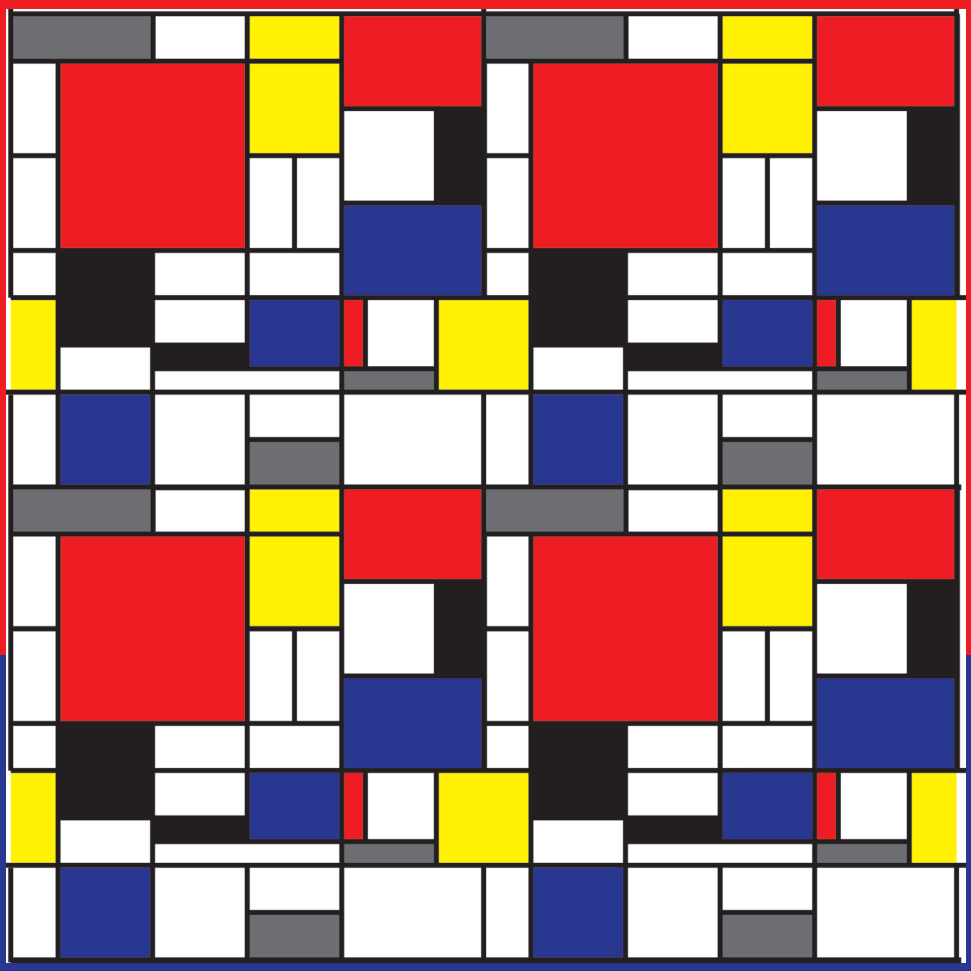
DISCRETE MATHEMATICS AND ITS APPLICATIONS
Series Editor KENNETH H. ROSEN

Handbook of Graph Drawing and Visualization



Edited by
Roberto Tamassia

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

Handbook of Graph Drawing and Visualization

DISCRETE MATHEMATICS AND ITS APPLICATIONS

Series Editor
Kenneth H. Rosen, Ph.D.

R. B. J. T. Allenby and Alan Slomson, How to Count: An Introduction to Combinatorics, Third Edition

Craig P. Bauer, Secret History: The Story of Cryptology

Juergen Bierbrauer, Introduction to Coding Theory

Katalin Bimbó, Combinatory Logic: Pure, Applied and Typed

Donald Bindner and Martin Erickson, A Student's Guide to the Study, Practice, and Tools of Modern Mathematics

Francine Blanchet-Sadri, Algorithmic Combinatorics on Partial Words

Miklós Bóna, Combinatorics of Permutations, Second Edition

Jason I. Brown, Discrete Structures and Their Interactions

Richard A. Brualdi and Dragoš Cvetković, A Combinatorial Approach to Matrix Theory and Its Applications

Kun-Mao Chao and Bang Ye Wu, Spanning Trees and Optimization Problems

Charalambos A. Charalambides, Enumerative Combinatorics

Gary Chartrand and Ping Zhang, Chromatic Graph Theory

Henri Cohen, Gerhard Frey, et al., Handbook of Elliptic and Hyperelliptic Curve Cryptography

Charles J. Colbourn and Jeffrey H. Dinitz, Handbook of Combinatorial Designs, Second Edition

Abhijit Das, Computational Number Theory

Martin Erickson, Pearls of Discrete Mathematics

Martin Erickson and Anthony Vazzana, Introduction to Number Theory

Steven Furino, Ying Miao, and Jianxing Yin, Frames and Resolvable Designs: Uses, Constructions, and Existence

Mark S. Gockenbach, Finite-Dimensional Linear Algebra

Randy Goldberg and Lance Riek, A Practical Handbook of Speech Coders

Titles (continued)

Jacob E. Goodman and Joseph O'Rourke, Handbook of Discrete and Computational Geometry, Second Edition

Jonathan L. Gross, Combinatorial Methods with Computer Applications

Jonathan L. Gross and Jay Yellen, Graph Theory and Its Applications, Second Edition

Jonathan L. Gross and Jay Yellen, Handbook of Graph Theory

David S. Gunderson, Handbook of Mathematical Induction: Theory and Applications

Richard Hammack, Wilfried Imrich, and Sandi Klavžar, Handbook of Product Graphs, Second Edition

Darrel R. Hankerson, Greg A. Harris, and Peter D. Johnson, Introduction to Information Theory and Data Compression, Second Edition

Darel W. Hardy, Fred Richman, and Carol L. Walker, Applied Algebra: Codes, Ciphers, and Discrete Algorithms, Second Edition

Daryl D. Harms, Miroslav Kraetzl, Charles J. Colbourn, and John S. Devitt, Network Reliability: Experiments with a Symbolic Algebra Environment

Silvia Heubach and Toufik Mansour, Combinatorics of Compositions and Words

Leslie Hogben, Handbook of Linear Algebra

Derek F. Holt with Bettina Eick and Eamonn A. O'Brien, Handbook of Computational Group Theory

David M. Jackson and Terry I. Visentin, An Atlas of Smaller Maps in Orientable and Nonorientable Surfaces

Richard E. Klima, Neil P. Sigmon, and Ernest L. Stitzinger, Applications of Abstract Algebra with Maple™ and MATLAB®, Second Edition

Richard E. Klima and Neil P. Sigmon, Cryptology: Classical and Modern with Maplets

Patrick Knupp and Kambiz Salari, Verification of Computer Codes in Computational Science and Engineering

William Kocay and Donald L. Kreher, Graphs, Algorithms, and Optimization

Donald L. Kreher and Douglas R. Stinson, Combinatorial Algorithms: Generation Enumeration and Search

Hang T. Lau, A Java Library of Graph Algorithms and Optimization

C. C. Lindner and C. A. Rodger, Design Theory, Second Edition

San Ling, Huaxiong Wang, and Chaoping Xing, Algebraic Curves in Cryptography

Nicholas A. Loehr, Bijective Combinatorics

Toufik Mansour, Combinatorics of Set Partitions

Alasdair McAndrew, Introduction to Cryptography with Open-Source Software

Elliott Mendelson, Introduction to Mathematical Logic, Fifth Edition

Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, Handbook of Applied Cryptography

Titles (continued)

Stig F. Mjølsnes, A Multidisciplinary Introduction to Information Security

Jason J. Molitierno, Applications of Combinatorial Matrix Theory to Laplacian Matrices of Graphs

Richard A. Mollin, Advanced Number Theory with Applications

Richard A. Mollin, Algebraic Number Theory, Second Edition

Richard A. Mollin, Codes: The Guide to Secrecy from Ancient to Modern Times

Richard A. Mollin, Fundamental Number Theory with Applications, Second Edition

Richard A. Mollin, An Introduction to Cryptography, Second Edition

Richard A. Mollin, Quadratics

Richard A. Mollin, RSA and Public-Key Cryptography

Carlos J. Moreno and Samuel S. Wagstaff, Jr., Sums of Squares of Integers

Gary L. Mullen and Daniel Panario, Handbook of Finite Fields

Goutam Paul and Subhamoy Maitra, RC4 Stream Cipher and Its Variants

Dingyi Pei, Authentication Codes and Combinatorial Designs

Kenneth H. Rosen, Handbook of Discrete and Combinatorial Mathematics

Douglas R. Shier and K.T. Wallenius, Applied Mathematical Modeling: A Multidisciplinary Approach

Alexander Stanoyevitch, Introduction to Cryptography with Mathematical Foundations and Computer Implementations

Jörn Steuding, Diophantine Analysis

Douglas R. Stinson, Cryptography: Theory and Practice, Third Edition

Roberto Tamassia, Handbook of Graph Drawing and Visualization

Roberto Togneri and Christopher J. deSilva, Fundamentals of Information Theory and Coding Design

W. D. Wallis, Introduction to Combinatorial Designs, Second Edition

W. D. Wallis and J. C. George, Introduction to Combinatorics

Jiacun Wang, Handbook of Finite State Based Models and Applications

Lawrence C. Washington, Elliptic Curves: Number Theory and Cryptography, Second Edition

DISCRETE MATHEMATICS AND ITS APPLICATIONS
Series Editor KENNETH H. ROSEN

Handbook of Graph Drawing and Visualization

Edited by
Roberto Tamassia
Brown University
Providence, Rhode Island, USA

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
A CHAPMAN & HALL BOOK

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Version Date: 20130719

International Standard Book Number-13: 978-1-4200-1026-8 (eBook - PDF)

Contents

Preface

Objective

This handbook aims at providing a broad survey of the field of graph drawing. It covers topological and geometric foundations, algorithms, software systems, and visualization applications in business, education, science, and engineering.

The intended readership of this handbook includes:

- Practitioners and researchers in traditional and emerging disciplines of the physical, life, and social sciences interested in understanding and using graph drawing methods and graph visualization systems in their field.
- Information technology practitioners and software developers aiming to incorporate graph drawing solutions into their products.
- Researchers and students in graph drawing and information visualization seeking an up-to-date survey of the field.
- Researchers and students in related fields of mathematics and computer science (including graph theory, computational geometry, information visualization, software engineering, user interfaces, social networks, and data management) interested in using graph-drawing techniques in support of their research.

Organization

The chapters of this handbook are organized into four parts, as follows.

Topological and Geometric Foundations of Graph Drawing The first part (Chapters 1–4) deals with fundamental topological and geometric concepts and techniques used in graph drawing: planarity testing and embedding, crossings and planarization, symmetric drawings, and proximity drawings.

Graph Drawing Algorithms The second part (Chapters 5–14) presents an extensive collection of algorithms for constructing drawings of graphs. Some methods are designed to draw special classes of graphs (e.g., trees, planar graphs, or directed acyclic graphs) while other methods work for general graphs. Topics covered in this part include tree drawing algorithms, planar straight-line drawing algorithms, planar orthogonal and polyline drawing algorithms, spine and radial drawings, circular drawing algorithms, rectangular drawing algorithms, simultaneous embeddings, force-directed methods, hierarchical drawing algorithms, three-dimensional drawing algoritms, and labeling algorithms.

Graph Drawing Systems The third part begins by introducing the GraphML language for representing graphs and their drawings (Chapter 16). Next, it overviews three software systems for constructing drawings of graphs: OGDF, GDToolkit, and PIGALE (Chapters 17–19).

Applications of Graph Drawing The fourth part (Chapters 20–26) gives examples of the use of graph drawing methods for the visualization of networks in various important application domains: biological networks, computer security, data analytics, education, computer networks, and social networks.

Each chapter is intended to be self-contained and has its own bibliography.

Acknowledgments

I would like to thank all the authors of the chapters in this handbook and all the reviewers who have provided expert feedback on the initial drafts and revised versions of the chapters.

This handbook is a collective effort of the graph drawing research community, which has developed around the annual Symposium on Graph Drawing. I am grateful to the people who have founded with me this conference and have continued providing leadership for it: Franz Brandenburg, Giuseppe Di Battista, Peter Eades, Hubert de Fraysseix, Takao Nishizeki, Pierre Rosenstiehl, and Ioannis Tollis.

A huge thanks goes to Sunil Nair for proposing this handbook project and supporting its development. I truly appreciate his constant encouragement and patience. Help received from the entire CRC Press production team, and especially from Andre Barnett, Kari Budyk, Rachel Holt, Jim McGovern, and Shashi Kumar, is gratefully acknowledged. I would like to thank also my executive assistant Angel Murakami for her expert proofreading of the chapters.

I am indebted to Carlo Batini for introducing me to the problem of drawing graphs and inspiring me to pursue an academic path. A special thanks also goes to Franco Preparata, whose guidance and support, first as PhD advisor and then as colleague, have shaped by career.

Finally, warm thanks go to Isabel Cruz, Giuseppe Di Battista, Michael Goodrich, and Ioannis Tollis for their encouragement and support throughout this project.

Roberto Tamassia

About the Editor

Roberto Tamassia is the Plastech Professor of Computer Science and the Chair of the Department of Computer Science at Brown University. He is also the Director of Brown's Center for Geometric Computing. His research interests include analysis, design, and implementation of algorithms, applied cryptography, cloud computing, computational geometry data security, and graph drawing. He has published six textbooks and more than 250 research articles and books in the above areas and has given more than 70 invited lectures worldwide. He is a Fellow of the American Association for the Advancement of Science (AAAS), the Association for Computing Machinery (ACM), and the Institute of Electrical and Electronics Engineers (IEEE). He is the recipient of a Technical Achievement Award from the IEEE Computer Society for pioneering the field of graph drawing. He is listed among the 360 most cited computer science authors by Thomson Scientific, Institute for Scientific Information (ISI). He serves regularly on program committees of international conferences. His research has been funded by ARO, DARPA, NATO, NSF, and several industrial sponsors. He co-founded the Journal of Graph Algorithms and Applications (JGAA) and the Symposium on Graph Drawing. He serves as Co-Editor-in-Chief of JGAA. He received the PhD degree in electrical and computer engineering from the University of Illinois at Urbana-Champaign and the "Laurea" in Electrical Engineering from the "Sapienza" University of Rome.

1

Planarity Testing and Embedding

Maurizio Patrignani
Roma Tre University

1.1 Introduction

Testing the planarity of a graph and possibly drawing it without intersections is one of the most fascinating and intriguing algorithmic problems of the graph drawing and graph theory areas. Although the problem *per se* can be easily stated, and a complete characterization of planar graphs has been known since 1930, the first linear-time solution to this problem was found only in the 1970s.

Planar graphs play an important role both in the graph theory and in the graph drawing areas. In fact, planar graphs have several interesting properties: for example, they are sparse and 4-colorable, they allow a number of operations to be performed more efficiently than for general graphs, and their inner structure can be described more succinctly and elegantly (see Section 1.2.2). From the information visualization perspective, instead, as edge crossings turn out to be the main reason for reducing readability, planar drawings of graphs are considered clear and comprehensible.

In this chapter, we review a number of different algorithms from the literature for efficiently testing planarity and computing planar embeddings. Our main thesis is that all known linear-time planarity algorithms fall into two categories: cycle based algorithms and vertex addition algorithms. The first family of algorithms is based on the simple obser-

vation that in a planar drawing of a graph any cycle necessarily partitions the graph into the inside and outside portion, and this partition can be suitably used to split the embedding problem. Vertex addition algorithms are based on the incremental construction of the final planar drawing starting from planar drawings of smaller graphs. The fact that some algorithms were based on the same paradigm was already envisaged by several researchers [Tho99, HT08]. However, the evidence that all known algorithms boil down to two simple approaches is a relatively new concept.

The chapter is organized as follows: Section 1.2 introduces basic definitions, properties, and characterizations for planar graphs; Section 1.3 formally defines the planarity testing and embedding problems; Section 1.4 follows a historic perspective to introduce the main algorithms and a conventional classification for them. Some algorithmic techniques are common to more than one algorithm and sometimes to all of them. These are collected in Section 1.5. Finally, Sections 1.6 and 1.7 are devoted to the two approaches to the planarity testing problem, namely, the "cycle based" and the "vertex addition" approaches, respectively.

Algorithms for constructing planar drawings of graphs are discussed in Chapters 6 (straight-line drawings), 7 (orthogonal and polyline drawings), and 10 (rectangular drawings). Methods for reducing crossings in nonplanar drawings of graphs are discussed in Chapter 2.

1.2 Properties and Characterizations of Planar Graphs

1.2.1 Basic Definitions

A *graph* $G(V, E)$ is an ordered pair consisting of a finite set V of *vertices* and a finite set E of *edges*, that is, pairs (u, v) of vertices. If each edge is an unordered (ordered) pair of vertices, then the graph is *undirected* (*directed*). An edge (u, v) is a *self-loop* if $u = v$. A graph $G(V, E)$ is *simple* if E is not a multiple set and it does not contain self-loops. For the purposes of this chapter, we can restrict us to simple graphs.

The sets of edges and vertices of G can be also denoted $E(G)$ and $V(G)$, respectively. If edge $(u, v) \in E$, vertices u and v are said to be *adjacent* and (u, v) is said to be *incident* to u and v. Two edges are *adjacent* if they have a vertex in common.

A (rooted) *tree* T is a connected acyclic graph with one distinguished vertex, called the *root* r. A *spanning tree* of a graph G is a tree T such that $V(T) = V(G)$ and $E(T) \subseteq E(G)$.

Given two graphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$, their *union* $G_1 \cup G_2$ is the graph $G(V_1 \cup V_2, E_1 \cup E_2)$. Analogously, their *intersection* $G_1 \cap G_2$ is the graph $G(V_1 \cap V_2, E_1 \cap E_2)$. A graph G_2 is a *subgraph* of G_1 if $G_1 \cup G_2 = G_1$.

Given a graph $G(V, E)$ and a subset V' of V, the subgraph *induced by* V' is the graph $G'(V', E')$, where E' is the set of edges of E that have both endvertices in V'. Given a graph $G(V, E)$ and a subset E' of E, the subgraph *induced by* E' is the graph $G'(V', E')$, where V' is the set of vertices incident to E'. A *subdivision* of an edge (u, v) consists of the insertion of a new node w and the replacement of (u, v) with edges (u, w) and (w, v). A graph G_2 is a *subdivision* of G_1 if it can be obtained from G_1 through a sequence of edge subdivisions.

A *drawing* Γ of a graph G maps each vertex v to a distinct point $\Gamma(v)$ of the plane and each edge (u, v) to a simple open Jordan curve $\Gamma(u, v)$ with endpoints $\Gamma(u)$ and $\Gamma(v)$. A drawing is *planar* if no two distinct edges intersect except, possibly, at common endpoints. A graph is *planar* if it admits a planar drawing. A planar drawing partitions the plane into connected regions called *faces*. The unbounded face is usually called *external face* or *outer face*. If all the vertices are incident to the outer face, the planar drawing is called *outerplanar* and the graph admitting it is an *outerplanar graph*. Given a planar drawing,

the (clockwise) circular order of the edges incident to each vertex is fixed. Two planar drawings are *equivalent* if they determine the same circular orderings of the edges incident to each vertex (sometimes called *rotation scheme*). A *(planar) embedding* is an equivalence class of planar drawings and is described by the clockwise circular order of the edges incident to each vertex. A graph together with one of its planar embedding is sometimes referred to as a *plane* graph.

A *path* is a sequence of distinct vertices $v_1, v_2, \ldots, v_k$, with $k \geq 2$, together with the edges $(v_1, v_2), \ldots, (v_{k-1}, v_k)$. The *length* of the path is the number of its edges.

A *cycle* is a sequence of distinct vertices $v_1, v_2, \ldots, v_k$, with $k \geq 2$, together with the edges $(v_1, v_2), \ldots, (v_{k-1}, v_k), (v_k, v_1)$. The *length* of a cycle is the number of its vertices or the number of its edges.

An undirected graph G is *connected* if, for each pair of nodes u and v, G contains a path from u to v. A graph G with at least $k+1$ vertices is *k-connected* if removing any $k-1$ vertices leaves G connected. Equivalently, by Menger's theorem, a graph is k-connected if there are k independent paths between each pair of vertices [Men27]. 3-connected, 2-connected, and 1-connected graphs are also called *triconnected*, *biconnected*, and *simply connected* graphs, respectively. It is usual in the planarity literature to relax the definition of biconnected graph so to include *bridges*, i.e., graphs composed by a single edge between two vertices. A *separating k-set* is a set of k vertices whose removal disconnects the graph. Separating 1- and 2-sets are called *cutvertices* and *separation pairs*, respectively. Hence, a connected graph is biconnected if it has no cutvertices and it is triconnected if it has no separation pairs.

If a graph G is not connected, its maximal connected subgraphs are called the *connected components* of G. If G is connected, its maximal biconnected subgraphs (including bridges) are called the *biconnected components*, or *blocks* of G. Note that a cutvertex belongs to several blocks and that a biconnected graph has only one block. The graph whose vertices are the blocks and the cutvertices of G and whose edges link cutvertices to the blocks they belong to is a tree and is called the *block-cutvertex tree* (or *BC-tree*) of G (see Figure 1.1 for an example).

Given a biconnected graph G, its *triconnected components* are obtained by a complex splitting and merging process. The first linear-time algorithm to compute them was introduced in [HT73], while an implementation of it is described in [GM01]. The computation has two phases: first, G is recursively split into its *split components*; second, some split components are merged together to obtain triconnected components. The *split* operation is performed with respect to a pair of vertices $\{v_1, v_2\}$ of the biconnected (multi)graph G. Suppose the edges of G are divided into the equivalence classes $E_1, E_2, \ldots, E_k$ such that two edges are in the same class if both lie in a common path not containing a vertex in $\{v_1, v_2\}$ except, possibly, as an end point. If there are at least two such classes, then $\{v_1, v_2\}$ is a *split pair*. Let G_1 be the graph induced by E_1 and G_2 be the graph induced by E/E_1. A *split* operation consists of replacing G with G'_1 and G'_2, where G'_1 and G'_2 are obtained from G_1 and G_2 by adding the same *virtual edge* (v_1, v_2). The two copies of the virtual edge added to G_1 and G_2 are called *twin virtual edges*. Figure 1.2(b) shows the result of a split operation performed on the graph of Figure 1.2(a) with respect to split pair $\{2, 4\}$. The *split components* of a graph G are obtained by recursively splitting G until no split pair can be found in the obtained graphs. Figure 1.2(c) shows the split components of the graph of Figure 1.2(a). Split components are not unique and, hence, are not suitable for describing the structure of G.

Two split components sharing the same twin virtual edges (v_1, v_2) can be merged by identifying the two copies of v_1 and v_2 and by removing the twin virtual edges. Split components consisting of cycles are called *series split components*, while split components

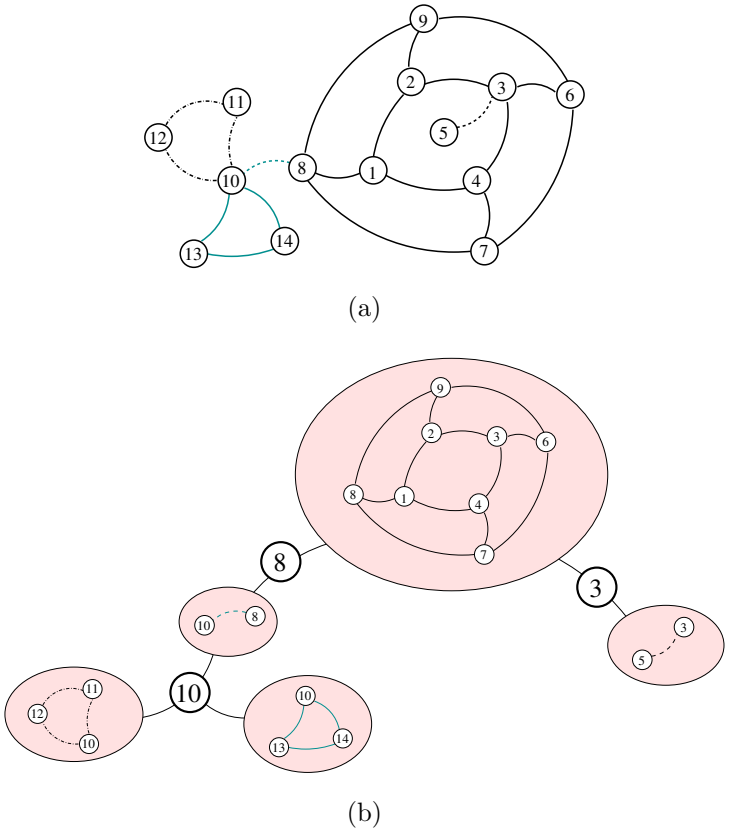
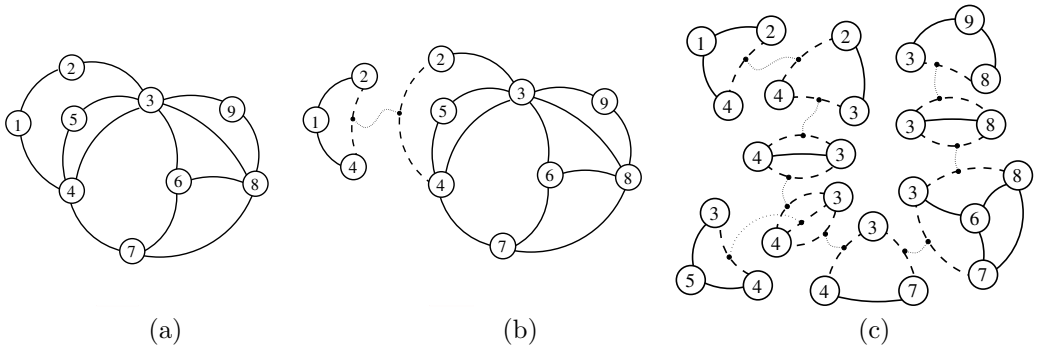


Figure 1.1 A connected graph (a) and its BC-tree (b). Different line styles are used for edges of different blocks.

that have only two vertices are called *parallel split components*. By recursively merging together series split components that share twin virtual edges we obtain *series triconnected components*, while by recursively merging together parallel split components that share twin virtual edges we obtain *parallel triconnected components*. Split components that are not affected by the merging operations described above are called *rigid triconnected components*. Figure 1.3(a) shows the triconnected components of the graph of Figure 1.2(a).

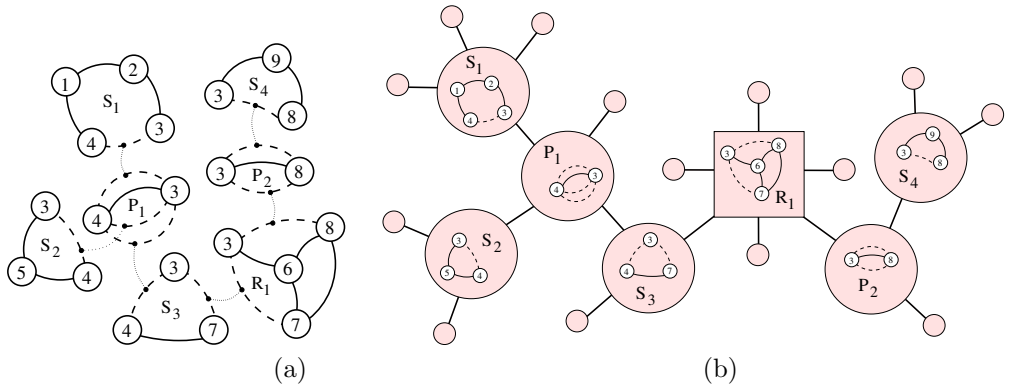
Triconnected components are unique and are used to describe the inner structure of a graph. In fact, a graph G can be succinctly described by its SPQR-tree $\mathcal{T}$, which provides a high-level view of the unique decomposition of the graph into its triconnected components [DT96a, DT96b, GM01]. Namely, each triconnected component corresponds to a node of $\mathcal{T}$. The triconnected component corresponding to a node μ of $\mathcal{T}$ is called the *skeleton of* μ. As there are parallel, series, and rigid triconnected components, their corresponding tree nodes are called P-, S-, and R-nodes, respectively. Triconnected components sharing a virtual edge are adjacent in $\mathcal{T}$. Usually, a fourth type of node, called Q-node, is used to represent an edge (u, v) of G. Q-nodes are the leaves of $\mathcal{T}$ and they don't have skeletons. Tree $\mathcal{T}$ is unrooted, but for some applications, it could be thought as rooted at an arbitrary Q-node. See Figure 1.3 for an example of SPQR-tree.

The connectivity properties of a graph have a strict relationship with its embedding properties. Triconnected planar graphs (and triconnected planar components) have a single



(a) (b) (c)

Figure 1.2 (a) A biconnected graph. (b) A split operation performed with respect to split pair $\{2, 4\}$. (c) The split components of the graph. Virtual edges are drawn dashed. Twin virtual edges are joined with dotted lines.



(a) (b)

Figure 1.3 (a) The triconnected components of the same graph of Figure 1.2. (b) The corresponding SPQR-tree. Q-nodes are represented by empty circles.

embedding up to a flip (that is, up to a reversal of all their incidence lists) [Whi32]. The same holds for biconnected outerplanar graphs and their unique outerplanar embedding (adding a star on the outer face yields a triconnected plane graph).

A non-connected graph is planar if and only if all its connected components are planar. Thus, in the following, without loss of generality, we only consider the planarity of connected graphs. Also, a planar embedding of a graph implies a planar embedding for each one of its blocks, while, starting from a planar embedding of the blocks, a planar embedding for the whole graph can be found [Whi32]. Thus, since the blocks can be identified in linear time [Tar72], a common strategy, both to test planarity and to compute a planar embedding, is to divide the graph into its blocks and to tackle each block separately.

Finally, a graph is planar if and only if its triconnected components are planar [Mac37b]. More precisely, as parallel and series triconnected components are always planar, a graph is planar if and only if all its rigid triconnected components are planar. However, since dividing a graph into its triconnected components is a linear but rather laborious process [HT73, GM01], usually planarity algorithms do not assume that the input graph is triconnected.

Also, from a planar embedding of the triconnected components of a graph, a planar embedding of the whole graph can be obtained. This property can be exploited to explore

the planar embeddings of a given graph when searching for some embedding with a specific property (see, for example, [MW99, MW00, BDBD00, GMW01, ADF+10]).

Given a plane (multi)graph G, its *plane dual* (or simply its *dual*) is the multigraph G^* such that G^* has one vertex for each face of G and two vertices of G^* are linked by one edge e^* if the corresponding faces in G share one edge e. Observe that the planar embedding of G induces a planar embedding of its dual and that the dual of the dual of G is G itself. Also, different embeddings of a planar graph G correspond to different dual graphs. Finally, a cycle in G corresponds to a minimal cut in G^* (anytime this property holds G and G^* are called *abstract dual*).

A graph $G(V, E)$ is k *colorable* if its vertices can be partitioned into k sets $V_1, V_2, \ldots, V_k$ in such a way that no edge is incident to two vertices of the same set. A graph $G(V, E)$ is *complete* if each vertex in V is adjacent to each other.

A graph $G(V, E)$ is *bipartite* if it is 2-colorable. A bipartite graph $G(V_1, V_2, E)$ is *complete* if each vertex in V_1 is adjacent to all vertices in V_2.

1.2.2 Properties

Planar graphs have a variety of properties whose exploitation allows us to efficiently perform a number of operations on them.

Perhaps the most renown property is the one stated by Euler's Theorem, which shows that planar graphs are sparse. Namely, given a plane graph with n vertices, m edges and f faces, we have $n - m + f = 2$. A simple corollary is that for a maximal planar graph with at least three vertices, where each face is a triangle ($2m = 3f$), we have $m = 3n - 6$, and, therefore, for any planar graph we have $m \leq 3n - 6$. This number reduces to $m = 2n - 3$ for maximal outerplanar graphs with at least three vertices (and $m \leq 2n - 3$ for general outerplanar graphs). Also, if $n \geq 3$ and the graph has no cycle of length 3, then $m \leq 2n-4$. Finally, if the graph is a tree, then $m = n - 1$.

These considerations allow us to replace m with n in any asymptotic calculation involving planar graphs, while for general graphs only $m \in O(n^2)$ can be assumed. From a more practical perspective, they allow us to decide the non-planarity of denser graphs without reading all the edges (which would yield a quadratic algorithm).

The Four Color Theorem [AH77, AHK77, RSST97] asserts that any planar graph is 4-colorable and settles a conjecture that was for more than a century the most famous unsolved problem in graph theory and perhaps in all of mathematics [Har69]. To stress how important this property is, it suffices to observe that, apart from being considered an important property of planar graphs, it has also been mentioned as the most notable property of the number 4.

While 3-colorability is NP-hard even on maximum degree four planar graphs [GJS76], every triangle-free planar graph is 3-colorable [Grö59] and such a 3-coloring can be found in linear-time [DKT09].

Determining whether the graph contains a k-clique, i.e., a set of k pairwise adjacent vertices, is polynomial for planar graphs, as no clique can have more than four vertices. This problem is polynomial even in the weighted case, where each vertex is associated with a weight and the sum of the weights of the pairwise adjacent vertices is requested to be at least k. Observe that both these problems are NP-complete on non-planar graphs.

Graph isomorphism is linear for planar graphs [HW74], while it is of unknown complexity for general graphs [GJ79].

The planar separator theorem [LT79] states that every planar graph $G = (V, E)$ admits a partition of its n vertices into three sets, A, B, and C, such that the size of C is $O(\sqrt{n})$, the size of A and B is at most $\frac{2}{3}n$, and there is no edge with one endpoint in A and the other

endpoint in B. Such a partition can be found in linear time and is the starting point of a hierarchical decomposition of the graph that may lead to efficient approaches to compute properties of the graph.

1.2.3 Characterizations

The first complete characterization of planar graphs is due to Kuratowski [Kur30] and states that a graph is planar if and only if it contains no subgraph that is a subdivision of K_5 or $K_{3,3}$, where K_5 is the complete graph of order 5 and $K_{3,3}$ is the complete bipartite graph with 3 vertices in each of the sets of the partition. An equivalent later result, recasted in terms of graph minors, is Wagner's theorem that states that a graph G is planar if and only if it has no K_5 or $K_{3,3}$ as minor, that is, K_5 or $K_{3,3}$ cannot be obtained from G by contracting some edges, deleting some edges, and deleting some isolated vertices [Wag37a, HT65]. Observe that the two characterizations are different since a graph may admit K_5 as minor without having a subgraph that is a subdivision of K_5 (consider, for example, a graph of maximum degree 3).

Similarly, it can be proved that a graph is outerplanar if and only if it contains no subgraph that is a subdivision of K_4 or $K_{2,3}$. Trivially, a graph is a tree if it does not contain a subdivision (or a minor) of K_3.

If the graph is triconnected, a less renown but much simpler characterization can be formulated. Namely, a triconnected graph distinct from K_5 is planar if and only if it contains no subgraph that is a subdivision of $K_{3,3}$ [Wag37b, Hal43, Kel93, Lie01].

Given a graph G with no isolated vertices, the associated height-two *vertex-edge poset* $<_G$ has $V \cup E$ as elements, and $v <_G e$ if and only if $v \in V$, $e \in E$, and v is an endpoint of edge e. The smallest number of total orders the intersection of which yields the poset is called the *dimension* of the poset. Graph G is planar if and only if its corresponding vertex-edge poset has dimension at most three [Sch89]. Unfortunately, checking if a poset has dimension at most t is proved to be NP-complete for $t \geq 3$ and for $t \geq 4$ if the poset has height two [Yan82].

Edges traversing a bipartition of the vertices of G are called a *cocycle*. Observe that while a cycle is a collection of edges that covers each vertex an even (possibly zero) number of times, a cocycle is a collection of edges that intersects each cycle in an even number of edges. A *bicycle* is a collection of edges that is both a cycle and a cocycle. Planarity can be characterized in terms of the properties of the vector spaces of cycles [Mac37a], cocycles [APBL95, LS10], and bicycles [APBL95].

A further planarity characterization is expressed via Colin de Verdiére's graph invariant $\mu(G)$, which in turn is based on the maximum multiplicity of the second eigenvalue of certain Schrödinger operators defined by the graph [Col90, Col91], and states that a graph G is planar if and only if $\mu(G) \leq 3$.

Alternative characterizations can be found in the literature based on the existence of an abstract dual graph [Whi32], on the edge poset dimension [dO96], on the relationship among theta-graph minors [AŠ98], on the orientability of circuits [LH77, Che81], on the arrangements of pseudo-lines [TT97], or on DFS traversals of the graph [dR82, dR85, SH93, SH99, BM99, BM04].

1.3 Planarity Problems

The main planarity problem is the decision problem of recognizing planar graphs, that is, of deciding the planarity of the input graph. Both with the purpose of exhibiting a planarity

certificate and of producing a planar embedding for information-visualization applications, planarity testing algorithms are usually coupled with planar embedding procedures, that sometimes, depending on the algorithmic approach, required a considerable research effort to be devised.

On the opposite, if the graph is not planar, the search for a non-planarity certificate is called Kuratowski subgraph isolation [CMS08], and the research concentrated on planarization algorithms that allow us to produce a planar graph where some degree-four vertices have been added to replace crossings [Lie01]. Since crossing number minimization is NP-complete [GJ79] planarization algorithms use heuristics to introduce a reduced number of dummy vertices.

Dynamic algorithms have also been devised for efficiently determining planarity and computing a planar embedding of graphs where edges and vertices are added or deleted one at a time [DT89, DT96b, GIS99, DBTV01].

Efficient algorithms for planarity testing in parallel have been investigated in [KR88, RR89, RR94].

1.3.1 Constrained Planarity

The problem of determining the planarity of a graph and of computing a possible embedding of it can be combined with additional constraints on the desired drawing that result in restrictions on the set of admissible planar embeddings [Tam98, GKM08]. Typical constraints ask for some vertices to be on the same face (usually the outer face), some vertex to have a specified circular ordering of its incident edges, some path to be drawn along a straight line, etc. In the easier cases, such constraints can be enforced by replacing sets of nodes and edges of the input graph with suitable gadgets, by launching an ordinary planarity algorithm, and by transferring the results back on the original graph. More complex cases require to efficiently explore the possible embeddings of the graph by considering their inner structures described by their BC-trees and SPQR-trees. In [GKM08], embedding constraints that restrict the admissible order of incident edges around a vertex are considered.

A very restrictive constraint is when the input graph G is partially embedded, i.e., when a subgraph H of G is provided with an embedding $\mathcal{H}$. In this case, the problem of determining a planar embedding of the whole graph that extends the embedding $\mathcal{H}$, if one exists, is linear [ADF$^+$10]. Also, if the answer is negative, an obstruction taken from a collection of minimal non-planar instances can be produced in polynomial time [JKR11].

A constrained planarization is implied anytime an embedding that minimizes some quality measure is desired. As pointed out in [BM90, PT00, Piz05], the quality of a planar embedding can be measured in terms of the maximum distance of its vertices from the external face. Such a distance can be given in terms of different incidence relationships between vertices and faces. For example, if two faces are considered adjacent when they share a vertex, then the maximum distance to the external face is called *radius* [RS84]. If two vertices are adjacent when they are endpoints of an edge, then the maximum distance to the external face is called *width* [DLT84]. If two vertices are adjacent when they are on the same face and the external face is adjacent to all its vertices, then the maximum distance to the external face is called *outerplanarity* [Bak94]. If two faces are adjacent when they share an edge, then the maximum distance to the external face is called *depth* [BM88]. In [PT00, GM04], algorithms are proposed to minimize the maximum distance of the biconnected components of the graph from the external face, where two biconnected components are adjacent if they share a cutvertex. This measure, which is also called "depth," is a rougher indicator of the quality of the embedding but can be computed in linear time.

In [BM90], Bienstock and Monma present an algorithm to compute the planar embedding of an n-vertex planar graph with minimum maximum distance to the external face in $O(n^5 \log n)$ time, which is improved to $O(n^4)$ time in [ADP11]. The considered distance is the depth. However, it is possible to compute the radius, the width, and the outerplanarity of a graph by modifying and simplifying the algorithm for the minimum depth, since such distance measures are intrinsically simpler to compute than the depth [BM90]. The complexity bounds for computing such simpler distance measures is improved in [Kam07], where an algorithm that computes the outerplanarity of an n-vertex planar graph in $O(n^2)$ time is described. Simple variations of this algorithm can lead to compute the radius in $O(n^2)$ time and the width in $O(n^3)$ time [Kam07].

1.3.2 Deletion and Partition Problems

Deleting the minimum number of edges in order to obtain a planar graph is called *maximum planar subgraph* and proved to be NP-hard in [GJ79]. Analogously, deleting the minimum number of vertices in order to obtain a planar graph is called *maximum induced planar subgraph* and proved to be NP-hard in [Yan78].

The problem of partitioning the edges of a graph $G = (V, E)$ into k sets $E_1, \ldots, E_k$ in such a way that each graph $G_i = (V, E_i)$, with $i = 1, 2, \ldots, k$ is planar is called *graph thickness* and is shown to be NP-hard for $k = 2$ in [Man83].

1.3.3 Upward Planarity

If the input graph G is directed, adding the requirement that the drawing of G is upward, that is, that each edge is a curve of increasing y-coordinates, transforms the planarity problem into the upward planarity one, which was shown to be NP-complete in [GT01].

However, upward planarity testing turns out to be polynomial for several families of directed graphs:

1. If the digraph G is outerplanar. This problem was shown to be $O(n^2)$ in [Pap95].
2. If the digraph G is triconnected [BD91, BDLM94].
3. If the digraph G has a fixed embedding. An $O(n^2)$-time algorithm was introduced in [BDLM94], and the problem is linear in the case of embedded outerplanar graphs ([Pap95]).
4. If the digraph G is single-source. The $O(n^2)$-time algorithm described in [HL96] was improved to linear in [BDMT98].

1.3.4 Outerplanarity

Determining whether a graph is outerplanar and producing an outerplanar drawing of it is a problem that can be solved independently or by using a planarity algorithm as a subroutine. In fact, a graph $G = (V, E)$ is outerplanar if and only if the graph $G'(V', E')$ is planar, where $V' = V \cup \{v\}$ and E' is obtained from E by adding an edge (v_i, v) for each vertex $v_i \in V$.

Deleting the minimum number of vertices from a graph in order to make it outerplanar is NP-complete [Yan78].

1.4 History of Planarity Algorithms

Directly applying Kuratowski's characterization of planar graphs based on subdivisions would yield an exponential-time algorithm while Wagner's characterization based on minors would give a factorial-time algorithm. The first polynomial-time algorithms for planarity are due to Auslander and Parter [AP61], Goldstein [Gol63], and, independently, Bader [Bad64].

In 1974, Hopcroft and Tarjan [HT74] proposed the first linear-time planarity testing algorithm. This algorithm, also called "path-addition algorithm," starts from a cycle and adds to it one path at a time. However, the algorithm is so complex and difficult to implement that several other contributions followed their breakthrough. For example, about 20 years after [HT74], Mehlhorn and Mutzel [MM96] contributed a paper to clarify how to construct the embedding of a graph that is found to be planar by the original Hopcroft and Tarjan algorithm.

A different approach has its starting point in the algorithm presented by Lempel, Even, and Cederbaum [LEC67]. This algorithm, also called "vertex addition algorithm," is based on considering the vertices one-by-one, following an *st*-numbering; it has been shown to be implementable in linear time by Booth and Lueker [BL76], while a linear-time algorithm for computing the needed st-numbering was provided in [ET76]. Also in this case, a further contribution by Chiba, Nishizeki, Abe, and Ozawa [CNAO85] has been needed for showing how to construct an embedding of a graph that is found planar.

A further interesting algorithm [dOR06, dF08a, Bra09] is based on a characterization given by de Fraysseix and Rosenstiehl [dR82, dR85] in turn based on intuitions of Liu and Wu [Wu74, Ros80, Liu88, Liu89, Xu89]. For a long time, the algorithm has not been fully described in the literature but had a very efficient implementation in the Pigale software library [dO02].

However, although the planarity problem has been carefully studied in the above cited literature, the story of the planarity testing algorithms enumerates several more recent contributions. The motivations behind such relatively new papers are twofold. On one side, even if the known algorithms are combinatorially elegant, they are quite difficult to understand and to implement. On the other side, the researchers are interested in deepening the relationships between planarity and Depth First Search (DFS). Such relationships are clearly strong but, probably, up to now, not completely understood.

Two recent DFS-based planarity testing algorithms, whose similarities were stressed in [Tho99], are those presented by Shih and Hsu [SH93, SH99, Hsu03] and by Boyer and Myrvold [BM99, BM04].

The Shih-Hsu algorithm replaces biconnected portions of the graph with single nodes, called C-nodes, whose embedding is fixed.

The Boyer and Myrvold algorithm represents embedded biconnected portions of the graph with a data structure that allows the embeddings to be "flipped" in constant time.

1.5 Common Algorithmic Techniques and Tools

In this section, we introduce some definitions and common techniques used by the planarity testing algorithms. The most important technique, common to almost all the algorithms, is *Depth First Search*, or DFS. DFS is a method for visiting all the vertices of a graph G. It starts from an arbitrarily chosen vertex of G and continues moving from the current vertex to an adjacent one, as long as unexplored neighbors are found. When the current vertex has no unexplored neighbors, the traversal backtracks to the first vertex with an unexplored adjacent vertex.

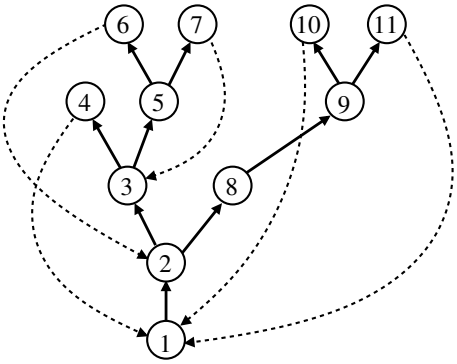


Figure 1.4 A DFS traversal of a graph. Thick lines represent the tree edges, while the back edges are drawn with dashed lines. Each vertex is identified with its DFS index.

The edges by which DFS discovers new vertices of G form a spanning tree T of G, called *Palm Tree*, or *DFS Tree*. The root of T is the vertex at which the traversal started. The edges of T are called *tree edges*, while the remaining edges of G are called *back edges* (or *co-tree edges*).

After performing a DFS traversal, each vertex v of G can be associated with a *DFS index*, $DFS(v)$, that is, the order in which v was reached during the DFS visit. The root of T has index one. For a tree edge (u, v), we have that $DFS(u) < DFS(v)$. On the contrary, a back edge is oriented from the end vertex with higher DFS index to the end vertex with lower DFS index. An example DFS is shown in Figure 1.4.

For each vertex v of G, we can also define two sets of edges, called $B_{in}(v)$ and $B_{out}(v)$. These sets contain, respectively, the back edges entering and exiting v. Note that each back edge in $B_{in}(v)$ connects v to a descendant in the DFS tree, while each back edge in $B_{out}(v)$ connects v to an ancestor. Given a tree edge $e = (u, v)$, its *returning edges* are those back edges that from a descendant of v (included v itself) go to an ancestor of u different from u itself. At last, the *lowpoint* of a vertex v, denoted by $lowpt(v)$, is the lowest DFS index of an ancestor of v reachable through a back edge from a descendant of v. Analogously, the *highpoint* of a vertex v, denoted by $highpt(v)$, is the highest DFS index of an ancestor of v reachable through a back edge from a descendant of v.

1.6 Cycle-Based Algorithms

The shared foundation of all algorithms in this section is an intuitive observation formalized in the Jordan curve theorem: every simple closed curve divides the plane into two connected regions, and hence there is no way to connect two points in both regions without crossing that curve.

Acyclic (undirected) graphs are forests and therefore planar. If a graph does contain a cycle, that cycle yields a simple closed curve in any planar drawing of it. Consequently, each of the remaining connected parts of the graph needs to be drawn entirely in one of the two connected regions bounded by the cycle. Deciding whether this is possible, and which region to choose, is the essence of planarity testing and embedding, respectively.

It will take three major steps to arrive at simple linear-time algorithms based on this observation. The first step consists in formalizing the approach in a recursive algorithm, the second step yields a linear-time realization of the algorithm, and the third step simplifies the second while adding a corresponding combinatorial characterization.

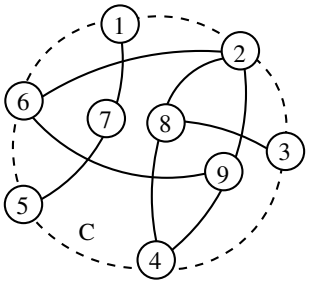


Figure 1.5 A biconnected graph G and a cycle C. The edges of C are drawn dashed.

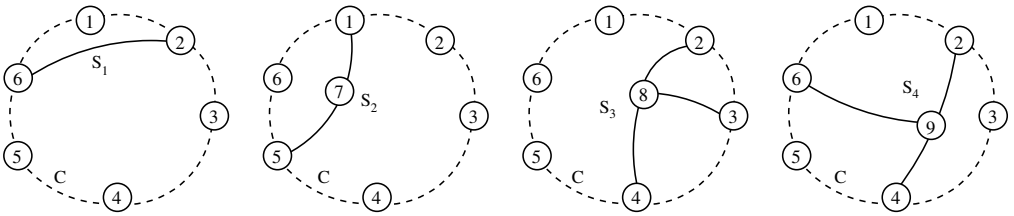


Figure 1.6 The four segments of graph G of Figure 1.5 separated by cycle C.

1.6.1 Adding Segments: The Auslander-Parter Algorithm

Algorithms based on the above cycle criterion were first proposed in [AP61] (see also [Gol63, Bad64, DETT99]).

To introduce the approach formally, consider a simple cycle C in a biconnected graph G. Recall that a graph is planar if and only if its biconnected components are, and that every edge of a biconnected graph is contained in at least one cycle. Each such cycle C yields a collection of connected, edge-induced subgraphs S_i, $i = 1, \ldots, k$ as follows. Either S_i is an edge that connects two vertices of C that are not consecutive (i.e., a *chord*), or S_i is induced by the edges of a connected component of $G \setminus C$ together with the edges connecting that component to C. Each S_i is called a *segment* and, because of biconnectivity, contains at least two vertices of C, referred to as the *attachments* of S_i. Note that vertices of C may be attachments of any number of segments. Figure 1.5 shows a biconnected graph G and a cycle C. The segments separated by C are depicted in Figure 1.6

A cycle C of G is said to be *separating* if it has at least two segments, while it is called *non-separating* otherwise. Of course, if G is a cycle, then C has no segments and is non-separating. In order to recur on subgraphs, the Auslander-Parter algorithm needs to pick a separating cycle.

LEMMA 1.1 [DETT99] Let G be a biconnected graph, let C be a non-separating cycle of G, and let S be its only segment. If S is not a path, then G has a separating cycle C' consisting of a subpath of C plus a path γ of S between two attachments.

Proof: Let u and v be two attachments of S that are consecutive in the circular ordering of C, let α be a subpath of C between u and v that does not contain any other attachment of S to C, and let β be the subpath of C between u and v different from α (see Figure 1.7 for an example). Since S is connected, there is a path γ in S between u and v. Let C' be the cycle obtained from C by replacing α with γ. We have that α is a segment of G with respect to C'. If S is not a path, let e be an edge of S not in γ. There is a segment

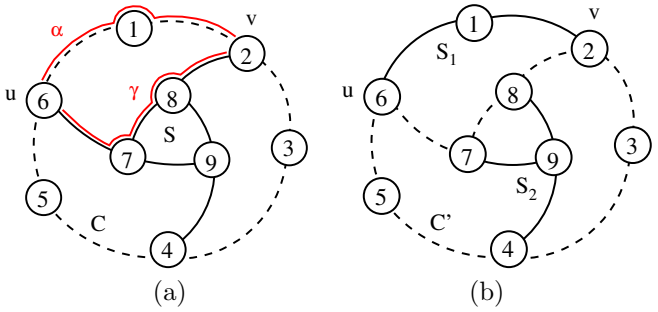


Figure 1.7 (a) A non-separating cycle C whose single segment S is not a path. Replacing subpath α with subpath γ as described in the proof of Lemma 1.1 yields a separating cycle C'.

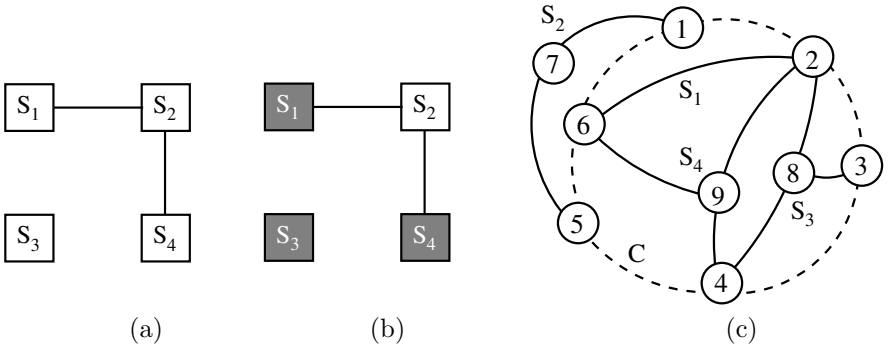


Figure 1.8 (a) The interlacement graph for the segments induced on graph G by the cycle C of Figure 1.5. (b) A possible bi-coloring of the interlacement graph. (c) The corresponding embedding choices for the segments of G, where segments colored black are placed inside C.

of C' distinct from α containing e. Therefore, if S is not a path, then C' has at least two segments and is thus a separating cycle of G. □

We have already argued that segments must be drawn entirely in one of the two regions created by the drawing of C. Two segments are said to be *compatible*, if they can be drawn in the same region of C, and *conflicting* otherwise. The following lemma shows that compatibility has a simple characterization.

LEMMA 1.2 Two segments are compatible, if and only if their attachments do not interleave.

The *interlacement graph* of the segments of G with respect to C is the graph whose vertices are the segments of G and whose edges are the pairs of interlacing segments. Figure 1.8(a) shows the interlacement graph for graph G and cycle C of Figure 1.5. If there are more than two pairwise incompatible segments, the graph is not planar, because there are only two regions in which they can be drawn. If G is planar, then the interlacement graph is bipartite and two-colorable, each color corresponding to one side of C (see Figures 1.8(b) and 1.8(c)). We can recursively check the planarity of all subgraphs obtained from the union of a segment S_i and C.

The AUSLANDER-PARTER algorithm is based on the following intuitive recursive characterization of planarity for biconnected graphs.

Theorem 1.1 *[DETT99] A biconnected graph G with a cycle C is planar if and only if the following two conditions hold:*

- *The interlacement graph of the segments of G with respect to C is bipartite.*
- *For each segment S of G with respect to C, the graph obtained by adding S to C is planar.*

Proof: If the graph is planar, it is easy to see that the two conditions hold by considering a planar drawing of it. If the two conditions hold, the proof is by construction and is based on the fact that compatible segments do not interleave (Lemma 1.2) and, hence, can be planarly arranged on the same side of C. □

The algorithm has three cases:

Trivial case. Graph G is a single cycle C. This case can only occur at the beginning of the computation and terminates it.

Base case. Cycle C separates a single segment, which is a path. This terminates the current branch of the computation (there will be no recursion).

Recursive case. A separating cycle C can be found in G. If the interlacement graph is not bipartite, the algorithm terminates with a non-planarity. Otherwise, recursion is needed on the subgraphs composed by C and each segment.

Here it is not necessary to describe this algorithm in more detail, because, in fact, the subsequent ones are instantiations of this rather generic approach.

It can be shown that the number of recursions is $O(n)$ and that the interlacement graph has size $O(n^2)$, yielding an $O(n^3)$ time algorithm. Also, it is worth mentioning that for a graph that turns out to be planar, the embedding is constructed bottom-up, where planar embeddings may have to be flipped, depending on which region they are placed in. There is an interesting alternative approach presented by Demoucron, Malgrange, and Pertuiset [DMP64]. Instead of recursively testing segments for planarity, they start from a fixed embedding of one cycle, and incrementally add only a path connecting two attachments of a segment into a face of the current embedding. This approach requires a careful selection of (facial) cycles and paths and yields a quadratic-time algorithm but is the only algorithm known to us that does not require alterations of preliminary embeddings.

1.6.2 Adding Paths: The HOPCROFT-TARJAN Algorithm

The relative inefficiency of recursively testing augmented segments for planarity is caused by a lack of control over the instances obtained when selecting a cycle.

By exploiting the special structure of DFS trees, Hopcroft and Tarjan [HT74] (see also [Deo76, RND77, Eve79, Wil80]) were able to serialize the combination of trivially planar segments (namely, paths) in a bottom-up fashion.

Let us start from a *spine cycle*, i.e., a fundamental cycle consisting of a path of tree edges that start at the root of the DFS tree together with a single back edge returning to the root. Call the subgraph consisting of only the spine cycle G_0. Next, segments are added recursively, one path at a time, which is why the algorithm is often referred to as the *path-addition approach*.

To explain the order in which paths are selected, consider the subgraph G_i consisting of the spine cycle and the first i paths, and an edge e that is incident to but not contained in G_i.

Define the segment $S(e)$ of e to be the inclusion-maximal connected subgraph containing e, in which no vertex of G_i has degree larger than one. Moreover, define the vertex with the lowest DFS number in $S(e)$ to be the *lowpoint of the segment*. Since G is biconnected, $S(e)$ contains at least two vertices of G_i, which we call attachments as well. By the order in which paths are inserted, the lowpoint of $S(e)$ will always be an attachment.

Now assume that the DFS tree was re-built to determine lowpoints and biconnected components. When exploring the tree once again, but this time by traversing edges with lower lowpoints first, we are effectively performing a recursive traversal of segments in which segments with lower lowpoints are traversed first. This order is crucially important for our ability to test efficiently whether segments are conflicting, because it ensures that the attachments of a segment are visited in order of non-decreasing lowpoints. We can therefore place lowpoints on a stack and remove them from the top of the stack during backtracking, thus maintaining in the stack all attachments in the order in which they appear in the lower part of the segment-defining cycle not yet backtracked over. Recall that two segments are compatible if their attachments do not interleave.

Again, we do not go into further details, because the approach is further simplified below. We just note that the algorithm can actually be implemented to run in linear time, but that this is quite difficult and that it took many years until this test was complemented by an embedding phase [MM96] (which runs in linear time).

Part of the difficulty is in the absence of a characterization of planarity that is closely tied to the workings of the algorithm.

1.6.3 Adding Edges: The de Fraysseix-Ossona de Mendez-Rosenstiehl Algorithm

While we have argued that the test of Hopcroft and Tarjan implements that of Auslander and Parter by recursively building up segments one path at a time, it turns out that the original approach can be further simplified by interpreting it on an even more detailed level, adding one edge at a time.

This does not only simplify the algorithm, it also yields a characterization of planarity that provides a less procedural proof of correctness and a straightforward embedding. Therefore, following the approach of [Bra09], we first recall the characterization and then revisit the algorithm.

Consider a connected undirected graph which needs not to be biconnected, and let $G = (V, T \uplus B)$ be the directed graph obtained from a DFS, where T is the set of tree edges and B the set of back edges. We say that G is a *DFS-orientation* of the original graph. Note that this is not a procedural definition, since such an orientation is characterized by consisting of a rooted spanning tree such that each non-tree edge defines a directed cycle. As each back edge returns to an ancestor of its source, it implicitly defines a cycle, which is called *fundamental cycle*. A back edge (u, v) is a *return edge* for each tree edge of its fundamental cycle, with the exception of the first tree edge exiting v.

DEFINITION 1.1 [dOR06] Let $G = (V, T \uplus B)$ be a DFS-oriented graph. A partition $B = L \uplus R$ of its back edges into two classes, referred to as *left* and *right*, is called *left-right partition*, or *LR partition* for short, if for every vertex v with incoming tree edge e and outgoing edges e_1, e_2

- all return edges of e_1 ending strictly higher than $lowpt(e_2)$ belong to one class and

- all return edges of e_2 ending strictly higher than $lowpt(e_1)$ belong to the other class.

Intuitively, the partition of the back edges into classes L and R corresponds to orienting the fundamental cycles in such a way that those closed by back edges in L are counterclockwise while those closed by back edges in R are clockwise.

Theorem 1.2 *A graph is planar if and only if it admits an LR partition.*

Necessity of the condition of Theorem 1.2 is straightforward: given a DFS tree and a planar embedding of the graph it suffices to assign each back edge to the classes L or R, depending on whether the fundamental cycle it closes is counterclockwise or clockwise, respectively. Sufficiency is shown by constructing a planar embedding from a given LR partition. First, observe that in an LR partition it can be assumed that all return edges from a tree edge e that return to $lowpt(e)$ are on the same side. Such a LR partition is called *aligned*. If a partition is not aligned, an equivalent aligned partition can be found.

In order to obtain a planar embedding, the LR partition is extended to cover also outgoing tree edges and, for each vertex v, a linear nesting order is defined on its exiting tree edges. This order contains both right and left outgoing edges of v mixed together: restricted to the right outgoing tree edges it gives their clockwise order around v and restricted to the left outgoing tree edges it gives their counterclockwise order around v. The final embedding for each vertex v is obtained by suitably interleaving outgoing tree edges with back edges entering v.

The extension of the LR partition to tree edges is straightforward. If a tree edge has some return edges (i.e., its source is neither the root nor a cut vertex), it is assigned to the same side as one of its return edges ending at the highest return point. Otherwise, the side is arbitrary.

To determine the linear nesting order for tree edges outgoing v, suppose first that all back edges belong to R and consider a fork consisting of tree edge $e = (u, v)$ and outgoing tree edges e_1 and e_2 exiting v. If both e_1 and e_2 have some return edges, v is a branching point of at least two overlapping fundamental cycles sharing e. Since both cycles are clockwise (all edges belong to R), they must be properly nested in order to avoid edge crossings. As the root of the DFS tree is assumed to be on the outer face, we have to put e_2 clockwise after e_1 (i.e., inside the cycle defined by it) if and only if the lowpoint of e_1 is strictly lower than that of e_2. The same holds if both have the same lowpoint but only e_2 is *chordal*, i.e., has another return point above it. On the contrary, if both L and R are not empty, it can happen that both e_1 and e_2 are chordal. In this case the tie is broken arbitrarily, because in any planar embedding these two edges must be on different sides.

Let $e = (v, w)$ be a tree edge. We denote by $L(e)$ ($R(e)$, respectively) the sequence of incoming back edges entering v from descendants of w ordered in such a way that if $b_1 = (x_1, v)$ and $b_2 = (x_2, v)$ are two such back edges, and if (z, x), (x, y_1), and (x, y_2) is the fork of the two cycles closed by b_1 and b_2, then b_1 comes before b_2 in $L(e)$ ($R(e)$, respectively) if and only if (x, y_1) comes before (x, y_2) ((x, y_1) comes after (x, y_2), respectively) in the adjacency list of x.

DEFINITION 1.2 Given an LR partition and a vertex v, let $e_1^L, \ldots, e_l^L$ be the left outgoing tree edges of v, and $e_1^R, \ldots, e_r^R$ its right outgoing tree edges. If v is not the root, let u be its parent. The clockwise *left-right ordering*, or *LR ordering* for short, of the edges around v is defined as follows:

$$(u, v), L(e_l^L), e_l^L, R(e_l^L), \ldots, L(e_1^L), e_1^L, R(e_1^L), L(e_1^R), e_1^R, R(e_1^R), \ldots, L(e_r^R), e_r^R, R(e_r^R),$$

where (u, v) is absent if v is the root.

The following lemma shows the sufficiency of the left-right planarity criterion of Theorem 1.2 (the proof by contradiction can be found in [Bra09]).

LEMMA 1.3 Given an LR partition, its LR ordering yields a planar embedding.

Hence, the search for a planar embedding of the input graph boils down to the search for an LR partition of its back edges. Fortunately, from the definition of LR partition directly come two constraints that have to be satisfied by back edges in L and R classes. Let $b_1 = (u_1, v_1)$ and $b_2 = (u_2, v_2)$ be two back edges with overlapping fundamental cycles and let $(u, v), (v, w_1), (v, w_2)$ be their fork.

1. b_1 and b_2 belong to different classes if $lowpt(w_2) < v_1$ and $lowpt(w_1) < v_2$.
2. b_1 and b_2 belong to the same class if there is an edge $e' = (x, y)$, with $x \in C(b_1) \cap C(b_2)$ and $y \notin C(b_1) \cap C(b_2)$ such that $lowpt(y) < min\{v_1, v_2\}$.

Of course, if a pair of back edges is subject to both the constraints above, no LR partition can exist and hence the graph is non-planar. By exploiting the constraints a quadratic planarity test and embedding algorithm can be found immediately. Namely, build a *constraint graph*, analogous to the interlacement graph of the Auslander-Parter algorithm, where each back edge is a vertex and each constraint is an edge, labeled "-1" if the two back edges have to belong to different classes and labeled "$+1$" if they have to belong to the same class. After contracting "$+1$" edges, test if the constraint graph is bipartite.

In order to transform this quadratic-time algorithm into a linear one, the constraint graph cannot be explicitly built and the tentative assignment of back edges to the L and R classes may be changed several times during the computation, which is structured as a further traversal of the DFS tree. Details of the linear-time algorithm can be found in [dOR06, dF08a, Bra09].

1.7 Vertex Addition Algorithms

Given a planar drawing Γ of a graph $G(V, E)$, we could delete one vertex at a time from Γ to obtain a sequence of smaller planar drawings ending with a single isolated vertex. The intuition that this process could be suitably reversed yields the so-called "vertex addition" algorithms.

We classify in this family the Lempel-Even-Cederbaum, the Shih-Hsu, and the Boyer-Myrvold algorithms, although we know that some authors proposed a different classification for their approach. The similarities between the Shih-Hsu and the Boyer-Myrvold algorithms were already pointed out in [Tho99], while a common view encompassing all the three algorithms was envisaged by Haeupler and Tarjan in [HT08].

Vertex addition algorithms start from an initial graph G_1 composed by one isolated vertex v_1. At each step $i = 2, \ldots n$, a new vertex v_i is added to the graph and the subgraph $G_i(V_i, E_i)$, induced by the current vertices $V_i = \{v_1, \ldots, v_i\} \subseteq V$, is considered. Two kinds of operations are performed: first, G_i is checked for planarity; second, some data structures are updated in order to allow analogous checks to be efficiently performed at step $i + 1$.

A key feature, common to this family of algorithms, is that the order in which the vertices are added is not arbitrary. Let $\overline{G}_i(\overline{V}_i, \overline{E}_i)$ be the subgraph of G induced by the vertices $\overline{V}_i = V - V_i$ that have still to be added to the graph. All the algorithms based on vertex

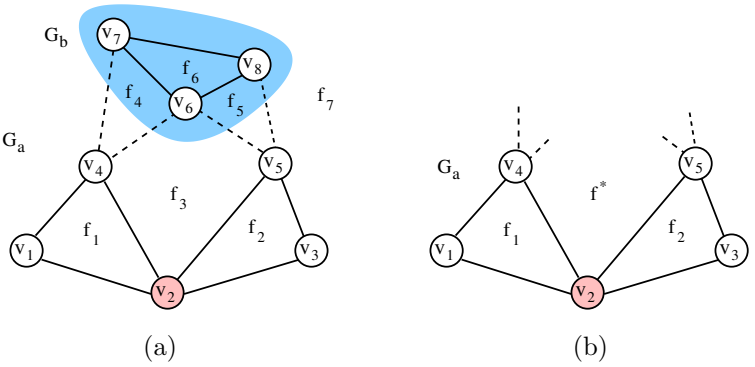


Figure 1.9 Properties of Lemma 1.4. (a) The embedding Γ where the connected subgraph G_b is highlighted. (b) The embedding Γ_a of G_a. By Property (α), v_6, v_7, and v_8 fall into f^*. By Property (β), f_1 and f_2 are also faces of Γ. By Property (γ), the cutvertex v_2 is on f^*.

addition require that $\overline{G}_i$ is connected for $i = 1, \ldots, n$, that is, the vertex addition order is a leaf-to-root order for some spanning tree of G. Lempel-Even-Cederbaum's algorithm, for example, requires that the vertices are added in the order given by an st-numbering; in the Shih-Hsu and in the Boyer-Myrvold algorithms, the order is that of a reverse DFS traversal of the graph. The importance of this requirement is stated by the following lemma.

LEMMA 1.4 Let $G(V, E)$ be a planar, connected graph and let $\{V_a, V_b\}$ be a bipartition of the vertices in V such that the graph $G_b(V_b, E_b)$ induced by V_b is connected. Consider any planar embedding Γ of G and denote by Γ_a the planar embedding Γ restricted to G_a. The following properties hold:

(α) Vertices of V_b are on the same face f^* of Γ_a.

(β) Each face f of Γ_a, with $f \neq f^*$, is also a face of Γ.

(γ) If G is biconnected, cutvertices of G_a are also incident to face f^* of Γ_a.

Proof: Property (α) trivially descends from the fact that G_b is connected and Γ is a planar embedding of G. Property (β) is also trivial. Suppose for a contradiction that $f \neq f^*$ is a face of Γ_a but not a face of Γ. Observe that f is a cycle of Γ and, since it is not a face of Γ, it contains at least one edge $e = (u, v)$ of Γ that is not an edge of Γ_a. If both u and v belong to V_a, we have a contradiction as e belongs to the graph induced by V_a but it is not in Γ_a. Otherwise, if one among u and v is not in V_a, we have again a contradiction since Property (α) ensures that $f = f^*$. This proves Property (β). Suppose that G is biconnected. If v is a cutvertex of G_a, then there is a face f of Γ_a that is incident at least two times on v. Since v is not a cutvertex of Γ, face f is a face of Γ_a but is not a face of Γ, and Property (β) ensures that f^* is the only face of Γ_a that has this property. □

An example that shows the three properties of Lemma 1.4 is depicted in Figure 1.9. Property (α) was also proved in [Eve79, Lemma 8.10] for the special case of connected subgraphs induced by an st-numbering.

Let ψ be a function $\psi : V \to \{1, \ldots, n\}$ that assigns a different index to each vertex of G. We say that ψ is a *proper numbering* of G if for each i we have that the subgraph $\overline{G}_i(\overline{V}_i, \overline{E}_i)$ induced by $\overline{V}_i = \{v \mid \psi(v) > i\}$ is connected. In order to simplify the notation

in the remaining part of this chapter, we denote by v_i the vertex for which $\psi(v_i) = i$. Vertex addition algorithms require that vertices are considered in the order imposed by a proper numbering, hence exploiting at each step the properties of Lemma 1.4. Namely, Property (α) guarantees that vertices and edges can be added to a single face f^* of Γ_i, which can be assumed to be the outer face. Property (β) implies that once a vertex or edge is closed inside an internal face of Γ_i it does not need to be considered again (this is a key point to ensure linearity). Finally, Property (γ) justifies the usual assumption, common to most vertex addition algorithms, that G is biconnected.

Properties (α) and (β) lead to the following lemma.

LEMMA 1.5 Let ψ be any proper numbering of a planar, connected graph G. Denote by G_i the subgraph of G induced by vertices in $V_i = \{v \mid \psi(v) \leq i\}$. There exists a sequence of planar embeddings Γ_i of G_i, with $i = 1, \ldots, n$, such that, for $i = 1, \ldots, n-1$, all internal faces of Γ_i are also internal faces of Γ_{i+1}.

Proof: Let Γ_n be a planar drawing of G with v_n on the external face and let Γ_i, with $i = 1, \ldots, n-1$, be the embedding of G_i obtained from Γ_n by removing the vertices v_j, with $j = i+1, \ldots, n$. Vertex v_n is on the external face of Γ_n by definition. Since $\overline{G}_i$ is connected, vertex v_i is also on the external face of Γ_i for any $i = n-1, n-2, \ldots, 1$. Also, $V_a = \{v_1, \ldots, v_{i-1}\}$ and $V_b = \{v_i\}$ is a bipartition of the vertices of G_i of which Γ_i is a planar embedding and $G_b(V_b, \emptyset)$ is trivially connected. Lemma 1.4 applies and by Property (β) we have that all the faces of Γ_{i-1} with the exception of f^* are also faces of Γ_i. Since the external face of Γ_{i-1} is not a face of Γ_i, any other internal face f of Γ_{i-1} is also a face of Γ_i. Finally, as the external face of Γ_i contains v_i, which does not belong to G_{i-1}, face f is an internal face of Γ_i. □

Provided that G is planar, Lemma 1.5 can be exploited for devising an incremental planarity algorithm that, starting from Γ_1, i.e., the trivial embedding of the isolated vertex v_1, computes Γ_i, with $i = 2, \ldots, n$, by adding at each step a vertex v_i on the outer face of Γ_{i-1}, until an embedding Γ_n of the whole graph is produced. Also, Lemma 1.4 provides an indication of what are the properties that these Γ_i should have. Namely, call *outer vertices of* G_i the cutvertices of G_i and the vertices of G_i adjacent to v_{i+1}, v_{i+2}, ..., v_n. Properties (α) and (γ) of Lemma 1.4 state that if G is biconnected, which can be assumed, each Γ_i necessarily has its outer vertices on the outer face.

Still, computing the sequence of Γ_i, with $i = 1, \ldots, n$, is not an easy task. First, G_i may be not connected. Second, it is easy to see that not any embedding of G_i with its outer vertices on the external face is equivalent to any other. In fact, given a planar graph G, there may exist a planar embedding Γ_i of G_i that has the outer vertices of G_i on the external face but is not obtainable from some planar embedding of G by vertex deletion (Figure 1.10 provides an example).

Hence, although we know that, starting from any proper numbering ψ of G, the planarity of G implies the existence of a sequence of planar embeddings Γ_i satisfying the conditions of Lemma 1.5, we do not know how to find such a sequence, and choosing a wrong embedding Γ_i along the way would lead to a failure of the whole process even if G is planar. The following lemma comes in help.

LEMMA 1.6 In any planar embedding of a biconnected graph G where vertices $v_1, v_2, \ldots, v_k$ share the same face, they appear in the same circular order up to a reversal.

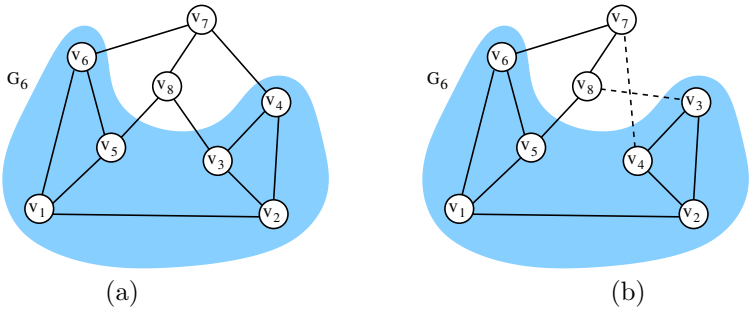


Figure 1.10 A planar graph G with subgraph G_6 highlighted. (a) and (b) show two planar embeddings of G_6, both with the outer vertices of G_6 on the external face. The embedding in (a) is compatible with a planar drawing of G while the embedding in (b) is not.

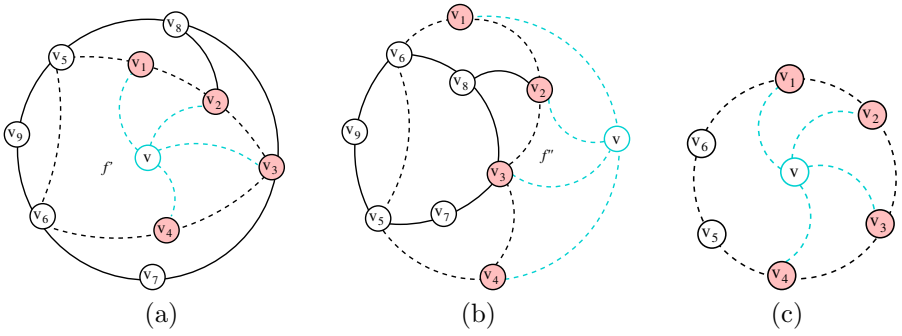


Figure 1.11 (a), (b) Two planar embeddings of a biconnected graph where vertices v_1, v_2, v_3, and v_4 (highlighted in the figure) share the same face. Vertex v is added as in the proof of Lemma 1.6.

Proof: The statement is trivial for $k = 2, 3$, since any circular sequence of 2 or 3 labels is equal to any other up to a reversal. Consider two planar embeddings Γ' and Γ'' of G such that vertices $v_1, v_2, \ldots, v_k$, with $k \geq 4$, share the face f' in Γ' and f'' in Γ'' (see Figure 1.11(a) and 1.11(b) for an example). The proof is based on the trivial observation that a dummy vertex v can be inserted into both f' and f'' and planarly connected to $v_1, v_2, \ldots, v_k$. Since G is biconnected, the cycle face f' is simple (see Figure 1.11(a)). Hence, the subgraph composed by the edges and vertices of f' and v is a wheel (dashed lines in Figs. 1.11(a), 1.11(b), and 1.11(c)) and admits a unique planar embedding up to a reversal. It follows that the circular order of the edges around v is the same in Γ' and in Γ'' up to a reversal. □

Lemma 1.6 applied to each block of G_i is stated in [Eve79, Lemma 8.12] for the special case of subgraphs induced by st-numberings. When iteratively computing a planar embedding for G, the practical use of Lemma 1.6 is that, although in general no definitive choice can be made on the embedding of G_i, something can be said about the embedding of its blocks. Namely, apart from a possible flip, an embedding for them can be computed that is always compatible with a planar embedding of the whole graph, provided it exists. Surprisingly, this is the only thing that can be safely computed for the embedding of G_i. All the more so, this little amount of information suffices for computing analogous embeddings for the blocks of G_{i+1}, and, since $G_n = G$ is biconnected, at the last step a planar embedding Γ_n of

the whole graph is obtained. Finally, the following lemma shows that if the process stops, the graph is not planar.

LEMMA 1.7 Let G be a graph and let ψ be any proper numbering of G. Denote by G_i, with $i = 1, \ldots, n$ the subgraph of G induced by vertices in $V_i = \{v \mid \psi(v) \leq i\}$ and by $B_i^1, B_i^2, \ldots, B_i^{b_i}$ the b_i blocks of G_i. For a given k, $1 \leq k \leq n-1$, let $\Gamma(B_k^j)$, $1 \leq j \leq b_k$, be arbitrary embeddings of B_k^j with the outer vertices of G_k on their outer faces. If the blocks of G_{k+1} cannot be embedded such that the outer vertices of G_{k+1} are on the outer face and $\Gamma(B_k^j)$, $1 \leq j \leq b_k$, are preserved up to a flip, then G is not planar.

Proof: Suppose for a contradiction that G is planar and that there is no planar embedding for all its blocks B_{k+1}^j, $1 \leq j \leq b_k$, such that the outer vertices of G_{k+1} are on the outer face and the blocks of G_k are embedded, up to a flip, as in $\Gamma(B_k^j)$, $1 \leq j \leq b_k$. Since G is planar, by Lemma 1.5, there is a pair of planar drawings Γ_k^* of G_k and Γ_{k+1}^* of G_{k+1}, both with their outer vertices on the outer face. By Lemma 1.6 the outer vertices of each block of G_k appear in the same order, up to a reversal, both in $\Gamma(B_k^j)$, $1 \leq j \leq b_k$, and in Γ_{k+1}^*. Hence, all embeddings $\Gamma(B_k^j)$ can be inserted into Γ_{k+1}^* yielding a planar embedding for the blocks B_{k+1}^j, $1 \leq j \leq b_k$, such that the outer vertices of G_{k+1} are on the outer face: a contradiction. □

Lemma 1.7 proves the soundness of the vertex addition approach. In fact, it shows that iteratively building a planar embedding of the input graph G is not only a sufficient condition for the planarity of G, which is obvious, but also a necessary condition, as G is not planar if one step of the iterative process cannot be accomplished. Usually, in the vertex addition literature, the non-planarity of the input graph in case of failure of the proposed algorithms is proved by a complex case analysis, spread all over the description of the algorithm steps, aimed at identifying a subgraph isomorphic to K_5 or $K_{3,3}$ for each possible cause of failure. Instead, Lemma 1.7 provides a direct proof of the correctness of the approach that avoids the use of Kuratowski's theorem, as claimed in [HT08].

Observe that, since the internal faces of the blocks are preserved in the final embedding of G, at each iterative step of the vertex addition algorithms the embedded blocks may be flipped and composed together, but they are never inserted one into the other. Hence, all vertex addition algorithms make use of suitable data structures to describe the subgraph G_i that has been explored so far and in particular the embedding of its blocks. These data structures allow for permuting the blocks around the cutvertices and for flipping the blocks in constant time. In the LEMPEL-EVEN-CEDERBAUM algorithm, the data structure is Booth and Lueker's PQ-tree. The SHIH-HSU algorithm uses PC-trees. The BOYER-MYRVOLD algorithm uses the `bicomp` data structure. The purpose of these data structures is analogous: they allow us to flip a portion of the graph (a block) in constant time; they allow us to permute (or to leave undecided) the order of the blocks around a cutvertex until the blocks are merged together.

1.7.1 The LEMPEL-EVEN-CEDERBAUM Algorithm

The LEMPEL-EVEN-CEDERBAUM algorithm was the first one to exploit the vertex addition paradigm [LEC67] (see also [Eve79, BFNd04]). It is no surprise, therefore, that in order to ease the computation several simplifying assumptions are made. First, but this is usual, the input graph is assumed to be biconnected. Second, the description of the algorithm in [LEC67] only checks the planarity of the input graph, without actually computing a planar

embedding if it exists. This gap was closed by Chiba, Nishizeki, Abe, and Ozawa [CNAO85] some decades later. Third, a proper numbering of the vertices of G is required that also ensures that G_i, the graph induced by V_i, is connected. Namely, given any edge (s,t) of a biconnected graph $G(V,E)$ with n vertices an *st-numbering* of G is a function $\psi : V \to \{1,\ldots,n\}$ that assigns to each vertex a different index, such that: (i) $\psi(s) = 1$; (ii) $\psi(t) = n$; and (iii) any vertex except s and t is adjacent both to a lower-numbered and to a higher-numbered vertex. This strong constraint, which implies that both the st-numbering and its reversal are proper numberings, fostered the search for a linear-time algorithm to actually compute an st-numbering of a biconnected graph. Such an algorithm was not known when the approach was introduced (the time complexity of the algorithm used in [LEC67] is $O(nm)$ [ET76]), and was finally found in [ET76].

Working of the algorithm

A *bush* is a single-source connected planar directed graph that admits a planar embedding, called a *bush form*, where all vertices of degree one are on the outer face.

Let G be a biconnected graph, let ψ be an st-numbering of G, and let G_i be the graph induced by vertices $\{v_1,\ldots,v_i\}$. Graph G can be assumed to be directed, where each edge is oriented from the vertex with the lower value to the vertex with higher value of ψ (see Figure 1.12(a) for an example). Denote by $\mathcal{B}_i$ the graph G_i augmented with the edges of G incident to the outer vertices of G_i. These edges are called *virtual edges*, while the leaves that they introduce in $\mathcal{B}_i$ are *virtual vertices*. Virtual vertices are labeled with the same indexes they have in G, and multiple instances of the same vertex are kept separate in $\mathcal{B}_i$. Since G_i is determined by an st-numbering, $\mathcal{B}_i$ is connected. Observe that a planar embedding of $\mathcal{B}_i$ with the virtual vertices on the outer face corresponds to a planar embedding of G_i with the outer vertices on the outer face. Hence, if G is planar, by Lemma 1.5 $\mathcal{B}_i$ is a bush. See Figure 1.12 for an example of a graph G_i and the corresponding bush $\mathcal{B}_i$. A bush form $\Gamma_{\mathcal{B}_i}$ is usually represented by drawing all the virtual vertices on the same horizontal line (dashed line of Figure 1.12(b)).

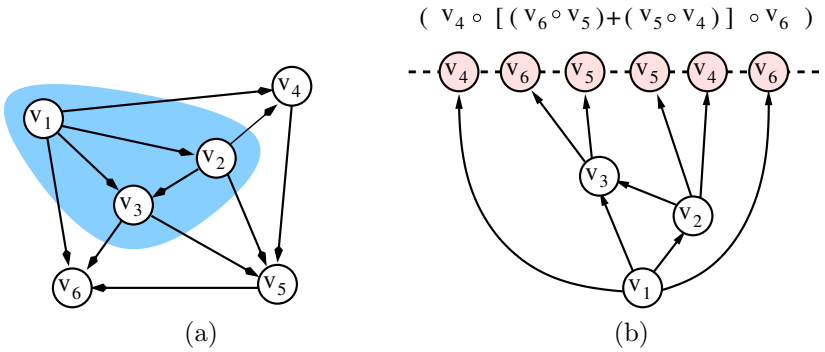


Figure 1.12 (a) A directed planar graph. Labels correspond to an st-numbering of the vertices. The highlighted area is the subgraph G_3 induced by $\{v_1, v_2, v_3\}$. Observe that, due to the st-numbering, both G_3 and $G - G_3$ are connected. (b) The bush form $\mathcal{B}_3$.

Bush form $\Gamma_{\mathcal{B}_i}$ contains a planar embedding of all biconnected components of G_i, and Lemma 1.7 ensures that such embeddings can be kept fixed up to a flip when searching for a planar drawing of G.

The strategy of the algorithm is that of focusing on the virtual vertices of $\mathcal{B}_i$ and of encoding the linear order that they have in $\Gamma_{\mathcal{B}_i}$ into a suitable algebraic expression $\varepsilon(\Gamma_{\mathcal{B}_i})$ that implicitly represents all their permutations compatible with a planar embedding of $\mathcal{B}_i$ with virtual vertices on the outer face.

The definition of $\varepsilon(\Gamma_{\mathcal{B}_i})$ can be inductively provided as follows. Let v be the source of $\Gamma_{\mathcal{B}}$. If $\Gamma_{\mathcal{B}}$ is a trivial bush form consisting of a single directed edge (v, u) then $\varepsilon(\Gamma_{\mathcal{B}}) = u$. Otherwise, if v is a cutvertex splitting $\Gamma_{\mathcal{B}}$ into bush forms $b_1, b_2, \ldots, b_k$, let $\varepsilon(b_1), \varepsilon(b_2), \ldots, \varepsilon(b_k)$ be the corresponding expressions for $b_1, b_2, \ldots, b_k$. The algebraic expression associated with $\Gamma_{\mathcal{B}}$ is $\varepsilon(\Gamma_{\mathcal{B}}) = (\varepsilon(b_1) \circ \varepsilon(b_2) \circ \ldots \circ \varepsilon(b_k))$. Observe that any permutation of $b_1, b_2, \ldots, b_k$ is compatible with a planar embedding of $\mathcal{B}$. Finally, if v is not a cut vertex of $\Gamma_{\mathcal{B}}$, consider the biconnected component b of $\Gamma_{\mathcal{B}}$ including v and let $u_1, u_2, \ldots, u_k$ be the cut vertices of $\mathcal{B}$ belonging to b. Observe that each subgraph of $\Gamma_{\mathcal{B}}$ routed at u_i, with $i = 1, \ldots, k$, is a bush form b_i. Let $\varepsilon(b_1), \varepsilon(b_2), \ldots, \varepsilon(b_k)$ be the corresponding expressions for $b_1, b_2, \ldots, b_k$. The algebraic expression associated with $\Gamma_{\mathcal{B}}$ is $\varepsilon(\Gamma_{\mathcal{B}}) = [\varepsilon(b_1) + \varepsilon(b_2) + \ldots + \varepsilon(b_k)]$. Observe that flipping the biconnected component b corresponds to flipping the expression $[\varepsilon(b_k) + \varepsilon(b_{k-1}) + \ldots + \varepsilon(b_1)]$.

Figure 1.13 illustrates an example of permutations and flipping in a bush form.

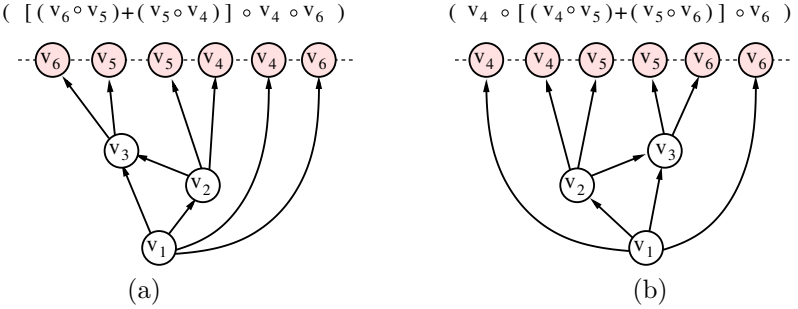


Figure 1.13 (a) A permutation of the bush form of Figure 1.12(b). (b) A flip of the bush form of Figure 1.12(b).

Given a bush form $\Gamma_{\mathcal{B}_i}$, the *reduction* operation changes the embedding of $\mathcal{B}_i$, by permuting bush forms attached to cut vertices and by flipping biconnected components, and produces a bush form $\Gamma'_{\mathcal{B}_i}$ where all virtual vertices labeled v_{i+1} are consecutively disposed. If this is not possible, then there is no way of adding vertex v_{i+1} to the embedding while keeping all outer vertices of G_{i-1} on the outer face, and by Lemma 1.7 the graph is not planar. If this is possible, then a *substitution* operation is performed on $\Gamma_{\mathcal{B}_i}$, obtaining a drawing $\Gamma_{\mathcal{B}_{i+1}}$. Namely, the virtual vertices labeled v_{i+1} are merged together, and for each edge (v_{i+1}, v_j) exiting v_{i+1} a new virtual vertex v_j is introduced and connected to v_{i+1}.

In the original description of the Lempel-Even-Cederbaum algorithm, these operations are not actually performed. Instead, it is shown that the reduction operation on $\Gamma_{\mathcal{B}_i}$ corresponds to an equivalent transformation on $\varepsilon(\Gamma_{\mathcal{B}_i})$ that produces an algebraic expression $\varepsilon(\Gamma'_{\mathcal{B}_i})$ where all the variables v_{i+1} are consecutive. Analogously, the substitution operation corresponds to the removal of the sequence of variables v_{i+1} which are replaced by $(v_{j_1} \circ v_{j_2} \circ \ldots \circ v_{j_k})$, where $v_{j_1}, \ldots, v_{j_k}$ are the vertices directly attached to v_{i+1}.

Data structures

The problem of efficiently identifying the flips and the permutations needed to reduce $\Gamma_{\mathcal{B}_i}$ (or, equivalently, needed to normalize $\varepsilon(\Gamma_{\mathcal{B}_i})$) is solved in [BL76], where the PQ-tree data structure is introduced. Intuitively, a PQ-tree is a data structure corresponding to the syntax tree of the expression $\varepsilon(\Gamma_{\mathcal{B}_i})$. Namely, a *PQ-tree* is a rooted, directed, ordered tree with three types of nodes: P-nodes, Q-nodes, and leaves. For each $\circ$ operation $(\epsilon_1 \circ \epsilon_2 \circ \ldots \circ \epsilon_k)$ in ε, the corresponding PQ-tree has a *P-node* with children $PQ(\epsilon_1), \ldots, PQ(\epsilon_k)$. Also, for each $+$ operation $(\epsilon_1 \circ \epsilon_2 \circ \ldots \circ \epsilon_k)$ in ε, the corresponding PQ-tree has a *Q-node* with children $PQ(\epsilon_1), \ldots, PQ(\epsilon_k)$. The children of a P-node can be arbitrarily permuted, while the order of the children of a Q-node can be reversed. In [BL76], it is shown how a bottom-up computation starting from all leaves labeled v_{i+1} is sufficient to compute a sequence of permutations and flips that consecutively disposes all v_{i+1} leaves. Only the smallest subtree that contains the v_{i+1} leaves is traversed.

1.7.2 The Shih-Hsu Algorithm

The Shih-Hsu algorithm either constructs a planar embedding of the input graph G or fails and outputs the information that G is not planar [SH93, SH99] (see also [Hsu01, Boy05]). The proper numbering ψ of the vertices of G used by the Shih-Hsu algorithm is obtained by a DFS traversal of G. Namely, vertices are considered in reverse DFS order, where the root r of the DFS tree has $\psi(r) = n$. Therefore, differently from the Lempel-Even-Cederbaum algorithm, although the graph $\overline{G}_i(\overline{V}_i, \overline{E}_i)$ induced by $\overline{V}_i = \{v \mid \psi(v) > i\}$ is always connected, the graph $G_i(V_i, E_i)$ induced by vertices in $V_i = \{v \mid \psi(v) \leq i\}$ is not guaranteed to be connected. At step 1, graph G_1 has vertex v_1 only. At a generic step i, with $i = 2, \ldots, n$, an embedding Γ_{G_i} is obtained from the embedding of $\Gamma_{G_{i-1}}$ by adding vertex v_i together with all edges connecting it to vertices with lower values of ψ. The strategy used by the Shih-Hsu algorithm is that of characterizing those configurations that determine a non-planarity, and by giving a recipe to build Γ_{G_i} otherwise.

As G_i is not necessarily connected, at each step i a planar embedding of each connected component of G_i is encoded into a data structure called PC-tree. A *PC-tree* $\mathcal{T}$ is a rooted, ordered tree with two types of nodes: P-nodes and C-nodes. While the neighbors of a P-node can be arbitrarily permuted, C-nodes come with a cyclic ordering of their adjacency list, which can only be reversed. Intuitively, P-nodes represent regular nodes of an embedded partial graph, while C-nodes represent biconnected components. Consider a planar embedding of a connected component C of graph G_i such that the outer vertices of C are on its outer face. The PC-tree $\mathcal{T}$ associated with C can be easily obtained from $\mathcal{C}$ by replacing each biconnected component of C with a C-node connected to the outer vertices of it in the same circular order as they appear on the border of the biconnected component. In order to simplify the tree, each C-node representing a trivial biconnected component composed of a single edge connecting two cutvertices v_j and v_k is replaced with a single edge attached to the two P-nodes corresponding to v_j and v_k. Let r be the *root* of the connected component C, i.e., the vertex of C with higher value of ψ. Observe that r is an outer vertex of C and always corresponds to a P-node of its PC-tree.

The PC-trees associated with G_i represent all the planar embeddings of G_i such that each connected component of G_i has its outer vertices on the outer face. In particular, if G_i is connected, the correspondence between its single PC-tree $\mathcal{T}$ and the PQ-tree of G_i used by the Lempel-Even-Cederbaum algorithm is apparent, since the former is obtained from the latter by removing leaves and replacing Q-nodes with C-nodes.

Working of the algorithm

At step 1, graph G_1 only has one isolated vertex labeled v_1 and its PQ-tree is a single P-node associated with vertex v_1.

At a generic step i, graph G_{i-1} has been already processed and its PC-trees have been computed. When the new vertex v_i is added, all tree edges and back edges connecting v_i to G_{i-1} are considered. Suppose that only a tree edge (v_i, u) exits from v_i. In this case, only the PC-tree corresponding to the connected component of G_{i-1} needs to be updated. Otherwise, if v_i has more than one child $u_1, u_2, \ldots, u_k$, then v_i is a cutvertex of G_i; a new P-node is introduced for it and suitably attached to the PC-trees of the connected components $C_1, C_2, \ldots, C_k$ of G_{i-1} containing $u_1, u_2, \ldots, u_k$, respectively, producing a single PC-tree for the new connected component of G_i. Consider a child u of v_i. The PC-tree $\mathcal{T}$ corresponding to the connected component containing u can be attached to v_i in a way that is independent of the PC-trees corresponding to the other children of v_i. Hence, for simplicity of description, we will assume that v_i has a single child u.

Let C be the connected component of G_{i-1} containing u. If no back edge from C attaches to v_i, then the P-node introduced for v_i is attached to the P-node representing r in $\mathcal{T}_i$, and step i concludes. Otherwise, suppose some vertex of C has some back edge to v_i. Since the input graph is biconnected, nodes of $\mathcal{T}_{i-1}$ either have $highpt = i$ or have $lowpt > i$, or both. Call *relevant node* each node w of $\mathcal{T}_{i-1}$ such that $highpt(w) = i$ and $lowpt(w) > i$. It is easy to see that the parent of w either is r or is a relevant node in its turn. Hence, relevant nodes form a subtree of the PC-tree rooted at r. By leveraging the relevant nodes subtree, it is possible to efficiently check the planarity of G_i and to compute the PC-tree updated with the P-node for u.

Namely, call *terminal nodes* the leaves of the subtree of the PC-tree composed by relevant nodes. We have the following lemma.

LEMMA 1.8 If $\mathcal{T}$ has more than two terminal nodes, then G_i (and hence G) is not planar.

Therefore, if G is planar, $\mathcal{T}_{i-1}$ has one or two terminal nodes and the relevant nodes subtree of $\mathcal{T}_{i-1}$ is either a path or a Y-shaped tree, respectively (see Figure 1.14).

Also, observe that an edge exiting a relevant node may be of five different types:

(i) a tree edge to another relevant node;
(ii) a back edge to v_i;
(iii) a tree edge to a subtree whose back edges are all type-(ii) edges; or
(iv) a back edge to a node v_j with $j > i$;
(v) a tree edge to a subtree whose back edges are all type-(iv) edges.

Subtrees attached to edges of type (iii) are called *i-subtrees*, while subtrees attached to edges of type (v) are called *i*-subtrees*. In Figure 1.14, i-subtrees are represented with black triangles and i*-subtrees with white triangles.

The Shih-Hsu algorithm either identifies a non-planarity or finds a planar arrangement of the back edges to v_i and the i-subtrees to produce a new C-node that represents the block determined by the additions of the back edges to v_i. The algorithm considers four main cases, depending on whether some relevant node is a C-node, and depending on whether $\mathcal{T}_{i-1}$ has one or two terminal nodes.

The easiest case is when $\mathcal{T}_{i-1}$ has exactly one terminal and all the relevant nodes are P-nodes. In fact, in this case all i-subtrees and back edges to i can always be embedded

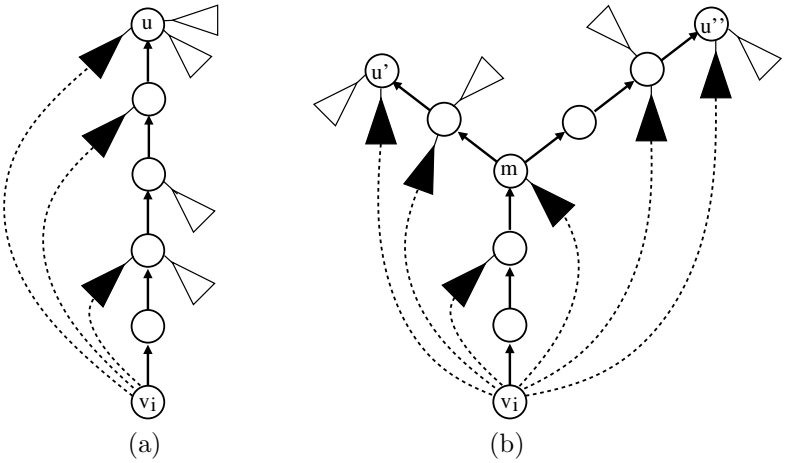


Figure 1.14 (a) Relevant nodes of $\mathcal{T}_{i-1}$ have one terminal. (b) Relevant nodes of $\mathcal{T}_{i-1}$ have two terminals.

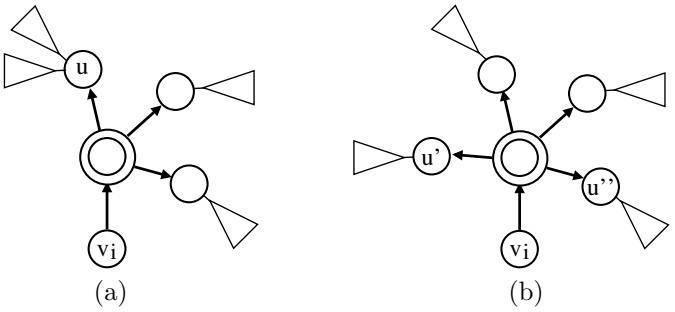


Figure 1.15 (a) The example of Figure 1.14(a) after contraction. The double border identifies C-nodes. (b) The example of Figure 1.14(b) after contraction.

on one side of $\mathcal{T}_{i-1}$, and the embedded part can be replaced with a C-node, as shown in Figure 1.15(a).

The case when $\mathcal{T}_{i-1}$ has exactly one terminal and some relevant node is a C-node, is analogous, with the difference that the constraints enforced by C-nodes (whose adjacency list can only be flipped) have to be taken into account and may cause a non-planarity whenever, no matter how they are flipped, they force one i-subtree (or back edge to v_i) to be outside the new block or one i*-subtree (or back edge to v_j, with $j > i$) to be inside it.

The most difficult case is when $\mathcal{T}_{i-1}$ has two terminal nodes u' and u''. Let m be their common ancestor, P be the unique path in $\mathcal{T}_{i-1}$ from u' to u'', and P' be the path from r to m. If all relevant nodes are P-nodes, then we have the following planarity criterion:

LEMMA 1.9 Graph G_i is planar if and only if any node internal to P' has edges of type (i), (ii), or (iii).

In fact, it is easy to see that if the conditions of Lemma 1.9 are satisfied a new block can be planarly embedded, its border being composed by path P and two paths from the two terminal nodes to v_i and containing all i-subtrees and back edges to v_i. Such a block is replaced by a C-node as shown in Figure 1.15(b). Again, if some relevant node is a C-node,

its constraints on the embedding need to be taken into account and yield a more intricate, although not difficult, case.

Data Structures

A tricky point of the SHIH-HSU algorithm is when a newly identified block has to be replaced with a C-node. To understand why this operation is critical, consider that, in order to have a linear-time algorithm, each node of the PC-tree should have a pointer to the parent node. Such a pointer is used, for example, when, starting from the current vertex v_i, its incoming back edges are considered and i-subtrees are traversed moving from child to parent. This operation is needed to identify the relevant node subtree and its terminals. Observe that i*-subtrees cannot be traversed without losing linearity. Also, even identifying them by browsing the adjacency list of a relevant node would have the same result. If the block was naively replaced by a C-node structure as shown in Figure 1.15 the pointers to the parent of a possibly linear number of children would have to be updated.

Perhaps the easiest way to address this problem is that of encoding the neighborhood of C-nodes with a strategy analogous to that used in [BM99, BM04], which allows us to efficiently traverse the boundary of a block in parallel and flip it when needed [Hsu01, Hsu03, BFNd04]. A second approach, inspired by the analogous operation on Q-nodes of PQ-trees [LEC67], is that of borrowing the parent pointer from sibling to sibling. The two approaches turn out to be similar, since browsing siblings of a C-node in search for the parent pointer is equivalent to traversing the corresponding block border [Hsu01, Hsu03].

1.7.3 The BOYER-MYRVOLD Algorithm

The BOYER-MYRVOLD algorithm [BM99, BM04] (see also [Tho99, BCPD04, HT08]) has several features in common with the SHIH-HSU algorithm, so much so that the two have been sometimes identified [Tho99, HT08]. The proper numbering ψ of the vertices of G used by the BOYER-MYRVOLD algorithm is again a reverse DFS order. The general strategy is that of explicitly maintaining a "flexible" planar embedding of each connected component of G_i with the outer vertices on the outer face. This embedding is "flexible" in the sense that each block can be flipped in constant time, whatever its size, while the permutation of the blocks around cutvertices is left undecided. In order to achieve this, each block of G_i is maintained separately from the others in a special structure, and the cutvertex that has higher value of ψ in one block B, called the *root* of B, has a pointer to the corresponding cutvertex in the parent block.

Working of the algorithm

The algorithm described in [BM99] was simplified in [BM04]. First, we describe the primitive version in [BM99], which, in our opinion, is more intuitive. Second, we sketch the differences with [BM04].

The computation starts with an initial set of blocks corresponding to the tree edges of the DFS tree of G (see Figure 1.16). Hence, it could be argued that this is not a vertex addition algorithm, since all vertices are in place from the first iteration. Actually, a vertex v_j with index higher than the current iteration i is ignored until iteration j is reached. Vertices are considered in reverse DFS order, starting from v_1 and ending with the root v_n of the DFS tree (see Figure 1.16(a)). If vertex v_i has no incoming back edges, no operation is needed at iteration i.

So, for example, running the algorithm on the example of Figure 1.16(b) would not perform any operation at steps $1, 2, \ldots, 8$, as vertices $v_1, v_2, \ldots, v_8$ don't have incoming

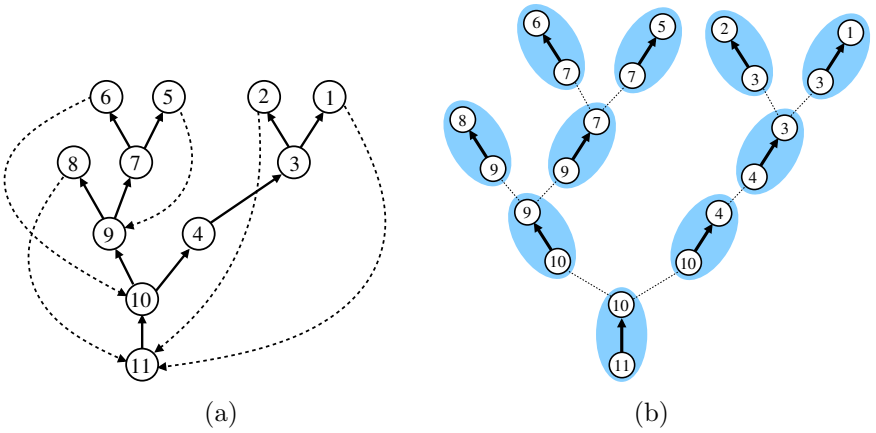


Figure 1.16 (a) The same DFS tree of Figure 1.4, where vertices are labeled with their reverse DFS index. (b) The BOYER-MYRVOLD algorithm starts by creating a block for each edge of the tree.

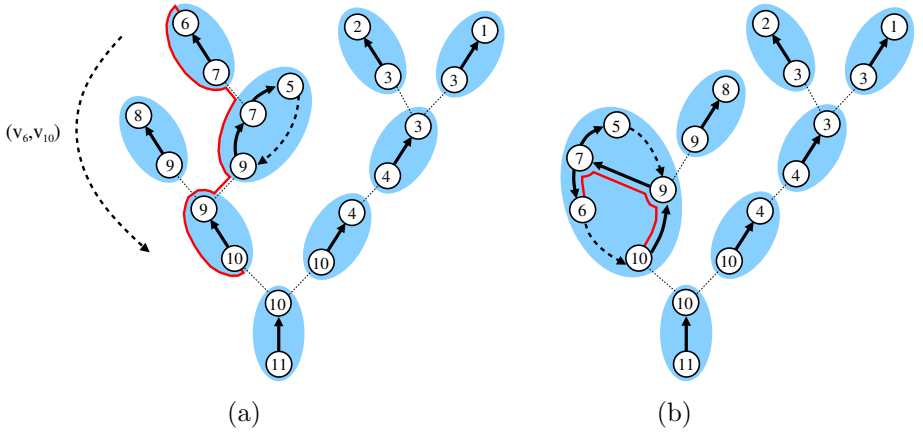


Figure 1.17 The BOYER-MYRVOLD algorithm on the DFS tree of Figure 1.16(a). (a) When vertex v_{10} is considered the back edge (v_6, v_{10}) needs to be embedded. (b) The embedding of back edge (v_6, v_{10}) corresponding to the choice of the red path from v_6 to v_{10} along the borders of blocks shown in (a).

back edges. Otherwise, if v_i has some incoming back edges, the strategy of the algorithm is that of deciding how to embed them by exploring the borders of the current blocks of G_{i-1}. To give an intuitive example Figure 1.17(a) represents G_9. At iteration 10 vertex v_{10} is considered by the algorithm and the back edge (v_6, v_{10}) needs to be embedded. In the embedding choice shown in Figure 1.17(b), the red path inside the closed face of the new block can be identified in Figure 1.17(a) as the red path going from v_6 to v_{10} along the borders of the blocks. The approach of the BOYER-MYRVOLD algorithm is that of first choosing suitable paths for the back edges returning to v_i and then using such paths to close a new block and update the data structures. Hence, each iteration has two phases: *Path searching* and *Block embedding* (in [BM99, BM04] these phases are called *Walkup* and *Walkdown*, respectively, but the tree is drawn upside down with respect to the convention used here).

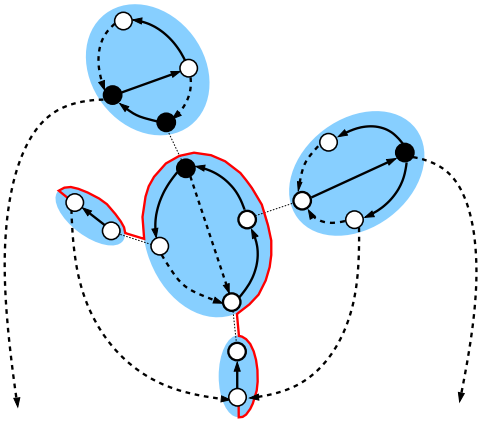


Figure 1.18 Properties of the admissible path to v_i. The lower vertex is the currently processed one while outer vertices of G_i are drawn black. The red path is not admissible, as it traverses an outer vertex of G_i (a cut vertex of G_i in this example).

Let's start from the Path searching phase. Suppose that some back edges enter v_i from vertices $u_1, u_2, \ldots, u_k$. For each j, with $j = 1, \ldots, k$, the algorithm searches for a path p_j from u_j to v_i with the following properties:

1. Vertices and edges of p_j are on the boundary of the blocks of G_{i-1}.
2. Each vertex of p_j that is the root of a block is followed by the corresponding cutvertex in the parent block, until v_i is reached.
3. Each vertex of p_j that is different from the entry point and the root of the current block is not an outer vertex of G_i (see Figure 1.18(a)).

Also, two paths (i.e., two embedding choices) may be incompatible with each other. Namely, let p_l and p_m be two paths to v_i and let b be a block b traversed by them. The following compatibility properties are enforced:

1. If p_l and p_m don't share edges of b, they do not share edges in any other block (see Figure 1.19(a)).
2. Paths p_l and p_m do not share edges of b if they traverse two other distinct outer blocks, where an *outer* block is one containing an outer vertex of G_i (see Figure 1.19(b)).
3. If p_l and p_m don't share edges of b and the root r_b of b is different from v_i, then r_b is not an outer vertex of G_i (see Figure 1.19(c)).

The above properties guarantee that when the new block is closed, no outer vertex of G_i falls inside a face of the block.

In order to be linear, the algorithm does not explicitly compute all the paths p_j, for $j = 1, \ldots, k$. In fact, if two paths share one edge, the second path can follow the same route toward v_1 used by the first one without the need of checking the above properties. Also, whenever a path enters a block b, it searches both sides of b in parallel, searching for the root r_b of b. In this way, the shorter admissible path to r_b is found by exploring at most twice the number of its edges. Since the edges used by the paths will be closed inside some face of the new block, they are never explored again in a subsequent iteration, and the total number of steps required by the algorithm for the computation of such paths is linear.

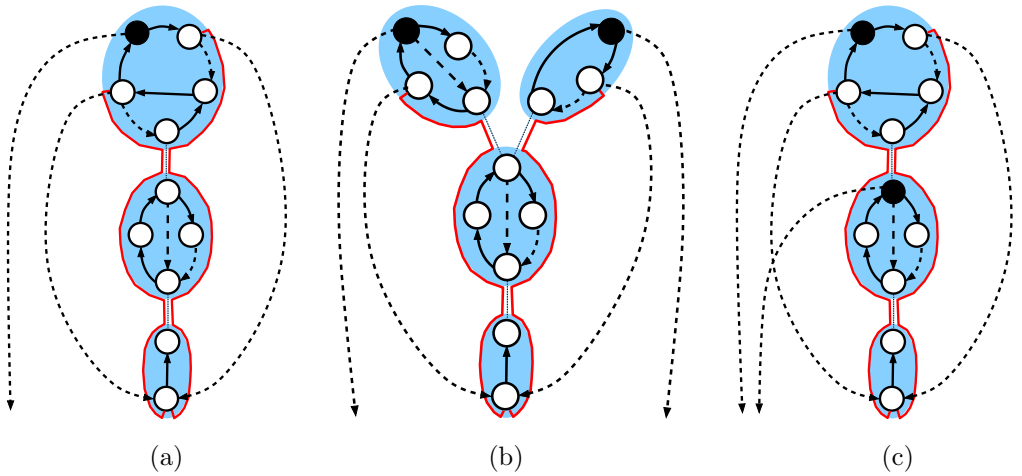


Figure 1.19 Compatibility properties of the paths to v_i. The lower vertex is the currently processed one while outer vertices of G_i are drawn black. (a) Two compatible paths not sharing edges in any block they traverse. (b) Two admissible paths coming from two distinct outer blocks. (c) Two non-admissible paths.

If the Path searching phase does not detect a non-planarity, the Block embedding phase starts. This is a simpler phase in which, starting from v_i and moving along the boundary of G_{i-1}, the blocks traversed by the paths are merged together and the back edges are embedded based on their corresponding paths to produce a planar embedding for G_i.

The simplified version of the algorithm described in [BM04] is based on the same two phases, Path searching and Block embedding. However, the check for the paths' compatibility, which in the primitive version were demanded to the first phase are moved to the second phase, which may, therefore, also detect a non-planarity.

Data Structures

The tricky point of the Boyer-Myrvold algorithm is when two blocks, traversed by a path, are merged together. It may happen that a path traverses the child block clockwise and the parent block counterclockwise, or viceversa. Fortunately, it can be shown that the properties of the paths guarantee that if one path does so all other paths comply with this embedding choice. However, in order to merge the two blocks, one of them needs to be flipped, and reversing the adjacency lists of all the vertices of the block may result in a linear-time operation that would yield a quadratic planarity algorithm. In order to solve this problem, the authors introduced a suitable data structure, called `bicomp`, that allows us to flip a block in $O(1)$-time, whatever its size. Such a data structure is based on circular lists that do not have a predefined orientation.

Namely, suppose that the list items of a circular list instead of having the usual `next` and `prev` pointers have two generic pointers `ref1` and `ref2` which could be used arbitrarily to store a reference to the next or previous list item. Suppose, also, that you maintain a reference to the last element encountered while traversing the list. If you want a reference to the next element you compare this reference with `ref1` and `ref2` and choose the one that is different from it. Hence, the circular list is traversed in the direction that is decided by the first step. If the circular order of the list has to be reversed, it suffices to begin the traversal in the opposite way.

Of course, if the clockwise direction of the adjacency list of each vertex of a block is independently chosen, this would not necessarily produce a planar embedding. However, it is not difficult to devise some convention to transfer the orientation of the adjacency list of one vertex to the adjacency lists of the neighboring vertices. For example, it may be prescribed that if in the adjacency list of vertex v_i the list item of vertex v_j uses `ref1` as `next` and `ref2` as `prev`, the same choice is made for the list item of v_i in the adjacency list of v_j (in [BM99, BM04] a less intuitive, but more practical, convention is adopted).

Hence, when two blocks are merged and their common cutvertex is identified, the two adjacency lists of the cutvertex can be suitably joined in such a way to implicitly reverse all the adjacency lists of one of the two blocks.

1.8 Frontiers in Planarity

1.8.1 Simultaneous Planarity

A recent variant of the planarity problem asks for the simultaneous embedding of two graphs on the same set of vertices V. Namely, a *simultaneous embedding* of $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ consists of two planar drawings Γ_1 and Γ_2 of G_1 and G_2, respectively, such that any vertex $v \in V$ is mapped to the same point in each of the two drawings. When Γ_1 and Γ_2 are required to be straight-line drawings, this problem is called *geometric simultaneous embedding*. When edges common to E_1 and E_2 are required to be represented by the same Jordan curve in Γ_1 and Γ_2 this problem is called *simultaneous embedding with fixed edges* (or SEFE, for short). The above definition can be easily generalized to k graphs $G_i = (V, E_i)$, with $i = 1, 2, \ldots, k$.

Geometric simultaneous embedding turns out to have limited usability, since testing whether two planar graphs admit such an embedding is NP-hard [EBGJ+07] and since a geometric simultaneous embedding does not always exist for two outerplanar graphs [BCD+07], for two trees [GKV09], and even for a tree and a path [AGKN12].

Conversely, for several classes of graphs the computation of a simultaneous embedding with fixed edges, if any, can be performed in polynomial time [EK05, DL07, Fra06, FGJ+08, JS09, ADF+10, HJL10, ADF+11], although the general problem is of unknown complexity. Refer to Chapter 11, "Simultaneous Embedding of Planar Graphs," for an in-depth exploration of this research area.

1.8.2 Clustered Planarity

The user's need of drawing some set of vertices near one to the other naturally leads to the requirement of drawing them inside the same simple closed region of the plane. This target is pursued by clustered planarity where the containment relationship among regions and vertices is described by an arbitrary hierarchy. More formally, a *clustered graph* $C(G, T)$ is a graph G and a rooted tree T whose leaves are the vertices of G. A *c-planar* drawing of $C(G, T)$ is such that G is planarly drawn and each internal node ν of T is drawn as a simple closed region $R(\nu)$ such that:

- $R(\nu)$ contains the drawing of the graph $G(\nu)$ induced by the vertices that are leaves of the subtree rooted at ν;
- $R(\nu)$ contains a region $R(\mu)$ if and only if μ is a descendant of ν in T;
- any two regions $R(\nu_1)$ and $R(\nu_2)$ do not intersect if ν_1 is not an ancestor or a descendant of ν_2; and
- an edge e does not cross the boundary of a region $R(\nu)$ more than once.

Restrictions on the c-planarity testing problem studied in the literature include: (i) assuming that each cluster induces a small number of connected components [FCE95b, FCE95a, Dah98, GJL+02, GLS05, CW06, CDF+08, JJKL08]; (ii) considering only *flat* hierarchies, where all clusters different from the root of T are children of the root [CDPP04, DF08b]; (iii) focusing on particular families of underlying graphs [CDPP04, CDPP05, JKK+08]; and (iv) fixing the embedding of the underlying graph [DF08b, JKK+08].

Although the general problem is of unknown complexity, it has been shown to be polynomial-time solvable in the following cases:

- If the subgraph $G(\nu)$ induced by each cluster ν is connected the clustered graph is called *c-connected*. The algorithm proposed in [FCE95b, FCE95a] is quadratic. Linear-time algorithms are described in [Dah98, CDF+08]. The case when each cluster induces at most two connected components has been investigated in [JJKL08].
- The results [BKM98, Bie98] on "partitioned drawings" of graphs can be interpreted as linear-time c-planarity tests for non-connected flat clustered graphs with exactly two clusters. The same result (flat clustered planarity for non-connected graphs with exactly two clusters) is shown in [HN09] where the problem is modeled as a two-page book embedding.
- Gutwenger et al. presented a polynomial-time algorithm for c-planarity testing for *almost connected* clustered graphs [GJL+02], i.e., graphs for which all nodes corresponding to the non-connected clusters lie on the same path in T starting at the root of T, or graphs in which for each non-connected cluster its parent cluster and all its siblings in T are connected.
- Cortese et al. studied the class of non-connected clustered graphs such that the underlying graph is a cycle and the clusters at the same level of T also form a cycle, where two clusters are considered adjacent if they are incident to the same edge [CDPP04, CDPP05]. The c-planarity testing and embedding problem is linear for this class of graphs.
- Goodrich et al. introduced a polynomial-time algorithm for producing planar drawings of *extrovert* clustered graphs [GLS05], i.e., graphs for which all clusters are connected or extrovert. A cluster μ with parent ν is extrovert if and only if ν is connected and each connected component of μ has a vertex with an edge that is incident to a cluster which is external to ν.
- Jelínková et al. presented a polynomial-time algorithm for testing the c-planarity of "k-rib-Eulerian" graphs [JKK+08]. A graph is *k-rib-Eulerian* if it is Eulerian and it can be obtained from a 3-connected planar graph with k vertices, for some constant k, by replacing some edges with one or more paths in parallel.

1.8.3 Decomposition-Based Planarity

Since a graph is planar if and only if its triconnected components are planar, it is somehow surprising that all known linear-time planarity algorithms require at most the biconnectivity of the input graph. It could be asked whether the triconnectivity of the graph could be leveraged in order to obtain planarity algorithms that are easier to understand and to implement. A triconnected graph has several helpful properties with this respect: if it is planar, it admits a single planar embedding up to a flip (in contrast, a biconnected graph admits an exponential number of embeddings); if it is not planar and it is different from K_5, it contains a subdivision of $K_{3,3}$.

An intriguing research line in this direction is that of exploiting construction sequences: it is well known that a triconnected graph can be reduced by means of sequences of planarity-preserving transformations to graphs as simple as a wheel [Tut61] or as a K_4 [Tut66, BG69]. Such transformations, if reversed, yield construction sequences that could be possibly exploited to find a planar embedding of the input graph starting from a planar embedding of the reduced graph. The polynomial-time planarity algorithm described in [BSW70] uses the reduction sequences described in [Tut61]. The reduction sequences described in [Tut66, BG69] have been used to give a short proof of Kuratowski's theorem [Kel81, Tho81], while their application to planarity algorithms has been only recently investigated in [Sch12].

References

[ADF+10] P. Angelini, G. Di Battista, F. Frati, V. Jelinek, J. Kratochvil, M. Patrignani, and I. Rutter. Testing planarity of partially embedded graphs. In M. Charikar, editor, *Symposium on Discrete Algorithms (SODA '10)*, pages 202–221, 2010.

[ADF+11] P. Angelini, G. Di Battista, F. Frati, M. Patrignani, and I. Rutter. Testing the simultaneous embeddability of two graphs whose intersection is a biconnected graph or a tree. In *Workshop on Combinatorial Algorithms (IWOCA '10)*, volume 6460 of *LNCS*, pages 212–225, 2011.

[ADP11] P. Angelini, G. Di Battista, and M. Patrignani. Finding a minimum-depth embedding of a planar graph in $o(n^4)$ time. *Algorithmica*, 60(4):890–937, 2011.

[AGKN12] P. Angelini, M. Geyer, M. Kaufmann, and D. Neuwirth. On a tree and a path with no geometric simultaneous embedding. *Journal of Graph Algorithms and Applications*, 16(1):37–83, 2012. Special Issue on Selected Papers from GD '10.

[AH77] K. Appel and W. Haken. Every planar map is four colourable, part I: discharging. *Illinois J. Math.*, 21:429–490, 1977.

[AHK77] K. Appel, W. Haken, and J. Koch. Every planar map is four colourable, part II: Reducibility. *Illinois Journal of Mathematics*, 21:491–567, 1977.

[AP61] L. Auslander and S. V. Parter. On imbedding graphs in the sphere. *Journal of Mathematics and Mechanics*, 10(3):517–523, 1961.

[APBL95] D. Archdeacon, C. Paul Bonnington, and C. H. C. Little. An algebraic characterization of planar graphs. *Journal of Graph Theory*, 19(2):237–250, 1995.

[AŠ98] D. Archdeacon and J. Šráň. Characterizing planarity using theta graphs. *Journal of Graph Theory*, 27(1):17–20, 1998.

[Bad64] W. Bader. Das topologische Problem der gedruckten Schaltung und seine Lösung. *Electrical Engineering (Archiv für Elektrotechnik)*, 49(1):2–12, 1964.

[Bak94] B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, 41:153–180, 1994.

[BCD+07] P. Braß, E. Cenek, C. A. Duncan, A. Efrat, C. Erten, D. Ismailescu, S. G. Kobourov, A. Lubiw, and J. S. B. Mitchell. On simultaneous planar graph embeddings. *Computational Geometry*, 36(2):117–130, 2007.

[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. In Giuseppe Liotta, editor, *Graph Drawing (Proc. GD '03)*, volume 2912 of *LNCS*, pages 25–36, 2004.

[BD91] P. Bertolazzi and G. Di Battista. On upward drawing testing of triconnected digraphs. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 272–280, 1991.

[BDBD00] P. Bertolazzi, G. Di Battista, and W. Didimo. Computing orthogonal drawings with the minimum number of bends. *IEEE Transaction on Computers*, 49:826–840, August 2000.

[BDLM94] P. Bertolazzi, G. Di Battista, G. Liotta, and C. Mannino. Upward drawings of triconnected digraphs. *Algorithmica*, 6(12):476–497, 1994.

[BDMT98] P. Bertolazzi, G. Di Battista, C. Mannino, and R. Tamassia. Optimal upward planarity testing of single-source digraphs. *SIAM J. Comput.*, 27(1):132–169, 1998.

[BFNd04] J. Boyer, C. Fernandes, A. Noma, and J. de Pina. Lempel, Even, and Cederbaum planarity method. In Celso Ribeiro and Simone Martins, editors, *Experimental and Efficient Algorithms*, volume 3059 of *LNCS*, pages 129–144. Springer, 2004.

[BG69] D. W. Barnette and B. Grünbaum. On Steinitz's theorem concerning convex 3-polytopes and on some properties of 3-connected graphs. In *Many Facets of Graph Theory*, pages 27–40, 1969.

[Bie98] T. C. Biedl. Drawing planar partitions III: Two constrained embedding problems. Technical Report RRR 13-98, RUTCOR Rutgen University, 1998.

[BKM98] T. C. Biedl, M. Kaufmann, and P. Mutzel. Drawing planar partitions II: HH-Drawings. In J. Hromkovic and O. Sýkora, editors, *Workshop on Graph-Theoretic Concepts in Computer Science (WG '98)*, volume 1517, pages 124–136. Springer, 1998.

[BL76] K. Booth and G. Lueker. Testing for the consecutive ones property interval graphs and graph planarity using PQ-tree algorithms. *J. Comput. Syst. Sci.*, 13:335–379, 1976.

[BM88] D. Bienstock and C. L. Monma. On the complexity of covering vertices by faces in a planar graph. *SIAM Journal on Computing*, 17:53–76, 1988.

[BM90] D. Bienstock and C. L. Monma. On the complexity of embedding planar graphs to minimize certain distance measures. *Algorithmica*, 5(1):93–109, 1990.

[BM99] J. Boyer and W. Myrvold. Stop minding your P's and Q's: A simplified O(n) planar embedding algorithm. In *10th Annual ACM-SIAM Symposium on Discrete Algorithms*, volume 1027 of *LNCS*, pages 140–146. Springer-Verlag, 1999.

[BM04] J. Boyer and W. Myrvold. On the cutting edge: Simplified O(n) planarity by edge addition. *Journal of Graph Algorithms and Applications*, 8(3):241–273, 2004.

[Boy05] J. Boyer. Additional PC-tree planarity conditions. In J. Pach, editor, *Graph Drawing*, volume 3383 of *LNCS*, pages 82–88. Springer, 2005.

[Bra09] U. Brandes. The left-right planarity test. Manuscript submitted for publication, 2009.

[BSW70] J. Bruno, K. Steiglitz, and L. Weinberg. A new planarity test based on 3-connectivity. *IEEE Transactions on Circuit Theory*, 17(2):197–206, 1970.

[CDF+08] P. F. Cortese, G. Di Battista, F. Frati, M. Patrignani, and M. Pizzonia. C-planarity of c-connected clustered graphs. *Journal of Graph Algorithms and Applications*, 12(2):225–262, Nov 2008.

[CDPP04] P. F. Cortese, G. Di Battista, M. Patrignani, and M. Pizzonia. Clustering cycles into cycles of clusters. In János Pach, editor, *Proc. Graph Drawing 2004 (GD '04)*, volume 3383 of *LNCS*, pages 100–110. Springer, 2004.

[CDPP05] P. F. Cortese, G. Di Battista, M. Patrignani, and M. Pizzonia. Clustering cycles into cycles of clusters. *Journal of Graph Algorithms and Applications, Special Issue on the 2004 Symposium on Graph Drawing, GD '04*, 9(3):391–413, 2005.

[Che81] C. C. Chen. On a characterization of planar graphs. *Bulletin of the Australian Mathematical Society*, 24:289–294, 1981.

[CMS08] M. Chimani, P. Mutzel, and J. M. Schmidt. Efficient extraction of multiple Kuratowski subdivisions. In Seok-Hee Hong, Takao Nishizeki, and Wu Quan, editors, *Graph Drawing (GD 2007)*, volume 4875 of *LNCS*, pages 159–170. Springer, 2008.

[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(1):54–76, 1985.

[Col90] Y. Colin de Verdière. Sur un nouvel invariant des graphes et un critère de planarité. *Journal of Combinatorial Theory, Series B*, 50(1):11–21, 1990.

[Col91] Y. Colin de Verdière. On a new graph invariant and a criterion for planarity. In Neil Robertson and Paul D. Seymour, editors, *Graph Structure Theory*, volume 147 of *Contemporary Mathematics*, pages 137–148. American Mathematical Society, 1991.

[CW06] S. Cornelsen and D. Wagner. Completely connected clustered graphs. *Journal of Discrete Algorithms*, 4(2):313–323, 2006.

[Dah98] E. Dahlhaus. Linear time algorithm to recognize clustered planar graphs and its parallelization. In Claudio L. Lucchesi and Arnaldo V. Moura, editors, *Proc. Latin American Theoretical INformatics (LATIN '98)*, volume 1380 of *LNCS*, pages 239–248. Springer, 1998.

[DBTV01] G. Di Battista, R. Tamassia, and L. Vismara. Incremental convex planarity testing. *Information Computation*, 169:94–126, August 2001.

[Deo76] N. Deo. Note on Hopcroft and Tarjan planarity algorithm. *Journal of the Association for Computing Machinery*, 23:74–75, 1976.

[DETT99] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing*. Prentice Hall, Upper Saddle River, NJ, 1999.

[dF08a] H. de Fraysseix. Trémaux trees and planarity. *Electronic Notes in Discrete Mathematics*, 31:169–180, 2008.

[DF08b] G. Di Battista and F. Frati. Efficient c-planarity testing for embedded flat clustered graphs with small faces. In Seok-Hee Hong, Takao Nishizeki, and Wu Quan, editors, *Proc. Graph Drawing 2007 (GD '07)*, volume 4875 of *LNCS*, pages 291–302. Springer, 2008.

[DKT09] Z. Dvorak, K. Kawarabayashi, and R. Thomas. Three-coloring triangle-free planar graphs in linear time. In Claire Mathieu, editor, *SODA*, pages 1176–1182. SIAM, 2009.

[DL07] E. Di Giacomo and G. Liotta. Simultaneous embedding of outerplanar graphs, paths, and cycles. *Int. J. Computational Geometry and Applications*, 17(2):139–160, 2007.

[DLT84] D. Dolev, F. T. Leighton, and H. Trickey. Planar embedding of planar graphs. In Franco P. Preparata, editor, *VLSI Theory*, volume 2 of *Adv. Comput. Res.*, pages 147–161. JAI Press, Greenwich, Conn., 1984.

[DMP64] G. Demoucron, Y. Malgrange, and R. Pertuiset. Graphes planaires: Reconnaissance et construction des représentations planaires topologiques. *Revue Fran cais de Recherche Opérationelle*, 8:33–47, 1964.

[dO96] H. de Fraysseix and P. Ossona de Mendez. Planarity and edge poset dimension. *European Journal of Combinatorics*, 17(8):731–740, 1996.

[dO02] H. de Fraysseix and P. Ossona de Mendez. P.I.G.A.L.E — Public Implementation of a Graph Algorithm Library and Editor, 2002. SourceForge project page `http://pigale.sourceforge.net/` (GPL License).

[dOR06] H. de Fraysseix, P. Ossona de Mendez, and P. Rosenstiehl. Trémaux trees and planarity. *International Journal of Foundations of Computer Science*, 17(5):1017–1029, 2006.

[dR82] H. de Fraysseix and P. Rosenstiehl. A depth-first characterization of planarity. *Annals of Discrete Mathematics*, 13:75–80, 1982.

[dR85] H. de Fraysseix and P. Rosenstiehl. A characterization of planar graphs by Trémaux orders. *Combinatorica*, 5(2):127–135, 1985.

[DT89] G. Di Battista and R. Tamassia. Incremental planarity testing. In *Proc. 30th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 436–441, 1989.

[DT96a] G. Di Battista and R. Tamassia. On-line maintenance of triconnected components with SPQR-trees. *Algorithmica*, 15:302–318, 1996.

[DT96b] G. Di Battista and R. Tamassia. On-line planarity testing. *SIAM J. Comput.*, 25:956–997, 1996.

[EBGJ+07] A. Estrella-Balderrama, E. Gassner, M. Jünger, M. Percan, M. Schaefer, and M. Schulz. Simultaneous geometric graph embeddings. In S. H. Hong, T. Nishizeki, and W. Quan, editors, *Graph Drawing (GD '07)*, volume 4875 of *LNCS*, pages 280–290, 2007.

[EK05] C. Erten and S. G. Kobourov. Simultaneous embedding of planar graphs with few bends. *Journal of Graph Algorithms and Applications*, 9(3):347–364, 2005.

[ET76] S. Even and R. E. Tarjan. Computing an st-numbering. *Theoret. Comput. Sci.*, 2:339–344, 1976.

[Eve79] S. Even. *Graph Algorithms*. Computer Science Press, Potomac, Maryland, 1979.

[FCE95a] Q. W. Feng, R. F. Cohen, and P. Eades. How to draw a planar clustered graph. In Ding-Zhu Du and Ming Li, editors, *Proc. Computing and Combinatorics (COCOON '95)*, volume 959 of *LNCS*, pages 21–30. Springer, 1995.

[FCE95b] Q. W. Feng, R. F. Cohen, and P. Eades. Planarity for clustered graphs. In *Proc. European Symposium on Algorithms (ESA '95)*, volume 979 of *LNCS*, pages 213–226. Springer, 1995.

[FGJ+08] J. J. Fowler, C. Gutwenger, M. Jünger, P. Mutzel, and M. Schulz. An SPQR-tree approach to decide special cases of simultaneous embedding with fixed edges. In I. G. Tollis and M. Patrignani, editors, *Graph Drawing (GD '08)*, volume 5417 of *LNCS*, pages 157–168, 2008.

[Fra06] F. Frati. Embedding graphs simultaneously with fixed edges. In M. Kaufmann and D. Wagner, editors, *Graph Drawing (GD '06)*, volume 4372 of *LNCS*, pages 108–113, 2006.

[GIS99] Z. Galil, G. F. Italiano, and N. Sarnak. Fully dynamic planarity testing with applications. *Journal of the Association for Computing Machinery*, 46:28–91, January 1999.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, NY, 1979.

[GJL+02] C. Gutwenger, M. Jünger, S. Leipert, P. Mutzel, M. Percan, and R. Weiskircher. Advances in C-planarity testing of clustered graphs. In Stephen G. Kobourov and Michael T. Goodrich, editors, *Proc. Graph Drawing 2002 (GD '02)*, volume 2528 of *LNCS*, pages 220–235. Springer, 2002.

[GJS76] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976.

[GKM08] C. Gutwenger, K. Klein, and P. Mutzel. Planarity testing and optimal edge insertion with embedding constraints. *Journal of Graph Algorithms and Applications*, 12(1):73–95, 2008.

[GKV09] M. Geyer, M. Kaufmann, and I. Vrt'o. Two trees which are self-intersecting when drawn simultaneously. *Discrete Mathematics*, 307(4):1909–1916, 2009.

[GLS05] M. T. Goodrich, G. S. Lueker, and J. Z. Sun. C-planarity of extrovert clustered graphs. In P. Healy and N. S. Nikolov, editors, *Proc. Graph Drawing 2005 (GD '05)*, volume 3843 of *LNCS*, pages 211–222. Springer, 2005.

[GM01] C. Gutwenger and P. Mutzel. A linear time implementation of SPQR-trees. In Joe Marks, editor, *Graph Drawing (GD 2000)*, volume 1984 of *LNCS*, pages 77–90. Springer, 2001.

[GM04] C. Gutwenger and P. Mutzel. Graph embedding with minimum depth and maximum external face. In Giuseppe Liotta, editor, *Graph Drawing*, volume 2912 of *LNCS*, pages 259–272. Springer, 2004.

[GMW01] C. Gutwenger, P. Mutzel, and R. Weiskircher. Inserting an edge into a planar graph. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete algorithms*, SODA '01, pages 246–255, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[Gol63] A. J. Goldstein. An efficient and constructive algorithm for testing whether a graph can be embedded in the plane. In John R. Edmonds, Jr., editor, *Graphs and Combinatorics Conference*, Technical Report, page 2 unn. pp. Princeton University, 1963.

[Grö59] H. Grötzsch. Ein dreifarbensatz fü dreikreisfreie netze auf der kugel. Wiss. Z. Martin-Luther-Univ. Halle-Wittenberg Math.-Natur. Reihe 8, 1959.

[GT01] A. Garg and R. Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM J. Comput.*, 31(2):601–625, 2001.

[Hal43] D. W. Hall. A note on primitive skew curves. *Bulletin of the American Mathematical Society*, 49(2):935–936, 1943.

[Har69] F. Harary. *Graph Theory*. Addison-Wesley, Reading, Mass., 1969.

[HJL10] B. Haeupler, K. R. Jampani, and A. Lubiw. Testing simultaneous planarity when the common graph is 2-connected. In *Proceedings of the 21st*

Symposium on Algorithms and Computation (ISAAC'10), volume 6507 of *LNCS*, pages 410–421. Springer Heidelberg/Berlin, 2010.

[HL96] M. D. Hutton and A. Lubiw. Upward planar drawing of single-source acyclic digraphs. *SIAM J. Comput.*, 25(2):291–311, 1996.

[HN09] S. H. Hong and H. Nagamochi. Two-page book embedding and clustered graph planarity. Technical Report 2009-004, Department of Applied Mathematics & Physics, Kyoto University, 2009.

[Hsu01] W. L. Hsu. PC-trees vs. PQ-trees. In *Proceedings of the 7th Annual International Conference on Computing and Combinatorics*, COCOON '01, pages 207–217, London, UK, 2001. Springer-Verlag.

[Hsu03] W. L. Hsu. An efficient implementation fo the PC-Tree algorithm of Shih and Hsu's planarity test. Technical Report TR-IIS-03-015, Inst. of Inf. Science, Academia Sinica, 2003.

[HT65] F. Haray and W. T. Tutte. A dual form of Kuratowski's theorem. *Canad. Math. Bull.*, 8:17–20, 1965.

[HT73] J. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM J. Comput.*, 2(3):135–158, 1973.

[HT74] J. Hopcroft and R. E. Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, 1974.

[HT08] B. Haeupler and R. E. Tarjan. Planarity algorithms via PQ-trees (extended abstract). *Electronic Notes in Discrete Mathematics*, 31:143–149, 2008.

[HW74] J. E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, pages 172–184, New York, NY, USA, 1974. ACM.

[JJKL08] V. Jelinek, E. Jelinkova, J. Kratochvil, and B. Lidicky. Clustered planarity: Embedded clustered graphs with two-component clusters. In *GD '08*, volume 5417 of *LNCS*, pages 121–132, 2008.

[JKK+08] E. Jelínková, J. Kára, J. Kratochvíl, M. Pergel, O. Suchý, and T. Vyskocil. Clustered planarity: Small clusters in Eulerian graphs. In Seok-Hee Hong, Takao Nishizeki, and Wu Quan, editors, *Proc. Graph Drawing 2007 (GD '07)*, volume 4875 of *LNCS*, pages 303–314. Springer, 2008.

[JKR11] V. Jelínek, J. Kratochvíl, and I. Rutter. A Kuratowski-type theorem for planarity of partially embedded graphs. In *Proceedings of the 27th Annual ACM symposium on Computational Geometry*, SoCG '11, pages 107–116, New York, NY, USA, 2011. ACM.

[JS09] M. Jünger and M. Schulz. Intersection graphs in simultaneous embedding with fixed edges. *Journal of Graph Algorithms and Applications*, 13(2):205–218, 2009.

[Kam07] F. Kammer. Determining the smallest k such that g is k -outerplanar. In L. Arge, M. Hoffmann, and E. Welzl, editors, *ESA '07*, volume 4698 of *LNCS*, pages 359–370, 2007.

[Kel81] A. K. Kelmans. A new planarity criterion for 3-connected graphs. *Journal of Graph Theory*, 5:259–267, 1981.

[Kel93] A. K. Kelmans. Graph planarity and related topics. In Neil Robertson and Paul Seymour, editors, *Graph Structure Theory, Proceedings of the*

AMS-IMS-SIAM Joint Summer Research Conference on Graph Minors, 1991, volume 147 of *Contemporary Mathematics*, pages 635–667, 1993.

[KR88] P. N. Klein and J. H. Reif. An efficient parallel algorithm for planarity. *J. Comput. Syst. Sci.*, 37(2):190–246, 1988.

[Kur30] K. Kuratowski. Sur le problème des courbes gauches en topologie. *Fund. Math.*, 15:271–283, 1930.

[LEC67] A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. In *Theory of Graphs: Internat. Symposium (Rome 1966)*, pages 215–232, New York, 1967. Gordon and Breach.

[LH77] C. H. C. Little and D. A. Holton. A new characterization of planar graphs. *Bulletin of the American Mathematical Society*, 83(1):137–138, 1977.

[Lie01] A. Liebers. Planarizing graphs – a survey and annotated bibliography. *Journal of Graph Algorithms and Applications*, 5(1):1–74, 2001.

[Liu88] Y. Liu. A new approach to the linearity of testing planarity of graphs. *Acta Mathematicae Applicatae Sinica (English Series)*, 4(3):257–265, 1988.

[Liu89] Y. Liu. Boolean approach to planar embeddings of a graph. *Acta Mathematica Sinica (New Series)*, 5(1):64–79, 1989.

[LS10] C. H. C. Little and G. Sanjith. Another characterisation of planar graphs. *The Electronic Journal of Combinatorics*, 17(15), 2010.

[LT79] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math.*, 36:177–189, 1979.

[Mac37a] S. MacLane. A combinatorial condition for planar graphs. *Fundamenta Mathematicae*, 28:22–32, 1937.

[Mac37b] S. MacLane. A structural characterization of planar combinatorial graphs. *Duke Mathematical Journal*, 3:466–472, 1937.

[Man83] A. Mansfield. Determining the thickness of graphs is NP-hard. *Proc. Math. Cambridge Philos. Soc.*, 93:9–23, 1983.

[Men27] Karl Menger. Zur allgemeinen kurventheorie. *Fund. Math.*, 10:96–115, 1927.

[MM96] K. Mehlhorn and P. Mutzel. On the embedding phase of the Hopcroft and Tarjan planarity testing algorithm. *Algorithmica*, 16:233–242, 1996.

[MW99] P. Mutzel and R. Weiskircher. Optimizing over all combinatorial embeddings of a planar graph. In *Proceedings of the 7th International IPCO Conference on Integer Programming and Combinatorial Optimization*, pages 361–376, London, UK, 1999. Springer-Verlag.

[MW00] P. Mutzel and R. Weiskircher. Computing optimal embeddings for planar graphs. In *Proceedings of the 6th Annual International Conference on Computing and Combinatorics*, COCOON '00, pages 95–104, London, UK, 2000. Springer-Verlag.

[Pap95] A. Papakostas. Upward planarity testing of outerplanar dags. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes Comput. Sci.*, pages 298–306. Springer-Verlag, 1995.

[Piz05] M. Pizzonia. Minimum depth graph embeddings and quality of the drawings: An experimental analysis. In P. Healy and N. S. Nikolov, editors, *Graph Drawing '05*, volume 3843 of *LNCS*, pages 397–408, 2005.

[PT00] M. Pizzonia and R. Tamassia. Minimum depth graph embedding. In M. Paterson, editor, *ESA '00*, volume 1879 of *LNCS*, pages 356–367, 2000.

[RND77] E. M. Reingold, J. Nievergelt, and N. Deo. *Combinatorial Algorithms: Theory and Practice*. Prentice Hall, Englewood Cliffs, NJ, 1977.

[Ros80] P. Rosenstiehl. Preuve algébrique du critère de planarité du Wu-Liu. *Annals of Discrete Mathematics*, 9:67–78, 1980.

[RR89] V. Ramachandran and J. H. Reif. An optimal parallel algorithm for graph planarity. In *Proc. 30th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 282–293, 1989.

[RR94] V. Ramachandran and J. Reif. Planarity testing in parallel. *Journal of Computer and System Sciences*, 49:517–561, December 1994.

[RS84] N. Robertson and P. D. Seymour. Graph minors. III. Planar tree-width. *Journal on Combinatorial Theory, Series B*, 36(1):49–64, 1984.

[RSST97] N. Robertson, D. P. Sanders, P. D. Seymour, and R. Thomas. The four color theorem. *J. Combin. Theory Ser. B*, 70:2–4, 1997.

[Sch89] W. Schnyder. Planar graphs and poset dimension. *Order*, 5:323–343, 1989.

[Sch12] J. M. Schmidt. A planarity test via construction sequences. *CoRR*, abs/1202.5003, 2012.

[SH93] W. K. Shih and W. L. Hsu. A simple test for planar graphs. In *Int. Workshop on Discrete Math. and Algorithms*, pages 110–122, 1993.

[SH99] W. K. Shih and W. L. Hsu. A new planarity test. *Theor. Comp. Sci.*, 223, 1999.

[Tam98] R. Tamassia. Constraints in graph drawing algorithms. *Constraints*, 3:87–120, April 1998.

[Tar72] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.

[Tho81] C. Thomassen. Kuratowski's theorem. *Journal of Graph Theory*, 5(3):225–241, 1981.

[Tho99] R. Thomas. Graph planarity and related topics. In Jan Kratochvíl, editor, *Graph Drawing (Proc. GD '99)*, volume 1731 of *LNCS*, pages 137–144. Springer-Verlag, 1999.

[TT97] Hisao Tamaki and Takeshi Tokuyama. A characterization of planar graphs by pseudo-line arrangements. In *Proc. 8th Annu. Internat. Sympos. Algorithms Comput.*, volume 1350 of *Lecture Notes Comput. Sci.*, pages 123–132. Springer-Verlag, 1997.

[Tut61] W. T. Tutte. A theory of 3-connected graphs. *Indag. Math.*, 23:441–455, 1961.

[Tut66] W. T. Tutte. *Connectivity in Graphs*. University of Toronto Press, 1966.

[Wag37a] K. Wagner. Über eine Eigenschaft der ebenen Komplexe. *Mathematische Annalen*, 114:570–590, 1937.

[Wag37b] K. Wagner. Über eine Erweiterung eines Satzes von Kuratowski. *Deutsche Mathematik*, 2:280–285, 1937.

[Whi32] H. Whitney. Non-separable and planar graphs. *Transactions of the American Mathematical Society*, 34:339–362, 1932.

[Wil80] S. G. Williamson. Embedding graphs in the plane – algorithmic aspects. *Annals of Discrete Mathematics*, 6:349–384, 1980.

[Wu74] W. Wu. Planar embedding of linear graphs. *Kexue Tongbao*, 2:226–282, 1974. (In Chinese).

[Xu89] W. Xu. Improved algorithm for planarity testing based on Wu-Liu's criterion. *Annals of the New York Academy of Science*, 576:641–652, 1989.

[Yan78] M. Yannakakis. Node-and edge-deletion NP-complete problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, pages 253–264, New York, NY, USA, 1978. ACM.

[Yan82] M. Yannakakis. The complexity of the partial order dimension problem. *SIAM J. Algebraic Discrete Methods*, 3(3):351–358, 1982.

2

Crossings and Planarization

Christoph Buchheim
TU Dortmund

Markus Chimani
Friedrich-Schiller-Universität Jena

Carsten Gutwenger
TU Dortmund

Michael Jünger
University of Cologne

Petra Mutzel
TU Dortmund

2.1 Introduction

In many respects, crossing minimization is an exceptional problem in the wide range of optimization problems arising in automatic graph drawing. First of all, it is one of the most basic and natural problems among these, and, at the same time, very easy to formulate: given a graph, draw it in the plane with a minimum number of crossings between its edges. In fact, this problem is much older than automatic graph drawing. Crossing number problems were first examined by Turán when he worked in a brick factory during the Second World War. This work motivated him to search for crossing minimal drawings of the complete bipartite graph $K_{n,m}$, without success. Later, Zarankiewicz gave a rule for creating a drawing of $K_{n,m}$ with $\lfloor \frac{m}{2} \rfloor \lfloor \frac{m-1}{2} \rfloor \lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor$ crossings, but his proof of optimality was shown to be incorrect. Still today, this is an open question. The same is true for the crossing number of K_n.

Besides its theoretical relevance as a topological problem, crossing minimization has many practical applications. In automatic graph drawing, it is well known that the readability of a two-dimensional graph layout strongly depends on the number of edge crossings. This was verified by empirical studies of Purchase [Pur97]. In fact, the main information given by an abstract graph is whether two vertices are connected by an edge. This information should be easily recognizable. In particular, it should be easily possible to trace the edges in the drawing. This task is complicated by the presence of crossing edges, as they distract the concentration of the human viewer. See Figure 2.1 for a comparison.

Another important application for the crossing minimization problem is VLSI (very large scale integration) design. In this context, the problem was first discussed in depth. The

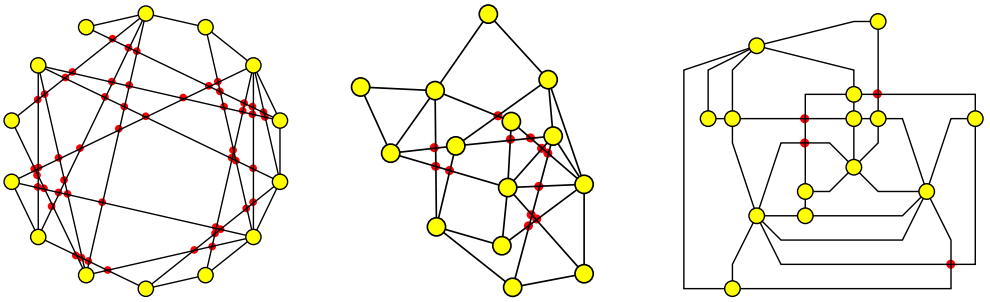


Figure 2.1 Different drawings of the same abstract graph with different numbers of edge crossings (51, 12, and 4, respectively). Most aesthetic criteria like few edge bends, uniform edge lengths, or a small drawing area favor the first two drawings. However, with respect to the number of edge crossings the last drawing is preferable.

aim of VLSI design is to arrange transistors on two-dimensional chips. Certain transistors need to be connected by wires, which have to be routed on the chip. Every crossing of two wires causes additional costs for realizing the chip, so that a small number of such crossings is desired to reduce these costs as far as possible.

An outstanding property of the crossing minimization problem is its hardness. It was shown by Garey and Johnson [GJ83] that this problem is NP-hard; see Section 2.3. However, several optimization problems arising in the area of automatic graph drawing are NP-hard and have nevertheless been solved in practice. In contrast, crossing minimization is extremely hard also practically. So far, exact approaches can only solve relatively sparse, medium sized instances within a reasonable running time; see Section 2.4. This is drastically shown by the fact that even the crossing numbers of the complete graphs K_n are unknown for $n \geq 13$.

Given the NP-hardness of the general problem, many restricted versions of crossing minimization have been considered, in the hope of finding polynomial time algorithms in these cases. However, in most cases, the problem remains NP-hard. Examples are bipartite drawings or linear embeddings; see Section 2.3. In practice, however, some of the resulting problems become easier as the degrees of freedom are reduced.

Besides considering special cases, it is natural to ask for approximation algorithms. However, up to now, only for graphs with bounded degree it was possible to find algorithms yielding provably near-optimal solutions; see Section 2.6. On the other hand, no negative results about approximability are known.

Currently used approaches to the general crossing minimization problem are of heuristic nature. The state-of-the-art approach for general crossing minimization is the planarization method, which is described in detail in Section 2.5. The main idea is to split up the problem into two steps: in the first step, a planar subgraph is computed. The aim in this step is to find a subgraph with as many edges as possible. In the second step, all edges not contained in this subgraph are reinserted into the drawing. Whenever an edge is inserted, the produced crossings are replaced by dummy vertices, so that the result is a planar graph again. Having added all edges in this way, a planar drawing algorithm can be used to compute a layout of the graph; see Chapters 6 and 7. After this, the dummy vertices are removed. For both steps of the planarization approach, a variety of possible algorithmic realizations has been discussed; see Section 2.5. This approach is also particularly interesting with respect to approximation algorithms, as it can be shown that certain insertion algorithms in fact approximate the crossing number in case of special graph classes; again see Section 2.6.

The second step of the planarization approach can also be realized in many different ways; see Section 2.5.3. Usually, it is again solved heuristically; edges are reinserted one after another, each with a minimal number of new edge crossings. It was shown recently that one can add a single edge optimally over all possible embeddings of the planar graph constructed so far.

After 60 years of research in different areas of mathematics and computer science, the crossing minimization problem is still far from being fully explored, both theoretically and practically. On the theoretical side, the most interesting open problems in our opinion are the crossing numbers of the complete graphs, including Turán's brick factory problem, as well as the approximability of crossing minimization. Practically, one can hope for new and better heuristic methods or faster exact approaches. At this point, we can only report on the status quo. We hope that parts of this chapter will become obsolete sooner or later.

2.2 Crossing Numbers

A drawing of a graph $G = (V, E)$ in the plane is a mapping of each vertex $v \in V$ to a distinct point and each edge $e = (v, w) \in E$ to a curve connecting the incident vertices v and w without passing through any other vertex. A common point of two edges in a drawing that is not an incident vertex is called a *crossing*. The *crossing number* $\mathrm{cr}(G)$ is defined to be the minimum number of crossings in any drawing of G.

In their paper "Which Crossing Number Is It, Anyway?", Pach and Tóth define two further possibilities on how to count the number of crossings in a graph (see [PT00]).

DEFINITION 2.1 Let $G = (V, E)$ be a simple graph.

1. The *pairwise crossing number* of G, denoted with $\mathrm{pcr}(G)$, is the minimum number of pairs of edges $(e_1, e_2) \in E \times E,\ e_1 \neq e_2$ such that e_1 and e_2 determine at least one crossing, over all drawings of G.
2. The *odd-crossing number* of G, denoted with $\mathrm{ocr}(G)$, is the minimum number of pairs of edges $(e_1, e_2) \in E \times E,\ e_1 \neq e_2$ such that e_1 and e_2 cross an odd number of times, over all drawings of G.

It is clear that $\mathrm{ocr}(G) \leq \mathrm{pcr}(G) \leq \mathrm{cr}(G)$, and we know that $\mathrm{cr}(G)$ cannot be arbitrarily large if $\mathrm{ocr}(G)$ is bounded. More precisely we have that $\mathrm{cr}(G) \leq 2(\mathrm{ocr}(G))^2$. Only after some years, an example was conceived by Pelsmajer et al. [PSŠ06], showing that there in fact exist graphs with $\mathrm{ocr}(G) \neq \mathrm{cr}(G)$. Yet, it is still unknown whether $\mathrm{pcr}(G) = \mathrm{cr}(G)$.

Another well-studied variant of the crossing minimization problem is the *rectilinear crossing number* $\mathrm{cr}_1(G)$, which is defined to be the minimum number of crossings in any drawing of a graph G where all edges are drawn as straight lines. Bienstock and Dean proved in [BD93] that for graphs with crossing number at most three, the rectilinear crossing number and the usual crossing number coincide. They could further show that there are graphs G_k such that $\mathrm{cr}_1(G_k)$ is arbitrarily large, even if $\mathrm{cr}(G_k)$ is only four.

As a generalization of the rectilinear crossing number, Bienstock introduced in [Bie91] the concept of the *t-polygonal crossing number*.

DEFINITION 2.2 Let $G = (V, E)$ be a graph. A t-polygonal drawing of G, for $t \geq 1$, is a good drawing where every edge is drawn as a t-polygonal line, i.e., a polygonal line with

at most t segments. The *t-polygonal crossing number* $\mathrm{cr}_t(G)$ is defined as the minimum number of crossings in any t-polygonal drawing of G.

A *good drawing* is a drawing that satisfies the following conditions:

1. no edge crosses itself
2. adjacent edges do not cross one another
3. non-adjacent edges cross each other at most once

Bienstock also showed that there cannot be a polynomial time algorithm for producing optimal t-polygonal drawings of G unless P = NP and that there is no fixed t such that $\mathrm{cr}(G) = \mathrm{cr}_t(G)$ for any graph G.

An even more restricted version of the crossing number problem is the *linear crossing number*: We call a drawing of a graph a *linear drawing* if all vertices lie on a straight line and edges are drawn as semicircles above and below this line. It is easy to see that the crossing number resulting from this drawing style is an upper bound for $\mathrm{cr}(G)$. Surprisingly, there is a further connection to the general crossing number problem, as was shown by Nicholson [Nic68]. He proved that any drawing in the plane with a minimum number of crossings can be converted into a linear drawing with an equivalent crossing structure such that all vertices are placed on a horizontal line and edges are drawn as a series of semicircles while successive semicircles lie on different sides of the horizontal line.

It is interesting to see that the complexity of the linear crossing number problem stays the same, even if we fix the ordering of the vertices of V (this is the so-called *fixed linear crossing minimization problem*). Masuda et al. proved in [MNKF90] that even this variant is NP-complete.

2.2.1 Known Bounds

No matter which definition or variant of the crossing number problem is used, its solution seems to be a difficult task. Even though crossing numbers have been investigated extensively in the past, useful theoretical results are rather limited. One of the first major results has been claimed in 1953 by Zarankiewicz (and, independently, by Urbaník) as a solution to Turán's brick factory problem, which in fact asks for the crossing number of the complete bipartite graph $K_{m,n}$.

$$\mathrm{cr}(K_{m,n}) = \lfloor \frac{m}{2} \rfloor \lfloor \frac{m-1}{2} \rfloor \lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor \qquad \text{(conjecture)} \tag{2.1}$$

Over ten years later, an error in the induction argument of Zarankiewicz's proof was unveiled, which is still unremedied. Hence, the correctness of equation (2.1) is still unknown. The conjecture is derived from the following drawing rule for complete bipartite graphs $K_{m,n} = (A \cup B, E)$: place the vertices in vertex set A at coordinates $(i(-1)^i, 0)$ for all $i = 1, \ldots, m$ and the vertices of vertex set B at coordinates $(0, j(-1)^j)$ for all $j = 1, \ldots, n$. All edges are drawn as straight lines. Figure 2.2 shows a sample drawing of $K_{6,6}$ with 36 crossings. Even though the correctness of equation (2.1) could never be verified, the provided drawing rule gives us an upper bound $Z(m,n)$ for $\mathrm{cr}(K_{m,n})$. Recently, de Klerk et al. [KMP+06, KPS07] devised a method for computing asymptotic lower bounds for $\mathrm{cr}(K_{m,n})$ based on semidefinite programming. They show that

$$\lim_{n\to\infty} \frac{\mathrm{cr}(K_{m,n})}{Z(m,n)} \geq 0.8594 \frac{m}{m-1} .$$

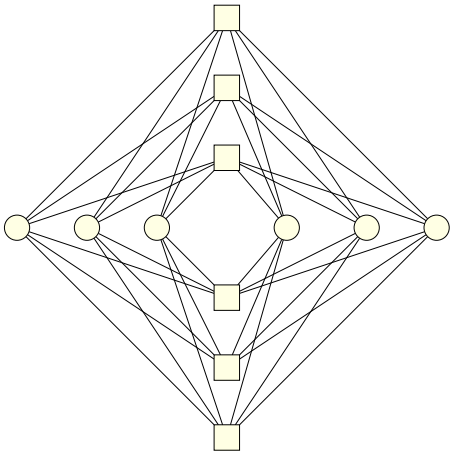


Figure 2.2 A drawing of $K_{6,6}$ with 36 crossings using Zarankiewicz's rule.

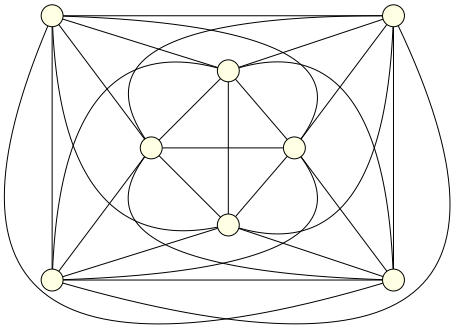


Figure 2.3 A drawing of K_8 with a minimum number of 18 crossings.

As for complete bipartite graphs, there is also a conjecture for the number of crossings of the complete graph K_n with n vertices.

$$\operatorname{cr}(K_n) = \frac{1}{4}\lfloor\frac{n}{2}\rfloor\lfloor\frac{n-1}{2}\rfloor\lfloor\frac{n-2}{2}\rfloor\lfloor\frac{n-3}{2}\rfloor \qquad \text{(conjecture)} \tag{2.2}$$

Constructions of corresponding drawings [GJJ68] show that also this conjecture yields an upper bound $Z(n)$ on $\operatorname{cr}(K_n)$. For complete graphs on up to ten vertices, its correctness has been verified by Guy [Guy72]. Pan and Richter [PR07] have extended this verification to K_{11} and K_{12}. We show a sample drawing of K_8 with a minimum number of 18 crossings in Figure 2.3. For K_n, the best-known asymptotic lower bound is again due to de Klerk et al. [KMP+06]:

$$\lim_{n\to\infty} \frac{\operatorname{cr}(K_n)}{Z(n)} \geq 0.83.$$

The crossing number of a graph G with n vertices cannot exceed the crossing number of the complete graph K_n, hence $Z(n)$ also marks an upper bound for general graphs. Unfortunately, it is the only known upper bound. A simple *lower bound* can be obtained from Euler's formula. Since any planar simple connected graph $G = (V, E)$ cannot have more than $3|V| - 6$ edges, clearly $\operatorname{cr}(G) \geq |E| - 3|V| + 6$. If, in addition, G contains no triangle, then $\operatorname{cr}(G) \geq |E| - 2|V| + 4$.

In 1983, Leighton used induction on the number of vertices to show the following theorem; see [Lei83].

Theorem 2.1 *Let $G = (V, E)$ be a simple graph. If $|E| \geq 4|V|$, we have*

$$cr(G) \geq \frac{1}{100} \frac{|E|^3}{|V|^2} . \tag{2.3}$$

Ajtai et al. obtained the same result independently with a smaller constant of $\frac{1}{375}$ in [ACNS82]. One of the best-known results has been derived by Pach and Tóth [PT97]. For any simple graph $G = (V, E)$, $\mathrm{cr}(G)$ satisfies

$$\mathrm{cr}(G) \geq \frac{1}{33.75} \frac{|E|^3}{|V|^2} - 0.9|V| . \tag{2.4}$$

Apart from bounds with respect to the number of vertices and edges, several approaches to obtain tight lower bounds based on different graph properties can be found in the literature.

A simple example is the *skewness* $\mathrm{sk}(G)$ of a graph G. It is defined as the minimum number of edges that must be removed from G in order to obtain a planar subgraph. Clearly, the crossing number of a graph cannot be smaller than its skewness. Hence, we have that

$$\mathrm{cr}(G) \geq \mathrm{sk}(G) . \tag{2.5}$$

Cimikowski showed in [Cim92] that there is a family of graphs with skewness one, but an arbitrarily high crossing number. An example is shown in Figure 2.4. Computing the skewness is equivalent to the *maximum planar subgraph problem*, which was shown to be NP-hard by Liu and Geldmacher in [LG77] in general. For certain classes of graphs, i.e., complete and complete bipartite graphs, the skewness is known. We can derive it for K_n from Euler's formula and the observation that every maximal planar graph is also a maximum planar subgraph of K_n. Hence, the skewness for complete graphs K_n is given by

$$\mathrm{sk}(K_n) = \frac{n(n-1)}{2} - 3n + 6, \tag{2.6}$$

and we can use similar arguments to derive the skewness for complete bipartite graphs as

$$\mathrm{sk}(K_{m,n}) = mn - 2(m+n) + 4 . \tag{2.7}$$

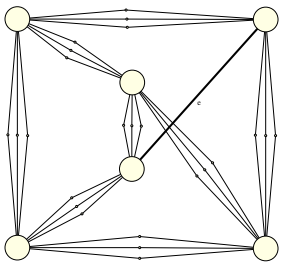


Figure 2.4 Construction of graphs with skewness one and arbitrarily high crossing number.

Another bound can be obtained from the *bisection width* $\mathrm{bw}(G)$. For any disjoint partition of the vertex set V into sets V_1 and V_2, we denote the edges (v_1, v_2) with $v_1 \in V_1$ and $v_2 \in V_2$ by $E(V_1, V_2)$. The bisection width $\mathrm{bw}(G)$ is defined as follows:

$$\mathrm{bw}(G) = \min_{|V_1|,|V_2| \geq \frac{|V|}{3}} \{|E(V_1, V_2)|\}.$$

More intuitively, the bisection width is the minimum number of edges that must be removed from G in order to partition the graph into two separate components with nearly equal size. The first known bound for the crossing number based on the bisection width goes back to Leighton. He proved the following theorem [Lei84].

Theorem 2.2 *For any graph $G = (V, E)$ of bounded degree, we have*

$$\mathrm{cr}(G) + |V| = \Omega(\mathrm{bw}(G)^2) \; .$$

Pach, Shahrokhi, and Szegedy [PSS96] use the bisection width to show the following, more general, result, which can be used to derive a lower bound for $\mathrm{cr}(G)$.

Theorem 2.3 *Let $G = (V, E)$ be a simple graph with $|V| \geq 2$ vertices, and let $k \geq 1$ be an integer. If G has a drawing with at most k crossings, then*

$$|E| \leq 3|V|(10 \log_2 |V|)^{2k-2} \; .$$

A very similar parameter is the *cutwidth* $\mathrm{cw}(G)$. Let $\phi : V \to \{1, 2, \ldots, |V|\}$ be an injection. We define $\mathrm{cw}(G)$ as follows:

$$\mathrm{cw}(G) = \min_{\phi} \max_{i} |\{(u, v) \in E : \phi(u) \leq i \leq \phi(v)\}|.$$

As a graphical interpretation, consider an injection of the vertices to the horizontal line and draw edges on one side of this line using semicircles. For each injection, we can "cut" the horizontal line between a pair of consecutive vertices such that the number of edges between each of the segments is maximized. The minimum value over all possible injections is the cutwidth. So far, the following relations are known (see [DV02], [PSS96], [SV94]; here $\delta(v)$ denotes the set of neighbors of vertex v):

$$\mathrm{cr}(G) + \frac{1}{16} \sum_{v \in V} |\delta(v)|^2 \geq \frac{1}{40} \mathrm{bw}^2(G), \tag{2.8}$$

$$\mathrm{cr}(G) + \frac{1}{16} \sum_{v \in V} |\delta(v)|^2 \geq \frac{1}{1176} \mathrm{cw}^2(G). \tag{2.9}$$

Unfortunately, the computation of both parameters $\mathrm{bw}(G)$ and $\mathrm{cw}(G)$ is NP-hard.

Both the bisection width and the cutwidth can be seen as a measure for the "non-planarity" of a graph. This applies also to the *thickness* $\Theta(G)$, which is defined as the minimum number of planar graphs whose union forms G. The only families of graphs whose thickness is known are complete graphs, complete bipartite graphs, and hypercubes. Mansfield proved in [Man83] that the determination of $\Theta(G)$ is NP-hard in general. There is a simple connection between thickness and crossing number:

$$\Theta(G) \leq \mathrm{cr}(G) + 1$$

So far, all those bounds are only of limited use. Either their quality is poor or their computation often exceeds practical limits. The investigation of tighter bounds could help to improve practical applications and lead to more insight into the crossing minimization problem.

2.3 Complexity of Crossing Minimization

Crossing minimization is not only one of the most important problems arising in automatic graph drawing, it is also one of the hardest. This is true both in practice and in theory: until recently, not a single exact algorithm being able to solve instances of nontrivial size had been devised. In fact, even for a graph as small and regular as K_{13}, the minimal number of crossings is still unknown. For a discussion of exact crossing minimization approaches, see Section 2.4.

2.3.1 NP-hardness

On the theoretical side, it is a well-known fact that the general crossing minimization problem is NP-hard. More precisely, consider the following *crossing number problem*:

> Given a graph G and a nonnegative integer K, decide whether there is a drawing of G with at most K edge crossings.

In 1983, Garey and Johnson proved that this problem is NP-complete [GJ83]. In the following, we reproduce their proof. It is based on a transformation of the NP-complete *optimal linear arrangement problem*:

> Given a graph $G = (V, E)$ and a nonnegative integer K, decide whether there is a one-to-one function $f: V \to \{1, \ldots, |V|\}$ with $\sum_{(v,w)\in E} |f(v) - f(w)| \leq K$.

The corresponding optimization problem is thus to order the vertices of G such that the total length of edges is minimal, where the length of an edge is defined as the distance of the two adjacent vertices in this ordering.

As an intermediate step in the proof, Garey and Johnson show the NP-completeness of the bipartite version of the crossing number problem for multigraphs, the *bipartite crossing number problem*:

> Given a bipartite multigraph $G = (V_1, V_2, E)$ and a nonnegative integer K, decide whether there is a drawing of G inside the unit square such that all vertices of V_1 lie on the northern boundary, all vertices of V_2 lie on the southern boundary, and the number of edge crossings is at most K.

In the following, we will call such drawings *bipartite drawings* for short. It is interesting that, contrary to widespread belief, the NP-completeness of the bipartite crossing number problem for *simple* graphs was long open—it was shown only very recently [Sch12].

Theorem 2.4 *The crossing number problem is NP-complete.*

It is easy to see that this problem is in NP: for every edge of G, one can guess all crossings involving this edge, and their order along the edge. To answer the question whether such a guessed crossing configuration is feasible, one can place dummy vertices on all chosen crossings and test the resulting graph for planarity. Clearly, the result is positive if and only if the given crossing configuration can be realized by some drawing of G.

The proof of completeness consists of several reduction steps and is split up into three separate lemmas in the following.

LEMMA 2.1 The optimal linear arrangement problem can be reduced to the bipartite crossing number problem in polynomial time.

Proof: The rough idea of the reduction is as follows: every vertex is doubled and the linear ordering is modeled on two parallel layers (the northern and southern boundary of the unit square) at the same time, with edges leading from one layer to the other. By a large number of artificial edges connecting corresponding pairs of vertices, the ordering is forced to be the same on both layers. The distance between two adjacent vertices in the linear ordering problem is then essentially proportional to the number of artificial edges crossed.

More formally, the transformation is defined as follows: Let an instance for the optimal linear arrangement problem be given, consisting of a graph G and an integer K, and assume $V = \{v_1, \ldots, v_n\}$. We then construct an instance $G' = (V_1, V_2, E_1 \cup E_2)$ and K' of the bipartite crossing number problem as follows:

$$\begin{aligned} V_1 &= \{u_i \mid i = 1, \ldots, n\} \\ V_2 &= \{w_i \mid i = 1, \ldots, n\} \\ E_1 &= \{|E|^2 \text{ copies of } (u_i, w_i) \mid i = 1, \ldots, n\} \\ E_2 &= \{(u_i, w_j) \mid (v_i, v_j) \in E \text{ with } i < j\} \\ K' &= |E|^2(K - |E|) + |E|^2 - 1 \end{aligned}$$

This construction is obviously polynomial. We have to show that the graph G admits a linear ordering with total edge length at most K if and only if the bipartite multigraph G' admits a bipartite drawing with at most K' crossings.

If a linear ordering f of G with total edge length at most K exists, we construct a bipartite drawing as follows. We place vertex u_i on position $(f(v_i)/(n+1), 1)$ and vertex w_i on position $(f(v_i)/(n+1), 0)$. Furthermore, we draw all edges as straight lines; bundles of parallel edges are drawn as nearly straight lines without mutual crossings; see Figure 2.5 for an example.

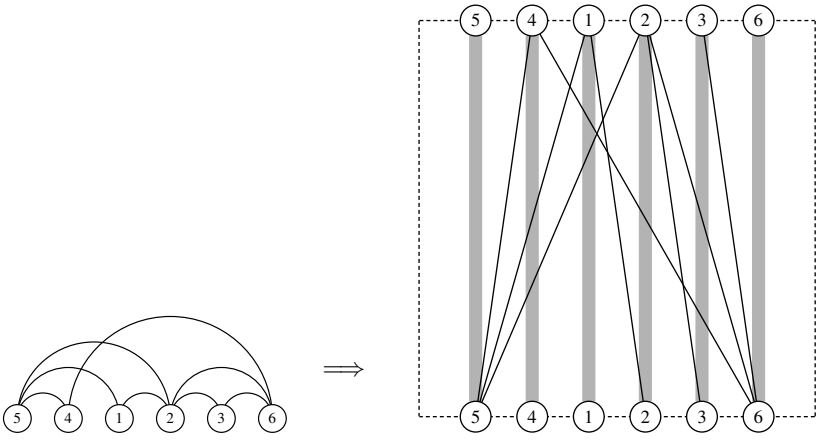


Figure 2.5 Reducing the optimal linear arrangement problem to the bipartite crossing number problem. Bold grey lines represent bundles of $|E|^2$ edges each.

In the constructed drawing, no artificial edge from E_1 will cross any other edge. Moreover, an edge $(u_i, w_j) \in E_2$ crosses exactly $|f(v_i) - f(v_j)| - 1$ bundles of $|E|^2$ edges each.

Consequently, the total number of such crossings is

$$\sum_{(u_i,w_j)\in E_2} |E|^2\Big(|f(v_i)-f(v_j)|-1\Big) = |E|^2 \sum_{(v,w)\in E}\Big(|f(v)-f(w)|-1\Big) \le |E|^2(K-|E|)\ .$$

The remaining crossings in the constructed drawing can only occur between pairs of edges in E_2, so that their total number is at most $|E|^2-1$. Summing up, the total number of edge crossings in our drawing is at most $|E|^2(K-|E|)+|E|^2-1=K'$.

For showing the other direction, assume that a bipartite drawing of G' with at most K' crossings is given. Then define $f(v_i)$ as the position of vertex u_i in the order of vertices on the northern boundary of the unit square. We claim that the linear ordering f leads to a total edge length of at most K. To see this, first observe that the order of vertices on both boundaries must be the same, as otherwise two bundles of $|E|^2$ edges each would cross each other, leading to $|E|^4 > K'$ crossings. Because of that, for each edge (v_i,v_j) with $i<j$, the distance $|f(v_i)-f(v_j)|$ is at most one more than the number of crossings of (u_i,w_j) with any edge bundle, so that

$$\sum_{(v,w)\in E} |f(v)-f(w)| \le |E| + K'/|E|^2 = |E| + (K-|E|) + 1 - 1/|E|^2 < K+1\ .$$

As the left-hand side of this inequality is integer, it is at most K. □

LEMMA 2.2 The bipartite crossing number problem can be reduced to the general crossing number problem for multigraphs in polynomial time.

Proof: Let $G=(V_1,V_2,E)$ and K be an instance of the bipartite crossing number problem. We construct a multigraph G' as follows: we add two vertices u and w to G. Moreover, we connect u with all vertices of V_1 by $K+1$ edges each. Analogously, we connect w with all vertices of V_2 by $K+1$ edges each. Finally, we add $K+1$ edges connecting u and w. Now we claim that G has a bipartite drawing with at most K crossings if and only if G' has a general drawing with at most K crossings.

The basic idea of this construction is that w.l.o.g. no bundle of $K+1$ edges will be crossed by any other edge, and that by this the crossing minimal bipartite drawings of G correspond to the crossing minimal general drawings of G'. In particular, it is clear that a bipartite drawing of G with at most K crossings yields a drawing of G' with at most K crossings by placing the vertices u and w outside the unit square; see Figure 2.6.

For showing the other direction, a drawing of G' with at most K crossings has to be converted into a bipartite drawing of G with at most K crossings. For this, a sequence of so-called normalization steps is applied in order to transform the original drawing of G' into one of the type of Figure 2.6 without increasing the number of edge crossings; deleting the vertices u and w then yields the desired drawing of G. This part of the proof was the most technical one in the original presentation; we give a simplified version here.

In the first normalization step, multiple crossings between one pair of edges and crossings between edges incident to a common vertex are removed in the obvious way. In particular, every bundle of $K+1$ edges connecting the same pair of vertices now defines a sequence of K regions in the drawing.

In a second step, one can obtain a drawing such that none of these bundle regions contains any vertex of G' or is crossed by any edge of G'. Indeed, for a fixed $v\in V_1$, consider the edge e connecting u and v that in the current drawing has the minimum number of crossings

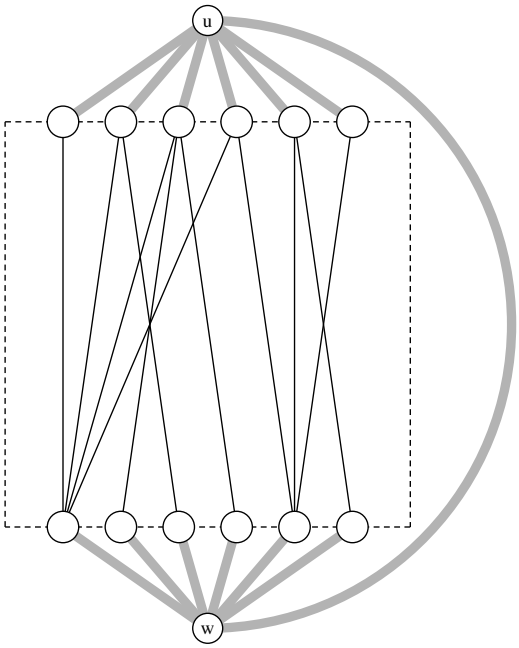


Figure 2.6 Reducing the bipartite crossing number problem to the general crossing number problem for multigraphs. Bold grey lines represent bundles of $K + 1$ edges each.

with other edges. Then one can reroute all edges (u, v), i.e., all edges parallel to e, along the same route as e. This yields a new drawing of G' with at most as many edge crossings as before. Repeating this for every $v \in V_1$ and analogously for w and every $v \in V_2$, we get a drawing without vertices in the bundle regions. Now it follows that no edge can cross any of these regions. The reason is that such an edge would have to cross all $K + 1$ edges of a bundle, as no vertices are contained in the bundle regions and multiple crossings were eliminated in the first normalization step.

Clearly, the drawing resulting from these two normalization steps is topologically equivalent to one of the type displayed in Figure 2.6. □

LEMMA 2.3 The crossing number problem for multigraphs can be reduced to the crossing number problem for simple graphs.

Proof: For the given multigraph, place an artificial vertex in the middle of every edge. The result is a simple graph with the same crossing number as the original multigraph. □

In the above scheme, we observe that the graph for which deciding on the crossing number is NP-hard requires two distinct vertices u and v of very high degree. One may think that they are central to the construction. Yet, using a different reduction strategy from optimal linear arrangement, Hliněný showed in [Hli06]:

Theorem 2.5 *The crossing number problem remains NP-complete even when restricted to cubic graphs, i.e., graphs where every vertex has degree 3.*

Theorem 2.4 shows that the crossing number problem is NP-complete. In particular, the crossing minimization problem is NP-hard, i.e., the problem of constructing a drawing of

a given graph with a minimal number of edge crossings. Nevertheless, one might hope for polynomial time algorithms at least for special classes of graphs, or for situations where the class of allowed drawings is restricted.

However, no interesting special class of graphs is known for which crossing minimization can be done in polynomial time. Exceptions are the classes of graphs for which a constant bound c on the number of crossings is given a priori, see Section 2.3.2, but this is a purely theoretical result in that this bound is not at hand in general and the running time increases heavily with the constant c.

The results also remain mostly negative if we restrict the set of feasible drawings by additional conditions. For instance, the problem is still NP-hard (even for simple graphs) if we require that

- the drawing is bipartite and the vertex order on one of the layers is fixed [EW94].
- all vertices have the same vertical coordinate and edges are drawn as semicircles. This is the so-called *linear crossing minimization problem* [MKNF86]. This problem remains NP-hard even if the horizontal order of vertices is fixed [MNKF90].
- the vertices lie on the unit circle and edges are drawn as straight lines. This is the *circular crossing minimization problem* [MKNF87].

However, we would like to point out that practically the problem might become considerably easier with the degrees of freedom for the drawing decreasing. To give an example, the bipartite crossing minimization problem with one layer fixed is NP-hard but can be solved quickly in practice [JM97]. By now, also reasonably sized general multi-layer crossing minimization instances can be tackled effectively with integer linear and semidefinite programs; see [CHJM11] for an overview.

2.3.2 Fixed Parameter Tractability

In the last section, we reproduced a proof by Garey and Johnson showing that it is NP-hard to decide whether a given graph G can be drawn with at most K edge crossings. We can also consider the situation where K is not given as part of the input but as a fixed parameter. It is then easy to see that one can decide in polynomial time whether a drawing of G with at most K crossings exists: broadly speaking, one could check all possible configurations of the up to K crossings, replace the chosen crossings by dummy vertices, and check the resulting graph for planarity. We can answer the original question affirmatively if and only if we find any planar graph in this way.

Even if the above algorithm runs in polynomial time for fixed K, the obvious drawback is the strong increase in running time for increasing K: if implemented in the straightforward way, the runtime is $O(|V| \cdot |E|^{2K})$. For a long time, it was an open question whether the problem is *fixed-parameter tractable*, i.e., whether the problem can be solved in $O(f(K) \cdot |V|^c)$ running time for some function $f(K)$ that is independent from the instance and some constant c that is independent from K. This question was answered by Grohe in 2001 with $c = 2$ [Gro01]. However, the running time of Grohe's algorithm is $O(2^{2^{p(K)}}|V|^2)$, where p is a polynomial, and hence grows strongly with K. Thus, the relevance of this algorithm is rather theoretical than practical. Kawarabayashi and Reed [KR07] improved on this result by giving a linear algorithm, i.e., $c = 1$; yet $f(K)$ remains too large for any practical application.

2.4 Exact Crossing Minimization

Exact methods to solve the crossing minimization problem constitute the youngest research field we are discussing in this chapter. The development showcases various algorithm engineering aspects of algorithm development, as its iterative improvements were always based on the analysis of the bottlenecks of the earlier approaches. The first approach [BEJ+05] already lay the setting used in the subsequent developments: it relies on mathematical programming in combination with branch-and-cut. Yet, its applicability was limited to very small graphs. By introducing column generation schemes into the branch-and-cut framework, its central ILP model, which we will describe in detail below, was later brought into the realm of applicability [CGM09, BCE+08] to some real-world graphs. The currently best exact approach replaces a key concept (the so-called *simple* crossing number) of the first formulation by integrating multiple linear-ordering problems instead [CMB08]. This leads to a mathematically more complex model but offers the advantage of fewer variables on the one hand, and the possibility for even stronger column generation strategies, on the other hand. Together with other developments like strong upper bounds (cf. Section 2.5), preprocessing strategies like the *non-planar core reduction* [CG09], and efficient extraction of multiple Kuratowski subdivisions (see below) at once [CMS08], we are now in the position to compute the exact crossing number of sparse graphs with up to 100 vertices. Figure 2.4 gives an overview of the algorithmic progress over the last years, comparing the various algorithms on the way to the currently most successful one. For a more detailed description of all exact algorithms discussed in the following, see [Chi08].

A *linear program* (LP) is an optimization problem consisting of continuous variables, a linear objective function, and linear constraints. The "father" of linear programming, George B. Dantzig, proposed the following standard model:

$$\begin{gathered} \text{maximize } c^\top x \\ \text{subject to } Ax \leq b \\ x \geq 0 \end{gathered}$$

where $c \in \mathbf{R}^n$, $A \in \mathbf{R}^{m \times n}$, and $b \in \mathbf{R}^m$. The linear function $c^\top x : \mathbf{R}^n \to \mathbf{R}$ is called the *objective function* and the inequalities in the system $Ax \leq b$ are called *constraints*. A vector $\hat{x}$ that satisfies the system of inequalities $Ax \leq b$ is called a *feasible* solution of the LP. Moreover, $\hat{x}$ is called an *optimal solution* if $c^\top \hat{x} \geq c^\top x'$ for all feasible solutions x'.

Linear programs proved to provide a powerful tool for various optimization problems in the past and extensive research led to efficient algorithms able to solve them in polynomial time, *e.g.*, the simplex [Chv83], the ellipsoid [GLS88], and the interior point method [RT97]. However, additional constraints that require some or all of the variables to be integer, render the problem NP-complete in general [GJ79].

Anyway, *(mixed) integer linear programs* are widely used to solve NP-hard combinatorial optimization problems in conjunction with polyhedral combinatorics, which aims at describing combinatorial optimization problems as linear programs and solving these with special-purpose methods. A key feature therefore is the possibility to alternatively describe the convex hull of the feasible points and extreme rays of a problem by a system of linear inequalities. For an introduction into this field, the interested reader is referred to [Pul89].

Before introducing the ideas of the ILP formulation presented in this section, we have to mention *Kuratowski's theorem*, which is one of the most important results in the field of planarity testing providing a full characterization of planar graphs based on the complete graph K_5 and the complete bipartite graph $K_{3,3}$.

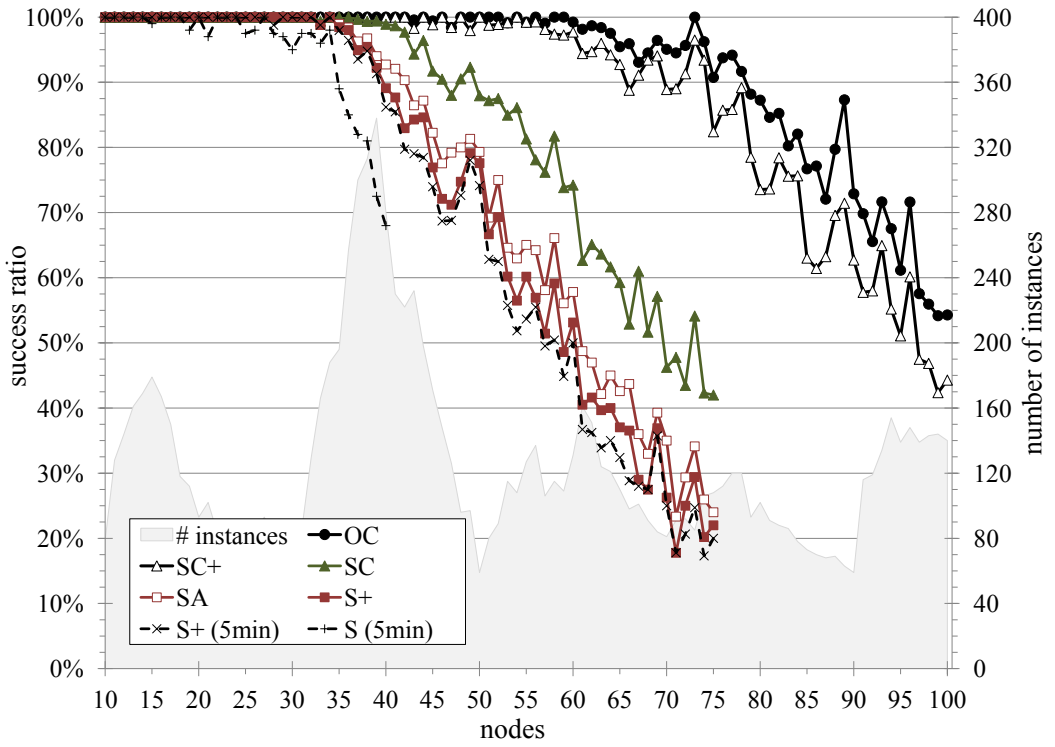


Figure 2.7 Success ratio (i.e., percentage of instances solved to provable optimality) in dependency to the graphs' size (number of vertices). Benchmark set: *Rome library* [DGL+97], see also Section 2.5. Different lines give different development steps of the algorithms; each instance was given 30 minutes of computation time unless specified otherwise. **S** is the first implementation of [BEJ+05], **S+** a more efficient reimplementation of the same algorithm. **SA** and **SC** denote the subdivision-based algorithms as considered in [CGM09], with algebraic pricing and the combinatorial column generation scheme, respectively. Finally, **SC+** and **OC** denote the latest implementations of the subdivision-based and the ordering-based ILPs, respectively, with combinatorial column generation and all further described improvements, as presented in [CMB08].

Theorem 2.6 *A finite graph is planar if and only if it contains no subgraph that is a subdivision of K_5 or $K_{3,3}$.*

We can obtain a subdivision S of a graph G by repeatedly replacing edges e by a path of length two. As a consequence of Theorem 2.6, at least two edges belonging to each Kuratowski subdivision have to cross. Based on this observation, we can try to address the crossing minimization problem using mathematical programming in the following way: we introduce a zero-one decision variable $x_{e,f}$ for each pair of edges $(e, f) \in E \times E$ that encode the crossings in an associated drawing: edges e and f cross each other if and only if $x_{e,f} = 1$. For each subdivision of K_5 or $K_{3,3}$, we can add constraints that force at least one of the involved variables to one.

Mutzel and Jünger [MJ01] pointed out the problems with this formulation. To our knowledge, there is no known polynomial time separation algorithm to identify the constraints of this type that are violated by a given fractional solution. Moreover, those constraints are not strong enough since it is not guaranteed that there is a realizing drawing if at least one of the involved crossing variables is one in every Kuratowski subdivision. Another severe problem of this formulation is the NP-hardness of the *realizability problem* [Kra91]:

> Given a vector $x \in \{0,1\}^{\binom{E}{2}}$, does there exist a drawing consistent with x?

In order to efficiently answer this question, we also need to know the *order* of the edge crossings for a particular edge e. This additional information can be exploited by the introduction of a dummy vertex for each crossing and the application of a linear-time planarity testing algorithm to test the existence of a realizing drawing in polynomial time. Despite all these drawbacks, it is interesting that under certain conditions the above-described constraints, as well as similar ones, in fact constitute *facets* of the polytope defined by the convex hull of the feasible solutions [Chi11].

2.4.1 Subdivision-Based Formulation

One way to work around the realizability problem is the reduction to *simple drawings*. We call a drawing simple if each edge crosses at most one other edge. As for planar graphs, we can find a bound for the maximum number of edges of graphs that admit a simple drawing. More precisely, Pach and Tóth show the following theorem [PT97]:

Theorem 2.7 *Let $G = (V, E)$ be a simple graph drawn in the plane so that every edge is crossed by at most k others. If $0 \leq k \leq 4$, then we have*

$$|E| \leq (k+3)(|V|-2)\ . \tag{2.10}$$

They could further prove that this bound cannot be improved for $0 \leq k \leq 2$ and that for *any* $k \geq 1$ the following inequality holds:

$$|E| \leq \sqrt{16.875k}|V| \approx 4.108\sqrt{k}|V|\ . \tag{2.11}$$

Furthermore, Bodlaender and Grigoriev prove in [BG04] that it is NP-complete to determine whether there is a simple drawing for a given graph G. If there is such a drawing, we denote the minimum number of crossings among all simple drawings of G by $\mathrm{crs}(G)$.

It is easy to see that $\mathrm{cr}(G) \leq \mathrm{crs}(G)$. We cannot state equality because there are graphs G such that $\mathrm{crs}(G) > \mathrm{cr}(G)$. Consider the sample graph in Figure 2.8. The left drawing shows an optimal drawing with two crossings while the right drawing shows an optimal drawing among all simple drawings with three crossings.

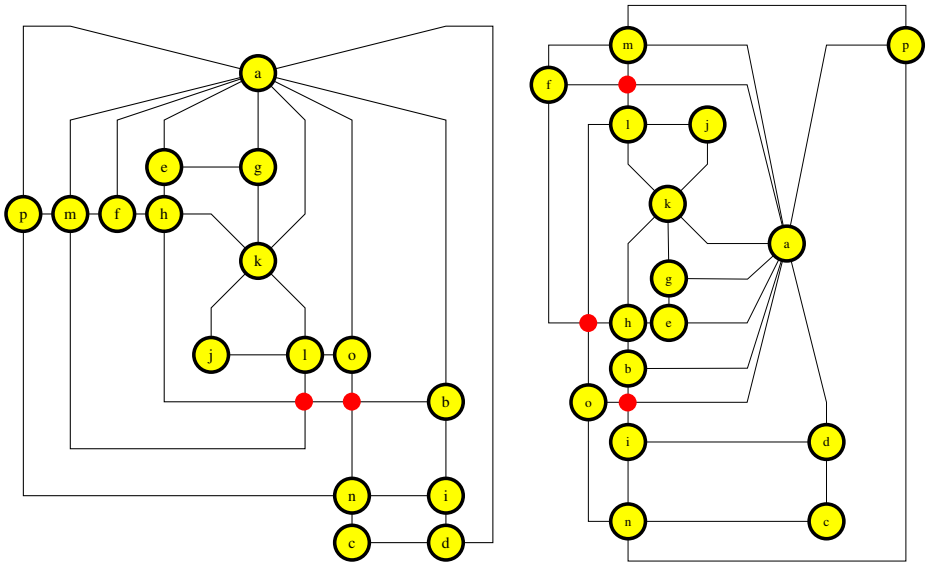


Figure 2.8 An optimal drawing of a graph with two crossings (left) and an optimal simple drawing of the same graph with three crossings (right). Both drawings were produced with the exact algorithm presented in this section.

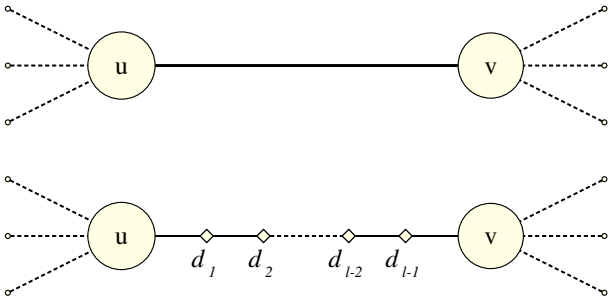


Figure 2.9 Edges are replaced with a path of length ℓ by inserting $\ell - 1$ dummy vertices.

Given an integer ℓ and a graph $G = (V, E)$ such that $\ell \geq |E|$, we can create a graph $G^* = (V^*, E^*)$ by replacing every edge $e \in E$ with a path of length ℓ. Figure 2.9 shows an example that illustrates this transformation. The graph G^* contains a total number of $|V| + (\ell - 1)|E|$ vertices and $\ell|E|$ edges.

It is easy to show that G can be drawn with n crossings if and only if there is a *simple drawing* of G^* with n crossings. Therefore, it is "sufficient" to solve the crossing minimization problem restricted to simple drawings in order to solve the "general" crossing minimization problem. Since the transformation obviously can be done in polynomial time, the NP-completeness of the corresponding decision problem for simple drawings follows immediately from the proof by Garey and Johnson; see Section 2.3. Since an edge $e = (u, v)$ never crosses itself or an adjacent edge in an optimal drawing it is sufficient to replace e with a path of length $|E| - |\delta(u)| - |\delta(v)| - 1$.

Let $G = (V, E)$ be a graph and let $D \subseteq E \times E$ be a set of unordered pairs of edges. We call D *realizable* if there is a drawing of G such that there is a crossing between edges e and f if and only if $(e, f) \in D$. Furthermore, D is called *simple* if for every $e \in E$ there is at most one $f \in E$ such that $(e, f) \in D$.

For every graph G and every simple D, G_D denotes the graph that is obtained by introducing a dummy vertex $d_{e,f}$ for each pair of edges $(e, f) \in D$: in other words, $d_{e,f}$ is the unique vertex when identifying the two vertices arising from subdividing both e and f. Note that G_D is only well defined if D is simple. For both edges e_1 and e_2 resulting from splitting e, we set $\hat{e}_1 = \hat{e}_2 = e$, analogously for f. Given a subgraph $H = (V', E') \subseteq G_D$, we denote with $\hat{H} \subseteq E$ the subset of original edges of G involved in the subgraph H of G_D, i.e., $\hat{H} = \{\hat{e} \mid e \in E'\} \subseteq E$. In the following, H will usually be a Kuratowski subdivision. We call the path corresponding to a single edge of the underlying K_5 or $K_{3,3}$ a *Kuratowski path*. By $\hat{H}^{[2]} \subset \hat{H}^2$ we will then denote the edge pairs (e, f) where e and f belong to different, nonadjacent Kuratowski paths. Hence, these are the only edge pairs that may actually form a crossing meaningful to the Kuratowski subdivision H.

COROLLARY 2.1 Let D be simple. Then D is realizable if and only if G_D is planar.

Using a linear-time planarity testing algorithm, we can test in time $O(|V| + |D|)$ whether D is realizable, and compute a realizing drawing if so.

DEFINITION 2.3 For a set of pairs of edges $D \subseteq E \times E$, we define

$$x^D_{e,f} = \begin{cases} 1 & \text{if } (e, f) \in D \\ 0 & \text{otherwise.} \end{cases}$$

PROPOSITION 2.1 Let D be simple and realizable. For an arbitrary set of pairs of edges $D' \subseteq E \times E$ of $G = (V, E)$ and any subdivision H of K_5 or $K_{3,3}$ in $G_{D'}$ the following inequality holds:

$$C_{D',H} : \sum_{(e,f) \in \hat{H}^{[2]} \setminus D'} x^D_{ef} \geq 1 - \sum_{(e,f) \in \hat{H}^{[2]} \cap D'} (1 - x^D_{ef}) \, . \tag{2.12}$$

Proof: Suppose inequality (2.12) is violated. Since every $x^D_{e,f} \in \{0, 1\}$, the left-hand side of the inequality must be zero and the right-hand side must be one, which means that

$$\begin{aligned} x^D_{e,f} &= 0 \quad \text{for all } (e, f) \in \hat{H}^{[2]} \setminus D', \text{ and} \\ x^D_{e,f} &= 1 \quad \text{for all } (e, f) \in \hat{H}^{[2]} \cap D' \, . \end{aligned}$$

It follows from the definition of x^D that $\hat{H}^{[2]} \cap D' = \hat{H}^{[2]} \cap D$, in other words, G_D agrees to $G_{D'}$ on the subgraph induced by $\hat{H}$, so that H is also a forbidden subgraph in G_D, i.e., a subdivision of K_5 or $K_{3,3}$. It follows from Kuratowski's Theorem that G_D is not planar. This contradicts the realizability of D by Corollary 2.1. □

Theorem 2.8 *Let G=(V,E) be a simple graph. A set of pairs of edges $D \subseteq E \times E$ is simple and realizable if and only if the following set of conditions holds:*

$$\begin{array}{ll} x^D_{e,f} \in \{0, 1\} & \forall\ e, f \in E,\ e \neq f \\ \sum_{f \in E} x^D_{e,f} \leq 1 & \forall\ e \in E \\ C_{D',H} & \textit{for every simple } D' \subseteq E \times E \\ & \textit{and every forbidden subgraph } H \textit{ in } G_{D'} \end{array}$$

Proof: It is easy to see that the constraints from the second row are satisfied if and only if D is simple. It remains to show that a simple D is realizable if and only if the conditions $C_{D',H}$ from the last row hold. For a realizable D, every $C_{D',H}$ is satisfied according to Proposition 2.1.

We have to show that any set of pairs of edges D that is not realizable violates at least one of the constraints $C_{D',H}$. It follows from Corollary 2.1 that G_D is not planar if D is not realizable and we know from Theorem 2.6 that there exists a subdivision H of K_5 or $K_{3,3}$ in G_D. Let $D' = D$ and consider the constraint $C_{D,H}$:

$$C_{D,H}: \sum_{(e,f)\in \hat{H}^{[2]}\setminus D} x^D_{ef} \;\geq\; 1 - \sum_{(e,f)\in \hat{H}^{[2]}\cap D} (1 - x^D_{ef}) \tag{2.13}$$

It follows from the definition of x^D that every $x^D_{e,f} \in \hat{H}^{[2]} \setminus D$ is zero, hence the left-hand side of inequality (2.13) is also zero. Since $\hat{H}^{[2]} \cap D \subseteq D$ we also know that $\sum_{(e,f)\in \hat{H}^{[2]}\cap D}(1 - x^D_{ef})$ is zero and the right-hand side of $C_{D,H}$ is one. Thus, $C_{D,H}$ is violated. □

Since we can compute a corresponding drawing for a simple and realizable D in polynomial time, we can reformulate the crossing minimization problem for simple drawings as

> Given a graph $G = (V, E)$, find a simple realizable subset $D \subseteq E \times E$ of minimum cardinality.

This leads to the following ILP-Formulation. We use $x(F)$ as an abbreviation for the term $\sum_{(e,f)\in F} x_{e,f}$.

minimize $x(E \times E)$

subject to

$\sum_{f\in E} x_{e,f} \leq 1$	$\forall\ e \in E$
$x(\hat{H}^{[2]} \setminus D') - x(\hat{H}^{[2]} \cap D') \geq 1 - \lvert\hat{H}^{[2]} \cap D'\rvert$	for every simple D' and every forbidden subgraph H in $G_{D'}$
$x_{e,f} \in \{0, 1\}$	$\forall\ e, f \in E$

Given a simple set of crossings D we can easily check if D is realizable by applying a planarity testing algorithm to G_D. If the answer is "no" we also get a forbidden subdivision H of G_D and we can separate an additional constraint $C_{D,H}$ according to the proof of Theorem 2.8 that excludes D.

2.4.2 Ordering-Based Formulation

The above subdivision-based formulation requires up to $\Omega(|E|^4)$ variables, as every edge may have to be subdivided into $\Omega(|E|)$ segments. The currently best-performing ILP model avoids this subdivision and instead considers *linear ordering problems* on each edge. Recall that the reason for the graph extension was to be able to model the order of the crossings, in order to obtain a tractable realizability problem. The *ordering-based* ILP formulation achieves this by explicitly computing an ordering of the edge crossings.

Consider an arbitrary, fixed *orientation* of the given graph G, i.e., for each undirected edge we decide on one of the two possible directions. As in the original (problematic)

approach, we introduce $\Omega(|E|^2)$ many binary variables $x_{e,f}$, one for each edge pair e, f, which should be 1 if the two indexed edges cross. The objective function is simply the sum over all these variables. We then introduce binary variables $y_{e,f,g} \in \{0,1\}$ for all edge triples e, f, g. This results in only $\Omega(|E|^3)$ additional variables. A variable $y_{e,f,g}$ should be 1 if and only if the edge e is crossed by both edges f and g, and the crossing with f occurs prior to the crossing with g, w.r.t. the fixed edge orientation. Conceptually, the variables $y_{e,\cdot,\cdot}$, when properly bound to their corresponding x variables, then form the variables of a linear-ordering problem with the additional property that some elements need not to be ordered at all. We can achieve this via

$$\begin{aligned} x_{e,f} &\geq y_{e,f,g} \\ x_{e,g} &\geq y_{e,f,g} \\ 1 + y_{e,f,g} + y_{e,g,f} &\geq x_{e,f} + x_{e,g} \\ y_{e,f,g} + y_{e,g,f} &\leq 1 \\ y_{e,f,g} + y_{e,g,h} + y_{e,h,f} &\leq 2 \end{aligned}$$

over the suitable edge indices. The first two constraints guarantee that the x variables (counting the crossing in the objective function) are set whenever a corresponding y variable is set. The third constraint ensures that whenever there are two crossings occurring on the same edge (here: on e), their relative order has to be specified. Then, we have to ensure in the fourth constraint that this order is unique. The last constraint, known as a *3-cycle constraint*, ensures that the order given by the y variables is in fact a linear, i.e., acyclic, order.

Using this setup it remains to introduce Kuratowski constraints much like the ones described for the subdivision-based formulation. Recall that D' in the subdivision-based formulation described a simple set of edge crossings. Similarly, we now consider a (technically more involved) *crossing shadow* $(\mathcal{X}, \mathcal{Y})$ instead. It can be thought of as a minimal description of a not-necessarily realizable crossing situation. I.e., $\mathcal{X}$ ($\mathcal{Y}$) lists x variables (y variables, respectively) that should be set. The "minimality" of this description is achieved by avoiding to list an x variable if a corresponding y variable in $\mathcal{Y}$ already induces that it has to be set. Similarly, we use the transitivity property of a linear ordering, and, e.g., do not include the variable $y_{e,f,h}$ if $y_{e,f,g}$ and $y_{e,g,h}$ are already in $\mathcal{Y}$. For a concise definition, we refer the reader to [CMB08]. Considering all possible crossing shadows $(\mathcal{X}, \mathcal{Y})$ and all thereby induced Kuratowski subdivisions H, we can require:

$$x(\hat{H}^{[2]}) \geq 1 - \sum_{x' \in \mathcal{X}} (1 - x') - \sum_{y' \in \mathcal{Y}} (1 - y')$$

2.4.3 Branch-and-Cut-and-Prize

For a practical implementation, we can omit variables in some cases. The graph G can be split up into its blocks first, which can be solved separately. The crossing number of G is equal to the sum of the crossing numbers of its blocks. Furthermore, it is easy to show that adjacent edges do not cross in an optimal drawing and no edge crosses itself, i.e., we can restrict ourselves to good drawings. Furthermore, we may apply more sophisticated preprocessing strategies like the *non-planar core reduction* [CG09], which further shrinks the graph based on its triconnectivity structure. Thereby, it may be necessary to introduce integer weights $w : E \to \mathbb{N}$ on the edges. A crossing between the edges e and f should then be counted as $w(e) \cdot w(f)$, which is easily achievable in both above ILP formulations by using these products as the coefficient for the respective x variables.

```
L := {initial problem}                    {L denotes the list of unsolved problems}
repeat
  Choose a subproblem Π ∈ L and set L := L \ {Π}
  repeat
    Let x̂ be an optimal solution for the linear relaxation of Π
    if x̂ is not feasible for Π then
      Separate violated inequalities and add them to the LP
    end if
  until no more violated inequalities can be found
  if no feasible solution for Π could be found then
    Split Π into subproblems and add them to L
  end if
until L = ∅
Print the best found feasible solution
```

Figure 2.10 An overview of the branch-and-cut approach.

Because of the exponential number of constraints, we cannot create them in advance and solve the ILP in a single step. A well-suited method for this class of ILPs is the *branch-and-cut approach*. The basic structure of a branch-and-cut based algorithm is outlined in Figure 2.10. The referred linear relaxation of Π can be easily obtained by dropping the integrality constraints, i.e., variables are allowed to be fractional.

In the case of zero-one integer linear programs, the set of unsolved subproblems L is organized as a binary tree, called the *branch-and-bound tree*. Each subproblem corresponds to a node in the tree and the list of unsolved problems L is represented by its leaves. If we need to split a problem Π into subproblems, we choose a fractional *branching variable* and create two new subproblems by setting the branching variable to zero and one, respectively.

Whenever we split a problem into two subproblems by setting the branching variable to zero and one, respectively, we can compute a local lower bound. This is the best value for the objective function that can be obtained subject to the assignments of values for the branching variables up to the root node. If this value is greater than the global upper bound, we can discard all descendants of the current subproblem since they can never improve the current feasible solution.

A severe problem of this approach is the *separation problem*: "Given a class of valid inequalities and a vector $z \in \mathbf{R}^n$, either prove that z satisfies all inequalities in the class, or find an inequality which is violated by z." Although we can easily separate violated inequalities for integral solution vectors according to the proof of Theorem 2.8, the problem becomes severe within the branch-and-cut framework since we have to deal with fractional values.

This problem can be solved heuristically by rounding variables to either zero or one, but one cannot guarantee that there is no violated inequality if the graph realizing the crossing D or $(\mathcal{X}, \mathcal{Y})$ is planar. In this case, we have to select a branching variable and split the current problem into two subproblems by setting the branching variable to 0 and 1, respectively.

The major bottleneck when following this approach then remains the large number of variables, rendering both approaches useless as such. Yet the concept of *column generation* turns out to allow drastic speed-ups of the algorithms. Conceptually, and somewhat similar to the separation approach, we start with a small subset of the variables. After solving the LP relaxation, we not only have to solve the separation problem ("is our solution too good

because some constraints are missing?"), but also the pricing problem: "Is our solution too bad because some variables are missing?" Observe the difference in the obtained bounds when constraints or variables are missing, which, in general, leads to weaker bounding strategies than applicable to pure branch-and-cut approaches.

Following the traditional approach based on the Dantzig-Wolfe decomposition [DW60], we can solve the pricing problem in a purely algebraic way by computing the reduced costs of the variables not already in the model and adding them based on their sign. It turns out that this approach, denoted by *algebraic pricing*, already speeds up the computation, but we can do much better.

In a *combinatorial column generation scheme*, we refrain from computing reduced costs, but try to incorporate our problem-specific knowledge to obtain more efficient strategies. In particular, our special-purpose generation schemes allow us to overcome the aforementioned bounding problem, and retain the fact that the LP-relaxation always gives a lower bound to the problem, even when constraints and variables are missing.

We start with the observation that in most practical applications, most of the edges will not be crossed at all or are only involved in one crossing. On these edges, we do not have any ambiguity with the order of crossings, and the realizability problem is easy. The central idea for the combinatorial column generation scheme for both formulations can be roughly described as such: we start without any special constructions to avoid crossing-order ambiguities, i.e., we do not subdivide the edges for the subdivision-based formulation and do not introduce any y variables for the ordering-based formulation. Recall that the crossing order only becomes crucial when considering Kuratowski constraints; these are only generated via separation on a rounded solution. So, we use the branch-and-cut framework as outlined above. Whenever the separation routine considers a rounded solution where two or more edges cross the same edge, and their order is hence ambiguous, we introduce the necessary variables and constraints from the original model which are necessary to decide this order. Then, the LP relaxation is recomputed. We refrain from discussing the relatively technical details of which variables or subdivisions are necessary, and refer to [CGM09] and [CMB08] for the two formulations instead. The interesting part is that adding variables in such a way will never decrease the objective function.

Overall, the currently most efficient approach from the practical point of view is the ordering-based formulation, together with its combinatorial column generation scheme, the aforementioned preprocessing strategies and upper bounds obtained via the strong planarization heuristic that we will discuss in the following section. Furthermore, the separation routine is improved by not looking for single Kuratowski subdivisions in the rounded solution, but by applying an algorithm that obtains several such subdivisions in one pass requiring only linear time in input and output size [CMS08, CMS07]. This allows to solve sparse real-world graphs with up to 100 vertices to provable optimality within reasonable time bounds on average hardware; cf. Figure 2.4. Yet the subdivision-based formulation allows extensions to other crossing number concepts where a pair of edges crosses multiple times, e.g., the simultaneous crossing number [CJS08].

2.5 The Planarization Method

2.5.1 Overview

The most prominent and practically successful method for solving the crossing minimization problem heuristically is the *planarization approach*. This approach was introduced by Batini, Talamo, and Tamassia in [BTT84] and can be viewed as a general framework that addresses the problem with a two-step strategy. Each step aims at solving a particular

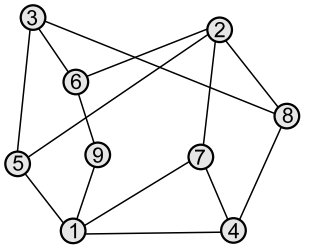
optimization problem for which various solution methods are possible. Let $G = (V, E)$ be the graph for which we want to find a crossing minimal drawing. Then, the two steps to be executed are:

1. Compute a planar subgraph $P = (V, E_p)$ of G. The objective is to have as many edges in P as possible.
2. Reinsert the edges not contained in the planar subgraph, i.e., insert the edges in $E \setminus E_p$ into P. During this edge insertion process, edge crossings that occur when inserting an edge are replaced by dummy vertices with degree four, so that the graph remains planar. The objective is to keep the number of dummy vertices (and thus the number of crossings in the final drawing) as small as possible.

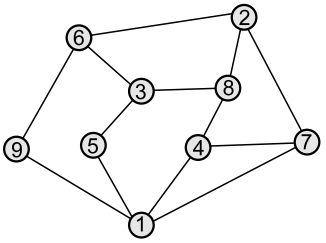
Figure 2.11 shows an example with the different stages of the approach. In this case, the planar subgraph contains all but one edge (edge (2,5) is missing) and the final drawing of G has only a single crossing.

The outcome of the planarization procedure is a planar graph $G_p = (V \cup V_d, E_p)$ such that every planar drawing of G_p implies a drawing of G with at most $|V_d|$ crossings. Hence, we also say that G_p is a *planarized representation* of G with (at most) $|V_d|$ crossings. We can obtain such a drawing of G as follows. First, we compute an embedding of G_p. Then, we have to distinguish two situations for each dummy vertex $v \in V_d$ (see Figure 2.12). If the corresponding edges of G, say, e and e', cross each other, then v in fact represents a crossing between e and e' in the drawing of G. Otherwise, e and e' just touch and we can save a crossing.

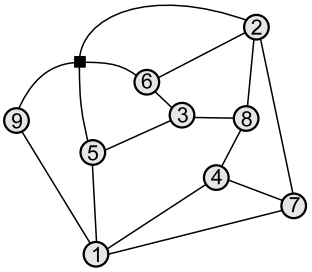
The two optimization problems we have to solve in the planarization approach are the *maximum planar subgraph problem* (MPSP) and the *edge insertion problem* (EIP). Both problems are NP-hard and are usually solved in practice by applying heuristic approaches. One reason for that is that even an optimal solution of MPSP in the first step and of EIP in the second step does not yield a crossing minimal solution in general. We show an example



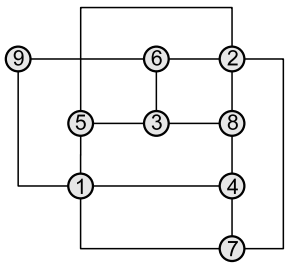
(a) Graph G.



(b) Planar subgraph P of G.



(c) Planarized representation G_p of G.



(d) Final drawing of G.

Figure 2.11 A sample application of the planarization method.

Figure 2.12 (a) The edges e and e' cross at dummy vertex v; (b) e and e' just touch at v.

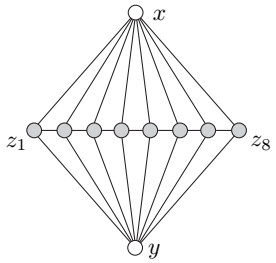


Figure 2.13 A wall with width 8.

(see [GMW05]) where the maximum planar subgraph contains all but one edge, but even an optimal solution of the edge insertion problem results in arbitrary many crossings, whereas a crossing minimal drawing has only two crossings.

We define a wall graph as follows. A *wall* with width k consists of the vertices $x, y, z_1, \ldots, z_k$, the edges (z_i, z_{i+1}) for $1 \leq i < k$, and the edges (x, z_i) and (y, z_i) for $1 \leq i \leq k$; see Figure 2.13 for an example of a wall with width 8. The vertices x and y are called the *poles* of the wall. A wall with width greater than 2 is a triconnected planar graph.

For an even number $m \geq 2$, the graph G_m is constructed in the following way; compare Figure 2.14(a). We start with a ring of walls $W_1, \ldots, W_6$ with width $m+1$, where the poles of adjacent walls in the ring are identified. We denote the pole vertices with $w_1, \ldots, w_6$ such that the poles of W_1 are w_1 and w_2, and so forth. For each wall W_j, the other two vertices on the boundary are denoted with u_j^i and u_j^e; see Figure 2.14(a). Moreover, the edges $e_1 = (u_1^e, w_3)$, $e_2 = (u_6^e, w_5)$, $e_3 = (u_2^i, u_3^i)$, and $e_4 = (u_5^i, u_4^i)$ are added, $m/2$ vertices are inserted by splitting edge (u_3^i, w_4) and $m/2$ vertices are inserted by splitting edge (w_4, u_4^i), and every created split vertex is connected with vertex w_1 by an edge h_j, $1 \leq j \leq m$. We want to insert edge (v_1, v_2) with $v_1 := u_1^i$ and $v_2 := u_6^i$, and we call the graph after addition of this edge G'_m.

By construction, G_m is triconnected and planar. In particular, G_m has only two embeddings which are mirror images of each other. It is easy to see that an optimal insertion of edge (v_1, v_2) crosses m edges, namely, $h_1, \ldots, h_m$, since passing through a wall would require at least $m+1$ crossings. On the other hand, there is a drawing of G'_m with only 2 crossings as shown in Figure 2.14(b). Here, only the two crossings e_1 with e_3 and e_2 with e_4 occur, independent of the choice of m.

In summary, this construction shows that the planarization approach may yield arbitrarily bad solutions even if both steps are solved optimally. On the other hand, practical experience has shown that it leads to excellent results in many applications even if each step is only solved heuristically. In the sequel, we address the two optimization problems—finding a planar subgraph and reinserting a set of edges—in detail, and discuss various solution methods.

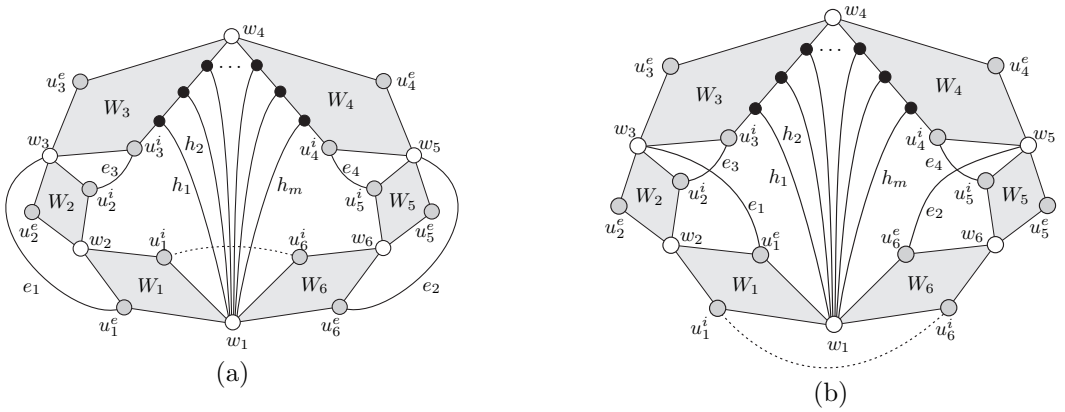


Figure 2.14 (a) The graph G_m; each shaded region represents a wall with width $m+1$. The dashed edge (u_1^i, u_6^i) is the edge to be inserted. (b) A drawing of the graph G'_m with only two crossings.

2.5.2 Planar Subgraphs

In many practical applications, we expect that a graph $G = (V, E)$ can be made planar by removing only a few edges. Therefore, it is reasonable to use a planar subgraph with as many edges as possible as a starting point for crossing minimization. A *maximum planar subgraph* of G is a planar subgraph with the maximum number of edges among all planar subgraphs of G. If, in addition, a weight w_e is given for each edge of G, a *maximum weight planar subgraph* is a planar subgraph $P = (V, E')$ of G such that the sum of all edge weights $\sum_{e \in E'} w_e$ of P is maximum. Hence, a maximum planar subgraph is a special case of the weighted version with $w_e = 1$ for every edge $e \in G$. In both the weighted and the unweighted case, the problem of finding such a subgraph is NP-complete as shown in [LG77, GJ79].

Jünger and Mutzel [JM96] presented a branch-and-cut algorithm for finding a maximum weight planar subgraph. An overview of the branch-and-cut approach can be found in Section 2.4.

Let $\mathcal{P}_G$ be the set of all planar edge-induced subgraphs of G. For each planar subgraph $P = (V, F) \in \mathcal{P}_G$, we define its incidence vector $\chi^P \in \mathbf{R}^E$ by setting $\chi^P_e = 1$ if $e \in F$ and $\chi^P_e = 0$ if $e \notin F$. This yields a 1-1-correspondence of the planar subgraphs with certain $\{0,1\}$-vectors in $\mathbf{R}^E$. The planar subgraph polytope $\mathcal{PLS}(G)$ of G is defined as the convex hull over all incidence vectors of planar subgraphs of G:

$$\mathcal{PLS}(G) := \operatorname{conv}\{\chi^P \in \mathbf{R}^E \mid P \in \mathcal{P}_G\} .$$

Let $w \in \mathbf{R}^E$ be a vector assigning a weight to each edge. The problem of finding a maximum weight planar subgraph can thus be written as the linear program

$$\max\{w^{\mathrm{T}} x \mid x \in \mathcal{PLS}(G)\},$$

since the vertices of the polytope $\mathcal{PLS}(G)$ are exactly the incidence vectors of the planar subgraphs of G. In order to apply linear programming techniques, $\mathcal{PLS}(G)$ has to be represented as the solution of an inequality system. Because of the NP-hardness of the problem, we cannot expect to find a full description of $\mathcal{PLS}(G)$. Jünger and Mutzel show several facet-defining inequalities of the polytope, including Kuratowski inequalities, which are based on the fact that a planar graph contains no subdivision of K_5 and $K_{3,3}$, and Euler inequalities, which are based on the maximal number of edges in a planar graph given by Euler's formula. Further facet-defining inequalities can be found in [JM96].

```
Require: graph G = (V, E)
Ensure: maximal planar subgraph P of G
  P := a spanning tree of G
  F := E \ E(P)
  for all e ∈ F do
    if P ∪ e is planar then
      P := P ∪ e
    end if
  end for
```

Figure 2.15 A simple algorithm for computing a maximal planar subgraph.

Using these inequalities, a branch-and-cut algorithm can be derived that adopts the planarity testing algorithm by Hopcroft and Tarjan [HT74] for cutting plane generation and as lower-bound heuristic. Computational results show that the algorithm is able to provide a provably optimal solution quite fast if the number of edges to be deleted is small. However, the method is quite complicated to understand and to implement. Moreover, if the number of deleted edges exceeds 10, the algorithm usually needs far too long to be acceptable for practical computation.

Since finding a maximum planar subgraph is hard, the problem of finding just a maximal planar subgraph has received much attention. A *maximal planar subgraph* of $G = (V, E)$ is a planar subgraph $P = (V, E \setminus F)$ of G such that adding any edge of F to P destroys the planarity, i.e., $P \cup e$ is not planar for every $e \in F$.

A widely used standard heuristic for finding a maximal planar subgraph is to start with a spanning tree of G, and to iteratively try to add the remaining edges one by one; see Figure 2.15. In every step, a planarity testing algorithm is called for the obtained graph. If the addition of an edge would lead to a nonplanar graph, then the edge is disregarded; otherwise, the edge is added permanently to the planar graph obtained so far. After $|F|$ planarity tests, we obtain a maximal planar subgraph P of G. Planarity can be tested in linear time; see Chapter 1, or [HT74, BL76, BM04]. Hence, the running time of the procedure is $\mathcal{O}((1 + |F|)(|V| + |E|))$.

This incremental approach can be made more efficient by using incremental planarity testing algorithms. Di Battista and Tamassia [DT96] presented an algorithm that tests in $\mathcal{O}(\log |V|)$ time if an edge can be added while preserving planarity, and that performs the required updates of the data structure when adding an edge in $\mathcal{O}(\log |V|)$ amortized time. The algorithm uses the data structures BC-tree and SPQR-tree equipped with efficient, dynamic update operations. A *BC-tree* represents the *block-cutvertex tree* of a connected graph G which consists of the interrelation of the blocks (B-nodes) and cutvertices (C-nodes) of G. It has an edge (c, B) if c is a cutvertex of G contained in block B. *SPQR-trees* have been introduced by Di Battista and Tamassia in [DT89]. They represent the decomposition of a biconnected graph into its triconnected components, which essentially consists of serial (expressed by S-nodes), parallel (P-nodes), and simple, triconnected structures (R-nodes). Additionally, Q-nodes represent the original edges of G. The specific structures of tree nodes are given by skeleton graphs that are associated with each node. Using these data structures, a maximal planar subgraph can be found in $\mathcal{O}(|E| \log |V|)$ time. SPQR-trees are also useful in a static environment for the representation of all planar embeddings of a graph. The static data structure can be built in linear time [HT73, GM01] using an algorithm for dividing a graph into its triconnected components.

The running time for incremental planarity testing has been improved by La Poutré [La 94] to $\mathcal{O}(\alpha(|E|,|V|))$ amortized time per query and update operation. This yields an almost linear time algorithm for the maximal planar subgraph problem that runs in $\mathcal{O}(|V|+|E|\cdot \alpha(|E|,|V|))$ time. Here, $\alpha(x,y)$ denotes the inverse Ackermann function, which means that $\alpha(x,y)$ is a function that grows extremely slowly. A linear time algorithm for finding a maximal planar subgraph is given by Djidjev [Dji95]. This algorithm uses BC- and SPQR-trees and applies a fast data structure for online planarity testing in triconnected graphs.

Jayakumar et al. [JTS89, JLM98] proposed a method for computing a planar subgraph that is based on PQ-trees. The PQ-tree data structure has been developed by Booth and Lueker [BL76] for solving the problem of finding permissible permutations of a set U. The permissible permutations are those in which certain subsets $S \subseteq U$ occur as consecutive subsequences. Drawbacks of this planar subgraph algorithm are that it cannot guarantee to find a maximal planar subgraph, and that the theoretical worst-case running time is $\mathcal{O}(|V|^2)$. However, in practice it is usually very fast and the quality of the results can be improved by introducing random events and calling the algorithm several times. The algorithm starts by computing an st-numbering of G, which determines the order in which the vertices are processed. A simple but useful randomization is to choose a random edge (s,t) for each run.

The trivial approach for finding a planar subgraph consists of computing a spanning tree. If G is a graph with n vertices and c components, then this approach has an approximation factor of $\frac{n-c}{3n-6c} > \frac{1}{3}$ for MPSP, since a spanning tree of G contains $n-c$ edges, and a planar graph with c components has at most $3n-6c$ edges by Euler's formula. Surprisingly, we cannot guarantee a better approximation factor than that of the spanning tree approach if we demand that the computed subgraph must be maximal planar; see [DFF85]. Călinescu et al. [CFFK98] present an algorithm with approximation factor 4/9 that runs in $\mathcal{O}(m^{3/2} n \log^6 n)$ time, where m is the number of edges of G. For the maximum weight planar subgraph problem, the simple approach is to compute a maximum weight spanning tree, which gives an approximation factor of 1/3, and the best algorithm [CFKZ03] achieves an approximation factor of $1/3 + 1/72$.

2.5.3 Edge Insertion

The planar subgraph P computed in the first step of the planarization approach is a good starting point for finding a planarized representation G_p of G with few crossings. In practice, we expect that only a small number of edges has to be inserted into P in order to obtain G_p. However, the edge insertion step fixes the crossings in the final drawing, and the choice of the edge insertion technique may have a significant impact on the quality of the final solution. Ziegler and Mutzel [MZ99, Zie00] have shown that even a restricted variant of the edge insertion problem is NP-hard: The *constrained crossing minimization problem* (*CCMP*) asks for the minimum number of crossings required for inserting a set of edges into a fixed embedding. They also present a branch-and-cut algorithm to solve CCMP. However, experiments show that it can only solve instances to provable optimality if there are less than 10 edges to be inserted.

Gutwenger [GM04, Gut10] has conducted an extensive study on crossing minimization heuristics, including different methods for edge insertion. Figure 2.16 shows the general framework for edge insertion used in this study. It contains three essential parts leaving room for enhancement:

Single edge insertion: The edges are inserted into the planarized representation individually one after the other. The simple approach for inserting a single edge e

```
Require: planar subgraph P = (V, E_P) of G = (V, E)
Ensure: planarized representation G*_p of G
  Let E \ E_P = {e_1, ..., e_k}
  best := ∞
  for i := 1 to nPermutations do
    Let σ be a randomly chosen permutation of {1, ..., k}
    G_p := P
    for j := 1 to k do
      Insert edge e_σ(j) into G_p
    end for
    Determine a set R ⊆ E of edges for which postprocessing shall be applied
    repeat
      for all e ∈ R do
        Remove edge e from G_p
        Insert edge e into G_p
      end for
    until number of crossings in G_p has not decreased
    current := number of crossings in G_p
    if current < best then
      G*_p := G_p; best := current
    end if
  end for
```

Figure 2.16 Edge insertion with postprocessing and permutation.

is to fix an embedding Π of G_p and to insert e into Π. However, the choice of Π may have a considerable influence on the number of edges that e has to cross. A more sophisticated algorithm introduced by Gutwenger et al. [GMW05] is able to insert e with the minimum number of crossings among all embeddings of G_p.

Postprocessing: After all edges have been inserted, a simple postprocessing technique tries to improve the current solution. It determines a set of edges R which have one or more crossings and repeatedly tries to find a better insertion path for each of them by removing an edge from G_p and inserting it again. Variants for the choice of R include all edges, only the edges $e_1, \ldots, e_k$, or some portion of the edges with the most crossings (see [Gut10] for more details).
An alternative approach combines the edge insertion with the postprocessing. Instead of performing the remove-reinsert strategy after all edges have been inserted, we can perform this strategy after each edge insertion. The idea behind this variation is to keep the number of crossings low as early as possible. We call this strategy incremental postprocessing.

Permutation: The order in which the edges $e_1, \ldots, e_k$ are processed also affects the final number of crossings. Calling the complete edge insertion process several times with different, randomly chosen permutations of the edge list $e_1, \ldots, e_k$ may significantly improve the solution. The parameter *nPermutations* in the algorithm determines the number of permutation rounds.

Apart from the choice of some parameters like the number of permutation rounds or the selection of the edges for postprocessing, the challenging part of the algorithm is the insertion of a single edge. We consider the two variants—insertion with *fixed* and with *variable* embedding—in more detail.

Fixed Embedding. Suppose, we want to insert edge $e = (v, w)$ into the planar graph G_p. Let Π be a fixed embedding of G_p. We construct the *extended dual graph* G^* of Π with respect to e as follows. The vertices of G^* are the faces of Π plus two new vertices v^* and w^* representing v and w. For each edge e' in G_p, we have an edge in G^* connecting the two faces separated by e' (if e' is a bridge, we have a self-loop in G_p). Additionally, we have an edge (v^*, f_v) for each face f_v adjacent to v, and (w^*, f_w) for each face f_w adjacent to w.

We observe that inserting e into Π corresponds to finding an (undirected) path from v^* to w^* in G^*. If such a path has length ℓ, then we can insert e with $\ell - 2$ crossings, since the first and the last edge on this path do not produce a crossing. Therefore, in order to insert e into Π with the minimum number of crossings, we have to find a shortest path from v^* to w^* in G^*. This is possible in linear time using a simple breadth-first search traversal starting at v^*. Figure 2.17 shows a nontrivial example. Here, we want to connect the vertices 1 and 2. The dashed vertices and edges belong to the extended dual graph. The optimal solution highlighted in bold crosses four edges.

Though we can easily find a crossing minimal solution if the embedding of G_p is fixed, the drawback of this method is that fixing an unfavorable embedding may result in an arbitrarily bad solution. Figure 2.18(a) gives an example of such a family of graphs G_k with embeddings Γ_k. The black fat lines in this figure denote bundles of $k + 1$ parallel

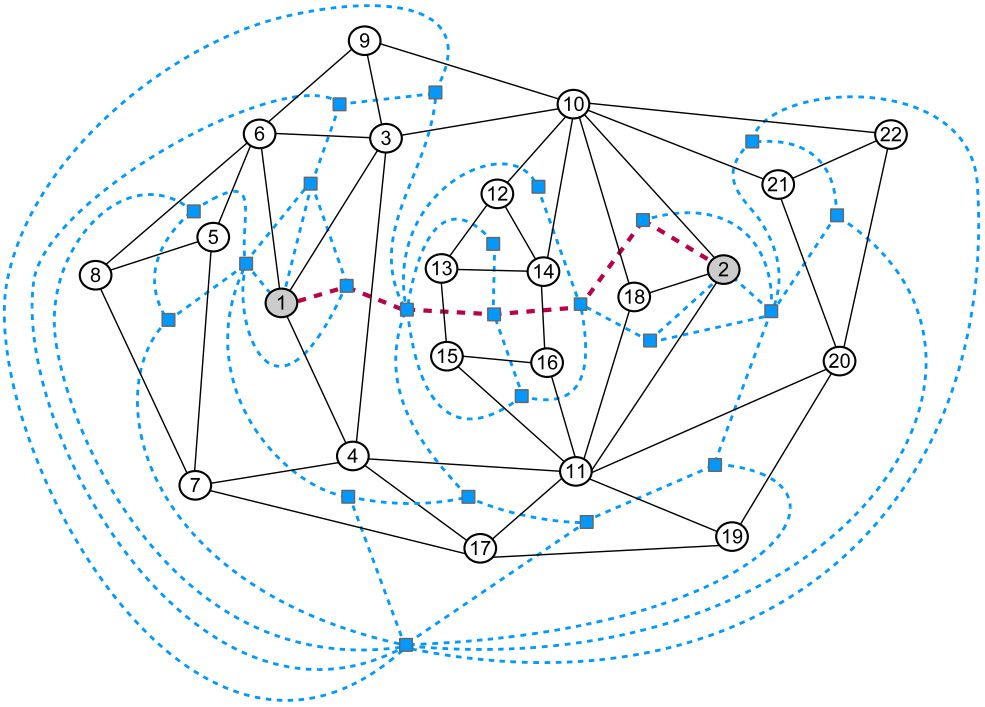


Figure 2.17 Edge insertion with fixed embedding by finding a shortest path in the extended dual graph.