is no index s such that $p < s < q$ and $y(w_{s-1}) < y(w_s) > y(w_{s+1})$, and $w_p, w_{p+1}, \dots, w_q, w_p$ is a convex polygon in particular if U_k satisfies the condition (c3)(b) (as illustrated in Fig. 5.26(a)); and
- (iii) if a vertex v on $C_o(G_k^{i-1})$ is an inner vertex of G , and the interior angle of the polygon $C_o(G_k^{i-1})$ at vertex v is less than 180° , then v has at least one neighbor in \overline{G}_k^{i-1} .

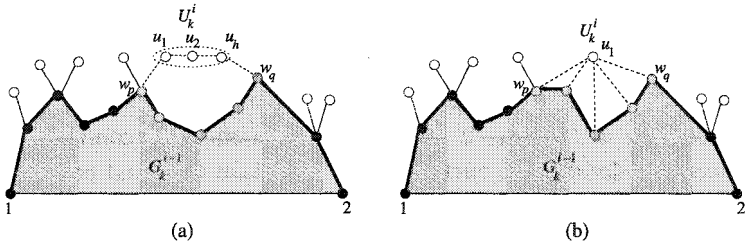


Fig. 5.26 Illustration for Lemma 5.5.3: (a) $|U_k^i| \geq 2$, and (b) $|U_k^i| = 1$.

Using Lemma 5.5.3, one can prove that the algorithm obtains a convex grid drawing [MNN00]. Note that all inner faces newly formed in G_k^i are convex polygons as illustrated in Fig. 5.26 (all such faces are not shaded

in Fig. 5.26). Furthermore the algorithm obtains a drawing in a grid with $W + H \leq n - 1$, because either W or H is increased by one in effect when each vertex is drawn except for vertex 1 [MNN00]. Thus we have the following theorem.

Theorem 5.5.4 *Algorithm 4-Convex-Draw finds a convex drawing of a 4-connected plane graph G on an integer grid such that $W + H \leq n - 1$ if G has four or more outer vertices.*

5.6 Bibliographic Notes

Using the algorithm in Section 5.2, Chiba *et al.* presented a linear algorithm for producing an aesthetic drawing of any 2-connected plane graph that makes the resulting drawing “as convex as possible” in some sense [CON85]. Schnyder and Trotter [ST92] claimed that a convex drawing of a 3-connected plane graph can be obtained by using an extension of the concept of a realizer for 3-connected plane graphs. However, no proof was published until Di Battista *et al.* [DTV99] presented such an algorithm. Felsner [Fel01] gave an algorithm to find a convex drawing of a 3-connected plane graph having f faces on an $(f - 1) \times (f - 1)$ grid.

Exercises

1. Using Corollary 5.3.5, show that every plane graph has a straight line drawing.
2. Devise an $O(n)$ time algorithm to examine whether a planar graph has a convex drawing or not.
3. Show how to find all extendible facial cycles of a planar graph.
4. Modify Algorithm **Convex-Grid-Drawing** in Section 5.4.2 to produce a convex grid drawing of a 3-connected plane graph on an $(n - 2) \times (n - 2)$ grid.
5. Assume that each vertex of a plane graph G has degree three or more, and that G has a convex drawing, that is, G satisfies Condition II although G is not always 3-connected. Show that G has an internally convex grid drawing on an $(n - 1) \times (n - 2)$ grid.
6. Prove the following claim on Algorithm **Convex-Grid-Drawing**.

Let $1 \leq k \leq l$, and let $C_o(G_k) = w_1, w_2, \dots, w_t$, where $w_1 = v_1$ and $w_t = v_2$. Then in the drawing of G_k , $P(v_1) = (0, 0)$, $P(v_2) = (|\cup_{i=1}^k U_k| - 1, 0)$,

and any line segment $w_i w_{i+1}$ has slope in $\{-45^\circ, 0^\circ\} \cup [45^\circ, 90^\circ]$.

7. Prove Lemma 5.5.1.
8. Obtain a necessary and sufficient condition for a plane graph to have a canonical ordering [MAN04].
9. Extend the concept of a realizer and a Schnyder labeling to those for plane graphs which are not always triangulated. Prove that a plane graph G has these extended realizer and Schnyder labeling if and only if G satisfies the condition in Exercise 8 [MAN04].

Chapter 6

Rectangular Drawing

6.1 Introduction

A *rectangular drawing* of a plane graph G is a drawing of G in which each vertex is drawn as a point, each edge is drawn as a horizontal or vertical line segment without edge-crossings, and each face is drawn as a rectangle. Thus a rectangular drawing is a special case of a convex drawing. Figure 6.1(b) illustrates a rectangular drawing of the plane graph in Fig. 6.1(a). In Section 1.5.1 we have seen applications of a rectangular drawing to VLSI floorplanning and architectural floorplanning. In a rectangular drawing of G , the outer cycle $C_o(G)$ is drawn as a rectangle and hence has four convex corners such as a, b, c and d drawn by white circles in Fig. 6.1. Such a convex corner is an outer vertex of degree two and is called a *corner of the rectangular drawing*. Not every plane graph G has a rectangular drawing. Of course, G must be 2-connected and the maximum degree Δ of G is at most four if G has a rectangular drawing.

Miura *et al.* recently showed that a plane graph G with $\Delta \leq 4$ has rectangular drawing D if and only if a new bipartite graph constructed from G has a perfect matching, and D can be found in time $O(n^{1.5})$ whenever G has D [MHN04]. In Section 6.2 we present their result on rectangular drawings of plane graphs with $\Delta \leq 4$.

Since a planar graph with $\Delta \leq 3$ often appears in many practical applications, we devote most of this chapter on rectangular drawings of planar graphs with $\Delta \leq 3$. In Section 6.3 we present a necessary and sufficient condition for a plane graph G with $\Delta \leq 3$ to have a rectangular drawing when four outer vertices of degree two are designated as the corners [Tho84]. We also present a linear-time algorithm to obtain a rectangular drawing with the designated corners [RNN98].

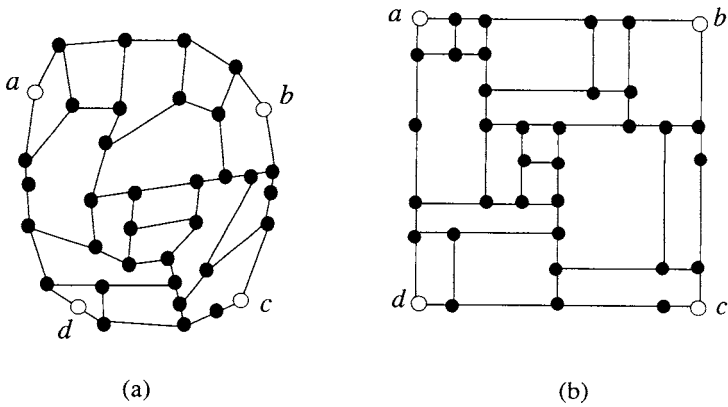


Fig. 6.1 (a) Plane graph, and (b) its rectangular drawing for the designated corners a, b, c and d .

The problem of examining whether a plane graph has a rectangular drawing becomes difficult when four outer vertices are not designated as the corners. In Section 6.4 we present a necessary and sufficient condition for a plane graph with $\Delta \leq 3$ to have a rectangular drawing for some quadruplet of outer vertices appropriately chosen as the corners, and present a key idea, behind a linear time algorithm to find such a quadruplet, together with an example [RNN02].

A planar graph may have many embeddings. We say that a *planar graph* G has a *rectangular drawing* if at least one of the plane embeddings of G has a rectangular drawing. Since a planar graph may have an exponential number of embeddings, it is not a trivial problem to examine whether a planar graph has a rectangular drawing or not. In Section 6.5 we give a linear-time algorithm to examine whether a planar graph G with $\Delta \leq 3$ has a rectangular drawing or not, and find a rectangular drawing of G if it exists [RNG04].

6.2 Rectangular Drawing and Matching

In this section we consider rectangular drawings of plane graphs with $\Delta \leq 4$. We show that a plane graph G with $\Delta \leq 4$ has rectangular drawing D if and only if a new bipartite graph G_d constructed from G has a perfect matching, and D can be found in time $O(n^{1.5})$ if D exists [MHN04]. G_d is

called a decision graph.

We may assume without loss of generality that G is 2-connected and $\Delta \leq 4$, and hence every vertex of G has degree two, three or four.

An angle formed by two edges e and e' incident to a vertex v in G is called an *angle of v* if e and e' appear consecutively around v . An angle of a vertex in G is called an *angle of G* . An angle formed by two consecutive edges on a boundary of a face F in G is called an *angle of F* . An angle of the outer face is called an *outer angle of G* , while an angle of an inner face is called an *inner angle*.

In any rectangular drawing, every inner angle is 90° or 180° , and every outer angle is 180° or 270° . Consider a labeling Θ which assigns a label 1, 2, or 3 to every angle of G , as illustrated in Fig. 6.2(c). Labels 1, 2 and 3 correspond to angles 90° , 180° and 270° , respectively. Therefore each inner angle has label either 1 or 2, exactly four outer angles have label 3, and all other outer angles have label 2.

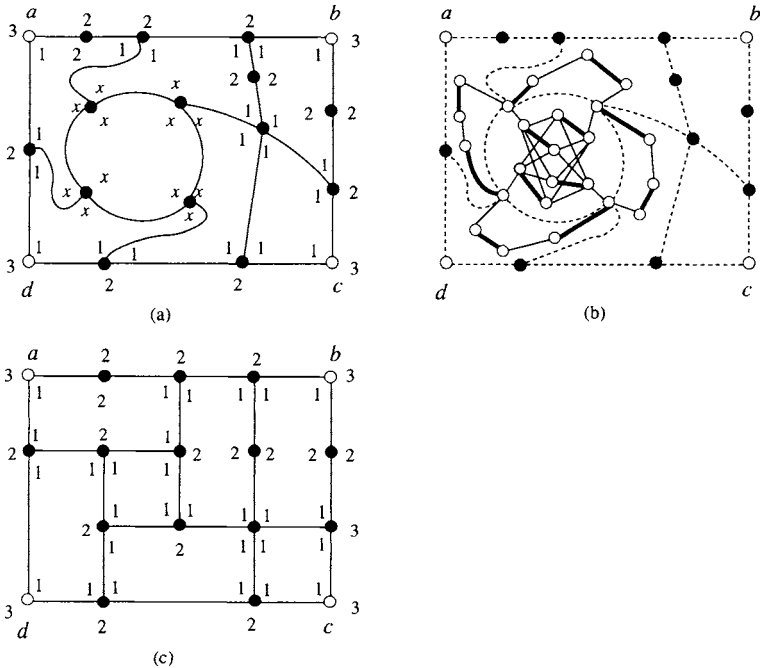


Fig. 6.2 (a) Plane graph G , (b) decision graph G_d , and (c) rectangular drawing D and regular labeling Θ of G .

We call Θ a *regular labeling* of G if Θ satisfies the following three conditions (a)–(c):

- (a) For each vertex v of G , the sum of the labels of all the angles of v is equal to 4;
- (b) The label of any inner angle is 1 or 2, and every inner face has exactly four angles of label 1; and
- (c) The label of any outer angle is 2 or 3, and the outer face has exactly four angles of label 3;

A regular labeling Θ of the plane graph in Fig. 6.2(a) and a rectangular drawing D corresponding to Θ are depicted in Fig. 6.2(c). A regular labeling is a special case of an orthogonal representation of an orthogonal drawing in Section 8.2.1.

Conditions (a) and (b) implies the following (i)–(iii):

- (i) If a non-corner vertex v has degree two, that is, $d(v) = 2$, then the two labels of v are 2 and 2.
- (ii) If $d(v) = 3$, then exactly one of the three angles of v has label 2 and the other two have label 1.
- (iii) If $d(v) = 4$, then all the four angles of v have label 1.

If G has a rectangular drawing, then clearly G has a regular labeling. Conversely, if G has a regular labeling, then G has a rectangular drawing, as can be proved by means of elementary geometric considerations. We thus have the following fact.

Fact 6.2.1 *A plane graph G has a rectangular drawing if and only if G has a regular labeling.*

We now assume that four outer vertices a, b, c and d of degree two are designated as corners. Then the outer angles of a, b, c and d must be labeled with 3, and all the other outer angles of G must be labeled with 2, as illustrated in Fig. 6.2(a). Some of the inner angles of G can be immediately determined, as illustrated in Fig. 6.2(a). If v is a non-corner outer vertex of degree two, then the inner angle of v must be labeled with 2. The two angles of any inner vertex of degree two must be labeled with 2. If v is an outer vertex of degree three, then the outer angle of v must be labeled with 2 and both of the inner angles of v must be labeled with 1. We label all the three angles of an inner vertex of degree three with x , because one cannot determine their labels although exactly one of them must be labeled with 2

and the others with 1. We label all the four angles of each vertex of degree four with 1. Label x means that x is either 1 or 2, and exactly one of the three labels x 's attached to the same vertex must be 2 and the other two must be 1. (See Figs. 6.2(a) and (c).)

We now present how to construct a decision graph G_d of G . Let all vertices of G attached a label x be vertices of G_d . Thus all the inner vertices of degree three are vertices of G_d , and none of the other vertices of G is a vertex of G_d . We then add to G_d a complete bipartite graph inside each inner face F of G , as illustrated in Fig. 6.3 where G_d is drawn by solid lines and G by dotted lines. Let n_x be the number of angles of F labeled with x . For example, $n_x = 3$ in Fig. 6.3. Let n_1 be the number of angles of F which have been labeled with 1. Then n_1 is the number of vertices v on F such that either v is a corner vertex or $d(v) = 4$. Thus $n_1 = 2$ for the example in Fig. 6.3. One may assume as a trivial necessary condition that $n_1 \leq 4$; otherwise, G has no rectangular drawing. Exactly $4 - n_1$ of the n_x angles of F labeled with x must be labeled with 1 by a regular labeling. We add a complete bipartite graph $K_{(4-n_1), n_x}$ in F , and join each of the n_x vertices in the second partite set with one of the n_x vertices on F whose angles are labeled with x . Repeat the operation above for each inner face F of G . The resulting graph is a *decision graph* G_d of G . The decision graph G_d of the plane graph G in Fig. 6.2(a) is drawn by solid lines in Fig. 6.2(b), where G is drawn by dotted lines. The idea of adding a complete bipartite graph originates from Tutte's transformation for finding an "*f-factor*" of a graph [Tut54].

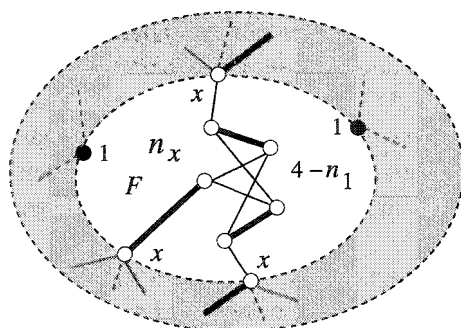


Fig. 6.3 Construction of G_d for an inner face F of G .

A *matching* of G_d is a set of pairwise non-adjacent edges in G_d . A *maximum matching* of G_d is a matching of the maximum cardinality. A

TEAM LING - LIVE, INFORMATIVE, NON-COST AND GENUINE !

matching M of G_d is called a *perfect matching* if an edge in M is incident to each vertex of G_d . A perfect matching is drawn by thick solid lines in Figs. 6.2(b) and 6.3.

Each edge e of G_d incident to a vertex v attached a label x corresponds to an angle α of v labeled with x . A fact that e is contained in a perfect matching M of G_d means that the label x of α is 2. Conversely, a fact that e is not contained in M means that the label x of α is 1.

We now have the following theorem.

Theorem 6.2.2 *Let G be a plane graph with $\Delta \leq 4$ and four outer vertices a, b, c and d be designated as corners. Then G has a rectangular drawing D with the designated corners if and only if the decision graph G_d of G has a perfect matching. D can be found in time $O(n^{1.5})$ whenever G has D .*

Proof. Suppose that G has a rectangular drawing with the designated corners a, b, c and d . Then by Fact 6.2.1 G has a regular labeling Θ by which the outer angles of a, b, c and d are labeled with 3. We include in a set M all the edges of G_d corresponding to angles of label $x = 2$, while we do not include in M the edges of G_d corresponding to angles of label $x = 1$. For each vertex v of G_d attached a label x , the labeling Θ assigns 2 to exactly one of the angles of v labeled with x . Therefore exactly one of the edges of G_d incident to v is contained in M . The labeling Θ assigns 1 to exactly four of the angles of each inner face F , and n_1 angles of F have been labeled with 1. Therefore exactly $4 - n_1$ of the n_x angles of F labeled with x must be labeled with 1 by Θ , and hence all the edges of G_d corresponding to these angles are not contained in M . Including in M a number $4 - n_1$ of edges in each complete bipartite graph, we can extend M to a perfect matching of G_d . Thus G_d has a perfect matching.

Conversely, if G_d has a perfect matching, then G has a regular labeling by which the outer angles of a, b, c and d are labeled with 3, and hence by Fact 6.2.1 G has a rectangular drawing with the designated corners a, b, c and d .

Clearly, G_d is a bipartite graph, and $4 - n_1 \leq 4$. Obviously, n_x is no more than the number of edges on face F . Let m be the number of edges in G , then we have $2m \leq 4n$ since $\Delta \leq 4$. Therefore the sum $2m$ of the numbers of edges on all faces is at most $4n$. One can thus know that both the number n_d of vertices in G_d and the number m_d of edges in G_d are $O(n)$. Since G_d is a bipartite graph, a maximum matching of G_d can be found in time $O(\sqrt{n_d} m_d) = O(n^{1.5})$ [HK73, MV80, PS82]. One can find a

regular labeling Θ of G from a perfect matching of G_d in linear time. It is easy to find a rectangular drawing of G from Θ in linear time. (Indeed a rectangular grid drawing of G can be found from Θ in linear time similarly as we will know it for a plane graph with $\Delta \leq 3$ later in Section 6.3.4.) \square

6.3 Linear Algorithm for Rectangular Drawings of Plane Graphs

In this section we present Thomassen's theorem on a necessary and sufficient condition for a plane graph G with $\Delta \leq 3$ to have a rectangular drawing when four outer vertices of degree two are designated as the corners [Tho84], and give a linear-time algorithm to find a rectangular drawing of G if it exists [RNN98].

6.3.1 Thomassen's Theorem

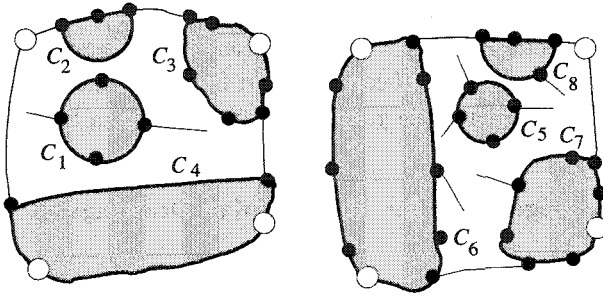
Before presenting Thomassen's theorem we recall some definitions. An edge of a plane graph G is called a *leg* of a cycle C if it is incident to exactly one vertex of C and located outside C . The vertex of C to which a leg is incident is called a *leg-vertex* of C . A cycle in G is called a *k-legged cycle* of G if C has exactly k legs in G and there is no edge which joins two vertices on C and is located outside C . Figure 6.4(a) illustrates 2-legged cycles C_1, C_2, C_3 and C_4 , while Fig. 6.4(b) illustrates 3-legged cycles C_5, C_6, C_7 and C_8 , where corners are drawn by white circles.

If a 2-legged cycle contains at most one corner like C_1, C_2 and C_3 in Fig. 6.4(a), then some inner face cannot be drawn as a rectangle and hence G has no rectangular drawing. Similarly, if a 3-legged cycle contains no corner like C_5 and C_8 in Fig. 6.4(b), then G has no rectangular drawing. One can thus observe the following fact.

Fact 6.3.1 *In any rectangular drawing D of G , every 2-legged cycle of G contains two or more corners, every 3-legged cycle of G contains one or more corner, and every cycle with four or more legs may contain no corner, as illustrated in Fig 6.5.*

The necessity of the following Thomassen's theorem is immediate from Fact 6.3.1.

Theorem 6.3.2 *Assume that G is a 2-connected plane graph with $\Delta \leq 3$ and four outer vertices of degree two are designated as the corners a, b, c*



(a) 2-legged cycles

(b) 3-legged cycles

Fig. 6.4 Good cycles C_4, C_6 and C_7 , and bad cycles C_1, C_2, C_3, C_5 and C_8 .




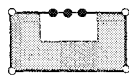




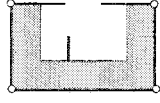
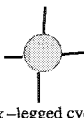
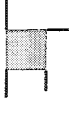
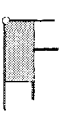
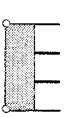

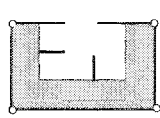
	the number of corners contained in a cycle				
	0	1	2	3	4
 2-legged cycle	none	none			
 3-legged cycle	none				
 k-legged cycle $k \geq 4$					

Fig. 6.5 Numbers of corners in drawings of cycles.

and d. Then G has a rectangular drawing if and only if

- (r1) any 2-legged cycle contains two or more corners, and
- (r2) any 3-legged cycle contains one or more corners.

A cycle of type (r1) or (r2) is called *good*. Cycles C_4, C_6 and C_7 in Fig. 6.4 are good cycles; the 2-legged cycle C_4 contains two corners, and the

3-legged cycles C_6 and C_7 contain one or two corners. On the other hand, a 2-legged or 3-legged cycle is called *bad* if it is not good. Thus 2-legged cycles C_1, C_2 and C_3 and 3-legged cycles C_5 and C_8 are bad cycles. Thus Theorem 6.3.2 can be rephrased as follows: G has a rectangular drawing if and only if G has no bad cycle. In particular, a 2-legged bad cycle is called a *bad corner* if it contains exactly one corner like C_3 .

6.3.2 Sufficiency

In this section we present a constructive proof of the sufficiency of Theorem 6.3.2 [RNN98].

The *union* $G = G' \cup G''$ of two graphs G' and G'' is a graph $G = (V(G') \cup V(G''), E(G') \cup E(G''))$.

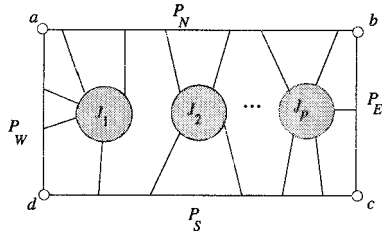
In a given 2-connected plane graph G , four outer vertices of degree two are designated as the corners a, b, c and d . These four corners divide the outer cycle $C_o(G)$ of G into four paths, the north path P_N , the east path P_E , the south path P_S , and the west path P_W , as illustrated in Fig. 6.6(a). We will draw the north and south paths on two horizontal straight line segments and the east and west paths on two vertical line segments. We thus fix the embedding of $C_o(G)$ as a rectangle. We call a rectangular embedding of $C_o(G)$ an *outer rectangle*. Clearly the following lemma holds.

Lemma 6.3.3 *Let J_1, J_2, \dots, J_p be the $C_o(G)$ -components of a plane graph G , and let $G_i = C_o(G) \cup J_i$, $1 \leq i \leq p$, as illustrated in Fig. 6.6. Then G has a rectangular drawing with corners a, b, c and d if and only if, for each index i , $1 \leq i \leq p$, G_i has a rectangular drawing with corners a, b, c and d .*

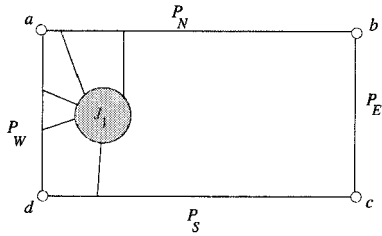
If G has a rectangular drawing, then J_1, J_2, \dots, J_p must be “in series” as illustrated in Fig. 6.6(a). If J_1, J_2, \dots, J_p are not in series as illustrated in Fig. 6.7(a), then G has no rectangular drawing; Fig. 6.7(b) illustrates G_1 which may have a rectangular drawing; Fig. 6.7(c) illustrates G_3 , which has no rectangular drawing since G_3 has a bad 2-legged cycle indicated by dotted lines.

In the remainder of this section, because of Lemma 6.3.3, we may assume that G has exactly one $C_o(G)$ -component J , as illustrated in Fig. 6.8.

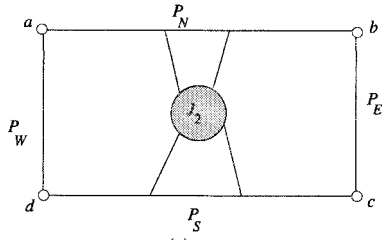
We now outline the proof. Assume that G has no bad cycle. We divide G into two subgraphs having no bad cycle by slicing G along one or two paths. For example, the graph G in Fig. 6.9(a) is divided into two subgraphs G_1 and G_2 , each having no bad cycle, by slicing G along a path drawn by thick



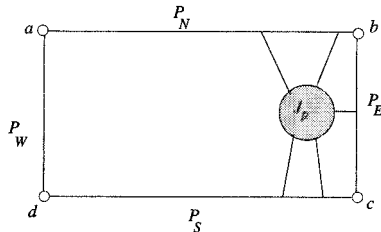
(a) G



(b) G_1



(c) G_2



(d) G_p

Fig. 6.6 (a) G , (b) G_1 , (c) G_2 , and (d) G_p .

lines, as illustrated in Fig. 6.9(b). We then recursively find rectangular drawings of the two subgraphs as illustrated in Fig. 6.9(c), and obtain a rectangular drawing of G by patching them, as illustrated in Fig. 6.9(d).

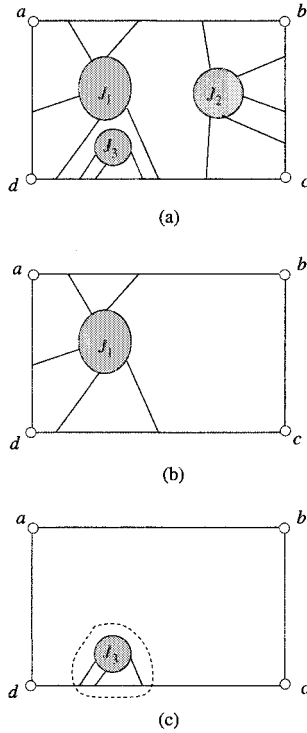


Fig. 6.7 (a) G , (b) G_1 , and (c) G_3 .

However, the problem is not so simple, because, for some graphs having no bad cycles like one in Fig. 6.10(a), there is no such path that the resulting two subgraphs have no bad cycle. For any path, one of the resulting two subgraphs has a bad 3-legged cycle C although C is not a bad cycle in G , as illustrated in Fig. 6.10(b) where a bad cycle C in a subgraph is indicated by dotted lines. For such a case, we split G into two or more subgraphs by slicing G along two paths P_c and P_{cc} having the same ends on P_N and P_S . For example, as illustrated in Fig. 6.10(c), the graph G in Fig. 6.10(a) is divided into three subgraphs G_1 , G_2 and G_3 , each having no bad cycle, by slicing G along path P_c indicated by dotted lines and path P_{cc} drawn by thick lines in Fig. 6.10(a). We then recursively find rectangular drawings of G_1 , G_2 and G_3 as illustrated in Fig. 6.10(d), and slightly deform the drawings of G_1 and G_2 , as illustrated in Fig. 6.10(e). We finally obtain a rectangular drawing of G by patching the drawings of the three subgraphs as illustrated in Fig. 6.10(f).

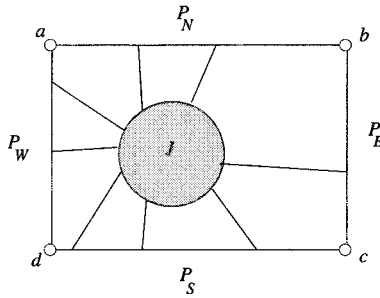


Fig. 6.8 Plane graph with exactly one $C_o(G)$ -component.

We need some definitions before presenting the detail of a constructive proof. A cycle C in G like one in Fig. 6.10(a) is called “critical,” because C is not a bad cycle in G but C would become a bad cycle in a subgraph obtained from G by splitting G along a path P . We now give a formal definition of a critical cycle. A cycle C in a plane graph G is *attached to a path* P if

- (i) P does not contain any vertex in the proper inside of C , and
- (ii) the intersection of C and P is a single subpath of P ,

as illustrated in Fig. 6.11. Let v_t be the starting vertex of the subpath, and let v_h be the ending vertex. We then call v_t the *tail vertex* of C for P , and v_h the *head vertex*. Denote by $Q_c(C)$ the path on C turning clockwise around C from v_t to v_h , and denote by $Q_{cc}(C)$ the path on C turning counterclockwise around C from v_t to v_h . A leg of C is called a *clockwise leg* for P if it is incident to a vertex in $V(Q_c(C)) - \{v_t, v_h\}$. Denote by $n_c(C)$ the number of clockwise legs of C for P . Similarly we define a *counterclockwise leg* and denote by $n_{cc}(C)$ the number of counterclockwise legs of C for P . A cycle C attached to P is called a *clockwise cycle* if $Q_{cc}(C)$ is a subpath of P , and is called a *counterclockwise cycle* if $Q_c(C)$ is a subpath of P . A cycle C is called a *critical cycle* if either C is a clockwise cycle and $n_c(C) \leq 1$ or C is a counterclockwise cycle and $n_{cc}(C) \leq 1$. Figure 6.11 illustrates a clockwise critical cycle with $n_c(C) = 1$.

We are now ready to give a constructive proof for the sufficiency of Theorem 6.3.2. Assume that G has no bad cycle. By Lemma 6.3.3, we further assume that G has exactly one $C_o(G)$ -component. Let $P_N = v_0, v_1, \dots, v_p$ where $v_0 = a$ and $v_p = b$, and let $P_S = u_0, u_1, \dots, u_q$ where $u_0 = c$ and $u_q = d$, as illustrated in Fig. 6.12(a). An *NS-path* P is defined to be a path

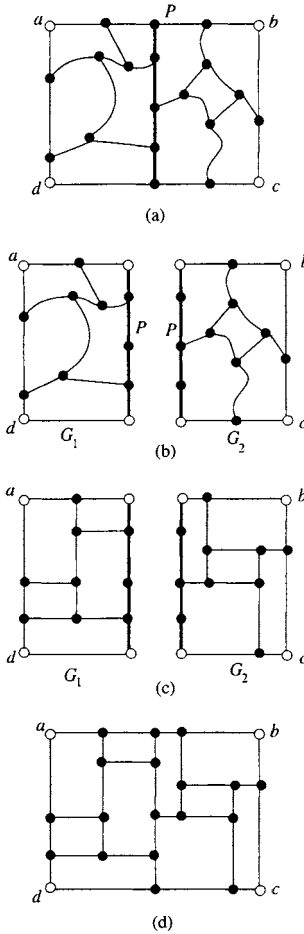


Fig. 6.9 (a) G and P , (b) G_1 and G_2 , (c) rectangular drawings of G_1 and G_2 , and (d) rectangular drawing of G .

starting at a vertex v_i on P_N and ending at a vertex u_j on P_S without passing through any outer edge and any outer vertex other than the ends v_i and u_j . An NS-path P divides graph G into two subgraphs G_W^P and G_E^P ; G_W^P is the west part of G including P and has four corners a, v_i, u_j and d , and G_E^P is the east part of G including P and has four corners v_i, b, c and u_j . G_E^P and G_W^P are illustrated in Figs. 6.12(b) and (c), respectively. We say that P is an NS-partitioning path if neither G_W^P nor G_E^P has a bad cycle. Similarly we define a WE-partitioning path. If G has a partitioning

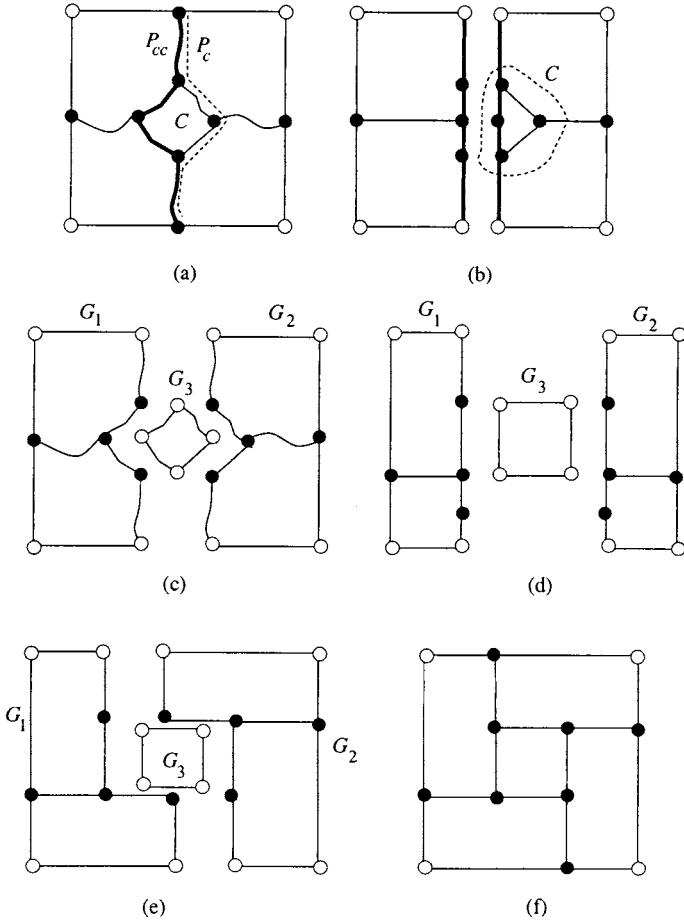


Fig. 6.10 (a) G , (b) splitting G along a single path P_{cc} , (c) splitting G along two paths P_{cc} and P_c , (d) rectangular drawings of three subgraphs, (e) deformation, and (f) rectangular drawing of G .

path, say an NS-partitioning path P , then one can obtain a rectangular drawing of G by recursively finding rectangular drawings of G_W^P and G_E^P and patching them together along P , as illustrated in Fig. 6.9.

An inner face of G is called a *boundary face* if its contour contains at least one outer edge. In Fig 6.13, F_1, F_2, \dots, F_5 are the boundary faces. A *boundary path* is a maximal path on the contour of a boundary face connecting two outer vertices without passing through any outer edge. The boundary path P on F_1 and the boundary path P' on F_3 are drawn by thick

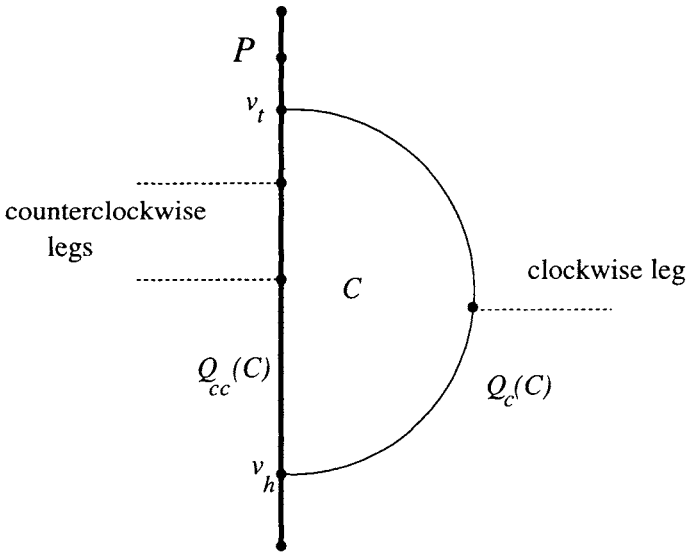


Fig. 6.11 Clockwise critical cycle C attached to path P .

lines in Fig. 6.13. Note that the direction of a boundary path is the same as the contour of the face, and hence is clockwise. For $X, Y \in \{N, E, S, W\}$, a *boundary XY -path* is a boundary path starting at a vertex on path P_X and ending at a vertex on path P_Y . In Fig. 6.13 P is a boundary NS-path, and P' is a boundary EN-path. We easily have the following lemma, whose proof is left as an exercise.

Lemma 6.3.4 *If G has no bad cycle, then every boundary NS-, SN-, EW- or WE-path P of G is a partitioning path, that is, G can be splitted along P into two subgraphs, each having no bad cycle. (See Fig. 6.13.)*

Thus we may assume that G has no boundary NS-, SN-, EW- or WE-paths. Then the $C_o(G)$ -component J has at least one vertex on each of the paths P_N, P_E, P_S and P_W , as illustrated in Fig. 6.8. In this case we find a pair of partitioning paths P_c and P_{cc} , and divide G into two or more subgraphs having no bad cycles by splitting G along P_c and P_{cc} . Both P_c and P_{cc} are NS-paths which have the same ends and do not cross each other in the plane although they may share several edges. Thus, if $P_c \neq P_{cc}$, then the edge set $E(P_c) \oplus E(P_{cc}) = E(P_c) \cup E(P_{cc}) - E(P_c) \cap E(P_{cc})$ induces vertex-disjoint cycles $C_1, C_2, \dots, C_k, k \geq 1$, as illustrated in Figs. 6.14 and 6.15 where P_c and P_{cc} are indicated by dotted lines. Thus P_c and P_{cc} share

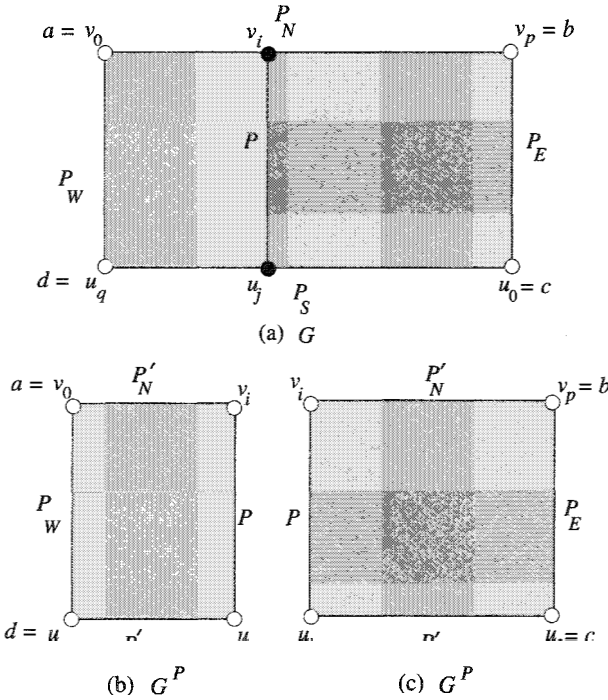


Fig. 6.12 (a) Plane graph G and NS-path P , (b) G^P_W , and (c) G^P_E

$k + 1$ maximal subpaths P_1, P_2, \dots, P_{k+1} , as illustrated in Fig. 6.14(a). We assume that P_c turns around cycles C_1, C_2, \dots, C_k clockwise, and P_{cc} turns around them counterclockwise. We choose P_c and P_{cc} so that each cycle C_i has exactly four legs; assuming clockwise order, the first one is contained in P_i , $1 \leq i \leq k$, the second one is a clockwise leg, the third one is contained in P_{i+1} and the fourth one is a counterclockwise leg; and the four leg-vertices of C_i will be designated as the corners of the subgraph $G(C_i)$ of G inside C_i . Thus G is divided into subgraphs $G^{P_{cc}}, G^{P_c}, G(C_1), G(C_2), \dots, G(C_k)$, as illustrated in Figs. 6.14(b) and 6.15(b). $G^{P_{cc}}$ has a, d and the two ends of P_{cc} as the corners, while G^{P_c} has b, c and the two ends of P_c as the corners. $G_i(C_i)$, $1 \leq i \leq k$, has the four leg-vertices of C_i as the corners. We now have the following lemma.

Lemma 6.3.5 *Assume that a cycle C in the $C_o(G)$ -component J has exactly four legs. Then the subgraph $G(C)$ of G inside C has no bad cycle when the four leg-vertices are designated as corners of $G(C)$.*

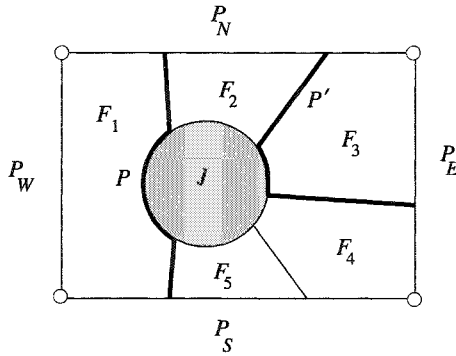


Fig. 6.13 Plane graph with a boundary NS-path P .

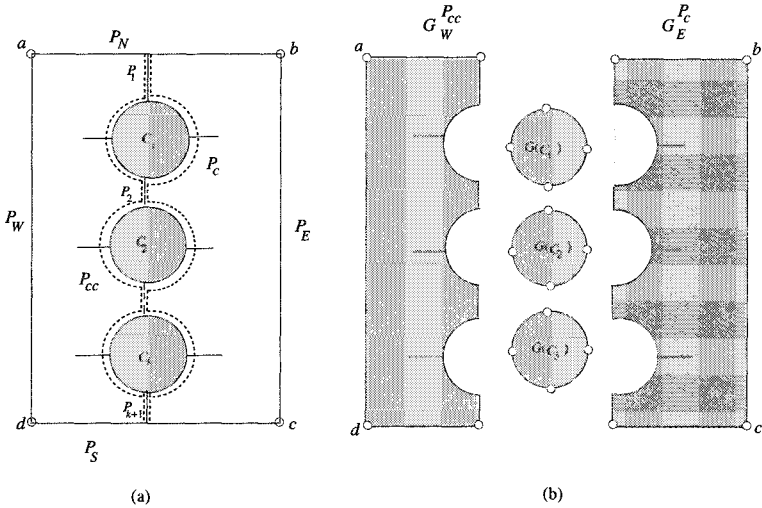


Fig. 6.14 (a) G with partition-pair P_c and P_{cc} , and (b) splitting G along P_c and P_{cc} .

Proof. If $G(C)$ has a bad cycle, then it is also a bad cycle in G , a contradiction to the assumption that G has no bad cycle. \square

By Lemma 6.3.5 we can assume that none of $G(C_1), G(C_2), \dots, G(C_k)$ has a bad cycle. For each cycle $C_i, 1 \leq i \leq k$, there are two alternative rectangular embeddings of C_i as illustrated in Fig. 6.16, where P'_N, P'_E, P'_S and P'_W are the four subpaths of C_i divided by the four leg-vertices. Therefore there are 2^k different embeddings of cycles C_1, C_2, \dots, C_k when P_c and

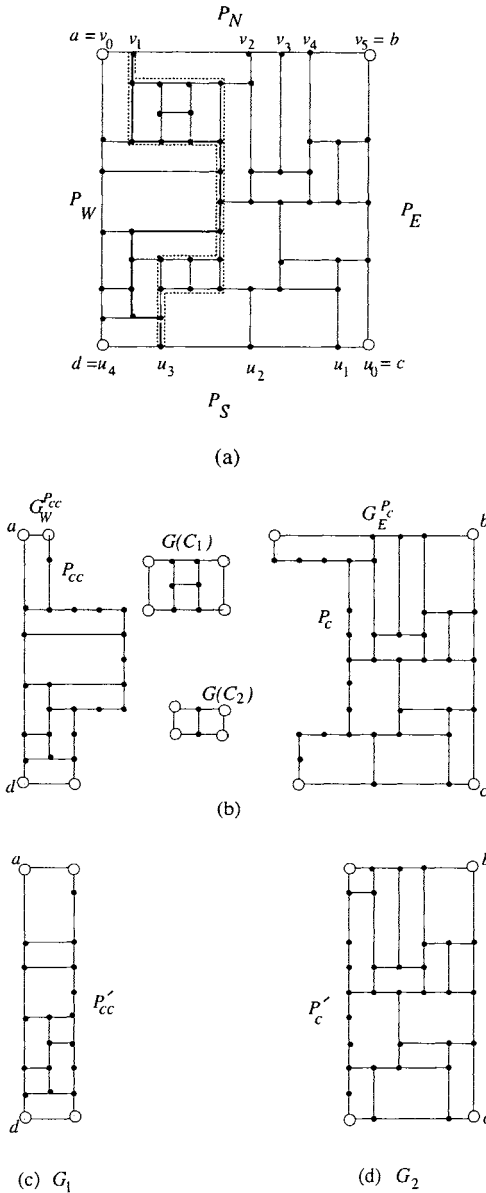


Fig. 6.15 (a) G with P_c and P_{cc} , (b) splitting G along P_c and P_{cc} , (c) drawings of G_1 , and (d) drawing of G_2 .

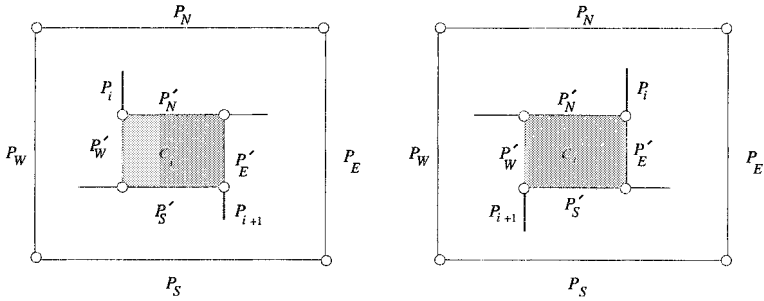


Fig. 6.16 Two alternative rectangular embeddings of cycle C_i .

P_{cc} are embedded as alternating sequences of horizontal and vertical line segments as illustrated in Fig. 6.17. We arbitrarily choose one of them. Let G_1 be the graph obtained from $G_W^{P_{cc}}$ by contracting all edges of P_{cc} that are on the horizontal sides of rectangular embeddings of C_1, C_2, \dots, C_k , as illustrated in Fig. 6.15(c). Note that every intermediate vertex on such a horizontal side has degree two in G_W^P . We denote by P'_{cc} the resulting path obtained from P_{cc} by the contraction above. Let G_1 have four corners a, d and the two ends of P_{cc} . Then one can observe that if G_1 has a rectangular drawing, in which the path P'_{cc} is drawn as a vertical straight line segment, then the rectangular drawing of G_1 can be easily modified to a drawing of $G_W^{P_{cc}}$ fitted in the area for $G_W^{P_{cc}}$ where P_{cc} is drawn as an alternating sequence of horizontal and vertical line segments, as illustrated in Figs. 6.15(b) and (c). Let G_2 be the graph obtained from $G_E^{P_c}$ by contracting all edges of P_c that are on the horizontal sides of rectangular embeddings of C_1, C_2, \dots, C_k , and let P'_c be the resulting path obtained from P_c by the contraction, as illustrated in Fig. 6.15(d). Then, if G_2 has a rectangular drawing, then it can be easily modified to a drawing of $G_E^{P_c}$ fitted in the area for $G_E^{P_c}$ where P_c is drawn as an alternating sequence of horizontal and vertical line segments, as illustrated in Figs. 6.15(b) and (d). Thus if we have drawings of graphs $G_W^{P_{cc}}, G_E^{P_c}, G(C_1), G(C_2), \dots, G(C_k)$, then we can immediately patch them to get a rectangular drawing of G . One can observe that G_1 and G_2 have no bad cycles if and only if $G_W^{P_{cc}}$ and $G_E^{P_c}$ have no bad cycles, respectively. We thus call P_c and P_{cc} a *pair of partitioning paths* or simply a *partition-pair* if neither $G_E^{P_c}$ nor $G_W^{P_{cc}}$ has a bad cycle. Especially when $P_c = P_{cc}$, it is a single partitioning path.

Thus the problem is how to prove that G has a partition-pair and to find a partition-pair efficiently. An idea is to find a partition-pair from the

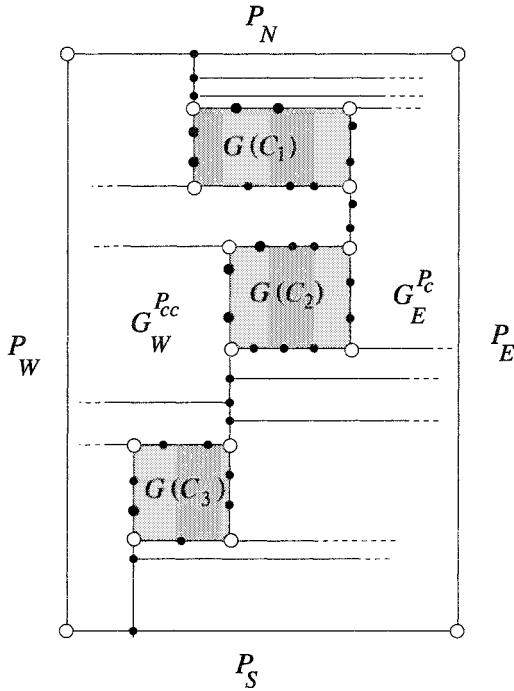


Fig. 6.17 An example of the embedding of a partition-pair.

“westmost NS-path” defined as follows. An NS-path P is *westmost* if

- (1) P starts at the second vertex v_1 of P_N ,
- (2) P ends at the second last vertex u_{q-1} of P_S , and
- (3) the number of edges in G_W^P is minimum.

The westmost NS-path is drawn in thick lines by Fig. 6.15(a). One can find the westmost NS-path by the “counterclockwise depth-first search” starting from v_1 , that is, a depth-first search where the edge counterclockwise next to the currently visited edge is searched in each step.

Let the westmost NS-path $P = w_1, w_2, \dots, w_j$, where $w_1 = v_1$ and $w_j = u_{q-1}$. There may exist critical cycles attached to P . Clearly all these cycles are clockwise attached to P , as illustrated in Fig. 6.18 where the westmost path P is drawn on a vertical line. All these cycles are “laminar,” that is, either the proper insides of any two of the cycles are disjoint or one is contained in the other since every vertex has degree two or three in G . A cycle among them is said to be *maximal* if its inside is not

contained in the inside of any of the other cycles. Figure 6.18 illustrates the hierarchical structure of the cycles, the insides of seven maximal critical cycles $C_{m1}, C_{m2}, \dots, C_{m7}$ are shaded. We now have the following lemma.

Lemma 6.3.6 *If G has no bad cycle and has no boundary NS-, SN-, EW- or WE-path, then G has a partition-pair P_c and P_{cc} .*

Proof. Assume that G has no bad cycle and has no boundary NS-, SN-, EW- or WE-path. Then we can find a partition-pair P_c and P_{cc} from the westmost NS-path P , as follows. In Fig. 6.18 P_c and P_{cc} are indicated by dotted lines.

Firstly, we find two paths P_{st} and P_{en} so that neither $G_W^{P_{cc}}$ nor $G_E^{P_c}$ has a bad corner; P_{st} is the starting subpath shared by P_c and P_{cc} , and P_{en} is the ending subpath shared by P_c and P_{cc} . Let α be the largest index among $1, 2, \dots, j$ such that vertex w_α is contained in a boundary NN- or EN-path Q . Let $Q = y_1, y_2, \dots, y_l$, where $y_1 \in V(P_N) \cup V(P_E)$ and $y_l \in V(P_N)$, and let $w_\alpha = y_i, 1 \leq i \leq l$, as illustrated in Fig. 6.18. Choose $P_{st} = y_l, y_{l-1}, \dots, y_i$. Similarly, let β be the smallest index such that vertex w_β is contained in a boundary SS- or SE-path R . Let $R = z_1, z_2, \dots, z_h$, where $z_1 \in V(P_S)$ and $z_h \in V(P_S) \cup V(P_E)$, and let $w_\beta = z_r, 1 \leq r \leq h$. Choose $P_{en} = z_r, z_{r-1}, \dots, z_1$. Then clearly $\alpha \leq \beta$; otherwise, there would exist a boundary SN-path. Furthermore neither $G_E^{P'}$ nor $G_W^{P'}$ has a bad corner for any NS-path P' whose starting subpath is P_{st} and ending subpath is P_{en} ; otherwise, a contradiction either to the selection of P_{st} and P_{en} or to the assumption that G has no bad cycle would occur.

Secondly, for each edge e on the subpath $P_{\alpha\beta}$ of P connecting w_α and w_β , if e is not contained in any maximal critical cycle C attached to P , then we let both P_c and P_{cc} pass through e , as illustrated in Fig. 6.18.

Lastly, we choose subpaths of P_c and P_{cc} for each of the clockwise maximal critical cycles C attached to the subpath $P_{\alpha\beta}$, as follows. (In Fig. 6.18 C_{m3}, C_{m4}, C_{m5} and C_{m6} are these maximal cycles.) Since $n_c(C) \leq 1$, we have the following three cases.

Case 1 : $n_c(C) = 0$.

In this case $n_{cc}(C) \geq 2$; otherwise, C would be a bad cycle in G , contrary to the assumption. We let both P_c and P_{cc} pass through $Q_c(C)$. (In Fig. 6.18 $n_c(C_{m3}) = 0$ and $n_{cc}(C_{m3}) = 2$.)

Case 2 : $n_c(C) = 1$ and $n_{cc}(C) \leq 1$.

In this case $n_{cc}(C) = 1$; if $n_{cc}(C) = 0$, then C would be a bad 3-legged cycle in G . Thus C has exactly four legs. We let path P_c pass through $Q_c(C)$ and let P_{cc} pass through $Q_{cc}(C)$. Thus C is one of the cycles induced

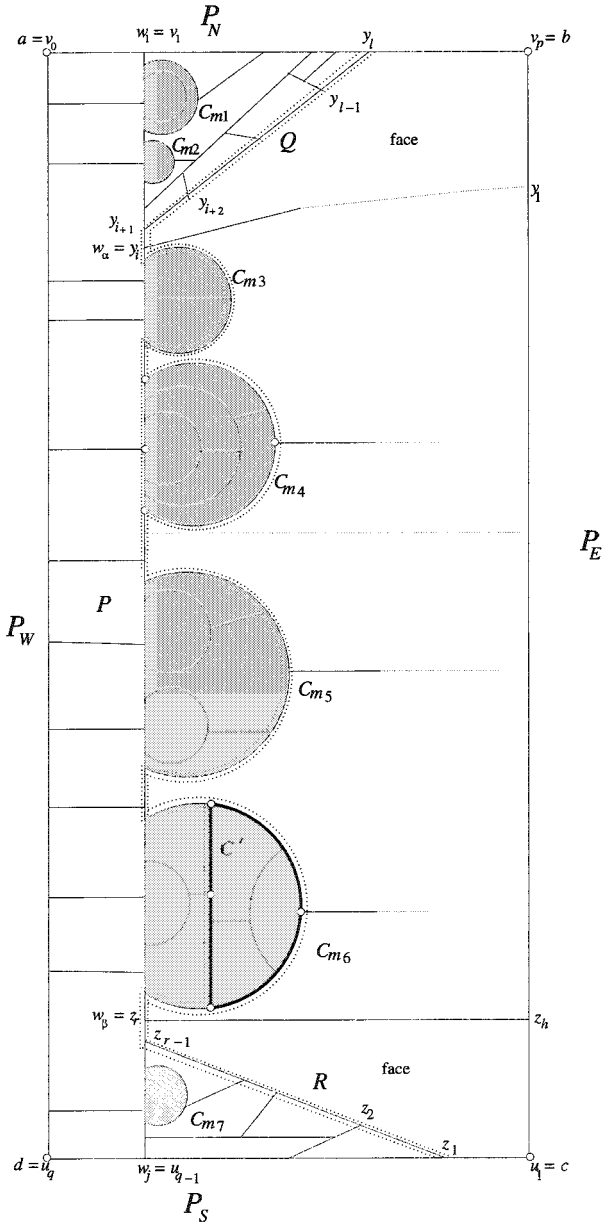


Fig. 6.18 Clockwise critical cycles attached to the westmost NS-path P .

by $E(P_c) \oplus E(P_{cc})$. (In Fig. 6.18 $n_c(C_{m4}) = 1$ and $n_{cc}(C_{m4}) = 1$.)

Case 3: $n_c(C) = 1$ and $n_{cc}(C) \geq 2$

In this case there are two subcases. In the first subcase where $G(C)$ has no counterclockwise critical cycle attached to $Q_c(C)$, we let both P_c and P_{cc} pass through $Q_c(C)$. (In Fig. 6.18 $n_c(C_{m5}) = 1$ and $n_{cc}(C_{m5}) = 2$ and $G(C_{m5})$ has no counterclockwise critical cycle attached to $Q_c(C_{m5})$.) Consider the second subcase in which $G(C)$ has one or more counterclockwise critical cycles attached to $Q_c(C)$. In this case $G(C)$ has exactly one maximal counterclockwise critical cycle C' attached to $Q_c(C)$; if $G(C)$ has two maximal counterclockwise critical cycles C' and C'' attached to $Q_c(C)$ as illustrated in Fig. 6.19, then one of them would be a bad cycle in G , contrary to the assumption. Furthermore $n_{cc}(C') = 1$; otherwise, $n_{cc}(C') = 0$ and hence C' would be a bad 3-legged cycle in G . Since $n_{cc}(C') = 1$, C' has exactly four legs, and hence we let both P_c and P_{cc} pass through edges in $E(Q_c(C)) - E(C')$, let P_c pass through $Q_c(C')$, and let P_{cc} pass through $Q_{cc}(C')$. Thus C' is one of the cycles induced by $E(P_c) \oplus E(P_{cc})$. (In Fig. 6.18 $n_c(C_{m6}) = 1, n_{cc}(C_{m6}) = 2$, $G(C_{m6})$ has exactly one maximal counterclockwise critical cycle C' attached to $Q_c(C_{m6})$ which is drawn by thick lines, and $n_{cc}(C') = 1$.)

If we choose P_c and P_{cc} as above, then clearly each of the cycles induced by $E(P_c) \oplus E(P_{cc})$ has exactly four legs, and hence by Lemma 6.3.5 the subgraph inside it has no bad cycle. Furthermore one can observe that both $G_E^{P_c}$ and $G_W^{P_{cc}}$ have no bad cycle. Therefore P_c and P_{cc} are a pair of partitioning paths. (In Figs. 6.15 and 6.18 Case 2 and the second subcase of Case 3 have occurred, and $E(P_c) \oplus E(P_{cc})$ induces two cycles.) \square

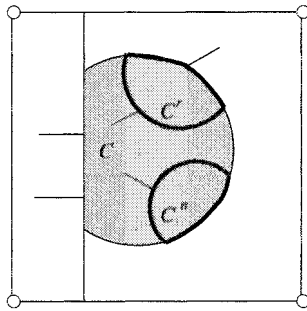


Fig. 6.19 Illustration for Case 3.

Using Lemmas 6.3.3, 6.3.4, 6.3.5 and 6.3.6, one can recursively find a

TEAM LING - LIVE, INFORMATIVE, NON-COST AND GENUINE !

rectangular drawing of a given plane graph G if G has no bad cycle. Thus we have constructively proved the sufficiency of Theorem 6.3.2.

6.3.3 Rectangular Drawing Algorithm

In this section, we assume that a given plane graph G has no bad cycle, and present an algorithm **Rectangular-Draw** to find a rectangular drawing of G . The algorithm outputs only the directions (vertical or horizontal) of edges of G . From the directions one can decide the integer coordinates of vertices as shown later in Section 6.3.4. It is easy to modify the algorithm so that it examines whether a given plane graph has a bad cycle or not.

We treat each $C_o(G)$ -component independently as in Lemma 6.3.3. If there exists a boundary NS-, SN-, WE-, or EW-path, we choose it as a partitioning path. Otherwise, we find a partition-pair P_c and P_{cc} from the westmost NS-path, and then recurse to the subgraphs divided by P_c and P_{cc} .

Algorithm Rectangular-Draw(G)

begin

- 1 Draw the outer cycle $C_o(G)$ of G as a rectangle by two horizontal line segments P_N and P_S and two vertical line segments P_E and P_W ;
 {The directions of edges on $C_o(G)$ are decided.}
 - 2 Find all $C_o(G)$ -components J_1, J_2, \dots, J_p ; {See Fig. 6.6(a).}
 - 3 **for** each component J_i **do**
 begin
 4 $G_i = C_o(G) \cup J_i$;
 { G_i is the union of graphs $C_o(G)$ and J_i .}
 5 Draw(G_i, J_i)
 end
- end.**

Procedure Draw(G, J)

{ G has exactly one $C_o(G)$ -component J .}

begin

- 1 **if** G has a boundary NS-, SN-, EW-, or WE-path P
 then
 { P is a partitioning path. See Fig. 6.13.}
 begin
 2 Assume without loss of generality that P is a boundary

```

NS-path;
3 Draw all edges of  $P$  on a vertical line;
  {The directions of edges of  $P$  are decided to be vertical.}
4 if  $|E(P)| \geq 2$  then
  begin
5     Let  $F_1, F_2, \dots, F_q$  be the  $C_o$ -components of  $G_E^P$ ;
     {See Fig. 6.20.  $G_W^P$  is a single cycle.}
6     for each component  $F_i, 1 \leq i \leq q$ , do
7         Draw( $C_o(G_E^P) \cup F_i, F_i$ )
  end
end
else { $G$  has no boundary NS-, SN-, EW-, or WE-path,
  as illustrated in Fig. 6.8}
begin
8 Find the westmost NS-path  $P$ ;
9 Find a partition-pair  $P_c$  and  $P_{cc}$  from  $P$  as in the proof
  of Lemma 6.3.6;
10 if  $P_c = P_{cc}$  then {See Fig. 6.9.}
  begin
11 Draw all edges of  $P_c$  on a vertical line segment;
12 Let  $G_1 = G_W^{P_c}$  and  $G_2 = G_E^{P_c}$  be two resulting
  subgraphs;
13 for each subgraph  $G_i, i = 1, 2$ , do
  begin
14 Let  $F_1, F_2, \dots, F_q$  be the  $C_o$ -components of  $G_i$ ;
15 for each component  $F_j, 1 \leq j \leq q$ , do
16 Draw( $C_o(G_i) \cup F_j, F_j$ )
  end
end
else { $P_c \neq P_{cc}$ . See Fig. 6.15.}
  begin
18 Draw all edges of  $P_c$  and  $P_{cc}$  on alternating sequences
  of horizontal and vertical line segments as in Fig. 6.17;
  {The directions of all edges of  $P_c$  and  $P_{cc}$  are
  decided.}
19 Let  $G_1$  be the graph obtained from  $G_W^{P_{cc}}$  by
  contracting all edges of  $P_{cc}$  that are on horizontal
  sides of rectangular embeddings of  $C_1, C_2, \dots, C_k$ ;
20 Let  $G_2$  be the graph obtained from  $G_E^{P_c}$  by

```



```

contracting all edges of  $P_c$  that are on horizontal
sides of rectangular embeddings of  $C_1, C_2, \dots, C_k$ ;
21   Let  $G_3 = G(C_1), G_4 = G(C_2), \dots, G_{k+2} = G(C_k)$ ;
22   for each graph  $G_i, 1 \leq i \leq k+2$ , do
      begin
23       Let  $F_1, F_2, \dots, F_q$  be the  $C_o$ -components of  $G_i$ ;
24       for each component  $F_j, 1 \leq j \leq q$ , do
25           DRAW( $C_o(G_i) \cup F_j, F_j$ )
      end
    end
  end
end

```

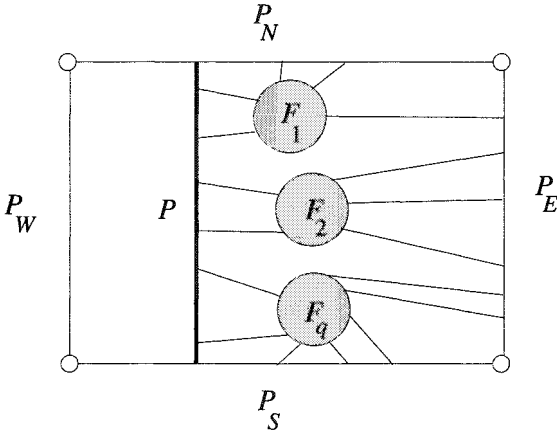


Fig. 6.20 C_o -Components F_1, F_2, \dots, F_q of G_E^P .

We now show that the algorithm **Rectangular-Draw**(G) takes linear time.

We find all C_o -components of G , and for each C_o -component we find boundary NS-, SN-, EW- and WE-paths if they exist. We do this by traversing all boundary faces of G using the counterclockwise depth-first search. During the traversal each boundary path gets a label, NN, NE, \dots , WW, according to the location of their starting and ending vertices on P_N, P_E, P_S and P_W , and every edge on a boundary path is marked according to the boundary path. Therefore boundary NS-, SN-, EW- and WE-paths, if they exist, can be readily found from the labels of the boundary paths in constant

time.

We then need to find the westmost NS-path P if no boundary NS-, SN-, EW- or WE-paths exists. We find P by traversing the boundary paths with ends on P_W using the counterclockwise depth-first search. During the traversal, we can find all edges that are on P and on a boundary NN-, EN-, SS-, or SE-path by checking the labels of boundary paths on which edges of P lie. Thus we can find paths P_{st} and P_{en} in the proof of Lemma 6.3.6. Traversing all facial cycles clockwise attached to path $P_{\alpha\beta}$, we detect the clockwise critical cycles attached to $P_{\alpha\beta}$ if they exist. An edge, which is not incident to a vertex on P and is traversed twice during this traversal, is detected as the leg of a clockwise critical cycle. One can observe that the head vertices and the tail vertices of all the critical cycles attached to P obey the so-called parenthesis rule. Therefore, considering the parenthesis structures of the found critical cycles attached to P , we can find the maximal critical cycles attached to P . From the found maximal critical cycles we find a partition-pair P_c and P_{cc} as mentioned in the proof of Lemma 6.3.6. One can do this by traversing the following edges a constant number of times: (i) the edges on P_c and P_{cc} , (ii) the edges on the facial cycles clockwise attached to $P_{\alpha\beta}$, and (iii) the edges on boundary paths newly created in the graphs divided by P_c and P_{cc} .

After finding a partitioning path or a partition-pair, we give labels to the newly created boundary paths by traversing them. The labels of some old boundary paths are updated for the newly found partitioning path or partition-pair. Clearly this can be done by traversing the respective boundary paths only once.

A problem arises if a subpath of the westmost NS-path P , which is neither on P_c nor on P_{cc} , is chosen as the westmost NS-path P' in a later recursive stage. If we again traversed the facial cycles attached to P' as mentioned before, then the time complexity of the algorithm would not be bounded by linear time. To overcome this difficulty, we keep the following information for later use when P is first constructed:

- (i) a list of all edges $e_i \in E(P)$ contained in boundary NN- and EN-paths;
- (ii) a list of all edges $e_i \in E(P)$ contained in boundary SS- and SE-paths;
- (iii) an array of length n , each element of which corresponds to a vertex of G and contains marks indicating whether the vertex is a head or a tail vertex of a clockwise critical cycle C attached to P and whether $n_{cc}(C) = 1$ or $n_{cc}(C) > 1$.

We use lists of (i) and (ii) to find P_{st} and P_{en} directly in later recursive

stages. Marks of vertices in (iii) indicate the existence of critical cycles attached to P' , and hence we need not to find these critical cycles again in a later recursive stage.

Throughout the execution of the algorithm, every face of G become a boundary face and then will never become non-boundary face. Hence each face is traversed by a constant number of times. Therefore the algorithm runs in linear time.

Theorem 6.3.7 *If a 2-connected plane graph G with $\Delta \leq 3$ has no bad cycle, then the algorithm **Rectangular-Draw** finds a rectangular drawing of G in linear time.*

6.3.4 Rectangular Grid Drawing

The algorithm **Rectangular-Draw**(G) in the Section 6.3.3 finds only the directions of all edges in G . From the directions the integer coordinates of vertices in G can be determined in linear time as follows. In this subsection we assume for simplicity that all vertices other than the four corners have degree three.

We now give a method of determining y-coordinates of the vertices in G ; x-coordinates can be determined similarly. Consider a graph T_y obtained from G by deleting all upward vertical edges of three types drawn by dotted lines in Fig. 6.21. Thus any upward edge drawn by a thick line in Fig. 6.22 is not deleted. Clearly T_y is a spanning tree of G . (T_y for the graph G in Fig. 6.15(a) is drawn by thick lines in Fig. 6.23.) A rectangular drawing of G is composed of several maximal horizontal and vertical line segments. The drawing in Fig. 6.23 is composed of 16 maximal vertical line segments together with 15 maximal horizontal line segments. All these maximal horizontal line segments are contained in T_y , and every vertex of G is contained in one of them. For each maximal horizontal line segment L , we will assign an integer $y(L)$ as the y-coordinate of every vertex on L . P_S is the lowermost maximal horizontal line segment, while P_N is the topmost one. We first set $y(P_S) = 0$. We then compute $y(L)$ from bottom to top. For each vertex v in G we will assign an integer $temp(v)$ as a temporary value of the y-coordinate of v .

For every vertex v on L there are two cases: either v has a neighbor u located below v or v has no neighbor u located below v . For the former case, we set $temp(v) = y(L') + 1$ where L' is the maximal horizontal line segment containing vertex u . For the latter case, we set $temp(v) = 0$. We then set

TEAM LING - LIVE, INFORMATIVE, NON-COST AND GENUINE !

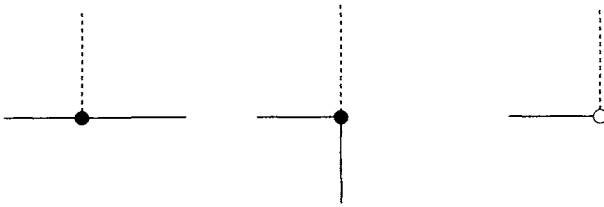


Fig. 6.21 Deleted upward edges.

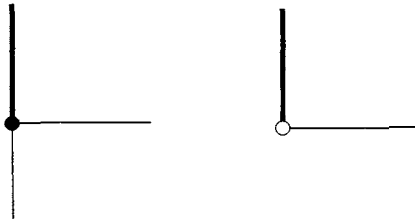


Fig. 6.22 Non-deleted upward edges.

$y(L) = \max_v \{temp(v)\}$ where the maximum is taken over all vertices v on L . One can easily compute $y(L)$ for all maximal horizontal line segments L from bottom to top using the counterclockwise depth-first search on T_y starting from the downward edge incident to the north-west corner a .

Thus we have showed that the integer coordinates of all vertices in a rectangular grid drawing can be computed in linear time.

We now give upper bounds on the area and half perimeter of a grid for a rectangular grid drawing. Let the coordinate of the south-west corner d be $(0,0)$, and let that of the north-east corner b be (W,H) . Then our grid drawing is “compact” in a sense that there is at least one vertical line segment of x -coordinate i for each integer $i, 0 \leq i \leq W$, and there is at least one horizontal line segment of y -coordinate j for each integer $j, 0 \leq j \leq H$. We have the following theorem on the sizes of a compact rectangular grid drawing.

Theorem 6.3.8 *If all vertices of a plane graph G have degree three except the four corners, then the sizes of any compact rectangular grid drawing D of G satisfy $W + H \leq \frac{n}{2}$ and $W \cdot H \leq \frac{n^2}{16}$.*

Proof. Let l_h be the number of maximal horizontal line segments, let l_v the number of maximal vertical line segments in D , and let $l = l_h + l_v$.

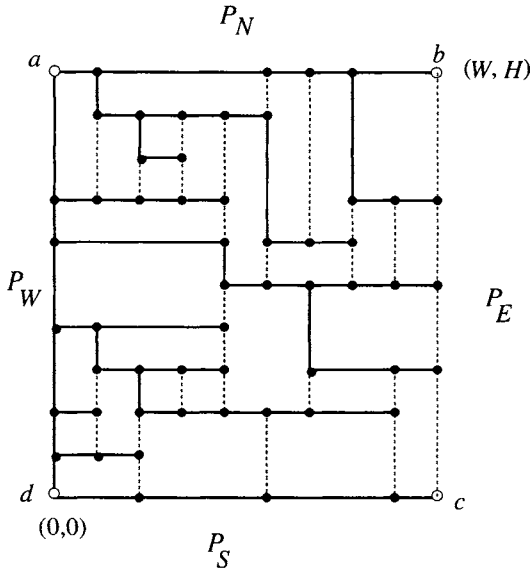


Fig. 6.23 Illustration of T_y by thick lines.

Each of the segments has exactly two ends. Each of the vertices except the four corners is an end of exactly one of the $l - 4$ maximal line segments other than $P_N, P_E, P_S,$ and P_W . Therefore we have

$$n - 4 = 2(l - 4)$$

and hence

$$l - 2 = \frac{n}{2}. \tag{6.1}$$

Since D is compact, we have

$$H \leq l_h - 1 \tag{6.2}$$

and

$$W \leq l_v - 1. \tag{6.3}$$

By using Eqs. (6.1)–(6.3), we obtain

$$W + H \leq l_v + l_h - 2 = l - 2 = \frac{n}{2}.$$

This relation immediately implies the bound on area: $W \cdot H \leq \frac{n^2}{16}$. \square

The bounds above are tight, because there are an infinite number of examples attaining the bounds [RNN98]; finding such an example is left as an exercise.

6.4 Rectangular Drawings without Designated Corners

In Section 6.3 we considered a rectangular drawing of a plane graph G with $\Delta \leq 3$ for the case where four outer vertices of degree two are designated as the corners. In this section we consider a general case where corners are not designated in advance. Then our problem is how to examine whether G has four outer vertices of degree two such that there is a rectangular drawing of G having them as the corners, and how to efficiently find them if there is.

Remember that cycles C and C' in a plane graph G are independent if $G(C)$ and $G(C')$ have no common vertex, and that a set S of cycles is independent if any pair of cycles in S are independent. Figure 6.24 illustrates a 2-connected plane graph G with $\Delta \leq 3$. Since every vertex of G has degree two or three, all 3-legged cycles are “laminar” in essence. Some of 2-legged and 3-legged cycles are indicated by dotted lines; C_1 and C_2 are 2-legged cycles, and C_3, C_4, C_5 and C_6 are 3-legged cycles. C_3 and C_4 are contained in $G(C_1)$, C_5 and C_6 are contained in $G(C_2)$, and C_5 is contained in $G(C_6)$. There are many independent sets of cycles. For example, $S_1 = \{C_1, C_2\}$ and $S_2 = \{C_2, C_3, C_4\}$ are independent sets of cycles.

We are now ready to present a necessary and sufficient condition for the existence of appropriate four outer vertices as in Theorem 6.4.1 [RNN02].

Theorem 6.4.1 *Assume that G is a 2-connected plane graph with $\Delta \leq 3$ and has four or more outer vertices of degree two. Then four of them can be designated as the corners so that G has a rectangular drawing with the designated corners if and only if G satisfies the following three conditions:*

- (a) every 2-legged cycle in G contains at least two outer vertices of degree two;
- (b) every 3-legged cycle in G contains at least one outer vertex of degree two; and
- (c) if an independent set S of cycles in G consists of c_2 2-legged cycles and

c_3 3-legged cycles, then $2c_2 + c_3 \leq 4$.

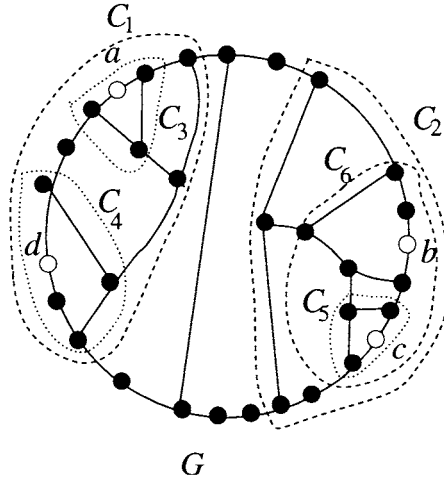


Fig. 6.24 Plane graph.

For the set $S_1 = \{C_1, C_2\}$ above $c_2 = 2$, $c_3 = 0$ and hence $2c_2 + c_3 = 4$, while for $S_2 = \{C_2, C_3, C_4\}$ $c_2 = 1$ and $c_3 = 2$ and hence $2c_2 + c_3 = 4$.

It is rather easy to prove the necessity of Theorem 6.4.1 as follows.

Necessity of Theorem 6.4.1. Assume that G has a rectangular drawing D for four outer vertices of degree two appropriately chosen as the corners. Then by Fact 6.3.1 every 2-legged cycle in G contains at least two corners of D . Similarly every 3-legged cycle contains at least one corner of D .

Let an independent set S consist of c_2 2-legged cycles and c_3 3-legged cycles in G . Then each of the c_2 2-legged cycles in S contains at least two corners of D , and each of the c_3 3-legged cycle in S contains at least one corner. Since all cycles in S are independent, they are vertex-disjoint with each other. Therefore there are at least $2c_2 + c_3$ corners in D . Since there are exactly four corners in D , we have $2c_2 + c_3 \leq 4$. \square

In order to prove the sufficiency of Theorem 6.4.1, it suffices to show that if the three conditions (a)–(c) in Theorem 6.4.1 hold then one can choose four outer vertices of degree two as the corners a, b, c and d so that the conditions (r1) and (r2) in Theorem 6.3.2 hold. We now outline a

proof. See [RNN02] for the detail of a proof and a linear algorithm to find appropriate four outer vertices of degree two.

Let J_1, J_2, \dots, J_p , $p \geq 1$, be the $C_o(G)$ -components of G , then they must be “in series” as illustrated in Fig. 6.25. There are two 2-legged cycles C_1 and C_2 , drawn by thick lines, which are contained in $G_1 = C_o(G) \cup J_1$ and $G_p = C_o(G) \cup J_p$, respectively. We choose a corner from each of the innermost 3-legged cycles. If four corners have not been chosen, we choose the remaining corners from C_1 and C_2 so that each of the 2-legged cycles including C_1 and C_2 contains at least two corners. (In Fig. 6.24 corners a, c and d are chosen from the innermost 3-legged cycles C_3, C_5 and C_4 , respectively, and the remaining corner b is chosen from a 2-legged cycle C_2 .)

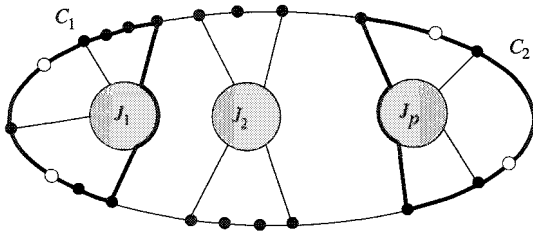


Fig. 6.25 Plane graph G with $C_o(G)$ -components J_1, J_2, \dots, J_p .

6.5 Rectangular Drawings of Planar Graphs

In Sections 6.2, 6.3 and 6.4 we considered rectangular drawings of plane graphs (with fixed embedding). In this section we consider rectangular drawings of planar graphs with $\Delta \leq 3$ (without fixed embedding). We say that a planar graph G has a rectangular drawing if at least one of the plane embeddings of G has a rectangular drawing. Figures 6.26(b), (c) and (d) depict three different plane embeddings of the same planar graph. The embedding in Fig. 6.26(b) has a rectangular drawing as illustrated in Fig. 6.26(a), and hence the planar graph has a rectangular drawing. On the other hand, neither the embedding in Fig. 6.26(c) nor that in Fig. 6.26(d) has a rectangular drawing, because there are 3-legged cycles indicated by dotted lines which have no outer vertex of degree two. Examining whether a planar graph G with $\Delta \leq 3$ has a rectangular drawing is not a trivial problem, since G may have an exponential number of plane embeddings in general. A straightforward algorithm checking each of all the embeddings

by the linear algorithm in Section 6.4 does not run in polynomial time. In this section we present a linear algorithm to examine whether there is a plane embedding of a planar graph G with $\Delta \leq 3$ which has a rectangular drawing [RNG04]. The key idea is that it is sufficient to check at most four embeddings of G .

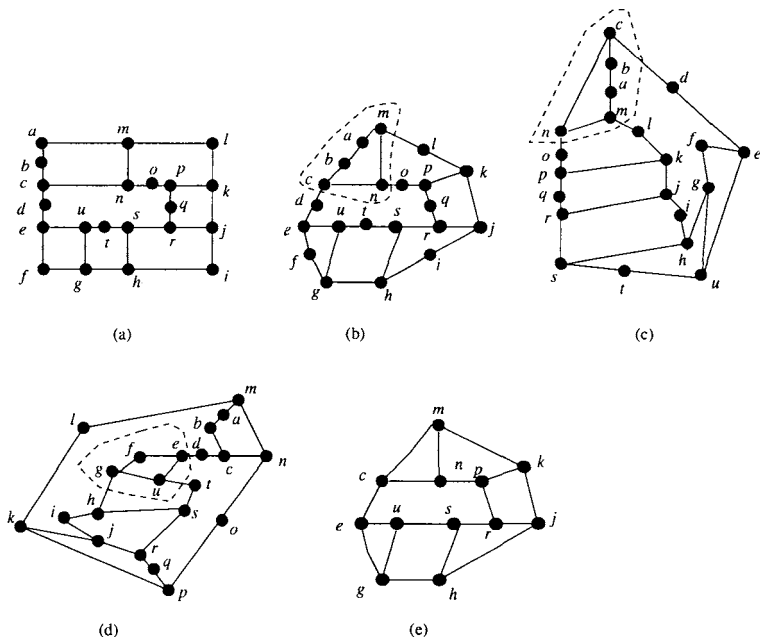


Fig. 6.26 A rectangular drawing (a) and three different embeddings (b), (c) and (d) of the same graph which is subdivision of the graph in (e).

Let G be a planar 2-connected graph with $\Delta \leq 3$, and let Γ be a plane embedding of G . We denote by $F_o(\Gamma)$ the outer face of Γ . For a cycle C of Γ , we call the plane subgraph of Γ inside C (including C) the *inner subgraph* $\Gamma_I(C)$ for C , and call the plane subgraph of Γ outside C (including C) the *outer subgraph* $\Gamma_O(C)$ for C . A rectangular drawing of Γ is a convex drawing of G , and hence by Corollary 5.3.3 G should not have a forbidden separation pair, and by Corollary 5.3.6 $F_o(\Gamma)$ should contain every vertex of critical separation pairs. Since $\Delta \leq 3$, G has no forbidden separation pair although G may have critical separation pair. The pair of leg-vertices of any 2-legged cycle in Γ is a separation pair, but is not always a critical separation pair. We call a cycle C in Γ *regular* if the plane graph $\Gamma - \Gamma_I(C)$

has a cycle. The 2-legged cycles C_1 and C_2 in Fig. 6.24 are regular, while the 2-legged cycle indicated by thin dotted lines in Fig. 6.27(a) is not regular. Clearly a 2-legged cycle C in Γ is not regular if and only if $\Gamma - \Gamma_I(C)$ is a chain on $F_o(\Gamma)$. Thus the pair of leg-vertices of a regular 2-legged cycle is a critical separation pair, and hence $F_o(\Gamma)$ should contain the pair of leg-vertices of every regular 2-legged cycle. In the plane graph depicted in Fig. 6.27(b), the cycle C drawn by thick solid lines is a regular 3-legged cycle, while the cycle C' indicated by thin dotted lines is not a regular 3-legged cycle. A 3-legged cycle C is not regular if and only if $\Gamma - \Gamma_I(C)$ contains exactly one vertex that has degree three in G .

In Section 6.5.1 we consider the case where G is a subdivision of a planar 3-connected cubic graph and in Section 6.5.2 we consider the other case.

6.5.1 Case for a Subdivision of a Planar 3-connected Cubic Graph

In this section we deal with a subdivision G of a planar 3-connected cubic graph. All the graphs in Figs. 6.26 and 6.27 are subdivisions of planar 3-connected cubic graphs. By Corollary 5.3.4 G always has a convex drawing, but G does not always have a rectangular drawing. We present a necessary and sufficient condition for G to have a rectangular drawing.

A graph G is called *cyclically 4-edge-connected* if the removal of any three or fewer edges leaves a graph such that exactly one of the connected components has a cycle [Tho92]. The graph in Fig. 6.27(a) is cyclically 4-edge-connected. On the other hand, the graph in Fig. 6.27(b) is not cyclically 4-edge-connected, since the removal of the three edges drawn by thick dotted lines leaves a graph with two connected components each of which has a cycle. Similarly all the graphs in Fig. 6.26 are not cyclically 4-edge-connected.

We call a face F of Γ a *peripheral face* for a 3-legged cycle C in Γ if F is in $\Gamma_o(C)$ and the contour of F contains some edges on C . Clearly there are exactly three peripheral faces for any 3-legged cycle in Γ . In Fig. 6.27(b) F_1 , F_2 and F_3 are the three peripheral faces for the 3-legged cycle C drawn by thick solid lines.

Let Γ be an arbitrary plane embedding of a subdivision G of a planar 3-connected cubic graph. Then Γ has no regular 2-legged cycle and hence Γ has no critical separation pair. However, Γ may have a regular 3-legged cycle. Γ has no regular 3-legged cycle if and only if G is cyclically 4-edge-connected. We have the following theorem on a necessary and sufficient

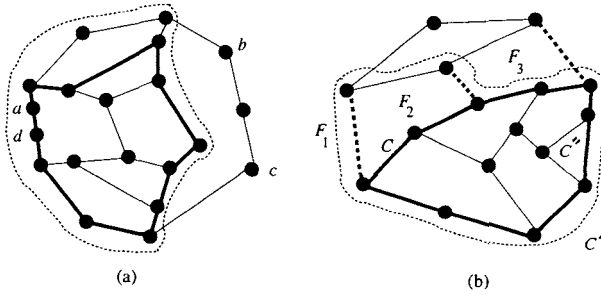


Fig. 6.27 (a) A cyclically 4-edge connected graph, and (b) a graph which is not cyclically 4-edge connected.

condition for G to have a rectangular drawing [RNG04].

Theorem 6.5.1 *Let G be a subdivision of a planar 3-connected cubic graph, and let Γ be an arbitrary plane embedding of G .*

(a) *Suppose first that G is cyclically 4-edge-connected, that is, Γ has no regular 3-legged cycle, as the graph in Fig. 6.27(a). Then the planar graph G has a rectangular drawing if and only if Γ has a face F such that*

- (i) *F contains at least four vertices of degree two;*
- (ii) *there are at least two chains on F ; and*
- (iii) *if there are exactly two chains on F , then they are not adjacent and each of them contains at least two vertices.*

(The plane embedding in Fig. 6.27(a) has four outer chains, and has a rectangular drawing as illustrated in Fig. 6.28.)

(b) *Suppose next that G is not cyclically 4-edge-connected, that is, Γ has a regular 3-legged cycle C as the graphs in Figs. 6.26(b) and 6.27(b). Let F_1 , F_2 and F_3 be the three peripheral faces for C , and let Γ_1 , Γ_2 and Γ_3 be the plane embeddings of G taking F_1 , F_2 and F_3 , respectively, as the outer face. Then the planar graph G has a rectangular drawing if and only if at least one of the three embeddings Γ_1 , Γ_2 and Γ_3 has a rectangular drawing. (Figures 6.26(b), (c) and (d) depict Γ_1 , Γ_2 and Γ_3 for the regular 3-legged cycle indicated by dotted lines in Fig. 6.26(b).)*

Before giving a proof of Theorem 6.5.1, we observe the following Fact 6.5.2 and Lemma 6.5.3. Fact 6.5.2 is immediate from Theorem 2.2.2.

TEAM LING - LIVE, INFORMATIVE, NON-COST AND GENUINE !

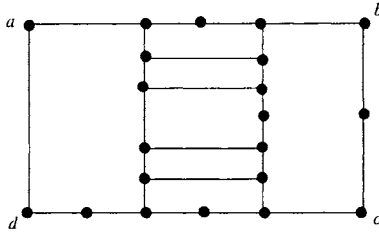


Fig. 6.28 Rectangular drawing of the graph in Fig. 6.27(a).

The proof for Lemma 6.5.3 is left as an exercise.

Fact 6.5.2 *Let G be a subdivision of a 3-connected planar graph. Then there is exactly one embedding of G for each face embedded as the outer face. Furthermore, for any two plane embeddings Γ and Γ' of G , any facial cycle in Γ is a facial cycle in Γ' .*

Lemma 6.5.3 *Let G be a subdivision of a planar 3-connected cubic graph, and let Γ be an arbitrary plane embedding of the planar graph G . Then the following (a) – (c) hold.*

- (a) *If C is a 2-legged cycle in Γ , then the legs and the leg-vertices v_1 and v_2 of C are on the outer face $F_o(\Gamma)$, and the set of all vertices not in $\Gamma_1(C)$ induces a chain of G on $F_o(\Gamma)$ with supports v_1 and v_2 .*
- (b) *For any chain P on $F_o(\Gamma)$, the outer face of the plane graph $\Gamma - P$ is a 2-legged cycle in Γ .*
- (c) *Any pair of 2-legged cycles in Γ are not independent.*

We are now ready to give a proof of Theorem 6.5.1.

Proof of Theorem 6.5.1(a): Necessity: Assume that the planar graph G is cyclically 4-edge-connected and has a rectangular drawing D . Then there is a plane embedding Γ' of G which has a rectangular drawing. By Fact 6.5.2, the outer face $F_o(\Gamma')$ of Γ' corresponds to a face F of Γ . We show that F satisfies (i), (ii) and (iii) as follows.

(i) The four corners of D have degree two in G , and hence $F = F_o(\Gamma')$ contains at least four vertices of degree two.

(ii) If all the four corner vertices are contained in the same chain on $F_o(\Gamma')$ as illustrated in Fig. 6.29(a), then Γ' would not have a rectangular drawing. Therefore, there are at least two chains on $F = F_o(\Gamma')$.

(iii) Suppose that there are exactly two chains on $F = F_o(\Gamma')$. Then

clearly each of them contains exactly two of the four corner vertices of degree two; otherwise, Γ' would not have a rectangular drawing as illustrated in Fig. 6.29(b). Furthermore, the two chains must be non-adjacent; otherwise, Γ' would have a 3-legged cycle C containing no vertex of degree two on $F_o(\Gamma')$, contrary to Theorem 6.4.1(b). In Fig. 6.29(c) C is drawn by thick lines.

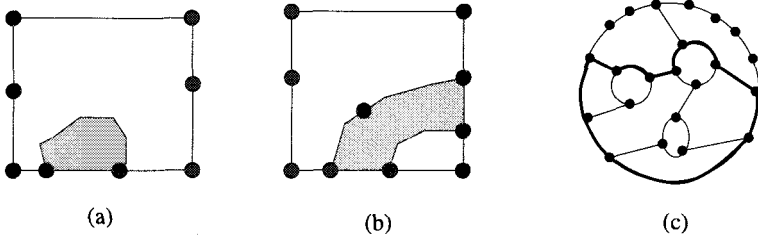


Fig. 6.29 (a) Single chain, (b) one of the two chains contains exactly one vertex, and (c) adjacent chains on the outer cycle.

Sufficiency: We give a constructive proof for the sufficiency of Theorem 6.5.1(a). Assume that Γ has a face F satisfying Conditions (i) – (iii) of Theorem 6.5.1(a). Let Γ' be an embedding of G such that $F = F_o(\Gamma')$. Then by Condition (i) in Theorem 6.5.1(a) $F_o(\Gamma') = F$ contains at least four vertices of degree two. Thus it is sufficient to prove that Γ' satisfies Conditions (a) – (c) in Theorem 6.4.1.

(a) We now show that Γ' satisfies Condition (a) in Theorem 6.4.1, that is, every 2-legged cycle C in Γ' contains at least two vertices of degree two on $F_o(\Gamma')$. By Condition (ii) in Theorem 6.5.1(a) there are at least two chains on $F_o(\Gamma')$, and hence we have the following two cases to consider.

Case 1: there are exactly two chains P_1 and P_2 on $F_o(\Gamma')$.

By Lemma 6.5.3 Γ' has exactly two 2-legged cycles C_1 and C_2 as illustrated in Fig. 6.30(a), where C_1 is drawn by thick solid lines and C_2 is indicated by dotted lines. Clearly each of the 2-legged cycles C_1 and C_2 contains exactly one chain. By Condition (iii) of Theorem 6.5.1(a) each of the two chains P_1 and P_2 contains at least two vertices of degree two. Therefore, each 2-legged cycle contains at least two vertices of degree two on $F_o(\Gamma')$. Thus Γ' satisfies Condition (a) of Theorem 6.4.1.

Case 2: there are more than two chains on $F_o(\Gamma')$.

Assume that there are exactly r chains on $F_o(\Gamma')$ and $r \geq 3$. Then each 2-legged cycle contains $r - 1$ (≥ 2) chains on $F_o(\Gamma')$, and hence contains at least two vertices of degree two on $F_o(\Gamma')$. Thus Γ' satisfies Condition (a)

of Theorem 6.4.1.

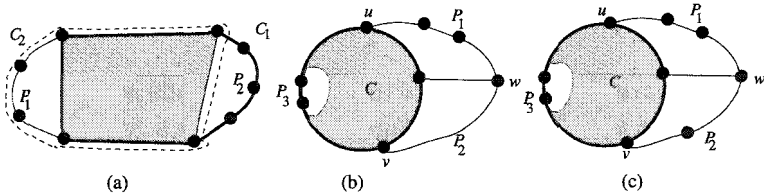


Fig. 6.30 Illustration for the proof of Theorem 6.5.1(a).

(b) We next show that Γ' satisfies Condition (b) in Theorem 6.4.1, that is, every 3-legged cycle C in Γ' contains at least one vertex of degree two on $F_o(\Gamma')$. Since G is cyclically 4-edge connected, C is not regular and hence $\Gamma' - \Gamma'_I(C)$ has no cycle. Therefore, C contains edges on $F_o(\Gamma')$, two of the three leg-vertices of C , say u and v , are on $F_o(\Gamma')$, and $\Gamma' - \Gamma'_I(C)$ contains exactly one vertex w that has degree three in G . Clearly w is on $F_o(\Gamma')$. The three vertices u , w and v divide $F_o(\Gamma')$ into three paths P_1 , P_2 and P_3 ; P_1 goes from u to w on $F_o(\Gamma')$, P_2 from w to v , and P_3 from v to u , as illustrated in Figs. 6.30(b) and (c). By Condition (ii) in Theorem 6.5.1(a) there are at least two chains on $F = F_o(\Gamma')$. We now have the following two cases.

Case 1: P_1 or P_2 does not contain a chain.

In this case there is at least one chain on P_3 since there are at least two chains on $F_o(\Gamma')$. Therefore, C contains at least one vertex of degree two on $F_o(\Gamma')$, as illustrated in Fig. 6.30(b). Thus Γ' satisfies Condition (b) in Theorem 6.4.1.

Case 2: both P_1 and P_2 contain a chain.

In this case the two chains on P_1 and P_2 are adjacent, as illustrated in Fig. 6.30(c). Therefore by Condition (iii) in Theorem 6.5.1(a) $F_o(\Gamma')$ contains three or more chains, and hence there is at least one chain on P_3 . Thus C contains at least one vertex of degree two on $F_o(\Gamma')$, and hence Γ' satisfies Condition (b) in Theorem 6.4.1.

(c) We finally show that Γ' satisfies Condition (c) in Theorem 6.4.1. Let \mathcal{S} be any independent set of cycles in Γ' . Then $c_2 \leq 1$ by Lemma 6.5.3(c).

We now claim that $c_3 \leq 1$. Otherwise, there are two different 3-legged cycles C and C' in \mathcal{S} . Since G is cyclically 4-edge-connected, C and C' are not regular and hence each of the plane graphs $\Gamma' - \Gamma'_I(C)$ and $\Gamma' - \Gamma'_I(C')$ contains exactly one vertex that has degree three in G . Then clearly $\Gamma'_I(C)$ and $\Gamma'_I(C')$ have common vertices, contrary to the assumption that \mathcal{S} is an

independent set of cycles.

Since $c_2 \leq 1$ and $c_3 \leq 1$, we have $2c_2 + c_3 \leq 3$. Thus Γ' satisfies Condition (c) of Theorem 6.4.1. \square

Proof of Theorem 6.5.1(b): Since the proof for the sufficiency is obvious, we give a proof for the necessity. Suppose that Γ has a regular 3-legged cycle C and that the planar graph G has a rectangular drawing. Then there is a plane embedding Γ' of G which has a rectangular drawing. Let F be the face of Γ corresponding to $F_o(\Gamma')$. It suffices to show that F is one of the three peripheral faces F_1, F_2 and F_3 for C in Γ .

We first consider the case where C contains an edge on $F_o(\Gamma)$. Let C' be the cycle in $\Gamma - \Gamma_I(C)$ such that $\Gamma_I(C')$ has the maximum number of edges, as illustrated in Fig. 6.31(a). One can observe that C' is a 3-legged cycle in Γ , and any face of Γ other than F_1, F_2 and F_3 is in $\Gamma_I(C)$ or $\Gamma_I(C')$. Therefore it is sufficient to prove that F is neither in $\Gamma_I(C)$ nor in $\Gamma_I(C')$. If F is in $\Gamma_I(C)$, then C' is a 3-legged cycle in Γ' and contains no vertex on $F_o(\Gamma') = F$, a contradiction to Theorem 6.4.1(b). Similarly, if F is in $\Gamma_I(C')$, then C is a 3-legged cycle in Γ' and contains no vertex on $F_o(\Gamma') = F$, a contradiction to Theorem 6.4.1(b).

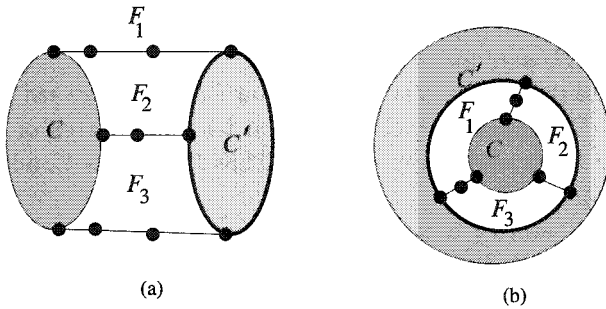


Fig. 6.31 Cycles C and C' in Γ .

We next consider the case where C does not contain any edge on $F_o(\Gamma)$. Let C' be the cycle in $\Gamma - \Gamma_I(C)$ such that $\Gamma_I(C')$ includes $\Gamma_I(C)$ and has the minimum number of edges, as illustrated in Fig. 6.31(b). Any face of Γ other than F_1, F_2 and F_3 is in $\Gamma_I(C)$ or $\Gamma_O(C')$. Therefore it is sufficient to prove that F is neither in $\Gamma_I(C)$ nor in $\Gamma_O(C')$. If F is in $\Gamma_I(C)$, then C' is a 3-legged cycle in Γ' and contains no vertex on $F_o(\Gamma') = F$, a contradiction to Theorem 6.4.1(b). If F is in $\Gamma_O(C')$, then C is a 3-legged cycle in Γ' and

contains no vertex on $F_o(\Gamma') = F$, a contradiction to Theorem 6.4.1(b). \square

Theorem 6.5.1 immediately leads to a linear algorithm to examine whether a subdivision G of a planar 3-connected cubic graph has a rectangular drawing.

6.5.2 The Other Case

In this section we assume that G is a planar 2-connected graph with $\Delta \leq 3$ but is not a subdivision of a planar 3-connected cubic graph, and give a necessary and sufficient condition for G to have a rectangular drawing.

Let Γ be an arbitrary plane embedding of G . If G has at most two vertices of degree three, then one can easily examine whether G has a rectangular drawing. (See Fig. 6.32.) We may thus assume that G has three or more vertices of degree three. Then Γ has a regular 2-legged cycle C ; otherwise, G would be a subdivision of a 3-connected cubic graph. The pair of leg-vertices of C is a critical separation pair. Let $(x_1, y_1), (x_2, y_2), \dots, (x_l, y_l)$

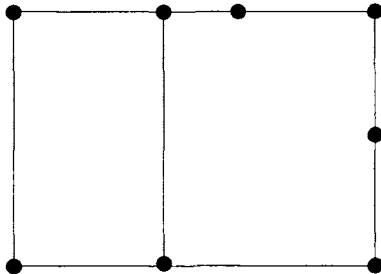
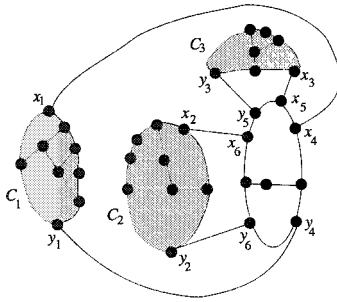


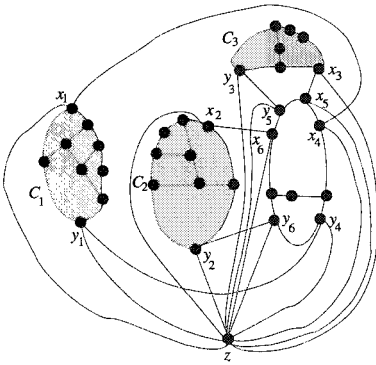
Fig. 6.32 Plane graph exactly two vertices of degree three.

be all critical separation pairs of G . Clearly $l = O(n)$. If there is a plane embedding Γ' of G having a rectangular drawing, then the outer face $F_o(\Gamma')$ must contain all vertices $x_1, y_1, x_2, y_2, \dots, x_l, y_l$. Construct a graph G^+ from G by adding a dummy vertex z and dummy edges (x_i, z) and (y_i, z) for all indices $i, 1 \leq i \leq l$. Then G has a plane embedding whose outer face contains all vertices $x_1, y_1, x_2, y_2, \dots, x_l, y_l$ if and only if G^+ is planar. (Figure 6.33(b) illustrates G^+ for G in Fig. 6.33(a).)

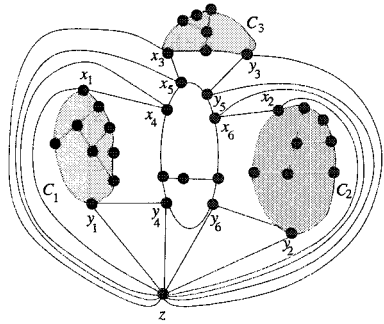
We may thus assume that G^+ is planar. Let Γ^+ be an arbitrary plane embedding of G^+ such that z is embedded on the outer face, as illustrated in Fig. 6.33(c). We delete from Γ^+ the dummy vertex z and all dummy edges incident to z , and let Γ^* be the resulting plane embedding of G ,



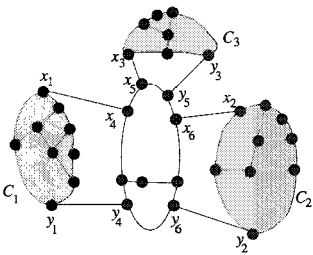
(a) G and Γ



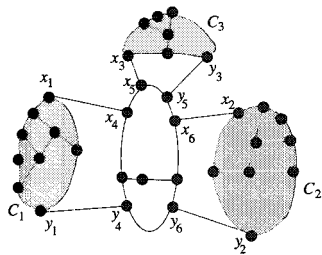
(b) G^+



(c) Γ^+



(d) Γ^*



(e) Γ_1^*

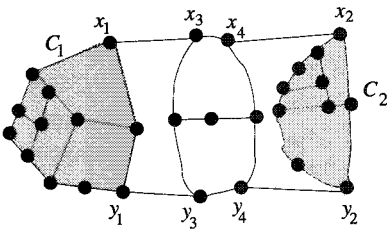
Fig. 6.33 $G, \Gamma, G^+, \Gamma^+, \Gamma^*$ and Γ_1^* .

in which $F_o(\Gamma^*)$ contains all vertices $x_1, y_1, x_2, y_2, \dots, x_l, y_l$, as illustrated in Fig. 6.33(d). One can observe that every 2-legged cycle in Γ^* has the leg-vertices on $F_o(\Gamma^*)$.

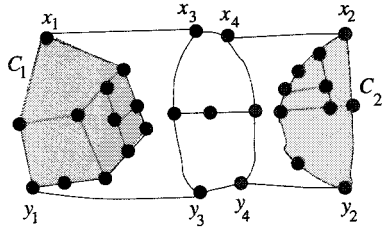
Let p be the largest integer such that a number p of 2-legged cycles in Γ^* are independent with each other. Then $p \geq 2$ since Γ and hence Γ^* has a regular 2-legged cycle. Γ^* has a number p of minimal 2-legged cycles. (In Fig. 6.33(d) the $p = 3$ minimal 2-legged cycles C_1, C_2 and C_3 are shaded, which corresponds to the “leaves” of a “3-connected component decomposition tree” of G .) If Γ_1^* is a plane embedding obtained from Γ^* by flipping $\Gamma_I^*(C)$ for a minimal 2-legged cycle C , then the leg-vertices of all 2-legged cycles in Γ_1^* are on $F_o(\Gamma_1^*)$. (The embedding Γ_1^* in Fig. 6.33(e) is obtained from Γ^* in Fig. 6.33(d) by flipping $\Gamma_I^*(C_1)$.) One can observe that only the plane embeddings of G that can be obtained from Γ^* by flipping $\Gamma_I^*(C)$ for some minimal 2-legged cycles C have $x_1, y_1, x_2, y_2, \dots, x_l, y_l$ on the outer face. We now have $p = 2$; if $p \geq 3$, then any plane embedding of G whose outer face contains all vertices $x_1, y_1, x_2, y_2, \dots, x_l, y_l$ has three or more independent 2-legged cycles, and hence by Theorem 6.4.1(c) the embedding has no rectangular drawing, and consequently the planar graph G has no rectangular drawing.

Since $p = 2$, Γ^* has exactly two independent 2-legged cycles C_1 and C_2 . We may assume without loss of generality that C_1 and C_2 are minimal 2-legged cycles, as illustrated in Fig. 6.34(a). By flipping $\Gamma_I^*(C_1)$ or $\Gamma_I^*(C_2)$ around the leg-vertices of C_1 or C_2 , we have four different embeddings $\Gamma_1 (= \Gamma^*), \Gamma_2, \Gamma_3$ and Γ_4 such that each $F_o(\Gamma_i), 1 \leq i \leq 4$, contains all vertices $x_1, y_1, x_2, y_2, \dots, x_l, y_l$, as illustrated in Fig. 6.34. Since only the four embeddings $\Gamma_1, \Gamma_2, \Gamma_3$ and Γ_4 of G have all vertices $x_1, y_1, x_2, y_2, \dots, x_l, y_l$ on the outer face, G has a rectangular drawing if and only if at least one of $\Gamma_1, \Gamma_2, \Gamma_3$ and Γ_4 has a rectangular drawing. (None of the embeddings Γ_1, Γ_2 , and Γ_4 in Figs. 6.34(a), (b) and (d) has a rectangular drawing since there are no four vertices of degree two on the outer face, while the embedding Γ_3 in Fig. 6.34(c) has a rectangular drawing as illustrated in Fig. 6.35.) We thus have the following theorem.

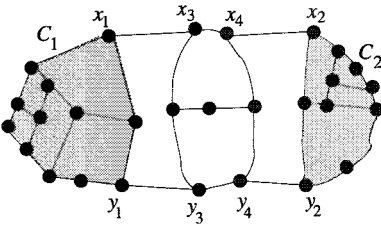
Theorem 6.5.4 *Let G be a planar 2-connected graph with $\Delta \leq 3$ which is not a subdivision of a planar 3-connected cubic graph. Let Γ be a plane embedding of G such that every 2-legged cycle in Γ has leg-vertices on $F_o(\Gamma)$, let Γ have exactly two independent 2-legged cycles, and let C_1 and C_2 be the two minimal 2-legged cycles in Γ . Let $\Gamma_1 (= \Gamma), \Gamma_2, \Gamma_3$, and Γ_4 be the four embeddings of G obtained from Γ by flipping $\Gamma_I(C_1)$ or $\Gamma_I(C_2)$ around the leg-vertices of C_1 and C_2 . Then G has a rectangular drawing if and only if at least one of the four embeddings $\Gamma_1, \Gamma_2, \Gamma_3$, and Γ_4 has a rectangular drawing.*



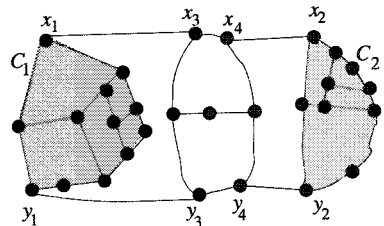
(a) $\Gamma_1 = \Gamma^*$



(b) Γ_2



(c) Γ_3



(d) Γ_4

Fig. 6.34 $\Gamma_1, \Gamma_2, \Gamma_3$ and Γ_4 .

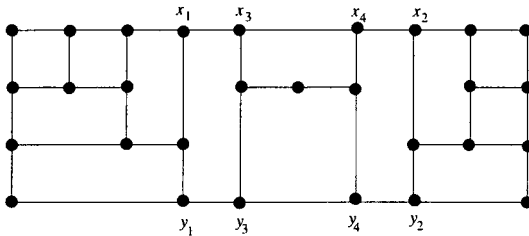


Fig. 6.35 Rectangular drawing of Γ_3 in Fig. 6.34(c).

Following the argument above one can easily implement a linear algorithm to examine whether a planar graph G with $\Delta \leq 3$ has a rectangular drawing and to find a rectangular drawing of G if it exists.

6.6 Bibliographic Notes

A drawing of a plane graph G is called an *inner rectangular drawing* of G if each vertex of G is drawn as a point, each edge of G is drawn as a horizontal or vertical line segment, each inner face of G is drawn as a rectangle and the outer cycle of G is drawn as a rectilinear polygon [MMN02]. Miura *et al.* [MHN04] reduced the problem of finding an inner rectangular drawing of a plane graph G with $\Delta \leq 4$ to a problem of finding a perfect matching of a new bipartite graph constructed from G . It immediately yields the result presented in Section 6.2 on an ordinary rectangular drawing of plane graphs with $\Delta \leq 4$.

Kozminski and Kinnen [KK84] established a necessary and sufficient condition for the existence of a “rectangular dual” of an inner triangulated plane graph, that is, a rectangular drawing of the dual graph of an inner triangulated plane graph, and gave an $O(n^2)$ algorithm to obtain it. Based on the characterization of [KK84], Bhasker and Sahni [BS88] and Xin He [He93] developed linear-time algorithms to find a rectangular dual. Kant and Xin He [KH97] presented two more linear-time algorithms. Xin He [He95] presented a parallel algorithm for finding a rectangular dual. Lai and Leinwand [LL90] reduced the problem of finding a rectangular dual of an inner triangulated plane graph G to a problem of finding a perfect matching of a new bipartite graph constructed from G . Their construction is different from that in Section 6.2, their bipartite graph has an $O(n^2)$ number of edges, and hence their method takes time $O(n^{2.5})$ to find a rectangular dual or a rectangular drawing of a plane graph with $\Delta \leq 3$.

Exercise

1. Show that a plane graph G with $\Delta \leq 4$ has a rectangular drawing for some quadruplet of outer vertices of degree two appropriately chosen as the corners if and only if a certain bipartite graph constructed from G has a perfect matching [LL90, MHN04].
2. Prove Lemma 6.3.4.
3. Find an infinite number of plane graphs G with $\Delta \leq 3$ such that any rectangular drawing of G needs area at least $n^2/16$, where n is the number of vertices in G .
4. Let G be a 2-connected plane graph with $\Delta \leq 3$, and let k be the number of vertices of degree three on $C_o(G)$. Prove that a $C_o(G)$ -component J

contains a cycle with k or less legs if J contains a cycle.

5. Prove that a 2-connected plane graph G with $\Delta \leq 3$ has a pair of independent 2-legged cycles if G has two or more $C_o(G)$ -components.
6. Let G be a 2-connected plane graph such that all vertices have degree three or four except four outer vertices of degree two. Let G' be the graph obtained from G by replacing each vertex of degree four by a cycle as illustrated in Fig. 6.36. Then prove that G has a rectangular drawing if and only if G' has a rectangular drawing in which each replaced cycle together with its four legs has one of the two embeddings illustrated in Fig. 6.37.
7. Prove Lemma 6.5.3.
8. Implement a linear algorithm to examine whether a planar graph G with $\Delta \leq 3$ has a rectangular drawing and to find a rectangular drawing of G if G has.

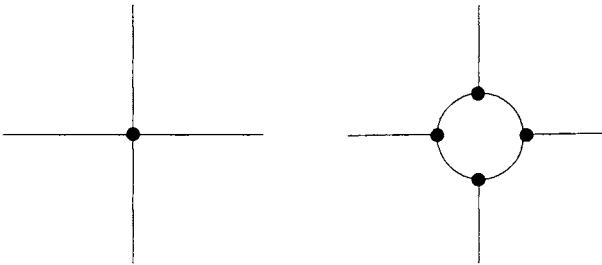


Fig. 6.36 Replacing a vertex of degree four with a cycle.

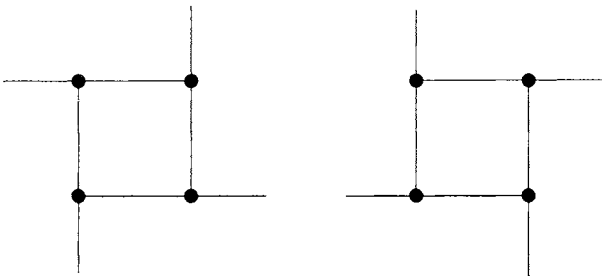


Fig. 6.37 Embeddings of four legs.

Chapter 7

Box-Rectangular Drawing

7.1 Introduction

A *box-rectangular drawing* of a plane graph G is a drawing of G such that each vertex is drawn as a rectangle, called a *box*, and the contour of each face is drawn as a rectangle, as illustrated in Fig. 1.6(d). A vertex may be drawn as a degenerate rectangle, that is, a point. We have seen in Section 1.5.1 that box-rectangular drawings have practical applications in floorplanning of MultiChip Modules (MCM) and in architectural floorplanning. If G has multiple edges or a vertex of degree five or more, then G has no rectangular drawing but may have a box-rectangular drawing. However, not every plane graph has a box-rectangular drawing. Rahman *et al.* [RNN00] established a necessary and sufficient condition for the existence of a box-rectangular drawing of a plane graph, and gave a linear algorithm to find a box-rectangular drawing if it exists.

In this chapter we present the results of Rahman *et al.* In Section 7.2 we presents some definitions and observations regarding box-rectangular drawings. In Section 7.3 we present a linear algorithm to find a box-rectangular drawing for a case where four outer vertices are designated as the “corner boxes” of the drawing. In Section 7.4 we present a linear algorithm to find a box-rectangular drawing for a general case in which no vertices are designated as the “corner boxes.”

7.2 Preliminaries

In this section we give some definitions and preliminary observations regarding box-rectangular drawings.

Throughout this chapter we assume that a *graph* G is a so-called multi-

TEAM LING - LIVE, INFORMATIVE, NON-COST AND GENUINE !

graph, which may have *multiple edges*, i.e., edges sharing both ends. If G has no multiple edges, then G is called a *simple graph*. For simplicity's sake we assume that G has three or more vertices and is 2-connected.

We call a box-rectangular drawing D of G a *box-rectangular grid drawing* if each edge as well as each side of a box is drawn along a grid line. A vertex may be drawn as a *degenerate box*, that is, a point, in a box-rectangular drawing D . We often call a degenerate box in D a *point* and call a non-degenerate box a *real box*. We call a rectangle corresponding to $C_o(G)$ the *outer rectangle*, which has exactly four corners. We call a corner of the outer rectangle simply a *corner*. A box in D is called a *corner box* if it contains at least one corner. A corner box may be degenerate.

We now have the following four facts and a lemma.

Fact 7.2.1 *Any box-rectangular drawing has either two, three, or four corner boxes.*

Fact 7.2.2 *Any corner box contains either one or two corners.*

Figure 7.1(a) illustrates a box-rectangular drawing having two corner boxes; each of them is a real box and contains two corners. Figure 7.1(b) illustrates a box-rectangular drawing having three corner boxes. Figure 7.2(c) illustrates a box-rectangular drawing having four corner boxes, one of which is degenerate.

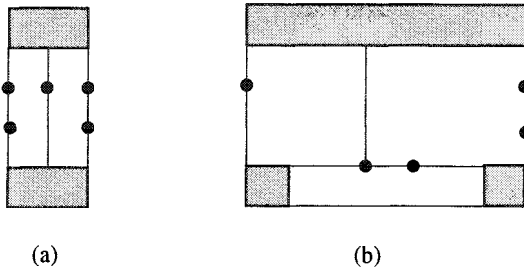


Fig. 7.1 (a) Two corner boxes, and (b) three corner boxes.

Fact 7.2.3 *In a box-rectangular drawing D of G , any vertex v of degree two or three satisfies one of the following (i), (ii) and (iii).*

- (i) *Vertex v is drawn as a point containing no corner;*
- (ii) *v is drawn as a corner box containing exactly one corner; and*
- (iii) *v is drawn as a corner real box containing exactly two corners.*

Fact 7.2.4 *In any box-rectangular drawing D of G , every vertex of degree five or more is drawn as a real box.*

Lemma 7.2.5 *If G has a box-rectangular drawing, then G has a box-rectangular drawing in which every vertex of degree four or more is drawn as a real box.*

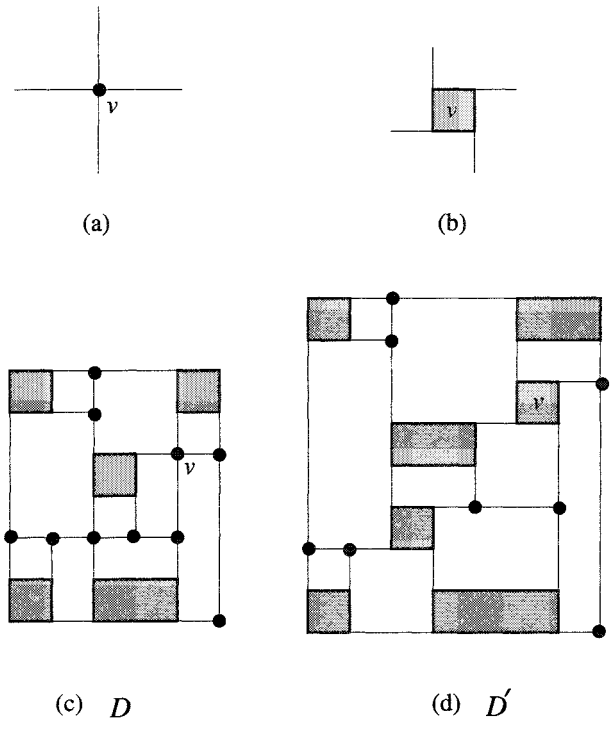


Fig. 7.2 Illustration for the proof of Lemma 7.2.5.

Proof. Assume that G has a box-rectangular drawing D . By Fact 7.2.4 every vertex of degree five or more in G must be drawn as a real box in D . If a vertex v of degree four in G is drawn as a point in D as illustrated in Fig. 7.2(a), then we modify the drawing D so that v is drawn as a real box, as illustrated in Fig. 7.2(b). Repeating the modification above for all vertices of degree four drawn as points in D , one can obtain a box-rectangular drawing D' of G in which every vertex of degree four or more is drawn as a real box, as illustrated in Figs. 7.2(c) and (d). \square

The choice of vertices as corner boxes plays an important role in finding a box-rectangular drawing. For example, the graph in Fig. 7.3(a) has a box-rectangular drawing if we choose outer vertices a, b, c and d as corner boxes as illustrated in Fig. 7.3(b). However, the graph has no box-rectangular drawing if we choose outer vertices p, q, r and s as corner boxes. If all vertices corresponding to corner boxes are designated for a drawing, then it is rather easy to examine whether G has a box-rectangular drawing with the designated corner boxes. We deal this case in Section 7.3. In Section 7.4 we deal with a general case where no vertices of G are designated as corner boxes.

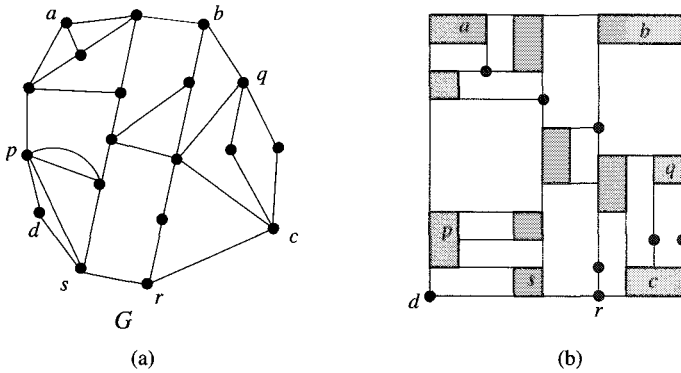


Fig. 7.3 A graph G and its box-rectangular drawing with four corner boxes a, b, c and d .

7.3 Box-Rectangular Drawings with Designated Corner Boxes

In this section we present a necessary and sufficient condition for a plane graph G to have a box-rectangular drawing D for a case where all vertices of G corresponding to corner boxes are designated, and give a simple linear-time algorithm to find D if there exists D [RNN00].

We now define two operations on a graph as follows. Let v be a vertex of degree two in G . We replace the two edges u_1v and u_2v incident to v with a single edge u_1u_2 and delete v . We call the operation above *the removal of a vertex of degree two* from G . Let v be a vertex of degree d in a plane graph, let $e_1 = vw_1, e_2 = vw_2, \dots, e_d = vw_d$ be the edges incident to v , and assume that these edges e_1, e_2, \dots, e_d appear clockwise

around v in this order as illustrated in Fig. 7.4(a). Replace v with a cycle $v_1, v_2, \dots, v_d, v_1$, and replace edge vw_i with v_iw_i for $i = 1, 2, \dots, d$, as illustrated in Fig. 7.4(b). We call the operation above the *replacement of a vertex by a cycle*. The cycle $v_1, v_2, \dots, v_d, v_1$ in the resulting graph is called the *replaced cycle* corresponding to vertex v .

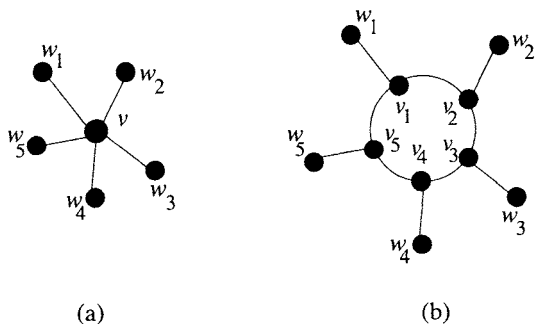


Fig. 7.4 Replacement of a vertex by a cycle.

By Fact 7.2.1, any box-rectangular drawing has either two, three or four corner boxes. However, we consider only box-rectangular drawings having four corner boxes for simplicity' sake, and assume that exactly four outer vertices a, b, c and d in G are designated as the four corner boxes. We construct a new graph G'' from G through an intermediate graph G' , and reduce the problem of finding a *box-rectangular drawing* of G with the four designated vertices to a problem of finding a *rectangular drawing* of G'' .

We first construct G' from G as follows. If a vertex v of degree two in G , as vertex d in Fig. 7.5(a), is designated as a corner, then v must be drawn as a corner point in a box-rectangular drawing of G . On the other hand, if a vertex v of degree two is not designated as a corner, then the two edges incident to v must be drawn on a straight line segment. We thus remove all non-designated vertices of degree two one by one from G , as illustrated in Fig. 7.5(b). The resulting graph is G' . Thus all vertices of degree two in G' are designated vertices.

Clearly, G has a box-rectangular drawing with the four designated corner boxes if and only if G' has a box-rectangular drawing with the four designated corner boxes. Figure 7.5(f) illustrates a box-rectangular drawing D' of G' in Fig. 7.5(b), and Fig. 7.5(g) illustrates a box-rectangular drawing D of G in Fig. 7.5(a).

Since every vertex of degree two in G' is a designated vertex, it must

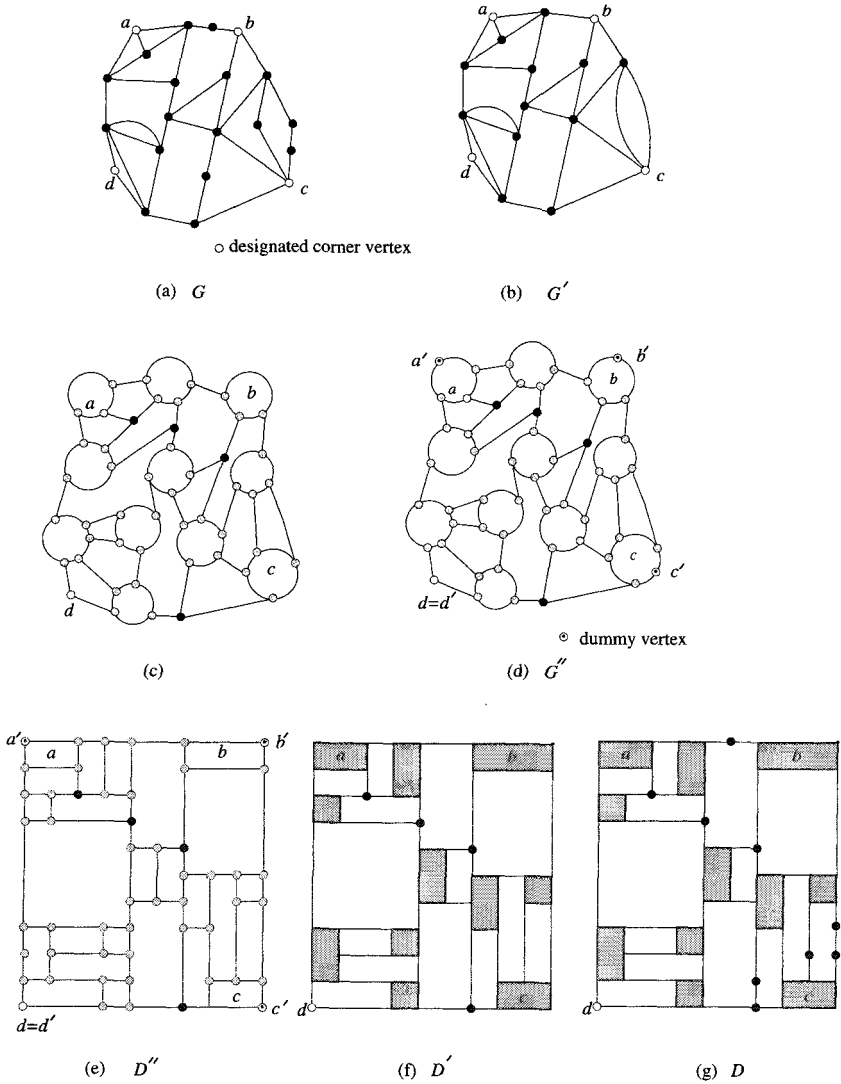


Fig. 7.5 Illustration of G, G', G'', D'', D' and D .

be drawn as a corner point in any box-rectangular drawing of G' . Every designated vertex of degree three in G' , as vertex a in Fig. 7.5(b), must be drawn as a real box since it is a corner. On the other hand, every non-designated vertex of degree three in G' must be drawn as a point. These facts together with Lemma 7.2.5 imply that if G' has a box-rectangular

drawing then G' has a box-rectangular drawing D' in which all designated vertices of degree three and all vertices of degree four or more in G' are drawn as real boxes.

We now construct G'' from G' . Replace by a cycle each of the designated vertices of degree three and the vertices of degree four or more, as illustrated in Fig. 7.5(c). The replaced cycle corresponding to a designated vertex x of degree three or more contains exactly one outer edge, say e_x , where $x = a, b, c$ or d . Put a dummy vertex x' of degree two on e_x , as illustrated in Fig. 7.5(d). The resulting graph is G'' . We let $x' = x$ if a designated vertex x has degree two. G'' is a simple graph and has exactly four outer vertices a', b', c' , and d' of degree two, and all the other vertices have degree three.

We now have the following theorem.

Theorem 7.3.1 *Let G be a plane graph with four designated outer vertices a, b, c and d , and let G'' be the graph transformed from G as mentioned above. Then G has a box-rectangular drawing with corner boxes a, b, c and d if and only if G'' has a rectangular drawing with designated corners a', b', c' and d' .*

Proof. The necessity is trivial, and hence it suffices prove the sufficiency.

Assume that G'' has a rectangular drawing D'' as illustrated in Fig. 7.5(e). In D'' , each replaced cycle is drawn as a rectangle, since it is a face in G'' . Furthermore, the four outer vertices a', b', c' and d' of degree two in G'' are drawn as the corners of the rectangle corresponding to $C_o(G'')$. Therefore, D'' immediately gives a box-rectangular drawing D' of G' having the four vertices a, b, c and d as corner boxes, as illustrated in Fig. 7.5(f). Then, inserting non-designated vertices of degree two on horizontal or vertical line segments in D' , one can immediately obtain from D' a box-rectangular drawing D of G having the designated vertices a, b, c and d as corner boxes, as illustrated in Fig. 7.5(g). \square

We now have the following theorem.

Theorem 7.3.2 *Given a plane graph G of m edges and four designated outer vertices a, b, c and d , one can examine in time $O(m)$ whether G has a box-rectangular drawing D with corner boxes a, b, c and d , and if G has D , then one can find D in time $O(m)$. The half perimeter of the box-rectangular grid drawing is bounded by $m + 2$.*

Proof. *Time Complexity.* Clearly one can construct G'' from G in linear time. G'' is a plane 2-connected graph such that all vertices have degree

three except the four outer vertices a' , b' , c' and d' of degree two. Therefore, by the algorithm in Section 6.3.3 one can examine in linear time whether G'' has a rectangular drawing D'' or not and find D'' if it exists. One can easily obtain a box-rectangular drawing D of G from D'' in linear time.

Grid size. Let n_2 be the non-designated vertices of degree two in G . Let $n' = |V(G')|$ and $m' = |E(G')|$. Then $m' = m - n_2$. We replace some vertices of G' by cycles and add at most four dummy vertices to construct G'' from G' . Therefore G'' has at most $2m' + 4$ vertices. From Theorem 6.3.8, the half perimeter of the produced rectangular drawing D'' is bounded by $\frac{2m'+4}{2} = m' + 2$. The insertion of each of the n_2 vertices of degree two on a horizontal line segment or a vertical line segment increases the half perimeter of the box-rectangular drawing by at most one. Thus the half perimeter of the produced box-rectangular drawing D of G is bounded by $m' + 2 + n_2 = m + 2$. \square

There are infinitely many examples for which half perimeter of any box-rectangular drawing is $m - 2$. Finding such examples is left for an exercise.

7.4 Box-Rectangular Drawings without Designated Corners

In Section 7.3 we considered a case where a set of outer vertices of a plane graph G are designated as corner boxes. In this section we consider a general case where no vertices are designated as corner boxes in advance. Then our problem is how to examine whether G has some set of outer vertices such that there is a box-rectangular drawing of G having them as the corner boxes, and how to find them if there are. We present a necessary and sufficient condition for G to have a box-rectangular drawing D for some set of outer vertices. The characterization leads to a linear-time algorithm to find D .

In Section 7.4.1, we first derive a necessary and sufficient condition for a plane graph G with $\Delta \leq 3$ to have a box-rectangular drawing D , and then give a linear-time algorithm to find D if it exists. In Section 7.4.2 we reduce the box-rectangular drawing problem of a plane graph G with $\Delta \geq 4$ to that of a new plane graph J with $\Delta \leq 3$.

7.4.1 Box-Rectangular Drawings of G with $\Delta \leq 3$

Let G be a plane graph with the maximum degree Δ at most three. The following theorem is a main result of Section 7.4.1.

Theorem 7.4.1 *A plane graph G with $\Delta \leq 3$ has a box-rectangular drawing (for some set of outer vertices designated as corner boxes) if and only if G satisfies the following two conditions:*

- (br1) every 2-legged or 3-legged cycle in G contains an outer edge; and
- (br2) $2c_2 + c_3 \leq 4$ for any independent set S of cycles in G , where c_2 and c_3 are the numbers of 2-legged cycles and 3-legged cycles in S , respectively.

Before proving the necessity of Theorem 7.4.1, we observe the following fact.

Fact 7.4.2 *In a box-rectangular drawing D of G , any 2-legged cycle of G contains at least two corners of the outer rectangle, any 3-legged cycle of G contains at least one corner, and any cycle with four or more legs may contain no corner. (See Fig. 6.5.)*

We now prove the necessity of Theorem 7.4.1.

[Necessity of Theorem 7.4.1.]

Assume that G has a box-rectangular drawing D . By Fact 7.4.2 any 2-legged or 3-legged cycle in D contains a corner of the outer rectangle, and hence contains an outer edge.

Let S be any independent set of cycles in G . Then by Fact 7.4.2 each of the c_2 2-legged cycles in S contains at least two corners, and each of the c_3 3-legged cycles in S contains at least one corner. Since all cycles in S are independent, they are vertex-disjoint with each other. Therefore there are at least $2c_2 + c_3$ corners of the outer rectangle. Since the outer rectangle has exactly four corners, we have $2c_2 + c_3 \leq 4$. \square

In the rest of this section we give a constructive proof for the sufficiency of Theorem 7.4.1, and show that the proof leads to a linear-time algorithm to find a box-rectangular drawing of G if it exists.

One can easily prove the following lemma (Exercise 3).

Lemma 7.4.3 *Let G be a plane graph with $\Delta \leq 3$. Assume that G satisfies Conditions (br1) and (br2) in Theorem 7.4.1, and that G has at*

most three outer vertices of degree three. Then G has a box-rectangular drawing.

By Lemma 7.4.3 we may assume that G has four or more outer vertices of degree three. In this case, we can choose four distinct outer vertices of degree three as the corner boxes. Since any vertex v of degree two is not chosen as corner boxes, the two edges incident to v are drawn on a common straight line segment in a box-rectangular drawing of G . Let G' be a plane cubic graph obtained from G by removing all vertices of degree two one by one. Then one can immediately construct a box-rectangular drawing of G from any box-rectangular drawing of G' . We may thus assume that our input graph G itself is a plane cubic graph with four or more outer vertices.

We now recall some definitions from Chapter 2. For a graph G and a set $V' \subseteq V(G)$, $G - V'$ denotes a graph obtained from G by deleting all vertices in V' together with all edges incident to them. For a plane graph G , we define a $C_o(G)$ -component as follows. A subgraph J of G is a $C_o(G)$ -component if J consists of a single inner edge joining two outer vertices. The graph $G - V(C_o(G))$ may have a connected component. Add to the component all edges of G , each joining a vertex in the component and an outer vertex. The resulting subgraph J of G is a $C_o(G)$ -component, too. All these subgraphs J of G and only these are the $C_o(G)$ -components. The $C_o(G)$ -components J_1 , J_2 and J_3 of G in Fig. 2.9(a) are depicted in Fig. 2.9(b). We say that cycles C and C' in a plane graph G are *independent* if $G(C)$ and $G(C')$ have no common vertex. A k -legged cycle C is *minimal* if $G(C)$ does not contain any other k -legged cycle of G . We now have the following lemmas.

Lemma 7.4.4 *Let G be a plane cubic graph. Assume that G satisfies Condition (br1) in Theorem 7.4.1, that G has four or more outer vertices, and that there is exactly one $C_o(G)$ -component. Then*

- (a) G has a 3-legged cycle; and
- (b) if G has two or more independent 3-legged cycles, then the set of all minimal 3-legged cycles in G is independent.

Proof. (a) Let w be any outer vertex, and let e be the inner edge which is incident to w . Let x be the other end of e . Then x is an inner vertex, because G has exactly one $C_o(G)$ -component and G has four or more outer vertices. The edge e is contained in the contours of exactly two faces of G , say F_1 and F_2 . Since the cubic graph G has exactly one $C_o(G)$ -component, the contour of F_1 contains exactly two outer vertices: w and another vertex,

say, y . Similarly, the contour of F_2 contains exactly two outer vertices: w and another vertex, say, z . Clearly $y \neq z$ since $d(y) = d(z) = 3$. Thus G has a 3-legged cycle C with leg-vertices x, y and z .

(b) Assume for a contradiction that G has two or more independent 3-legged cycles but two minimal 3-legged cycles C and C' of G are not independent. Then $G(C)$ and $G(C')$ share a common vertex. Let e_1, e_2 and e_3 be the legs of C , and let e'_1, e'_2 and e'_3 be the legs of C' .

We now claim that $G(C)$ and $G(C')$ do not share any common face. Suppose for a contradiction that $G(C)$ and $G(C')$ share a common face. Then, since both C and C' contain an outer edge by Condition (br1), exactly two of the three legs of C , say e_1 and e_2 , are outer edges, and exactly two of the three legs of C' , say e'_1 and e'_2 , are outer edges. Since both C and C' are minimal 3-legged cycles, e_1, e_2 and e_3 share a same end and e'_1, e'_2 and e'_3 share a same end. Furthermore G has four faces F_1, F_2, F_3 and F_4 ; their contours contain e_1 and e_3, e_2 and e_3, e'_1 and $e'_3,$ and e'_2 and $e'_3,$ respectively; and any of the four contours is not a 3-legged cycle, since both C and C' are minimal 3-legged cycles. Thus G has a subgraph illustrated in Fig. 7.6(a), where C and C' are drawn by thick solid lines and thick dotted lines, respectively. All edges of C and C' not in the shaded region in Fig. 7.6(a) are on $C_o(G)$. There is no 3-legged cycle other than C and C' ; if there was a 3-legged cycle other than C and C' , then it would be contained in the shaded region in Fig. 7.6(a) and C would not be a minimal 3-legged cycle. Since there is no 3-legged cycle other than C and C' , G does not have two or more independent 3-legged cycles, a contradiction.

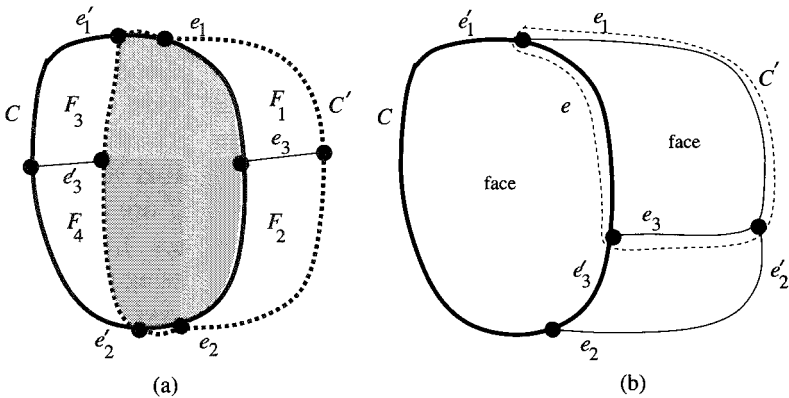


Fig. 7.6 Illustration for the proof of Lemma 7.4.4(b).

Since G is cubic, C and C' are 3-legged cycles and $G(C)$ and $G(C')$ do not share any common face, one can observe that $G(C)$ and $G(C')$ share an edge e on C and C' , as illustrated in Fig. 7.6(b). Since G has exactly one $C_o(G)$ -component J , J contains all edges of C and C' that are inner edges of G . The cycle C is a contour of a face in G ; otherwise, either C would not be a minimal 3-legged cycle or G would have two or more $C_o(G)$ -components. Similarly, C' is a contour of a face in G . Moreover, every edge of J is on C or C' ; otherwise, C or C' would have four or more legs. Hence J has exactly three outer vertices, and G is K_4 as illustrated in Fig. 7.6(b). Therefore G has exactly three outer vertices, a contradiction. \square

We are now ready to prove the following lemma.

Lemma 7.4.5 *Let G be a plane cubic graph. Assume that G satisfies Conditions (br1) and (br2) in Theorem 7.4.1, and that G has four or more outer vertices. Then G has a box-rectangular drawing.*

Proof. We first claim that if G has a 2-legged cycle C then G has a pair of independent 2-legged cycles. By Condition (br1) C contains an outer edge, and hence the two legs of C are outer edges, say (v, v') and (w, w') . One may assume that v and w are the leg-vertices of C . Clearly $v' \neq w'$ and G has a 2-legged cycle C' which has v' and w' as the leg-vertices. C and C' are independent.

Thus we shall consider the following two cases.

Case 1: G has no 2-legged cycle.

In this case G has exactly one $C_o(G)$ -component; otherwise, G would have a 2-legged cycle. Then by Lemma 7.4.4(a) G has a 3-legged cycle. We choose four outer vertices as the four corner boxes for a box-rectangular drawing of G , as follows.

We first consider the case where G has no pair of independent 3-legged cycles. We arbitrarily choose four outer vertices and regard them as the four designated vertices for a box-rectangular drawing of G . We now claim that every 3-legged cycle C in G has at least one designated vertex. Since C has an outer edge, exactly two of the three legs of C are outer edges of G . Let x and y be the two leg-vertices of the two legs. Let P be the path on $C_o(G)$ starting at x and ending at y without passing through any edge on C . Then P has exactly one intermediate vertex, say z : otherwise, either G would have more than one $C_o(G)$ -components or G would have a pair of independent 3-legged cycles, a contradiction. Thus one can easily know that all three legs of C are incident to z . Therefore, all the outer vertices except z lie on C . Hence, regardless of whether z is one of the four

designated vertices or not, C contains at least one of the four designated vertices.

We then consider the case where G has a pair of independent 3-legged cycles. Let \mathcal{M} be the set of all minimal 3-legged cycles in G . By Lemma 7.4.4(b) \mathcal{M} is independent. Let $k = |\mathcal{M}|$, then $k \leq 4$ by Condition (br2). For each 3-legged cycle C_j , $1 \leq j \leq k$, in \mathcal{M} , we arbitrarily choose an outer vertex on C_j . If $k < 4$, we arbitrarily choose $4 - k$ outer vertices which are not chosen so far. Thus we have chosen exactly four outer vertices, and we regard them as the four designated vertices for a box-rectangular drawing of G . In Fig. 7.7(a) four outer vertices a, b, c and d are chosen as the designated vertices; vertices a, b and d are chosen on three independent minimal 3-legged cycles indicated by dotted lines, whereas vertex c is chosen arbitrarily on $C_o(G)$. We now claim that every 3-legged cycle C of G has at least one designated vertex. Clearly C contains a designated vertex if C is minimal, that is, $C \in \mathcal{M}$. Thus one may assume C is not minimal. Then $G(C)$ contains a minimal 3-legged cycle $C_j \in \mathcal{M}$, and C_j has a designated outer vertex. Of course, the vertex is also on C .

Thus we have chosen four designated outer vertices. We now give a method to find a box-rectangular drawing of G with the four designated vertices. We replace each of the four designated vertices by a cycle, and put a dummy vertex of degree two on the edge of the cycle that is contained in the outer cycle. Let G' be the resulting graph. (See Fig. 7.7(b) where the dummy vertices of degree two are drawn by white circles.) G' has exactly four outer vertices of degree two, and all other vertices of G' have degree three. One can easily observe that G' satisfies Conditions (r1) and (r2) of Theorem 6.3.2 with the four vertices of degree two designated as the four corners of a rectangular drawing of G' . Thus by Theorem 6.3.2 G' has a rectangular drawing D' as illustrated in Fig. 7.7(c). Regarding the four faces in D' corresponding to the replaced cycles as boxes, we obtain a box-rectangular drawing D of G from D' as illustrated in Fig. 7.7(d).

Case 2: G has a pair of independent 2-legged cycles.

Let C_1 and C_2 be independent 2-legged cycles in G . One may assume that both C_1 and C_2 are minimal 2-legged cycles. By Condition (br2) at most two 2-legged cycles of G are independent. Therefore, for any other 2-legged cycle $C' (\neq C_1, C_2)$, $G(C')$ contains either C_1 or C_2 .

Let k_i , $i = 1$ or 2 , be the number of all minimal (not always independent) 3-legged cycles in $G'(C_i)$. Then we claim that $k_i \leq 2$. First consider the case where C_i has exactly three outer vertices. Then $G(C_i)$ has exactly two inner faces; otherwise, $G(C_i)$ would have a cycle which has two or three legs

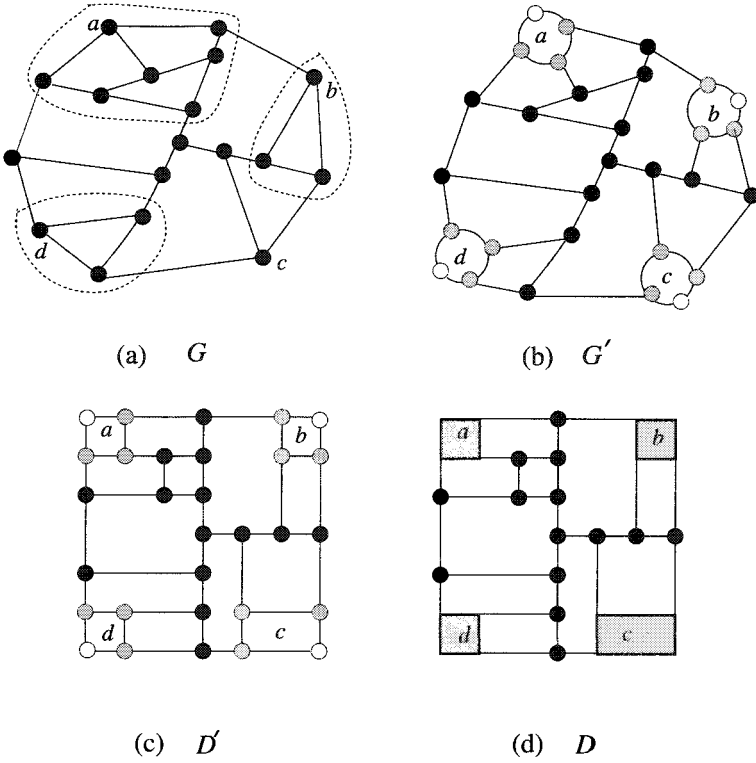


Fig. 7.7 Illustration for Case 1.

and has no outer edge, contrary to Condition (br1). The contour of the two faces are minimal 3-legged cycles, and there is no other minimal 3-legged cycle in $G(C_i)$. Thus $k_i = 2$ in this case. We next consider the case where C_i has four or more outer vertices. Then we can show, similarly as in the proof of Lemma 7.4.4(b), that the set of all minimal 3-legged cycles of G in $G(C_i)$ is independent. Therefore, $k_i \leq 2$ in this case; otherwise, Condition (br2) would not hold for the independent set S of $k_i + 1$ cycles: the k_i (≥ 3) 3-legged cycles in $G(C_i)$ and the 2-legged cycle C_j , $j = 1$ or 2 and $j \neq i$.

We choose two vertices on each C_i , $1 \leq i \leq 2$, as follows. For each of the k_i minimal 3-legged cycle in $G(C_i)$, we arbitrarily choose exactly one outer vertex on it. If $k_i < 2$, then we arbitrarily choose $2 - k_i$ outer vertices on C_i which have not been chosen so far. This can be done because C_i has at least two outer vertices. Thus we have chosen four outer vertices, and we regard them as the designated vertices for a box-rectangular drawing

of G . In Fig. 7.8(a) G has a pair of independent 2-legged cycles C_1 and C_2 , $k_1 = 2$, $k_2 = 1$, and four outer vertices a, b, c and d are chosen as the designated vertices. Vertices a and d are chosen from the vertices on C_1 ; each on a minimal 3-legged cycle in $G(C_1)$. Vertices b and c are chosen from the vertices on C_2 ; b is on a minimal 3-legged cycle in $G(C_2)$, and c is an arbitrary outer vertex on C_2 other than b .

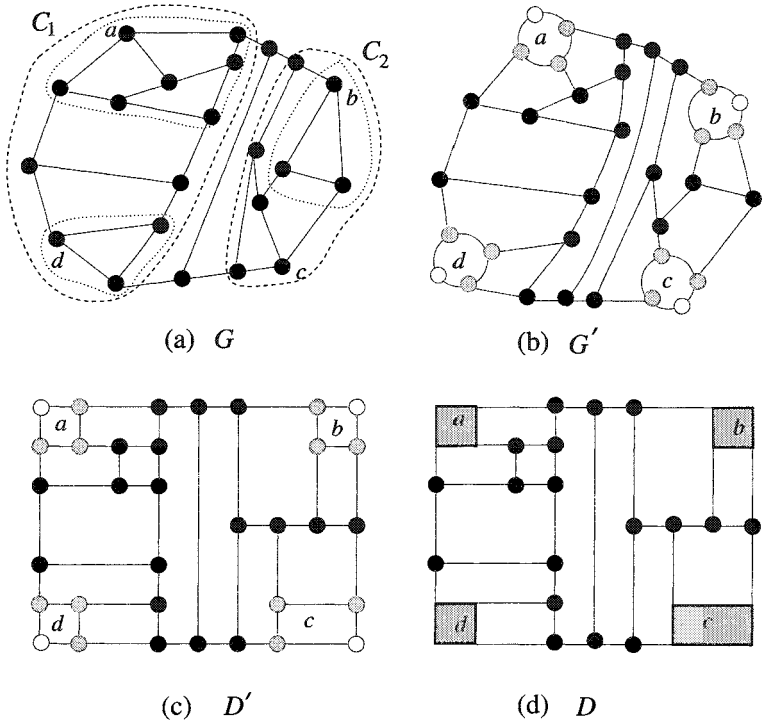


Fig. 7.8 Illustration for Case 2.

We now claim that any 2-legged cycle C in G has two designated vertices. If C is C_1 or C_2 , then clearly C has exactly two designated vertices. Otherwise, $G(C)$ contain either cycle C_1 or C_2 , and hence C has exactly two designated vertices. We then claim that any 3-legged cycle C_3 in G has a designated vertex. By Condition (br2) $\{C_1, C_2, C_3\}$ is not independent, and hence either $G(C_3)$ contains C_1 or C_2 , or C_3 is contained in $G(C_1)$ or $G(C_2)$. If $G(C_3)$ contains C_1 or C_2 , then C_3 contains a designated vertex. Otherwise, C_3 is contained in either $G(C_1)$ or $G(C_2)$. In this case C_3 con-

tains a designated vertex, since we have chosen a designated vertex on each minimal 3-legged cycle inside $G(C_1)$ and $G(C_2)$.

We can find a box-rectangular drawing of G as follows. We replace each of the four designated vertices by a cycle, and put a dummy vertex of degree two on the edge of the cycle which is on the contour of the outer face, as illustrated in Fig. 7.8(b). Let G' be the resulting graph, then G' has exactly four outer vertices of degree two, and all other vertices of G' have degree three. One can easily observe that G' satisfies Conditions (r1) and (r2) of Theorem 6.3.2 with the four vertices of degree two designated as the four corners of a rectangular drawing of G' . Thus G' has a rectangular drawing D' by Theorem 6.3.2, as illustrated in Fig. 7.8(c). Regarding the four faces in D' corresponding to the replaced cycles as boxes, we obtain a box-rectangular drawing D of G from D' as illustrated in Fig. 7.8 (d). \square

Using Lemmas 7.4.3 and 7.4.5, one can find a box-rectangular drawing of G if G satisfies the conditions in Theorem 7.4.1. Thus we have constructively proved the sufficiency of Theorem 7.4.1.

The proof of Lemma 7.4.5 imply the following corollary.

Corollary 7.4.6 *A plane graph G with $\Delta \leq 3$ has a box-rectangular drawing if and only if G satisfies the following four conditions:*

- (c1) every 2-legged or 3-legged cycle in G has an outer edge;
- (c2) at most two 2-legged cycles of G are independent of each other;
- (c3) at most four 3-legged cycles of G are independent of each other; and
- (c4) if G has a pair of independent 2-legged cycles C_1 and C_2 , then $\{C_1, C_2, C_3\}$ is not independent for any 3-legged cycle C_3 in G , and neither $G(C_1)$ nor $G(C_2)$ has more than two independent 3-legged cycles of G .

We now have the following theorem.

Theorem 7.4.7 *Given a plane graph with $\Delta \leq 3$, one can examine in time $O(m)$ whether G has a box-rectangular drawing D or not, and if G has D , one can find D in time $O(m)$, where m is the number of edges in G .*

Proof. One can find all 2-legged and 3-legged cycles in G , as follows. We first traverse the contour of each inner face of G containing an outer edge as illustrated in Fig. 7.9, where the traversed contours of faces are indicated by dotted lines. Clearly each outer edge is traversed exactly once, and each inner edge is traversed at most twice. The inner edges traversed exactly

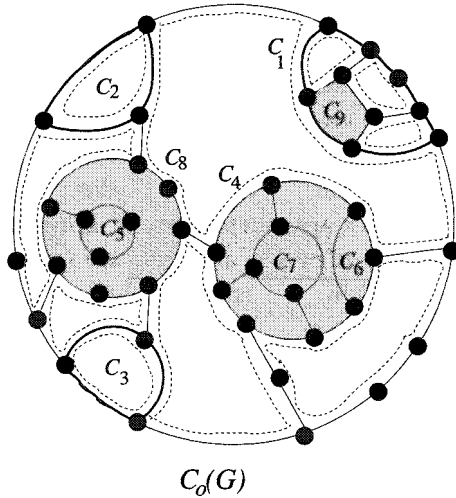


Fig. 7.9 Finding all 2-legged and 3-legged cycles.

once form cycles, called *singly traced cycles*, the insides of which have not been traversed. In Fig. 7.9 C_4 , C_8 and C_9 are singly traced cycles, the insides of which are shaded. During this traversal one can easily find all 2-legged and all 3-legged cycles that contain outer edges; C_1 , C_2 and C_3 drawn by thick lines in Fig. 7.9 are some of these cycles. (Note that a 3-legged cycle containing outer edges has two legs on $C_o(G)$ and the other leg is an inner edge which is traversed twice; if an end of a doubly traversed inner edge is an inner vertex, then it is a leg-vertex of such a 3-legged cycle.) Any of the remaining 2-legged and 3-legged cycles either is a singly traced cycle or is located inside a singly traced cycle. One can find all 2-legged and 3-legged cycles inside a singly traced cycle by recursively applying the method to the singly traced cycle. The method traverses the contour of each face by a constant number of times. Hence we can examine in time $O(m)$ whether G satisfies Condition (c1) in Corollary 7.4.6 or not.

We examine Condition (c2) in Corollary 7.4.6 as follows. Assume that G satisfies Condition (c1). Then each 2-legged cycle must have an outer edge, and hence has the two leg-vertices on $C_o(G)$. By traversing the faces of G containing an outer edge, one can detect the leg-vertices of all 2-legged cycles of G on $C_o(G)$. While detecting the leg-vertices of 2-legged cycles, we give labels to the two leg-vertices of each 2-legged cycle; the labels indicate the name of the cycle. In Fig. 7.10, the leg-vertices of 2-legged cycles are

drawn by white circles, and their labels are written next to them. It is clear that if G has k 2-legged cycles which are independent of each other then G has k minimal 2-legged cycles which are independent of each other. A 2-legged cycle C is minimal if and only if no intermediate vertex of the maximal subpath of C on $C_o(G)$ is a leg-vertex of any other 2-legged cycle. Therefore, traversing the outer vertices and checking the labels of leg-vertices, we can find all minimal 2-legged cycles, and we can also know whether two 2-legged cycles are independent or not. In Fig. 7.10 C_1, C_2 and C_3 are minimal 2-legged cycles, and they are independent. Thus we can examine Condition (c2) by traversing the edges on the contours of faces containing an outer edge by a constant number of times, and hence we can examine Condition (c2) in linear time.

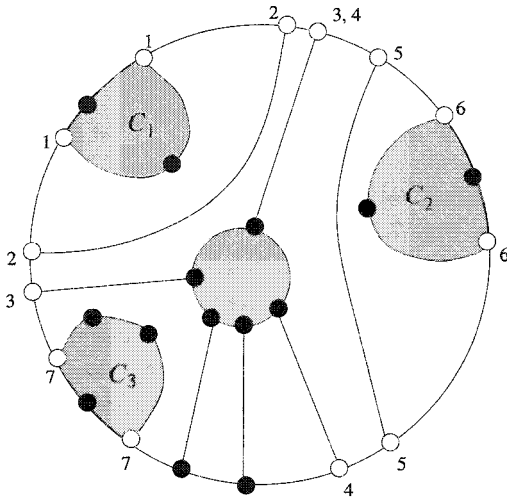


Fig. 7.10 Illustration for minimal 2-legged cycles.

We can examine Condition (c3) in linear time using a similar technique used to examine Condition (c2). One can easily examine Condition (c4) by checking the labels of the leg-vertices of minimal 2-legged cycles and minimal 3-legged cycles.

If G satisfies the conditions in Corollary 7.4.6, then a box-rectangular drawing of G can be found by the algorithm described in the proof of Lemma 7.4.5. One can find all minimal 2-legged cycles and all minimal 3-legged cycles in linear time by the technique used to examine Conditions (c2) and (c3), and hence one can choose the four designated vertices in

linear time. Thus one can find a box-rectangular drawing of G in linear time. \square

7.4.2 Box-Rectangular Drawings of G with $\Delta \geq 4$

In this section we give a necessary and sufficient condition for a plane graph with $\Delta \geq 4$ to have a box-rectangular drawing when no vertices are designated as corner boxes.

Let G be a plane graph with $\Delta \geq 4$. We construct a new plane graph J from G by replacing each vertex v of degree four or more in G by a cycle. Figures 7.11(a) and (b) illustrate G and J , respectively. A replaced cycle corresponds to a real box in a box-rectangular drawing of G . We do not replace a vertex of degree two or three by a cycle since such a vertex may be drawn as a point by Fact 7.2.3. Thus $\Delta(J) \leq 3$. Then the following theorem holds.

Theorem 7.4.8 *Let G be a plane graph with $\Delta \geq 4$, and let J be the graph transformed from G as above. Then G has a box-rectangular drawing if and only if J has a box-rectangular drawing.*

We present a proof for the necessity of Theorem 7.4.8. We omit the proof for the sufficiency, which can be found in [RNN00].

[Necessity of Theorem 7.4.8.]

Assume that G has a box-rectangular drawing. Then by Lemma 7.2.5 G has a box-rectangular drawing D in which every vertex of degree four or more is drawn as a real box, as illustrated in Fig. 7.11(c) for the graph G in Fig. 7.11(a). For simplicity's sake, we assume that D has four corner boxes. Then, as illustrated in Fig. 7.11(d), one can obtain a box-rectangular drawing D_J of J from D by the following transformation:

- (i) regard each non-corner real box in D as a face in D_J ;
- (ii) if a corner box in D is a vertex of degree three in G , then regard it as a corner box in D_J ; and
- (iii) if a corner box in D is a vertex of degree four or more in G , then transform it to a drawing of a replaced cycle with one real box as illustrated in Fig. 7.11(e). \square

Figures 7.11(c) and (d) illustrate D and D_J , respectively. Box f in D is a non-corner real box, and it is regarded as a face in D_J . Corner boxes

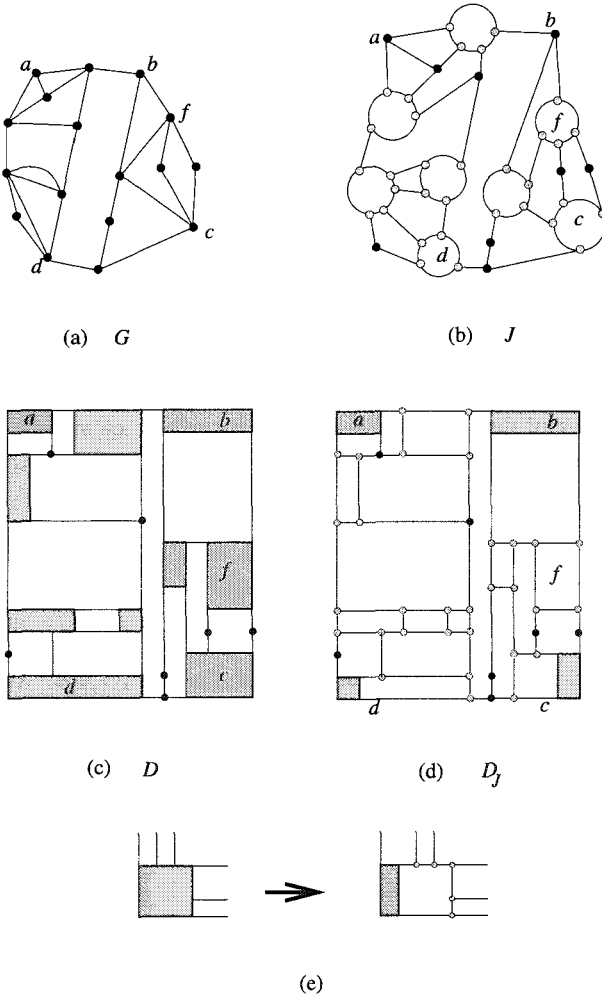


Fig. 7.11 Illustration of G, J, D_J, D and a transformation.

a and b in D are vertices of degree three in G , and they remains as boxes in D_J . Corner boxes c and d in D are vertices of degree four or more in G , and are transformed to a drawing of a replaced cycle with one real box in D_J as illustrated in Fig. 7.11(e).

7.5 Bibliographic Notes

A similar concept of a box-rectangular drawing, called a strict 2-box drawing, is presented by Thomassen in [Tho86]. A polynomial time algorithm can be designed for finding a strict 2-box drawing of a graph by following his method.

A box-rectangular drawing of G is called a *proper box-rectangular drawing* if every vertex of G is drawn as a real box, i.e., no vertex of G is drawn as a degenerate box. Xin He [He01] presents a necessary and sufficient condition for a plane graph G to have a proper box-rectangular drawing and gives a linear algorithm for finding a proper box-rectangular drawing of G if it exists.

Exercise

1. Find an infinite number of plane graphs for which the half perimeter of any box-rectangular grid drawing is $m - 2$, where m is the number of edges.
2. Obtain an algorithm to find a box-rectangular drawing of a given plane graph G for a case where exactly two or three outer vertices are designated as the corner boxes [RNN00].
3. Give a proof of Lemma 7.4.3.
4. Show that if a plane graph G has a proper box-rectangular drawing then G has no non-corner vertex of degree two or three.
5. Show that the algorithms presented in this chapter always find a proper box-rectangular drawing of a plane graph G if G has no vertex of degree two or three.

This page intentionally left blank

Chapter 8

Orthogonal Drawing

8.1 Introduction

An *orthogonal drawing* of a plane graph G is a drawing of G , with the given embedding, in which each vertex is mapped to a point, each edge is drawn as a sequence of alternate horizontal and vertical line segments, and any two edges do not cross except at their common end, as illustrated in Fig. 8.1. Orthogonal drawings have numerous practical applications in circuit schematics, data flow diagrams, entity relationship diagrams, etc., as mentioned in Chapter 1. Clearly the maximum degree Δ of G is at most four if G has an orthogonal drawing. Conversely, every plane graph with $\Delta \leq 4$ has an orthogonal drawing, but may need *bends*, that is, points where an edge changes its direction in a drawing. For the cubic plane graph in Fig. 8.1(a), two orthogonal drawings are depicted in Figs. 8.1(b) and (c), which have six and five bends, respectively. If a graph corresponds to a VLSI circuit, then one may be interested in an orthogonal drawing such that the number of bends is as small as possible, because bends increase the manufacturing cost of a VLSI chip. However, for a given planar graph G , if one is allowed to choose its planar embedding, then finding an orthogonal drawing of G with the minimum number of bends is NP-complete [GT01]. On the other hand, Tamassia [Tam87] and Garg and Tamassia [GT97] presented algorithms which find an orthogonal drawing of a given plane graph G with the minimum number $b(G)$ of bends in time $O(n^2 \log n)$ and $O(n^{7/4} \sqrt{\log n})$ respectively, where G has a fixed planar embedding and one is not allowed to alter the planar embedding. Such a drawing is called a *bend-optimal* orthogonal drawing of a plane graph G . They reduce the problem of finding a bend-optimal orthogonal drawing of G to a minimum cost flow problem. Rahman *et al.* gave a linear algorithm to find a bend-

optimal orthogonal drawing for 3-connected cubic plane graphs [RNN99], and Rahman and Nishizeki gave a linear algorithm to find a bend-optimal orthogonal drawing for plane graphs with $\Delta \leq 3$ [RN02].

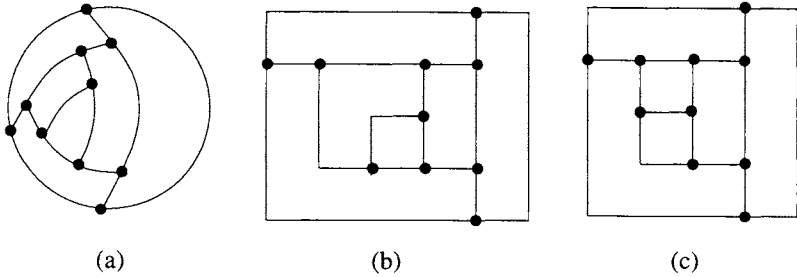


Fig. 8.1 (a) A plane graph G , (b) an orthogonal drawing of G with 6 bends, and (c) an orthogonal drawing of G with 5 bends.

In Section 8.2 we describe a network flow model for finding a bend-optimal orthogonal drawing of a plane graph with $\Delta \leq 4$ [Tam87, GT97]. In Section 8.3 we give a linear algorithm for finding a bend-optimal orthogonal drawing of plane 3-connected cubic graphs [RNN99], which depicts the key idea behind the linear algorithm for plane graphs with $\Delta \leq 3$ [RN02]. In Section 8.4 we deal with orthogonal grid drawings. In Section 8.5 we present a necessary and sufficient condition for a plane graph with $\Delta \leq 3$ to have an orthogonal drawing without any bends [RNN03].

8.2 Orthogonal Drawing and Network Flow

In this section we describe a network flow model for an orthogonal drawing of a plane graph. In Section 8.2.1 we describe an orthogonal representation of an orthogonal drawing of a plane graph, in Section 8.2.2 we give some definitions related to a flow network, and in Section 8.2.3 we give a network flow model for finding a bend-optimal orthogonal drawing of a plane graph.

8.2.1 Orthogonal Representation

Let G be a plane connected graph with $\Delta \leq 4$. The topological structure of G can be described by listing edges that appear on the contour of each face, and by specifying the outer face. A *planar representation* P of a plane graph G is a set of circularly ordered edge lists $P(F)$, one for each face F .

Edges in a list $P(F)$ appear as they are encountered when going around the contour of F in the “positive” direction, i.e. having the face at one’s right. Note that every edge of G appears exactly twice in lists. If this happens in the same list $P(F)$, the edge is called a *bridge*. For the plane graph G in Fig. 8.2(a), a planar representation P is depicted in Fig. 8.2(c), and edge e_7 is a bridge in G .

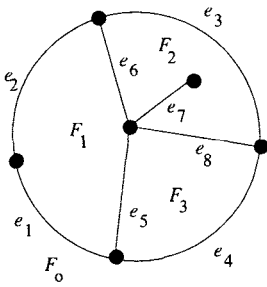
Let D be an orthogonal drawing of G like one in Fig. 8.2(b). Then each face of G is drawn in D as a rectilinear polygon. Note that a facial polygon is not always a simple polygon. There are two types of angles in D . We call an angle formed by two edges incident to a vertex a *vertex-angle*, and call an angle formed by two line segments at a bend a *bend-angle*. Clearly both a vertex-angle and a bend-angle are $k \cdot 90^\circ$ for some integer k , $1 \leq k \leq 4$. We now have the following two facts.

Fact 8.2.1 *The sum of the vertex-angles around any vertex is 360° .*

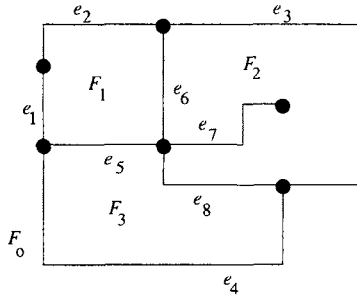
Fact 8.2.2 *The sum of the angles inside any facial polygon is $(2p-4)90^\circ$, and the sum of the angles of the outer facial polygon is $(2p+4)90^\circ$, where p is the number of angles of the polygon.*

We now introduce a concept of an orthogonal representation R of an orthogonal drawing D in terms of bends occurring along edges and of angles formed by edges. This orthogonal representation is obtained by enriching the lists of the planar representation with information about bends and angles formed by edges. An *orthogonal representation* R of D is a set of circularly ordered lists $R(F)$, one for each face F of G . Each element r of a list is a triple (e_r, s_r, a_r) ; e_r is an edge, s_r is a bit string, and a_r is an integer in the set $\{90, 180, 270, 360\}$. The bit string s_r provides information about the bends along edge e_r ; the k th bit of s_r describes the k th bend on the right side of e_r ; bit 0 indicates a 90° bend, and bit 1 indicates a 270° bend. An empty string ϵ is used to characterize a straight line edge. The number a_r specifies the angle formed in face F by edges e_r and $e_{r'}$, where r' is the element following r in the circular list $R(F)$. Figure 8.2(d) depicts an orthogonal representation R of the orthogonal drawing D in Fig. 8.2(b). Clearly R preserves only the shape of D without considering lengths of line segments, and hence describes actually an equivalence class of orthogonal drawings of G with “similar shape,” that is, with the same lists of triples r for the edges of G .

For a set R of circular lists to be an orthogonal representation of an orthogonal drawing D of a plane graph G , the following properties are



(a) G



(b) D

$$P(F_1) = (e_1, e_2, e_6, e_5)$$

$$P(F_2) = (e_3, e_8, e_7, e_6)$$

$$P(F_3) = (e_5, e_8, e_4)$$

$$P(F_0) = (e_1, e_4, e_3, e_2)$$

(c) P

$$R(F_1) = ((e_1, \epsilon, 180), (e_2, 0, 90), (e_6, \epsilon, 90), (e_5, \epsilon, 90))$$

$$R(F_2) = ((e_3, 00, 180), (e_8, 0, 90), (e_7, 10, 360), (e_7, 10, 90), (e_6, \epsilon, 90))$$

$$R(F_3) = ((e_5, \epsilon, 90), (e_8, 1, 90), (e_4, 00, 90))$$

$$R(F_0) = ((e_1, \epsilon, 180), (e_4, 11, 90), (e_3, 11, 180), (e_2, 1, 180))$$

(d) R

Fig. 8.2 (a) A plane graph G , (b) an orthogonal drawing D of G , (c) a planar representation P of G , and (d) an orthogonal representation R of D .

necessary and sufficient, as can be proved by means of elementary geometric considerations.

- (p1) There is some planar graph whose planar representation is given by the e -fields of the lists in R .
- (p2) For each pair of elements r and r' in R with $e_r = e_{r'}$, string $s_{r'}$ can be obtained by applying bitwise negation to the reversion of s_r .
- (p3) For each element r in R , define the rotation $\rho(r)$ as follows:

TEAM LING - LIVE, INFORMATIVE, NON-COST AND GENUINE !

$$\rho(r) = |s_r|_0 - |s_r|_1 + (2 - \frac{a_r}{90}) \tag{8.1}$$

where $|s_r|_0$ is the number of zeros in string s_r and $|s_r|_1$ the number of ones. Then

$$\sum_{r \in R(F)} \rho(r) = \begin{cases} +4 & \text{if } F \text{ is an inner face;} \\ -4 & \text{if } F \text{ is the outer face } F_o. \end{cases} \tag{8.2}$$

(p4) For each vertex $v \in V$, the sum of the vertex-angles around v given by the a -fields in R is equal to 360° .

Property (p2) means that each edge must have consistent descriptions in the faces in which it appears. Fact 8.2.2 implies Property (p3), which means that every face described by R is a rectilinear polygon. Fact 8.2.1 implies Property (p4).

We say that an orthogonal drawing D of a plane graph G realizes an orthogonal representation R if R is a valid description for the shape of D . Figure 8.2(d) depicts an orthogonal representation R of the plane graph G in Fig. 8.2(a), and Fig. 8.2(b) depicts an orthogonal drawing D realizing R . Note that the number $b(R)$ of bends in any orthogonal drawing D that realizes R is

$$b(R) = \frac{1}{2} \sum_{F \in \mathcal{F}} \sum_{r \in R(F)} |s_r|, \tag{8.3}$$

where $|s_r|$ is the number of bits in string s_r and \mathcal{F} is the set of all faces in G .

8.2.2 Flow Network

A *flow network* \mathcal{N} is a directed graph such that \mathcal{N} has two disjoint non-empty sets of distinguished nodes called its *sources* and *sinks*, and each arc e of \mathcal{N} is labeled with three nonnegative integers

- a *lower bound* $\lambda(e)$,
- a *capacity* $\mu(e)$, and
- a *cost* $c(e)$.

A *flow* ϕ in \mathcal{N} associates a nonnegative integer $\phi(e)$ with each arc e ; $\phi(e)$ is called a *flow of arc* e . The flow $\phi(e)$ of each arc e must satisfy $\lambda(e) \leq \phi(e) \leq \mu(e)$. Furthermore, ϕ must satisfy the so-called conservation

law as follows. For each node u of \mathcal{N} that is neither a source nor a sink, the sum of the flows of the outgoing arcs from u must be equal to the sum of the flows of the incoming arcs to u . Each source u has a *production* $\sigma(u) \geq 0$ of flow, and each sink u has a *consumption* $-\sigma(u) \geq 0$ of flow. That is, for each u of the sources and sinks, the sum of the flows of the outgoing arcs from u minus the sum of the flows of the incoming arcs to u must be equal to $\sigma(u)$.

The total amount of production of the sources is equal to the total amount of consumption of the sinks.

The *cost* $COST(\phi)$ of a flow ϕ in \mathcal{N} is the sum of $c(e)\phi(e)$ over all the arcs e of \mathcal{N} . The *minimum cost flow problem* is stated as follows. Given a network \mathcal{N} , find a flow ϕ in \mathcal{N} such that the cost of ϕ is minimum.

8.2.3 Finding Bend-Optimal Drawing

In this section, we present a flow network \mathcal{N} for an orthogonal drawing of a plane graph G [Tam87, GT97]. All angles in a drawing of G are viewed as commodities that are produced by the vertices, are transported between faces by the edges through their bends, and are eventually consumed by the faces in \mathcal{N} . The nodes of \mathcal{N} are vertices and faces of G . Since all angles we deal with have measure $k \cdot 90^\circ$ with $1 \leq k \leq 4$, we establish the convention that a unit of flow represents an angle of 90° . We shall see Facts 8.2.1 and 8.2.2 express the conservation of flow at vertices and faces, respectively. The formal description of \mathcal{N} is given below.

Let $G = (V, E)$ be a plane graph of maximum degree $\Delta \leq 4$ with face set \mathcal{F} . We construct a flow network \mathcal{N} from G as follows. The nodes of \mathcal{N} are the vertices and faces of G . That is, the node set U of \mathcal{N} is $U = U_{\mathcal{F}} \cup U_V$. Each node $u_F \in U_{\mathcal{F}}$ corresponds to a face F of G , while each node $u_v \in U_V$ corresponds to a vertex v of G . Each node $u_v \in U_V$ is a source and has a production

$$\sigma(u_v) = 4. \tag{8.4}$$

Each node $u_F \in U_{\mathcal{F}}$ is a sink and has a consumption

$$-\sigma(u_F) = \begin{cases} 2p(F) - 4 & \text{if } F \text{ is an inner face;} \\ 2p(F) + 4 & \text{if } F \text{ is the outer face } F_o \end{cases} \tag{8.5}$$

where $p(F)$ is the number of vertex-angles inside face F . (See Figs. 8.3(a)

and (b).) Thus every node in \mathcal{N} is either a source or a sink. Clearly the total production is $4n$, and the total consumption is

$$\sum_{F \neq F_o} (2p(F) - 4) + 2p(F_o) + 4 = 4m - 4f + 8$$

where n, m and f are the numbers of vertices, edges and faces of G , respectively. The total consumption is equal to the total production $4n$, according to Euler's formula in Theorem 2.2.3.

The arc set of network \mathcal{N} is $A = A_V \cup A_{\mathcal{F}}$.

- (i) A_V consists of all arcs of type (u_v, u_F) such that vertex v is on face F (see Fig. 8.3(c)); the flow $\phi(u_v, u_F)$ in arc (u_v, u_F) represents the sum of vertex-angles at vertex v inside face F , the lower bound $\lambda(u_v, u_F)$ is equal to the number of vertex-angles at v inside face F , the capacity is $\mu(u_v, u_F) = 4$, and the cost is $c(u_v, u_F) = 0$ (see Fig. 8.4); and
- (ii) $A_{\mathcal{F}}$ consists of all arcs of type $(u_F, u_{F'})$ such that face F shares an edge with face F' (see Fig. 8.3(d)); the flow $\phi(u_F, u_{F'})$ in arc $(u_F, u_{F'})$ represents the number of bends with an angle of 90° inside face F along the edges which are common to F and F' , and the lower bound is $\lambda(u_F, u_{F'}) = 0$, the capacity is $\mu(u_F, u_{F'}) = +\infty$, and the cost is $c(u_F, u_{F'}) = 1$ (see Fig. 8.5).

The conservation rule implies that for each source $u_v \in U_V$

$$\sum_{F \in \mathcal{F}} \phi(u_v, u_F) = 4 \tag{8.6}$$

and for each sink $u_F \in U_{\mathcal{F}}$

$$\sum_{F' \in \mathcal{F}} \phi(u_F, u_{F'}) - \left(\sum_{F' \in \mathcal{F}} \phi(u_{F'}, u_F) + \sum_{v \in V} \phi(u_v, u_F) \right) = \sigma(u_F). \tag{8.7}$$

A plane graph G together with a transformation into a network \mathcal{N} is illustrated in Fig. 8.3. The intuitive interpretation of the assignment above to the arcs is as follows.

- (1) Each unit of flow in network \mathcal{N} represents an angle of 90° ; for each arc $(u_v, u_F) \in A_V$, flow $\phi(u_v, u_F)$ represents the sum of the vertex-angles formed inside face F by the edges incident to v , which is given by $\phi(u_v, u_F) \cdot 90^\circ$ (see Fig. 8.4); for each arc $(u_F, u_{F'}) \in A_{\mathcal{F}}$, flow

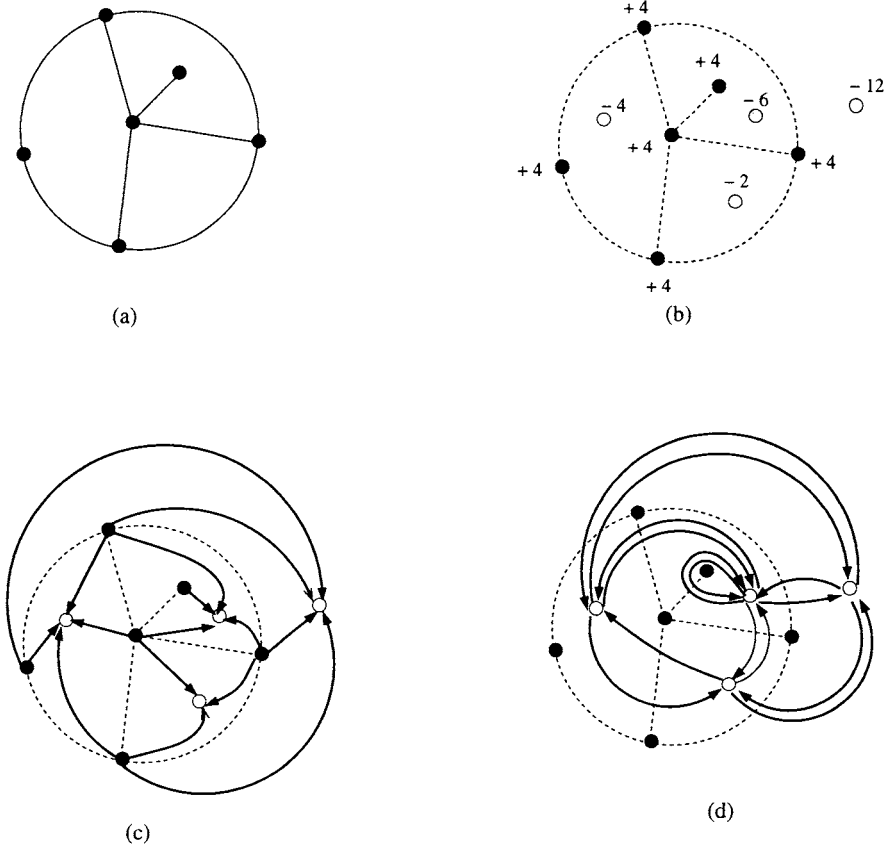


Fig. 8.3 (a) A plane graph G , (b) nodes of \mathcal{N} with their productions and consumptions, (c) arcs in A_V , and (d) arcs in A_F .

$\phi(u_F, u_{F'})$ represents the number of bends with an angle of 90° inside face F that appear along the edges separating face F from face F' (see Fig. 8.5).

- (2) The conservation rule at a vertex-node, Eq. (8.6), means that the sum of vertex-angles around each vertex must be equal to 360° . The conservation rule at a face-node, Eq. (8.7), means that each face must be a rectilinear polygon, as shown in the proof of Theorem 8.2.3 below.
- (3) The cost $COST(\phi)$ of the flow ϕ is equal to the number of bends of an orthogonal representation corresponding to ϕ .

TEAM LING - LIVE, INFORMATIVE, NON-COST AND GENUINE !

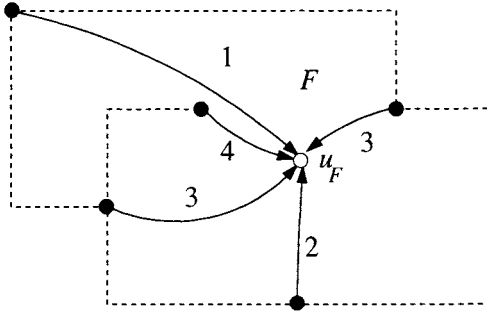


Fig. 8.4 Face F and flows in arcs in A_V .

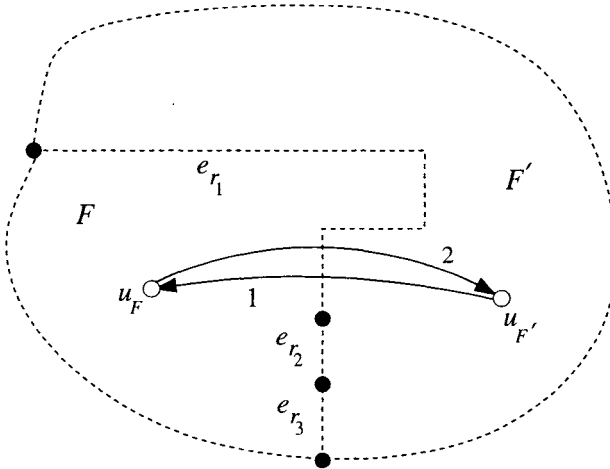


Fig. 8.5 Faces F and F' , and arcs $e = (u_F, u_{F'})$ and $e' = (u_{F'}, u_F)$ in A_F .

It is easy to see that every orthogonal representation R of G yields a feasible flow ϕ in network \mathcal{N} . Conversely, every feasible flow ϕ can be used to construct an orthogonal representation R of G as in the following theorem.

Theorem 8.2.3 *Let G be a plane graph, and let \mathcal{N} be the network constructed from G . For each integer flow ϕ in network \mathcal{N} , there is an orthogonal representation R that represents an orthogonal drawing D of G and whose number of bends is equal to the cost of the flow ϕ . In particular, the minimum cost flow can be used to construct a bend-optimal orthogonal drawing of G .*

Proof. We construct an orthogonal representation R of G by computing the a - and s -fields from the values taken by ϕ in arcs of A_V and $A_{\mathcal{F}}$, respectively, as follows.

We first set the values of a -fields using flows through arcs in A_V . A flow $\phi(u_v, u_F)$ in arc $(u_v, u_F) \in A_V$ corresponds to the vertex-angles formed at vertex v inside face F , and each unit of flow corresponds to an angle of 90° . Note that there can be more than one vertex-angles at v inside F . (See for example Fig. 8.6 where vertex v builds three vertex-angles inside face F .) For arc $(u_v, u_F) \in A_V$, let

$$R(v, F) = \{r \in R(F) : \text{both the edges } e_r \text{ and } e_{r'} \text{ are incident to vertex } v, \text{ and } r' \text{ follows } r \text{ in the list } R(F)\},$$

let $l = |R(v, F)|$, and let $R(v, F) = \{r_1, r_2, \dots, r_l\}$, where $1 \leq l \leq 4$. (For the example in Fig. 8.6, $l = 3$ and $R(v, F) = \{r_1, r_2, r_3\}$.) We then set

$$a_{r_1} = 90(\phi(u_v, u_F) - l + 1) \tag{8.8}$$

and

$$a_{r_i} = 90 \text{ for each } i, 2 \leq i \leq l. \tag{8.9}$$

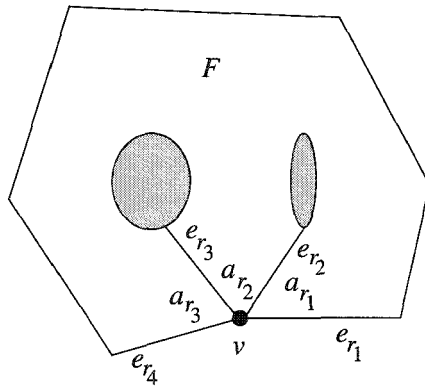


Fig. 8.6 Face F and angles a_{r_1}, a_{r_2} and a_{r_3} , where $l = 3$.

We next set the values of s -fields using flows through arcs in $A_{\mathcal{F}}$. Let F and F' be any pair of faces in G . A unit of flow in arc $(u_F, u_{F'}) \in A_{\mathcal{F}}$ represents a bend-angle of 90° inside face F appearing along edges separating F from F' . Let $E(F, F')$ be the set of edges which appear

in both planar representations $P(F)$ and $P(F')$ of faces F and F' , that is, $E(F, F')$ is the set of edges which are common to F and F' . Let $R(F, F') = \{r \in R(F) : e_r \in E(F, F')\}$. Let r_1, r_2, \dots, r_l be the elements of $R(F, F')$, and let r'_1, r'_2, \dots, r'_l be the corresponding elements in $R(F', F)$. Figure 8.5 illustrates an example for which $R(F, F') = \{e_{r_1}, e_{r_2}, e_{r_3}\}$ and $l = 3$. (If $F = F'$, then r_1 and r'_1 are the pair of elements in $R(F)$ corresponding to the two sides of the same bridge.) We denote by $\mathbf{0}^x$ a sequence of a number x of 0's, and by $\mathbf{1}^x$ a sequence of a number x of 1's:

$$\mathbf{0}^x = \overbrace{00 \cdots 0}^x \quad \text{and}$$

$$\mathbf{1}^x = \overbrace{11 \cdots 1}^x.$$

We then set

$$s_{r_1} = \mathbf{0}^{\phi(u_F, u_{F'})} \mathbf{1}^{\phi(u_{F'}, u_F)}, \tag{8.10}$$

$$s_{r'_i} = \mathbf{0}^{\phi(u_{F'}, u_F)} \mathbf{1}^{\phi(u_F, u_{F'})}, \tag{8.11}$$

and

$$s_{r_i} = s_{r'_i} = \epsilon \quad \text{for each } i, 2 \leq i \leq l. \tag{8.12}$$

For the example in Fig. 8.5 $\phi(u_F, u_{F'}) = 2$ and $\phi(u_{F'}, u_F) = 1$, and hence $s_{r_1} = 001$ and $s_{r_2} = s_{r_3} = \epsilon$.

We now need to show that $R(F)$ satisfies Properties (p1)–(p4) of an orthogonal representation. Property (p1) is automatically satisfied, since we have built the orthogonal representation R from a planar representation P . Property (p2) easily follows from Eqs. (8.10), (8.11) and (8.12). Property (p4) follows from Eq. (8.6). Property (p3) follows from Eqs. (8.1), (8.2),

(8.5) and (8.7)–(8.12), as follows. For each inner face F , we have

$$\begin{aligned}
 \sum_{r \in R(F)} \rho(r) &= \sum_{r \in R(F)} \left\{ 2 + |s_r|_0 - |s_r|_1 - \frac{a_r}{90} \right\} \\
 &= 2p(F) + \left\{ \sum_{F' \in \mathcal{F}} (\phi(u_F, u_{F'}) - \phi(u_{F'}, u_F)) - \sum_{v \in V} \phi(u_v, u_F) \right\} \\
 &= 2p(F) + \sigma(u_F) \\
 &= 2p(F) - \{2p(F) - 4\} \\
 &= 4,
 \end{aligned}$$

where $p(F)$ is the number of vertex-angles inside F . Similarly, for the outer face F_o , we have

$$\sum_{r \in R(F_o)} \rho(r) = -4.$$

By Eqs. (8.3) and (8.10)–(8.12) the number $b(R)$ of bends of an orthogonal representation R satisfies

$$\begin{aligned}
 b(R) &= \frac{1}{2} \sum_{F \in \mathcal{F}} \sum_{r \in R(F)} |s_r| \\
 &= \sum_{F, F' \in \mathcal{F}} (\phi(u_F, u_{F'}) + \phi(u_{F'}, u_F)) \\
 &= \sum_{e \in A} c(e)\phi(e) \\
 &= COST(\phi).
 \end{aligned}$$

Vice versa, it can be shown that the number of bends in each orthogonal representation of G is equal to the cost of some feasible flow in \mathcal{N} . \square

Garg and Tamassia [GT97] have shown that the minimum cost flow problem in this specific network can be solved in time $O(n^{7/4}\sqrt{\log n})$. Figure 8.7 depicts a minimum cost flow in the network constructed in Fig. 8.3 and a realizing grid embedding for the derived bend-optimal orthogonal representation.

8.3 Linear Algorithm for Bend-Optimal Drawing

One can find a bend-optimal orthogonal drawing of a plane graph with $\Delta \leq 4$ in time $O(n^{7/4}\sqrt{\log n})$ as explained in Section 8.2. In a VLSI floorplanning

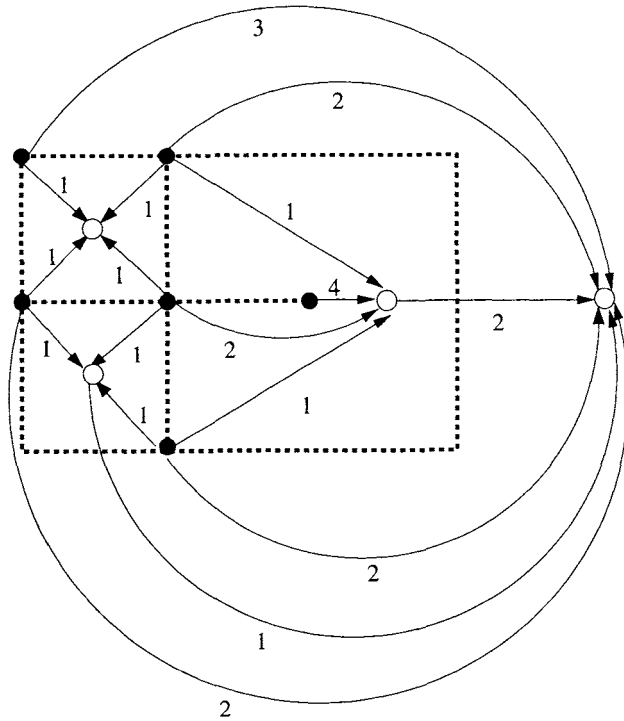


Fig. 8.7 Minimum cost flow in network \mathcal{N} associated with G in Fig. 8.7(a) and the corresponding orthogonal grid drawing of G . Arcs with zero flow are omitted.

problem, an input is often a plane graph with $\Delta \leq 3$ [Len90]. In this section we present a linear algorithm to find a bend-optimal orthogonal drawing of a 3-connected cubic plane graph G [RNN99], which depicts the key idea behind a linear algorithm for plane graphs with $\Delta \leq 3$ [RN02].

Let G be a plane 3-connected cubic graph. We assume for simplicity' sake that G has four or more outer edges. Since G is 3-connected, G has no 1- or 2-legged cycle. In any orthogonal drawing of G , every cycle C of G is drawn as a rectilinear polygon, and hence has at least four convex corners, i.e., polygonal vertices of inner angle 90° . Since G is cubic, such a corner must be a bend if it is not a leg-vertex of C . Thus we have the following facts for any orthogonal drawing of G .

Fact 8.3.1 *At least four bends must appear on the outer cycle $C_o(G)$ of G .*

Fact 8.3.2 *At least one bend must appear on each 3-legged cycle in G .*

The algorithm is outlined as follows.

Let G' be a graph obtained from G by adding four dummy vertices a, b, c and d of degree two, as corners, on any four distinct outer edges, one for each. Then the resulting graph G' has exactly four outer vertices of degree two designated as corners, and all other vertices of G' have degree three. Figure 8.8(b) illustrates G' for the graph G in Fig. 8.8(a). If G' has a rectangular drawing D' with the designated corners a, b, c and d as illustrated in Fig. 8.8(c), that is, G' satisfies the condition in Theorem 6.3.2, then from D' one can immediately obtain an orthogonal drawing D of G with exactly four bends by replacing the four dummy vertices with bends at the corners a, b, c and d as illustrated in Fig. 8.8(d). By Fact 8.3.1 D is a bend-optimal orthogonal drawing of G .

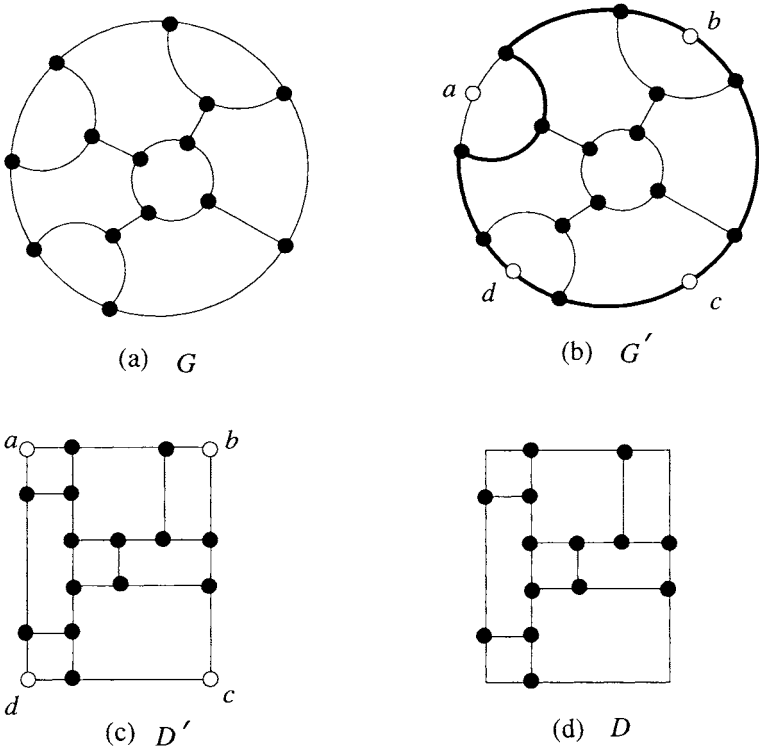


Fig. 8.8 G, G', D' and D .

One may thus assume that G' does not satisfy the condition in Theorem 6.3.2. **TEAM LING - LIVE, INFORMATIVE, NON-COST AND GENUINE !**

rem 6.3.2. Then G' has a bad cycle, that is, a 2-legged cycle containing at most one corner or a 3-legged cycle containing no corner. Since G is 3-connected, G has no 2-legged cycle. However, G' has four 2-legged cycles, each passing through all outer vertices except one of the four corners. (One of them is drawn by thick lines in Fig. 8.8(b).) Clearly all these four 2-legged cycles in G' are not bad, because each of them contains three corners. Thus every bad cycle in G' is a 3-legged cycle containing no corner. A bad cycle C in G' is defined to be *maximal* if C is not contained in the subgraph $G'(C')$ of G' inside C' for any other bad cycle C' in G' . In Fig. 8.9(a) C_1, C_2, \dots, C_6 are the bad cycles, C_1, C_2, \dots, C_4 are the maximal bad cycles in G' , and C_5 and C_6 are not maximal bad cycles since they are contained in $G'(C_4)$. The 3-legged cycle C_7 indicated by a dotted line in Fig. 8.9(a) is not a bad cycle in G' since it contains a corner a . As defined in Section 2.2.1, we say that cycles C and C' in G' are *independent* if $G'(C)$ and $G'(C')$ have no common vertex. Since G is a plane 3-connected cubic graph, all maximal bad cycles in G' are independent of each other. (Exercise 2.) Let C_1, C_2, \dots, C_l be the maximal bad cycles in G' . (In Fig. 8.9(a) $l = 4$.) Let G'' be the graph obtained from G' by contracting $G'(C_i)$ into a single vertex v_i for each maximal bad cycle C_i , $1 \leq i \leq l$, as illustrated in Fig. 8.9(b). Clearly G'' has no bad cycle, and hence by Theorem 6.3.2 G'' has a rectangular drawing. We first find a rectangular drawing of G'' , and then recursively find a suitable orthogonal drawing of $G'(C_i)$, $1 \leq i \leq l$, with the minimum number of bends, called a *feasible drawing*, and finally patch them to get an orthogonal drawing of G . (See Figs. 8.9 and 8.17.)

This completes an outline of the algorithm.

The remainder of Section 8.3 is organized as follows. In Section 8.3.1 we present a hierarchical structure called a genealogical tree of bad cycles in $G(C)$ and in Section 8.3.2 we give an assignment and labeling scheme for the edges on a bad cycle C based on the genealogical tree. The assignment and labeling will be useful in Section 8.3.3 for finding a feasible drawing of $G(C)$. Finally, in Section 8.3.4 we present a linear algorithm for finding a bend-optimal orthogonal drawing of G .

8.3.1 Genealogical Tree

Let C be a 3-legged cycle in a plane 3-connected cubic graph G . We denote by \mathcal{C}_C the set of all 3-legged cycles of G that are contained in $G(C)$. Clearly $C \in \mathcal{C}_C$. For the cycle C in Fig. 8.10(a) $\mathcal{C}_C = \{C, C_1, C_2, \dots, C_7\}$, where all cycles in \mathcal{C}_C are drawn by thick lines. For any two 3-legged cycles C' and

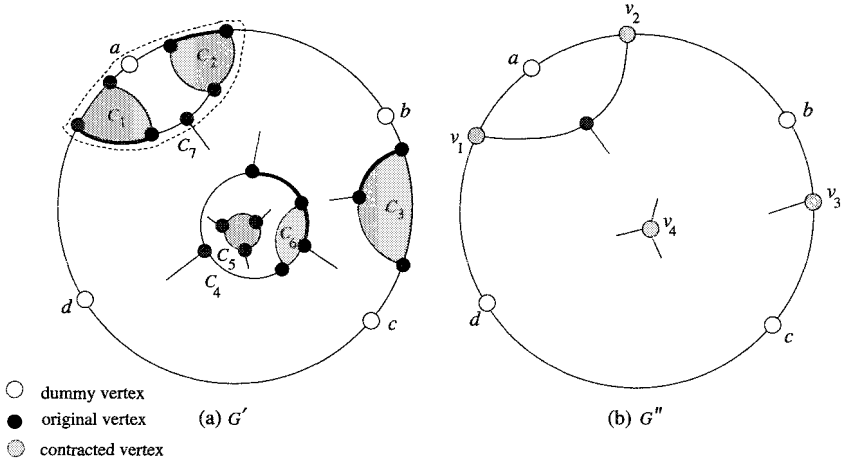


Fig. 8.9 G' and G'' .

C''' in \mathcal{C}_C , we say that C''' is a *descendant cycle* of C' and C' is an *ancestor cycle* of C''' if C''' is contained in $G(C')$. We also say that a descendant cycle C''' of C' is a *child-cycle* of C' if C''' is not a descendant cycle of any other descendant cycle of C' . In Fig. 8.10(a) cycles C_1, C_2, \dots, C_7 are the descendant cycles of C , cycles C_1, C_2, \dots, C_5 are the child-cycles of C , and cycles C_6 and C_7 are the child-cycles of C_4 . We now have the following lemma.

Lemma 8.3.3 *Let C be a 3-legged cycle in a 3-connected cubic plane graph G . Then the child-cycles of C are independent of each other.*

Proof. Suppose for a contradiction that a pair of distinct child-cycles C_1 and C_2 of C are not independent. Then C_1 and C_2 have a common vertex. However, either cannot be a descendant cycle of the other since both are child-cycles of C . Therefore C_2 has a vertex in $G(C_1)$ and a vertex not in $G(C_1)$. Thus C_2 must pass through two of the three legs of C_1 . Let v be the leg-vertex of the other leg of C_1 . Similarly, C_1 must pass through two of the three legs of C_2 . Let w be the leg-vertex of the other leg of C_2 . Then the removal of two vertices v and w disconnects G , contrary to the 3-connectivity of G . □

Lemma 8.3.3 implies that the containment relation among cycles in \mathcal{C}_C is represented by a tree as illustrated in Fig. 8.10(b); the tree is called the *genealogical tree* of C and denoted by T_C .

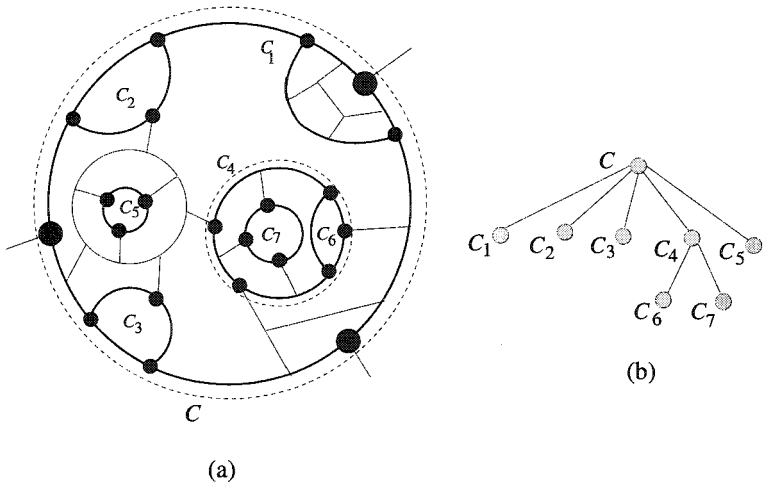


Fig. 8.10 (a) Cycles in C_C and (b) genealogical tree T_C .

Using the method described in the proof of Theorem 7.4.7, we have the following lemma.

Lemma 8.3.4 *Let C be a 3-legged cycle in a plane 3-connected cubic graph G . Then the genealogical tree T_C can be found in linear time.*

8.3.2 Assignment and Labeling

We define the following terms for each 3-legged cycle C in a plane 3-connected cubic graph G in a recursive manner based on a genealogical tree T_C . Each 3-legged cycle C in G is divided into three paths P_1, P_2 and P_3 by the three leg-vertices x, y and z of C as illustrated in Fig. 8.11. These three paths P_1, P_2 and P_3 are called the *contour paths* of C . Each contour path of C is classified as either a *green path* or a *red path*. In addition, we assign an integer $bc(C)$, called the *bend-count* of C , to each 3-legged cycle C in G . We will show later that $G(C)$ has an orthogonal drawing with $bc(C)$ bends and has no orthogonal drawing with fewer than $bc(C)$ bends, that is, the minimum number $b(G(C))$ of bends in an orthogonal drawing of $G(C)$ is equal to the bend-count $bc(C)$. Furthermore we will show that, for any green path of C , $G(C)$ has an orthogonal drawing with $bc(C)$ bends including a bend on the green path. On the other hand, for any red path of C , $G(C)$ does not have any orthogonal drawing with $bc(C)$ bends including

a bend on the red path. We define $bc(C)$, red paths and green paths in a recursive manner based on a genealogical tree T_C , as follows.

Let C be a 3-legged cycle in G , and let C_1, C_2, \dots, C_l in C_C be the child-cycles of C . Assume that we have already defined the classification and the assignment for all child-cycles of C and are going to define them for C . There are three cases.

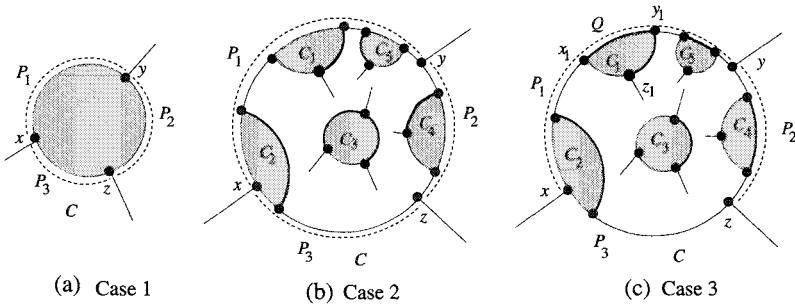


Fig. 8.11 $G(C)$ with three legs.

Case 1: C has no child-cycle, that is, $l = 0$.

In this case, T_C consists of a single vertex (see Fig. 8.11(a)). We classify all the three contour paths of C as green paths, and set

$$bc(C) = 1. \tag{8.13}$$

(By Fact 8.3.2 we need at least one bend on C . In Fig. 8.11(a) the three green paths of C are indicated by dotted lines.)

Case 2: C has child-cycles C_1, C_2, \dots, C_l , $l \geq 1$, but none of them has a green path on C .

In this case, we classify all the three contour paths of C as green paths, and set

$$bc(C) = 1 + \sum_{i=1}^l bc(C_i). \tag{8.14}$$

(In Fig. 8.11(b) the child-cycles of C are C_1, C_2, \dots, C_5 , and all green paths of them, drawn by thick lines, do not lie on C . Since none of C_1, C_2, \dots, C_l and their descendant 3-legged cycles has a green path on C as known later, the bend-optimal orthogonal drawings of $G(C_1), G(C_2), \dots, G(C_l)$ have no

bend on C and hence we need to introduce a new bend on C in an orthogonal drawing of $G(C)$. In Fig. 8.11(b) the three green paths of C are indicated by dotted lines.)

Case 3: Otherwise (see Fig. 8.11(c)).

In this case at least one of the child-cycles C_1, C_2, \dots, C_l , for example C_1, C_4 and C_5 in Fig. 8.11(c), has a green path on C . Classify a contour path $P_i, 1 \leq i \leq 3$, of C as a green path if a child-cycle of C has its green path on P_i . Otherwise, classify P_i as a red path. Thus at least one of P_1, P_2 and P_3 is a green path. We set

$$bc(C) = \sum_{i=1}^l bc(C_i). \tag{8.15}$$

(In Fig. 8.11(c) P_1 and P_2 are green paths, while P_3 is a red path. For a child-cycle C_j having a green path on C , $G(C_j)$ has an orthogonal drawing with $bc(C_j)$ bends including a bend on the green path, and hence we need not to introduce any new bend on C .)

We have the following lemmas.

Lemma 8.3.5 *Every 3-legged cycle C in G has at least one green path under the classification above.*

Proof. Immediate. □

Lemma 8.3.6 *Let C be a 3-legged cycle in G . Then the classification and assignment for all descendant cycles of C can be done in linear time, that is, in time $O(n(G(C)))$, where $n(G(C))$ is the number of vertices in graph $G(C)$.*

Proof. By Lemma 8.3.4 T_C can be found in linear time. Using T_C , the classification and assignment for all descendant cycles of C can be done in linear time. □

Lemma 8.3.7 *Let C be a 3-legged cycle in G , then $G(C)$ has at least $bc(C)$ vertex-disjoint 3-legged cycles of G which do not contain any edge on red paths of C .*

Proof. We will prove the claim by induction based on T_C .

We first assume that C has no child-cycle. According to the classification and assignment, all the three contour paths of C are green paths, and $bc(C) = 1$ by Eq. (8.13). Clearly $G(C)$ has a 3-legged cycle C of G , which does not contain any edge on red paths of C . Thus the claim holds for C .

We next assume that C has at least one child-cycle, and suppose inductively that the claim holds for all child cycles C_1, C_2, \dots, C_l of C . Then the hypothesis implies that, for each C_i , $1 \leq i \leq l$, $G(C_i)$ has at least $bc(C_i)$ vertex-disjoint 3-legged cycles of G which do not contain any edge on red paths of C_i . There are the following two cases to consider.

Case 1: None of the child-cycles of C has a green path on C (see Fig. 8.11(b)).

In this case, all the three contour paths of C are green, and $bc(C) = 1 + \sum_{i=1}^l bc(C_i)$ by Eq. (8.14). For each i , $1 \leq i \leq l$, a child-cycle C_i of C has no green path on C , and hence all C_i 's contour paths on C are red. By the induction hypothesis $G(C_i)$ has $bc(C_i)$ vertex-disjoint 3-legged cycles which do not contain any edge on red paths of C_i . Therefore, these $bc(C_i)$ cycles do not contain any edge on C . Furthermore by Lemma 8.3.3 the child-cycles C_1, C_2, \dots, C_l of C are independent of each other. Therefore $G(C)$ has $\sum_{i=1}^l bc(C_i)$ vertex-disjoint 3-legged cycles of G which do not contain any edge on C . Since G is cubic, C and these $\sum_{i=1}^l bc(C_i)$ 3-legged cycles are vertex-disjoint with each other. Trivially C does not contain any edge on red paths of C since all the contour paths of C are green. Thus $G(C)$ has at least $bc(C) = 1 + \sum_{i=1}^l bc(C_i)$ vertex-disjoint 3-legged cycles of G which do not contain any edge on red paths of C .

Case 2: At least one of the child-cycles of C has a green path on C (see Fig. 8.11(c)).

In this case, $bc(C) = \sum_{i=1}^l bc(C_i)$ by Eq. (8.15). By the induction hypothesis each cycle C_i , $1 \leq i \leq l$, has $bc(C_i)$ vertex-disjoint 3-legged cycles which do not contain any edge on red paths of C_i . Furthermore by Lemma 8.3.3 the child-cycles C_i , $1 \leq i \leq l$, are independent of each other. Therefore $G(C)$ has $\sum_{i=1}^l bc(C_i)$ vertex-disjoint 3-legged cycles which do not contain any edge on red paths of any child-cycle C_i . These $\sum_{i=1}^l bc(C_i)$ cycles may contain edges on green paths of C_i , but any green path of C_i is not contained in a red path of C by the classification of contour paths. Therefore, $G(C)$ has at least $bc(C) = \sum_{i=1}^l bc(C_i)$ vertex-disjoint 3-legged cycles which do not contain any edge on red paths of C . \square

Lemma 8.3.8 For every 3-legged cycle C of G , $b(G(C)) \geq bc(C)$.

Proof. By Lemma 8.3.7 $G(C)$ has at least $bc(C)$ vertex-disjoint 3-legged cycles. By Fact 8.3.2 at least one bend must appear on each of the 3-legged

cycles. Therefore any orthogonal drawing of $G(C)$ has at least $bc(C)$ bends, that is, $b(G(C)) \geq bc(C)$. \square

Conversely proving $b(G(C)) \leq bc(C)$ in Section 8.3.3, we have $b(G(C)) = bc(C)$ for any 3-legged cycle C in G . Indeed we will prove a stronger claim later in Lemmas 8.3.9 and 8.3.10 in Section 8.3.3 after introducing the definition of a feasible orthogonal drawing.

8.3.3 Feasible Orthogonal Drawing

In this section we define a feasible orthogonal drawing of $G(C)$ for a 3-legged cycle C in a plane 3-connected cubic graph G and give Algorithm **Feasible-Draw** for finding a feasible orthogonal drawing of $G(C)$ in linear time.

Let x, y and z be the three leg-vertices of C in G . One may assume that x, y and z appear on C in clockwise order, as illustrated in Fig. 8.12. For a green path P with ends x and y on C , an orthogonal drawing of $G(C)$ is defined to be *feasible for P* if the drawing has the following properties (f1)–(f3):

- (f1) The drawing of $G(C)$ has exactly $bc(C)$ bends.
- (f2) At least one bend appears on the green path P .
- (f3) The drawing of $G(C)$ intersects none of the the following six open halflines.

- the vertical open halfline with the upper end at x .
- the horizontal open halfline with the right end at x .
- the vertical open halfline with the lower end at y .
- the horizontal open halfline with the left end at y .
- the vertical open halfline with the upper end at z .
- the horizontal open halfline with the left end at z .

The property (f3) implies that, in the drawing of $G(C)$, any vertex of $G(C)$ except x, y and z is located in none of the following three areas (shaded in Fig. 8.12): the third quadrant with the origin x , the first quadrant with the origin y , and the fourth quadrant with the origin z . It should be noted that each leg of C must start with a line segment on one of the six open halflines above if an orthogonal drawing of G is extended from an orthogonal drawing of $G(C)$ feasible for P . Figure 8.12 illustrates an orthogonal drawing feasible for a green path P .

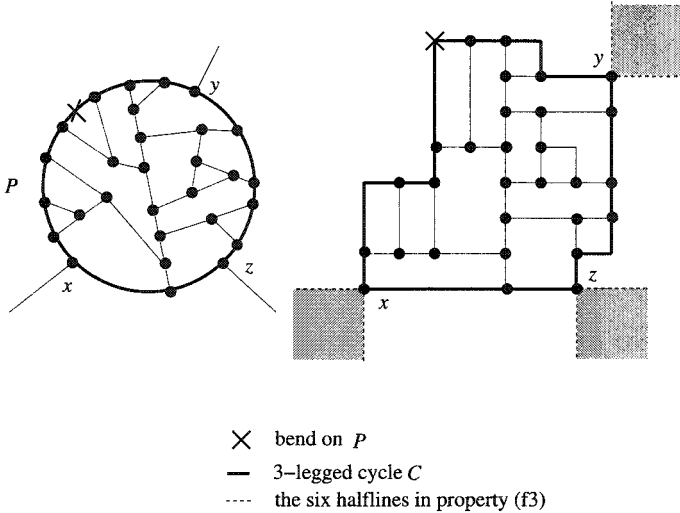


Fig. 8.12 An example of a feasible drawing.

We will often call an orthogonal drawing of $G(C)$ feasible for a green path of C simply a *feasible orthogonal drawing of $G(C)$* . We have the following lemma.

Lemma 8.3.9 *For any 3-legged cycle C of G and any green path P of C , $G(C)$ has an orthogonal drawing feasible for P .*

Proof. We give a recursive algorithm to find an orthogonal drawing of $G(C)$ feasible for P , as follows. There are three cases to consider.

Case 1: C has no child-cycle (see Fig. 8.11(a)).

In this case $bc(C) = 1$ by Eq. (8.13). We insert, as a bend, a dummy vertex t of degree two on an arbitrary edge on the green path P in graph $G(C)$, and let J be the resulting graph. (See Figs. 8.13(a) and (b).) Then every vertex of J has degree three except the four vertices of degree two: the three leg-vertices x , y and z , and the dummy vertex t . We regard x, t, y and z as the corners. Since G is a plane 3-connected cubic graph and C has no child-cycle, one can know that J has no bad cycle with respect to the four corners x, t, y and z , that is, J has neither a 2-legged cycle containing at most one corner nor a 3-legged cycle containing no corner. Therefore by Algorithm **Rectangular-Draw** in Section 6.3.3 one can find a rectangular drawing $D(J)$ of J with four corners on x, t, y and z , as illustrated in

Fig. 8.13(c). The drawing $D(J)$ immediately yields an orthogonal drawing $D(G(C))$ of $G(C)$ having exactly one bend at t , in which C is a rectangle, as illustrated in Fig. 8.13(d). Thus the drawing has the properties (f1)–(f3), and hence is feasible for P .

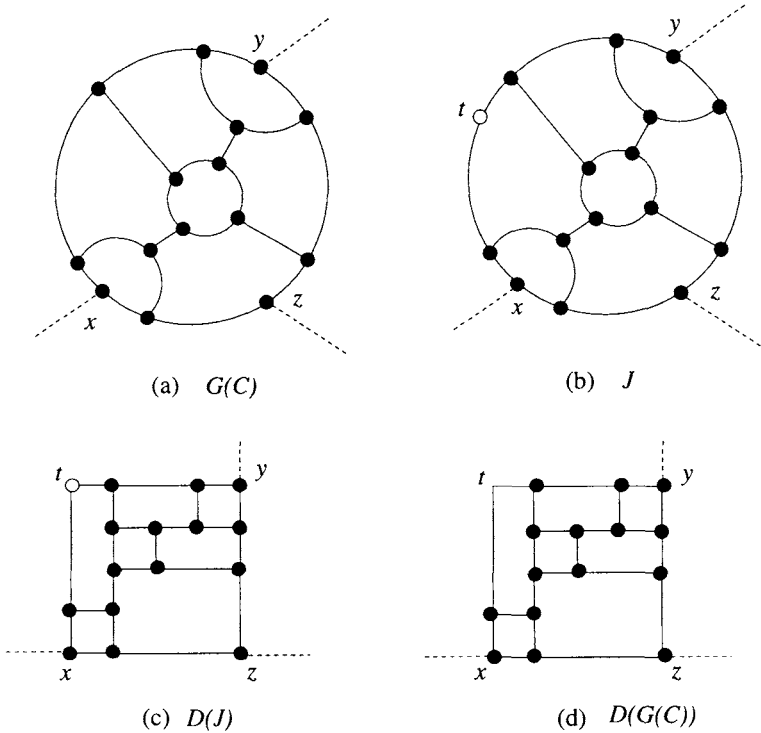


Fig. 8.13 Graphs $G(C)$ and J and their drawings $D(J)$ and $D(G(C))$.

Case 2: None of the child-cycles of C has a green path on C (see Fig. 8.11(b)).

Let C_1, C_2, \dots, C_l be the child-cycles of C , where $l \geq 1$. First, for each i , $1 \leq i \leq l$, we choose an arbitrary green path of C_i , and find an orthogonal drawing $D(G(C_i))$ of $G(C_i)$ feasible for the green path in a recursive manner.

Next, we construct a plane graph J from $G(C)$ by contracting each $G(C_i)$, $1 \leq i \leq l$, to a single vertex v_i . Figure 8.14(a) illustrates J for the graph $G(C)$ in Fig. 8.11(b) where the green path P is assumed to be

P_1 . One or more edges on P are contained in none of C_i , $1 \leq i \leq l$, and hence these edges remain in J . Add a dummy vertex t on one of these edges of P as illustrated in Fig. 8.14(b), and let K be the resulting plane graph. All vertices of K have degree three except the four vertices x, y, z

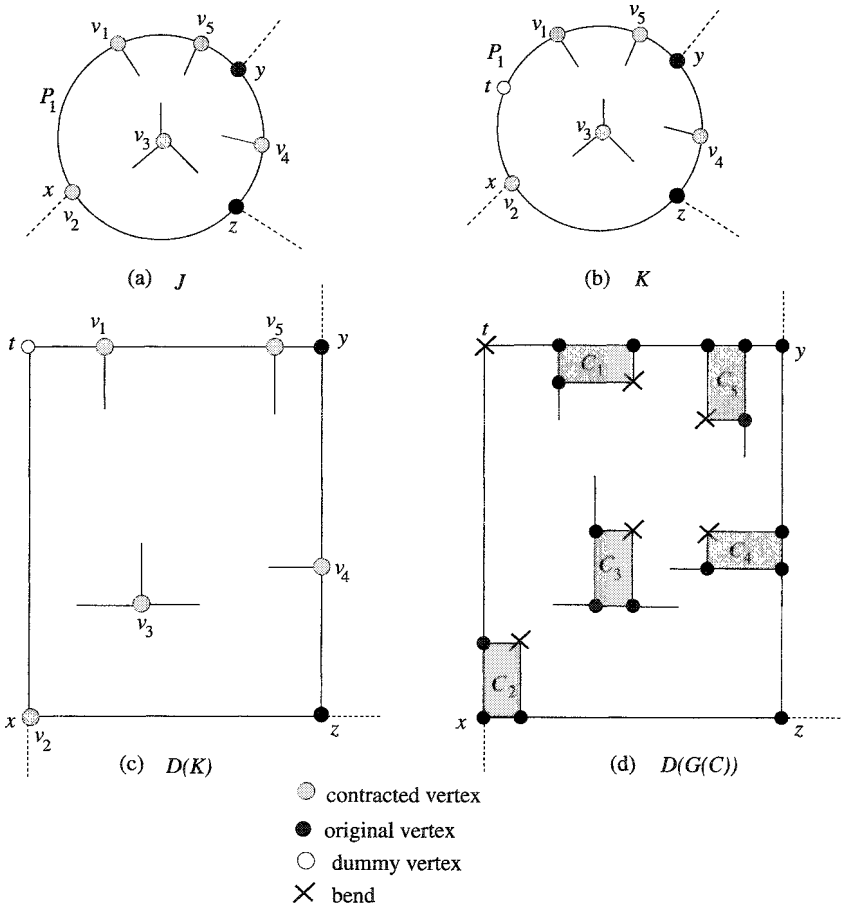


Fig. 8.14 Graphs J and K and drawings $D(K)$ and $D(G(C))$ for Case 2.

and t on $C_o(K)$ of degree two, and K has no bad cycle for the four corners x, t, y and z . Therefore, by **Rectangular-Draw** in Section 6.3.3, one can find a rectangular drawing $D(K)$ of K with four corners on x, t, y and z . $D(K)$ immediately yields an orthogonal drawing $D(J)$ of J with exactly one bend at t . Figure 8.14(c) illustrates a rectangular drawing of K for C

and $P = P_1$ in Fig. 8.11(b).

Finally, as explained below, patching the drawings $D(G(C_1))$, $D(G(C_2)), \dots, D(G(C_l))$ into $D(J)$, we can construct an orthogonal drawing of $G(C)$ with $bc(C) = 1 + \sum_{i=1}^l bc(C_i)$ bends (see Fig. 8.14). As illustrated in Fig. 8.15(b), there are twelve distinct embeddings of a contracted vertex v_i and the three legs incident to v_i , depending on both the directions of the three legs and the chosen green path of C_i , whose ends are denoted by x and y . For each of the twelve cases, we can replace a contracted vertex v_i with an orthogonal drawing of $G(C_i)$ feasible for the green path or a rotated one shown in Fig. 8.15(c), where the drawing of $G(C_i)$ is depicted as a rectangle for simplicity. For example, the embedding of the contracted vertex v_1 with three legs in Fig. 8.14(c) is the same as the middle one of the leftmost column in Fig. 8.15(b) (notice the green path of C_1 drawn by a thick line in Fig. 8.11(b)); and hence v_1 in $D(J)$ is replaced by $D(G(C_1))$, the middle one of the leftmost column in Fig. 8.15(c). Clearly t is a bend on P , and C is a rectangle in the drawing of $G(C)$. Thus the drawing is feasible for P . We call the replacement above a *patching operation*.

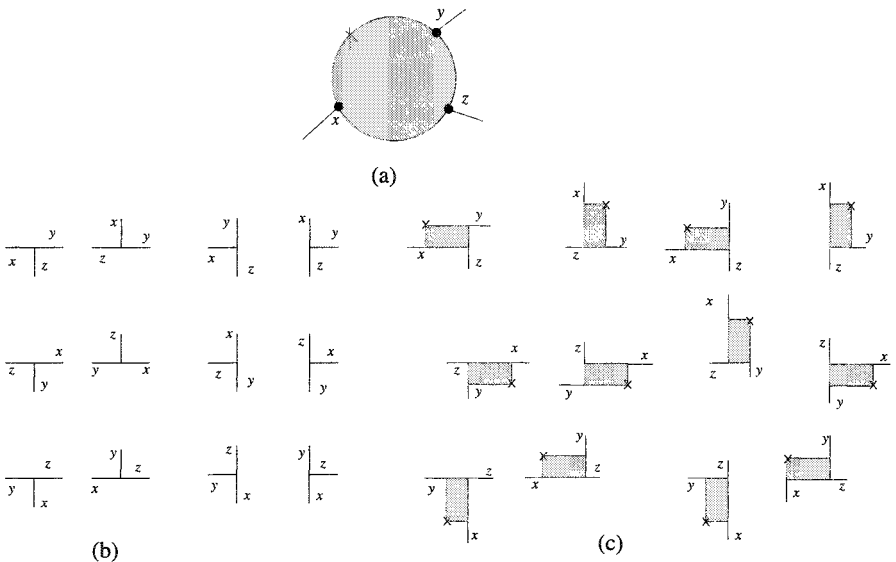


Fig. 8.15 (a) A 3-legged cycle, (b) twelve embeddings of a vertex v_i and three legs incident to v_i , and (c) twelve feasible orthogonal drawings of $G(C_i)$ and rotated ones.

Case 3: Otherwise (see Fig. 8.11(c)).

Let C_1, C_2, \dots, C_l be the child-cycles of C , where $l \geq 1$. In this case, for any green path P on C , at least one of C_1, C_2, \dots, C_l has a green path on P . One may assume without loss of generality that C_1 has a green path Q on the green path P of C , that the three leg-vertices x_1, y_1 and z_1 of C_1 appear on C_1 clockwise in this order, and that x_1 and y_1 are the ends of Q as illustrated in Fig. 8.11(c).

We first construct a plane graph J from $G(C)$ by contracting each $G(C_i), 1 \leq i \leq l$, to a single vertex v_i . Figure 8.16(a) illustrates J for $G(C)$ in Fig. 8.11(c). Replace v_1 in J with a quadrangle $x_1ty_1z_1$ as illustrated in Fig. 8.16(b) where t is a dummy vertex of degree two, and let K be the resulting plane graph. Thus all vertices of K have degree three except four vertices on $C_o(K)$ of degree two: the dummy vertex t and the three leg-vertices x, y and z of C . Furthermore K has no bad cycle for the four corners x, t, y and z . Therefore, by **Rectangular-Draw** in Section 6.3.3, one can find a rectangular drawing $D(K)$ of K with four corners on x, t, y and z , in which the contour $x_1ty_1z_1$ of a face is drawn as a rectangle. Figure 8.16(c) illustrates a rectangular drawing $D(K)$ of K for $G(C)$ in Fig. 8.11(c).

We next find feasible orthogonal drawings $D(G(C_1)), D(G(C_2)), \dots, D(G(C_l))$ in a recursive manner; $D(G(C_1))$ is feasible for the green path Q , and $D(G(C_i))$ is feasible for an arbitrary green path of C_i for each $i, 2 \leq i \leq l$.

Finally, patching the drawings $D(G(C_1)), D(G(C_2)), \dots, D(G(C_l))$ into $D(K)$, we can construct an orthogonal drawing $D(G(C))$ of $G(C)$ feasible for P ; we replace the rectangle $x_1ty_1z_1$ of $D(K)$ by $D(G(C_1))$, and replace each vertex $v_i, 2 \leq i \leq l$, by $D(G(C_i))$. In this case C is not always a rectangle in $D(G(C))$. One can observe with the help of Fig. 8.15 that each of the replacement above can be done without introducing any new bend or edge-crossing and without any conflict of coordinates of vertices as illustrated in Fig. 8.16(d). Note that the resulting drawing always expands outwards, and hence has the property (f3) of a feasible drawing. Since we replace the rectangle $x_1ty_1z_1$ in $D(K)$ by $D(G(C_1))$ and we have already counted the bend corresponding to t for C_1 , we need not count it for C . Thus one can observe that the drawing $D(G(C))$ has exactly $bc(C) = \sum_{i=1}^l bc(C_i)$ bends. Since a bend of $D(G(C_1))$ appears on Q , the bend appears on the green path P of C in $D(G(C))$. Hence $D(G(C))$ is an orthogonal drawing feasible for P . \square

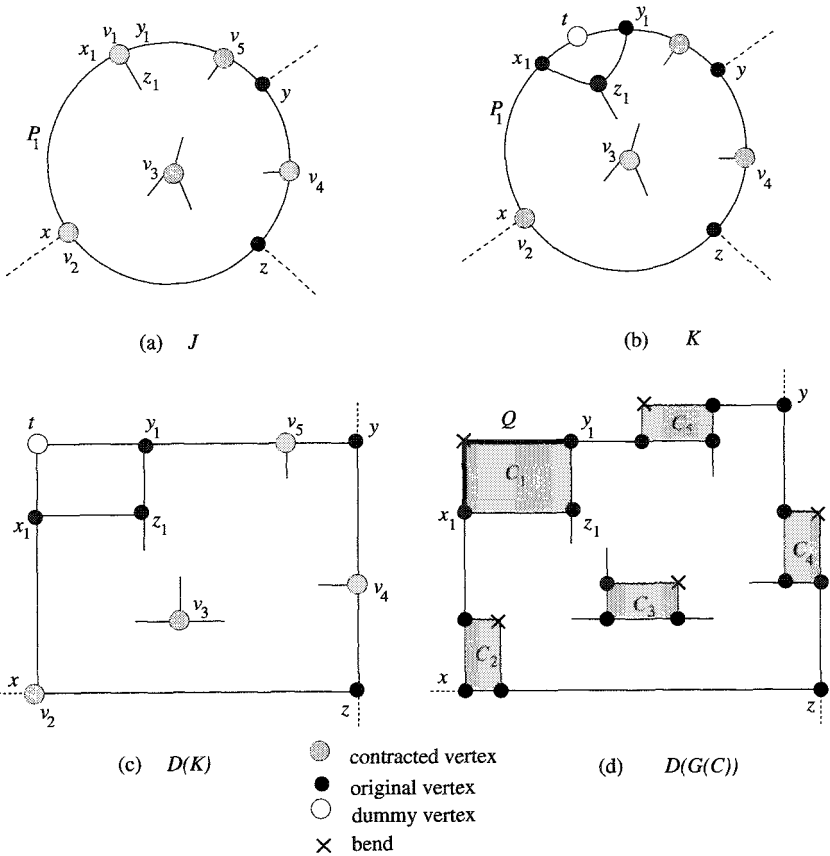


Fig. 8.16 Graphs J and K and drawings $D(K)$ and $D(G(C))$ for Case 3.

The definition of a feasible orthogonal drawing and Lemmas 8.3.8 and 8.3.9 immediately imply the following Lemma 8.3.10.

Lemma 8.3.10 *For any 3-legged cycle C in G , $b(G(C)) = bc(C)$, and a feasible orthogonal drawing of $G(C)$ has the minimum number $b(G(C))$ of bends.*

The algorithm for finding a feasible orthogonal drawing of $G(C)$ described in the proof of Lemma 8.3.9 above is hereafter called **Feasible-Draw**. It is not difficult to prove the following lemma on **Feasible-Draw** (Exercise 4).

Lemma 8.3.11 *Algorithm Feasible-Draw finds a feasible orthogonal*

drawing of $G(C)$ for a 3-legged cycle C in linear time, that is, in time $O(n(G(C)))$. \square

8.3.4 Algorithm

In this section we first present Algorithm **Orthogonal-Draw**(G) to find an orthogonal drawing of a plane 3-connected cubic graph G with at most $b(G) + 4$ bends, and then we present an idea behind Algorithm **Minimum-Bend**(G) to find an orthogonal drawing of G with $b(G)$ bends.

We now present Algorithm **Orthogonal-Draw**(G).

Algorithm **Orthogonal-Draw**(G)

begin

- 1 Add four dummy vertices of degree two on four distinct outer edges as the corners;
- 2 Let G' be the resulting graph; {See Fig. 8.9(a).}
- 3 Let C_1, C_2, \dots, C_l be the maximal bad cycles in G' ;
- 4 **for** each $i, 1 \leq i \leq l$, construct a genealogical tree T_{C_i} and determine green paths and red paths for every cycle in T_{C_i} ;
- 5 **for** each $i, 1 \leq i \leq l$, find an orthogonal drawing $D(G(C_i))$ of $G(C_i)$ feasible for an arbitrary green path of C_i by **Feasible-Draw**;
- 6 Let G'' be a plane graph derived from G' by contracting each $G(C_i), 1 \leq i \leq l$, to a single vertex v_i ; {See Fig. 8.9(b). G'' has no bad cycle.}
- 7 Find a rectangular drawing $D(G'')$ of G'' by **Rectangular-Draw**;
- 8 Patch the drawings $D(G(C_1)), D(G(C_2)), \dots, D(G(C_l))$ into $D(G'')$ to get an orthogonal drawing $D(G')$ of G' ;
- 9 Obtain an orthogonal drawing $D(G)$ of G by replacing the four dummy vertices a, b, c and d in $D(G')$ with bends

end.

Figure 8.17(a) illustrates a rectangular drawing of G'' in Fig. 8.9(b). The specified green path of each of the maximal bad cycles C_1, C_2, C_3 and C_4 of G' is drawn by a thick line in Fig. 8.9(a). Figure 8.17(b) illustrates an orthogonal drawing of G' in Fig. 8.9(a).

We now have the following theorem.

Theorem 8.3.12 *Let G be a plane 3-connected cubic graph, let G' be a graph obtained from G by adding four dummy vertices as in Algorithm*

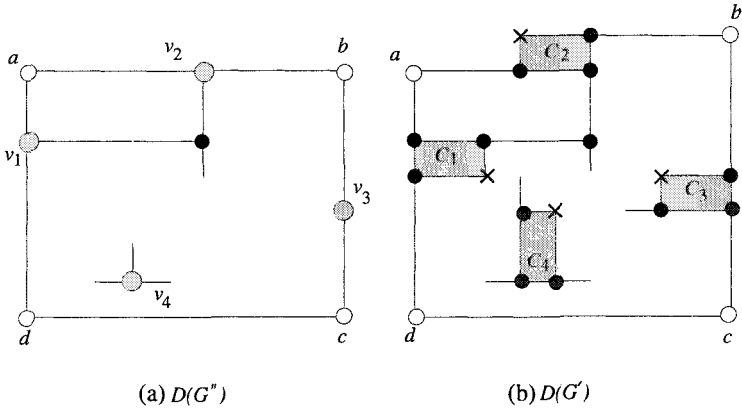


Fig. 8.17 (a) A rectangular drawing $D(G'')$ of G'' , and (b) an orthogonal drawing $D(G')$ of G' .

Orthogonal-Draw, and let C_1, C_2, \dots, C_l be the maximal bad cycles in G' . Then **Orthogonal-Draw** finds an orthogonal drawing of G with exactly $4 + \sum_{i=1}^l bc(C_i)$ bends in linear time. Furthermore $4 + \sum_{i=1}^l bc(C_i) \leq 4 + b(G)$.

Proof. (a) *Number of bends.*

There are two cases.

Case 1: G' has no bad cycle.

In this case we have a drawing with exactly four bends. (See Fig. 8.8.) By Fact 8.3.1 it is a bend-optimal orthogonal drawing.

Case 2: Otherwise.

Let C_1, C_2, \dots, C_l be the maximal bad cycles in G' . For each i , $1 \leq i \leq l$, an orthogonal drawing $D(G(C_i))$ feasible for an arbitrary green path of C_i has exactly $bc(C_i)$ bends. Furthermore the rectangular drawing $D(G'')$ has exactly four bends corresponding to the four dummy vertices. Algorithm **Orthogonal-Drawing** patches the drawings $D(G(C_1)), D(G(C_2)), \dots, D(G(C_l))$ into $D(G'')$ to get an orthogonal drawing $D(G')$ of G' , and replace the four dummy vertices with bends to get an orthogonal drawing $D(G)$ of G . Therefore $D(G)$ has exactly $4 + \sum_{i=1}^l bc(C_i)$ bends. Since the 3-legged cycles C_1, C_2, \dots, C_l are independent of each other, by Lemma 8.3.7 G has at least $\sum_{i=1}^l bc(C_i)$ vertex-disjoint 3-legged cycles. Therefore Fact 8.3.2 implies that $\sum_{i=1}^l bc(C_i) \leq b(G)$. Thus we have $4 + \sum_{i=1}^l bc(C_i) \leq 4 + b(G)$.

(b) *Time complexity.*

By a method similar to one in the proof of Theorem 7.4.7 we can find all maximal bad cycles in G' in linear time. **Orthogonal-Draw** calls **Rectangular-Draw** for G'' and **Feasible-Draw** for $G(C_i), 1 \leq i \leq l$. Both **Rectangular-Draw** and **Feasible-Draw** run in linear time. Since cycles $C_i, 1 \leq i \leq l$, are independent of each other, $\sum_{i=1}^l n(G(C_i)) \leq n$. Therefore the total time needed by **Feasible-Draw** is $O(n)$. Furthermore all contraction operations and all patching operations can be done in time $O(n)$ in total. Therefore **Orthogonal-Draw** runs in linear time. \square

Modifying **Orthogonal-Draw**, one can design a linear-time algorithm **Minimum-Bend** to find a bend-optimal orthogonal drawing of a plane 3-connected cubic graph G [RNN99]. The idea is as follows.

If a 3-legged cycle C in G has a green path on $C_o(G)$, then we can save one of the four bends mentioned in Fact 8.3.1, because a bend on the green path can be a bend on $C_o(G)$ and a bend on the 3-legged cycle C at the same time; hence one of the four bends mentioned in Fact 8.3.1 has been accounted for by the bends necessitated by Fact 8.3.2. We therefore want to find as many such 3-legged cycles as possible, up to a total number of four. We had better to find a 3-legged cycle which has a green path on $C_o(G)$ but none of whose child-cycles has a green path on $C_o(G)$, because a bend on such a cycle can play also a role of a bend on its ancestor cycle. We call such a cycle a “corner cycle”, that is, a *corner cycle* is a 3-legged cycle C in G such that C has a green path on $C_o(G)$ but no child-cycle of C has a green path on $C_o(G)$. (In Fig. 8.18 C'_1 and C'_2 drawn in thick lines are corner cycles. On the other hand, the two 3-legged cycles indicated by dotted lines are not corner cycles since C'_1 is their descendant cycle.) If G has $k (\leq 4)$ independent corner cycles C'_1, C'_2, \dots, C'_k , then we can save k bends. By a method similar to one given in the proof of Theorem 7.4.7 one can find independent corner cycles of G as many as possible in linear time. Let $C'_1, C'_2, \dots, C'_k, k \leq 4$, be the independent corner cycles of G , and let P'_i be the green path of C'_i on $C_o(G)$. We add a dummy vertex of degree two on any edge on each of the k paths P'_i , and add $4 - k$ dummy vertices of degree two on any $4 - k$ outer edges other than the k edges. Then there are exactly four vertices of degree two on $C_o(G)$. Executing Steps 2–8 of Algorithm **Orthogonal-Draw**, one can find a bend-optimal orthogonal drawing of G .

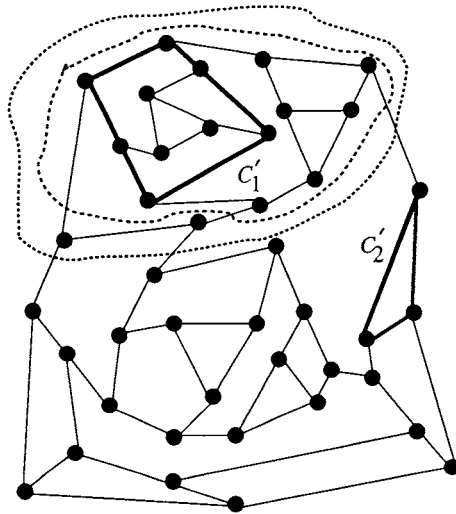


Fig. 8.18 Illustration for corner cycles.

8.4 Orthogonal Grid Drawing

An orthogonal drawing is called an *orthogonal grid drawing* if all vertices and bends are located on integer grid points. Given an orthogonal representation R of a plane graph G , one can obtain an orthogonal grid drawing of G in linear time, as follows. We first augment G to a new graph G' by adding dummy vertices and edges so that G' has a rectangular drawing D' , as illustrated in Fig. 8.19 where dummy edges are drawn by dotted lines and dummy vertices by white circles. From R we then construct an orthogonal representation R' of G' corresponding to the rectangular drawing D' of G' . R' is called a *rectangular refinement* of R . One can obtain R' in linear time by considering left turns and right turns in each facial polygon [Tam87, DETT99, KW01]. Using a method similar to that in Section 6.3.4, one can find a rectangular grid drawing D'_g from D' , where each of the vertices and bends has integer coordinates. D'_g immediately gives an orthogonal grid drawing D_g of G . D_g is “compact” in a sense that there is at least one vertical line segment of x -coordinate i for each integer i , $0 \leq i \leq W$, and there is at least one horizontal line segment of y -coordinate j for each integer j , $0 \leq j \leq H$, where W and H are the width and height of the grid, respectively. We have the following theorem on the size of a compact orthogonal grid drawing [Bie96b].

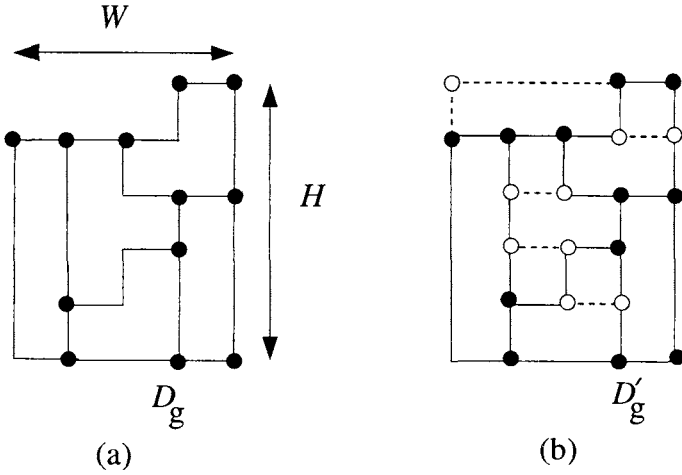


Fig. 8.19 (a) Orthogonal grid drawing of G , and (b) rectangular grid drawing of G' .

Theorem 8.4.1 *Let D_g be a compact orthogonal grid drawing of a plane graph G with b bends. Let W be the width of a grid, and let H be the height of a grid. Then $W + H \leq b + 2n - m - 2$.*

Proof. We call the four grid segments incident to a grid point the *ports* of the grid point and call them according to their direction as *top*, *bottom*, *right*, and *left*. Let $Top(D_g)$ be the number of vertices of G whose top ports are not used in D_g . Similarly we define $Bottom(D_g)$, $Right(D_g)$, and $Left(D_g)$.

We first claim that $W \leq \frac{1}{2}(b + Top(D_g) + Bottom(D_g)) - 1$. Let p be any of the $W + 1$ columns in the grid.

We first consider the case where p is used by a vertex of G . Let u be the top-most vertex, and let w be the bottom-most vertex in column p . If the top port of u is used, then it must contain a bend, since there is no vertex above u in the column p . So we either have a bend above u , or u contributes to $Top(D_g)$. Similarly, we either have a bend below w , or w contributes to $Bottom(D_g)$. Hence, p contributes at least two to the expression $b + Top(D_g) + Bottom(D_g)$.

We now consider the case where p is not used by any vertex. Then it contains a vertical line of an edge, and there are two bends at the end points of this line. Hence, p contributes at least two to the expression $b + Top(D_g) + Bottom(D_g)$.

Since each of the $W + 1$ columns contributes at least two to the expression $b + Top(D_g) + Bottom(D_g)$, we have $2(W + 1) \leq b + Top(D_g) + Bottom(D_g)$ and hence $W \leq \frac{1}{2}(b + Top(D_g) + Bottom(D_g)) - 1$.

Similarly we have $H \leq \frac{1}{2}(b + Right(D_g) + Left(D_g)) - 1$.

Each vertex v of G has $4 - d(v)$ unused ports, and hence $Top(D_g) + Bottom(D_g) + Right(D_g) + Left(D_g) = \sum_{v \in V} (4 - d(v)) = 4n - 2m$. Therefore

$$\begin{aligned} W + H &\leq \frac{1}{2}(b + Top(D_g) + Bottom(D_g)) - 1 \\ &\quad + \frac{1}{2}(b + Right(D_g) + Left(D_g)) - 1 \\ &= \frac{1}{2}(2b + 4n - 2m) - 2 \\ &= b + 2n - m - 2. \end{aligned}$$

□

Theorem 8.4.1 relates the number of bends with the grid size of a compact orthogonal grid drawing of a plane graph. Using Theorem 8.4.1 we can estimate the half perimeter $W + H$ of the grid required for a bend-optimal orthogonal grid drawing D_g of a plane 3-connected cubic graph G as follows: $W + H \leq b(G) + 2n - m - 2 = b(G) + \frac{1}{2}n - 2$, since $2m = 3n$.

Using Theorem 6.3.8 on a rectangular grid drawing, one can prove that any orthogonal drawing produced by the algorithm **Minimum-Bend** in Section 8.3 can be transformed to an orthogonal grid drawing on a grid such that $W \leq \frac{n}{2}$ and $H \leq \frac{n}{2}$ [RNN99]. The proof is left for an exercise.

8.5 Orthogonal Drawings without Bends

In a VLSI floorplanning problem, an input is often a plane graph with $\Delta \leq 3$ [Len90]. Such a plane graph G may have an orthogonal drawing without bends. The graph in Fig. 8.20(a) has an orthogonal drawing without bends as shown in Fig. 8.20(b). Not every plane graph with $\Delta \leq 3$ has an orthogonal drawing without bends. For example, the cubic plane graph in Fig. 8.1(a) has no orthogonal drawing without bends, since any orthogonal drawing of an outer cycle has at least four convex corners which must be bends in a cubic graph. One may thus assume that there are four or more outer vertices of degree two. It is interesting to know which classes of such plane graphs have orthogonal drawings without bends. The following theorem states a simple necessary and sufficient condition for a plane 2-

connected graph with $\Delta \leq 3$ to have an orthogonal drawing without bends [RNN03].

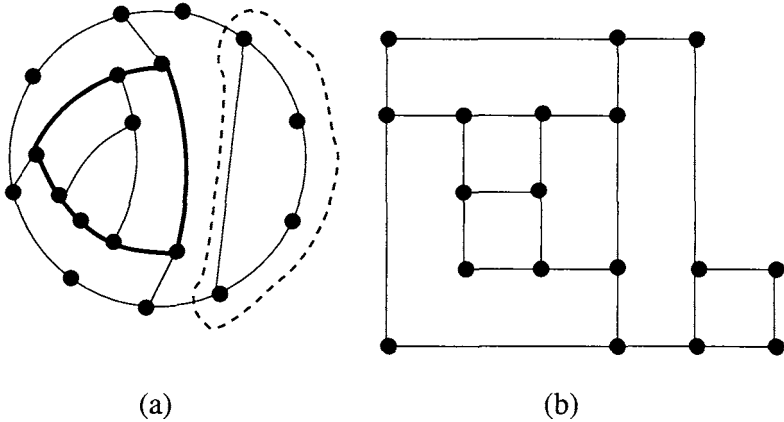


Fig. 8.20 (a) A plane graph G and (b) an orthogonal drawing of G without bends.

Theorem 8.5.1 *Assume that G is a plane 2-connected graph with $\Delta \leq 3$ and there are four or more outer vertices of degree two. Then G has an orthogonal drawing without bends if and only if every 2-legged cycle contains at least two vertices of degree two and every 3-legged cycle contains at least one vertex of degree two.*

The 2-legged cycle indicated by a dotted line in Fig. 8.20 contains two vertices of degree two, and the 3-legged cycle drawn by thick lines contains a vertex of degree two. Theorem 8.5.1 is a generalization of Thomassen's condition for rectangular drawings in Theorem 6.3.2; applying Theorem 8.5.1 to a plane 2-connected graph G in which all vertices have degree three except the four outer vertices of degree two, one can derive the condition.

It is easy to prove the necessity of Theorem 8.5.1, as follows.

[Necessity of Theorem 8.5.1]

Assume that a plane 2-connected graph G has an orthogonal drawing D without bends.

Let C be any 2-legged cycle. Then the rectilinear polygonal drawing of C in D has at least four convex corners. These convex corners must be vertices since D has no bends. The two leg-vertices of C may serve as two of the convex corners. However, each of the other convex corners must be

a vertex of degree two. Thus C must contain at least two vertices of degree two.

One can similarly show that any 3-legged cycle C in G contains at least one vertex of degree two. \square

Rahman *et al.* [RNN03] gave a constructive proof for the sufficiency of Theorem 8.5.1 and showed that the proof leads to a linear-time algorithm to find an orthogonal drawing without bends if it exists. They also extended their result on 2-connected graphs in Theorem 8.5.1 to arbitrary (not always 2-connected) graphs with $\Delta \leq 3$ as in the following theorem.

Theorem 8.5.2 *Let G be a plane graph with $\Delta \leq 3$. Then G has an orthogonal drawing without bends if and only if every k -legged cycle C in G contains at least $4 - k$ vertices of degree two for any integer k , $0 \leq k \leq 3$.*

8.6 Bibliographic Notes

Recently Rahman and Nishizeki [RN02] gave a linear algorithm for finding a bend-optimal orthogonal drawing of a plane graph G with $\Delta \leq 3$ by extending the ideas in Section 8.3. They classify the contour paths of each 2-legged or 3-legged cycle C into three types: blue, green and red paths. A blue path may have two or more bends, a green path may have one or more bends and a red paths does not have a bend in a bend-optimal orthogonal drawing of $G(C)$.

For a planar graph G with $\Delta \leq 3$, where the embedding of G is not fixed, Di Battista *et al.* [DLV98] gave an $O(n^5 \log n)$ time algorithm to find a bend-optimal orthogonal drawing of G . Rahman *et al.* presented a linear algorithm to examine whether a subdivision G of a planar 3-connected cubic graph has a no-bend drawing and to find one if G has [REN04].

Several linear-time algorithms are known for finding orthogonal drawings of plane graphs with a presumably small number of bends although they do not always find bend-optimal orthogonal drawings [Bie96b, Kan96, BK98]. Garg and Liotta [GL99] gave an $O(n^2)$ time algorithm for finding orthogonal drawings of planar 2-connected graphs with at most three bends more than the minimum number of bends.

Exercise

1. Show that an orthogonal representation R of a plane graph G does not represent a bend-optimal orthogonal drawing of G if $s_r = 10$ for an element r of a list $R(F)$.
2. Let G be a plane 3-connected cubic graph, and let G' be a graph obtained from G by adding four dummy vertices a, b, c and d of degree two on four distinct outer edges. Then show that all maximal bad cycles in G' are independent of each other.
3. Let C be a 3-legged cycle in a plane 3-connected cubic graph G . Then prove that

$$|C_C| \leq n(G(C))/2,$$

where $n(G(C))$ is the number of vertices in $G(C)$.

4. Prove Lemma 8.3.11.
5. Write a program to find an orthogonal grid drawing of a plane graph G from an orthogonal representation R of G .
6. Modifying **Orthogonal-Draw**, design a linear-time algorithm to find a bend-optimal orthogonal drawing of a plane 3-connected cubic graph G . Prove the correctness of your algorithm [RNN99].
7. Find an infinite number of plane connected graphs G with $\Delta \leq 3$ such that any orthogonal grid drawing of G needs a grid of size at least $(\frac{2}{3}n - 1) \times (\frac{2}{3}n - 1)$ and needs at least $\frac{5}{6}n - 1$ bends [Bie98].
8. Prove that a bend-optimal orthogonal drawing of a plane 3-connected cubic graph G has a corresponding orthogonal grid drawing on a grid such that $W \leq \frac{n}{2}$ and $H \leq \frac{n}{2}$ [RNN99].

Chapter 9

Octagonal Drawing

9.1 Introduction

In Chapters 6 and 8 we have studied rectangular drawings and orthogonal drawings of plane graphs, respectively. In this chapter we study an octagonal drawing which is an extension of a rectangular drawing and is a special type of an orthogonal drawing. An orthogonal drawing D of a plane graph G is called an *octagonal drawing* (with prescribed face areas) if (i) the outer cycle of D is a rectangle, (ii) each inner face has at most eight corners, and (iii) the area of each inner face is equal to a prescribed number. Figure 9.1(b) illustrates an octagonal drawing of the plane graph in Fig. 9.1(a), where a prescribed number is written inside each inner face.

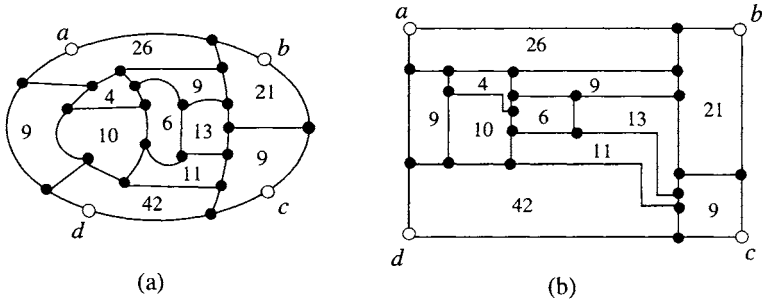


Fig. 9.1 (a) Plane graph and (b) its octagonal drawing.

An octagonal drawing of a plane graph G has practical applications in VLSI floorplanning. As we have seen in Section 1.5.1, a VLSI floorplan is often considered as a subdivision of a rectangle into a finite number of non-overlapping smaller rectangles, each of which corresponds to a functional

TEAM LING - LIVE, INFORMATIVE, NON-COST AND GENUINE !

entity called a *module* [Len90, SY99]. A “slicing floorplan” is often used by VLSI design [Shi96, YS93, YS95]. Divide a rectangle into two smaller rectangles by slicing it vertically or horizontally, divide any subrectangle into two smaller subrectangles by slicing it vertically or horizontally, and so on, as illustrated in Figs. 9.2(a)–(e). The resulting floorplan like one in Fig. 9.2(e) is called a *slicing floorplan*. An underlying plane graph of a slicing floorplan such as one illustrated in Fig. 9.2(f) is called a *slicing graph* G , where the four vertices a, b, c and d of degree two on the outer face of G represent the corners of the outer rectangle. (Note that the plane graph in Fig. 9.2(f) is isomorphic with that in Fig. 9.1(a).) Thus every vertex of a slicing graph G has degree two or three, and a slicing floorplan is a rectangular drawing of G . Since each module needs some physical area, each face of G in the drawing should satisfy some area requirements. However, when the area of each face is prescribed, there may not exist a rectangular drawing of G ; one example is illustrated in Fig. 9.3; the two faces of prescribed area 1 are adjacent in the plane graph in Fig. 9.3(a), but cannot be adjacent in any prescribed-area rectangular drawing as illustrated in Fig. 9.3(b). A floorplan using an octagonal drawing can overcome this problem. We say that a slicing graph is *good* if either the upper subrectangle or the lower one obtained by any horizontal slice will never be vertically sliced. All the graphs in Figs. 9.1(a), 9.2(f) and 9.3(a) are good slicing graphs.

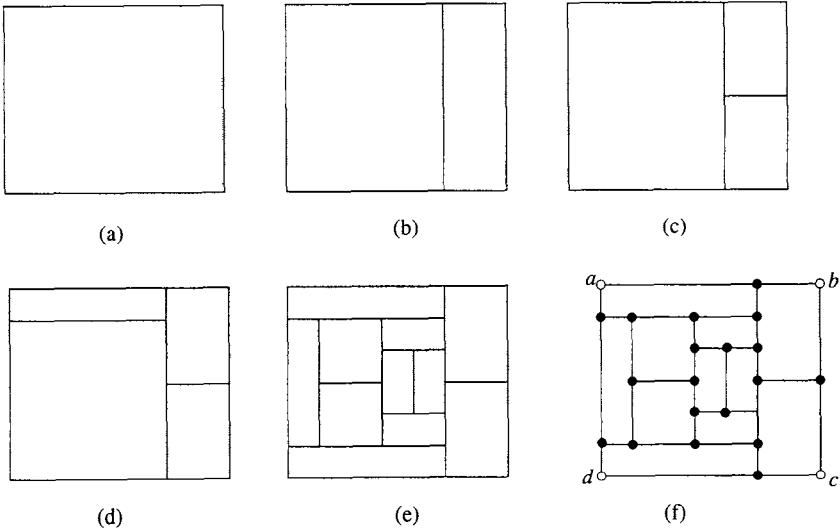


Fig. 9.2 Illustration of a slicing floorplan.

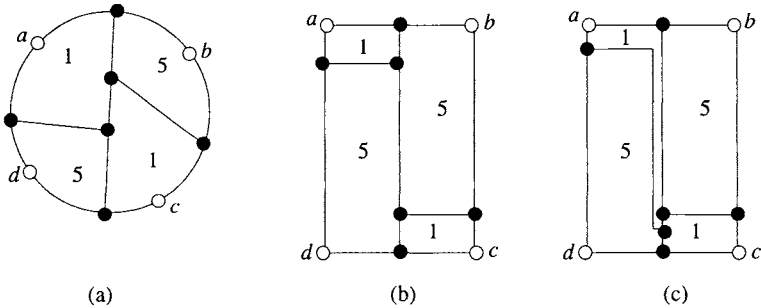


Fig. 9.3 (a) A slicing graph G , (b) a prescribed-area rectangular drawing of another graph, and (c) a prescribed-area octagonal drawing of G .

In this chapter we show that every good slicing graph has an octagonal drawing (with prescribed face areas) [RMN04]. We also present a linear algorithm for finding such a drawing.

In Section 9.2 we formally define good slicing graphs. In Section 9.3 we present a linear algorithm to obtain an octagonal drawing of a good slicing graph.

9.2 Good Slicing Graphs

In this section we define the class of good slicing graphs.

Assume that G is a *2-3 plane graph*. That is, G is 2-connected, every vertex of G has degree two or three, and there are four or more outer vertices of degree two, and exactly four of them, a, b, c and d , are designated as *corners*. The four corners a, b, c and d divide the outer cycle C_o into four paths, the north path P_N , the east path P_E , the south path P_S , and the west path P_W , as illustrated in Fig. 9.4. A path P in G is called an *NS-path* if P starts at a vertex on P_N , ends at a vertex on P_S , and does not pass through any other outer vertex and any outer edge. An NS-path P naturally divides G into two 2-3 plane graphs G_W^P and G_E^P ; G_W^P is the *west subgraph* of G including P , and G_E^P is the *east subgraph* of G including P . We call G_W^P and G_E^P the *two subgraphs corresponding to P* . Similarly, we define a *WE-path* P , the *north subgraph* G_N^P , and the *south subgraph* G_S^P .

We now present a formal recursive definition of a slicing graph. We call a 2-3 plane graph G a *slicing graph* if either it has exactly one inner face or it has an NS- or WE-path P such that each of the two subgraphs corresponding to P is a slicing graph. An NS- or WE-path P in a slicing graph

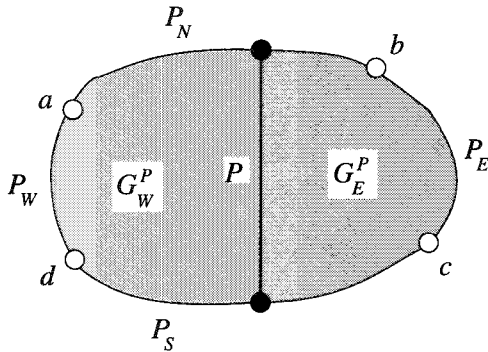


Fig. 9.4 Division of G into two subgraphs G_W^P and G_E^P by an NS-path P .

G is called a *slicing path* of G if each of the two subgraphs corresponding to P is a slicing graph. All the 2-3 plane graphs in Figs. 9.1(a), 9.2(f) and 9.3(a) are slicing graphs, while the 2-3 plane graph in Fig. 9.5 is not.

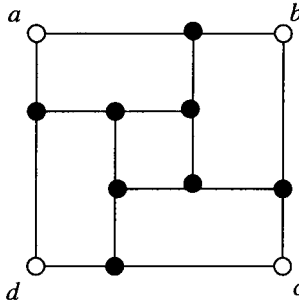


Fig. 9.5 A 2-3 plane graph which is not a slicing graph.

If G is a slicing graph, then all slicing paths appearing in the recursive definition can be represented by a binary tree T , called a *slicing tree*, as illustrated in Fig. 9.6 for the graph in Fig. 9.2(f). Each internal node u of T represents a slicing path, which is denoted by P_u . Each leaf u of T represents an inner face F_u of G . Each node u of T corresponds to a subgraph G_u of G induced by all inner faces that are leaves and are descendants of u in T . Thus $G = G_r$ for the root r of T . Figure 9.7 depicts a subgraph G_z of G in Fig. 9.6(a) for node z of T in Fig. 9.6(b). We classify the internal nodes of T into two types: (i) V-node and (ii) H-node. A V-node u represents an NS-slicing path P_u of G_u , while an H-node u represents a WE-slicing path

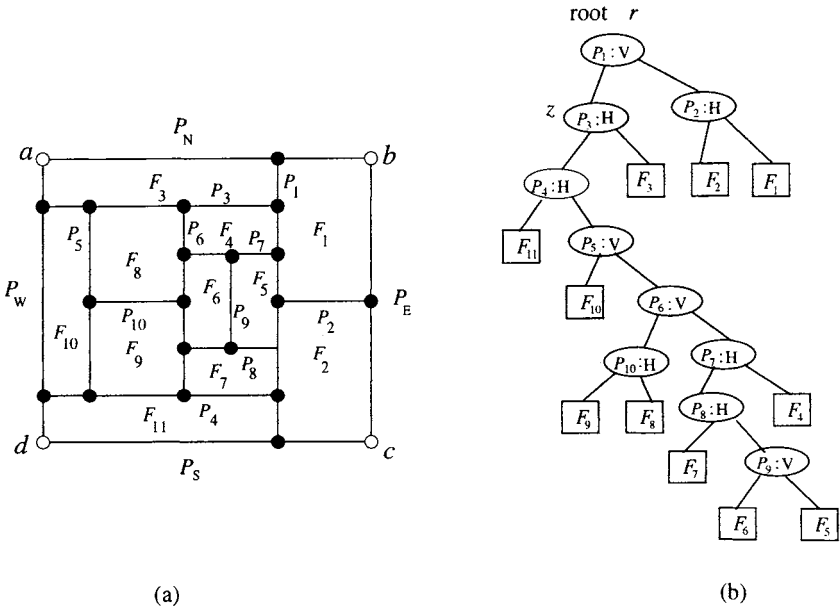


Fig. 9.6 (a)A slicing plane graph G , and (b) a good slicing tree T of G .

P_u of G_u .

We then give a formal definition of a good slicing graph. A *face path* of G is a WE-path on the contour of a single inner face of G . The graph G in Fig. 9.6(a) has no face path, while G_z in Fig. 9.7 has two face paths P_3 and P_4 , drawn by thick lines; P_3 is on face F_3 and P_4 is on face F_{11} . Any face path P of a slicing graph G is a slicing path, and either the north subgraph G_N^P or the south subgraph G_S^P corresponding to P will never be vertically sliced. We call a slicing tree T a *good slicing tree* if P_u is a face path of G_u for every H-node u in T . The tree in Figure 9.6(b) is a good slicing tree of the graph G in Fig. 9.6(a). We call a plane graph a *good slicing graph* if it has a good slicing tree for an appropriate labeling of designated corners as a, b, c and d .

The definitions above imply that every vertical slice of a good slicing graph is an arbitrary “guillotine cut” but every horizontal slice must be a “guillotine cut” along a face path. All the graphs in Figs. 9.1(a), 9.2(f), 9.3(a), 9.6(a) and 9.7 are good slicing graphs. A good slicing tree plays a crucial role in the algorithm for octagonal drawings. As we will show later in Section 9.3, the algorithm draws every vertical slice as a single vertical

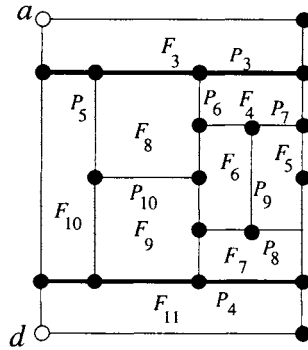


Fig. 9.7 Subgraph G_z of G for node z of T with two face paths P_3 and P_4 .

line segment, and draws every horizontal slice as either a single horizontal line segment or a sequence of three line segments, horizontal, vertical and horizontal ones, as illustrated in Figs. 9.1(b) and 9.3(c).

9.3 Octagonal Drawing

In this section we present a linear algorithm for finding an octagonal drawing of a good slicing graph G . We in fact give a proof of the following theorem [RMN04].

Theorem 9.3.1 *A good slicing graph G has an octagonal drawing D , and the drawing D can be found in linear time if a good slicing tree T is given.*

In the rest of this section we give a constructive proof of Theorem 9.3.1. Let G be a good slicing graph. One may assume without loss of generality that all vertices of G have degree three except for the four outer vertices a, b, c and d of degree two. We will show that every inner face of G is drawn as a rectilinear polygon of at most eight corners whose shape is one of the nine's in Fig. 9.8. We call a rectilinear polygon of shape like in Fig. 9.8 an *octagon* throughout the paper. Thus a rectangle is an octagon in our terminology, because the polygon in Fig. 9.8(i) is a rectangle. We denote by $A(R)$ the area of an octagon R , and by $A(G)$ the sum of the prescribed areas of all inner faces of a plane graph G .

The rest of this section is organized as follows. We give our Algorithm **Octagonal-Draw** in Section 9.3.1. In Section 9.3.2 we give the details for embedding a slicing path. In Section 9.3.3 we complete a proof of Theo-

rem 9.3.1 by verifying correctness and time complexity of the algorithm.

9.3.1 Algorithm Octagonal-Draw

In this section we give an algorithm for finding an octagonal drawing of a good slicing graph G .

An outline of the algorithm is as follows. Let T be a good slicing tree of G . Let u be an internal node of T , let v be the right child of u , and let w be the left child of u . One may assume that if u is a V-node then its right subtree rooted at v represents the east subgraph $G_E^{P_u}$ of G and its left subtree rooted at w represents the west subgraph $G_W^{P_u}$ and hence $G_v = G_E^{P_u}$ and $G_w = G_W^{P_u}$, and that if u is an H-node then $G_v = G_N^{P_u}$ and $G_w = G_S^{P_u}$, as illustrated in Fig. 9.6. We now traverse T by reverse preorder traversal, that is, we first traverse the root r of T , then traverse the right subtree and finally traverse the left subtree. We thus draw the inner faces F_1, F_2, \dots, F_{11} of G in Fig. 9.6(a) in this order, from east to west and north to south.

Before starting the traversal from root r , we choose an arbitrary rectangle R_r of area $A(G)$. Thus $A(G) = H \times W$ where H and W are the height and width of R_r . The outer cycle $C_o(G)$ is drawn as R_r . (See Fig. 9.9.) In general, when we traverse a node u of T , we have an octagon R_u of area $A(G_u)$; $C_o(G_u)$ is drawn as R_u . If u is an internal node, then we embed the slicing path P_u inside R_u so that P_u divides R_u into two octagons R_v and R_w so that $A(R_v) = A(G_v)$ and $A(R_w) = A(G_w)$, where v is the right child and w is the left child of u .

We start to traverse T from root r with the following initialization. We fix the positions of four designated vertices a, b, c and d of G as the corners of the initial rectangle R_r , as illustrated in Fig. 9.9. We then arbitrarily fix the positions of all vertices on the east path P_E^r of $G_r = G$ with preserving their relative positions. The positions of all other vertices on $C_o(G)$, drawn by black circles in Fig. 9.9, are not fixed.

When we traverse an internal node u of T , we have an octagon R_u such that $A(R_u) = A(G_u)$. Four vertices of degree two on $C_o(G_u)$ have been designated as the four corner vertices a, b, c and d of G_u as illustrated in Fig. 9.8. Let $a, x_{N1}, x_{N2}, b, c, x_{S1}, x_{S2}, d$ be the corners of octagon R_u , some of which may not exist. Note that a, b, c and d are vertices of G_u and $x_{N1}, x_{N2}, x_{S1}, x_{S2}$ are bends. We denote by P_N^u both the north side of R_u and the north path of $C_o(G_u)$ which connects a and b . Similarly we use the notation P_E^u, P_S^u and P_W^u . The positions of vertices a, b, c and d together with all the vertices on P_E^u have been fixed, but the positions of all vertices

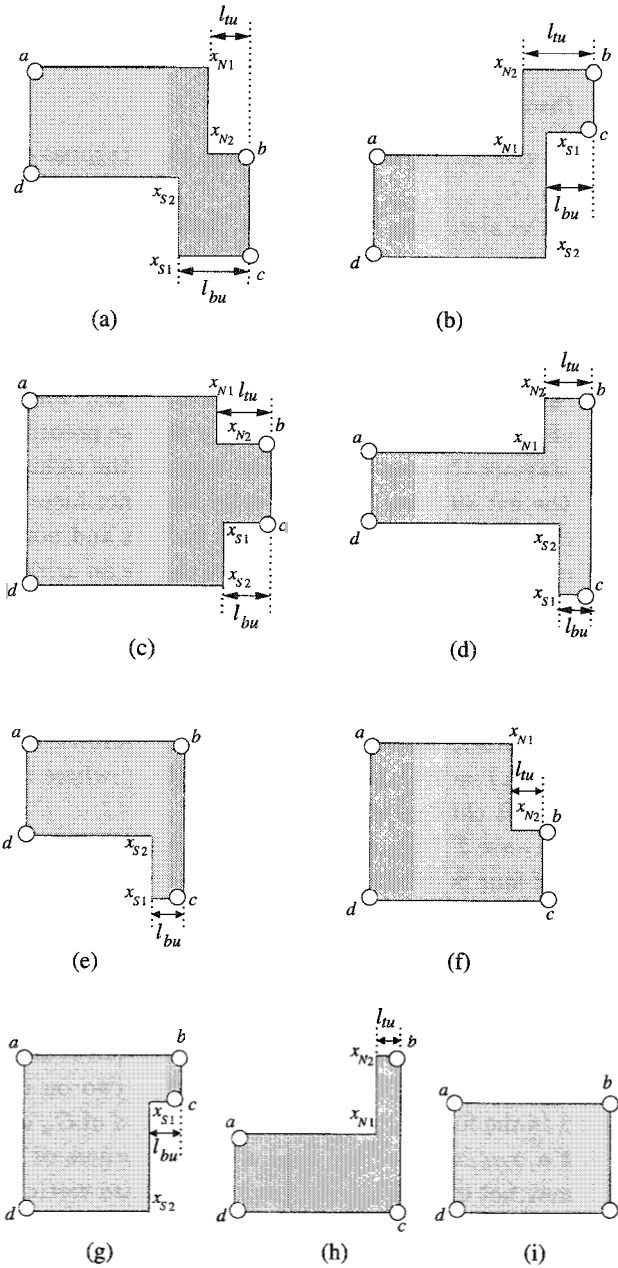


Fig. 9.8 Octagons.

on P_W^u and P_S^u except a , d and c have not been fixed.

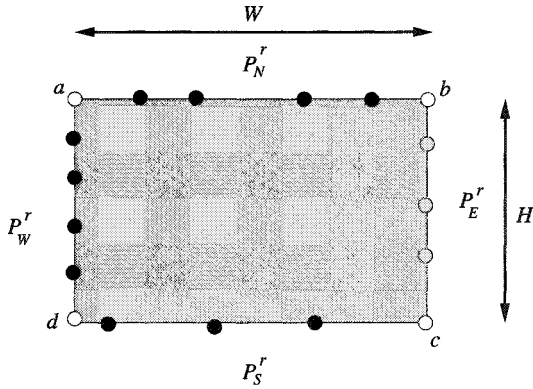


Fig. 9.9 Initial rectangle R_r .

We now describe the operations performed at each internal node u of T . Let v be the right child of u in T , and let w be the left child. We first consider the case where u is a V-node. One may assume that the NS-slicing path P_u connects a vertex y_N on P_N^u and a vertex y_S on P_S^u . (See Fig. 9.10.) As we will show later, the positions of corners a , b , c and d of R_u together with all vertices on P_E^u have been fixed, but the position of all other vertices of G_u have not been fixed. The goodness and the traversal order of T are crucial in the argument. We now fix the positions of y_N and y_S and divide R_u into two octagons R_v and R_w by embedding P_u as a vertical line segment so that $A(R_v) = A(G_v)$ and $A(R_w) = A(G_w)$. Indeed R_w is always a rectangle, as illustrated in Fig. 9.10. We will give the detail of this step later in Section 9.3.2. We now designate y_N, b, c and y_S as the four corner vertices of G_v , and designate a, y_N, y_S , and d as the four corner vertices of G_w . We then consider the case where u is an H-node. Assume that the face path P_u connects a vertex y_W on P_W^u and a vertex y_E on P_E^u , as illustrated in Fig. 9.11. The positions of all vertices on P_E^u including y_E have been fixed. We appropriately fix the position of y_W on P_W^u and divide R_u into two octagons R_v and R_w so that $A(R_v) = A(G_v)$ and $A(R_w) = A(G_w)$ by embedding P_u as either a single horizontal line segment or a sequence of three line segments, horizontal, vertical and horizontal ones, as illustrated in Fig. 9.11. We will give the detail of this step later in Section 9.3.2. We now designate a, b, y_E and y_W as the corner vertices of G_v , and designate

y_W, y_E, c , and d as the corner vertices of G_w .

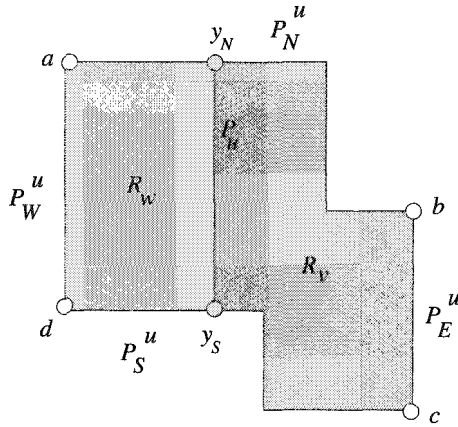


Fig. 9.10 Embedding of P_u in R_u for a V-node u .

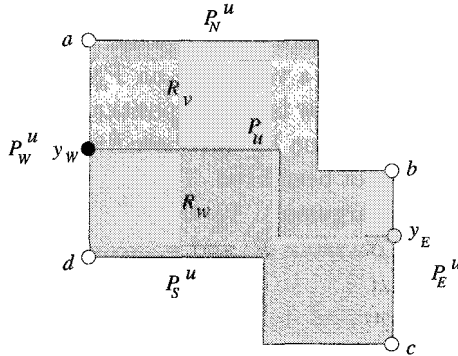


Fig. 9.11 Embedding of P_u in R_u for an H-node u .

We finally consider the case where we traverse a leaf u of T . In this case u corresponds to an inner face F_u , and the embedding of F_u has been already fixed as an octagon R_u . (See Fig. 9.12.) The positions of a, b, c and d and all vertices on P_E^u and P_N^u have been fixed. We arbitrarily fix the positions of all vertices on P_W^u other than a and d , preserving their relative positions on P_W^u . If there are vertices on P_S^u other than c and d , then their positions will be fixed in some later steps.

We call the algorithm described above Algorithm **Octagonal-Draw**.

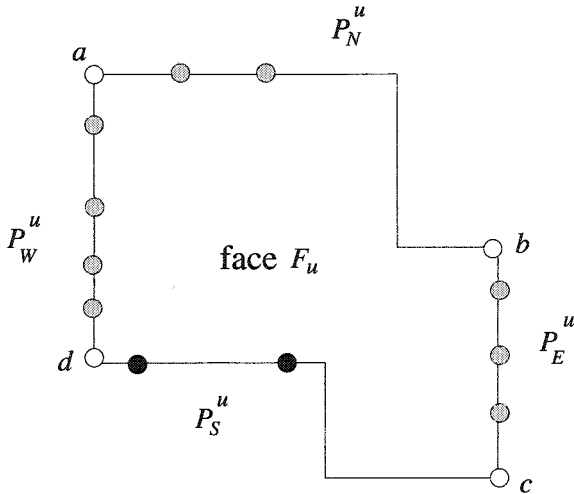


Fig. 9.12 R_u for a leaf u .

9.3.2 Embedding a Slicing Path

In this section we give the details of embedding a slicing path P_u inside an octagon R_u .

A polygonal vertex of R_u is called a *corner* of R_u . A corner of R_u has an interior angle 90° or 270° . A corner of an interior angle 90° is called a *convex corner* of R_u , while a corner of an interior angle 270° is called a *concave corner*. Let p and q be two consecutive polygonal vertices of R_u . We denote by pq the polygonal edge of R_u connecting p and q . We also denote by pq the straight line segment connecting two points p and q .

Let A_{\min} be the area of an inner face whose prescribed area is the smallest among all inner faces of G . Let H be the height of the whole drawing, that is, the height of the initial rectangle R_r . (See Fig 9.9.) Let f be the number of inner faces in G , and let

$$\lambda = \frac{A_{\min}}{fH}. \tag{9.1}$$

Since $A(G) = WH$, we have $\lambda = \frac{A_{\min}W}{fA(G)}$.

Let u be a node in T . Let l_{tu} be the length of line segment $x_{N2}b$ of an octagon R_u , and let l_{bu} be the length of line segment cx_{S1} , as illustrated in Fig. 9.8. If x_{N2} does not exist then let $l_{tu} = 0$, and if x_{S1} does not exist then let $l_{bu} = 0$. Let $l_u = \max\{l_{tu}, l_{bu}\}$. Thus $l_u = 0$ if and only if R_u is a rectangle. Let f_E^u be the number of inner faces in G_u each of which has an edge on the east path P_E^u of G_u . We call an octagon R_u a *feasible octagon* if the following eight conditions (i)–(viii) hold:

- (i) $A(R_u) = A(G_u)$;
- (ii) $l_u < f\lambda$;
- (iii) if x_{N2} is a concave corner as in Figs. 9.8(a), (c) and (f), then

$$l_{tu} < (f - f_E^u)\lambda;$$

- (iv) if x_{S1} is a concave corner as in Figs. 9.8(b), (c) and (g), then

$$l_{bu} < (f - f_E^u)\lambda;$$

- (v) if x_{N2} is a convex corner as in Figs. 9.8(b), (d) and (h), then $l_{tu} \geq f_E^u\lambda$;
- (vi) if x_{S1} is a convex corner as in Figs. 9.8(a), (d) and (e), then $l_{bu} \geq f_E^u\lambda$;
- (vii) if both x_{N2} and x_{S2} are concave corners as in Fig. 9.8(a), then

$$l_{bu} - l_{tu} \geq f_E^u\lambda;$$

and

- (viii) if both x_{N1} and x_{S1} are concave corners as in Fig. 9.8(b), then

$$l_{tu} - l_{bu} \geq f_E^u\lambda.$$

The initial octagon R_r for the root r of T is a rectangle of area $A(G_r)$, where $G_r = G$. Since R_r is a rectangle, $x_{N1}, x_{N2}, x_{S1}, x_{S2}$ do not exist and hence $l_u = l_{tu} = l_{bu} = 0$. Therefore the rectangle R_r is a feasible octagon.

We now have the following lemma on the embedding of P_u for a V-node u .

Lemma 9.3.2 *Let u be a V-node of T , let v be the right child of u , and let w be the left child. If R_u is a feasible octagon, then the NS-slicing path P_u can be embedded inside R_u as a single vertical line segment so that R_u is divided into two feasible octagons R_v and R_w .*

Proof. We first show that P_u can be embedded as a vertical line segment inside R_u so that $A(R_v) = A(G_v)$ and $A(R_w) = A(G_w)$, and hence R_v and R_w satisfies Condition (i). One may assume that P_u connects a vertex y_N on P_N^u and a vertex y_S on P_S^u . Since T is a good slicing tree, the north

path P_N^u of G_u is either on the north path P_N^r of $G_r = G$ or on a face path P_z of G_z for some H-node z which is an ancestor of u in T . Thus the part of G either above P_N^u or below P_N^u is a face of G . If the part of G below P_N^u were a face, then u could not be a vertical node. Hence the part of G above P_N^u is a face. Therefore the positions of all vertices on P_N^u other than a and b have not been fixed although the face above P_N^u has been drawn. Since the part of G below P_S^u has not been drawn, the position of all vertices on P_S^u other than c and d have not been fixed. We can therefore embed P_u as a vertical line by sliding y_N along P_N^u together with y_S along P_S^u until the equations $A(R_v) = A(G_v)$ and $A(R_w) = A(G_w)$ hold, as illustrated in Fig. 9.13.

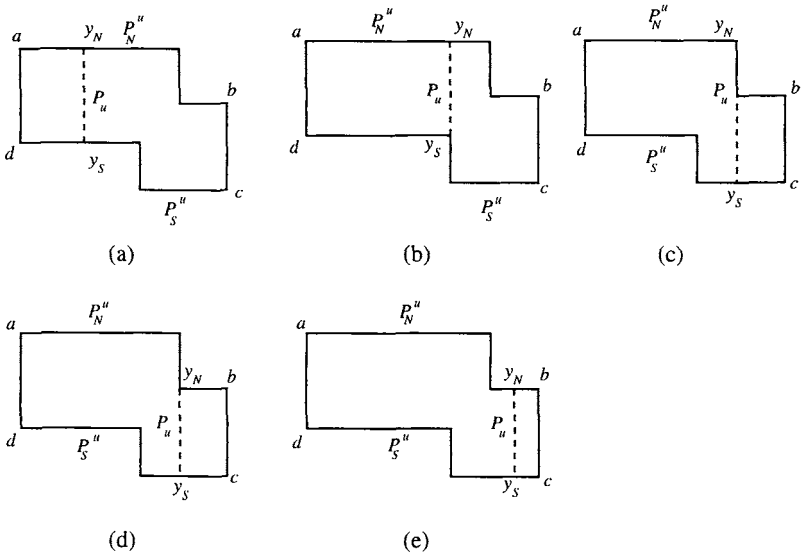


Fig. 9.13 Sliding P_u in R_u .

We then show that both R_v and R_w are octagons. Since R_u is a feasible octagon, by Condition (ii) $l_u < f\lambda$, and hence by Eq. (9.1) $l_u H < f\lambda H = A_{\min}$. This implies that each shaded rectangular area of width l_u and height $\leq H$ in Fig. 9.14 is smaller than the area A_{\min} of the smallest inner face in G regardless of the shape of octagon R_u . Since the east subgraph G_v of G_u contains at least one inner face of G , we have $A_{\min} \leq A(G_v)$. Thus sliding P_u above stops as in Fig. 9.13(a), and P_u is embedded so that R_w is drawn

as a rectangle as illustrated in Fig. 9.14 for all nine shapes. Thus R_v has a shape of the same type as R_u , and hence both R_v and R_w are octagons.

Since R_w is a rectangle, R_w satisfies condition (ii)–(viii).

We finally show that R_v satisfies Conditions (ii)–(viii). R_u satisfies Conditions (ii)–(viii). R_v has a shape of the same type as R_u . Furthermore, $l_{tv} = l_{tu}$, $l_{bv} = l_{bu}$, $l_v = l_u$ and $f_E^v = f_E^u$. Therefore R_v also satisfies Conditions (ii)–(viii). □

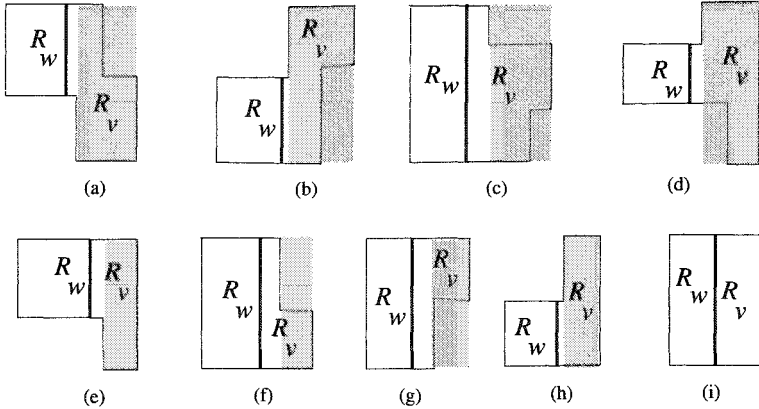


Fig. 9.14 Dividing R_u to R_v and R_w by P_u .

We now have the following lemma on an embedding of P_u for an H-node u .

Lemma 9.3.3 *Let u be an H-node of T , let v be the right child of u , and let w be the left child. If R_u is a feasible octagon, then the WE-slicing path P_u can be embedded inside R_u as either a single horizontal line segment or a sequence of three line segments, horizontal, vertical and horizontal ones, so that R_u is divided into two feasible octagons R_v and R_w .*

Proof. One may assume that the face path P_u connects a vertex y_W on P_W^u and a vertex y_E on P_E^u . We assume that the shape of R_u is as in Fig. 9.8(a). (The proof for the other shapes is similar.) In this case both x_{N_2} and x_{S_2} of R_u are concave corners, and hence by Condition (vii) $l_{bu} - l_{tu} \geq f_E^u \lambda > 0$. Since x_{N_2} is concave, by Condition (iii) $l_{tu} \leq (f - f_E^u) \lambda$. Also $l_u < f \lambda$ by Condition (ii). The position of vertex y_E has been fixed on P_E^u when the part of G to the right of bc was drawn. The horizontal line

L passing through y_E intersects either ad or $x_{S2}x_{S1}$, and hence there are the following two cases.

Case 1: L intersects ad .

Let L intersect ad at a point y' , as illustrated in Figs. 9.15(a), (b) and (c), and let Q be the polygon $a, x_{N1}, x_{N2}, b, y_E, y'$, then we have the following three subcases.

Subcase 1(a): $A(G_v) = A(Q)$.

In this case we fix the position of vertex y_W at point y' and embed the path P_u as a single horizontal line segment $y_W y_E$, as illustrated in Fig. 9.15(a). R_v is the octagon $a, x_{N1}, x_{N2}, b, y_E, y_W$, and R_w is the octagon $y_W, y_E, c, x_{S1}, x_{S2}, d$. R_v has the shape of a type as in Fig. 9.8(f), and R_w has the shape of type in Fig. 9.8(e).

We first show that R_v is feasible. Since $A(R_v) = A(G_v)$, Condition (i) holds for R_v . Furthermore, $l_{bv} = 0$, $l_{tv} = l_{tu}$, and hence $l_v = l_{tv} < l_u = l_{bu} < f\lambda$. Thus Condition (ii) also holds for R_v . Since $f_E^v < f_E^u$, we have $l_{tv} = l_{tu} \leq (f - f_E^u)\lambda < (f - f_E^v)\lambda$ and hence Condition (iii) also holds for R_v . Since x_{S1} and x_{S2} of R_v do not exist and x_{N2} of R_v is concave, Conditions (iv)-(viii) also hold. Thus R_v is feasible.

We then show that R_w is feasible. Since $A(R_w) = A(G_w)$, Condition (i) holds for R_w . Furthermore $l_{tw} = 0$, $l_{bw} = l_{bu}$, and hence $l_w = l_{bu} = l_u < f\lambda$. Thus Condition (ii) also holds for R_w . Since x_{S1} is a convex corner of R_w , $l_{bu} \geq f_E^u\lambda$ by Condition (vi). Clearly $f_E^w < f_E^u$. Therefore we have $l_{bw} = l_{bu} \geq f_E^u\lambda > f_E^w\lambda$, and hence Condition (vi) also holds for R_w . Since x_{N1} and x_{N2} of R_w do not exist and x_{S1} of R_w is convex, the other conditions also hold for R_w . Therefore R_w is feasible.

Subcase 1(b): $A(G_v) < A(Q)$.

Clearly $f_E^v < f_E^u$. We first fix a corner x'_{S1} of R_v on L so that the horizontal line segment $y_E x'_{S1}$ has length $l_{tu} + f_E^v\lambda$ and hence $l_{bv} = l_{tu} + f_E^v\lambda$, as illustrated in Fig. 9.15(b). We then fix the positions of x'_{S2} and y_W so that $A(R_v) = A(G_v)$. We now claim that $y_W x'_{S2}$ is below ax_{N1} . By Condition (vii) $l_{bu} \geq f_E^u\lambda + l_{tu}$, and hence $l_u = l_{bu} \geq f_E^u\lambda + l_{tu} > f_E^v\lambda + l_{tu} = l_{bv}$. Since $l_u < f\lambda$ by Condition (ii), we have $l_{bv}H < l_uH < f\lambda H = A_{\min}$ by Eq. (9.1) and hence the shaded rectangular area of width l_{bv} and height $< H$ in Fig. 9.15(b) is smaller than A_{\min} . Since G_v has at least one inner face, we have $A_{\min} \leq A(G_v)$. Therefore $y_W x'_{S2}$ is below ax_{N1} , and hence R_v is a (simple) octagon $a, x_{N1}, x_{N2}, b, y_E, x'_{S1}, x'_{S2}, y_W$, and R_w is an octagon $y_W, x'_{S2}, x'_{S1}, y_E, c, x_{S1}, x_{S2}, d$. Both R_v and R_w have a shape in Fig. 9.8(a).

We now show that R_v is feasible. Since $A(R_v) = A(G_v)$, Condition (i) holds for R_v . Clearly, $l_{tv} = l_{tu}$, $l_{bv} < l_{bu}$, and hence $l_v = l_{bv} < l_u < f\lambda$.

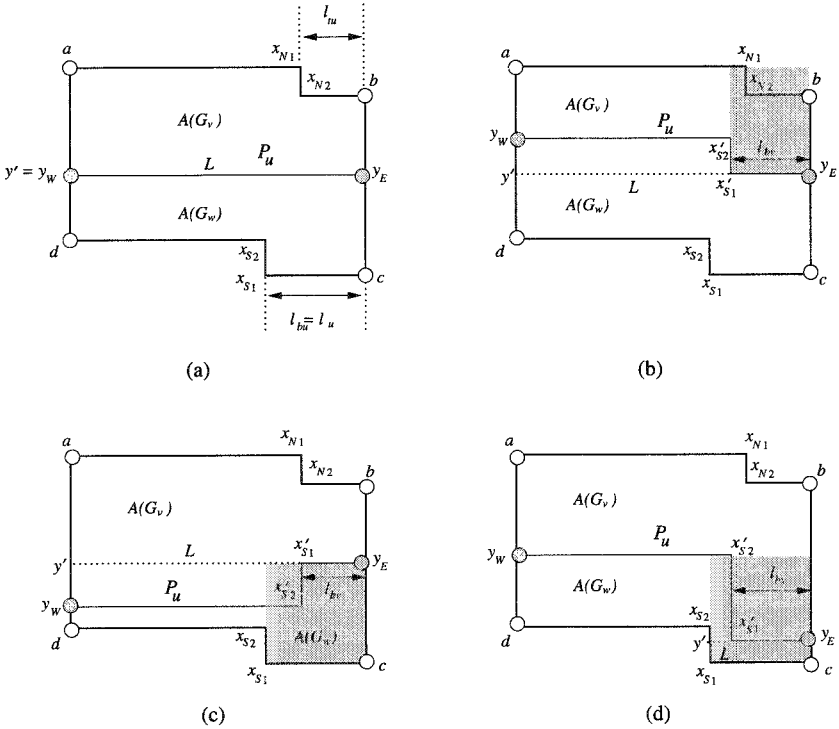


Fig. 9.15 Division of R_u to R_v and R_w by horizontal slice P_u

Thus Condition (ii) also holds for R_v . Since $l_{tv} = l_{tu} \leq (f - f_E^u)\lambda < (f - f_E^v)\lambda$, Condition (iii) also holds for R_v . Since $l_{bv} = l_{tu} + f_E^u\lambda \geq f_E^v\lambda$, Condition (vi) holds for R_v . Since $l_{bv} - l_{tv} = l_{bv} - l_{tu} = f_E^v\lambda$, Condition (vii) also holds. Since x_{N1} and x'_{S1} of R_v are convex corners, the other conditions also hold. Thus R_v is feasible.

Similarly one can show that R_w is a feasible octagon.

Subcase 1(c): $A(G_v) > A(Q)$.

Clearly $f_E^w < f_E^u$. We first fix x'_{S1} on L so that $l_{bv} = f_E^w\lambda$, and then fix the positions of x'_{S2} and y_w so that $A(R_v) = A(G_v)$, as illustrated in Fig. 9.15(c). Since $A(Q) < A(G_v)$, x'_{S2} is below x'_{S1} . We now claim that $y_w x'_{S2}$ is above dx_{S2} . By Condition (vi) $l_{bu} \geq f_E^u\lambda$ and hence $l_u = l_{bu} \geq f_E^u\lambda > f_E^w\lambda = l_{bv}$. Since $l_u H < f\lambda H = A_{\min}$, the shaded area in Fig. 9.15(c) is smaller than A_{\min} . Since $A_{\min} \leq A(G_w)$, $y_w x'_{S2}$ is above dx_{S2} . Thus R_v is a simple octagon $a, x_{N1}, x_{N2}, b, y_E, x'_{S1}, x'_{S2}, y_w$ and R_w is an octagon $y_w, x'_{S2}, x'_{S1}, y_E, c, x_{S1}, x_{S2}, d$. R_v has a shape in Fig. 9.8(c),

and R_w has a shape in Fig. 9.8(d).

We then show that R_v is feasible. Since $A(R_v) = A(G_v)$, Condition (i) holds for R_v . Since $l_{bv} < l_u$ and $l_{tv} = l_{tu} \leq l_u$, we have $l_v = \max\{l_{bv}, l_{tv}\} < l_u < f\lambda$. Hence Condition (ii) also holds for R_v . Since $l_{tu} < (f - f_E^u)\lambda$ by Condition (iii) for R_u , we have $l_{tv} = l_{tu} \leq (f - f_E^u)\lambda < (f - f_E^v)\lambda$ and hence Condition (iii) also holds for R_v . Since $f_E^u = f_E^v + f_E^w$, we have $l_{bv} = f_E^w\lambda = (f_E^u - f_E^v)\lambda \leq (f - f_E^v)\lambda$ and hence Condition (iv) also holds for R_v . Since x_{N1} and x_{S2} of R_v are convex corners and x_{N2} and x_{S1} are concave corners, the other conditions also hold for R_v . Thus R_v is feasible.

We finally show that R_w is feasible. Since $A(R_w) = A(G_w)$, Condition (i) holds for R_w . Since $l_{bw} = l_u > l_{bv} = l_{tw}$, we have $l_w = l_{bw} = l_u < f\lambda$ and hence Condition (ii) also holds for R_w . Since $l_{tw} = l_{bv} = f_E^w\lambda$, Condition (v) also holds for R_w . Since $l_{bu} \geq f_E^u\lambda$ by Condition (vi) for R_u , we have $l_{bw} = l_{bu} = l_u \geq f_E^u\lambda > f_E^w\lambda$ and hence Condition (vi) also holds for R_w . Since x_{N2} and x_{S1} of R_w are convex corners, the other conditions also hold for R_w . Thus R_w is feasible.

Case 2: L intersects $x_{S2}x_{S1}$ as illustrated in Fig. 9.15(d).

Let L intersect $x_{S2}x_{S1}$ at a point y' as illustrated in Fig. 9.15(d). Clearly $1 \leq f_E^v < f_E^u$. We first fix a corner x'_{S1} of R_v on L so that the horizontal line segment $y_E x'_{S1}$ has length $l_{tu} + f_E^u\lambda$ and hence $l_{bv} = l_{tu} + f_E^v\lambda$. By Condition (vii) $l_{bu} \geq l_{tu} + f_E^u\lambda$ for R_u . We thus have $l_{bv} = l_{tu} + f_E^v\lambda < l_{tu} + f_E^u\lambda \leq l_{bu}$, and hence $y_E x'_{S1}$ is shorter than $y_E y'$. We then fix the positions of x'_{S2} and y_W so that $A(R_w) = A(G_w)$. Since $l_u H < f\lambda H = A_{\min}$, the shaded area in Fig. 9.15(d) is smaller than A_{\min} . Since $A_{\min} \leq A(G_w)$, $y_W x'_{S2}$ is above dx_{S2} . Similarly one can show that $y_W x'_{S2}$ is below ax_{N1} . Thus R_v is an octagon $a, x_{N1}, x_{N2}, b, y_E, x'_{S1}, x'_{S2}, y_W$, and R_w is an octagon $y_W, x'_{S2}, x'_{S1}, y_E, c, x_{S1}, x_{S2}, d$. Both R_v and R_w have a shape in Fig. 9.8(a).

Similarly to the proof of Subcase 1(b), one can show that both R_v and R_w are feasible. □

9.3.3 Correctness and Time Complexity

In this section we verify the correctness and time complexity of Algorithm **Octagonal-Draw**, and mention some remarks on the algorithm.

We first prove the following lemma on the correctness of Algorithm **Octagonal-Draw**.

Lemma 9.3.4 *Algorithm Octagonal-Draw finds an octagonal drawing of a good slicing graph G .*

Proof. The initial rectangle R_r at the root r of T is a feasible octagon. Assume inductively that u is an internal node of T and R_u is a feasible octagon. Let v and w be the right child and left child of u , respectively. By Lemmas 9.3.2 and 9.3.3 one can embed P_u inside R_u so that R_v and R_w are feasible octagons. Thus, after the execution of the algorithm, each inner face of G corresponding to a leaf of T is a feasible octagon. Of course, the contour of the outer face of G is the rectangle R_r . Thus Algorithm **Octagonal-Draw** finds an octagonal drawing of G . \square

We now have the following lemma on the time complexity of Algorithm **Octagonal-Draw**.

Lemma 9.3.5 *Algorithm Octagonal-Draw runs in linear time.*

Proof. Since $A(G_u)$ for each leaf u in T is known, using a bottom-up computation one can compute $A(G_u)$ for each node u in T . This operation takes linear time in total. One can compute the number f_E^u for all nodes u of T in linear time. From R_u , $A(G_u)$, $A(G_v)$, $A(G_w)$, f_E^u , f_E^v and f_E^w , one can decide R_v and R_w in constant time. Thus the algorithm **Octagonal-Draw** runs in linear time. \square

Lemmas 9.3.4 and 9.3.5 complete the proof of Theorem 9.3.1.

9.4 Bibliographic Notes

A cycle C in a plane graph G is called a *proper inner cycle* if C does not contain any outer edge. Obviously every slicing graph is a 2-3 plane graph which has a rectangular drawing. Conversely, Rahman *et al.* showed that if a 2-3 plane graph G has a rectangular drawing and every proper inner cycle in G has at least five legs, then G is a good slicing graph and a good slicing tree T of G can be found in linear time [RMN04].

A connected graph is *cyclically k -edge connected* if the removal of any set of less than k edges leaves a graph such that exactly one of the connected components has a cycle. Let G be plane graph obtained from a cyclically 5-edge connected plane cubic graph by inserting four vertices a, b, c and d of degree two on the outer cycle. Thomassen [Tho92] showed that such a graph G has a drawing in which each edge is drawn as a single straight line segment which is not always horizontal or vertical, each inner face attains its prescribed area, and the outer cycle is a rectangle having the four vertices as corners. His proof is based on a proof for a result in [Tho84], and a

straightforward implementation of his method takes time $O(n^3)$, where n is the number of vertices in G .

Exercise

1. Find an example of a slicing graph which is not good.
2. Find an example of a good slicing graph in which a proper inner cycle has less than five legs.
3. Rewrite the proof of Lemma 9.3.3 for the shape of R_u as in Fig. 9.8(c).
4. Let A_{\max} be the area of an inner face whose prescribed area is the largest among all inner faces of a good slicing graph G , and let $k = A_{\max}/A_{\min}$. Then show that λ defined by Eq. (9.1) satisfies $\lambda \geq \frac{W}{k^2}$, where W is the width of the initial rectangle R_r . (See Fig. 9.9.)

This page intentionally left blank

Appendix A

Planar Embedding

A.1 Introduction

There are many practical situations in which one may wish to examine whether a given graph is planar, and if so, to find a planar embedding of the graph. For example, in the layout of printed or VLSI circuits, one is interested in knowing whether a graph G representing a circuit is planar and also in finding a planar embedding of G if it is planar. In this appendix we present a linear algorithm for the problem, due to [LEC67, BL76, CNAO85].

There are two typical algorithms which test planarity in linear time: one by Hopcroft and Tarjan [HT74], and the other by Booth and Lueker [BL76]. We present the latter called the “vertex addition algorithm” in Section A.2, because it is conceptually simpler than the former. The vertex addition algorithm was first presented by Lempel *et al.* [LEC67], and later improved to a linear algorithm by Booth and Lueker [BL76] employing an “ st -numbering” algorithm and a data structure called a “ PQ -tree.” The algorithm adds one vertex in each step. Previously embedded edges incident on this vertex are connected to it, and new edges incident on it are embedded and their ends are left unconnected. Sometimes whole pieces have to be reversed (flipped) around or permuted so that some ends occupy consecutive positions. If the representation of the embedded subgraph is updated with each alternation of the embedding, then the final representation will be an actual embedding of a given whole graph. Thus it is not difficult to derive an $O(n^2)$ time embedding algorithm from the testing algorithm.

In Section A.3 we present a linear algorithm for embedding planar graphs, which is based on the vertex addition algorithm. The embedding algorithm, due to Chiba *et al.* [CNAO85], runs in two phases. In the first phase the algorithm embeds a digraph obtained from a given planar (undi-

rected) graph by assigning a direction to every edge from the end having a greater *st*-number to the other end. In the second phase the algorithm extends the embedding of the digraph into an embedding of the given planar (undirected) graph.

A.2 Planarity Testing

We first define some terms and concepts and then present the vertex addition algorithm for testing planarity.

Let $G = (V, E)$ be a graph with vertex set V and edge set E . A graph is represented by a set of n lists, called “adjacency lists;” the list $\text{Adj}(v)$ for vertex $v \in V$ contains all the neighbors of v . For each $v \in V$, an actual drawing of a planar graph G determines, within a cyclic permutation, the order of v ’s neighbors embedded around v . *Embedding* a planar graph G means constructing adjacency lists of G such that, in each list $\text{Adj}(v)$, all the neighbors of v appear in clockwise order with respect to an actual drawing. Such a set Adj of adjacency lists is called an *embedding* of G . An example is illustrated in Fig. A.1(b) which is an embedding of a graph G in Fig. A.1(a).

Planarity testing examines whether a given graph is planar or not, that is, whether a given graph has an embedding on the plane without edge crossings. A graph G is planar if and only if all the 2-connected components of G are planar [Har72]. Moreover, one can easily obtain an embedding of an entire graph G from embeddings of all the 2-connected components of G . We hence assume that a given graph G is 2-connected. Due to Corollary 2.2.4 we may further assume that G satisfies $m < 3n$; otherwise G is nonplanar.

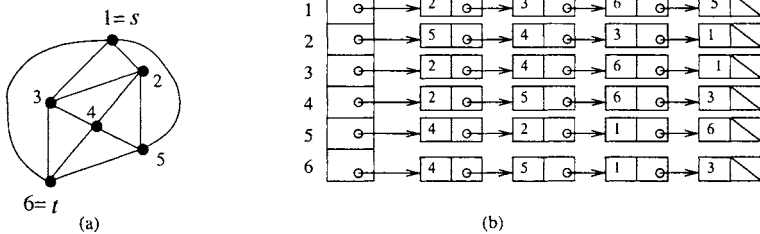


Fig. A.1 (a) An *st*-numbering, and (b) adjacency lists.

A numbering of vertices in a graph, called an “*st*-numbering,” plays a
TEAM LING - LIVE, INFORMATIVE, NON-COST AND GENUINE !

crucial role in the testing algorithm. In Section A.2.1 we study an “*st*-numbering” of a graph. We present the data structure called a “*PQ*-tree” in Section A.2.2 and a planarity testing algorithm in Section A.2.3.

A.2.1 *st*-Numbering

A numbering of vertices of a graph G by $1, 2, \dots, n$ is called an *st*-numbering if the two vertices “1” and “ n ” are necessarily adjacent and each j of the other vertices is adjacent to two vertices i and k such that $i < j < k$. The vertex “1” is called a *source* and is denoted by s , while the vertex “ n ” is called a *sink* and is denoted by t . Figure A.1(a) illustrates an *st*-numbering of a graph. Every 2-connected graph G has an *st*-numbering and an algorithm given by Even and Tarjan [ET76] finds an *st*-numbering in linear time as described below.

We first present a procedure DFS which executes the depth-first search with choosing an arbitrary edge (t, s) as the first edge, and computes, for each vertex v , its depth-first number $\text{DFN}(v)$, its father $\text{FATH}(v)$, and its lowpoint $\text{LOW}(v)$. These values are used in the *st*-numbering algorithm. The lowpoint is defined as follows:

$$\text{LOW}(v) = \min(\{v\} \cup \{w \mid \text{there exists a back edge } (u, w) \text{ such that} \\ u \text{ is a descendant of } v \text{ and } w \text{ is an ancestor} \\ \text{of } v \text{ in a DFS tree } T\}),$$

where we assume that the vertices v are named by their depth-first numbers $\text{DFN}(v)$. This definition is equivalent to

$$\text{LOW}(v) = \min(\{v\} \cup \{\text{LOW}(x) \mid x \text{ is a son of } v\} \\ \cup \{w \mid (v, w) \text{ is a back edge}\}).$$

Figure A.2 illustrates these values. The following depth-first search algorithm DFS, a variant of one in Chapter 3, computes the required values.

```

procedure DFS( $G$ )
  begin
    Set  $T = \emptyset$ ;  $\{T$  is a DFS tree $\}$ 
    Set COUNT = 1;
    Mark each vertex of  $G$  “new”;
    Let  $v$  be any vertex of  $G$ ;
  
```

```

Search(v)
end.

procedure Search(v);
  begin
    Mark v "old";
    Set  $DFN(v) = COUNT$ ;
    Set  $COUNT = COUNT + 1$ ;
    Set  $LOW(v) = DFN(v)$ ;
    for each vertex  $w$  in  $Adj(v)$ 
      do if  $w$  is marked "new"
        then begin
          Add  $(v, w)$  to  $T$ ;
          Set  $FATH(w) = v$ ;
          Search(w);
          Set  $LOW(v) = \min\{LOW(v), LOW(w)\}$ 
        end
      else if  $w$  is not  $FATH(v)$ 
        then set  $LOW(v) = \min\{LOW(v), DFN(w)\}$ 
    end;
end;

```

Figure A.2(b) illustrates a DFS Tree, $DFN(v)$ and $LOW(v)$.

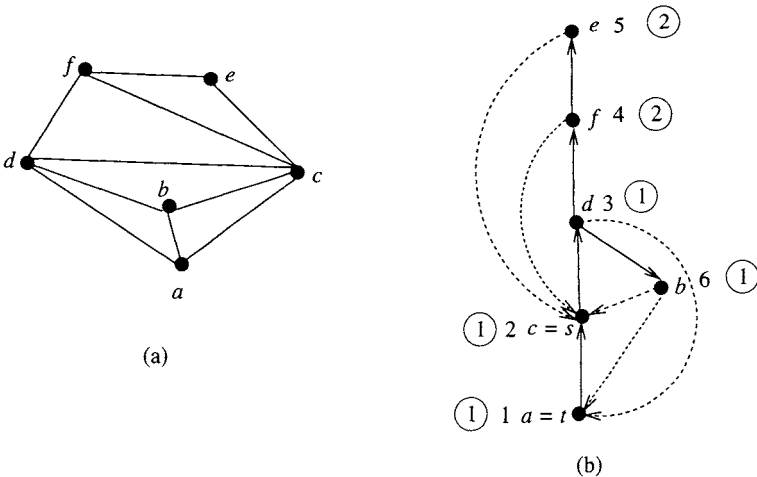


Fig. A.2 (a) Graph G , and (b) DFS tree T , DFN and LOW ; tree edges are drawn by solid lines, and back edges by dotted lines; DFS numbers are attached to vertices, and LOW numbers are written in circles.

We next present a function *PATH*. Initially, the two vertices t and s together with the first searched edge (t, s) are marked “old,” and all the other vertices and edges are marked “new.” Note that $\text{DFS}(t) = 1$ and $\text{DFS}(s) = 2$. $\text{Path}(v)$ takes as the value a path going from v to an “old” vertex. The procedure $\text{PATH}(v)$ is as follows:

Case 1: there is a “new” back edge (v, w) .

Mark (v, w) “old”;

Set $\text{PATH} = vw$;

return.

Case 2: there is a “new” tree edge (v, w) .

Let $w_0(= w)w_1w_2 \cdots w_k(= \text{LOW}(w))$ be the path which defined $\text{LOW}(w)$, that is, it runs up the tree and ends with a back edge into a vertex u such that $\text{DFN}(u) = \text{LOW}(w)$;

Set $\text{PATH} = vw_0w_1 \cdots w_k$;

Mark all the vertices and edges on the path “old”;

return.

Case 3: there is a “new” back edge (w, v) .

{ $\text{DFN}(w) > \text{DFN}(v)$ in this case};

Let $w_0(= w)w_1w_2 \cdots w_k$ be the path going backward on tree edges to an “old” vertex;

{We can trace the path using father pointers *FATH*.}

Set $\text{PATH} = vw_0w_1 \cdots w_k$;

Mark all vertices and edges “old”;

return.

Case 4: otherwise, that is, all edges (v, w) incident to v are “old”.

Set $\text{PATH} = \emptyset$;

return.

We now give one example. Mark vertices a and c and edge (a, c) of G in Fig. A.2(b) “old,” and repeatedly apply the function $\text{PATH}(c)$ until $\text{PATH}(c) = \emptyset$, then we successively get paths

$$\text{PATH}(c) = cda, \quad (\text{A.1})$$

$$\text{PATH}(c) = cbd, \quad (\text{A.2})$$

$$\text{PATH}(c) = cefd, \quad (\text{A.3})$$

and

$$\text{PATH}(c) = cf, \quad (\text{A.4})$$

as illustrated in Fig. A.3. Then applying $\text{PATH}(b)$, we get

$$\text{PATH}(b) = ba. \quad (\text{A.5})$$

The path (A.1) applies to Case 2, and paths (A.2), (A.3), and (A.4) to Case 3, and the path (A.5) to Case 1. Thus such iterative applications of PATH partition the edges of a graph into several paths.

We are now ready to present the st -numbering algorithm.

Algorithm ST-NUMBER(G);

begin

Mark t, s and (t, s) “old” and all the vertices and edges “new”;

Push down t and s into stack S in this order;

{ s is over t }

Set $\text{COUNT} = 1$;

Pop up the top entry v from S ;

while $v \neq t$

do begin

if $\text{PATH}(v) = \emptyset$

then begin

Set $\text{STN}(v) = \text{COUNT}$ and $\text{COUNT} = \text{COUNT} + 1$

end

else begin

Let $\text{PATH}(v) = vu_1u_2 \cdots u_k w$;

Push down the vertices $u_k, u_{k-1}, \dots, u_1, v$ into S
in this order

{ v is a top entry of S }

end;

Pop up the top entry v from S

end;

Set $\text{STN}(t) = \text{COUNT}$

end.

Assume that we apply ST-NUMBER to G in Fig. A.2 with letting $s = c$ and $t = a$ and that iterative calls for PATH in the algorithm produce a sequence of paths (A.1) – (A.5). Then the stack will become as illustrated

in Fig. A.3(b) just after the path (A.3) is processed, and st -numbers will be assigned as in Fig. A.3(a). We have the following theorem on the algorithm.

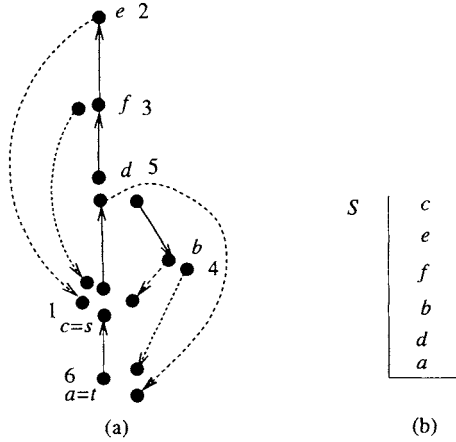


Fig. A.3 (a) st -numbering of vertices of G and (b) stack S .

Theorem A.2.1 *The algorithm $ST\text{-NUMBER}$ computes correctly an st -numbering for a 2-connected graph G .*

Proof. Every vertex v is numbered and permanently removed from stack S only after all the edges incident to v become “old.” Since G is 2-connected, every vertex v is reachable from s by a path not passing through t . Therefore every vertex is placed on S before t is removed, and hence every vertex is numbered. Thus we shall show that the numbering is indeed an st -numbering. Clearly $STN(s) = 1$ and $STN(t) = n$. Every other vertex v is placed on S , for the first time, as an intermediate vertex on a path. Moreover, the last vertex w of the path marked “old” has already been pushed and yet stays in S , since w has been incident to a “new” edge. Thus, two neighbors of v are stored in S below and above v , and hence one above v is assigned a lower number and one below v a higher number. Thus the numbering is indeed an st -numbering. \square

A.2.2 Bush Form and PQ-Tree

In this section we present a partial embedding and its related data structure required for planarity testing.

From now on, we refer to the vertices of a graph G by their st -numbers. Let $G_k = (V_k, E_k)$ be the subgraph induced by the vertices $V_k = \{1, 2, \dots, k\}$. If $k < n$, then there must exist an edge of G with one end in V_k and the other in $V - V_k$. Let G'_k be the graph formed by adding to G_k all these edges, in which the ends in $V - V_k$ of added edges are kept separate. These edges are called *virtual edges*, and their ends in $V - V_k$ are called *virtual vertices* and labeled as their counterparts in G , but they are kept separate. Thus there may be several virtual vertices with the same label, each with exactly one entering edge. Let B_k be an embedding of G'_k such that all the virtual vertices are placed on the outer face. B_k is called a *bush form* of G'_k . The virtual vertices are usually placed on a horizontal line. G, G_k , and B_k are illustrated in Fig. A.4. The following lemma implies that every planar graph G has a bush form B_k for $1 \leq k \leq n$.

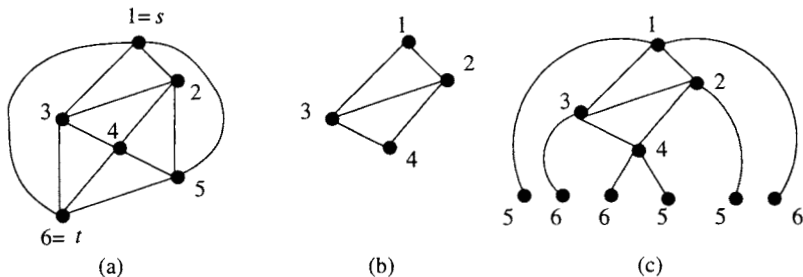


Fig. A.4 (a) An st -numbered graph G , (b) G_4 , and (c) bush form B_4 .

Lemma A.2.2 [Eve79] *Let $1 \leq k \leq n$. If edge (s, t) is drawn on the boundary of the outer face in an embedding of a planar graph G , then all vertices and edges of $G - G_k$ are drawn in the outer face of the plane subgraph G_k of G .*

Proof. Assume that some vertices of $G - G_k$ lie on a face F of G_k . Since all these vertices are higher than the vertices on F , the highest one must be sink $t (= n)$. Thus the face F must be the outer face of the plane graph G_k . □

An *upward digraph* D_u is defined to be a digraph obtained from G by assigning a direction to every edge so that it goes from the larger end to the smaller. An *upward embedding* A_u of G is an embedding of the digraph D_u . In an embedding of an *undirected* graph G , a vertex v appears in list $\text{Adj}(w)$ and w appears in list $\text{Adj}(v)$ for every edge (v, w) . However, in an upward

embedding A_u of G , the head w appears in adjacency list $A_u(v)$, but the tail v does not appear in $A_u(w)$ for every directed edge (v, w) . Figure A.5 depicts an upward digraph D_u and an upward embedding A_u for the graph G in Fig. A.1(a).

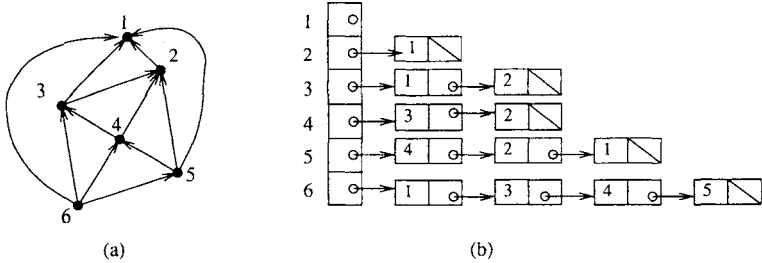


Fig. A.5 (a) Upward digraph D_u , and (b) upward embedding A_u for a graph G in Fig. A.1(a).

We use a special data structure “ PQ -tree” to represent B_k . A PQ -tree consists of “ P -nodes,” “ Q -nodes” and “leaves.” A P -node represents a cut vertex of B_k , so the sons of a P -node can be permuted arbitrarily. A Q -node represents a 2-connected component of G_k , and the sons of a Q -node are allowed only to reverse (flip over). A leaf indicates a virtual vertex of B_k . In an illustration of a PQ -tree, a P -node is drawn by a circle and a Q -node by a rectangle. A bush form B_k and a PQ -tree representing B_k are illustrated in Fig. A.6. Thus a PQ -tree represents all the permutations and reversionions possible in a bush form B_k .

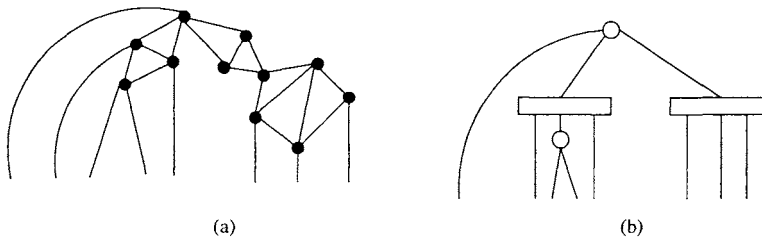


Fig. A.6 (a) Bush form B_k , and (b) PQ -tree.

The key idea of the vertex addition algorithm is to reduce the planarity testing of G_{k+1} to the problem which asks for permutations and reversionions to make all the virtual vertices labeled “ $k+1$ ” occupy consecutive positions. The following lemma guarantees that this reduction is possible.

TEAM LING - LIVE, INFORMATIVE, NON-COST AND GENUINE !

Lemma A.2.3 [Eve79] *Let B_k be any bush form of a subgraph G_k of a planar graph G . Then there exists a sequence of permutations and reversionions to make all the virtual vertices labeled " $k + 1$ " occupy consecutive positions on the horizontal line.*

Proof. We prove a more general proposition: if B and B' are two sub-bush forms, having the same vertices, of any distinct bush forms of G_k , then there exists a sequence of permutations and reversionions which transforms the sub-bush form B' into a sub-bush form B'' such that the virtual vertices appear on the horizontal line in the same order as in B .

The proof is by induction on the number of vertices in the sub-bush forms B and B' . If each of the sub-bush forms consists of only one vertex and one virtual vertex, then clearly the proposition is true.

Let v be the smallest vertex of the sub-bush forms B and B' . Consider first the case where v is a cut vertex of the sub-bush forms. Then there exist components (sub-bush forms) of B and B' , which can be decomposed at v . If the components of B' are not in the same order as in B , then by permuting them, one can put them in the same order as in B . Then, applying the inductive hypothesis to each of the components (sub-bush forms), we can get a sub-bush form B'' whose virtual vertices are in the same order as in B .

Consider next the case where v is not a cut vertex. Let H be the 2-connected component of the sub-bush forms containing v , and let u_1, u_2, \dots, u_k be the cut vertices of B contained in H which appear on the outer cycle of H in this order. These vertices in B' appear on the outer cycle of H in the same or reversed order. If the order is reversed, then reverse H about v in B' . Thus we can put the sub-bush forms of B' rooted with $u_i, 1 \leq i \leq k$, in the same order as in B . Applying the inductive hypothesis to each of the sub-bush forms, we can get B'' whose virtual vertices are in the same order as in B . \square

Booth and Lueker [BL76] showed that the permutations and reversionions mentioned in Lemma A.2.3 can be found by repeatedly applying the nine transformation rules called the *template matchings* to the PQ -tree. A leaf labeled " $k + 1$ " is said to be *pertinent* in a PQ -tree corresponding to B_k . The *pertinent subtree* is the minimal subtree of a PQ -tree containing all the pertinent leaves. A node of a PQ -tree is said to be *full* if all the leaves among its descendants are pertinent. In Figs. A.7, A.8, and A.9 the template matchings are illustrated. A pattern at the left hand side is a PQ -tree to be transformed and a pattern at the right hand side is a resulting PQ -tree.

A full node or subtree is hatched, and a Q -node which roots a pertinent subtree is hatched partially. The template matchings (c) and (e) are for the case when the root of a PQ -tree is a P -node and is the root of a pertinent subtree.

We now present an example of a reduction on a PQ -tree performed by using these template matchings. Assume that the PQ -tree shown in Fig. A.10(a) represents a bush form B_{13} of a given graph whose pertinent leaves are labeled "14." Then the PQ -tree is transformed as follows. The PQ -tree in Fig. A.10(b) is obtained by applying to a P -node P_1 the template matching of Type (d) in Fig. A.7. (Since P_1 has only one full son and one non-full son, P -nodes in the template matching (d) are not created.) Then a template matching of type (h) in Fig. A.9 applied to the PQ -tree in Fig. A.10(b) at a Q -node Q_1 produces the PQ -tree in Fig. A.10(c). A matching of Type (h) in Fig. A.9 to Q_2 in Fig. A.10(c) produces the PQ -tree in Fig. A.10(d). A matching of Type (g) in Fig. A.8 to P_2 in Fig. A.10(d) produces the PQ -tree in Fig. A.10(e). In order to construct B_{14} , the PQ -tree in Fig. A.10(f) is formed from the PQ -tree in Fig. A.10(e) by replacing all the full nodes by a single P -node having sons which correspond to the neighbors of vertex 14 having st -numbers larger than 14. One can easily observe that the next reduction for the vertex 15 will fail, that is, G_{15} is nonplanar.

A.2.3 Planarity Testing Algorithm

In this section we formally present Algorithm **Planar** for planarity testing of a given graph as follows.

Algorithm **Planar**(G)

{ G is a given graph}

begin

Assign st -numbers to all the vertices of G ;

Construct a PQ -tree corresponding to G'_1 ; {a single P -node with virtual edges incident on source $s = 1$ }

for $v = 2$ to n

do begin

 {reduction step}

 Try to gather all the pertinent leaves by repeatedly applying the template matchings from the leaves to the root of the pertinent subtree;

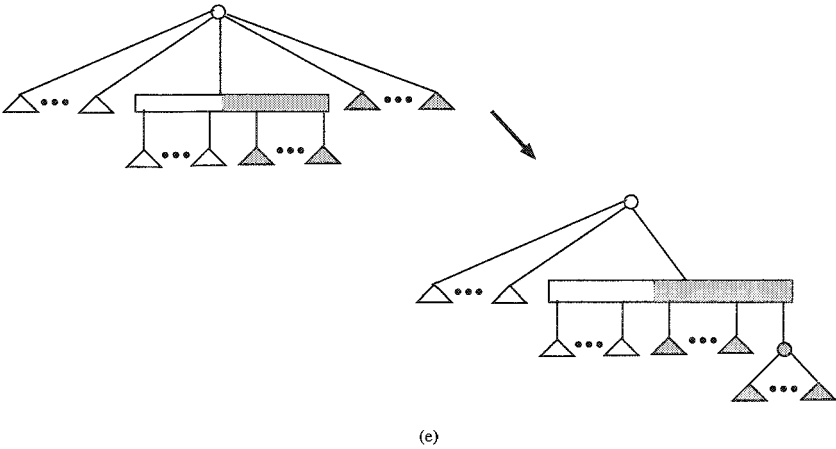
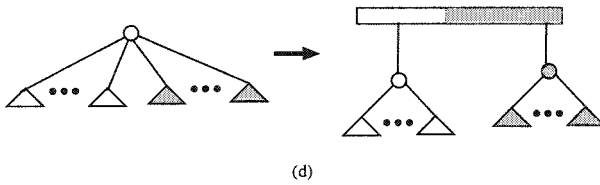
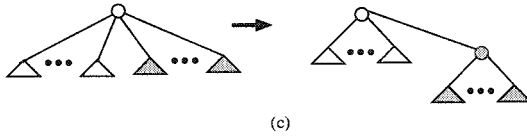
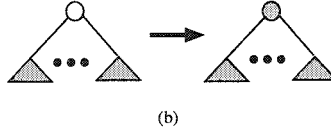
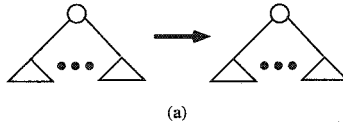
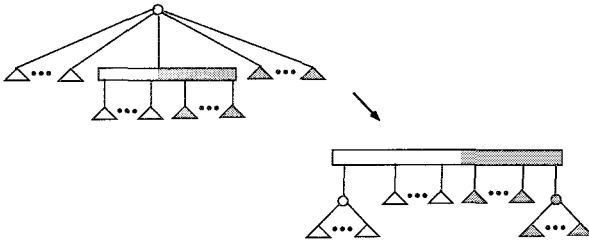
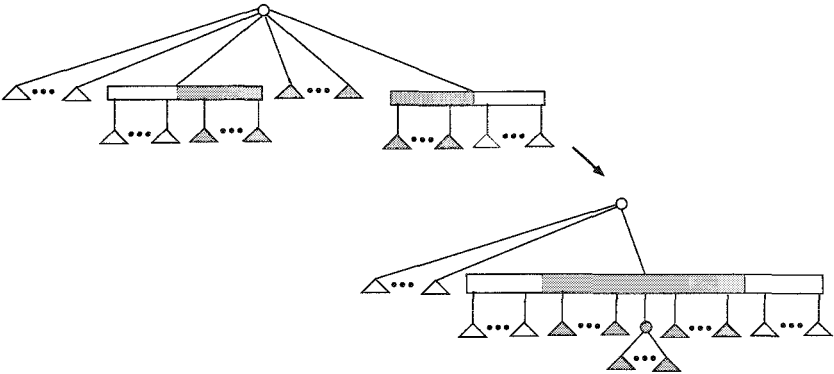


Fig. A.7 The template matchings of Types (a)–(e).



(f)



(g)

Fig. A.8 The template matchings of Types (f) and (g).

if the reduction fails

then begin

G is nonplanar;

return

end

{vertex addition step}

Replace all the full nodes of the PQ -tree by a new P -node
(which corresponds to a cut vertex v in G'_v);

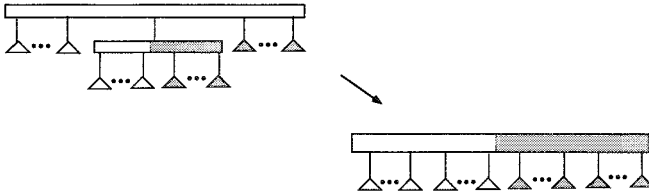
Add to the PQ -tree all the neighbors of v larger than v
as sons of the P -node

end;

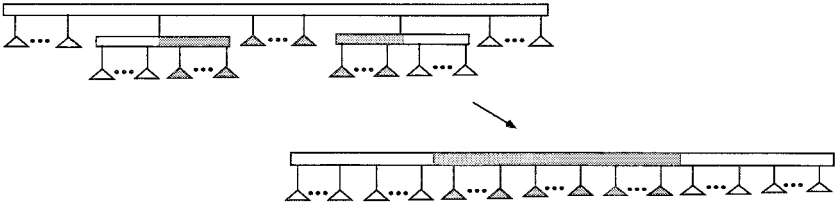
G is planar

end.

TEAM LING - LIVE, INFORMATIVE, NON-COST AND GENUINE !



(h)



(i)

Fig. A.9 The template matchings of Types (h) and (i).

Figure A.11 illustrates a sequence of bush forms together with the corresponding PQ -trees which appeared in the execution of **Planar** for the graph G in Fig. A.1(a). B'_k is a bush form just after the “reduction step” for vertex $k + 1$.

Clearly the time spent by the vertex addition step for v is proportional to the degree of v . Therefore the step spends time $O(m) = O(n)$ in total. On the other hand the time spent by the reduction step for v is proportional to the number of pertinent leaves plus the number of unary nodes in the pertinent tree. It is not straightforward to show but was shown by Booth and Lueker that all reduction steps spend time $O(n)$ in total. Thus the algorithm spends time $O(n)$ in total.

A.3 Finding Planar Embedding

One can easily have the following naive embedding algorithm: first write down the partial embedding of the graph corresponding to B_1 ; and, with each reduction of the PQ -tree, rewrite (the adjacency lists of) the bush form. Clearly the final bush form is indeed an embedding of the graph.

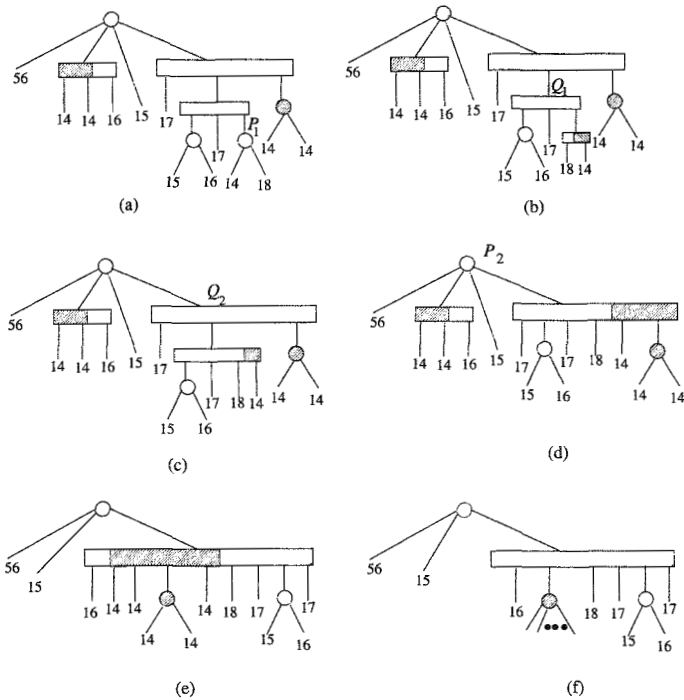


Fig. A.10 An example of a reduction of a PQ -tree.

Unfortunately the algorithm spends time $O(n^2)$, since it takes time $O(n)$ per reduction step of the PQ -tree to update the adjacency lists of the bush form.

In this section, we present a linear-time embedding algorithm **Embed** [CNAO85]. The algorithm **Embed** runs in two phases: in the first phase *procedure Upward-Embed* determines an upward embedding A_u of a planar graph G ; in the second phase *procedure Entire-Embed* constructs an entire embedding Adj of G from A_u . We will show that **Embed** takes linear time, by giving a linear algorithm **Entire-Embed** in Section A.3.1 and a linear algorithm **Upward-Embed** in Section A.3.2.

A.3.1 Algorithm for Extending A_u into Adj

In this subsection we describe an algorithm for the second phase. One can easily observe the following lemma.

TEAM LING - LIVE, INFORMATIVE, NON-COST AND GENUINE !

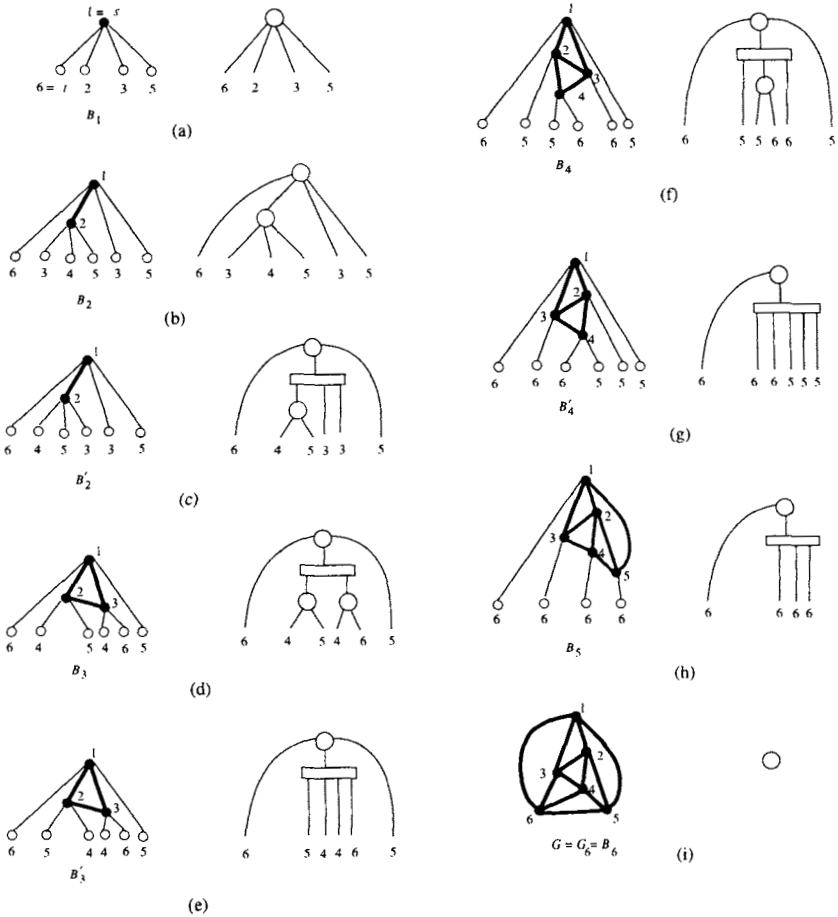


Fig. A.11 Reduction process of **Planar** for the graph G in Fig. A.1(a).

Lemma A.3.1 *Let Adj be an embedding of a planar graph G obtained by the naive algorithm above, and let v be a vertex of G . Then all the neighbors smaller than v are embedded consecutively around v . (See Fig. A.12.) That is, $Adj(v)$ does not contain four neighbors w_1, w_2, w_3 and w_4 appearing in this order and satisfying $w_1, w_3 < v$ and $w_2, w_4 > v$.*

Proof. Immediate from Lemmas A.2.2 and A.2.3. □

As shown later in Section 4.3.2, **Embed** finds an upward embedding A_u of G such that, for each vertex $v \in V$, the neighbors x_1, x_2, \dots, x_i smaller than v appear in $A_u(v)$ in this order as indicated by a dotted arrow in

Fig. A.12. That is, the v 's neighbors embedded around v counterclockwise next to the top entry x_1 of $A_u(v)$ is greater than v . In particular, the top entry of list $A_u(t)$ for the sink $t(=n)$ is the source $s(=1)$. We now present the algorithm **Entire-Embed** for extending such an upward embedding A_u into an embedding Adj of a given graph. The algorithm executes once the depth-first search starting at the sink t on an upward digraph D_u . The algorithm adds vertex y_k to the top of the list $A_u(v)$ when the directed edge (y_k, v) is searched.

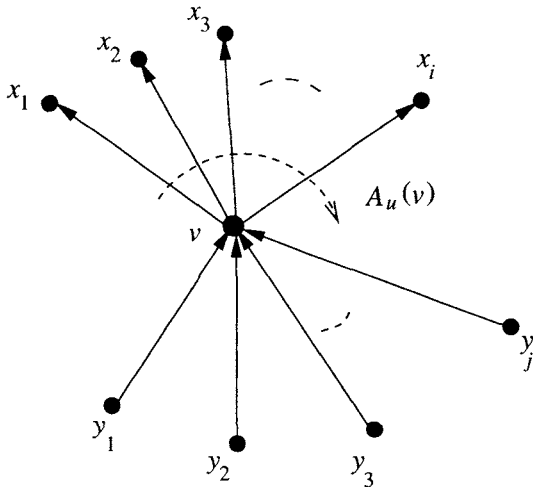


Fig. A.12 Embedding of the neighbors of v . (Number x_1, x_2, \dots, x_i are smaller than v , and y_1, y_2, \dots, y_j are greater than v .)

procedure Entire-Embed;

begin

Copy the upward embedding A_u to the lists Adj;

Mark every vertex "new";

$T := \emptyset$;

{A DFS-tree T is constructed only for analysis of the algorithm.}

DFS(t);

end;

procedure DFS(y);

begin

Mark vertex y "old";

for each vertex $v \in A_u(y)$

TEAM LING - LIVE, INFORMATIVE, NON-COST AND GENUINE !

```

do begin
  Insert vertex  $y$  to the top of  $A_u(v)$ ;
  if  $v$  is marked "new"
    then begin
      Add edge  $(y, v)$  to  $T$ ;
      DFS( $v$ );
    end
  end
end;

```

We have the following result on the algorithm.

Lemma A.3.2 *Let D_u be an upward digraph of a given graph G , and let A_u be an upward embedding of D_u . Then the algorithm **Entire-Embed** extends A_u into an embedding Adj of G in linear time.*

Proof. Clearly the algorithm terminates in linear time since the algorithm merely executes the depth-first search once. Thus we concentrate on the correctness. The definition of an st -numbering implies that there exists a directed path from t to every vertex. Therefore $DFS(t)$ traverses all the vertices and so all the directed edges of D_u . (This is not necessarily true for an arbitrary digraph.) Hence the final list $Adj(v)$ contains not only the neighbors of v larger than v but also those smaller than v . That is, the final lists Adj are surely adjacency lists of a given (undirected) graph G . Hence we shall prove that all the entries of Adj are stored correctly in clockwise order.

By Lemma A.3.1 all v 's neighbors x_1, x_2, \dots, x_i smaller than v appear in $A_u(v)$ in this order. The algorithm first copies list $A_u(v)$ to list $Adj(v)$ and then adds each neighbor y of v larger than v to the top of $Adj(v)$ in the order of the directed edge (y, v) being searched. Therefore it suffices to show that directed edges $(y_1, v), (y_2, v), \dots, (y_j, v)$ are searched in this order. (See Fig. A.12.) Assume on the contrary that (y_k, v) and (y_l, v) are searched in this order although $k > l$. Let P_k be the path from t to y_k , and let P_l be the path from t to y_l in the DFS-tree T . (See Fig. A.13.) Let z be the vertex at which P_l leaves P_k , and let $(z, y'_k) \in P_k$ and $(z, y'_l) \in P_l$. Thus the vertex y'_k precedes y'_l in $A_u(z)$. Moreover the subpaths $P'_k = z, y'_k, \dots, y_k$ of P_k and $P'_l = z, y'_l, \dots, y_l$ of P_l have no common vertices other than z . Therefore the two paths P'_k and P'_l together with two edges (y_l, v) and (y_k, v) form a cycle C . All the vertices of $A_u(v)$ must lie in the interior of the cycle; otherwise, Lemma A.3.1 would be violated. Since source $s(=1)$ is located on the boundary of the outer face,

the vertex v is not s . By the definition of an st -numbering the DFS -tree T contains a descending path P from v to s , all the vertices of which are smaller than or equal to v . Since s lies in the exterior of the cycle C , P must intersect the cycle C . However all the vertices of C are larger than or equal to v . This is a contradiction. \square

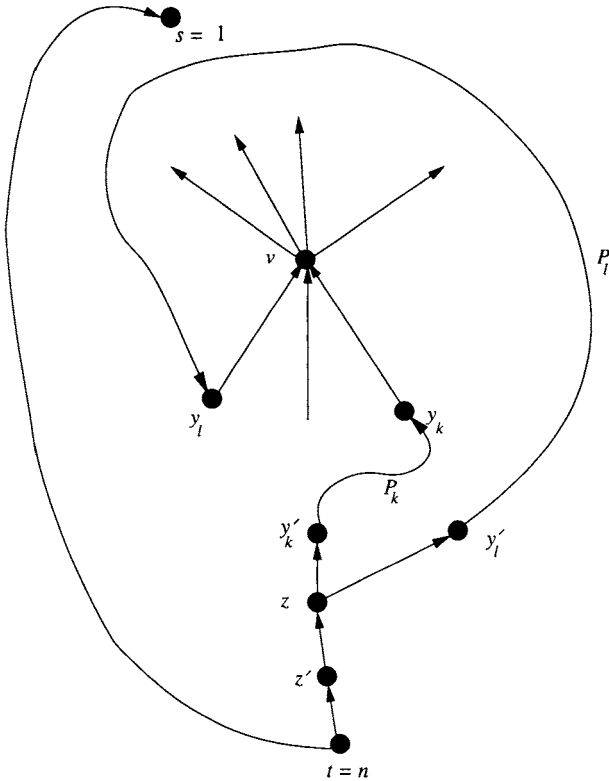


Fig. A.13 Illustration for the proof of Lemma A.3.2.

A.3.2 Algorithm for Constructing A_u

In this subsection we give an algorithm for constructing A_u . One can easily obtain list $A_u(v)$ or its reversion by scanning the leaves labeled “ v ” in the vertex addition step for v . If $A_u(v)$ is correctly determined in the step, then, counting the number of subsequent v ’s reverions, one can correct the

direction of $A_u(v)$ simply by reversing $A_u(v)$ if the number is odd. However a naive counting algorithm takes time $O(n^2)$. Moreover, the information on “ v ” may disappear from the PQ -tree. Thus an appropriate device is necessary to trace v ’s reversions.

We first show how to scan all the leaves labeled “ v ”. The root r of the pertinent subtree can be found by “bubble up” procedure [BL76]. Let b_1, b_2, \dots, b_m be the maximal sequence of full brothers that are sons of r . (See Fig. A.14(a).) To obtain $A_u(v)$, we scan the subtree rooted at b_i by the depth-first search for $i = 1, 2, \dots, m$ in this order. In a schematic illustration of a PQ -tree, one can easily recognize the direction of the maximal sequence, that is, whether b_1, b_2, \dots, b_m are in left-to-right or right-to-left order. However in the data structure of a PQ -tree, a Q -node is doubly linked only with the endmost sons, and a son of a Q -node has pointers only to the immediate brothers [BL76]. Therefore we must traverse sons of a Q -node from a full son to one of the endmost sons, and then check the direction of the sequence by using the pointer between the endmost son and the Q -node. Thus such a straightforward method requires time $O(n)$ to determine the direction of the sequence, that is, to know whether the constructed list is either $A_u(v)$ or its reversion.

The algorithm does not determine the direction of $A_u(v)$ at the vertex addition step for v , but adds a new special node to the PQ -tree as one of r ’s sons at an arbitrary position among them. The new node is called a “direction indicator”, also labeled “ v ”, and depicted by a triangle, as illustrated in Fig. A.14(b). The indicator “ v ” plays two roles. The first is to trace the subsequent reversions of v . The indicator will be reversed with each reversion of its father. (No physical action is taken in the indicator’s reversion — it is only done implicitly.) The second is to bear the relative direction of node v to its brothers. When the rightmost or leftmost brother of “ v ” is subsequently scanned together with the indicator “ v ”, the direction of the constructed $A_u(v)$ is known and so is corrected if necessary.

In the template matching algorithm, we ignore the presence of the direction indicators. When we access an immediate brother b of a node v , we skip the direction indicators between v and b if any. When we change pointers of a PQ -tree in a reduction step, we treat a direction indicator as a usual node of a PQ -tree. Note that all the direction indicators in a PQ -tree are necessarily leaves: none of the direction indicators has a son. Now we redefine a node to be *full* if all the leaves of its descendants that are not indicators are labeled “ v ”.

Thus we modify the vertex addition step in **Planar** as follows and call

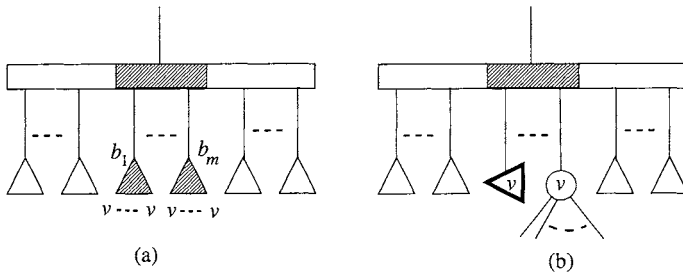


Fig. A.14 (a) Direction of a scanning, and (b) direction indicator “ v .”

the modified algorithm **Upward-Embed**.

{Vertex addition step}

begin

Let l_1, l_2, \dots, l_j be the leaves labeled “ v ”, and let f_1, f_2, \dots, f_k be the direction indicators scanned (using the DFS procedure just described) in this order;

{It is not necessary to recognize here whether l_1, l_2, \dots, l_j are in the left-to-right order.}

$A_u(v) = \{l_1, l_2, \dots, l_j\}$;

if root r of the pertinent subtree is not full

{The subtree has a leaf which is not an indicator and not labeled “ v ”}

then {the root r is a Q -node }

begin

Add an indicator “ v ” directed from l_j to l_1 to the PQ -tree as a son of the Q -node r at an arbitrary position among the sons;

Add the direction indicators f_1, f_2, \dots, f_k as sons of the Q -node r at arbitrary positions among the sons

end

else

begin {The pertinent subtree corresponds to a reversible component in an embedding of G , that is, the cut vertex of G corresponding to root r forms a “separation pair” with vertex v . Therefore we may assume that $A_u(v)$ is correctly in clockwise order.}

Delete f_1, f_2, \dots, f_k from the PQ -tree;

TEAM LING - LIVE, INFORMATIVE, NON-COST AND GENUINE !


```

for  $i = 1$  to  $k$  do
  if indicator  $f_i$  is directed from  $l_1$  to  $l_k$ 
  then reverse the adjacency list  $A_u(f_i)$ ;
    {The order of  $A_u(f_i)$  is corrected with the assumption
    that  $A_u(v)$  is in clockwise order.}
  end;
if root  $r$  is not full
then replace all the full sons of  $r$  by a  $P$ -node
  {which corresponds to a cut vertex  $v$  of  $G'_v$ }
  else replace the pertinent subtree by a  $P$ -node;
  Add all the virtual vertices adjacent to  $v$  (i.e. all neighbors of  $v$ 
  in  $G$  greater than  $v$ ) to the  $PQ$ -tree as the sons of the  $P$ -node
end;

```

We have the following result on **Upward-Embed**.

Lemma A.3.3 *The algorithm Upward-Embed obtains an upward embedding A_u of a given planar graph G .*

Proof. Let $v \in V$. Clearly the list $A_u(v)$ obtained by Upward-Embed contains all the neighbors of v smaller than v . Furthermore these vertices appear in either clockwise or counterclockwise order around v . Therefore we shall show that the vertices in each $A_u(v)$ appear in clockwise order. It suffices to consider the following two cases.

Case 1: the direction indicator “ v ” is not added to the PQ -tree.

The leaves of the pertinent subtree which are not indicators are all labeled “ v ” at the vertex addition step for v . Such a pertinent subtree corresponds to a reversible component in a plane embedding of G . (See Fig. A.15.) Therefore one may assume that the vertices in $A_u(v)$ appear in clockwise order even in the final embedding.

Case 2: the direction indicator “ v ” is added to the PQ -tree.

When the algorithm terminates, the PQ -tree consists of exactly one isolated P -node, and hence has no direction indicators in particular. That is, every indicator will be eventually deleted. Therefore one can assume that the indicator “ v ” is deleted in the vertex addition step for a vertex $w(> v)$. The direction indicator “ v ” follows reversions of the Q -node which is the father of node “ v ” as long as v remains in a PQ -tree. Therefore if the direction of indicator “ v ” is opposite relative to the scanning of the leaves l_1, l_2, \dots, l_j labeled “ w ”, then either the order (clockwise or counterclockwise) of $A_u(v)$ is the same as $A_u(w)$ and vertex v is reversed an odd

number of times, or the order of $A_u(v)$ is opposite to that of $A_u(w)$ and the vertex v is reversed an even number of times. In either case, we can correct adjacency list $A_u(v)$ simply by reversing it. Since the pertinent subtree for w corresponds to a reversible component of G , the direction indicator “ w ” is not added to the PQ -tree. Hence the adjacency lists $A_u(v)$ and $A_u(w)$ are never reversed after the vertex addition step for w . Thus $A_u(v)$ remains correctly in clockwise order. \square

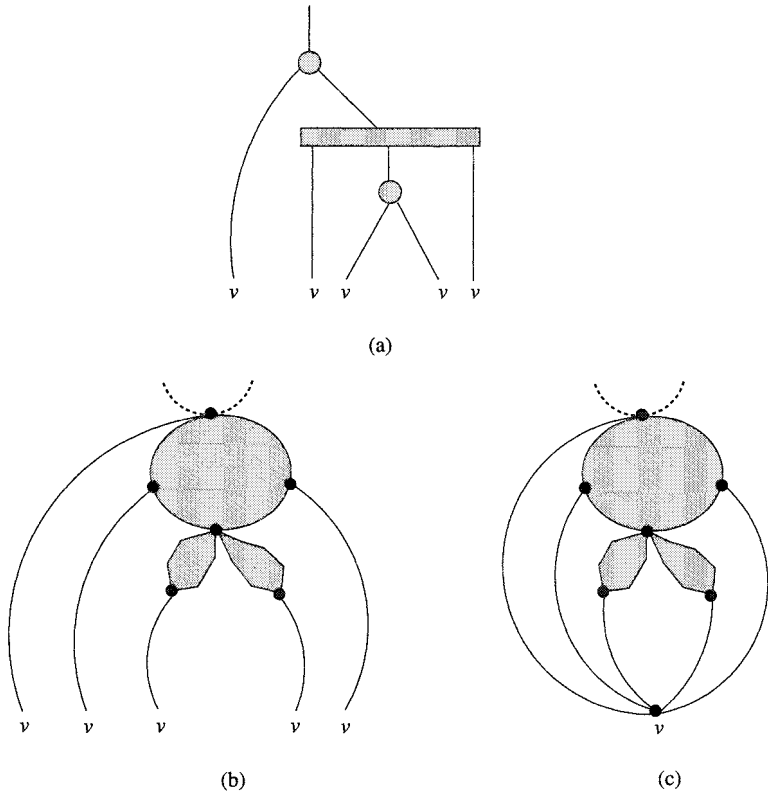


Fig. A.15 Reversible component. (a) pertinent subtree, (b) B_{v-1} , and (c) G_v .

However, Algorithm **Upward-Embed**, as it is, requires time $O(n^2)$ since it may scan the same indicator many times, say $O(n)$ times. Thus we shall refine the algorithm so that it takes time $O(n)$.

Now consider the role of a direction indicator in detail. Assume that root r of a pertinent subtree is not full, and define indicators “ v ” and

f_1, f_2, \dots, f_k as in the algorithm. After the direction indicator “ v ” is added to a PQ -tree, indicators “ v ” and f_1, f_2, \dots, f_k are reversed all together. Therefore it suffices to remember the directions of f_1, f_2, \dots, f_k relative to that of “ v ”. Thus we delete the indicators f_1, f_2, \dots, f_k from the PQ -tree and store them in $A_u(v)$ together with vertices l_1, l_2, \dots, l_j . Once the correct order of adjacency list $A_u(v)$ is known, we can easily correct the orders of adjacency lists $A_u(f_i)$, $1 \leq i \leq k$, simply by checking the direction of indicator f_i in $A_u(v)$. We execute such a correction for each v , $v = n, n - 1, \dots, 1$, in this order.

The following is the algorithm **Upward-Embed** refined as above. Figure A.16 illustrates an **Upward-Embed** applied to the graph in Fig. A.5(a).

procedure Upward-Embed(G);

{ G is a given planar graph.}

begin

Assign st -numbers to all the vertices of G ;

Construct a PQ -tree corresponding to G'_1 ;

for $v = 2$ to n

do begin

{reduction step}

Apply the template matchings to the PQ -tree, ignoring the direction indicators in it, so that the leaves labeled v occupy consecutive positions;

{vertex addition step}

Let l_1, l_2, \dots, l_k be the leaves labeled v and direction indicators scanned in this order;

Delete l_1, l_2, \dots, l_k from the PQ -tree and store them in $A_u(v)$;

if root r of the pertinent subtree is not full

then begin

Add an indicator “ v ”, directed from l_k to l_1 , to the PQ -tree as a son of root r at an arbitrary position among the sons;

Replace all the full sons of r by a new P -node

end

else

Replace the pertinent subtree by a new P -node;

Add to the PQ -tree all the virtual edges adjacent to v as the sons of the P -node

end;

```

{correction step}
for  $v = n$  down to 1
  do for each element  $x$  in  $A_u(v)$ 
    do if  $x$  is a direction indicator
      then begin
        Delete  $x$  from  $A_u(v)$ ;
        Let  $w$  be the label of  $x$ ;
        if the direction of indicator  $x$  is
          opposite to that of  $A_u(v)$ 
          then reverse list  $A_u(w)$ ;
      end
    end
  end;

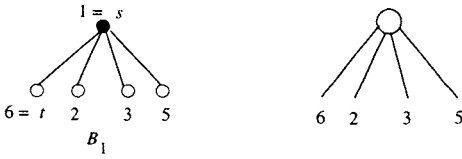
```

Lemma A.3.4 *Algorithm Upward-Embed obtains the upward embedding A_u of a given planar graph in linear time.*

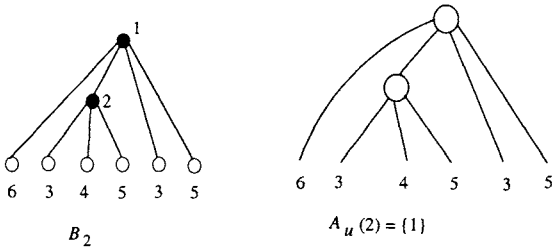
Proof. Noting the role of a direction indicator, one can easily verify the correctness of the algorithm. Therefore we consider the time required by the algorithm. At most $O(n)$ direction indicators are generated during an execution of the algorithm. A direction indicator scanned in a reduction step will be necessarily deleted from a PQ -tree in a succeeding vertex addition step. Therefore each direction indicator is scanned at most once. Thus **Upward-Embed** requires time $O(n)$ in addition to the time required by the linear testing algorithm **Planar**. Therefore **Upward-Embed** runs in linear time. \square

A.4 Bibliographic Notes

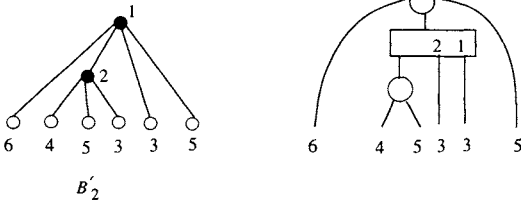
A detailed description of the embedding phase of the Hopcroft and Tarjan planarity testing algorithm can be found in [MM96]. Shih and Hsu [SH92, SH99] presented a simple algorithm for planarity testing and embedding based on a depth-first search tree. Another linear algorithm for planarity testing based on depth-first search is presented in [BM99, BCPD04].



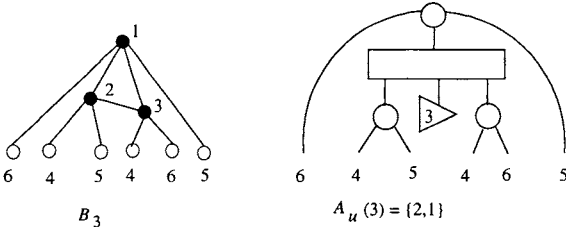
(a)



(b)

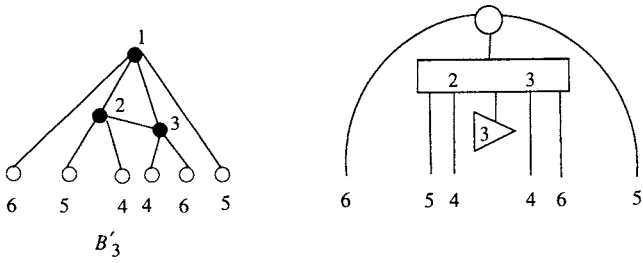


(c)

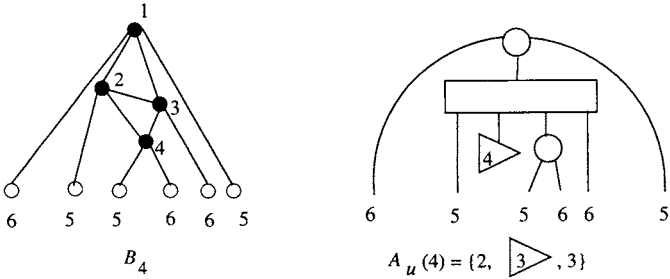


(d)

Fig. A.16 Process of Upward-Embed applied to graph G in Fig. A.5(a); (a)–(d) bush forms and corresponding PQ -trees and list A_u .



(e)



(f)

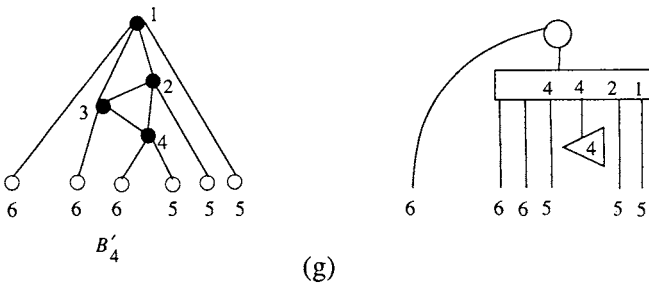
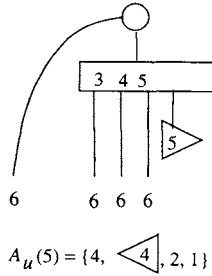
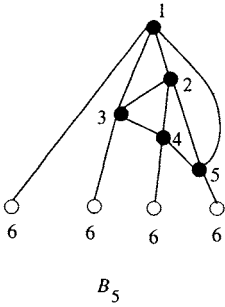
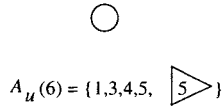
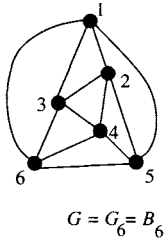


Fig. A.17 Process of Upward-Embed applied to graph G in Fig. A.5(a) (continuation of Fig. A.16); (e)–(g) bush forms and corresponding PQ -trees and list A_u .



(h)



(i)

$$A_u(1) = \emptyset$$

$$A_u(4) = \{3, 2\}$$

$$A_u(2) = \{1\}$$

$$A_u(5) = \{4, 2, 1\}$$

$$A_u(3) = \{1, 2\}$$

$$A_u(6) = \{1, 3, 4, 5\}$$

(j)

Fig. A.18 Process of Upward-Embed applied to graph G in Fig. A.5(a) (continuation of Figs. A.16 and A.17); (h)-(i) bush forms and corresponding PQ -trees and list A_u , and (j) corrected lists A_u .

Bibliography

- [AHU74] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
- [BCPD04] J. M. Boyer, P. F. Cortese, M. Patrignani and G. Di Battista, *Stop minding your P's and Q's: implementing a fast and simple DFS-based planarity testing and embedding algorithm*, Proc. of Graph Drawing 2003, Lect. Notes in Compt. Sci., Springer, 2912, pp. 25-36, 2004.
- [Bie96a] T. C. Biedl, *New lower bounds for orthogonal graph drawings*, Proc. Graph Drawing '95, Lect. Notes in Compt. Sci., Springer, 1027, pp. 28-39, 1996.
- [Bie96b] T. C. Biedl, *Optimal orthogonal drawings of triconnected plane graphs*, Proc. Scandinavian Workshop on Algorithm Theory, SWAT '96, Lect. Notes in Compt. Sci., Springer, 1097, pp. 333-344, 1996.
- [Bie97] T. C. Biedl, *Orthogonal Graph Visualization: The Three-Phase Method with Applications*, Ph. D. Thesis, RUTCOR, Rutgers University, 1997.
- [Bie98] T. C. Biedl, *Relating bends and size in orthogonal graph drawings*, Information Processing Letters, 65, pp. 111-115, 1998.
- [BCF01] J. D. Boissonnat, F. Cazals, and J. Flötotto, *2D-structure drawings of similar molecules*, Proc. of Graph Drawing 2000, Lect. Notes in Compt. Sci., Springer, 1984, pp. 115-126, 2001.
- [BK97] T. C. Biedl and M. Kaufmann, *Area-efficient static and incremental graph drawings*, Proc. of 5th European Symposium on Algorithms, Lect. Notes in Compt. Sci., Springer, 1284, pp. 37-52, 1997.
- [BK98] T. C. Biedl and G. Kant, *A better heuristic for orthogonal graph drawings*, Comp. Geom. Theo. Appl., 9, pp. 159-180, 1998.
- [BL76] K. S. Booth and G. S. Lueker, *Testing the consecutive ones property*,

- interval graphs, and graph planarity using PQ-tree algorithms*, J. Comput. Syst. Sci., 13, pp. 335-379, 1976.
- [BM99] J. Boyer and W. Myrvold, *Stop minding your P's and Q's: A simplified $O(n)$ planar embedding algorithm*, Proc. of SIAM-ACM Symposium on Discrete Algorithms, pp. 140-146, 1999.
- [Bra96] F. J. Brandenburg (editor), *Graph Drawing (Proc. of GD '95)*, Lect. Notes in Computer Science, Springer, 1027, 1996.
- [BS88] J. Bhasker and S. Sahni, *A linear algorithm to find a rectangular dual of a planar triangulated graph*, Algorithmica, 3, pp. 247-278, 1988.
- [BSM02] N. Bonichon, B. L. Saëc, and M. Mosbah, *Optimal area algorithm for planar polyline drawings*, Proc. of WG 2002, Lect. Notes in Computer Science, Springer, 2573, pp. 35-46, 2002.
- [BW76] N. L. Biggs and L. Wilson, *Graph Theory 1736-1936*, Clarendon Press, Oxford, 1976.
- [CE95] I. F. Cruz and P. Eades (editors), *Special Issue on Graph Visualization*, J. Visual Languages and Computing, 6 (3), 1995.
- [CK97] M. Chrobak and G. Kant, *Convex grid drawings of 3-connected planar graphs*, Inter. J. of Compu. Geom. and Appl., 7 (3), pp. 211-223, 1997.
- [CLL01] Y.-T. Chiang, C.-C. Lin and H.-I. Lu, *Orderly spanning trees with applications to graph encoding and graph drawing*, Proc. of SODA 2001, pp. 506-515, 2001.
- [CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, Cambridge, MIT Press, MA, 1990.
- [CN98] M. Chrobak and S. Nakano, *Minimum-width grid drawings of plane graphs*, Comp. Geom. Theory and Appl., 11, pp. 29-54, 1998.
- [CNAO85] N. Chiba, T. Nishizeki, S. Abe and T. Ozawa, *A linear algorithm for embedding planar graphs using PQ-trees*, J. Comput. Syst. Sci., 30, pp. 54-76, 1985.
- [CON85] N. Chiba, K. Onoguchi, and T. Nishizeki, *Drawing planar graphs nicely*, Acta Informatica, 22, pp. 187-201, 1985.
- [CP95] M. Chrobak and T. H. Payne, *A linear-time algorithm for drawing a planar graph on a grid*, Information Processing Letters, 54, pp. 241-246, 1995.
- [CR76] S. A. Cook and R. A. Reckhow, *Time bounded random access machines*, J. Comput. Syst. Sci., 7, pp. 354-375, 1976.

- [CYN84] N. Chiba, T. Yamanouchi, and T. Nishizeki, *Linear algorithms for convex drawings of planar graphs*, (Eds.) J. A. Bondy and U. S. R. Murty, Progress in Graph Theory, Academic Press Canada, pp. 153-173, 1984.
- [DDLW03] E. Di Giacomo, W. Didimo, G. Liotta and S. K. Wismath, *Drawing planar graphs on a curve*, Proc. WG '03, Lect. Notes in Computer Science, Springer, 2880, pp. 192-204, 2003.
- [DETT94] G. Di Battista, P. Eades, R. Tamassia and I. G. Tollis, *Algorithms for drawing graphs: an annotated bibliography*, Comp. Geom. Theory and Appl., 4, pp. 235-282, 1994.
- [DETT99] G. Di Battista, P. Eades, R. Tamassia, I. G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice-Hall Inc., Upper Saddle River, New Jersey, 1999.
- [Dib97] G. Di Battista (editor), *Graph Drawing (Proc. of GD '97)*, Lect. Notes in Computer Science, Springer, 1353, 1997.
- [DLV98] G. Di Battista, G. Liotta and F. Vargiu, *Spirality and optimal orthogonal drawings*, SIAM J. Comput., 27(6), pp. 1764-1811, 1998.
- [DM99] G. Di Battista and P. Mutzel (editors), *New Trends in Graph Drawing: Special Issue on Selected Papers from the 1997 Symposium on Graph Drawing*, Journal of Graph Alg. and Appl., 3(4), 1999.
- [DT96] G. Di Battista and R. Tamassia (editors), *Special Issue on Graph Drawing*, Algorithmica, 16 (1), 1996.
- [DT98] G. Di Battista and R. Tamassia (editors), *Special Issue on Geometric Representations of Graphs*, Comput. Geom. Theory and Appl., 9 (1-2), 1998.
- [DTV99] G. Di Battista, R. Tamassia and L. Vismara, *Output-sensitive reporting of disjoint paths*, Algorithmica, 23, pp. 302-340, 1999.
- [ET76] S. Even and R. E. Tarjan, *Computing an st-numbering*, Theor. Comput. Sci., 2, pp. 339-344, 1976.
- [Eve79] S. Even, *Graph Algorithms*, Computer Science Press, Potomac, 1979.
- [Far48] I. Fáry, *On straight line representations of planar graphs*, Acta Sci. Math. Szeged, 11, pp. 229-233, 1948.
- [Fel01] S. Felsner, *Convex drawings of planar graphs and order dimension of 3-polytopes*, Order, 18, pp. 19-37, 2001.
- [FKK97] U. Fößmeier, G. Kant and M. Kaufmann, *2-visibility drawings of plane graphs*, Proc. of Graph Drawing '96, Lect. Notes in Computer Science, Springer, 1190, pp. 155-168, 1997.

- [FPP90] H. de Fraysseix, J. Pach and R. Pollack, *How to draw a planar graph on a grid*, *Combinatorica*, 10, pp. 41-51, 1990.
- [FW74] R. L. Francis and J. A. White, *Facility Layout and Location*, Prentice-Hall, New Jersey, 1974.
- [FW91] M. Formann and F. Wagner, *The VLSI layout problem in various embedding models*, Proc. International Workshop on Graph Theoretic Concepts in Computer Science (WG '91), Lect. Notes in Compt. Sci., Springer, 484, pp. 130-139, 1991.
- [GJ83] M. R. Garey and D. S. Johnson, *Crossing number is NP-complete*, *SIAM J. Alg. Disc. Methods*, 4 (3), pp. 312-316, 1983.
- [GK02] M. T. Goodrich and S. G. Kobourov (editors), *Graph Drawing (Proc. GD '02)*, Lect. Notes in Computer Science, Springer, 2528, 2002.
- [GL99] A. Garg and G. Liotta, *Almost bend-optimal planar orthogonal drawings of biconnected degree-3 planar graphs in quadratic time*, Proc. of Graph Drawing '99, Lect. Notes in Compt. Sci., Springer, 1731, pp. 38-48, 1999.
- [GM98] C. Gutwenger and P. Mutzel, *Planar polyline drawings with good angular resolution*, Proc. of Graph Drawing '98, Lect. Notes in Compt. Sci., Springer, 1547, pp. 167-182, 1998.
- [GPS03] H. H. Gan, S. Pasquali and T. Schlick, *Exploring the repertoire of RNA secondary motifs using graph theory; implications for RNA design*, *Nucleic Acids Research*, 31 (11), pp. 2926-2943, 2003.
- [GT02] M. T. Goodrich and R. Tamassia, *Algorithm Design*, John Wiley & Sons, New York, 2002.
- [GT01] A. Garg and R. Tamassia, *On the computational complexity of upward and rectilinear planarity testing*, *SIAM J. Comput.*, 31(2), pp. 601-625, 2001.
- [GT97] A. Garg and R. Tamassia, *A new minimum cost flow algorithm with applications to graph drawing*, Proc. of Graph Drawing '96, Lect. Notes in Compt. Sci., Springer, 1190, pp. 201-216, 1997.
- [Har72] F. Harary, *Graph Theory*, Addison-Wesley, Reading, Mass., 1972.
- [HE03] S.-H. Hong and P. Eades, *Drawing trees symmetrically in three dimensions*, *Algorithmica*, 36(2), pp. 153-178, 2003.
- [He01] X. He, *A simple linear time algorithm for proper box rectangular drawings of plane graphs*, *Journal of Algorithms*, 40(1), pp. 82-101, 2001.
- [He93] X. He, *On finding the rectangular duals of planar triangular graphs*,

- SIAM J. Comput., 22(6), pp. 1218-1226, 1993.
- [He95] X. He, *An efficient parallel algorithm for finding rectangular duals of plane triangulated graphs*, *Algorithmica*, 13, pp. 553-572, 1995.
- [He97] X. He, *Grid embedding of 4-connected plane graphs*, *Discrete Comput. Geom.*, 17, pp. 339-358, 1997.
- [HK73] J. E. Hopcroft and R. M. Karp, *An $n^{5/2}$ algorithm for maximum matching in bipartite graphs*, *SIAM J. Comput.*, 2, pp. 225-231, 1973.
- [HT73] J. E. Hopcroft and R. E. Tarjan, *Dividing a graph into triconnected components*, *SIAM J. Comput.*, 2(3), pp. 135-158, 1973.
- [HT74] J. E. Hopcroft and R. E. Tarjan, *Efficient planarity testing*, *J. Assoc. Comput. Mach.*, 21, pp. 549-568, 1974.
- [JM04] M. Jünger and P. Mutzel (editor), *Graph Drawing Software*, Springer, Berlin, 2004.
- [Kam76] G. R. Kampen, *Orienting planar graphs*, *Discrete Mathematics*, 14, pp. 337-341, 1976.
- [Kan93] G. Kant, *Algorithms for Drawing Planar Graphs*, Ph. D. Thesis, Faculty of Information Science, Utrecht University, 1993.
- [Kan96] G. Kant, *Drawing planar graphs using the canonical ordering*, *Algorithmica*, 16, pp. 4-32, 1996.
- [Kan97] G. Kant, *A more compact visibility representation*, *Int. J. Comput. Geometry Appl.*, 7 (3), pp. 197-210, 1997.
- [Kau02] M. Kaufmann (editor), *Special Issue on Selected Papers from the 2000 Symposium on Graph Drawing*, *Journal of Graph Alg. and Appl.*, 6(3), (2002).
- [KFHR94] M. Y. Kao, M. Fürer, X. He and B. Raghavachari, *Optimal parallel algorithms for straight-line grid embeddings of planar graphs*, *SIAM J. Discrete Math.*, 7 (4), pp. 632-646, 1994.
- [KH97] G. Kant and X. He, *Regular edge labeling of 4-connected plane graphs and its applications in graph drawing problems*, *Theo. Comput. Sci.*, 172, pp. 175-193, 1997.
- [KK84] K. Kozminski and E. Kinnen, *An algorithm for finding a rectangular dual of a planar graph for use in area planning for VLSI integrated circuits*, *Proc. 21st DAC*, Albuquerque, pp. 655-656, 1984.
- [KL84] M. R. Kramer and J. van Leeuwen, *The complexity of wire routing and finding minimum area layouts for arbitrary VLSI circuits*, (Eds.) F. P. Preparata, *Advances in Computing Research*, vol. 2:

VLSI Theory, JAI Press, Reading, MA, pp. 129-146, 1984.

- [KMBW02] E. Kruja, J. Marks, A. Blair and R. Waters, *A short note on the history of graph drawing*, Proc. of Graph Drawing 2001, Lect. Notes in Compt. Sci., Springer, 2265, pp. 272-286, 2002.
- [Kra99] J. Kratochvíl (editor), *Graph Drawing (Proc. of GD '99)*, Lect. Notes in Computer Science, Springer, 1731, 1999.
- [Kur30] C. Kuratowski, *Sur le problème des courbes gauches en topologie*, Fundamenta Math., 15, pp. 271-283, 1930.
- [KW01] M. Kaufmann and D. Wagner (Eds.), *Drawing Graphs: Methods and Models*, Lect. Notes in Compt. Sci., Springer, 2025, Berlin, 2001.
- [LEC67] A. Lempel, S. Even and I. Cederbaum, *An algorithm for planarity testing of graphs*, Int. Symposium on Theory of Graphs 1966, Ed. P. Rosenstiehl, (Gordon and Breach, New York, 1967), pp. 215-232, 1967.
- [Len90] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, John Wiley & Sons, Chichester, 1990.
- [Lio04] G. Liotta (editor), *Graph Drawing (Proc. of GD '03)*, Lect. Notes in Computer Science, Springer, 2912, 2004.
- [LL90] Y.-T. Lai and S. M. Leinwand, *A theory of rectangular dual graphs*, Algorithmica, 5, pp. 467-483, 1990.
- [LRT79] R. J. Lipton, D. J. Rose, and R. E. Tarjan, *Generalized nested dissections*, SIAM J. Numer. Anal., 16(2), pp. 346-358, 1979.
- [Lub81] A. Lubiw, *Some NP-complete problems similar to graph isomorphism*, SIAM J. on Computing, 10(1), pp. 11-21, 1981.
- [LW00] G. Liotta and S. H. Whitesides (editors), *Special Issue on Selected Papers from the 1998 Symposium on Graph Drawing*, Journal of Graph Alg. and Appl., 4(3), 2000.
- [Man91] J. Manning, *Computational complexity of geometric symmetry detection in graphs*, Proc. of Great Lakes Computer Science Conference '89, Lecture Notes in Computer Science, 507, pp.1-7, 1991.
- [MAN04] K. Miura, M. Azuma and T. Nishizeki, *Canonical decomposition, realizer, Schnyder labeling and orderly spanning trees of plane graphs*, Proc. of COCOON 2004, Lecture Notes in Computer Science, to appear.
- [Mar01] J. Marks (editor), *Graph Drawing (Proc. of GD'00)*, Lect. Notes in Computer Science, Springer, 1984, 2001.

- [MHN04] K. Miura, H. Haga and T. Nishizeki, *Inner rectangular drawings of plane graphs*, Working paper, 2004.
- [MM96] K. Mehlhorn and P. Mutzel, *On the embedding phase of the Hopcroft and Tarjan planarity testing algorithm*, *Algorithmica*, 16, pp. 233-242, 1996.
- [MMN02] K. Miura, A. Miyazawa and T. Nishizeki, *Extended rectangular drawings of plane graphs with designated corners*, *Proc. of GD '02*, *Lect. Notes in Comput. Sci.*, Springer, 2528, pp. 256-267, 2002.
- [MNN01] K. Miura, S. Nakano and T. Nishizeki, *Grid drawings of 4-connected plane graphs*, *Discrete & Computational Geometry*, 26(1), pp. 73-87, 2001.
- [MNN00] K. Miura, S. Nakano and T. Nishizeki, *Convex grid drawings of four-connected plane graphs*, *Proc. 11th Annual International Symposium on Algorithms and Computation (ISAAC '00)*, *Lect. Notes in Comput. Sci.*, Springer, 1969, pp. 254-265, 2000.
- [MTNN99] K. Miura, D. Takahashi, S. Nakano and T. Nishizeki, *A linear-time algorithm to find four independent spanning trees in four-connected planar graphs*, *International Journal of Foundation of Computer Science*, 10(2), pp. 195-210, 1999.
- [MV80] S. Micali and V. V. Vazirani, *An $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matching in general graphs*, *Proc. 21st Annual Symposium on Foundations of Computer Science*, pp. 17-27, 1980.
- [NC88] T. Nishizeki and N. Chiba, *Planar Graphs: Theory and Algorithms*, North-Holland, Amsterdam, 1988.
- [Nor97] S. North (editor), *Graph Drawing (Proc. of GD '96)*, *Lect. Notes in Computer Science*, Springer, 1190, 1997.
- [NRN97] S. Nakano, M. S. Rahman and T. Nishizeki, *A linear-time algorithm for four-partitioning four-connected planar graphs*, *Information Processing Letters*, 62, pp. 315-322, 1997.
- [PS82] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization*, Prentice Hall, Englewood Cliffs, New Jersey, 1982.
- [PT00] A. Papakostas and I. G. Tollis, *Efficient orthogonal drawings of high degree graphs*, *Algorithmica*, 26, pp. 100-125, 2000.
- [PT95] A. Papakostas and I. G. Tollis, *Improved algorithms and bounds for orthogonal drawings*, *Proc. of GD '94*, *Lect. Notes in Computer Science*, Springer, 894, pp. 40-51, 1995.
- [PT97] A. Papakostas and I. G. Tollis, *A pairing technique for area efficient orthogonal drawings*, *Proc. of GD '96*, *Lect. Notes in Computer*

Science, Springer, 1190, pp. 355-370, 1997.

- [Rah99] M. S. Rahman, *Efficient Algorithms for Drawing Planar Graphs*, Ph. D. Thesis, Graduate School of Information Sciences, Tohoku University, Sendai, Japan, 1999.
- [REN04] M. S. Rahman, N. Egi and T. Nishizeki, *No-bend orthogonal drawings of subdivisions of planar triconnected cubic graphs*, Proc. of GD '03, Lect. Notes in Computer Science, Springer, 2912, pp. 387-392, 2004.
- [RMN04] M. S. Rahman, K. Miura and T. Nishizeki, *Octagonal drawings of plane graphs*, Proc. International Workshop on Graph Theoretic Concepts in Computer Science (WG '04), Lect. Notes in Computer Science, Springer, 2004 (to appear).
- [RN02] M. S. Rahman and T. Nishizeki, *Bend-minimum orthogonal drawings of plane 3-graphs*, Proc. International Workshop on Graph Theoretic Concepts in Computer Science (WG '02), Lect. Notes in Computer Science, Springer, 2573, pp. 367-378, 2002.
- [RNG04] M. S. Rahman, T. Nishizeki, and S. Ghosh *Rectangular drawings of planar graphs*, Journal of Algorithms, 50, pp. 62-78, 2004.
- [RNN00] M. S. Rahman, S. Nakano and T. Nishizeki, *Box-rectangular drawings of plane graphs*, Journal of Algorithms, 37, pp. 363-398, 2000.
- [RNN02] M. S. Rahman, S. Nakano and T. Nishizeki, *Rectangular drawings of plane graphs without designated corners*, Comp. Geom. Theo. and Appl., 21(3), pp. 121-138, 2002.
- [RNN03] M. S. Rahman, T. Nishizeki and M. Naznin, *Orthogonal drawings of plane graphs without bends*, Journal of Graph Alg. and Appl., <http://jgaa.info>, 7(4), pp. 335-362, 2003.
- [RNN98] M. S. Rahman, S. Nakano and T. Nishizeki, *Rectangular grid drawings of plane graphs*, Comp. Geom. Theo. Appl., 10(3), pp. 203-220, 1998.
- [RNN99] M. S. Rahman, S. Nakano and T. Nishizeki, *A linear algorithm for bend-optimal orthogonal drawings of triconnected cubic plane graphs*, Journal of Graph Alg. and Appl., <http://jgaa.info>, 3(4), pp. 31-62, 1999.
- [SAR01] S. Saha, A. K. M. Azad and M. S. Rahman, *A linear algorithm for automated VLSI floorplanning and routing*, Proc. of ICCIT 2001, pp. 165-170, 2001.
- [Sch90] W. Schnyder, *Embedding planar graphs on the grid*, Proc. First ACM-SIAM Symp. on Discrete Algorithms, San Francisco, pp. 138-148, 1990.

- [SF96] R. Sedgewick and P. Flajolet, *An Introduction to the Analysis of Algorithms*, Addison-Wesley, Reading, Mass., 1996.
- [SH92] W. K. Shih and W.-L. Hsu, *A simple test for planar graphs*, Proc. of Sixth Workshop on Discrete Mathematics and Theory of Computations, pp. 35-42, 1992.
- [SH99] W. K. Shih and W.-L. Hsu, *A new planarity test*, Theoretical Computer Science, 223, pp. 179-191, 1999.
- [She95] N. Sherwani, *Algorithms for VLSI Physical Design Automation*, 2nd edition, Kluwer Academic Publishers, Boston, 1995.
- [Shi96] W. Shi, *A fast algorithm for area minimization of slicing floorplans*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 15(12), pp. 1525-1532, 1996.
- [ST92] W. Schnyder and W. Trotter, *Convex drawings of planar graphs*, Abstracts of the AMS, 92T-05-135, 1992.
- [Ste51] K. S. Stein, *Convex maps*, Proc. Amer Math. Soc., 2, pp. 464-466, 1951.
- [Sto84] J. A. Storer, *On minimal node-cost planar embeddings*, Networks, 14, pp. 181-212, 1984.
- [Sug02] K. Sugiyama, *Graph Drawing and Applications for Software and Knowledge Engineers*, World Scientific, Singapore, 2002.
- [SY99] S. M. Sait and H. Youssef, *VLSI Physical Design Automation: Theory and Practice*, World Scientific, Singapore, 1999.
- [Tam87] R. Tamassia, *On embedding a graph in the grid with the minimum number of bends*, SIAM J. Computing, 16(3), pp. 421-444, 1987.
- [TDB88] R. Tamassia, G. Di Battista, and C. Batini, *Automatic graph drawing and readability of diagrams*, IEEE Trans. on Syst., Man, and Cybern., SMC-18, pp. 61-79, 1988.
- [Tho80] C. Thomassen, *Planarity and duality of finite and infinite graphs*, J. Combinat. Theory, Series B, 29, pp. 244-271, 1980.
- [Tho84] C. Thomassen, *Plane representations of graphs*, (Eds.) J. A. Bondy and U. S. R. Murty, Progress in Graph Theory, Academic Press Canada, pp. 43-69, 1984.
- [Tho86] C. Thomassen, *Interval representations of planar graphs*, J. of Combinat. Theory, Series B, 40, pp. 9-20, 1986.
- [Tho92] C. Thomassen, *Plane cubic graphs with prescribed face areas*, Combinatorics, Probability and Computing, 1, pp. 371-381, 1992.
- [TT95] R. Tamassia and I. G. Tollies (editors), *Graph Drawing (Proc. of*

- GD '94*), Lect. Notes in Computer Science, Springer, 894, 1995.
- [TTV91] R. Tamassia, I. G. Tollis and J. S. Vitter, *Lower bounds for planar orthogonal drawings of graphs*, Inf. Proc. Letters, 39, pp. 35-40, 1991.
- [Tur36] A. M. Turing, *On computable numbers, with an application to the Entscheidungs problem*, Proc. London. Math. Soc. 2-42, pp. 230-265; Correction, ibid, 2-43, pp. 544-546, 1936.
- [Tut54] W. T. Tutte, *A short proof of the factor theorem for finite graphs*, Canad. J. Math, 6, pp. 347-352, 1954.
- [Tut60] W. T. Tutte, *Convex representations of graphs*, Proc. London Math. Soc., 10, pp. 304-320, 1960.
- [Tut63] W. T. Tutte, *How to draw a graph*, Proc. of London Math. Soc., 13, pp. 743-768, 1963.
- [Wag36] K. Wagner, *Bemerkungen zum vierfarbenproblem*, Jahresber. Deutsch. Math-Verien., 46, pp. 26-32, 1936.
- [Wes96] D. B. West, *Introduction to Graph Theory*, Prentice-Hall, 1996.
- [Whi33] H. Whitney, *2-isomorphic graphs*, Amer. J. Math., 55, pp. 245-254, 1933.
- [Whi98] S. H. Whiteside (editor), *Graph Drawing (Proc. of GD '98)*, Lect. Notes in Computer Science, Springer, 1547, 1998.
- [Wil96] R. J. Wilson, *Introduction to Graph Theory*, 4th edition, Longman, 1996.
- [YS93] K. Yeap and M. Sarrafzadeh, *Floor-planning by graph dualization: 2-concave rectilinear modules*, SIAM J. Comput., 22(3), pp. 500-526, 1993.
- [YS95] G. K. H. Yeap and M. Sarrafzadeh, *Sliceable floorplanning by graph dualization*, SIAM J. Disc. Math., 8(2), pp. 258-280, 1995.

Index

- $A(G)$, 239
 C -component, 26
 $C_o(G)$, 26
 D_g , 228
 $E(G)$, 19
 $F_o(\Gamma)$, 162
 $G(C)$, 26
 $G - E'$, 21
 $G - V'$, 21
 G_E^P , 141, 235
 G_N^P , 235
 G_S^P , 235
 G_W^P , 141, 235
 G_u , 236
 G_d , 133
 K_n , 23
 $K_{s,r}$, 24
 $O()$, 34
 PQ -tree, 259
 full, 262
 P_E , 137
 P_N , 137
 P_S , 137
 P_W , 137
 P_u , 236
 $Q_c(C)$, 140
 $R(F)$, 199
 TC , 212
 $V(G)$, 19
 $W + H$, 9
 Δ , 19, 129, 197
 Γ , 162
 $\Gamma(G)$, 28
 $\Gamma_I(C)$, 162
 $\Gamma_O(C)$, 162
 Θ , 132
 ϕ , 201
 a_r , 199
 $bc(C)$, 213
 e_r , 199
 f , 29
 m , 19, 29
 n , 19, 29
 $n_c(C)$, 140
 s_r , 199
 st -numbering, 255
 \mathcal{C}_C , 211
 \mathcal{N} , 201
NP-complete, 35
3-connected component, 95
4-canonical decomposition, 117
4-canonical ordering, 74
 $Q_{cc}(C)$, 140
 $n_{cc}(C)$, 140
 R_r , 239
 $W \cdot H$, 9
 $W \times H$, 9
Adj, 254
adjacency list, 38, 254
adjacency matrix, 37
algorithm
 4-Convex-Draw, 119

- Canonical-Ordering, 48
- Convex-Drawing, 92
- Convex-Grid-Drawing, 113
- deterministic, 35
- Feasible-Draw, 217
- Four-Connected-Draw, 77
- linear-time, 35
- Minimum-Bend, 226
- nondeterministic, 35
- Octagonal-Draw, 238
- Orthogonal-Draw, 224
- polynomial, 35
- Realizer-Drawing, 71
- Rectangular-Draw, 152
- running time, 34
- Shift Algorithm, 50
- storage space, 34
- ancestor, 23
- ancestor cycle, 212
- angular resolution, 11
- apex, 91
- architectural floorplanning, 13
- area, 10
- array, 36
- aspect ratio, 11
- back edge, 39
- bad corner, 137
- bad cycle, 137
- barycentric representation, 58
- bend, 11, 197
- bend-angle, 199
- bend-count, 213
- bipartite graph, 24
 - complete, 24
- bond, 96
- boundary face, 142
- boundary path, 142
- box-orthogonal drawing, 7
- box-rectangular drawing, 8, 175
- breadth-first search, 39
- bridge, 199
- bush form, 259, 260
- canonical decomposition, 106
- canonical ordering, 46
 - chain, 21
 - adjacent, 21
 - support, 21
 - child-cycle, 212
 - chord, 46
 - chord-path, 105
 - minimal, 106
 - clockwise leg, 140
 - complete graph, 23
 - component, 22
 - concave corner, 243
 - connected, 22
 - connected component, 22
 - connectivity, 22
 - contour path, 213
 - convex corner, 243
 - convex drawing, 6, 89
 - convex grid drawing, 89
 - counterclockwise leg, 140
 - cover, 51
 - critical cycle, 140
 - crossing, 11
 - cut vertex, 22
 - cycle, 21
 - k -legged cycle, 26
 - attached, 140
 - chord, 46
 - clockwise, 140
 - counterclockwise, 140
 - independent, 26
 - leg, 26
 - maximal, 148
 - minimal, 26
 - cyclically k -edge connected, 250
 - cyclically 4-edge-connected, 163
- decision graph, 131, 133
- degree, 19
- depth-first search, 39, 255
- descendant, 23
- descendant cycle, 212
- dual edge, 30
- dual graph, 30
- dual-like, 13
- edge, 19

- multiple edges, 19
- edge contraction, 63
- embedding, 254
- equivalent embedding, 27
- Euler's formula, 29
- expansion sequence, 66
- extendible polygon, 91

- face path, 237
- facial cycle, 94
- feasible octagon, 244
- feasible orthogonal drawing, 217
- floorplanning, 12, 15
- flow network, 201
 - cost, 202
 - minimum cost, 202
- forest, 23

- genealogical tree, 212
- good cycle, 136
- good slicing floorplan, 234
- good slicing graph, 235, 237
- good slicing tree, 237
- graph, 19
 - connected, 22
 - cycle, 21
 - disconnected, 22
 - path, 21
 - planar, 24
 - simple, 19
 - subgraph, 20
 - tree, 22
 - walk, 21
- graph drawing
 - box-orthogonal drawing, 7
 - box-rectangular drawing, 8, 175
 - convex drawing, 6, 89
 - grid drawing, 8
 - octagonal drawing, 6, 233
 - orthogonal drawing, 6
 - planar drawing, 4
 - polyline drawing, 5
 - properties, 10
 - rectangular drawing, 8, 129
 - straight line drawing, 5, 45
 - visibility drawing, 9
- green path, 213
- grid drawing, 8
 - area, 9
 - convex, 89
 - half perimeter, 9
 - height, 9
 - size, 9
 - width, 9
- head vertex, 140

- in-degree, 119
- inner angle, 131
- inner edge, 26
- inner vertex, 26
- internally 3-connected, 105
- internally triangulated, 46

- Kuratowski's theorem, 25, 33

- larger neighbor, 80
- leg, 26
- leg-vertex, 26
- list, 36
- loop, 19

- matching, 133
 - maximum, 133
 - perfect, 134
- maximal planar graph, 29
- maximum matching, 133
- MCM, 13
- merging, 95
- minimum cost flow, 202
- modules, 12
- molecular structures, 15
- multichip module, 13

- neighbor, 19
 - larger, 80
 - smaller, 80
- network flow, 198
- NS-path, 140

- octagon, 238
- octagonal drawing, 6, 233

- orthogonal drawing, 6, 197
 - bend, 197
 - bend-optimal drawing, 202
 - box-orthogonal, 7
- orthogonal grid drawing, 227
- orthogonal representation, 198, 199
- out-degree, 119
- outer angle, 131
- outer boundary, 26
- outer cycle, 46
- outer edge, 26, 46
- outer face, 26
- outer vertex, 26, 46

- partition-pair, 147
- partitioning path, 141
- patching operation, 221
- path, 21
- perfect matching, 134
- peripheral face, 163
- pertinent, 262
- planar graph, 24
- planar representation, 198
- planarity testing, 254
 - algorithm, 263
- plane graph, 26
- port, 228
- PQ-tree, 261
- proper inner cycle, 250

- queue, 36
 - head, 36
 - tail, 36

- random-access machine, 34
- realizer, 66, 67
- rectangular drawing, 8, 129
- rectangular dual, 173
- rectangular grid drawing, 156
- rectangular refinement, 227
- red path, 213
- refined decomposition, 125
- regular labeling, 132
- ring, 96
- RNA, 15
- routing, 15

- running time, 34

- Schnyder labeling, 62, 67
- separating triangle, 64
- separation pair, 22, 95
 - critical, 96
 - forbidden, 96
 - prime, 96
- separator, 22
- shape of faces, 11
- slicing floorplan, 234
 - good, 234
- slicing graph, 235
 - good, 237
- slicing path, 236
- slicing tree, 236
 - good, 237
- smaller neighbor, 80
- spanning subgraph, 20
- spanning tree, 23
- split component, 95
- split graph, 95
- splitting, 95
- stack, 36
- storage space, 34
- straight line drawing, 45
 - realizer method, 45, 58
 - shift method, 45, 46
- strict convex polygon, 94
- subdivision, 24
- symmetry, 11

- tail vertex, 140
- template matching, 262
- tree, 22
 - ancestor, 23
 - child, 23
 - descendant, 23
 - internal node, 23
 - leaf, 23
 - parent, 23
 - rooted, 22
 - spanning, 23
- tree edge, 39
- triangulated plane graph, 29
- Turing machine, 33

- unique embedding, 27
- upward digraph, 260
- upward embedding, 260

- vertex, 19
 - cut, 22
 - degree, 19
- vertex cut, 22
- vertex-angle, 199
- virtual edge, 95, 260
- virtual vertex, 260
- visibility drawing, 9
 - 2-visibility, 10

- walk, 21
- weak barycentric representation, 60
- westmost NS-path, 148

- x-monotone, 78