Philipp Kindermann

Angular Schematization in Graph Drawing

Philipp Kindermann

Angular Schematization in Graph Drawing

Philipp Kindermann

Angular Schematization
in Graph Drawing

Preface

This thesis, by Philipp Kindermann, deals with the visualization of graphs. Graphs are a frequently used tool to model relationships among entities. A graph is a binary relation between objects, that is, it consists of a set of objects (*vertices*) and a set of pairs of objects (*edges*). Networks are commonly modeled as graphs. For example, relationships between actors in a social network, or links between computers in a telecommunication network can be represented by a graph. For human beings, arguably, the best way to approach such data is to draw the underlying graphs. The field of Graph Drawing is about finding algorithms that draw graphs "nicely". The niceness of a drawing can be measured by different criteria such as the number of crossings between edges or the area of the drawing.

Philipp Kindermann's thesis is devoted to *Angular Schematization in Graph Drawing*, where the task is to lay out graphs under angular restrictions. These restrictions concern different objects in the drawing. In many layouts, the directions of the edges are restricted to a small set of slopes; a famous example are *orthogonal layouts* where only horizontal and vertical edge segments are allowed. Often drawings of graphs are required to induce large angles between edges that meet at a vertex. Preferably, edges are drawn without crossings and in a way that makes them easy to follow for the human eye. If crossings cannot be avoided, it is important to have large (ideally right) angles between crossing edges. Philipp Kindermann considers these different types of angular restrictions and presents several results to problems that are discussed extensively in the graph drawing community.

Philipp Kindermann introduces us into his thesis by giving an overview on the field of graph drawing and on his results. Then, he presents the tools needed for the algorithms and proofs in the following chapters. The main part of the thesis consists of three parts, each of which is further divided into two chapters. The three parts address the placement of boxes, visual guidance, and crossings at large angles.

Part I deals with two different problems concerning the *placement of boxes*. First, Philipp Kindermann considers the problem of labeling sites on a map with text that cannot be placed directly next to these points because the sites are too close to each other with regards to the label sizes. Instead, he places the labels beyond the boundary of the map and connects them to the corresponding points by orthogonal curves (*leaders*). The boxes should be placed such that the leaders do not intersect each other and are preferably as short as possible. The author gives the first algorithms to decide whether there is a crossing-free drawing that places the labels on two or more adjacent sides of the map. The dynamic programming algorithms exploit some nice characterizations of feasible solution. Second, Philipp Kindermann considers box contact representations. In such a representation, each vertex is represented by a rectangle of specified height and width; the edges are modeled by contact between the rectangles. The task is to realize as many edges as possible. The author shows that this problem is APX-complete and presents approximation algorithms for several graph classes, including the first constant-factor approximation algorithm for general graphs, which I consider a very nice result.

Part II deals with *visual guidance* in graph drawing, that is, approaches that make it easy for the viewer to follow the trajectory of an edge or even a path between two vertices in a drawing of a graph. First, Philipp Kindermann considers *smooth orthogonal* drawings of planar graphs of maximum degree 4. In this drawing style, edges consist only of axis-aligned segments and circular arcs that meet without bends. By design, the angles between edges meeting in a vertex are at least 90°. The main result is that each planar graph of maximum degree 4 admits a crossing-free drawing with at most two segments per edge, and that this bound is tight. Second, the author examines drawings of graphs in which a path between two vertices can be easily identified. To this end, he considers *monotone* drawings, that is, drawings where each pair of vertices is connected by a path that is monotone in some direction. In *strongly* monotone drawings, this direction is the vector between the locations of the two vertices. Philipp Kindermann shows that any tree admits a monotone drawing in polynomial area such that the unbounded faces are convex and the vertex angles are close to optimal. Furthermore, he shows that every tree and every biconnected outerplanar graph admit a strongly monotone drawing, which is not the case for simply connected planar graphs.

In the final part of the thesis, part III, Philipp Kindermann investigates drawings of graphs with *crossings at large angles*. First, he considers *simultaneous drawings*, that is, drawings of two planar graphs on a common point set such that each drawing is crossing-free, but edges of different graphs may cross each other. He shows that any pair of planar graphs can be simultaneously drawn with at most six bends per edge such that crossings between edges of different graphs are always at right angles. For subclasses of planar graphs, he reduces the number of required bends. Second, the author studies *IC-planar* graphs, that is, graphs in which every edge is crossed at most once and every vertex is incident to at most one crossing edge. He shows that any IC-planar graph can be drawn with right-angle crossings. On the other hand, he shows that there are IC-planar graphs that require exponential area when insisting on right-angle crossings. Further, it is NP-hard to decide whether a given graph is IC-planar, even if the underlying rotation system is prescribed, that is, the cyclic ordering of the edges around each vertex is given. I find this a valuable contribution to the study of near-planar graphs.

In this thesis, Philipp Kindermann addresses a number of problems that are currently in the focus of the graph drawing community. I had the pleasure of working on some of these problems together with him, and I enjoyed seeing all this work come together in this beautiful thesis. I wish the author and the readers many more nice graph-drawing problems to work on!

Alexander Wolff
Chair I – Efficient Algorithms and Knowledge-Based Systems
Institute of Computer Science
University of Würzburg

Contents

Contents

1   Introduction

In Graph Theory, a graph is an abstract structure that represents a set of objects, called *vertices*, and a set of pairs of vertices, called *edges*. Graphs are a frequently used tool to model network data. For example, a social network can be modeled by a graph whose vertices correspond to persons, and whose edges correspond to relationships between these persons. Similarly, in integrated circuit design for the production of computer components such as microprocessors or memory systems, vertices represent the electronic components and edges represent connections. A wide range of problems can be solved efficiently by modeling as a graph, and then using graph algorithms to solve the underlying problem. For instance, a route between two positions computed by a navigation system can be found by finding a shortest path in the graph that represents the street network. The term "graph" has been introduced by Sylvester [Syl78].

Graph Drawing. The area of visualizing graphs is called *Graph Drawing*. Graphs are usually drawn as node-link diagrams in the Euclidean plane. In such diagrams, the vertices are represented as geometric objects such as points, disks, or boxes, and the edges are drawn as Jordan curves, for example, line segments, polygonal lines, or circular arcs. A drawing should be *readable*, that is, the human eye should be able to easily follow the edges and thus understand the relationship between the objects in the graph at first glance. In a good drawing, simple tasks can be answered quickly, for example, finding a shortest path between two vertices. Various properties that define a good drawing have been studied.

A very common approach is to draw graphs such that they induce no crossing between edges; or, if this is not possible, to minimize the number of crossings. Another commonly desired feature is the minimization of the drawing area, usually defined as the area of the smallest bounding box containing the drawing under the restriction that the vertices are placed on the integer grid. Recently, researchers have reconsidered the problem how to draw graphs with *angular restrictions*.

Angular Schematization. The problem of *angular schematization* deals with computing drawings of graphs under angular restrictions. In many network layouts, the directions of the edges are restricted, for example, any edge in an *orthogonal layout* consists exclusively of horizontal and vertical segments. These layouts are constructed in a wide range of research communities: graph drawing, geographic information science, information visualization, VLSI layout, computational geometry, and underground mining. Most fields of application require large angles between edges that meet at a common vertex or a crossing point. Clearly, it is hard to follow an edge if there are many edges that are drawn close to each other with a small angle between them. We distinguish between two types of angles: *vertex angles* and *crossing angles*. Another possible requirement in angular schematization is *monotonicity*, that

is, for each pair of vertices there exists some direction such that the drawing contains a path that connects the vertices and is increasing in this direction.

Vertex Angles.   A vertex angle is the angle formed by two edges that meet at a common vertex of the drawing. The *angular resolution* of a drawing is the smallest vertex angle. Angular resolution was first defined by Formann et al. [FHH$^+$93] for straight-line drawings. Drawings with small angular resolution make it difficult for the viewer to tell lines apart. Thus, an important aesthetic criterion is to have high angular resolution, preferably *perfect angular resolution*, that is, the edges are equally spaced around each vertex. Clearly, in a graph with maximum degree d, the angular resolution is bounded by $2\pi/d$. For trees, this bound can be achieved in polynomial area. However, even for outerplanar graphs, perfect angular resolution cannot always be achieved with straight-line edges, since the edges of a vertex on the convex hull of the drawing cannot be spaced equally. Moreover, for some planar graphs, the optimal angular resolution of a planar straight-line drawing of maximum degree d is $\Theta(1/d^3)$ [GT94]. We deal with drawings for planar graphs of maximum degree 4 with perfect angular resolution in Chapter 5, and with drawings of trees with close-to-perfect angular resolution in Chapter 6.

Crossing Angles.   A crossing angle is the angle formed by two crossing edges in their common crossing point. Similar to vertex angles, small crossing angles can make a viewer follow the wrong edge after encountering a crossing. In a user study by Huang et al. [HHE08], it has been shown that crossings with large angles are much less harmful to the readability of drawings than shallow crossings. Thus, not only the minimization of the number of crossings, but also the maximization of the crossing angles are important in the visualization of graphs. Furthermore, even more planar graphs it may be beneficial to allow some crossings if they induce right angles [vK11]. Such drawings are called *right-angle crossing* (RAC) drawings. In general, it is NP-hard to decide whether a given graph admits a RAC drawing with straight edges, and the maximum number of edges in such a drawing is $4n - 10$. However, if bends are allowed, this number increases to $6.5n$ (one bend per edge) and $74.2n$ (two bends per edge). Every graph as a RAC drawing with three bends per edge. In Chapter 7, we produce RAC drawings (with bends) for two planar graphs on a common vertex set, and in Chapter 8 we show that every graph with independent crossing edges admits a RAC drawing.

Contact Representations.   A *contact representation* describes an approach in visualizing graphs different from node-link diagrams. In such a representation, vertices are drawn as non-overlapping geometric objects of a given type such as disks, rectangles or polygons. In contrast to node-link diagrams, an edge is represented as a common region of two objects, that is, if two objects *touch*, then an edge between their corresponding vertices exists in the underlying graph. By a classical results of Koebe [Koe36], the class of graphs that are representable by touching disks in the plane is exactly the class of planar graphs. However, if the shape and area of the objects are prescribed, in general it is NP-hard to decide whether a graph admits a contact representation. We deal with contact representations of rectangles with given width and height in Chapter 4. Since not every graph admits such a drawing, we seek to maximize the number of represented edges.

Map Labeling. The placement of labels on a map is an important field of visualization that is related to graph drawing. In this problem, the task is to place the names (*labels*) of specific sites, for example towns or points of interest, on a map such that the reader can immediately identify which name describes which site. Usually, labels are represented by the smallest bounding rectangle containing the text in a given font. For readability, labels have to be placed without overlaps and close to their corresponding sites. If labels are too large to be placed on the map, they are usually placed into the margin, and connected to the sites by so-called *leaders*. We study this problem, called *boundary labeling*, in Chapter 3. In terms of graph drawing, this problem can be interpreted as drawing a matching between sites and labels with given coordinates of the sites and a specified region into which the labels have to be placed.

1.1 Outline of the Book

This book consists of three parts, each dealing with different types of angular restrictions. In Part I, we deal with the *placement of boxes*. In this setting, vertices are associated with a text, and are represented by the bounding rectangle of the words. The edges are represented by common horizontal or vertical segments of the rectangles. Part II is devoted to *visual guidance*, that is, planar drawings with high angular resolution and edges that are appealing and easy to follow. Finally, Part III deals with non-planar graphs that are drawn with *crossings with large angles*. In these drawings, crossing edges induce exclusively right angles, such that the reader does not inadvertently "jump" to another edge when encountering a crossing. Before starting with the main part of this book, we give a short introduction into the terminology that we use; see Chapter 2.

1.1.1 Placement of Boxes

In Part I of this book, we deal with the placement of boxes representing text. We consider two different problems. First, the boxes are used to label important sites on a map with text that is too large to be placed directly next to the corresponding site. Thus, we place the boxes in the margin of the map and connect them to the labeled points by so-called *leaders*. Second, we want to create semantic word clouds, that is, word clouds in which related words (according to some semantic) are placed next to each other. More formally, the task is to create a non-overlapping box contact representation of an underlying graph such that two boxes touch if there is an edge between the corresponding vertices.

Multi-Sided Boundary Labeling

In Chapter 3, we study the *Multi-Sided Boundary Labeling* problem, with labels lying on at least two sides of the enclosing rectangle; see Figure 1.1 for an illustration. We present a polynomial-time algorithm that computes a crossing-free leader layout if one exists. So far, such an algorithm has only been known for the cases in which labels lie on one side or on two opposite sides of R. In both cases, a crossing-free solution always exists. The case where labels may lie on adjacent sides is more difficult. We present an efficient algorithm for testing the existence of a crossing-free leader layout that labels all sites. We also study the problem
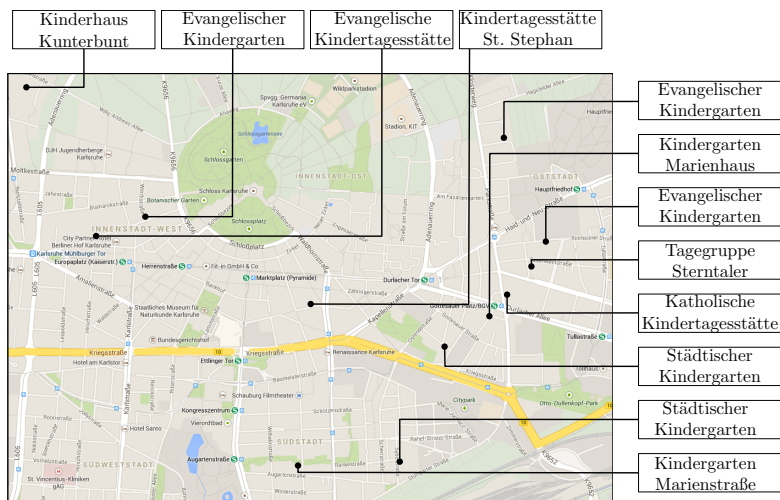
Figure 1.1: Boundary Labeling for kindergartens in Karlsruhe, Germany

of maximizing the number of labeled sites in a crossing-free leader layout. For two-sided boundary labeling with adjacent sides, we show how to minimize the total leader length in a crossing-free layout.

This chapter is based on joint work with Benjamin Niedermann, Ignaz Rutter, Marcus Schaefer, André Schulz, and Alexander Wolff [KNR⁺13].

Approximation Algorithms for Box Contact Representations

In Chapter 4, we study the following geometric representation problem: Given a graph whose vertices correspond to axis-aligned rectangles with fixed dimensions, arrange the rectangles without overlaps in the plane such that two rectangles touch if the graph contains an edge between them; see Figure 1.2 for an example. This problem was named CONTACT REPRESENTATION OF WORD NETWORKS (CROWN) since it formalizes the geometric problem behind drawing word clouds in which semantically related words are close to each other. CROWN is known to be NP-hard, and there are approximation algorithms for certain graph classes for the optimization version, MAX-CROWN, in which realizing each desired adjacency yields a certain profit.

We present the first $O(1)$-approximation algorithm for the general case, when the input is a complete weighted graph, and for the bipartite case. Since the subgraph of realized adjacencies is necessarily planar (if we insist on non-trivial contacts), we also consider several planar graph classes (namely stars, trees, outerplanar, and planar graphs), improving upon the known results. For some graph classes, we also describe improvements in the unweighted case, where each adjacency yields the same profit. Finally, we show that the problem is APX-hard even on bipartite graphs of bounded maximum degree.

This chapter is based on joint work with Michael A. Bekos, Thomas C. van Dijk, Martin Fink, Stephen Kobourov, Sergey Pupyrev, Joachim Spoerhase, and Alexander Wolff [BvDF⁺14].
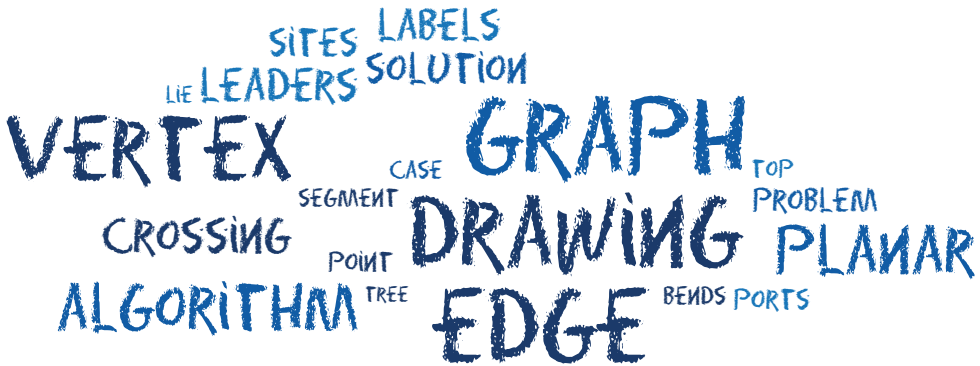
Figure 1.2: A semantic word cloud representing the 20 most frequently used words in this book, excluding common English words; created with the World Cloud Generator at http://wordcloud.cs.arizona.edu/cloud.html?id=715 (Layout: Star Forest).

1.1.2 Visual Guidance

In Part II of this book, we seek to create drawings of specific subclasses of planar graphs that visually guide the viewer, that is, it is easy for the human eye to continuously follow an edge and to find a path between to vertices. Naturally, bends interrupt the eye movement of the viewer, as it has to abruptly change the direction it follows. Thus, we seek to draw edges with straight-line segments and circular arcs that induce no bends. For circular arcs, we consider its tangent at the touching point with another segment to determine whether a bend occurs. In straight-line drawings, the task is to draw them in a way that a path between two vertices can be found by linearly following a specific direction, preferably the direction between the two endpoints of the path. The drawings computed by our algorithms have perfect or close-to-perfect angular resolution.

Smooth Orthogonal Layouts of Planar Graphs

Chapter 5 is devoted to *smooth orthogonal layouts* of planar graphs. In these layouts, every edge is an alternating sequence of axis-aligned segments and circular arcs with common axis-aligned tangents; see Figure 1.3 for some example drawings. In this chapter, we study the problem of finding smooth orthogonal layouts of low *edge complexity*, that is, with few segments per edge. We say that a graph has *smooth complexity k*—for short, an SC_k-layout—if it admits a smooth orthogonal drawing with at most k segments per edge.

Our main result is that every planar graph of maximum degree 4 has an SC_2-layout. While our drawings may have super-polynomial area, we show that for planar graph of maximum degree 3, cubic area suffices. We also show that any biconnected outerplane graph of maximum degree 4 has an SC_1-layout. On the negative side, we construct two infinite families of biconnected planar graphs of maximum degree 4 that (1) require exponential area for an SC_1-layout and (2) do not admit an SC_1-layout.

This chapter is based on joint work with Md. Jawaherul Alam, Michael A. Bekos, Michael Kaufmann, Stephen G. Kobourov, and Alexander Wolff [ABK+14].
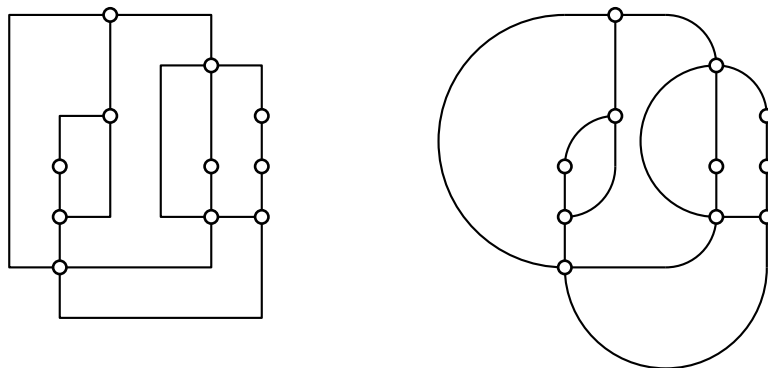
Figure 1.3: An orthogonal layout (left) and a smooth orthogonal layout (right) of the same graph

Monotone Drawings of Trees

In Chapter 6, we investigate *monotone* drawings. A crossing-free straight-line drawing of a graph is monotone if there is a monotone path between any pair of vertices with respect to *some* direction. We show how to construct a monotone drawing of a tree with n vertices on a section of size $O(n^{1.5}) \times O(n^{1.5})$ of the integer grid such that the angles at a vertex v are bounded from below by roughly $1/\deg(v)$. Our drawings are *convex*, that is, if every edge to a leaf is substituted by a ray, the (unbounded) faces form convex regions. It is known that convex drawings are monotone and, in the case of trees, crossing-free.

A monotone drawing is *strongly monotone* if, for every pair of vertices, the direction that witnesses the monotonicity comes from the vector that connects the two vertices. We show that every tree admits a strongly monotone drawing; see Figure 1.4 for an example drawing. For biconnected outerplanar graphs, this is easy to see. On the other hand, we present a simply-connected graph that does not have a strongly monotone drawing in any embedding.

This chapter is based on joint work with André Schulz, Joachim Spoerhase, and Alexander Wolff [KSSW14].
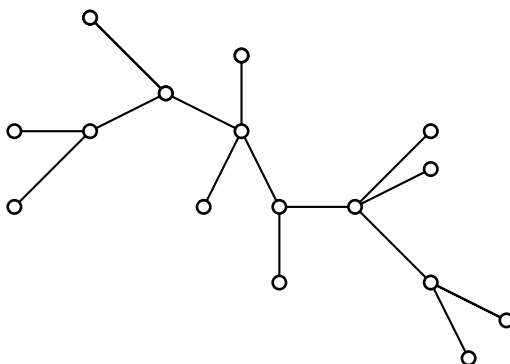
Figure 1.4: A monotone (and strictly convex) drawing of a tree
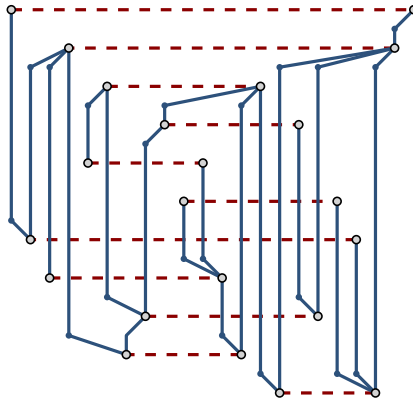
Figure 1.5: A RacSim drawing of a tree (solid edges) and a matching (dashed edges)

1.1.3 Crossings with Large Angles

In Part III of this book, we consider non-planar graphs. If a drawing of these graphs induces small crossing angles, the reader may follow the wrong edge after a crossing point. To avoid this, angles at edge crossings should be large. We deal with drawings of non-planar graphs in which all crossings form right angles (Rac). In simultaneous embeddings, two planar graphs are drawn on the same point set such that each graph is plane, and the crossings between the two graphs are Rac. Since only few classes of graphs admit such drawings, we allow a constant number of bends per edge. Then, we consider IC-planar graphs, that is, graphs that have a drawing in which every edge is crossed at most once and every vertex is incident to at most one crossing edge. We prove that every IC-planar graph admits a straight-line Rac drawing. If we drop the restriction of only one crossing edge per vertex, the graphs are called 1-planar graphs and cannot always be drawn Rac.

Simultaneous Drawing of Planar Graphs with Right-Angle Crossings

In Chapter 7, we study the *Rac simultaneous drawing* problem. Given two planar graphs that are defined on the same set of vertices, a Rac simultaneous drawing is one in which each graph by itself is drawn planar, there are no edge overlaps and the crossings between the two graphs form right angles. The geometric version restricts the problem to straight-line drawings. It is known, however, that there exists a wheel and a matching which do not admit a geometric Rac simultaneous drawing.

In order to enlarge the class of graphs that admit Rac simultaneous drawings, we allow bends when drawing the edges; see Figure 1.5. We prove that two planar graphs always admit a Rac simultaneous drawing with six bends per edge, in quadratic area. For more restricted classes of planar graphs (that is, matchings, paths, cycles, outerplanar graphs, and subhamiltonian graphs), we manage to significantly reduce the required number of bends per edge while keeping the drawing area quadratic.

This chapter is based on joint work with Michael A. Bekos, Thomas C. van Dijk, Alexander Wolff [BvDKW15].
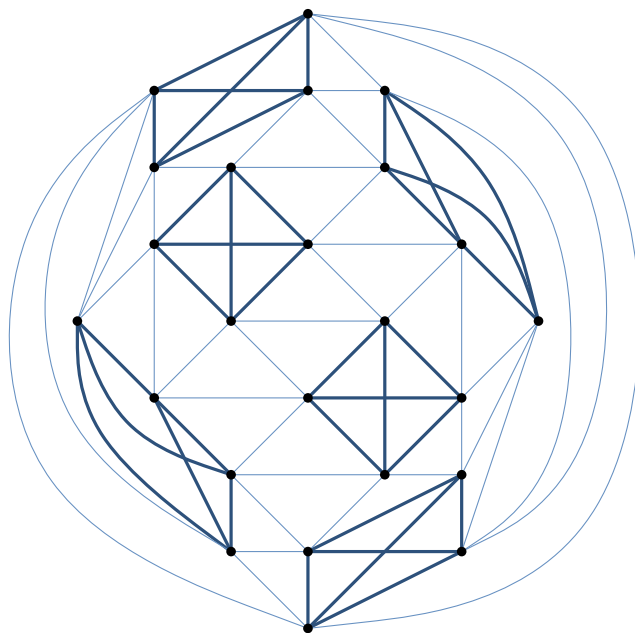
Figure 1.6: A drawing of an IC-planar graph

Recognizing and Drawing IC-Planar Graphs

In Chapter 8, we consider a subclass of 1-planar graphs, so-called *IC-planar* graphs. A graph is called *1-planar* if it can be drawn in the plane such that each edge is crossed at most once, and is called IC-planar if the crossings are independent and no vertex is incident to more than one crossing edge; see Figure 1.6 for an example graph. A RAC graph is a graph that admits a straight-line right-angle crossing drawing. It has been shown by Eades and Liotta [EL13] that not every 1-planar graph is a RAC graph and vise versa. They have stated the open problem of characterizing the intersection between the class of 1-planar graphs and the class of RAC graphs. We prove that the class of IC-planar graphs lies in this intersection.

We show that every IC-planar graph admits a straight-line drawing in polynomial area that can be computed in linear time. However, we observe that quadratic area is sometimes necessary for straight-line IC-planar drawings. Further, the angles formed by crossing angles in this drawing might be small. We present an algorithm that, given an embedded IC-planar graph, computes a RAC drawing of the given graph. Our drawings need exponential area. This area bound is justified by the fact that there are IC-planar graphs that cannot be drawn in polynomial area.

For a plane triangulated graph, it can be tested in polynomial time whether a disjoint set of matching edges can be added to form an IC-planar graph. However, testing IC-planarity of a graph is NP-complete, even if the graph is given with a rotation system.

This chapter is based on joint work with Franz J. Brandenburg, Walter Didimo, William S. Evans, Giuseppe Liotta, and Fabrizio Montecchiani.

Acknowledgements

2

Preliminaries

This chapter contains the most important preliminaries and definitions of graphs, graph drawing, and complexity of algorithms. We seek to give a small introduction to the notation and the techniques used in this book. For an extended introduction into these fields, we refer to books designated to this matter.

A basic introduction to graph theory is given in the book *Introduction to Graph Theory* by Trudeau [Tru93]. A more algorithmic approach is used in *Algorithmic Graph Theory* by Gibbons [Gib85]. The basic concepts and algorithms for graph drawing are presented in *Graph Drawing: Algorithms for the Visualization of Graphs* by Di Battista et al. [BETT99], *Drawing Graphs: Methods and Models* by Kaufmann and Wagner (editors) [KW01], and *Planar Graph Drawing* by Nishizeki and Rahman [NR04]. These works are complemented by the recently published *Handbook of Graph Drawing and Visualization* by Tamassia (editor) [Tam13]. For an introduction to the basic design and analysis of algorithms, we refer to *Introduction to Algorithms* by Cormen et al. [CLRS09]. Regarding approximation algorithms, more information can be found in *Approximation Algorithms* by Vazirani [Vaz03] an *The Design of Approximation Algorithms* by Williamson and Shmoys [WS11].

2.1 Graphs

A *directed graph* is defined by a tupel $G = (V, E)$ of a non-empty set of vertices V and a set of directed edges $E \subseteq V \times V = \{(u, v) \subseteq V \mid u \neq v\}$. We say that an edge $e = (u, v)$ starts in u and ends in v. We call u and v the *endpoints* of e. We call e an *outgoing* edge of u and an *incoming* edge of v. We forbid so-called *loops*, that is, edges where the two endpoints are the same. We denote the number of vertices by $n = |V|$ and the number of edges by $m = |E|$. Clearly, $m \leq \binom{n}{2} = n(n-1)/2$.

An *undirected graph* is defined by a tupel $G = (V, E)$ of a non-empty set of *vertices V* and a set of undirected *edges* $E = \{\{u, v\} \subseteq V \mid u \neq v\}$. The notation $e = (u, v)$ is often used for undirected edges, too. In this case, we identify (u, v) and (v, u). If not specified otherwise, we assume graphs to be undirected.

In a graph $G = (V, E)$, two vertices $u, v \in V$ are called *adjacent* or *neighbors* if $(u, v) \in E$. We denote the set of vertices that are adjacent to a vertex $u \in V$ by $\mathrm{Adj}(u) := \{v \in V \mid (u, v) \in E\}$. Given an edge $(u, v) \in E$, we say that the vertices u and v are *incident* to (u, v).

For a vertex v of an undirected graph, we define the *degree* $\deg(v) := |\mathrm{Adj}(v)|$ of v as the number of adjacent vertices. The *maximum degree* of a graph, which we denote by Δ, is the maximum of the degrees over all vertices, that is, $\Delta = \max_{v \in V} \deg(v)$. For a vertex v of an directed graph, we define the *outdegree* $\mathrm{outdeg}(v) := |\{u \in V \mid (v, u) \in E\}|$ as the number of edges leaving v and the *indegree* $\mathrm{indeg}(v) := |\{u \in V \mid (u, v) \in E\}|$ as the number of edges entering v.

We call $G' = (V', E')$ a *subgraph* of a graph $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. We say that G' is *induced* by V' if E' contains exactly the edges of G whose endpoints are both in V', that is, if $E' = \{(u, v) \in E \mid u, v \in V'\}$. This graph is denoted by $G[V']$.

A *subdivision* of a graph G is a graph G' that is obtained by replacing every edge by a path of arbitrary length. We call this operation *subdividing* an edge. The inverse operation, that is, replacing a path by an edge, is called *smoothing out* (the vertices on) the path. Similarly, and edge (u, v) can be *contracted*, that is, it is replaced by a single vertex w that is adjacent to the neighbors of u and v.

A *path* P of *length* k is a graph $P = (V, E)$ with $V = \{v_0, v_1, \ldots, v_k\}$ and $E = \{(v_i, v_{i+1}) \mid 0 \leq i \leq k - 1\}$. We also denote a path by the sequence of its vertices $P = \langle v_0, v_1, \ldots, v_k \rangle$, or by its endpoints $v_0 \rightarrow v_k$. A path is called *simple* if all vertices are pairwise different. We say that there is a path from u to v in a graph G if there is a subgraph $P = \langle u, \ldots, v \rangle$ of G. A *cycle* of length k is a path $C_k = \langle v_0, v_1, \ldots, v_k \rangle$ of length k with $v_0 = v_k$ and $k > 1$ for directed graphs, and $k > 2$ for undirected graphs. A cycle is called *simple* if all vertices are pairwise different, with the exception of $v_0 = v_k$.

A graph $G = (V, E)$ is *connected* if for every for every pair of vertices $u, v \in V$ there is a path from u to v in G; otherwise, G is *disconnected*. A *connected component* of G is a connected subgraph $G[V']$ induced by a maximal subset $V' \subseteq V$, that is, for every V'' with $V' \subsetneq V'' \subseteq V$, $G[V'']$ is not connected. Thus, $G[V] = G$ is the only connected component of G if G is connected.

A *Hamiltonian path* is a path that visits every vertex exactly one. Correspondingly, a *Hamiltonian cycle* is the augmentation of a path to a cycle. A graph is called *Hamiltonian* if it contains a Hamiltonian cycle.

A graph G is called *k-vertex-connected* or *k-connected* if G remains connected after the removal of any $k - 1$ vertices. Correspondingly, G is called *k-edge-connected* if G remains connected after the removal of any $k - 1$ edges. A *vertex cut* is a set of vertices whose removal renders G disconnected, and an *edge cut* is a set of edges whose removal renders G disconnected. For 2-connected and 3-connected graphs, we also use the terms *biconnected* and *triconnected*, respectively. If a vertex cut consists of a single vertex, we refer to it as a *cutvertex*. Analogously, if an edge cut consists of a single edge, we refer to it as a *bridge*. If an edge cut forms a cycle, we refer to it as *separating cycle*. A separating cycle of length 3 is called *separating triangle*.

A graph that contains no simple cycle is called a *forest*. A connected forest is known as a *tree*. It is well-known that a tree contains exactly $n - 1$ edges, and that there is a unique simple path between every pair of vertices. A tree $T = (V, E)$ is usually considered as *rooted* in a *root* $r \in V$. In a rooted tree, the edges are directed such that $\mathrm{indeg}(r) = 0$ and $\mathrm{indeg}(v) = 1$ for every other vertex $v \in V \setminus \{r\}$. A vertex $v \in V$ is called a *leaf* if $\mathrm{indeg}(v) = 1$ and $\mathrm{outdeg}(v) = 0$. If (u, v) is an edge of T, we call u a *parent* of v and v a *child* of u. Similarly, we call u an *ancestor* of v and v a *successor* of u if there exists a (directed) path from u to v. We define a *subtree* $T[v]$ of a tree to be the induced subgraph $T[V']$ with $V' = \{u \mid u \text{ is a successor of } v\}$. Consequently, v is the root of $T[v]$.

A *caterpillar* is a tree that consists of a path, called *spine*, and a set of leaves that are connected to the spine. An edge between a leaf and a spine vertex is called a *leg*. Another subclass of trees are *stars*, which consist of a set of leaves and a single vertex that is connected to all leaves, called *center*. A *wheel* consists of a star and a cycle that connects all leaves of the star.
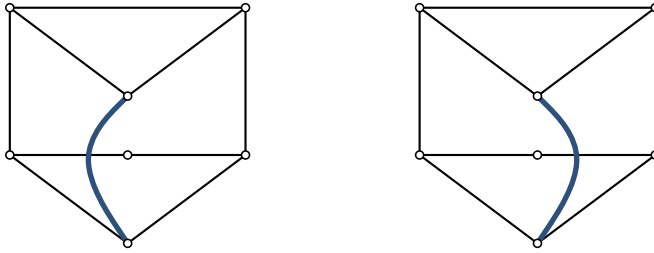
Figure 2.1: Two different IC-planar embeddings of the same graph with the same rotation system

A graph $G = (V, E)$ is called *bipartite* if the set of vertices can be decomposed into two disjoint sets V_1 and V_2 with $V = V_1 \cup V_2$ and $V_1 \cap V_2 = \varnothing$, such that all edges are incident to a vertex of each set: $E \subseteq \{(u, v) \mid u \in V_1, v \in V_2\}$. A *matching* is a bipartite graph of maximum degree 1.

2.2 Graph Drawing

A *drawing* of a graph $G = (V, E)$ is a mapping Γ of G into the plane $\mathbb{R}^2$. Each vertex $v \in V$ is mapped to a distinct point $\Gamma(v)$, that is, $\Gamma(u) \neq \Gamma(v)$ for any other vertex $u \neq v$. Each edge $e = (u, v) \in E$ is mapped to a simple Jordan curve $\Gamma(e)$ between the endpoints $\Gamma(u)$ and $\Gamma(v)$ of e. The relative interior of the curve $\Gamma(e)$ does not contain $\Gamma(w)$ for any vertex $w \in V$.

We say that two edges e_1 and e_2 *cross* in a drawing Γ if the relative interiors of $\Gamma(e_1)$ and $\Gamma(e_2)$ share at least one point. We call e_1 and e_2 *crossing edges* and the common point a *crossing point*. We do not allow edges to *overlap*, that is, to share an infinite number of points. A drawing is *planar* if it has no crossing edges, and *1-planar* if each edge is crossed at most once. An *IC-planar* drawing is a special case of a 1-planar drawing where each vertex is incident to at most one crossing edge. Following Euler's formula, a planar graph has at most $3n - 6$ edges, a 1-planar graph has at most $4n - 8$ edges, and an IC-planar graph has at most $13n/4 - 6$ edges.

A drawing Γ of a graph G induces an *embedding* $\mathcal{E}(G)$, which is the class of topologically equivalent drawings. An embedding specifies *faces*, whose boundary consists of a cyclic sequence of edges between two vertices and of segments of edges from a vertex to a crossing point or from a crossing point to a crossing point. We say that a vertex u is *incident* to a face f in $\mathcal{E}(G)$ if u lies on the boundary of f. Incident edges are defined analogously. A triconnected planar graph has a unique planar embedding: the faces of the embedding are exactly the non-separating cycles of the graph.

An embedding can also be defined by a *rotation system*, that is, the circular order of the incident edges around a vertex. Note that both definitions are equivalent for planar graphs. For non-planar graphs, the rotation system can directly be retrieved from a drawing or an embedding. The converse does not necessarily hold, as shown in Figure 2.1.

In every drawing, there is exactly one face of infinite area, called the *outer face*. All other faces, called *inner faces*, are bounded by the drawing of their surrounding edges. For *outer-planar graphs*, an embedding exists in which all vertices are incident to the outer face. Note

that every drawing induces a unique embedding, but for every embedding there is an infinite number of realizing drawings.

The planarization $G^\times$ of an embedding with crossings is obtained by treating the crossing points of two edges as specialized vertices of degree at least 4 and the edge segments as new edges. The planar embedding of $G^\times$ is inherited from $\mathcal{E}(G)$, and the embeddings can be taken as synonyms, in particular for an algorithmic treatment. In general, we assume that no more than two edges cross in a single point.

Let $\mathcal{G}$ be a graph property, for example, planar, 1-planar, IC-planar, or RAC. An embedding is *maximal-$\mathcal{G}$* if no further edge can be added without violating $\mathcal{G}$. We call the operation of adding edges to make a graph maximal-$\mathcal{G}$ a *maximal-$\mathcal{G}$ augmentation*. A graph $G \in \mathcal{G}$ is maximal-$\mathcal{G}$ if every $\mathcal{G}$-embedding is maximal-$\mathcal{G}$. Such graphs are maximally dense since the addition of an edge destroys the defining property.

An embedding $\mathcal{E}(G)$ of a planar graph G, called *primal graph*, induces a *dual graph G^**. The dual graph consists of a vertex corresponding to each face in $\mathcal{E}(G)$, and an edge between each pair of neighboring faces. This property is symmetric, meaning that G is the dual graph of G^*. Since any triconnected graph has a unique embedding, it induces a unique dual graph. A *weak dual* is the subgraph of the dual graph after removal of the vertex corresponding to the outer face. The weak dual can be used to get an alternative definition for outerplanar graphs: A graph is outerplanar if and only if its weak dual is a tree. Similarly, we can define an *outerpath* as a graph whose weak dual is a path.

The *drawing style* or *layout* determines the types of curves used to draw the edges. In *straight-line* or *geometric drawings*, every edge is represented by the straight-line segment that connects the representation of its endpoints. In *polyline drawings*, each edge consists of a finite number of edge segments. The touching points of these segments are called *bends*. A special type of polyline drawings are *orthogonal drawings*, in which every segment has to be horizontal or vertical. Note that orthogonal drawings restrict vertex degrees to at most 4. In *circular-arc drawings*, every edge is drawn as a segment of a circle. A *(strictly) convex drawing* is a drawing in which every face is (strictly) convex. A *monotone drawing* is a straight-line drawing such that, for every pair of vertices, there exists a path that monotonically increases with respect to some direction. A drawing is *strongly monotone* if this direction is the vector between the representation of the two vertices.

For planar graphs, many quality measures are well-studied. The most common measure is the *drawing area*. If a graph is drawn such that all its vertices lie on the integer grid, the area is defined by the *width*, that is, the difference between the extremal x-coordinates, and the *height*, that is, the difference between the extremal y-coordinates. If the vertices are not drawn on the integer grid, the area can be measured as the maximum distance over all pairs of vertices over the minimum distance over all pairs of vertices. Note that the area is bounded from below by the ratio between the length of the longest and the length of the shortest edge in the drawing. Another important quality measure is the *angular resolution* of a drawing. The angular resolution measure is defined as the smallest angle formed by any pair of edges that meet at a common vertex. For a graph with maximum degree d, a drawing has *perfect angular resolution* if the angular resolution is $2\pi/d$.

For non-planar graphs, there are different approaches to make the drawing appealing. In general, a drawing with many crossings is harder to comprehend than a drawing with fewer crossings. However, minimizing the number of crossings is NP-hard [GJ83], even for graphs

that become planar after removal of a single edge [CM13]. Another optimization criterion is the maximization of *crossing angles*, that is, the angles between two edges in a crossing point. A drawing in which all crossing angles are optimal, that is, 90°, is called a *right-angle crossing* (Rac) drawing.

2.3 Complexity

For algorithms and problems on graphs, we often analyze the complexity. There are two fundamental types of problems. *Decision problems* are yes-or-no questions, for example, whether a graph is planar. In *optimization problems*, we do not look for the existence of a solution, but for a *best* solution according to a given *target function*, for example, a drawing of a graph with the minimum number of crossings. In a *maximization problem*, a best solution is one that maximizes the value of the target function. Analogously, in a *minimization problem*, a best solution is one that minimizes the value of the target function.

When talking about the running time of an algorithm, we usually analyze the *asymptotic running time*, which is described by the *big O notation*. Let $f : \mathbb{N} \to \mathbb{N}$ be a function that maps the input size of a problem to the time that the algorithm requires to solve the problem. Then, we denote by

$$O(f) = \{g : \mathbb{N} \to \mathbb{N} \mid \exists c > 0, n_0 \in \mathbb{N}\ \ \forall n \geq n_0 : g(n) \leq c \cdot f(n)\}$$

the class of functions that asymptotically grow at most as fast as f. We say that the running time g of an algorithm *lies in the order* of f if $g \in O(f)$.

If the running time of an algorithm lies in $O(n^k)$ for some constant $k > 0$, then the algorithm is called a *polynomial-time* or *efficient* algorithm. The class P consists of all decision problems for which a deterministic algorithm exists that solves the problem in polynomial time. Similarly, the class NP consists of all decision problems for which a nondeterministic algorithm exists that solves the problem in polynomial time. It holds that P ⊆ NP, however, it is a long standing open problem whether P = NP or P ⊊ NP.

A problem A is NP-*hard* if every problem $B \in$ NP can be *reduced* to A in polynomial time. A reduction from B to A is a transformation that maps any instance J of B to an instance I of A such that there is a valid solution for J if and only if there is a valid solution for I. Consequently, if we can solve one NP-hard problem in polynomial time, then we can solve every problem in NP efficiently. Since the common conjecture is that P ≠ NP, most people do not expect that an NP-hard problem can be solved efficiently. A problem is NP-*complete* if it is NP-hard and in NP.

An NP-hard problem is *weakly* NP-hard if there is an algorithm for the problem with running time polynomial in the dimension of the problem and the magnitude of the input numbers if they are encoded in unary. Such an algorithm is called *pseudo-polynomial*; see Garey and Johnson [GJ79]. On the contrary, a problem is *strongly* NP-hard if it remains NP-hard even when the magnitude of the input numbers are encoded in unary; see Garey and Johnson [GJ78]. Weak and strong NP-completeness are defined analogously.

Usually, NP-hardness is proven by a reduction from a known NP-hard problem. In this book, we use the following NP-hard problems for NP-hardness proofs.

Sat The *Boolean satisfiability problem* (SAT) is the first problem known to be NP-complete, and is used as a starting point for proving NP-hardness of any other problem [Coo71]. A *SAT formula* is built from Boolean *variables*, the operators AND (denoted by $\land$), OR (denoted by $\lor$), and NOT (denoted by $\neg$). A SAT formula is called *satisfiable* if it is `true` for some assignment of Boolean values to the variables. SAT is the problem of deciding whether a given SAT formula is satisfiable.

A *literal l* is either a variable x or the negation of a variable $\neg x$. A *clause c* is a disjunction of literals $l_1, \ldots, k_k$, that is, $c = (l_1 \lor \ldots \lor l_k)$. A formula F is in *conjunctive normal form* if it is a conjunction of clauses $c_1, \ldots, c_m$, that is, $F = c_1 \land \ldots \land c_m$. It is known that every SAT formula can be converted into an equivalent formula in conjunctive normal form. However, the size of the formula can rise exponentially by this conversion.

3Sat The *3-satisfiability problem* (3SAT) is a special version of SAT. A *3SAT formula* is a formula in conjunctive normal form in which every clause consists of at most three literals. This problem is also NP-complete; see Karp's famous list of 21 NP-complete problems [Kar72]. On the other hand, the *2-satisfiability problem* (2SAT), in which every clause consists of at most two literals, can be solved efficiently; see Krom [Kro67].

Planar-3Sat The *planar 3-satisfiability problem* (PLANAR-3SAT) is a version of 3SAT that is often used to prove NP-hardness of planar graph drawing problems. An instance of 3SAT with a set X of variables and a set C of clauses is in PLANAR-3SAT if and only if the bipartite graph $G_{XC} = (X \cup C, E_{XC})$ with

$$E_{XC} = \{(x, c) \mid x \in c \text{ or } \neg x \in c, x \in X, c \in C\}$$

is planar. Thus, there is an edge connecting a variable $x \in X$ to a clause $c \in C$ if x or $\neg x$ is a literal of c.

1-Planarity testing 1-PLANARITY TESTING is the problem of testing whether a given graph is 1-planar. It was shown by Grigoriev and Bodlaender [GB07] to be NP-complete. The problem remains NP-complete even if the input graph is a maximal planar graph with one additional edge [CM13].

Recall that NP-hard problems are not expected to admit efficient algorithms. However, for optimization problems, we can still try to find a feasible solution that is close to optimal.

A usual approach for this is an *approximation algorithm*. For an instance I of an optimization problem Π, suppose that the value of an optimal solution is OPT_I, and there is an algorithm that finds a solution with value ALG_I. If Π is a minimization problem, then this algorithm is an α-approximation algorithm if there is a factor $\alpha > 1$ such that $\text{ALG}_I \leq \alpha \cdot \text{OPT}_I$ for every instance I of Π. Analogously, if Π is a maximization problem, then this algorithm is an α-approximation algorithm if there is a factor $0 < \alpha < 1$ such that $\text{ALG}_I \geq \alpha \cdot \text{OPT}_I$ for every instance I of Π. We call the algorithm an α-*approximation*, and α is the *approximation factor* of the algorithm. Note that α does not necessarily have to be a constant; there are also approximation algorithm whose approximation factor depends on the input size. If α is a constant, we call the algorithm a *constant-factor approximation algorithm*.

A *polynomial-time approximation scheme* (PTAS) is an approximation algorithm that, given an instance of an optimization problem and a factor $\varepsilon > 0$, solves the problem with approximation factor $1 + \varepsilon$. The running time of a PTAS is required to be polynomial in the input size for every fixed ε. Usually, the running time heavily depends on ε. For example, a PTAS can have a running time of $n^{O(1/\varepsilon)}$. The term PTAS can also be used to refer to the class of optimization problems that have a PTAS.

The class APX is the set of optimization problems for which a constant-factor approximation algorithm with polynomial running time exists. A problem A is called APX-*hard* if there is a PTAS reduction from every problem $B \in$ APX to A, that is, a transformation that maps any instance J of B to an instance I of A such that a $(1 + \varepsilon)$-approximation to I yields a $(1 + f(\varepsilon))$-approximation to J, for some surjective function $f : \mathbb{R}^+ \to \mathbb{R}^+$. A problem is APX-*complete* if it is APX-hard and in APX. Under the assumption that P $\neq$ NP, it holds that PTAS $\neq$ APX, that is, no APX-hard problem has a PTAS. In this book, we use the following APX-hard problems as starting points of APX-hardness reduction.

Gap The *Generalized Assignment Problem* (Gap) is defined as follows. Given a set X of items $x_1, \ldots, x_n$ and a set B of bins $b_1, \ldots, b_n$. Each bin b_i is associated with a budget w_i, and each item x_j has a profit p_{ij} and a weight w_{ij} depending on the bin it is placed in. A solution to Gap is an assignment of each item into at most one bin, such that, for each bin b_i, the sum of the weights w_{ij} of the items that are assigned to the bin does not exceed its budget w_i. The profit of a solution is the sum of the profits p_{ij} for each item–bin assignment. The goal is to find a solution with maximum profit. Chekuri and Khanna [CK05] proved that Gap is APX-hard.

3DM The *3-dimensional matching problem* (3DM) is a generalization of the efficiently solvable bipartite matching problem, that is, finding a maximum matching in a bipartite graph. In 3DM, the input is not a graph as we know it, but a 3-dimensional *hypergraph*, that is, an edge does not connect two but three vertices. The value of a solution is the number of picked hyperedges. More formally, an instance of this problem is given by three disjoint sets X, Y, Z with cardinalities $|X| = |Y| = |Z| = k$ and a set $E \subseteq X \times Y \times Z$ of hyperedges. The objective is to find a matching $M \subseteq E$, such that no element of $V = X \cup Y \cup Z$ is contained in more than one hyperedge in M and such that $|M|$ is maximized. This problem was shown to be APX-hard by Fleischer et al. [FGMS11].
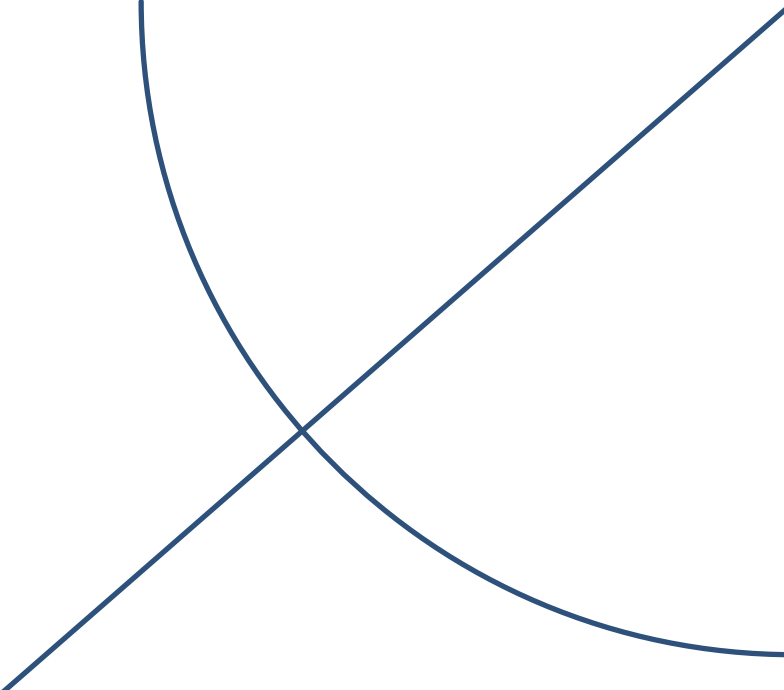
Part I

Placement of Boxes

3   Multi-Sided Boundary Labeling

Label placement is an important problem in cartography and, more generally, information visualization. Features such as points, lines, and regions in maps, diagrams, and technical drawings often have to be labeled so that users understand better what they see. Even very restricted versions of the label-placement problem are NP-hard [vKSW99], which explains why labeling a map manually is a tedious task that has been estimated to take 50% of total map production time [Mor80]. The ACM Computational Geometry Impact Task Force report [Cc99] identified label placement as an important research area. The point-labeling problem in particular has received considerable attention, from practitioners and theoreticians alike. The latter have proposed approximation algorithms for various objectives (label number versus label size), label shapes (such as axis-parallel rectangles or disks), and label-placement models (so-called fixed-position models versus slider models).

The traditional label-placement models for point labeling require that a label is placed such that a point on its boundary coincides with the point to be labeled, the *site*. This can make it impossible to label all sites with labels of sufficient size if some sites are very close together. For this reason, Freeman et al. [FMC96] and Zoraster [Zor97] advocated the use of *leaders*, (usually short) line segments that connect sites to labels. In order to ensure that the background image or map remains visible even in the presence of large labels, Bekos et al. [BKSW07] took a more radical approach. They introduced models and algorithms for *boundary labeling*, where all labels are placed beyond the boundary of the map and are connected to the sites by straight-line or rectilinear leaders (see Figure 3.1).

Problem statement. Following Bekos et al. [BKSW07] we define the Boundary Labeling problem as follows. We are given an axis-parallel rectangle $R = [0, W] \times [0, H]$, which is called the *enclosing rectangle*, a set $P \subset R$ of n points $p_1, \ldots, p_n$, called *sites*, within the rectangle R, and a set L of $m \leq n$ axis-parallel rectangles $\ell_1, \ldots, \ell_m$ of equal size, called *labels*. The labels lie in the complement of R and touch the boundary of R. No two labels overlap. We denote an instance of the problem by the triplet (R, L, P). A *solution* of a problem instance is a set of m curves $c_1, \ldots, c_m$ in the interior of R, called *leaders*, that connect sites to labels such that the leaders

   (a) induce a matching between the labels and (a subset of) the sites,

   (b) touch the associated labels on the boundary of R.

Following previous work, we do not define labels as the text associated with the sites, but as the empty rectangles into which that text will be placed (during a post-processing step). This approach is justified by our assumption that all label rectangles have the same size.

A solution is *planar* if the leaders do not intersect. We call an instance *solvable* if a planar solution exists. Note that we do not prescribe which site connects to which label. The endpoint

(a) straight-line leaders

(b) *opo*-leaders

(c) *po*-leaders
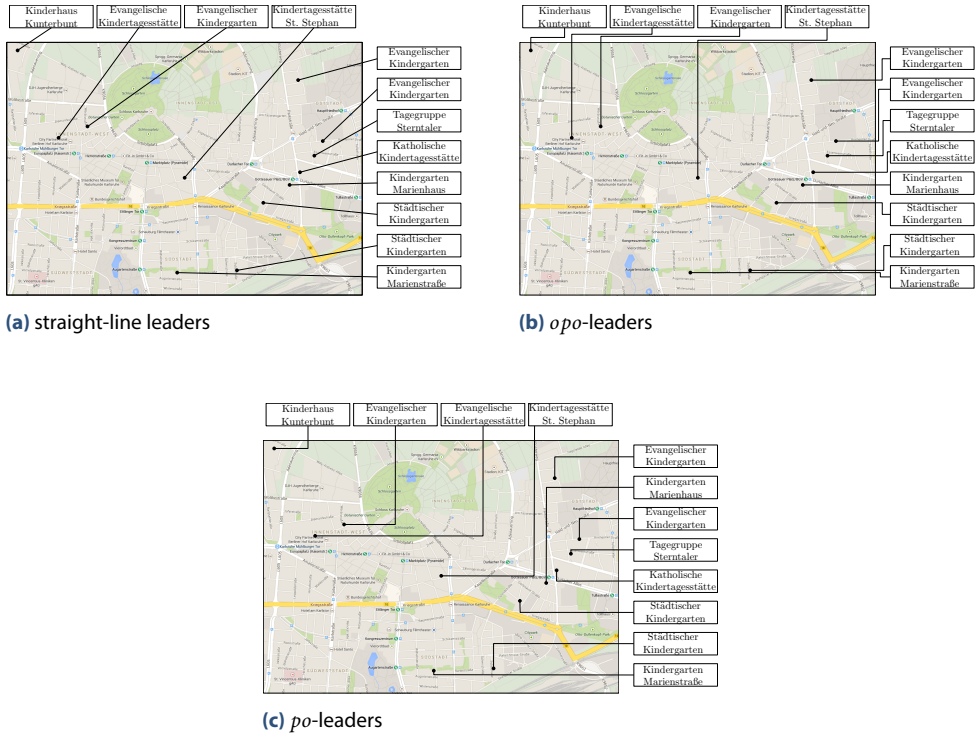
Figure 3.1: Labeling of kindergartens in Karlsruhe, Germany. The pictures show different types of leaders with labels on adjacent sides of the map. For better readability, we have simplified the label texts.

of a curve at a label is called a *port*. We distinguish two versions of the BOUNDARY LABELING problem: either the position of the ports on the boundary of R is fixed and part of the input, or the ports *slide*, that is, their exact location is not prescribed.

We restrict our solutions to *po-leaders*, that is, starting at a site, the first line segment of a leader is parallel (p) to the side of R touching the label it leads to, and the second line segment is orthogonal (o) to that side; see Figure 3.1c. (Figure 3.1b shows a labeling with so-called *opo-leaders*, which were investigated by Bekos et al. [BKSW07]). Bekos et al. [BKPS10, Figure 16] observed that not every instance admits a planar solution with *po*-leaders in which all sites are labeled (even if $m = n$).

Previous and related work.   For *po*-labeling, Bekos et al. [BKSW07] gave a simple quadratic-time algorithm for the one-sided case that, in a first pass, produces a labeling of minimum total leader length by matching sites and ports from bottom to top. In a second pass, their algorithm removes all intersections without increasing the total leader length. This result was improved by Benkert et al. [BHKN09] who gave an $O(n \log n)$-time algorithm for the same objective function and an $O(n^3)$-time algorithm for a very general class of objective functions, including, for example, bend minimization. They extend the latter result to the two-sided case (with labels on opposite sides of R), resulting in an $O(n^8)$-time algorithm. For
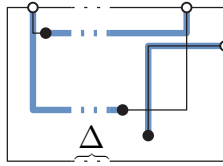
Figure 3.2: Length-minimal solutions may have crossings. By increasing Δ, we can make the ratio between the length-minimal matching and the length-minimal crossing-free matching arbitrarily small.

the special case of leader-length minimization, Bekos et al. [BKSW07] gave a simple dynamic program running in $O(n^2)$ time. All these algorithms work both for fixed and sliding ports.

Leaders that contain a diagonal part have been studied by Benkert et al. [BHKN09] and by Bekos et al. [BKNS10]. Recently, Nöllenburg et al. [NPS10] have investigated a dynamic scenario for the one-sided case, Gemsa et al. [GHN11] have used multi-layer boundary labeling to label panorama images, and Fink et al. [FHS$^+$12] have boundary labeled focus regions, for example, in interactive on-line maps. Lin et al. [LKY08] consider boundary labeling where more than one site may be labeled by the same label. Lin [Lin10] and Bekos et al. [BCF$^+$13] study hyperleaders that connect each label to a set of sites.

At its core, the boundary labeling problem asks for a non-intersecting perfect (or maximum) matching on a bipartite graph. Note that an instance may have a planar solution, although all of its leader-length minimal matchings have crossings. In fact, the ratio between a length-minimal solution and a length-minimal crossing-free matching can be arbitrarily bad; see Figure 3.2. When connecting points and sites with straight-line segments, the minimum Euclidean matching is necessarily crossing-free. For this case an $O(n^{2+\varepsilon})$-time $O(n^{1+\varepsilon})$-space algorithm exists [AES99].

Boundary labeling can also be seen as a graph-drawing problem where the class of graphs to be drawn is restricted to matchings. The restriction concerning the positions of the graph vertices (that is, sites and ports) has been studied for less restricted graph classes under the name *point-set embeddability (PSE)*, usually following the straight-line drawing convention for edges [GMPP91]. For polygonal edges, Bastert and Fekete [BF96] proved that PSE with minimum number of bends or minimum total edge length is NP-hard, even when the graph is a matching. For minimizing the total edge length and the same graph class, Liebling et al. [LMM$^+$95] introduced heuristics and Chan et al. [CHKL13] presented approximation algorithms. Chan et al. also considered paths and general planar graphs. PSE has also been combined with the ortho-geodesic drawing convention [KKRW10], which generalizes *po*-labeling by allowing edges to have more than one bend. The case where the mapping between ports and sites is given has been studied in VLSI layout [RCS86].

Our contribution.   In the first part of the chapter, we investigate the problem TWO-SIDED BOUNDARY LABELING WITH ADJACENT SIDES where all labels lie on two *adjacent* sides of R, without loss of generality, on the top and right side. Note that point data often comes in a coordinate system; then it is natural to have labels on adjacent sides (for example, opposite the coordinate axes). We argue that this problem is more difficult than the case where labels lie on opposite sides, which has been studied before: with labels on opposite sides, (a) there

is always a solution where all sites are labeled (if $m = n$) and (b) a feasible solution can be obtained by considering two instances of the one-sided case.

We present an algorithm that, given an instance with n labels with fixed ports and n sites, decides whether a planar solution exists where all sites are labeled and, if yes, computes a layout of the leaders (see Section 3.2). Our algorithm uses dynamic programming to "guess" a partition of the sites into the two sets that are connected to the leaders on the top side and on the right side. The algorithm runs in $O(n^2)$ time and uses $O(n)$ space.

We study several extensions of our main result (see Section 3.3). First, we show that our approach for fixed ports can also be made to work for sliding ports. Second, we optimally solve the label-number maximization problem (in $O(n^3 \log n)$ time using $O(n)$ space). This is interesting if the position of the sites and labels does not allow for a perfect matching or if there are more sites than labels. Finally, we present a modification of our algorithm that minimizes the leader length (in $O(n^8 \log n)$ time and $O(n^6)$ space).

In the second part of the chapter, we investigate the problems THREE-SIDED BOUNDARY LABELING and FOUR-SIDED BOUNDARY LABELING where the labels may lie on three or even all four sides of R, respectively. To that end we generalize the concept of partitioning the sites labeled by leaders of different sides. In this way we obtain subinstances that we can solve using the algorithm for the two-sided case. We obtain an algorithm solving the four-sided case in $O(n^9)$ time and $O(n)$ space and an algorithm solving the three-sided case in $O(n^4)$ time and $O(n)$ space. Except for the leader-length minimization, all extensions presented previously extend to the three- and four-sided case, of course with a corresponding impact on the running time and space requirements.

Notation.   We call the labels that lie on the right (left / top / bottom) side of R *right (left / top / bottom) labels*. The *type* of a label refers to the side of R on which it is located. The *type* of a leader (or a site) is simply the type of its label. We assume that no two sites lie on the same horizontal or vertical line, and no site lies on a horizontal or vertical line through a port or an edge of a label.

For a solution $\mathcal{L}$ of a boundary labeling problem, we define several measures that will be used to compare different solutions. We denote the total length of all leaders in $\mathcal{L}$ by $\mathrm{length}(\mathcal{L})$. Moreover, we denote by $|\mathcal{L}|_x$ the total length of all horizontal segments of leaders that connect a left or right label to a site. Similarly, we denote by $|\mathcal{L}|_y$ the total length of the vertical segments of leaders that connect top or bottom labels to sites. Note that generally, $|\mathcal{L}|_x + |\mathcal{L}|_y \neq \mathrm{length}(\mathcal{L})$.

We denote the (uniquely defined) leader connecting a site p to a port t of a label ℓ by $\lambda(p, t)$. We denote the bend of the leader $\lambda(p, t)$ by $\mathrm{bend}(p, t)$. In the case of fixed ports, we identify ports with labels and simply write $\lambda(p, \ell)$ and $\mathrm{bend}(p, \ell)$, respectively.

3.1 Structure of Two-Sided Planar Solutions

In this section, we tackle the two-sided boundary labeling problem with adjacent sides by presenting a series of structural results of increasing strength. We assume that the labels are located on the top and right sides of R. For simplicity, we assume that we have fixed ports. By identifying the ports with their labels, we use L to denote the set of ports of all labels. For sliding ports, we can simply fix all ports to the bottom-left corner of their corresponding
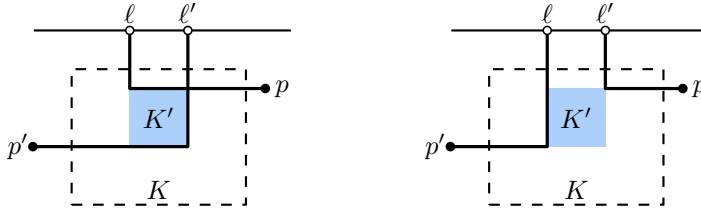
Figure 3.3: Illustration of the proof of Lemma 3.1. Rerouting $\lambda(p, \ell)$ and $\lambda(p', \ell')$ to $\lambda(p, \ell')$ and $\lambda(p', \ell)$ changes leaders only on the boundary of K'.

labels. First we show that a planar two-sided solution admits a transformation sustaining planarity such that the result of the transformation can be split into two one-sided solutions by constructing an xy-monotone, rectilinear curve from the top-right to the bottom-left corner of R; see Figure 3.4. Afterwards, we provide a necessary and sufficient criterion to decide whether there exists a planar solution for a given separation. This will form the basis of our dynamic programming algorithm, which we present in Section 3.2.

Lemma 3.1. *Consider a solution $\mathcal{L}$ for (R, L, P) and let $P' \subseteq P$ be sites of the same type. Let $L' \subseteq L$ be the set of labels of the sites in P'. Let $K \subseteq R$ be a rectangle that contains all bends of the leaders of P'. If the leaders of $P \smallsetminus P'$ do not intersect K, then we can rematch P' and L' such that the resulting solution $\mathcal{L}'$ has the following properties:*

(i) all intersections in K are removed,

(ii) there are no new intersections of leaders outside of K,

(iii) $|\mathcal{L}'|_x = |\mathcal{L}|_x$, $|\mathcal{L}'|_y = |\mathcal{L}|_y$, and

(iv) $\text{length}(\mathcal{L}') \leq \text{length}(\mathcal{L})$.

Proof. Without loss of generality, we assume that P' contains top sites; the other cases are symmetric. We first prove that, no matter how we change the assignment between P' and L', new intersection points can arise only in K. This enables us to construct the required solution.

Claim 1. Let $\ell, \ell' \in L'$ and $p, p' \in P'$ such that ℓ labels p and ℓ' labels p'. Changing the matching by rerouting p to ℓ' and p' to ℓ does not introduce new intersections outside of K.

Let $K' \subseteq K$ be the rectangle spanned by $\text{bend}(p, \ell)$ and $\text{bend}(p', \ell')$. When rerouting, we replace $\lambda(p, \ell) \cup \lambda(p', \ell')$ restricted to the boundary of K' by its complement with respect to the boundary of K'; see Figure 3.3 for an example. Thus, any changes concerning the leaders occur only in K'. The statement of the claim follows.

Since any rerouting can be seen as a sequence of pairwise reroutings, the above claim shows that we can rematch L' and P' arbitrarily without running the risk of creating new conflicts outside of K. To resolve the conflicts inside K, we use the length-minimization algorithm for one-sided boundary labeling by Benkert et al. [BHKN09], with the sites and ports outside K projected onto the boundary of K. Thus, we obtain a solution $\mathcal{L}'$ satisfying properties (i)–(iv). $\qquad\square$
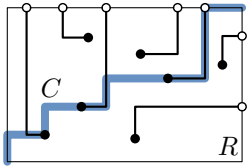
Figure 3.4: An xy-separating curve of a planar solution
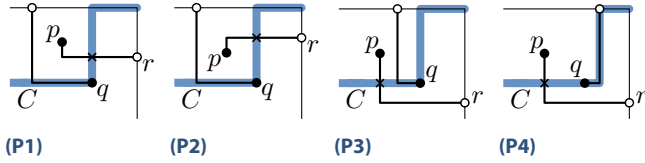
Figure 3.5: A planar solution that contains any of the above four patterns (P1)–(P4) is not xy-separated.

Definition 3.1. We call an xy-monotone, rectilinear curve connecting the top-right to the bottom-left corner of R an xy-separating curve. We say that a planar solution to TWO-SIDED BOUNDARY LABELING WITH ADJACENT SIDES is xy-separated if and only if there exists an xy-separating curve C such that

(a) the sites that are connected to the top side and all their leaders lie on or above C

(b) the sites that are connected to the right side and all their leaders lie below C.

It is not hard to see that a planar solution is not xy-separated if there exists a site p that is labeled to the right side and a site q that is labeled to the top side with $x(p) < x(q)$ and $y(p) > y(q)$. There are exactly four patterns in a possible planar solution that satisfy this condition; see Figure 3.5. In Lemma 3.2, we show that these patterns are the only ones that can violate xy-separability.

Lemma 3.2. *A planar solution is xy-separated if and only if it does not contain any of the patterns P1–P4 in Figure 3.5.*

Proof. Obviously, the planar solution is not xy-separated if one of these patterns occurs. Let us assume that none of these patterns exists. We construct an xy-monotone curve C from the top-right corner of R to its bottom-left corner. We move to the left whenever possible, and down only when we reach the x-coordinate of a site p that is connected to the top, or when we reach the x-coordinate of a port of a top label, labeling a site p. If we have to move down, we move down as far as necessary to avoid the corresponding leader, namely down to the y-coordinate of p. Finally, when we reach the left boundary of R, we move down to the bottom-left corner of R. If C is free of crossings, then we have found an xy-separating curve. For an example, see curve C in Figure 3.4.

Assume for a contradiction, that a crossing arises during the construction, and consider the topmost such crossing. Note that, by the construction of C, crossings can only occur with leaders that connect a site p to a right port r. We distinguish two cases, based on whether the crossing occurs on a horizontal or a vertical segment of the curve C.

If C is crossed on a vertical segment, then this segment belongs to a leader connecting a site q to a top port t, and we have reached the x-coordinate of either the port or the site. Had we, however, reached the x-coordinate of the port, this would imply a crossing between $\lambda(p, r)$ and $\lambda(q, t)$. Thus, we have reached the x-coordinate of q. This means that p lies to the left of and above q, and we have found one of the patterns (P1) and (P2); see Figure 3.5.

If C is crossed on a horizontal segment, then p must lie above r. Otherwise, there would be another crossing of C with the same leader, which is above the current one. This would

contradict the choice of the topmost crossing. Consider the previous segment of C, which is responsible for reaching the y-coordinate of the current segment. This vertical segment belongs to a leader connecting a site q to a top port t. Since leaders do not cross, q is to the right of p, and the crossing on C implies that q is below p. We have found one of the patterns (P3) and (P4); see Figure 3.5. □

Observe that patterns (P1) and (P2) can be transformed into patterns (P3) and (P4), respectively, by mirroring the instance diagonally. Next, we prove constructively that, by rerouting pairs of leaders, any planar solution can be transformed into an xy-separated planar solution.

Proposition 3.1. *If there exists a planar solution $\mathcal{L}$ to* Two-Sided Boundary Labeling with Adjacent Sides, *then there exists an xy-separated planar solution $\mathcal{L}'$ with* $\operatorname{length}(\mathcal{L}') \leq \operatorname{length}(\mathcal{L})$, $|\mathcal{L}'|_x \leq |\mathcal{L}|_x$, *and* $|\mathcal{L}'|_y \leq |\mathcal{L}|_y$.

Proof. Let $\mathcal{L}$ be a planar solution of minimum total leader length. We show that $\mathcal{L}$ is xy-separated. Assume, for the sake of contradiction, that $\mathcal{L}$ is not xy-separated. Then, by Lemma 3.2, $\mathcal{L}$ contains one of the patterns (P1)–(P4). Without loss of generality, we can assume that the pattern is of type (P3) or (P4). Otherwise, we mirror the instance diagonally.

Consider all patterns (p, q) in $\mathcal{L}$ of type (P3) or (P4) such that p is a right site (with port r) and q is a top site (with port t). Among all such patterns, consider the ones where p is rightmost and among these pick one where q is bottommost. Let A be the rectangle spanned by p and t; see Figure 3.6.. Let A' be the rectangle spanned by $\operatorname{bend}(q, t)$ and p. Let B be the rectangle spanned by q and r. Let B' be the rectangle spanned by q and $\operatorname{bend}(p, r)$. Then we claim the following:

(i) Sites in the interiors of A and A' are connected to the top.

(ii) Sites in the interiors of B and B' are connected to the right.

Property (i) is due to the choice of p as the rightmost site involved in such a pattern. Similarly, property (ii) is due to the choice of q as the bottommost site that forms a pattern with p. This settles our claim.

Our goal is to change the labeling by rerouting p to t and q to r, which decreases the total leader length, but may introduce crossings. We then use Lemma 3.1 to remove the crossings without increasing the total leader length. Let $\mathcal{L}''$ be the labeling obtained from $\mathcal{L}$ by rerouting p to t and q to r. We have $|\mathcal{L}''|_y \leq |\mathcal{L}|_y - (y(p) - y(q))$ and $|\mathcal{L}''|_x = |\mathcal{L}|_x - (x(q) - x(p))$. Moreover, $\operatorname{length}(\mathcal{L}'') \leq \operatorname{length}(\mathcal{L}) - 2(y(p) - y(q))$, as at least twice the vertical distance between p and q is saved; see Figure 3.6. Since the original labeling was planar, crossings may only arise on the horizontal segment of $\lambda(p, t)$ and on the vertical segment of $\lambda(q, r)$.

By properties (i) and (ii), all leaders that cross the new leader $\lambda(p, t)$ have their bends inside A', and all leaders that cross the new leader $\lambda(q, r)$ have their bends inside B'. Thus, we can apply Lemma 3.1 to the rectangles A' and B' to resolve all new crossings. The resulting solution $\mathcal{L}'$ is planar and has length less than $\operatorname{length}(\mathcal{L})$. This is a contradiction to the choice of $\mathcal{L}$. □

Since every solvable instance of Two-Sided Boundary Labeling with Adjacent Sides admits an xy-separated planar solution, it suffices to search for such a solution. Moreover, an

(a) pattern (P3)   **(b)** pattern (P4)
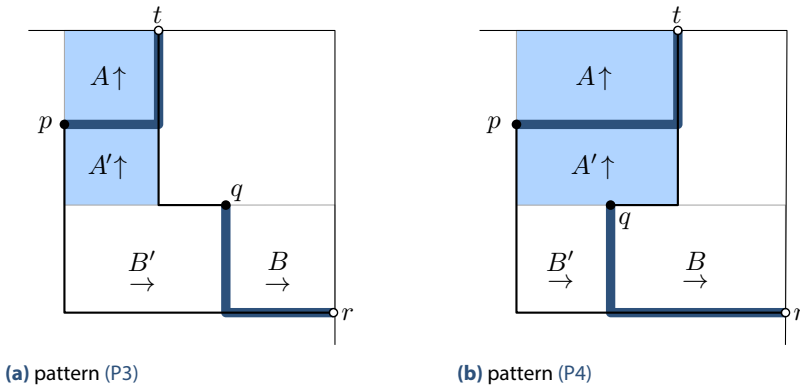
Figure 3.6: Types (top = ↑ / right = →) of the sites inside rectangles A, A', B, and B'

xy-separated planar solution that minimizes the total leader length is a solution of minimum length. In Lemma 3.3 we provide a necessary and sufficient criterion to decide whether, for a given xy-monotone curve C, there is a planar solution that is separated by C. We denote the region of R above C by R_T and the region of R below C by R_R. We do not include C in either R_T or R_R, so these regions are open at C.

For any point $a \in R$, we define the rectangle R_a, spanned by the top-right corner of R and a. We define R_a such that it is closed but does *not* contain its top-left corner. In particular, we consider the port of a top label as contained in R_a, only if it is not the upper left corner.

A rectangle R_a is *valid* if the number of sites of P above C that belong to R_a is at least as large as the number of ports on the top side of R_a. The central idea is that the labels on the top side of a valid rectangle R_a can be connected to the sites in R_a by leaders that are completely contained inside that rectangle. We are now ready to present the *strip condition*.

Condition 1. The *horizontal strip condition* of the point $b \in C$ is satisfied if there exists a point $a \in R_T$ with $y(a) = y(b)$ and $x(a) \leq x(b)$ such that R_a is valid.

Without loss of generality we may assume that the curve C is rectilinear. The condition is named after the horizontal segments through points in C.

We now prove that, for a given xy-monotone curve C connecting the top-right corner to the bottom-left corner of R, there exists a planar solution in R_T for the top labels if and only if all points of C satisfy the strip condition.

Lemma 3.3. *Let C be an xy-monotone curve from the top-right corner of R to the bottom-left corner of R. Let $P' \subseteq P$ be the sites that are in R_T. There is a planar solution that uses all top labels of R to label sites in P' in such a way that all leaders are in R_T if and only if each point of C satisfies the strip condition.*

Proof. For the proof we call a region $S \subseteq R$ *balanced* if it contains the same number of sites as it contains ports. To show that the conditions are necessary, let $\mathcal{L}$ be a planar solution for which all top leaders are above C. Consider a point $b \in C$. If $y(p) \geq y(b)$ for all sites $p \in P'$, rectangle R_a with $a = (0, y(b))$ is clearly valid, and thus the strip condition for b is satisfied.
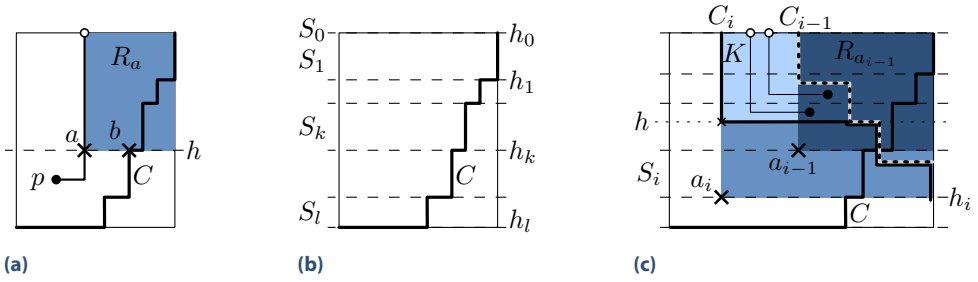
Figure 3.7: The strip condition. (a) The horizontal strip condition of b is satisfied by a. (b) The horizontal segments of C partition the strips $S_0, S_1, \ldots, S_k$. (c) Constructing a planar labeling from a sequence of valid rectangles.

Hence, assume that there is a site $p \in P'$ with $y(p) < y(b)$ that is labeled by a top label; see Figure 3.7a. Then, the vertical segment of this leader crosses the horizontal line h through b. Let a denote the rightmost such crossing of a leader of a site in P' with h. We claim that R_a is valid. To see this, observe that all sites of P' top-right of a are contained in R_a. Since no leader may cross the vertical segment defining a, the number of sites in $R_a \cap R_T$ is balanced, that is, R_a is valid.

Conversely, we show that if the conditions are satisfied, then a corresponding planar solution exists. For each horizontal segment of C consider the horizontal line through the segment. We denote the part of these lines within R by $h_1, \ldots, h_l$, respectively, and let h_0 be the top side of R. The line segments $h_1, \ldots, h_l$ partition R_T into l strips, which we denote by $S_1, \ldots, S_l$ from top to bottom, such that strip S_i is bounded by h_i from below for $i = 1, \ldots, l$; see Figure 3.7b. Additionally, we define S_0 to be the empty strip that coincides with h_0. Let S_k be the last strip that contains sites of P'. For $i = 0, \ldots, k - 1$, let a_i' denote the rightmost point of $h_i \cap R_T$ such that $R_{a_i'}$ is valid. Such a point exists since the leftmost point of $h_i \cap C$ satisfies the strip condition. We define a_i to be the point on $h_i \cap R_T$, whose x-coordinate is $\min_{j \leq i}\{x(a_j')\}$. Note that R_{a_i} is a valid rectangle, as, by definition, it completely contains some valid rectangle $R_{a_j'}$ with $x(a_j') = x(a_i)$. Also by definition the sequence formed by the points a_i has decreasing x-coordinates, that is, the R_{a_i} grow to the left; see Figure 3.7c.

We prove inductively that, for each $i = 0, \ldots, k$, there is a planar labeling $\mathcal{L}_i$ that matches the labels on the top side of R_{a_i} to points contained in R_{a_i}, in such a way that there exists an xy-monotone curve C_i from the top-left to the bottom-right corner of R_{a_i} that separates the labeled sites from the unlabeled sites without intersecting any leaders. Then $\mathcal{L}_k$ is the required labeling.

For $i = 0$, $\mathcal{L}_0 = \varnothing$ is a planar solution. Consider a strip S_i with $0 < i \leq k$; see Figure 3.7c. By the induction hypothesis, we have a curve C_{i-1} and a planar labeling $\mathcal{L}_{i-1}$, which matches the labels on the top side of $R_{a_{i-1}}$ to the sites in $R_{a_{i-1}}$ above C_{i-1}. To extend it to a planar solution $\mathcal{L}_i$, we additionally need to match the remaining labels on the top side of R_{a_i} and construct a corresponding curve C_i. Let P_i denote the set of unlabeled sites in R_{a_i}. By the validity of R_{a_i}, this number is at least as large as the number of unused ports at the top side of R_{a_i}. We arbitrarily match these ports to the topmost sites of P_i that are not labeled in $\mathcal{L}_{i-1}$. We denote the resulting labeling by $\mathcal{L}_i'$. We observe that no leader of $\mathcal{L}_i'$ crosses the curve C_{i-1},

and hence such leaders cannot cross leaders in $\mathcal{L}_{i-1}$. Let h be the topmost horizontal line such that all labeled sites of $\mathcal{L}_i'$ lie above h. Further, let K be the rectangle that is spanned by the top-left corner of $R_{a_{i-1}}$ and the intersection of h with the left side of R_{a_i}. Since the ports of $\mathcal{L}_i'$ lie on the top side of K, any leader's bend of $\mathcal{L}_i'$ lies in K. We apply Lemma 3.1 on $\mathcal{L}_i'$ to obtain a planar labeling $\mathcal{L}_i''$, which has no crossings with $\mathcal{L}_{i-1}$. Hence, the set $\mathcal{L}_i = \mathcal{L}_i'' \cup \mathcal{L}_{i-1}$ is the required labeling.

It remains to construct the curve C_i. For this, we start at the top-left corner of R_{a_i} and move down vertically, until we have passed all labeled sites. We then move right until we either hit C_{i-1} or the right side of R. In the former case, we follow C_{i-1} until we arrive at the right side of R. Finally, we move down until we arrive at the bottom-right corner of R_{a_i}. Note that all labeled sites are above C_i, unlabeled sites are below C_i, and no leader is crossed by C_i. This is true since we first move below the new leaders and then follow the previous curve C_{i-1}. □

A symmetric strip condition (with vertical strips) can be obtained for the right region R_R of a partitioned instance. The characterization is completely symmetric.

In the following we observe two properties of the strip condition. The first observation states that the horizontal strip condition at (x, y) is independent of the exact shape of the curve between the top-right corner r of R and (x, y), as long as the number of sites above the curve remains the same. This is crucial for using dynamic programming to test the existence of a suitable curve. The second observation states that the horizontal strip condition can only be violated when the curve passes the x-coordinate of a top site. This enables us to discretize the problem.

Observation 3.1. *The horizontal strip condition for a point $a \in C$ depends only on the number of sites in R_a above C, in the following sense: Let C and C' be two xy-monotone curves from r to a with u sites in R_a above C and C', respectively. Then, a satisfies the strip condition for C if and only if it satisfies the strip condition for C'.*

Observation 3.2. *Let $a, b \in C, x(a) \le x(b)$ such that there is no top site ℓ with $x(a) < x(\ell) \le x(b)$. Then, a satisfies the horizontal strip condition for C if and only if b satisfies the horizontal strip condition for C.*

Symmetric statements hold for the vertical strip condition. In the following, we say that a point (x, y) on a curve C satisfies the *strip condition* if it satisfies both the horizontal and the vertical strip condition.

3.2   Algorithm for the Two-Sided Case

How can we find an xy-monotone curve C that satisfies the strip conditions? For that purpose we only consider xy-monotone curves contained in some graph G that is dual to the rectangular grid induced by the sites and ports of the given instance. Note that this is not a restriction since all leaders are contained in the grid induced by the sites and ports. Thus, every xy-monotone curve that does not intersect the leaders can be transformed into an equivalent xy-monotone curve that lies on G.
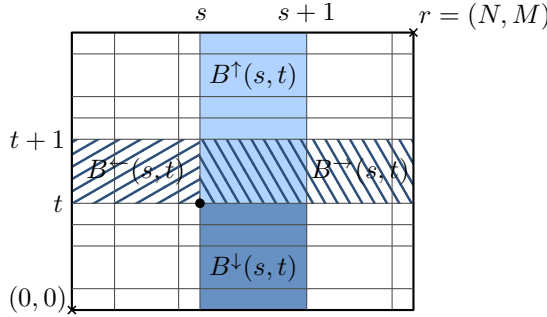
Figure 3.8: The four boxes $B^\uparrow(s,t)$, $B^\downarrow(s,t)$, $B^\leftarrow(s,t)$ and $B^\rightarrow(s,t)$ defined by grid point (s,t)

When traversing an edge e of G, we pass the x- or y-coordinate of exactly one entity of our instance; either a site (*site event*) or a port (*port event*). When passing a site, the position of the site relative to e (above/below e or right/left of e) decides whether the site is connected to the top or to the right side. Clearly, there is an exponential number of possible xy-monotone traversals through the grid. In the following, we describe a dynamic program that finds an xy-separating curve in $O(n^3)$ time.

Let m_R and m_T be the numbers of ports on the right and top side of R, respectively. Also, let $N = n + m_T + 2$ and $M = n + m_R + 2$, then the grid G has size $N \times M$. We define the grid points as $G(s,t)$, $0 \le s \le N$, $0 \le t \le M$ with $G(0,0)$ being the bottom-left and $r := G(N,M)$ being the top-right corner of R. Finally, let $G_x(s) := x(G(s,0))$ and $G_y(t) := y(G(0,t))$.

For each grid point (s,t) that is neither on the topmost row nor on the rightmost column, we define four boxes $B^\uparrow(s,t)$, $B^\downarrow(s,t)$, $B^\leftarrow(s,t)$ and $B^\rightarrow(s,t)$ as follows; see Figure 3.8 for an illustration.

1. $B^\uparrow(s,t) = \{(x,y) \in R \mid G_x(s) \le x \le G_x(s+1) \wedge y \ge G_y(t)\}$

2. $B^\downarrow(s,t) = \{(x,y) \in R \mid G_x(s) \le x \le G_x(s+1) \wedge y \le G_y(t)\}$

3. $B^\leftarrow(s,t) = \{(x,y) \in R \mid G_y(t) \le y \le G_y(t+1) \wedge x \le G_x(s)\}$

4. $B^\rightarrow(s,t) = \{(x,y) \in R \mid G_y(t) \le y \le G_y(t+1) \wedge x \ge G_x(s)\}$

We define a table $T[(s,t),u,b]$ that assigns to each grid position (s,t) and number of points u and b a Boolean value. We define $T[(s,t),u,b]$ to be true if and only if there exists an xy-monotone curve C satisfying the following conditions.

(i) Curve C starts at r and ends at $G(s,t)$.

(ii) Inside the rectangle spanned by r and $G(s,t)$, there are u sites of P above C and b sites of P below C.

(iii) For each grid point on C, the strip condition holds.

These conditions together with Proposition 3.1 and Lemma 3.3 imply that the instance admits a planar solution if and only if $T[(0,0),u,b] = \texttt{true}$ for some u and b.

We define a Boolean function $S[(s,t), u, b]$ that is true if and only if the strip condition at (s,t) is satisfied for some xy-monotone curve C (and thus by Observation 3.1 for all such curves) from r to $G(s,t)$ with u sites above and b sites below C. The following lemma gives a recurrence for T, which is essentially a disjunction of two values, each of which is determined by distinguishing three cases.

Lemma 3.4. *For $s = N$ and $t = M$, it holds that $T[(s,t), 0, 0] = \texttt{true}$. For $s \in [0, N-1]$ and $t \in [0, M-1]$, it holds that*

$$
T[(s,t), u, b] =
\begin{cases}
\begin{array}{lll}
T[(s+1,t), u, b] \wedge S[(s,t), u, b] & & L \cap B^{\uparrow}(s,t) \neq \varnothing \\
T[(s+1,t), u-1, b] & \text{if} & P \cap B^{\uparrow}(s,t) \neq \varnothing \\
T[(s+1,t), u, b] & & P \cap B^{\downarrow}(s,t) \neq \varnothing
\end{array}
\end{cases}
$$
$$
\vee
$$
$$
\begin{cases}
\begin{array}{lll}
T[(s,t+1), u, b] \wedge S[(s,t), u, b] & & L \cap B^{\rightarrow}(s,t) \neq \varnothing \\
T[(s,t+1), u, b-1] & \text{if} & P \cap B^{\rightarrow}(s,t) \neq \varnothing \\
T[(s,t+1), u, b] & & P \cap B^{\leftarrow}(s,t) \neq \varnothing
\end{array}
\end{cases}.
$$

Proof. We show equivalence of the two terms. Let C be an xy-monotone curve from r to (s,t). Let e be the last segment of C and let $C' = C - e$. Since C is xy-monotone, C' ends either at the grid point $(s+1, t)$ or at $(s, t+1)$. Without loss of generality, we assume that C' ends at $(s+1, t)$. We show that $T[(s,t), u, b] = \texttt{true}$ if and only if the first term of the right hand side is $\texttt{true}$. Analogous arguments apply for C' ending at $(s, t+1)$ and the second term. Note that, by construction, property (i) is satisfied for C and C'.

We distinguish cases based on whether the traversal along the segment e from $(s+1, t)$ to (s,t) is a port event or a site event.

Case 1: Traversal of e is a port event. Since e passes a port, all sites that lie in the rectangle spanned by r and $G(s,t)$ also lie in the rectangle spanned by r and $G(s+1, t)$. Thus, the numbers u and b of such sites above and below C is the same as the numbers of sites above and below C', respectively. Hence, property (ii) holds for C if and only if it holds for C'.

Because C' is a subset of C, the strip condition holds for every point of C if and only if it holds for every point of C' and for (s,t). Thus, property (iii) is satisfied for C if and only if it is satisfied for C' and $S[(s,t), u, b] = \texttt{true}$.

Case 2: Traversal of e passes a site p. For property (iii), observe that, since the traversal of e is a site event, the strip conditions for (s,t) and $(s+1, t)$ are equivalent by Observation 3.2.

For property (ii), note that, except for p, the sites that lie in the rectangle spanned by r and $G(s,t)$ also lie in the rectangle spanned by r and $G(s+1, t)$. If p lies above e, there are u sites above and b sites below C if and only if there are $u-1$ sites above and b sites below C', respectively. Symmetrically, if p lies below e, there are u sites above and b sites below C if and only if there are u sites above and $b-1$ sites below C', respectively. In either case, C satisfies condition (ii) if and only if C' does. $\qquad\square$

Clearly, the recurrence from Lemma 3.4 can be used to compute T in polynomial time via dynamic programming. Note that it suffices to store u, as the number of sites below the curve C can directly be derived from u and all sites that are contained in the rectangle spanned by r and $G(s,t)$. Thus, in the following we work with $T[(s,t), u]$. The running time crucially

relies on the number of strip conditions that need to be checked. We show that after a $O(n^2)$ preprocessing phase, such queries can be answered in $O(1)$ time.

To implement the test of the strip conditions, we use a table B_T, which stores in $B_T[s, t]$ how large a deficit of sites to the right can be compensated by sites above and to the left of $G(s, t)$. That is, $B_T[s, t]$ is the maximum value k such that there exists a rectangle $K_{B_T[s,t]}$ with lower right corner $G(s, t)$ whose top side is bounded by the top side of R, and that contains k more sites in its interior, than it has ports on its top side. Once we have computed this matrix, it is possible to query the strip condition in the dynamic program that computes T in $O(1)$ time as follows: Assume we have an entry $T[(s, t), u]$, and we wish to check its strip condition. Consider a curve C from r to $G(s, t)$ such that u sites are above C. The strip condition is satisfied if and only if $u + B_T[s, t]$ is at least as large as the number of top ports to the right of $G(s, t)$. This is true if the rectangle spanned by the lower left corner of $K_{B_T[s,t]}$ and r contains at least $u + B_T[s, t]$ sites, which is an upper bound on the number of ports on the top side of that rectangle.

We now show how to compute B_T in $O(n^2)$ time. We compute each row separately, starting from the left side. We initialize $B_T[0, t] = 0$ for $t = 0, \dots, M$, since in the final column, no deficit can be compensated. The matrix B can be filled by a horizontal sweep. The entry $B_T[s, t]$ can be derived from the already computed entry $B_T[s-1, t]$. If the step from $s-1$ to s is a site event, the amount of the deficit we can compensate increases by 1. If it is a port event the amount of the deficit we can compensate decreases by 1. Moreover, the compensation potential never goes below 0. We obtain

$$B_T[s, t] = \begin{cases} B_T[s-1, t] + 1 & \text{if step is site event,} \\ \max\{B_T[s-1, t] - 1, 0\} & \text{if step is port event.} \end{cases}$$

The table can be clearly filled out in $O(n^2)$ time. A similar matrix B_R can be computed for the vertical strips. Altogether, this yields an algorithm for TWO-SIDED BOUNDARY LABELING WITH ADJACENT SIDES that runs in $O(n^3)$ time and uses $O(n^3)$ space. However, the entries of each row and column of T depend only on the previous row and column, which allows us to reduce the storage requirement to $O(n^2)$. Using Hirschberg's algorithm [Hir75], we can still backtrack the dynamic program and find a solution corresponding to an entry in the last cell in the same running time. We have the following theorem.

Theorem 3.1. *TWO-SIDED BOUNDARY LABELING WITH ADJACENT SIDES can be solved in $O(n^3)$ time using $O(n^2)$ space.*

Our next goal is to improve the performance of our algorithm by reducing the number of dimensions of the table T by 1. As a first step, we show that for any search position $\mathbf{c} = (s, t)$, the set of all u with $T[\mathbf{c}, u] = \texttt{true}$ is an interval.

Lemma 3.5. *Let $T[\mathbf{c}, u] = T[\mathbf{c}, u'] = \texttt{true}$ with $u < u'$. Then, $T[\mathbf{c}, u''] = \texttt{true}$ for any $u \leq u'' \leq u'$.*

Proof. Let C be a curve corresponding to the entry $T[\mathbf{c}, u]$. That is C connects r to $\mathbf{c}$ such that any point on C satisfies the strip condition. Similarly, let C' be a curve corresponding to $T[\mathbf{c}, u']$; see Figure 3.9.

Since u and u' differ, there is a rightmost site p, such that p is below C and above C'. Let v and v' be the grid points of C and C' that are immediately to the left of p. Note that v is above v' since C is above p and C' is below it. Consider the curve C'' that starts at r and follows C until v, then moves down vertically to v', and from there follows C' to p. Obviously, C'' is an xy-monotone curve, and it has above it the same sites as C', except for p, which is below it. Thus there are $u'' = u' - 1$ sites above C'' in the rectangle spanned by p and r. If all points of C'' satisfy the strip condition, then this implies $T[\mathbf{c}, u''] = \texttt{true}$.

   We show that indeed the strip condition is satisfied for any point on C''. Let C_1 be the subcurve of C'' that connects r to v, let C_2 be the segment vv' and let C_3 be the subcurve of C'' that connects v' to $\mathbf{c}$. Since C_1 is also a subcurve of C and it starts at r, it directly follows that any point of C_1 satisfies the strip condition. For the points on C_2 we can argue as follows. Since C_2 lies below C and any point of C satisfies the horizontal strip condition, any point of C_2 must satisfy the horizontal strip condition. Analogously, because C_2 lies above C' and any point of C' satisfies the vertical strip condition, each point of C_2 must satisfy the vertical strip condition. Finally, since C_3 is a subcurve of C', any point of C' satisfies the strip condition and any point of C_1 and C_2 satisfies the strip condition, it directly follows that any point of C_3 satisfies the strip condition. □

Using Lemma 3.5, we can reduce the dimension of the table T by 1. It suffices to store at each entry $T[\mathbf{c}]$ the boundaries of the u-interval. This reduces the amount of storage to $O(n^2)$ without increasing the running time. Using Hirschberg's algorithm, the storage for T even decreases to $O(n)$. Tables B_T and B_R still have size $O(n^2)$, however.

   Our next goal is to reduce the running time to $O(n^2)$. An entry in $B_\mathrm{T}[s, t]$ tells us which deficits can be compensated. This can also be interpreted as a lower bound on the number of sites a curve from r to $G(s, t)$ must have above it, in order to satisfy the horizontal strip condition. Namely, let $\tau_{s,t}$ denote the number of ports on the top side of the rectangle spanned by $G(s, t)$ and r. Then $u \geq \tau_{s,t} - B_\mathrm{T}[s, t]$ is equivalent to satisfying the horizontal strip condition for the strip directly above $G(s, t)$.

   Similarly, the corresponding entry $B_\mathrm{R}[s, t]$ gives a lower bound on the number of sites below such a curve, which in turn, together with the number of sites contained in the rectangle spanned by $G(s, t)$ and r implies an *upper bound* on the number of sites above the curve. Thus, B_T, B_R, and the information on how many sites, top ports and right ports are in the rectangle spanned by $G(s, t)$ and r together imply a lower and an upper bound, and thus an interval of u-values, for which the horizontal and vertical strip conditions at $G(s, t)$ is satisfied. Hence, the program can simply intersect this interval with the union of the intervals obtained from $T[(s, t) - \Delta\mathbf{c}]$, where $\Delta\mathbf{c}$ has exactly one non-zero entry, which is 1. Consequently, the amount of work per entry of T is still $O(1)$. Note that by Lemma 3.5 the result of this computation is again an interval.

   Now, we turn to the space consumption. Hirschberg's algorithm [Hir75] immediately reduces the space consumption of T to $O(n)$. We would like to apply the same trick to B_T and to B_R. Recall that B_T is computed from left to right and B_R from bottom to top. Unfortunately, this is opposite to the order we use for computing T, where we proceed from top-right to bottom-left. We can fix this problem by running the dynamic programs for computing B_T and B_R backwards, by precomputing the entries of B_T and B_R on the top and right side, and then running the updates backwards. This allows us to use Hirschberg's algorithm, and the

algorithms can run in a synchronized manner such that at any point in time the required data is available, using only $O(n)$ space.

A new issue, however, appears. The update $B_T[s, t] = \max\{B_T[s-1, t] - 1, 0\}$ is not easily reversible. When running the dynamic program backwards, it is not clear whether $B_T[s, t] = 0$ implies $B_T[s-1, t] = 0$ or $B_T[s-1, t] = 1$ at a port step. To remedy this issue, fix a column s of the table corresponding to a port event and consider the circumstances under which $B_T[s-1, t] - 1 = -1$, that is, $B_T[s-1, t] = 0$. This implies that, for any rectangle K with lower right corner $G(s-1, t)$ whose top side is contained in the top side of R, there are at most as many sites in K as there are ports in the top side of K. Assume that this is the case for some fixed value t_0, that is, $B_T[s-1, t_0]$. Since the possible rectangles for an entry $B_T[s-1, t]$ with $t \geq t_0$ contain at most as many sites as the ones for $B_T[s-1, t_0]$, this implies $B_T[s-1, t_0] = B_T[s-1, t] = 0$ for all $t \geq t_0$. If on the other hand, t_0 is such that $B_T[s-1, t_0] > 0$, then the rectangles corresponding to $B_T[s-1, t]$ for $t < t_0$ contain at least as many sites as the ones for $B_T[s-1, t_0]$, and we have $B_T[s-1, t] \geq B_T[s-1, t_0]$ for $t < t_0$. Thus, there is a single gap t_0 such that, for any $t \geq t_0$, we have $B_T[s-1, t] = 0$ and, for any $t < t_0$, we have $B_T[s-1, t] > 0$; see Figure 3.10. Storing this gap for each column s that is a port event allows us to efficiently reverse the dynamic program. Note that storing one value per column only incurs $O(n)$ space overhead. Of course, the same approach works for the dynamic program computing B_R. Overall, we have shown the following theorem.

Theorem 3.2. *Two-Sided Boundary Labeling with Adjacent Sides can be solved in $O(n^2)$ time using $O(n)$ space.*

3.3 Extensions

The techniques we used to obtain Theorem 3.2 can be applied to solve a variety of different extensions of the two-sided labeling problem with adjacent sides. We now show how to

    (a) generalize to sliding ports instead of fixed ports,

    (b) maximize the number of labeled sites, and

    (c) minimize the total leader length in a planar solution.

3.3.1 Sliding Ports

First, observe that Proposition 3.1, which guarantees the existence of an xy-separated planar solution, also holds for sliding ports. The same proofs apply by conceptually fixing the ports of a given planar solution when applying the rerouting operations. The following lemma shows that, without loss of generality, we can simply fix all ports at the bottom-left corner of their corresponding labels. This immediately solves the problem.

Lemma 3.6. *If there exists an xy-separated planar solution $\mathcal{L}$ for the two-sided boundary labeling problem with adjacent sides and sliding ports, then there also exists an xy-separated planar solution $\mathcal{L}'$ in which the ports are fixed at the bottom left corners of the labels.*
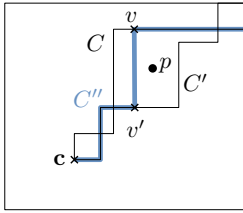
Figure 3.9: Illustration for the proof of Lemma 3.5
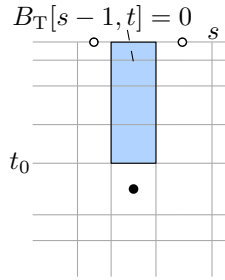
Figure 3.10: The gap t_0 is defined such that we have $B_{\mathrm{T}}[s-1,t] = 0$ for any $t \geq t_0$, and $B_{\mathrm{T}}[s-1,t] > 0$ for any $t < t_0$.
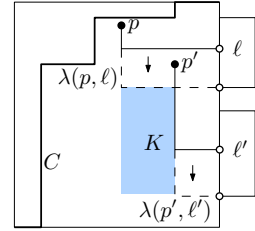
Figure 3.11: Illustration for the proof of Lemma 3.6

Proof. We show how to transform $\mathcal{L}$ into $\mathcal{L}'$. Let C be the xy-monotone curve that separates the top leaders from the right leaders of $\mathcal{L}$. We move the ports induced by $\mathcal{L}$ to the bottom-left corner of their corresponding labels such that the assignment between labels and sites remains; see Figure 3.11. Obviously, the bends of the leaders connected to the right side only move downwards. Thus, the leaders lie entirely below C. Symmetrically, the bends of the leaders connected to the top side only move to the left and thus these leaders lie entirely above C.

Consequently, only conflicts between the same type of leaders can arise. Consider the topmost intersection of two leaders $\lambda(p,\ell)$ and $\lambda(p',\ell')$ connected to the right side and assume that p lies to the left of p'. Let K be the rectangle that is spanned by the bends of $\lambda(p,\ell)$ and $\lambda(p',\ell')$. Due to moving the ports downwards, the leaders lie entirely below C and the bend of $\lambda(p',\ell')$ must lie below $\lambda(p,\ell)$. Hence, K lies completely in R_{R}. In order to resolve the conflict, we reroute p to ℓ' and p' to ℓ using the bottom-left corners of ℓ and ℓ' as ports. Obviously, the leaders only change on ∂K. Therefore, new conflicts can only arise on the left and bottom sides of K. In particular, only the leader of ℓ' can be involved in new conflicts, while the leader of ℓ is free of any conflict. Thus, after finitely many such steps we have resolved all conflicts, from top to bottom. Symmetric arguments apply for the leaders connected to the top side. □

Theorem 3.3. *Two-Sided Boundary Labeling with Adjacent Sides and Sliding Ports can be solved in $O(n^2)$ time using $O(n)$ space.*

3.3.2 Maximizing the Number of Labeled Sites

So far our algorithm only returns a leader layout if there is a planar solution that matches each label to a site. As Bekos et al. [BKPS10, Figure 16] observed, this need not always be the case, so it becomes important to be able to maximize the number of labels connected to sites in a planar solution. We achieve this by removing labels from a given instance and using our algorithm to decide whether a crossing-free solution exists.

Lemma 3.6 shows that we can move top ports to the left and right ports to the bottom without making a solvable instance unsolvable. Thus, it suffices to remove the rightmost top labels and the topmost right labels. Let k be the number of labels we want to use with k_{T} of them
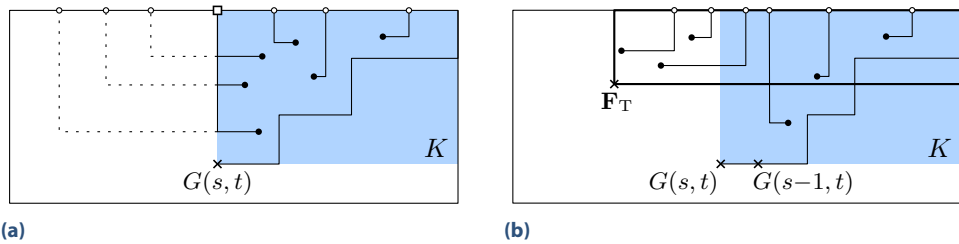
Figure 3.12: Illustration of the curve C and the rectangle K spanned by $G(s,t)$ and the top-right corner of R. (a) There are more sites than ports in K above C. The unlabeled sites are connected to a dummy port located at the top-left corner of K. The dummy port is illustrated as a square. (b) There are more ports than sites in K above C. The unlabeled ports are labeled to sites that lie to the left of K, which induce the front with bottom-left point $\mathbf{F}_\mathrm{T}$.

being top labels and k_R right labels, so that $k_\mathrm{T} + k_\mathrm{R} = k$. For a given k, we can decide whether a crossing-free solution that uses exactly k labels exists by removing the $m_\mathrm{T} - k_\mathrm{T}$ rightmost top labels and the $m_\mathrm{R} - k_\mathrm{R}$ topmost right labels for any possible k_T and k_R. We therefore start with $k_\mathrm{T} = \min\{k, m_\mathrm{T}\}$ and $k_\mathrm{R} = k - k_\mathrm{T}$. We keep decreasing k_T and increasing k_R by 1, until a crossing-free solution is found or $k_\mathrm{R} = \min\{k, m_\mathrm{R}\}$. In the latter case, no crossing-free solution that uses exactly k labels exists. With this approach we can use binary search to find the maximum k, using our algorithm up to k times per step. Since $k \le n$, this yields an algorithm for Two-Sided Boundary Labeling with Adjacent Sides that maximizes the number of labeled sites that runs in $O(n^3 \log n)$ time and uses $O(n)$ space.

Theorem 3.4. *Two-Sided Boundary Labeling with Adjacent Sides can be solved in $O(n^3 \log n)$ time using $O(n)$ space such that the number of labeled sites is maximized.*

Assume that t sites cannot be labeled. Then Two-Sided Boundary Labeling with Adjacent Sides can be solved in $O(n^2 t \log t)$ time using $O(n)$ space and such that the number of labeled sites is maximized. To that end we use the following approach. We check for $h = 2^i$ with $i = 0, 1, 2, \ldots$ whether there is a planar solution with h unlabeled sites. We stop this procedure when we have found such a solution, which takes place after $\lceil \log(t) \rceil$ steps. Using the approach described above, we need $O(n^2 t)$ time for each test. We then know that $\frac{h}{2} < t \le h$. We apply a binary search to determine t. Overall, this approach needs $O(n^2 t \log t)$ time.

3.3.3 Minimizing the Total Leader Length

Recall that, by Proposition 3.1, there always exists a length-minimal planar solution that is xy-separated. To obtain a length-minimal planar solution, we mainly change the table T used by the dynamic program given in Section 3.2. Let C be an xy-monotone curve C that starts at r and ends at $G(s,t)$. We assign to every table entry the length of the leaders that are connected to the ports in the rectangle K spanned by r and $G(s,t)$.

If there are more sites than top ports in K above C, we have to connect some of these sites to ports that lie to the left of K; see Figure 3.12a. The vertical lengths of their leaders, however, are fixed. We imagine a dummy top port at the left border of K and connect all unlabeled sites to this port. When traversing the grid horizontally, this dummy port moves to the left. In

order to update the total length of the leaders in K, we only have to keep track of the number of unlabeled sites and increase the horizontal length of their leaders. The sites in K below C are handled analogously.

If there are more top ports than sites in K above C, we have to connect these ports to sites that lie to the left of K; see Figure 3.12b. In order to remember which sites are already labeled, we store the *top front* as the rectangle with top-right corner r that includes all sites that are already connected to a top port inside K, and the *right front* as the rectangle with top-right corner r that includes all sites that are already connected to a right port inside K.

Let $\mathbf{F}_T = (x_T, y_T)$ be the bottom-left point of the top front for a given xy-monotone curve C that starts at r and ends at $G(s, t)$. Similarly, let $\mathbf{F}_R = (x_R, y_R)$ be the right front for C. We define $T[\mathbf{c} = (s, t), u, \mathbf{F}_T, \mathbf{F}_R] = (l, g_T, g_R)$ if there exists an xy-monotone curve C and leaders inside $K \cup F_T \cup F_R$ such that the following conditions hold, otherwise it contains $(-1, 0, 0)$.

(i) Curve C starts at the top-right corner r of R and ends at $G(s, t)$.

(ii) Inside the rectangle K spanned by r and $G(s, t)$, there are u sites of P above C.

(iii) For each strip in the two regions R_T and R_R defined by C the strip condition holds.

(iv) The sites in $K \cup \mathbf{F}_T \cup \mathbf{F}_R$ are connected to the ports on the border of $K \cup \mathbf{F}_T \cup \mathbf{F}_R$ such that the induced solution is planar, length-minimal, the sites above C or in $\mathbf{F}_T$ are only connected to top ports, and the sites below C or in $\mathbf{F}_R$ are only connected to right ports.

(v) There are g_T unlabeled top sites and g_R unlabeled right sites in K.

Note that g_T and g_R depend on s, t, u and can be precomputed, but to make the algorithm more intuitive, we update these values on-line and store them in T. We first describe how to handle the top front while traversing the grid.

Initially, $\mathbf{F}_T = G(s, t)$. As long as we have more top sites than top ports in K, we can connect all ports to sites and thus can maintain $\mathbf{F}_T = G(s, t)$. Once we have exactly the same number of top ports and top sites in K and we encounter a port event for a top port, we have to check the strip condition and find the rightmost point F_T with $y(\mathbf{F}_T) = G_y(t)$ such that the rectangle $R_{\mathbf{F}_T}$ spanned by $\mathbf{F}_T$ and r is valid. By storing $\mathbf{F}_T$, we know that all ports to the right of $x(\mathbf{F}_T)$ are already connected to a site, all sites to the top-right of $\mathbf{F}_T$ are already connected to a port, and all top sites to the bottom-left of $\mathbf{F}_T$ have to be connected to a port that lies to the left of $x(\mathbf{F}_T)$. Thus, we do not have to check new strip conditions until $s < x(\mathbf{F}_T)$. We handle $\mathbf{F}_R$ similarly.

We now look at the length of the top leaders, the length of the right leaders can be handled similarly. Note that by moving from t to $t-1$, the length of the top leaders does not change. If $g_T > 0$, we imagine an additional port at $x(\mathbf{F}_T)$ that can be connected to g_T top sites. When moving from s to $s-1$, we add $g_T \cdot (G_x(s) - G_x(s-1))$ to l. When we calculate a new value $\mathbf{F}_T$ by checking the strip condition, we can immediately connect all top sites inside the top front to top ports, and add the corresponding leader length to l. Thus, we only encounter site events for sites that are (a) inside $\mathbf{F}_T \setminus K$ or (b) have to be connected to a top port that lies to the left of $x(\mathbf{F}_T)$. In case (a) we do not change l, in case (b) we connect the site to the imaginary port, add the length of the corresponding leader to l and increase g_T by 1.

When we encounter a port event, if the port lies inside $\mathbf{F}_T \smallsetminus K$, we do not change l, otherwise we can connect any of the unlabeled sites to this port. We add the horizontal distance between $G_x(s)$ and the port to l and decrease g_T by 1. Note that by choosing any unlabeled site, the resulting solution may not be planar. However, because the bends of all unconnected sites will be above C, we can use Lemma 3.1 to remove the crossings without changing the total leader length.

Since the matrix now has four additional fields, the running time and storage is increased by a factor of n^4 over the algorithm from Theorem 3.1. Additionally, we need $O(n \log n)$ time to check the strip condition and to compute a length-minimal solution for the sites and ports inside $\mathbf{F}_T \smallsetminus K$ and $\mathbf{F}_R \smallsetminus K$.

Theorem 3.5. *Two-Sided Boundary Labeling with Adjacent Sides can be solved in $O(n^8 \log n)$ time using $O(n^6)$ space such that the total leader length is minimized.*

Using an appropriate data structure to precompute the fronts, it may be possible to decrease the running time slightly.

3.4 The Three- and Four-Sided Cases

In this section, we also allow labels on the bottom and the left side of R. In order to solve an instance of the three- and four-sided case, we adapt the techniques we developed for the two-sided case. We assume that the ports are fixed and the number of labels and sites is equal. In Section 3.4.1 we first analyze the structure of planar solutions obtaining a result similar to Proposition 3.1. In Sections 3.4.2 and 3.4.3, we present algorithms for the three- and four-sided cases.

3.4.1 Structure of Three- and Four-Sided Planar Solutions

Similar to our approach to two-sided boundary labeling, we pursue the idea that if there exists a planar solution, then we can also find a planar solution such that there are four xy-monotone curves connecting the four corners of R to a common point o, and such that these curves separate the leaders of the different label types from each other; see Figure 3.13. To that end, we first show that leaders of left and right labels can be separated vertically and leaders of top and bottom labels can be separated horizontally. Afterwards, we apply the result of Lemma 3.2 in order to resolve the remaining overlaps, for example, between top and right leaders. We first introduce some notions.

Definition 3.2. A planar solution for the four-sided boundary labeling problem is

   (i) *x-separated* if there exists a vertical line ℓ such that the sites that are labeled to the left side are to the left of ℓ and the sites that are labeled to the right side are to the right of ℓ, and

   (ii) *y-separated* if there exists a horizontal line ℓ such that the sites that are labeled to the top side are above ℓ and the sites that are labeled to the bottom side are below ℓ.
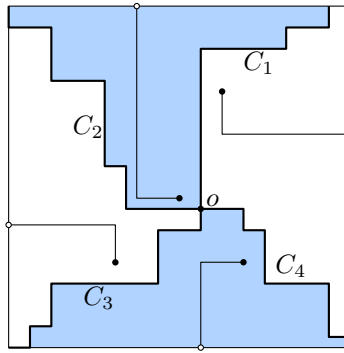
Figure 3.13: The curves C_1, C_2, C_3 and C_4 meeting at the point o partition the rectangle into four regions.

A left leader λ and a right leader λ' *overlap* if $x(\text{bend}(\lambda)) > x(\text{bend}(\lambda'))$. Analogously, a bottom leader λ and a top leader λ' *overlap* if $y(\text{bend}(\lambda)) > y(\text{bend}(\lambda'))$. Hence, a planar solution $\mathcal{L}$ is both x-separated and y-separated if and only if no left and right leaders overlap, and no bottom and top leaders overlap. We are now ready to prove that we can always find a planar solution that is both x-separated and y-separated, if a solution exists.

Lemma 3.7. *If there exists a planar solution for the four-sided boundary labeling problem, then there exists a planar solution $\mathcal{L}$ that is both x-separated and y-separated.*

Proof. Among all planar solutions let $\mathcal{L}$ be one that minimizes $|\mathcal{L}|_x + |\mathcal{L}|_y$. We prove that then $\mathcal{L}$ is x- and y-separated by showing that otherwise we could reroute some leaders and obtain a planar solution $\mathcal{L}'$ with $|\mathcal{L}'|_x + |\mathcal{L}'|_y < |\mathcal{L}|_x + |\mathcal{L}|_y$.

   Assume that $\mathcal{L}$ is not x-separated. Symmetric arguments hold for the case that $\mathcal{L}$ is not y-separated. Then there exist sites p_R and p_L with $x(p_R) < x(p_L)$, such that p_R is labeled by a right port r, and p_L is labeled by a left port ℓ; see Figure 3.14a. Without loss of generality, assume that the horizontal segment of $\lambda_R = \lambda(p_R, r)$ is above the horizontal segment of $\lambda_L = \lambda(p_L, \ell)$, otherwise we mirror the instance vertically.

   We choose p_L and p_R as a closest pair in the sense that the horizontal segments of their leaders have minimum vertical distance among all such pairs. Let A be the rectangle spanned by $\text{bend}(\lambda_L)$ and $\text{bend}(\lambda_R)$. By the minimality of p_L and p_R, that rectangle can only be intersected by top and bottom leader, but not by left or right leaders. If no such leader intersects A, we reroute p_R to the port of λ_L and p_L to the port of λ_R, which decreases $|\mathcal{L}|_x$ without increasing $|\mathcal{L}|_y$; see Figure 3.14a. It does not introduce any crossings.

   In the following we assume that some leaders intersect A. Without loss of generality we assume that there is a top leader λ_T that intersects A; otherwise we rotate the instance by 180°. We denote its site by p_T. Let S be the rectangle spanned by the ports ℓ and r; see Figure 3.14a. Depending on the leaders intersecting S, we distinguish two cases. Note that in particular λ_T intersects S.

Case 1: For any top leader λ intersecting S and for any bottom leader λ' intersecting S such that λ and λ' overlap, the site of λ lies to the left of the site of λ'; see Figure 3.14b. Let q_R denote the bottommost site that is connected by a right leader, and that lies in the rectangle spanned by $\text{bend}(\lambda_T)$ and p_R. Since p_R lies in that rectangle, the site q_R exists. We denote the

(a) empty rectangle

(b) case 1

(c) case 2.1

(d) case 2.2
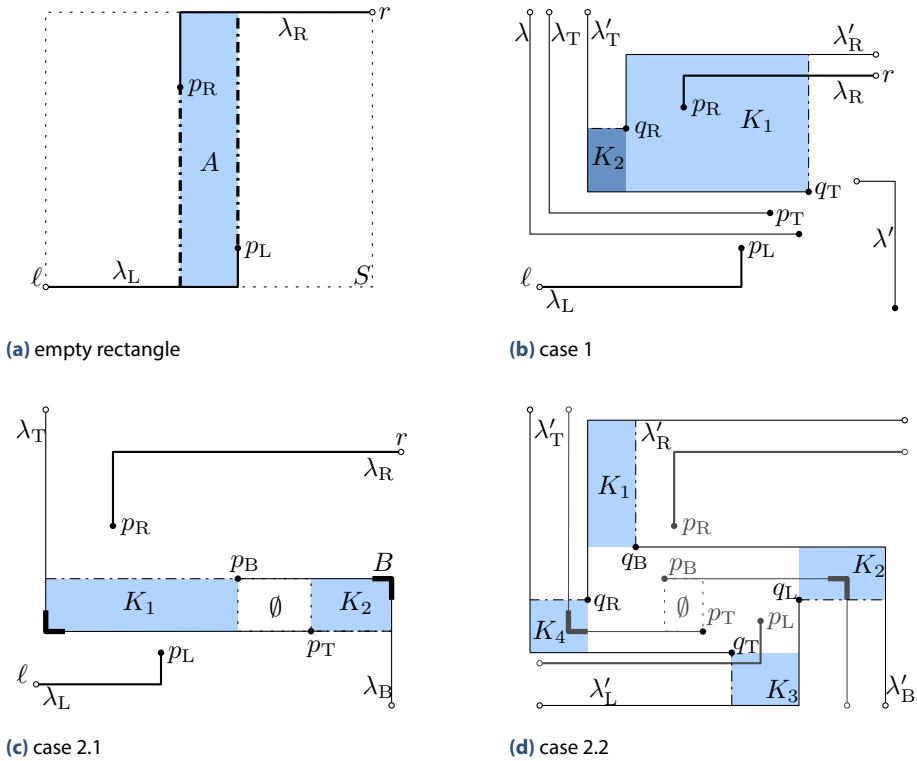
Figure 3.14: Different constellations of leaders intersecting the rectangle A. (a) The rectangle A is empty. (b)–(d) Different cases where A is intersected by a top leader. A is not explicitly illustrated, but spanned by $\mathrm{bend}(\lambda_\mathrm{R})$ and $\mathrm{bend}(\lambda_\mathrm{L})$.

leader of q_R by λ_R'. Further, let q_T be the topmost site that is connected by a top leader and that lies in the rectangle spanned by $\mathrm{bend}(\lambda_\mathrm{R}')$ and the bottom-right corner of R. Since p_T lies in that rectangle, the site q_T exists. We denote its leader by λ_T'.

We now define two rectangles that we use to reroute leaders such that $|\mathcal{L}|_x + |\mathcal{L}|_y$ is decreased and arising crossings can be resolved. The rectangle K_1 is spanned by $\mathrm{bend}(\lambda_\mathrm{R}')$ and q_T, and the rectangle K_2 is spanned by $\mathrm{bend}(\lambda_\mathrm{T}')$ and q_R.

Claim 2.

(1) K_1 is only intersected by right leaders whose bends are contained in K_1,

(2) K_2 is only intersected by top leaders whose bends are contained in K_2, and

(3) K_1 and K_2 are internally disjoint.

Assuming that the claim holds, we can reroute the sites as follows; we illustrate this rerouting by dash-dotted lines in Figure 3.14b. The site q_T is rerouted to the port of λ_R' creating crossings only on the right side of K_1. The site q_R is rerouted to the port of λ_T' creating crossings only

on the top side of K_2. Each rerouting decreases either $|\mathcal{L}|_x$ or $|\mathcal{L}_y|$ increasing the other one. Further, only crossings between leaders of the same type are created. We apply Lemma 3.1 to resolve the conflicts without increasing $|\mathcal{L}|_x$ or $|\mathcal{L}_y|$. We now show that the stated claim holds.

First, we show that K_1 is only intersected by right leaders whose bends lie in K_1. It is not intersected by any bottom leader, because such a leader would overlap λ'_T, and its site would lie to the left of q_T—a contradiction to the assumption of this case. It is not intersected by any left leader, because such a leader would intersect λ'_T. It is not intersected by any top leader, because such a leader would either intersect λ'_R or contradict the choice of λ'_T. Hence, K_1 can only be intersected by right leaders. Further, all those leaders have their bend in K_1, because the bottom-right corner is a site connected by a top leader. That leader would be intersected if a right leader intersecting K_1 had its bend outside of K_1.

Next, we show that K_2 is only intersected by top leaders whose bends lie in K_2. It is not intersected by any right leader, because such a leader would contradict the choice of λ'_R or intersect λ_T. It is not intersected by any bottom leader, because such a leader would overlap λ'_T, and its site would lie to the left of q_T—a contradiction to the assumption of this case. It is not intersected by any left leader, because such a leader would intersect λ'_T. Hence, K_2 can only be intersected by top leaders. Further, all those leaders have their bend in K_2, because the top-right corner is a site connected by a right leader.

Finally, the rectangles K_1 and K_2 are internally disjoint, because K_1 lies to the right of the vertical line through q_R, while K_2 lies to the left of that line.

Case 2: There exist a top leader λ_T intersecting S and a bottom leader λ_B intersecting S such that they overlap and the site of λ_T lies to the right of the site of λ_B; see Figure 3.14c. Among all such pairs we choose λ_T and λ_B such that their horizontal segments have minimal vertical distance. We denote the site of λ_T by p_T and the site of λ_B by p_B. Due to the choice of λ_T and λ_B, the open rectangle that is spanned by p_B and p_T is intersected by no leader. The open rectangle spanned by bend(λ_T) and bend(λ_B) is denoted by B. Depending on the sites that are contained in B, we distinguish four cases.

Case 2.1: The rectangle B contains no sites that are connected by left of right leaders; see Figure 3.14c. Let K_1 be the rectangle spanned by bend(λ_T) and p_B, and let K_2 be the rectangle spanned by bend(λ_B) and p_T. While K_1 is only intersected by left leaders, K_2 is only intersected by right leaders. Further, both rectangles are disjoint. We reroute p_B to the port of λ_T and p_T to the port of λ_B. Obviously, this decreases $|\mathcal{L}|_y$ without increasing $|\mathcal{L}|_x$. By applying Lemma 3.1, we resolve the arising conflicts.

Case 2.2: The rectangle B contains sites that are connected by left leaders as well as sites that are connected by right leaders; see Figure 3.14d. Let q_R be the bottommost site in B that is connected to the right. We denote the leader of q_R by λ'_R. Let q_B be the leftmost site with $y(q_B) \geq y(p_B)$ and $x(q_B) \leq x(p_B)$ that is connected to the bottom. Since p_B also satisfies these requirements, the site q_B exists. Similarly, we denote the leader of q_B by λ'_B. Let q_L be the topmost site in B that is connected to the left. We denote the leader of q_L by λ'_L. Finally, let q_T be the rightmost site with $y(q_T) \leq y(p_T)$ and $x(q_T) \geq x(p_T)$ that is connected to the top. Since p_T also satisfies these requirements, the site q_T exists. We denote the leader of q_T by λ'_T.

We now define four rectangles that we use to reroute leaders such that $|\mathcal{L}|_x + |\mathcal{L}|_y$ is decreased and arising crossings can be resolved. The rectangle K_1 is spanned by bend(λ'_R) and q_B, the rectangle K_2 is spanned by bend(λ'_B) and q_L, the rectangle K_3 is spanned by bend(λ'_L) and q_T,

and the rectangle K_4 is spanned by $\mathrm{bend}(\lambda'_\mathrm{T})$ and q_R. Note that the rectangles K_3 and K_4 are rotationally symmetric to K_1 and K_2, respectively.

Claim 3.

  (1) K_1 is only intersected by right leaders whose bends are contained in K_1,

  (2) K_2 is only intersected by bottom leaders whose bends are contained in K_2,

  (3) K_3 is only intersected by left leaders whose bends are contained in K_3,

  (4) K_4 is only intersected by top leaders whose bends are contained in K_4, and

  (5) K_1, K_2, K_3 and K_4 are pairwise internally disjoint.

Assuming that the claim holds, we can reroute the sites in a circular fashion as follows; we illustrate the rerouting as dash-dotted lines in Figure 3.14d. The site q_B is rerouted to the port of λ'_R creating crossings only on the right side of K_1. The site q_L is rerouted to the port of λ'_B creating crossings only on the bottom side of K_2. The site q_T is rerouted to the port of λ'_L creating crossing only on the left side of K_3. Finally, the site q_R is rerouted to the port of λ'_T creating crossings only on the top side of K_4. Each rerouting decreases either $|\mathcal{L}|_x$ or $|\mathcal{L}_y|$ without increasing the other one. Further, only crossings between leaders of the same type are created. We apply Lemma 3.1 to resolve the conflicts. In the remainder of this case we show that the stated claim holds.

First, we show that K_1 is only intersected by right leaders whose bends lie in K_1. This rectangle is not intersected by any bottom leader, because q_B is the leftmost site with $y(q_\mathrm{B}) \geq y(p_\mathrm{B})$ and $x(q_\mathrm{B}) \leq x(p_\mathrm{B})$ that is connected to the bottom. It is not intersected by any top leader, because such a leader would intersect λ'_R whose site lies below q_B. Finally, it is not intersected by any left leader, because such a leader would intersect λ'_T whose site lies to the right of q_B. Hence, only right leaders intersect K_1. In particular, all those leaders have their bend in K_1, because the bottom-right corner of K_1 is the site of a bottom leader. That leader would be intersected if a right leader intersecting K_1 had its bend outside of K_1. Since K_3 is rotationally symmetric to K_1, we can use symmetric arguments to prove that K_3 is only intersected by left leaders whose bends are contained in K_3

Next, we show that K_2 is only intersected by bottom leaders whose bends lie in K_2. This rectangle is not intersected by any left leader, because such a leader would contradict the choice of q_L. It is also not intersected by any top leader, because such a leader would intersect λ'_R or contradict the choice of λ_T and λ_B. Finally, it cannot be intersected by any right leader, because such a leader would intersect λ'_B. Hence, K_2 is only intersected by bottom leaders. Further, all those leaders have their bend in K_2, because the bottom-left corner of K_2 is a site connected to a left leader. That leader would be intersected if a bottom leader intersecting K_2 had its bend outside of K_2. Since K_4 is rotationally symmetric to K_2, we can use symmetric arguments to prove that K_4 is only intersected by top leaders whose bends are contained in K_4.

Finally, we show that the rectangles K_1, K_2, K_3 and K_4 are pairwise internally disjoint. For a site p let $v(p)$ denote the vertical line through p and let $h(p)$ denote the horizontal line through p. By construction we have that $h(q_\mathrm{B})$ lies above $h(q_\mathrm{T})$, K_1 lies above $h(q_\mathrm{B})$, and K_3

lies below $h(q_T)$. Hence, the rectangles K_1 and K_3 are internally disjoint. Analogously, we have that $v(q_L)$ lies to the right of $v(q_R)$, K_2 lies to the right of $v(q_L)$, and K_4 lies to the left of $v(q_R)$. Hence, the rectangles K_2 and K_4 are internally disjoint. Further, the sites q_L and q_R lie in between $h(q_B)$ and $h(q_T)$, because both lie in B. Consequently, K_1 and K_3 do not intersect K_2 and K_4, respectively.

Case 2.3: The rectangle B contains only sites connected by right leaders. We apply the same procedure as in the previous case. However, we do not need to consider left leaders. Hence, K_3 is removed and K_2 is the rectangle that is spanned by $\text{bend}(\lambda'_B)$ and p_T. By the choice of B, the rectangle K_2 is only intersected by right leaders whose bend is contained in K_2. Further, the remaining rectangles K_1, K_2 and K_4 are pairwise internally disjoint. The reroutings are again done in a circular fashion decreasing $|\mathcal{L}|_x + |\mathcal{L}|_y$. Finally, we apply Lemma 3.1 to resolve crossings.

Case 2.4: The rectangle B contains only sites connected by left leaders. This case can be handled analogously to the previous case by mirroring the instance vertically. $\qquad\square$

This lemma shows that, when searching for a planar solution of the labeling problem, we can restrict ourselves to solutions that are x-separated and y-separated. Let $\mathcal{L}$ denote such a solution, and let ℓ_v and ℓ_h be the lines separating the sites labeled by left and right labels, and the ones labeled by top and bottom labels, respectively. Let $o \in R$ denote the intersection of ℓ_v and ℓ_h, called *center point*. Let $r_1, \ldots, r_4$ denote the corners of R, named in counterclockwise ordering, and such that r_1 is the top-right corner. Consider the rectangles that are spanned by o and r_i for $i = 1, \ldots, 4$. Each of them contains only two types of leaders. For example, the top-right rectangle contains only top and right leaders. An x- and y-separated planar solution is *partitioned* if, for each rectangle spanned by o and one of the corners r_i of R, there exists an xy-monotone curve C_i from r_i to o that separates the two different types of leaders contained in that rectangle; see Figure 3.13. Our next step is to show that a planar solution can be transformed into a partitioned solution without increasing $|\mathcal{L}|_x$ and $|\mathcal{L}|_y$.

Proposition 3.2. *If there exists a planar solution $\mathcal{L}$ for* Four-Sided Boundary Labeling, *then there exists a partitioned solution $\mathcal{L}'$.*

Proof. By Lemma 3.7, we can assume that $\mathcal{L}$ is x- and y-separated. Let o be the center point as defined above and let ℓ_v be the vertical line through o. We show how to ensure that the area K of R right of ℓ_v admits an xy-monotone curve from the top-right corner of R to o that separates the top leaders from the right leaders inside K. The remaining cases are symmetric.

Essentially, we proceed as in the proof of Proposition 3.1 to remove the obstructions of types (P1)–(P4); see Figure 3.5. We note that in the rerouting, we only shorten vertical segments of top leaders and right segments of right leaders; hence the solution remains x- and y-separated. Moreover, in each step we decrease both $|\mathcal{L}|_x$ and $|\mathcal{L}|_y$. Hence, after finitely many steps all patterns between top and right leaders have been removed without creating new patterns with other types of leaders.

After all patterns have been removed, an xy-monotone curve connecting the top-right corner of R to o, separating the top labels from the right labels, can be found as in the proof of Lemma 3.2. $\qquad\square$
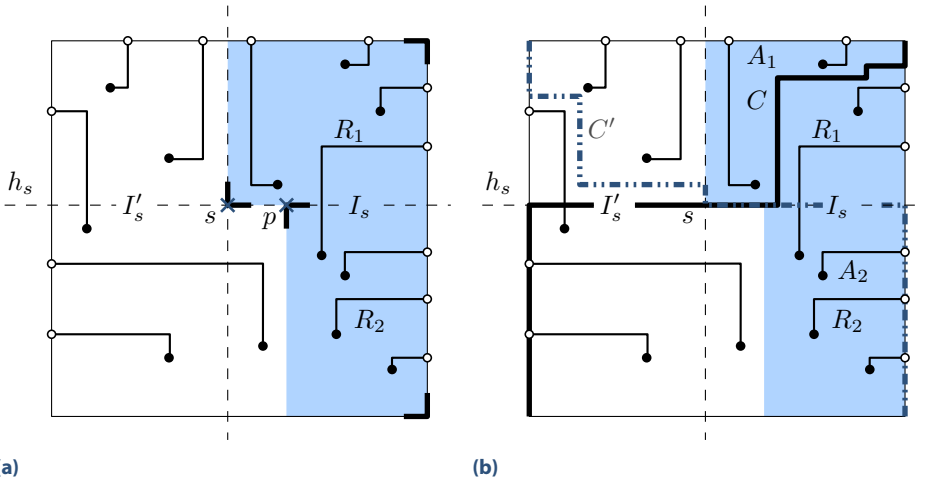
Figure 3.15: (a) The three-sided instance partitioned into two two-sided L-shaped instances I_s and I'_s. The instances are induced by the grid point s of G and are balanced. (b) Illustration of the proof for Lemma 3.8. Assuming that the grid point s of G, the balanced instances I_s and I'_s, and the curves C and C' are given, a planar solution for the whole instance can be constructed.

3.4.2 Algorithm for the Three-Sided Case

In the three-sided case, we assume that the ports of the given instance I are located on three sides of R; without loss of generality, on the left, top and right side of R. Basically, we solve a three-sided instance by splitting the instance into two two-sided L-shaped instances that can be solved independently; see Figure 3.15a.

Let G be the dual of the grid that is induced by the sites and ports of the given instance. The idea is that each grid point s of G induces two two-sided L-shaped instances with some useful properties. We will show that there is a planar solution for I if and only if there is a grid point s of G such that its induced two-sided instances both have planar solutions. Thus, considering all $O(n^2)$ grid points of G the problem reduces to solve those L-shaped instances of the two-sided case. By means of a simple adaption of the dynamic program presented in Section 3.2 we solve these instances in $O(n^2)$ time achieving $O(n^4)$ running time in total.

In the following we call horizontal and vertical lines through grid points of G *horizontal* and *vertical grid lines*, respectively. We now define the two two-sided L-shaped instances I_s and I'_s of a grid point s of G formally. To that end, let R_1 be the rectangle that is spanned by the top-right corner of R and s, and let $R_2(p)$ be the rectangle that is spanned by a point p on the horizontal grid line h through s and the bottom-right corner of R; see Figure 3.15a. The instance $I_s(p)$ contains all sites and ports in $R_1 \cup R_2(p)$ and $I'_s(p)$ contains all sites and ports in $R \setminus (R_1 \cup R_2(p))$. We say that $I_s(p)$ and $I'_s(p)$ are *balanced* if all right ports lie in $R_1 \cup R_2(p)$, all left ports lie in $R \setminus (R_1 \cup R_2(p))$ and $R_1 \cup R_2(p)$ contains the same number of sites as it contains ports. Since the number of ports and sites in I is equal, this directly implies that $R \setminus (R_1 \cup R_2(p))$ contains the same number of sites as it contains ports. In particular, the choice of balanced instances $I_s(p)$ and $I'_s(p)$ for a grid point s of G is unique with respect to the contained sites and ports; only the location of p might differ. We can therefore write I_s

and I'_s for balanced instances and R_1 and R_2 for their defining rectangles. For any solution of I_s and any solution of I'_s, we require that all leaders are completely contained in $R_1 \cup R_2$ and in $R \setminus (R_1 \cup R_2)$, respectively. The next lemma states that a three-sided instance I has a planar solution if and only if it can be partitioned into two two-sided L-shaped instances that have planar solutions. To that end let h_s denote the horizontal grid line through s. Figure 3.15 illustrates the lemma.

Lemma 3.8. *There is a planar solution $\mathcal{L}$ for a three-sided instance I if and only if there is a grid point s of G with balanced instances I_s and I'_s over rectangles (R_1, R_2), an xy-monotone curve C from the top-right corner to the bottom-left corner of R and an xy-monotone curve C' from the top-left corner to the bottom-right corner of R such that*

   (1) each point on C satisfies the strip condition with respect to the ports and sites in I_s,

   (2) C contains the top-left corner of R_2 and the intersection of h_s with the left segment of R,

   (3) each point on C' satisfies the strip condition with respect to the ports and sites in I'_s,

   (4) C' contains the top-left corner of R_2 and the intersection of h_s with the right segment of R.

Proof. First, assume that s, I_s, I'_s, (R_1, R_2), C and C' exist as required; see Figure 3.15b. The curve C partitions R into two regions; we denote the region above C by A_1 and the region below C by A_2. By Lemma 3.3, there is a planar solution $\mathcal{L}_1$ for the sites and ports in A_1 such that all leaders of $\mathcal{L}_1$ lie in A_1. Since C contains the top-left corner of R_2 and does not cross h_s until it reaches the intersection point of h_s with the left segment of R, we know that all leaders of $\mathcal{L}_1$ are contained in $R_1 \cup R_2$. Analogously, there is a planar solution $\mathcal{L}_2$ for the sites and ports in A_2 such that all leaders of $\mathcal{L}_2$ lie in A_2. Consequently, we can combine $\mathcal{L}_1$ and $\mathcal{L}_2$ into a planar solution $\mathcal{L}_s$ for the sites and ports in I_s. Using symmetric arguments, we obtain a planar solution $\mathcal{L}'_s$ for I'_s. As I_s and I'_s are defined over complementary areas, the solutions $\mathcal{L}_s$ and $\mathcal{L}'_s$ can be combined into a planar solution of I.

Assume that there is a planar solution $\mathcal{L}$ for a three-sided instance I; see Figure 3.16. First, note that we can imagine an instance of Three-Sided Boundary Labeling as a degenerated instance of Four-Sided Boundary Labeling with no bottom ports. Thus, Proposition 3.2 also holds for the three-sided case, when assuming that the four xy-monotone curves partitioning the solution meet on the bottom segment of R. Hence, without loss of generality, we assume that $\mathcal{L}$ is also partitioned by four xy-monotone curves C_1, C_2, C_3 and C_4. In particular, let C_1 denote the curve that starts at the top-right corner of R and let C_2 denote the curve that starts at the top-left corner of R; see Figure 3.16a. Then, the two curves C_3 and C_4 are completely contained in the bottom side of R and can therefore be omitted. Now, we first show how to construct the grid point s and the instances I_s and I'_s such that they are balanced. Afterwards, we explain how to obtain C and C' from C_1 and C_2, respectively. Finally, we prove that each point on C and C' satisfies the strip condition with respect to I_s and I'_s, respectively.

Let λ_T be the top leader in $\mathcal{L}$ with the longest vertical segment of all top leaders in $\mathcal{L}$. In case the site of λ_T lies to the right of $\text{bend}(\lambda_T)$, let v be the rightmost vertical grid line that lies to the left of λ_T, and otherwise if the site of λ_T lies to the left of $\text{bend}(\lambda_T)$, let v be the leftmost vertical grid line that lies to the right of λ_T. Furthermore, let h be the topmost horizontal grid

(a)   **(b)**
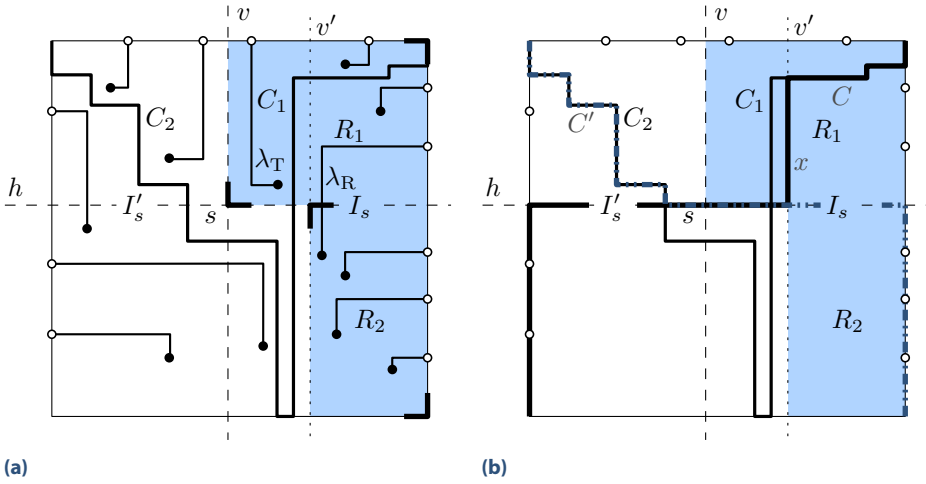
Figure 3.16: Illustration of the proof for Lemma 3.8. It is assumed that the partitioned planar solution $\mathcal{L}$ for the three-sided instance is given. (a) By Proposition 3.2, we can assume that $\mathcal{L}$ is partitioned by the curves C_1 and C_2. The extremal top leader λ_T induces the site s and the extremal right leader λ_R induces the line v'. (b) Based on C_1, C_2, h and v', the curves C and C' can be constructed such that they do not cross any leader of $\mathcal{L}$.

line that lies below bend(λ_T); see Figure 3.16a. Due to the choice of h and v all top leaders lie above h and none of them intersects h or v. Furthermore, no right or left leader of $\mathcal{L}$ intersects v above h. The desired grid point s is then the intersection point of h and v.

Now, let λ_R be the right leader in $\mathcal{L}$ with longest horizontal segment among all right leaders in $\mathcal{L}$ and let v' be the rightmost vertical grid line that lies to the left of bend(λ_R). Note that v' cannot be intersected by a left or a right leader, because both leader types are x-separated. We define R_1 to be the rectangle that is spanned by the top-right corner of R and s. Also, we define R_2 to be the rectangle spanned by the bottom-right corner of R and the intersection point of v' and h. The instance I_s is defined by $R_1 \cup R_2$ and the instance I'_s by $R \setminus (R_1 \cup R_2)$. We show that I_s and I'_s are balanced. To that end, we prove that a leader of $\mathcal{L}$ is either completely contained in $R_1 \cup R_2$ or in $R \setminus (R_1 \cup R_2)$, that $R_1 \cup R_2$ contains only right and top leaders, and that $R \setminus (R_1 \cup R_2)$ contains only left and top leaders.

Due to the choice of v', all right leaders lie to the right of v'. Moreover, all right leaders whose site or port lies above h, must lie to the right of v, because by definition of v no right leader intersects v above h (otherwise it would intersect λ_T), and because otherwise C_1 could not be an xy-monotone curve separating right and top leaders. Thus, all right leaders lie in $R_1 \cup R_2$. For left leaders we can argue similarly. Since left and right leaders of $\mathcal{L}$ are x-separated, all left leaders lie to the left of v'. All left leaders whose site or port lies above h, must lie to the left of v, because by definition of v no left leader intersects v above h, and because otherwise C_2 could not be an xy-monotone curve separating left from top leaders. Thus, all left leaders lie in $R \setminus (R_1 \cup R_2)$. Finally, consider the top leaders in $\mathcal{L}$. By definition of h and v, none of the top leaders intersects h or v. In particular all top leaders lie above h and cannot intersect R_2. Consequently, each top leader is either contained in R_1 or in $R \setminus (R_1 \cup R_2)$. This concludes the argument that I_s and I'_s are balanced.
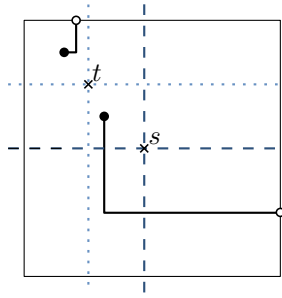
Figure 3.17: There are no balanced instances I_s and I'_s for the grid point s. However, by Lemma 3.8 there must be another grid point t with balanced instances I_t and I'_t if the instance has a planar solution.

We are left with the construction of the curves C and C'; see Figure 3.16b. The curve C is derived from C_1 as follows. Starting at the top-right corner of R, the curve C coincides with C_1 until C_1 intersects h or v' above h. If C intersects v' above h, it follows v' downwards until it hits h. Then, in both cases, it follows h until h intersects the left segment of R. Finally, C follows the left segment of R to the bottom-left corner of R. The curve C' is constructed symmetrically.

By construction, C contains the top-left corner of R_2 and the intersection point of h with the left segment of R. Symmetrically, C' contains the top-left corner of R_2 and the intersection point of h with the right segment of R. We finally show that each point on C satisfies the strip condition with respect to the sites and ports in I_s. Using symmetric arguments we can prove the analogous statement for C' and I'_s.

By the previous reasoning, we know that each leader of $\mathcal{L}$ either lies completely inside or completely outside of $R_1 \cup R_2$. Each leader that lies in $R_1 \cup R_2$ is either a top or a right leader and does not intersect C. Otherwise, if such a leader intersected C, it would also intersect C_1 or the segment x of v' that is contained in C. In particular, x cannot be intersected by any leader because it lies to the left of all right leaders and below C_1. Thus, the leaders in $R_1 \cup R_2$ form a planar solution for I_s without intersecting C. Hence, the claim directly follows from Lemma 3.3. □

Our approach now works as follows. For each grid point s of G we compute the instances I_s and I'_s such that they are balanced. Then, by Lemma 3.8, we can apply our algorithm presented in Section 3.2 in order to solve I_s and I'_s independently. To that end, we slightly adapt the dynamic program such that it only considers curves satisfying the properties required by Lemma 3.8. If both instances can be solved, we combine these solutions into one solution and return that solution as the final result. Otherwise, we continue with the next grid point of G. If all grid points of G have been explored without finding a planar solution, the algorithm decides that there is no planar solution.

Note that it may happen that, for a grid point s, there are no balanced instances I_s and I'_s; for an example see Figure 3.17. However, in that case, if I has a solution, we also know by Lemma 3.8 that there is another grid point t such that for t we find balanced instances. Hence, we can refrain from considering s.

Creating the two instances I_s and I'_s for a grid point s takes linear time, if we assume that the sites are sorted by their x-coordinates. By Theorem 3.2 we then need $O(n^2)$ time and $O(n)$ space to solve I_s and I'_s. Consequently, we need $O(n^2)$ time and $O(n)$ space to process a single grid point s. Since we consider $O(n^2)$ grid points, the following theorem follows.

Theorem 3.6. *Three-Sided Boundary Labeling can be solved in $O(n^4)$ time using $O(n)$ space.*

Note that, except for the length minimization, our approach for the three-sided case also carries over to the extensions from Section 3.3, because we only solve subinstances of Two-Sided Boundary Labeling with Adjacent Sides. In particular with corresponding impact on the running time we can soften the restriction that the number of labels and sites is equal.

3.4.3 Algorithm for the Four-Sided Case

In this section, we consider the case that the ports lie on all four sides of R. The main idea is to seek a partitioned solution, which exists by Proposition 3.2. For a given partitioned solution $\mathcal{L}$, we call a leader *extremal* if all other leaders of the same type in $\mathcal{L}$ have shorter orthogonal segments; recall that the *orthogonal* segment of a *po*-leader is the segment connecting the bend to the port. The algorithm consists of two steps. First, we explore all choices of (non-overlapping) extremal leaders λ_L and λ_R for the left and the right side of R, respectively, plus a horizontal line h that separates the top leaders and the bottom leaders. This information splits the instance into two independent three-sided instances; see Figure 3.18a. There are, however, two crucial differences from a usual three-sided instance. First, one side of the instance is not a straight-line segment but an x-monotone orthogonal curve C that is defined by λ_L, λ_R and h. Second, the extremal positions of λ_L and λ_R imply a separation of the points that are labeled from the left and the right side. Let I_1^3 be the three-sided instance above C and let I_2^3 be the three-sided instance below C. The algorithm solves I_1^3 and I_2^3 independently from each other. If for at least one of the two instances there is no solution, the algorithm continues with the next choice of λ_L, λ_R and h. Otherwise, it combines the planar solutions of I_1^3 and I_2^3 into one planar solution and returns this solution. In case that all choices of λ_L, λ_R and h have been explored without finding a solution, the algorithm returns that there is no planar solution.

We next describe how to solve the three-sided instance I_1^3. A symmetric approach can be applied to I_2^3. The algorithm explores all choices of the extremal leader λ_T for the top side of R. This extremal leader partitions the instance into two two-sided subinstances I_1^2 and I_2^2 as follows. Let $A_{T,R}$ be the rectangle that is spanned by $\text{bend}(\lambda_T)$ and the top-right corner of R; see Figure 3.18b. Analogously, let $A_{T,L}$ be the rectangle that is spanned by $\text{bend}(\lambda_T)$ and the top-left corner of R. Analogously, for λ_R we define the area $A_{R,T}$ to be the rectangle that is spanned by $\text{bend}(\lambda_R)$ and the top-right corner of R, and $A_{R,B}$ to be the rectangle spanned by $\text{bend}(\lambda_R)$ and the bottom-right corner of R. For the leader λ_L we define $A_{L,B}$ to be the rectangle spanned by $\text{bend}(\lambda_L)$ and the bottom-left corner of R, and $A_{L,T}$ to be the rectangle spanned by $\text{bend}(\lambda_L)$ and the top-left corner of R. We assume that the port p of λ_T is only contained in that area $A \in \{A_{T,R}, A_{T,L}\}$ that also contains the site of λ_T. We make analogous assumptions for λ_L and λ_R.

(a)   **(b)**   **(c)**
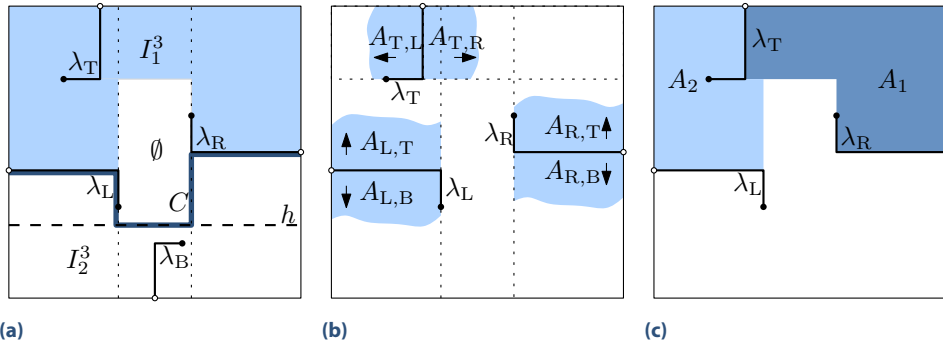
Figure 3.18: (a) The right leader λ_L, the left leader λ_R and the horizontal line h split the instance into two three-sided instances I_1^3 and I_2^3. (b) Sketch of the areas $A_{T,L}$, $A_{T,R}$, $A_{R,T}$, $A_{R,B}$, $A_{L,T}$ and $A_{L,B}$. (c) The leaders λ_L, λ_R and λ_T split the three-sided instance into two two-sided instances.

The instance I_1^2 consists of all ports and sites in $A_1 = (A_{R,T} \cup A_{T,R}) \setminus (A_{T,L} \cup A_{R,B})$, and I_2^2 consists of all ports and sites in $A_2 = (A_{L,T} \cup A_{T,L}) \setminus (A_{T,R} \cup A_{L,B})$; see Figure 3.18c. We solve I_1^2 and I_2^2 independently from each other using the dynamic program introduced in Section 3.2 for each instance. However, we enforce that it only considers xy-monotone curves that exclude top leaders crossing the horizontal line through $\text{bend}(\lambda_T)$, left leaders crossing the vertical line through $\text{bend}(\lambda_L)$ and right leaders crossing the vertical line through $\text{bend}(\lambda_R)$. If for at least one of the two instances there is no solution, the algorithm continues to explore the next choice of λ_T. Otherwise, it combines the solutions of I_1^2 and I_2^2 into one solution and returns the result as the solution of I_1^3. In case that all choices of λ_T have been explored without finding a solution, the algorithm returns that there is no solution for the given three-sided instance. The following lemma shows that the algorithm is correct.

Lemma 3.9. *Given an instance I of* FOUR-SIDED BOUNDARY LABELING, *the following two statements are true.*

  (1) *If there is no planar solution for I, the algorithm states this.*

  (2) *If there is a planar solution for I, the algorithm returns such a solution.*

Proof. In case the algorithm returns a solution, it has been constructed from planar solutions of disjoint instances of TWO-SIDED BOUNDARY LABELING WITH ADJACENT SIDES. As the union of these two-sided instances contains all sites and ports of I, the algorithm returns a planar solution of I, which shows the first statement.

Conversely, assume that I has a planar solution $\mathcal{L}$. By Proposition 3.2, we may assume that $\mathcal{L}$ is partitioned. In particular, let λ_T, λ_L, λ_B and λ_R be the extremal leaders in $\mathcal{L}$ of the top, left, bottom and right side of R, respectively, and let h be a horizontal line that separates the top leaders from the bottom leaders.

Obviously, λ_L, λ_R and h split the instance into two three-sided instances I_1^3 and I_2^3. As the algorithm systematically explores all choices of extremal right leaders, extremal left leaders and horizontal lines partitioning the set of sites, it must find λ_L, λ_R and a horizontal line h' that separates the same sets of sites as h. Thus, I_1^3 and I_2^3 are considered by the algorithm.

Let I_1^3 be the instance above the curve defined by λ_L, λ_R and h', and let I_2^3 be the instance below that curve. We now show that the algorithm finds a planar solution for I_1^3. Symmetric arguments hold for I_2^3. As the algorithm explores all choices of extremal top leaders in I_1^3, it also considers λ_T to be the extremal top leader. This leader partitions the area of I_1^3 into the two disjoint areas $A_1 = (A_{R,T} \cup A_{T,R}) \smallsetminus (A_{T,L} \cup A_{R,B})$ and $A_2 = (A_{L,T} \cup A_{T,L}) \smallsetminus (A_{T,R} \cup A_{L,B})$; see Figure 3.18a.

It directly follows from the extremal choice of λ_R, λ_T and λ_L that there is no leader in $\mathcal{L}$ that intersects both A_1 to A_2. In particular, no left leader intersects A_1 and no right leader intersects A_2. Thus, A_1 and A_2 split $\mathcal{L}$ into independent planar solutions $\mathcal{L}_1$ and $\mathcal{L}_2$ of two two-sided instances I_1^2 and I_2^2 induced by A_1 and A_2, respectively. Note that the algorithm considers the same two-sided instances independently from each other. As I_1^2 has a solution, namely $\mathcal{L}_1$, we know that the dynamic program finds a solution $\mathcal{L}_1^2$ for I_1^2. In particular, all leaders of $\mathcal{L}_1^2$ lie in A_1.

Applying symmetric arguments for I_2^2, the algorithm yields a planar solution $\mathcal{L}_2^2$ that stays in A_2. Consequently, combining $\mathcal{L}_1^2$ and $\mathcal{L}_2^2$ into one solution yields a planar solution $\mathcal{L}_1^3$ for I_1^3. Analogously, we obtain a planar solution $\mathcal{L}_2^3$ for I_2^3. Obviously, due to the separation by λ_L, λ_R and h', the union of $\mathcal{L}_1^3$ and $\mathcal{L}_2^3$ is also planar, which is the overall solution returned by the algorithm. This proves the second statement of the lemma. $\qquad\square$

Let us analyze the running time of the algorithm. Obviously, there are $O(n^5)$ possible combinations of left and right extremal leaders and a horizontal line separating the top and bottom-labeled sites. For each combination, we independently solve two three-sided instances. For such a three-sided instance, we consider $O(n^2)$ choices for the extremal leader λ_T and independently solve two independent two-sided instances with Theorem 3.2 in $O(n^2)$ time. This implies that solving one three-sided instances takes $O(n^4)$ time. Thus, the overall running time is $O(n^9)$. The following theorem summarizes this result.

Theorem 3.7. Four-Sided Boundary Labeling *can be solved in* $O(n^9)$ *time using* $O(n)$ *space.*

Note that, except for the length minimization, our approach for the four-sided case also carries over to the extensions from Section 3.3, because we only solve subinstances of Two-Sided Boundary Labeling with Adjacent Sides. In particular with corresponding impact on the running time we can soften the restriction that the number of labels and sites is equal.

3.5 Concluding Remarks

In this chapter, we have studied the problem of testing whether an instance of Two-Sided Boundary Labeling with Adjacent Sides admits a planar solution. We have given the first efficient algorithm for this problem, running in $O(n^2)$ time.

Our algorithm can also be used to solve a variety of different extensions of the problem. We have shown how to generalize to sliding ports instead of fixed ports without increasing the running time and how to maximize the number of labeled sites such that the solution is planar in $O(n^3 \log n)$ time. We further have given an extension to the algorithm that minimizes the total leader length in $O(n^8 \log n)$ time.

With some additional work, our approach can also be used to solve THREE-SIDED and FOUR-SIDED BOUNDARY LABELING in polynomial time. We have introduced an algorithm solving the three-sided case in $O(n^4)$ time and the four-sided case in $O(n^9)$ time. Also, except for the length minimization, all extensions carry over. It remains open whether a minimum length solution of THREE-SIDED and FOUR-SIDED BOUNDARY LABELING can be computed in polynomial time.

4 Approximation Algorithms for Box Contact Representations

In the last few years, word clouds have become a standard tool for abstracting, visualizing, and comparing text documents. For example, word clouds were used in 2008 to contrast the speeches of the US presidential candidates Obama and McCain. More recently, the German media used them to visualize the newly signed coalition agreement and to compare it to a similar agreement from 2009 [Wei13]. A word cloud of a given document consists of the most important (or most frequent) words in that document. Each word is printed in a given font and scaled by a factor roughly proportional to its importance (the same is done with the names of towns and cities on geographic maps, for example). The printed words are arranged without overlap and tightly packed into some shape (usually a rectangle). Tag clouds look similar; they consist of keyword metadata (tags) that have been attributed to resources in some collection such as web pages or photos.

Wordle [VWF09] is a popular tool for drawing word or tag clouds. The Wordle website allows users to upload a list of words and, for each word, its relative importance. The user can further select font, color scheme, and decide whether all words must be placed horizontally or whether words can also be placed vertically. The tool then computes a placement of the words, each scaled according to its importance, such that no two words overlap. Generally, the drawings are very compact and aesthetically appealing.

In the automated analysis of text one is usually not just interested in the most important words and their frequencies, but also in the connections between these words. For example, if a pair of words often appears together in a sentence, then this is often seen as evidence that this pair of words is linked semantically [Li02]. In this case, it makes sense to place the two words close to each other in the word cloud that visualizes the given text.

Problem Statement.   We capture the semantic connections between words by an input graph $G = (V, E)$ of desired contacts. We are also given, for each vertex $v \in V$, the dimensions (but not the position) of a *box* B_v, that is, an axis-aligned rectangle. We denote the height and width of B_v by $h(B_v)$ and $w(B_v)$, respectively, or, more briefly, by $h(v)$ and $w(v)$. For complexity reasons, we require the lengths of the boxes to be rational numbers. For each edge $e = (u, v)$ of G, we are given a positive number $p(e) = p(u, v)$ which corresponds to the *profit* of e. For ease of notation, we set $p(u, v) = 0$ for any non-edge $(u, v) \in V^2 \smallsetminus E$ of G.

Given a box B and a point q in the plane, let $B(q)$ be a placement of B with lower left corner q. A *representation* of G is a map $\lambda: V \to \mathbb{R}^2$ such that for any two vertices $u \neq v$, it holds that $B_u(\lambda(u))$ and $B_v(\lambda(v))$ have disjoint interior. Boxes may *touch*, that is, their boundaries may intersect. If the intersection is non-degenerate, that is, a line segment of positive length, we say that the boxes are *in contact*. We say that a representation λ *realizes* an edge (u, v) of G if boxes $B_u(\lambda(u))$ and $B_v(\lambda(v))$ are in contact.

This yields the problem *Contact Representation of Word Networks* (CROWN): Given an edge-weighted graph G whose vertices correspond to boxes, find a representation of G with the

vertex boxes such that every edge of G is realized. In this chapter, we study the optimization version of Crown, Max-Crown, where the aim is to maximize the total profit (that is, the sum of the weights) of the realized edges. We also consider the unweighted version of the problem, where all desired contacts yield a profit of 1.

Previous Work.   Barth et al. [BFK+14] recently introduced Max-Crown and showed that the problem is strongly NP-hard even for trees and weakly NP-hard even for stars. They presented an exact algorithm for cycles and approximation algorithms for stars, trees, planar graphs, and graphs of constant maximum degree; see the first column of Table 4.1. Some of their solutions use an approximation algorithm with ratio $\alpha = e/(e-1) \approx 1.58$ [FGMS11] for the Generalized Assignment Problem (Gap), defined as follows: Given a set of bins with capacity constraints and a set of items that possibly have different sizes and values for each bin, pack a maximum-valued subset of items into the bins. The problem is APX-hard [CK05].

Max-Crown is related to finding *rectangle representations* of graphs, where vertices are represented by axis-aligned rectangles with non-intersecting interiors and edges correspond to rectangles with a common boundary of non-zero length. Every graph that can be represented this way is planar and every triangle in such a graph is a facial triangle. These two conditions are also sufficient to guarantee a rectangle representation [BGPV08]. Rectangle representations play an important role in VLSI layout, cartography, and architecture (floor planning). In a recent survey, Felsner [Fel13] reviews many rectangulation variants. Several interesting problems arise when the rectangles in the representation are restricted. Eppstein et al. [EMSV12] consider rectangle representations which can realize any given area-requirement on the rectangles, so-called *area-preserving rectangular cartograms*, which were introduced by Raisz [Rai34] already in the 1930s. Unlike cartograms, in our setting there is no inherent geography, and hence, words can be positioned anywhere. Moreover, each word has fixed dimensions enforced by its importance in the input text, rather than just fixed area. Nöllenburg et al. [NPR13] recently considered a variant where the edge weights prescribe the length of the desired contacts.

Finally, the problem of computing semantics-aware word clouds is related to classic graph layout problems, where the goal is to draw graphs so that vertex labels are readable and Euclidean distances between pairs of vertices are proportional to the underlying graph distance between them. Typically, however, vertices are treated as points and label overlap removal is a post-processing step [DMS05, GH10]. Most tag cloud and word cloud tools such as Wordle [VWF09] do not show the semantic relationships between words, but force-directed graph layout heuristics are sometimes used to add such functionality [BKP14, CWL+10, PTT+12, WPW+11]. For an example output of such a tool, see Figure 4.1.

Model.   We consider two different models. In the model with *proper contacts*, we consider two boxes in contact only if their intersection is a line segment of positive length. Hence, the contact graph of the boxes is planar. This is the standard model used in work on rectangle contact representations. In the model with *point contacts*, we consider two boxes in contact if their intersection is a line segment or a point. In this model, a realized graph is not necessarily planar as four boxes can meet in a point and both diagonals correspond to edges of the input graph. It is, however, easy to see that the graphs that can be realized are 1-planar. If not stated

Figure 4.1: Semantics-preserving word cloud for the 35 most "important" words in this section. Following the text processing pipeline of Barth et al. [BKP14], these are the words ranked highest by LexRank [ER04], after removal of stop words such as "the". The edge profits are proportional to the relative frequency with which the words occur in the same sentences. The layout algorithm of Barth et al. [BKP14] first extracts a heavy star forest from the weighted input graph as in Theorem 4.6 and then applies a force-directed post-processing.

otherwise, our approximation algorithms work for both models. We allow words only to be placed horizontally.

Our Contribution. Known results and our contributions to Max-Crown are shown in Table 4.1. Note that the results of Barth et al. [BFK+14] in column 1 are simply based on existing decompositions of the respective graph classes into star forests or cycles.

Our results rely on a variety of algorithmic tools. First, we devise sophisticated decompositions of the input graphs into heterogeneous classes of subgraphs, which also requires a more general combination method than that of Barth et al. Second, we use randomization to obtain a simple constant-factor approximation for general weighted graphs. Previously, such a result was not even known for unweighted bipartite graphs. Third, to obtain an improved algorithm for the unweighted case, we prove a lower bound on the size of a matching in a planar graph of high average degree. Fourth, we use a planar separator result of Frederickson [Fre87] to obtain a polynomial-time approximation scheme (PTAS) for degree-bounded planar graphs.

Our other main result is the use of the combination lemma, which, among others, yielded the first approximation algorithms for bipartite and for general graphs; see Section 4.2. For general graphs, we present a simple randomized solution (based on the solution for bipartite graphs), which we then derandomize. For trees, planar graphs of constant maximum degree, and general graphs, we have improved results in the unweighted case; see Section 4.3. Finally, we show APX-completeness for bipartite graphs of maximum degree 9 (see Section 4.4) and list some open problems (see Section 4.5).

Runtimes. Most of our algorithms involve approximating a number of Gap instances as a subroutine, using either the PTAS [BKV11] if the number of bins is constant or the approximation algorithm of Fleischer et al. [FGMS11] for general instances. Because of this, the runtime of our algorithms consists mostly of approximating Gap instances. Both algorithms to approximate Gap instances solve linear programs, so we refrain from explicitly stating the runtime of these algorithms.

Graph class	Weighted			Unweighted	
	Ratio [BFK⁺14]	Ratio [new]	Theorem	Ratio	Theorem
cycle, path	1				
star	α	$1 + \varepsilon$	4.1		
tree	2α	$2 + \varepsilon$	4.1	2	4.7
	NP-hard				
max-degree Δ	$\lfloor (\Delta + 1)/2 \rfloor$				
planar max-deg. Δ				$1 + \varepsilon$	4.8
outerplanar		$3 + \varepsilon$	4.1		
planar	5α	$5 + \varepsilon$	4.1		
bipartite		APX-complete	4.10		
with proper contacts		≈ 8.4	4.2		
with point contacts		≈ 9.5	4.2		
general					
with proper contacts		≈ 16.9	4.5	≈ 13.4	4.9
with point contacts		≈ 19	4.5	≈ 16.5	4.9

Table 4.1: Previously known and new results for the unweighted and weighted versions of Max-Crown (for $\alpha \approx 1.58$ and any $\varepsilon > 0$). The exact approximation factors are denoted in the corresponding theorems.

For practical purposes, one can use a purely combinatorial approach for approximating GAP [CKR06], which utilizes an algorithm for the KNAPSACK problem as a subroutine. The algorithm translates into a 3-approximation for GAP running in $O(NM)$ time (or a $(2 + \varepsilon)$-approximation running in $O(MN \log 1/\varepsilon + M/\varepsilon^4)$ time), where N is the number of items and M is the number of bins. In our setting, the simple 3-approximation implies a randomized 32-approximation (or a deterministic 40-approximation) algorithm with running time $O(|V|^2)$ for MAX-CROWN on general weighted graphs.

4.1 Some Basic Results

In this section, we present two technical lemmas that will help us to prove our main results in the following two sections where we treat the weighted and unweighted cases of MAX-CROWN. The second lemma immediately improves the results of Barth et al. [BFK⁺14] for stars, trees, and planar graphs.

4.1.1 A Combination Lemma

Several of our algorithms cover the input graph with subgraphs that belong to graph classes for which the MAX-CROWN problem is known to admit good approximations. The following lemma allows us to combine the solutions for the subgraphs. We say that a graph $G = (V, E)$ is *covered* by graphs $G_1 = (V, E_1), \ldots, G_k = (V, E_k)$ if $E = E_1 \cup \cdots \cup E_k$.

Lemma 4.1. *Let graph $G = (V, E)$ be covered by graphs $G_1, G_2, \ldots, G_k$. If, for $i = 1, 2, \ldots, k$, weighted MAX-CROWN on graph G_i admits an α_i-approximation, then weighted MAX-CROWN on G admits a $\left(\sum_{i=1}^k \alpha_i\right)$-approximation.*

Proof. Our algorithm works as follows. For $i = 1, \ldots, k$, we apply the α_i-approximation algorithm to G_i and report the result with the largest profit as the result for G. We show that this algorithm has the claimed performance guarantee. For the graphs $G, G_1, \ldots, G_k$, let $\mathrm{OPT}, \mathrm{OPT}_1, \ldots, \mathrm{OPT}_k$ be the optimum profits and let $\mathrm{ALG}, \mathrm{ALG}_1, \ldots, \mathrm{ALG}_k$ be the profits of the approximate solutions. By definition, $\mathrm{ALG}_i \geq \mathrm{OPT}_i / \alpha_i$ for $i = 1, \ldots, k$. Moreover, $\mathrm{OPT} \leq \sum_{i=1}^k \mathrm{OPT}_i$ because the edges of G are covered by the edges of $G_1, \ldots, G_k$. Assume, without loss of generality, that $\mathrm{OPT}_1 / \alpha_1 = \max_i (\mathrm{OPT}_i / \alpha_i)$. Then,

$$\mathrm{ALG} \;=\; \mathrm{ALG}_1 \;\geq\; \frac{\mathrm{OPT}_1}{\alpha_1} \;\geq\; \frac{\sum_{i=1}^k \mathrm{OPT}_i}{\sum_{i=1}^k \alpha_i} \;\geq\; \frac{\mathrm{OPT}}{\sum_{i=1}^k \alpha_i} \;. \qquad \square$$

4.1.2 Improvement on Existing Approximation Algorithms

The approximation algorithms for stars, trees, and planar graphs that was provided by Barth et al. [BFK+14] use an α-approximation algorithm for GAP instances. We prove that these instances require only a constant number of bins and thus can be approximated using the PTAS of Briest et al. [BKV11].

Lemma 4.2 ([BKV11]). *For any $\varepsilon > 0$, there is a $(1 + \varepsilon)$-approximation algorithm for GAP with a constant number of bins. The algorithm takes $n^{O(1/\varepsilon)}$ time.* $\qquad \square$

Hakimi et al. [HMS96] have studied the star arboricity of certain graph classes, that is, the number of star forests into which a graph can be partitioned. The algorithm of Barth et al. [BFK+14] uses some of these results to get approximation algorithms for trees and planar graphs.

Lemma 4.3 ([HMS96]). *Any tree can be partitioned into two star forests, any outerplanar graph can be partitioned into three star forests, and any planar graph can be partitioned into five star forests. All these partitions can be found efficiently.*

Using Lemmas 4.1, 4.2, and 4.3, we improve the approximation algorithms provided by Barth et al. [BFK+14].

Theorem 4.1. *Weighted MAX-CROWN admits a $(1 + \varepsilon)$-approximation algorithm on stars, a $(2 + \varepsilon)$-approximation algorithm on trees, a $(3 + \varepsilon)$-approximation algorithm on outerplanar graphs, and a $(5 + \varepsilon)$-approximation algorithm on planar graphs.*

Proof. By combining Lemma 4.3 and Lemma 4.1, the claim for stars implies the other three claims. Note that a PTAS for stars immediately gives a PTAS for star forests since the algorithm can be applied to each star in the forest independently.

We now show that we can use Lemma 4.2 to get a PTAS for stars. First, we give the PTAS for the model with point contacts. Note that Barth et al. [BFK+14] already gave a similar reduction from MAX-CROWN on stars with point contacts to GAP. However, instead of trying
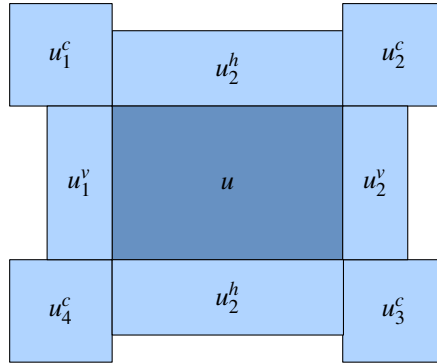
Figure 4.2: Notation for the PTAS for stars

all combinations for the adjacencies in the corners of the center box, we add an additional bin for each corner. We proceed to show that a constant number of bins suffices.

Let u be the center vertex of the star. We create eight bins: four *corner bins* u_1^c, u_2^c, u_3^c, and u_4^c modeling adjacencies on the four corners of the box u, two *horizontal bins* u_1^h and u_2^h modeling adjacencies on the top and bottom side of u, and two *vertical bins* u_1^v and u_2^v modeling adjacencies on the left and right side of u; see Figure 4.2. The capacity of the corner bins is 1, the capacity of the horizontal bins is the width $w(u)$ of u, and the capacity of the vertical bins is the height $h(u)$ of u. Next, we introduce an item $i(v)$ for any leaf vertex v of the star. The size of $i(v)$ is 1 in any corner bin, $w(v)$ in any horizontal bin, and $h(v)$ in any vertical bin. The profit of $i(v)$ in any bin is the profit $p(u, v)$ of the edge (u, v).

Note that any feasible solution to the MAX-CROWN instance can be normalized so that any box that touches a corner of u has a point contact with u. Hence, the above is an approximation-preserving reduction from weighted MAX-CROWN on stars (with point contacts) to GAP. By Lemma 4.2, we obtain a PTAS.

We now give a reduction from MAX-CROWN on stars with proper contacts to MAX-CROWN on stars with point contacts which has not been proven before. We first assume that all boxes have integral edge lengths, which can be accomplished by scaling. Consider a feasible solution with proper contacts. We modify the solution as follows. Each box that touches a corner of u is moved so that it has a point contact with this corner. Afterwards, we move some of the remaining boxes until all corners of u have point contacts or until we run out of boxes. This yields a solution with point contacts in which there are two opposite sides of u—say the two horizontal sides—which either do not touch any box or from which we removed one box during the modification. Now observe that, if we shrink the two horizontal sides by an amount of $1/2$, then all contacts can be preserved since there was a slack of at least 1 at both horizontal sides. Conversely, observe that any feasible solution with point contacts to the modified instance with shrunk horizontal sides can be transformed into a solution with proper contacts since we always have a slack of at least $1/2$ on both horizontal sides. This shows that there is a correspondence between feasible solutions with proper contacts and feasible solutions with point contacts to a modified instance where we either shrink the horizontal or the vertical sides by $1/2$. The PTAS for MAX-CROWN on stars consists in applying a PTAS to

two instances of MAX-CROWN with point contacts where we shrink the horizontal or vertical sides, respectively, and in outputting the better of the two solutions. □

4.2 The Weighted Case

In this section, we provide new constant-factor approximation algorithms for more involved classes of (weighted) graphs than in the previous section. Recall that $\alpha = e/(e-1) \approx 1.58$. First, we present an approximation algorithm for bipartite graphs. Then, for general graphs, we provide a simple randomized approximation algorithm and show how to derandomize it to get a deterministic approximation algorithm that makes $O(n)$ many calls to the α-approximation for GAP. Finally, we provide a deterministic approximation algorithm for general graphs that has a slightly worse approximation factor but only makes a single call to the α-approximation for GAP.

Theorem 4.2. *Weighted MAX-CROWN on bipartite graphs admits*

  (i) *a $16\alpha/3(\approx 8.4)$-approximation with proper contacts and*

  (ii) *a $6\alpha(\approx 9.5)$-approximation with point contacts.*

Proof. Let $G = (V, E)$ be a bipartite input graph with $V = V_1 \,\dot\cup\, V_2$ and $E \subseteq V_1 \times V_2$. Using G, we build an instance of GAP as follows. For each vertex $u \in V_1$, we create eight bins $u_1^c, u_2^c, u_3^c, u_4^c, u_1^h, u_2^h, u_1^v, u_2^v$ and set the capacities exactly as we did for the star center in Theorem 4.1. Next, we add an item $i(v)$ for every vertex $v \in V_2$. The size of $i(v)$ is, again, 1 in any corner bin, $w(v)$ in any horizontal bin, and $h(v)$ in any vertical bin. For $u \in V_1$, the profit of $i(v)$ is $p(u, v)$ in any bin of u.

It is easy to see that solutions to the GAP instance are equivalent to word cloud solutions (with point contacts) in which the realized edges correspond to a forest of stars with all star centers being vertices of V_1. Hence, we can find an approximate solution of profit $\mathrm{ALG}_1 \geq \mathrm{OPT}_1 / \alpha$ where OPT_1 is the profit of an optimum solution (with point contacts) consisting of a star forest with centers in V_1. Similarly, we can find a solution of profit $\mathrm{ALG}_2 \geq \mathrm{OPT}_2 / \alpha$ with star centers in V_2, where OPT_2 is the maximum profit that a star forest with centers in V_2 can realize.

We first show how to get a solution with point contacts. Among the two solutions described above, we pick the one with larger profit $\mathrm{ALG} = \max \{\mathrm{ALG}_1, \mathrm{ALG}_2\}$. Let $G^\star = (V, E^\star)$ be the contact graph realized by a fixed optimum solution, and let $\mathrm{OPT} = p(E^\star)$ be its total profit. We now show that $\mathrm{ALG} \geq \mathrm{OPT}/(6\alpha)$. Recall that a realized graph in this model is 1-planar. Hence, $G^\star$ is a bipartite 1-planar graph. Following a very recent proof of Ackerman [Ack14], we can cover $G^\star$ by a planar bipartite graph G_1^* and a tree G_2^*. As G_1^* is a planar bipartite graph, each (connected) subgraph $G' = (V', E')$ of G_1^* is also planar bipartite and thus has at most $|E'| \leq 2|V'| - 4$ edges. Hence, we can decompose $G_1^\star$ into two forests H_1 and H_2 using a result of Nash-Williams [NW64]; see Figure 4.3.

We can further decompose H_1 into two star forests S_1 and S_1' in such a way that the star centers of S_1 are in V_1 and the star centers of S_1' are in V_2. Similarly, we decompose H_2 into a forest S_2 of stars with centers in V_1 and a forest S_2' of stars with centers in V_2, and G_2^* into a forest S_3 of stars with centers in V_1 and a forest S_3' of stars with centers in V_2. As

(a) The graph $G^\star$ realized by an optimum solution is planar and bipartite.

(b) $G^\star$ can be decomposed into two forests H_1 and H_2 and further into four star forests S_1, S_2 (black) with centers in V_1 (disks) and S_1', S_2' (dashed) with centers in V_2 (boxes).
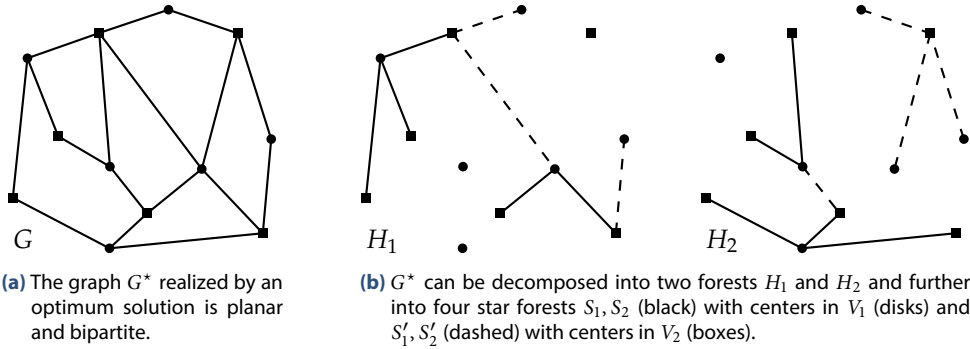
Figure 4.3: Partitioning the optimum solution in the proof of Theorem 4.2.

we decomposed the optimum solution into six star forests, one of them—say S_1—has profit $p(S_1) \geq \mathrm{OPT}/6$. On the other hand, $\mathrm{OPT}_1 \geq p(S_1)$. Summing up, we get

$$\mathrm{ALG} \geq \mathrm{ALG}_1 \geq \mathrm{OPT}_1/(6\alpha) \geq p(S_1)/(6\alpha) \geq \mathrm{OPT}/(6\alpha).$$

We now show how to get a solution with proper contacts. Recall the GAP instance that we constructed at the start of the proof. If the three bins on the top side of a vertex u (two corner bins and one horizontal bin) are not completely full, we can slightly move the boxes in the corners so that point contacts are avoided. Otherwise, we remove the lightest item from one of these bins. We treat the three bottommost bins analogously. Note that in both cases we only remove an item if all three bins are completely full. The resulting solution can be realized with proper contacts. We do the same for the three left and three right bins and choose the heavier of the two solutions.

It is easy to see that we lose at most $1/4$ of the profit for the star center u: Assume that the heaviest solution results from removing weight w_1 from one of the upper and weight w_2 from one of the lower bins. As we remove the lightest items only, the remaining weight from the upper and lower bins is at least $2(w_1 + w_2)$. On the other hand, the weight in the two vertical bins is at least $w_1 + w_2$; otherwise, dropping everything from these vertical bins would be cheaper. Hence, we keep at least weight $3(w_1 + w_2)$.

If we do so for all star centers, we get a solution with profit

$$\mathrm{ALG}_1' \geq 3/4 \cdot \mathrm{ALG}_1 \geq 3\,\mathrm{OPT}_1/(4\alpha) \geq 3\,\mathrm{OPT}_1'/(4\alpha)$$

where OPT_1' is the profit of an optimum solution (with proper contacts) consisting of a star forest with centers in V_1. Similarly, we can find a solution of profit $\mathrm{ALG}_2' \geq 3\,\mathrm{OPT}_2'/(4\alpha)$ with star centers in V_2, where OPT_2' is the maximum profit that a star forest with centers in V_2 can realize. Among the two solutions, we pick the one with larger profit $\mathrm{ALG}' = \max\{\mathrm{ALG}_1', \mathrm{ALG}_2'\}$.

Let $G^\star = (V, E^\star)$ be the contact graph realized by a fixed optimum solution, and let $\mathrm{OPT} = p(E^\star)$ be its total profit. We now show that $\mathrm{ALG} \geq 3\,\mathrm{OPT}/(16\alpha)$. As $G^\star$ is a planar bipartite graph, using the same argument as above, we can decompose it into four star

forests S_1, S_1', S_2, S_2' in such a way that the star centers of S_1 and S_2 are in V_1 and the star centers of S_1' and S_2' are in V_2. As we decomposed the optimum solution into four star forests, one of them—say S_1—has profit $p(S_1) \geq \mathrm{OPT}/4$. On the other hand, $\mathrm{OPT}_1 \geq p(S_1)$. Summing up, we get

$$\mathrm{ALG} \; \geq \; \mathrm{ALG}_1 \; \geq \; 3\,\mathrm{OPT}_1/(4\alpha) \; \geq \; 3p(S_1)/(4\alpha) \; \geq \; 3\,\mathrm{OPT}/(16\alpha). \qquad \square$$

Theorem 4.3. *Weighted MAX-CROWN on general graphs admits*

 (i) *a randomized $32\alpha/3 (\approx 16.9)$-approximation with proper contacts and*

 (ii) *a randomized $12\alpha (\approx 19)$-approximation with point contacts.*

Proof. Let $G = (V, E)$ be the input graph and let OPT be the weight of a fixed optimum solution. Our algorithm works as follows. We first randomly partition the set of vertices into two sets V_1 and $V_2 = V \smallsetminus V_1$. To this end we assign every $v \in V$ either to V_1 or to V_2 with probability $1/2$ each, so that the random decisions for the nodes are mutually independent. Now, we consider the bipartite graph $G' = (V_1 \,\dot\cup\, V_2, E')$ with $E' = \{(v_1, v_2) \in E \mid v_1 \in V_1 \text{ and } v_2 \in V_2\}$ that is induced by V_1 and V_2. By applying Theorem 4.2 on G', we can find a feasible solution for G with weight $\mathrm{ALG} \geq 3\,\mathrm{OPT}'/(16\alpha)$ in the model with proper contacts or weight $\mathrm{ALG} \geq \mathrm{OPT}'/(6\alpha)$ in the model with point contacts, where OPT' is the weight of an optimum solution for G'.

Any edge of the optimum solution is contained in G' with probability $1/2$. Let $\overline{\mathrm{OPT}}$ be the total weight of the edges of the optimum solution that are present in G'. Then, $E[\overline{\mathrm{OPT}}] = \mathrm{OPT}/2$. Hence,

$$E[\mathrm{ALG}] \; \geq \; 3E[\mathrm{OPT}']/(16\alpha) \; \geq \; 3E[\overline{\mathrm{OPT}}]/(16\alpha) \; = \; 3\,\mathrm{OPT}/(32\alpha)$$

in the model with proper contacts and

$$E[\mathrm{ALG}] \; \geq \; E[\mathrm{OPT}']/(6\alpha) \; \geq \; E[\overline{\mathrm{OPT}}]/(6\alpha) \; = \; \mathrm{OPT}/(12\alpha)$$

in the model with point contacts. $\qquad \square$

In order to derandomize the above algorithms, we make use of the following classical result.

Theorem 4.4 (Folklore [AS92]). *Let $n = 2^k - 1$ and $d = 2t + 1 \leq n$. Then there exists a probability space Ω of size $2(n+1)^t$ and d-wise independent random variables $X_1, \dots, X_n$ over Ω each of which takes values 0 and 1 with probability $1/2$. The space and the variables can be constructed in $O(tn^{t+1} \log n)$ time.*

Proof. The construction is described, for example, by Alon and Spencer [AS92] (Theorem 16.2.1). They don't state the running time of the construction explicitly, so we will do this now.

First, they set up a $(1 + kt) \times n$ matrix H over the field $\mathbb{Z}_2$. Computing an entry of H takes $O(tk^2)$ time, hence $O(t^2 n \log^3 n)$ time in total.

Second, they construct the probability space Ω and the variables $X_1, \dots, X_n$. To this end, they compute the linear combinations of all $2^{kt+1} = 2(n+1)^t$ many subsets of rows of H, each of which takes $O(nkt)$ time. Using $k = O(\log n)$, this sums up to $O(tn^{t+1} \log n)$, which dominates the first step since $t \leq n$. $\qquad \square$

Theorem 4.5. *Weighted MAX-CROWN on general graphs admits*

  (i) a (deterministic) $32\alpha/3 (\approx 16.9)$-approximation with proper contacts and

  (ii) a (deterministic) $12\alpha (\approx 19)$-approximation with point contacts.

Both approximation algorithms make $O(n)$ many calls to the α-approximation algorithm for GAP on instances with $O(n)$ items and bins.

Proof. Let $G = (V, E)$ be the input graph and let $V = \{v_1, \ldots, v_n\}$. We set $t = 1$ and apply Theorem 4.4 to construct, in $O(n^2 \log n)$ time, a probability space of size $O(n)$ and random variables $X_1, \ldots, X_n$ that are 3-wise (and thus pairwise) independent.

   We now run the algorithms described in the proof of Theorem 4.3 with the only difference that we do not assign the vertices to V_1 or V_2 mutually independently but by sampling from the space Ω. In doing so, we add v_i to V_1 if $X_i = 0$ and to V_2 if $X_i = 1$. Let (v_i, v_j) be an edge of the optimum solution. Since the variables X_i and X_j are independent, the probability that the edge (v_i, v_j) ends up in the bipartite graph G' is precisely $1/2$ as in the original algorithms. Using an analogous analysis, we can prove that both algorithms retain their expected approximation performance if we use the space Ω. Hence, there must be at least one elementary event in Ω that achieves the expected approximation ratio.

   The crucial point is that we can exhaustively check the space Ω running the algorithms for each of the elementary events to *deterministically* find a solution whose approximation ratio is bounded by the expected approximation ratio. □

Note that the above derandomization makes $O(n)$ many calls to the α-approximation algorithm for GAP on $O(n)$ items and bins, two for each of the $O(n)$ many bipartite graphs that are considered. The α-approximation algorithm for GAP is based on LP rounding [CKR06], which might be time consuming.

   Below, we give a deterministic approximation algorithm solving only one GAP instance with $O(n)$ items and bins. The algorithm obtains a weaker but still constant approximation ratio.

Theorem 4.6. *Weighted MAX-CROWN on general graphs admits*

  (i) a $40\alpha/3 (\approx 21.1)$-approximation with proper contacts and

  (ii) a $14\alpha (\approx 22.1)$-approximation with point contacts.

Both approximation algorithms make a single call to the α-approximation algorithm for GAP on an instance with $O(n)$ items and bins.

Proof. Let $G = (V, E)$ be the input graph. As in the proof of Theorem 4.2, our algorithm constructs an instance of GAP based on G. The difference is that, *for every vertex $v \in V$, we create both eight bins and an item $i(v)$*. Capacities and sizes remain as before. The profit of placing item $i(v)$ in a bin of vertex u, with $u \neq v$, is $p(u, v)$. We stress here that every vertex $v \in V$ corresponds to both a bin and an item in the instance of GAP.

   We first show how to get a solution with proper contacts. Let OPT be the value of an optimum solution of MAX-CROWN in G, and let $\mathrm{OPT_{GAP}}$ be the value of an optimum solution for the constructed instance of GAP. Since any optimum solution of MAX-CROWN is a planar
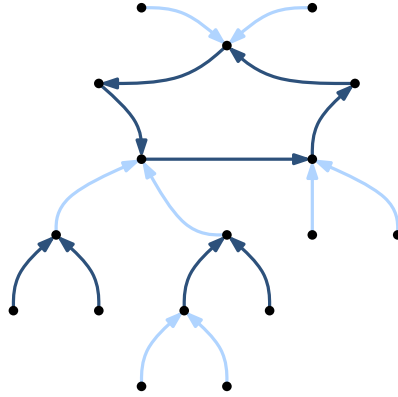
Figure 4.4: Partitioning a 1-tree into a star forest (light) and the union of a cycle and a star forest (dark)

graph, it can be decomposed into five star forests following a result of Hakimi et al. [HMS96]. Hence, there exists a star forest carrying at least OPT /5 of the total profit. Such a star forest corresponds to a solution of GAP for the constructed instance; therefore, $\text{OPT}_{\text{GAP}} \geq \text{OPT}/5$.

Now we compute an α-approximation for the GAP instance, which results in a solution of total profit $\text{ALG}_{\text{GAP}} \geq \text{OPT}_{\text{GAP}}/\alpha$. Next, we show how our solution induces a feasible solution of MAX-CROWN where every vertex $v \in V$ is either a bin or an item.

Consider the directed graph $G_{\text{GAP}} = (V, E_{\text{GAP}})$ with $(u, v) \in E_{\text{GAP}}$ if and only if the item corresponding to $u \in V$ is placed into a bin corresponding to $v \in V$. A connected component in G_{GAP} with n' vertices has at most n' edges since every item can be placed into at most one bin. If $n' = 2$, we arbitrarily make one of the vertices a bin and the other one an item. If $n' > 2$, the connected component is a 1-tree, that is, a tree and an edge. In this case, we partition the edges into two subgraphs; a star forest and the disjoint union of a star forest and a cycle; see Figure 4.4. Note that both subgraphs can be represented by touching boxes if we allow point contacts. This is due to the fact that the stars correspond to a solution of GAP. Hence, choosing a subgraph with larger weight and post-processing the solution as in the proof of Theorem 4.2 results in a feasible solution of MAX-CROWN with proper contacts. Initially, we discarded at most half of the weight and the post-processing keeps at least 3/4 of the weight, so in total

$$\text{ALG} \geq 3\,\text{ALG}_{\text{GAP}}/8 \geq 3\,\text{OPT}_{\text{GAP}}/(8\alpha) \geq 3\,\text{OPT}/(40\alpha).$$

We now show how to get a solution with point contacts. Since any optimal solution of MAX-CROWN is a 1-planar graph, it can be decomposed into seven star forests by using the decomposition into a planar graph and a tree proven by Ackerman [Ack14]; thus, $\text{OPT}_{\text{GAP}} \geq \text{OPT}/7$. We proceed with the same algorithm as in the model with proper contacts. However, we do not need the post-processing that discards up to 1/4 of the weight. Therefore,

$$\text{ALG} \geq \text{ALG}_{\text{GAP}}/2 \geq \text{OPT}_{\text{GAP}}/(2\alpha) \geq \text{OPT}/(14\alpha). \qquad \square$$

4.3 The Unweighted Case

In this section, we consider the unweighted MAX-CROWN problem, that is, all desired contacts have profit 1. Thus, we want to maximize the number of edges of the input graph realized by the contact representation. We present approximation algorithms for different graph classes. First, we give a 2-approximation for trees. Then, we present a PTAS for planar graphs of bounded degree. Finally, for general graphs, we provide a $(5 + 16\alpha/3)$-approximation with proper contacts and a $(7 + 6\alpha)$-approximation with point contacts.

Theorem 4.7. *Unweighted MAX-CROWN on trees admits a 2-approximation.*

Proof. Let T be the input tree. We first decompose T into edge-disjoint stars as follows. If T has at most two vertices, then the decomposition is straight-forward. So, we assume without loss of generality that T has at least three vertices and is rooted at a non-leaf vertex. Let u be a vertex of T such that all its children, say $v_1, \ldots, v_k$, are leaf vertices. If u is the root of T, then the decomposition contains only one star centered at u. Otherwise, denote by π the parent of u in T, create a star S_u centered at u with edges $(u, \pi), (u, v_1), \ldots, (u, v_k)$ and call the edge (u, π) of S_u the *anchor edge* of S_u. The removal of $u, v_1, \ldots, v_k$ from T results in a new tree. Therefore, we can recursively apply the same procedure. The result is a decomposition of T into edge-disjoint stars covering all edges of T.

We next remove, for each star, its anchor edge from T. We apply the PTAS of Theorem 4.1 to the resulting star forest and claim that the result is a 2-approximation for T. To prove the claim, consider a star S'_u of the new star forest, centered at u with edges $(u, v_1), \ldots, (u, v_k)$ and let ALG be the total number of contacts realized by the $(1 + \varepsilon)$-approximation algorithm on S'_u. We consider the following two cases.

(a) $1 \leq k \leq 4$: Since it is always possible to realize four contacts of a star, ALG $\geq k$. Note that an optimal solution may realize at most $k + 1$ contacts (due to the absence of the anchor edge from S'_u). Hence, our algorithm has approximation ratio $(k + 1)/k \leq 2$.

(b) $k \geq 5$: Since it is always possible to realize four contacts of a star, we have ALG ≥ 4. On the other hand, an optimal solution realizes at most $(1 + \varepsilon)$ ALG $+1$ contacts. Thus, the approximation ratio is

$$((1 + \varepsilon) \text{ ALG} + 1) / \text{ALG} \leq (1 + \varepsilon) + 1/4 < 2.$$

The theorem follows from the fact that all edges of T are incident to the star centers. $\square$

Next, we develop a PTAS for bounded-degree planar graphs. Our construction needs two lemmas, one of which was shown by Barth et al. [BFK+14].

Lemma 4.4 ([BFK+14]). *If the input graph has maximum degree Δ, then* OPT $\geq 2|E|/(\Delta + 1)$.

The other lemma provides an exponential-time exact algorithm for MAX-CROWN.

Lemma 4.5. *There is an exact algorithm for unweighted MAX-CROWN with running time $2^{O(n \log n)}$.*

Proof. Consider a placement which assigns a position $[\ell_B, r_B] \times [b_B, t_B]$ to every box B, with $\ell_B + w(B) = r_B$ and $b_B + h(B) = t_B$. For the x-axis, this gives a (possibly nonstrict) linear order on the values ℓ_B and r_B, where some might be equal. An order on the y-axis is implied similarly. Together, these two orders fully determine the combinatorial structure of overlaps and contacts: for contact, two boxes must have a side of equal value and a side with overlap.

The algorithm enumerates all possible combinations of two such orders using the representation sketched above. On a single axis, this is a permutation of $2n$ variables and, between every two variables adjacent in this permutation, whether they are equal or the second variable has strictly larger value. This representation demonstrates that the number of distinct orders in one dimension is bounded by

$$O((2n)! \cdot 2^{2n}) = 2^{O(n \log n)}.$$

The number of combinations of two such orders also satisfies this bound.

For any given pair of orders, it can be determined if they imply overlaps and what the objective value is: count the number of profitable contacts. If there are no overlaps, the existence of an actual placement realizing the orders is tested using linear programming. As these tests run in polynomial time, an optimal placement can be found in $2^{O(n \log n)}$ time. □

We will now utilize these two lemmas to obtain a PTAS for planar graphs with bounded degree.

Theorem 4.8. *Unweighted* MAX-CROWN *on planar graphs with n vertices and maximum degree Δ admits a* PTAS. *More specifically, for any $\varepsilon > 0$ there is an $(1 + \varepsilon)$-approximation algorithm with linear running time $n2^{(\Delta/\varepsilon)^{O(1)}}$.*

Proof. Let $G = (V, E)$ be a graph and let r be a parameter to be determined later. Frederickson [Fre87] showed that we can find a vertex set $X \subseteq V$ (called r-*division*) of size $O(n/\sqrt{r})$ in $O(n \log n)$ time such that the following holds. The vertex set $V \setminus X$ can be partitioned into n/r vertex sets $V_1, \ldots, V_{n/r}$ such that (i) $|V_i| \leq r$ for $i = 1, \ldots, n/r$ and (ii) there is no edge running between any two distinct vertex sets V_i and V_j. In what follows, we assume without loss of generality that G is connected, as we can apply the PTAS to every connected component separately.

We apply the result of Frederickson to the input graph and compute an r-division X. By removing the vertex set X from the graph, we remove $O(n\Delta/\sqrt{r})$ edges from G. Now, we apply the exact algorithm of Lemma 4.5 to each of the induced subgraphs $G[V_i]$ separately. The solution is the union of the optimum solutions to $G[V_i]$.

Since no edge runs between the distinct sets V_i and V_j, the subgraphs $G[V_i]$ cover $G - X$. Let $E^\star$ be the set of edges realized by an optimum solution to G, let OPT $= |E^\star|$, and let OPT$' = |E^\star \cap E(G - X)|$. Since G is connected, it has $|E| \geq n - 1$ edges. Thus, by Lemma 4.4, we have that

$$\text{OPT} \geq 2(n-1)/(\Delta+1) = \Omega(n/\Delta).$$

When we removed X from G, we removed $O(n\Delta/\sqrt{r})$ edges. Hence,

$$\text{OPT} = \text{OPT}' + O(n\Delta/\sqrt{r}) \text{ and } \text{OPT}' = \Omega(n(1/\Delta - \Delta/\sqrt{r})).$$

Since we solved each sub-instance $G[V_i]$ optimally and since these sub-instances cover $G - X$, the solution created by our algorithm realizes at least OPT' many edges. Using this fact and the above bounds on OPT and OPT', the total performance of our algorithm can be bounded by

$$\frac{\text{OPT}}{\text{OPT}'} = \frac{\text{OPT}' + O(n\Delta/\sqrt{r})}{\text{OPT}'} = 1 + O\left(\frac{n\Delta/\sqrt{r}}{n(1/\Delta - \Delta/\sqrt{r})}\right) = 1 + O\left(\frac{\Delta^2}{\sqrt{r} - \Delta^2}\right).$$

We want this last term to be smaller than $1 + \varepsilon$ for some prescribed error parameter $0 < \varepsilon \le 1$. It is not hard to verify that this can be achieved by letting $r = \Theta(\Delta^4/\varepsilon^2)$. Since each of the subgraphs $G[V_i]$ has at most r vertices, the total running time for determining the solution is $n2^{(\Delta/\varepsilon)^{O(1)}}$. $\qquad\square$

Before tackling the case of general graphs, we need a lower bound on the size of maximum matchings in planar graphs in terms of the numbers of vertices and edges.

Lemma 4.6. *Any planar graph with n vertices and m edges contains a matching of size at least $(m - 2n)/3$.*

Proof. Let G be a planar graph. If G has $n \le 9$ vertices, due to planarity, we have

$$(m - 2n)/3 \le (n - 6)/3 \le 1.$$

Hence, any nonempty matching is large enough.

Nishizeki and Baybars [NB79] showed that any connected planar graph with at least $n \ge 10$ vertices and minimum degree 3 has a matching of size at least

$$\lceil (n + 2)/3 \rceil \ge n/3 > (n - 6)/3 \ge (m - 2n)/3$$

since $m \le 3n - 6$.

It remains to tackle the case where G has at least $n \ge 10$ vertices and is not connected or has a vertex of degree less than 3. Our proof is by induction on n. If G is not connected, the claim follows by applying the inductive hypothesis to every connected component. Now assume that G has a vertex u of degree less than 3. Consider the graph $G' = G - u$ with $n' = n - 1$ vertices and $m' \ge m - 2$ edges. By the inductive hypothesis G' (and hence, G, too) has a matching of size at least

$$(m' - 2n')/3 \ge ((m - 2) - 2(n - 1))/3 = (m - 2n)/3. \qquad\square$$

We are now ready to present an approximation algorithm for general graphs.

Theorem 4.9. *Unweighted MAX-CROWN on general graphs admits*

  (i) *a $(5 + 16\alpha/3)(\approx 13.4)$-approximation with proper contacts and*

  (ii) *a $(7 + 6\alpha)(\approx 16.4)$-approximation with point contacts.*

Proof. We first show how to get a solution with proper contacts. The algorithm first computes a maximal matching M in G. Let V' be the set of vertices matched by M, let G' be the subgraph

(a) G is covered by $\bar{G}$ (bipartite, light) and G' (black); maximal matching M (bold).

(b) maximum matching M'' (bold) in $G'' = G' - M$

(c) optimum solution to G': graph G^* (black) and part of M (bold)
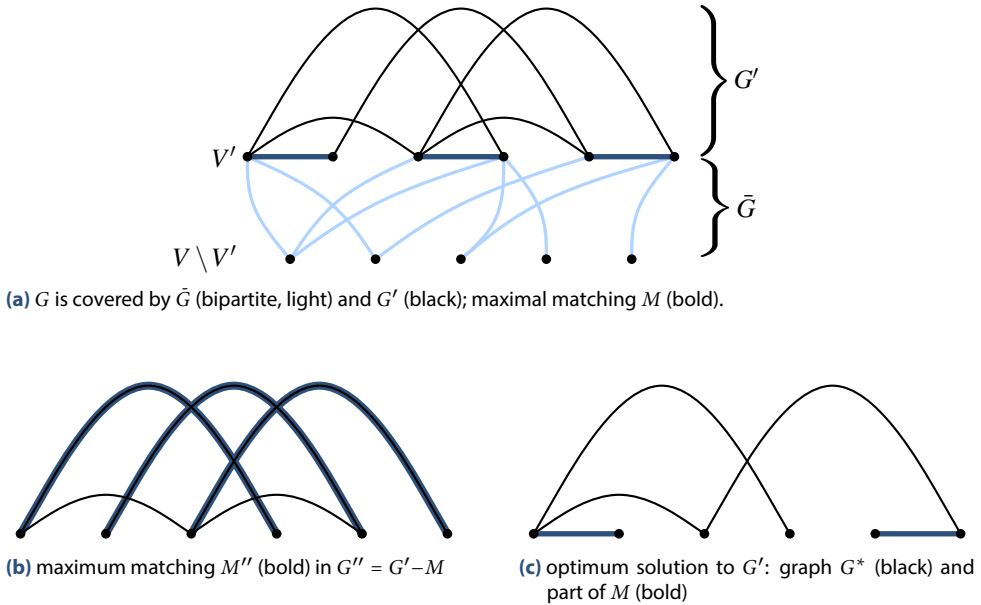
Figure 4.5: Partitioning the input graph and the optimum solution in the proof of Theorem 4.9.

induced by V', and let E' be the edge set of G'. Note that $\bar{G} = G - E'$ is a bipartite graph with partition $(V', V \smallsetminus V')$. This is because the matching M is maximal, which implies that every edge in $E \smallsetminus E'$ is incident to a vertex in V' and to a vertex not in V'; see Figure 4.5a. Hence, we can compute a $16\alpha/3$- approximation to $\bar{G}$ using the algorithm presented in Theorem 4.2.

Consider the graph $G'' = (V', E' \smallsetminus M)$ and compute a maximum matching M'' in G''; see Figure 4.5b. The edge set $M \cup M''$ is a set of vertex-disjoint paths and cycles and can therefore be completely realized [BFK+14]. The algorithm realizes this set. Below, we argue that this realization is in fact a 5-approximation for G', which completes the proof (due to Lemma 4.1 and since G is covered by G' and $\bar{G}$).

Let $n' = |V'|$ be the number of vertices of G'. Let E^* be the set of edges realized by an optimum solution to G', and let $\mathrm{OPT} = |E^*|$. Consider the subgraph $G^* = (V', E^* \smallsetminus M)$ of G''; see Figure 4.5c. Note that G^* is planar and contains at least $\mathrm{OPT} - n'/2$ many edges. Applying Lemma 4.6 to G^*, we conclude that the maximum matching M'' of G'' has size at least $(\mathrm{OPT} - 5n'/2)/3$. Hence, by splitting OPT appropriately, we obtain

$$\mathrm{OPT} = (\mathrm{OPT} - 5n'/2) + 5n'/2 \leq 3|M''| + 5|M| \leq 5|M'' \cup M|.$$

We now show how to get a solution with point contacts. We use the same algorithm as described for the model with proper contacts, so we only have to adjust the analysis. We can compute a 6α- approximation to the bipartite graph $\bar{G}$ using the algorithm presented in Theorem 4.2.

It is easy to prove that any 1-planar graph with m edges and n vertices contains a matching of size at least $(m - 3n)/3$: we planarize the graph (by removing at most n edges) and then

apply Lemma 4.6. Thus, the maximum matching M'' of G'' has size at least $(\text{OPT} - 7n'/2)/3)$. Hence, by splitting OPT appropriately, we obtain

$$\text{OPT} = (\text{OPT} - 7n'/2) + 7n'/2 \leq 3|M''| + 7|M| \leq 7|M'' \cup M|. \qquad \square$$

4.4 APX-Completeness

In this section, we prove APX-completeness of weighted MAX-CROWN by giving a reduction from 3-dimensional matching. This reduction works both in the model without and in the model with point contacts.

Theorem 4.10. *Weighted MAX-CROWN is APX-complete even if the input graph is bipartite of maximum degree 9, each edge has profit 1, 2 or 3, and each vertex corresponds to a square of one out of three different sizes.*

Proof. We give a reduction from 3-dimensional matching (3DM). Recall that an instance of this problem is given by three disjoint sets X, Y, Z with cardinalities $|X| = |Y| = |Z| = k$ and a set $E \subseteq X \times Y \times Z$ of hyperedges. The objective is to find a set $M \subseteq E$, called *matching*, such that no element of $V = X \cup Y \cup Z$ is contained in more than one hyperedge in M and such that $|M|$ is maximized.

The problem is known to be APX-hard [FGMS11]. More specifically, for the special case of 3DM where every $v \in V$ is contained in at most three hyperedges (hence $|E| \leq 3k$) it is NP-hard to decide whether the maximum matching has cardinality k or only $k(1 - \varepsilon_0)$ for some constant $0 < \varepsilon_0 < 1$. We reduce from this special case of 3DM to MAX-CROWN.

To this end, we construct the following MAX-CROWN instance from a given 3DM instance. We create, for each $v \in V$, a square of side length 1. For each hyperedge $e \in E$, we create nine squares $e^\star, e_1, \ldots, e_8$ where $e^\star$ has side length 3.5 and $e_1, \ldots, e_8$ have side length 3. In the desired contact graph, we create an edge $(e^\star, e_1)$ of profit 2 and, for $i = 2, \ldots, 8$, an edge $(e^\star, e_i)$ of profit 3. We also create an edge $(e^\star, v)$ of profit 1 if v is incident to e in the 3DM instance.

Consider an optimum solution to the above MAX-CROWN instance. It is not hard to verify that, for any hyperedge $e = (x, y, z)$, the solution will realize the edges $(e^\star, e_i)$ for $i = 2, \ldots, 8$. Moreover, we can assume without loss of generality that the solution either realizes all three adjacencies $(e^\star, x)$, $(e^\star, y)$, and $(e^\star, z)$ of total profit 3 or the adjacency $(e^\star, e_1)$ of profit 2; see Figure 4.6. We call such a solution *well-formed*.

Assume that there is a solution M to the 3DM instance of cardinality k. Then this can be transformed into a well-formed solution to MAX-CROWN of profit $(7 \cdot 3 + 2)|E| + |M| = 23|E| + k$.

Conversely, suppose that the maximum matching has cardinality at most $(1 - \varepsilon_0)k$. Consider an optimum solution to the respective MAX-CROWN instance. We may assume that the solution is well-formed. Let M be the set of hyperedges $e = (x, y, z)$ for which all three adjacencies $(e^\star, x), (e^\star, y), (e^\star, z)$ are realized. Then, the profit of this solution is $(7 \cdot 3 + 2)|E| + |M| = 23|E| + |M|$.

Note that M is in fact a matching because the solution to MAX-CROWN was well-formed. Thus, the optimum profit is bounded by $23|E| + (1 - \varepsilon_0)k$.

(a) profit $7 \cdot 3 + 2 = 23$ **(b)** profit $7 \cdot 3 + 3 \cdot 1 = 24$
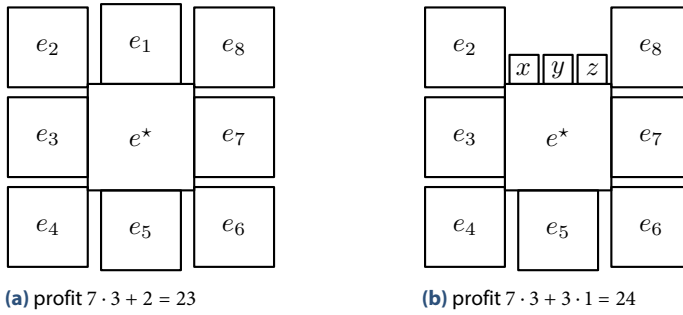
Figure 4.6: The two possible configurations of a hyperedge $e = (x, y, z)$ in the proof of Theorem 4.10

Hence, it is NP-hard to distinguish between instances with $\text{OPT} \geq 23|E| + k$ and instances with $\text{OPT} \leq 23|E| + (1 - \varepsilon_0)k$. Using $|E| \leq 3k$, this implies that there cannot be any approximation algorithm of ratio less than

$$\frac{23|E| + k}{23|E| + (1 - \varepsilon_0)k} \;=\; 1 + \frac{\varepsilon_0 k}{23|E| + (1 - \varepsilon_0)k} \;\geq\; 1 + \frac{\varepsilon_0 k}{(70 - \varepsilon_0)k} \;=\; 1 + \frac{\varepsilon_0}{70 - \varepsilon_0} \,,$$

which is a constant strictly larger than 1. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\square$

4.5 Concluding Remarks

In this chapter, we have presented approximation algorithms for the Max-Crown problem, which can be used for constructing semantics-preserving word clouds. Apart from improving approximation factors for various graph classes, many open problems remain. Most of our algorithms are based on covering the input graph by subgraphs and packing solutions for the individual subgraphs. Both subproblems—covering graphs with special types of subgraphs and packing individual solutions together—are interesting problems in their own right which may lead to algorithms with better guarantees. Practical variants of the problem are also of interest, for example, restricting the heights of the boxes to predefined values (determined by font sizes), or defining more than immediate neighbors to be in contact, thus considering non-planar "contact" graphs. Another interesting variant is when the bounding box of the representation has a certain fixed size or aspect ratio.

Part II

Visual Guidance

5 Smooth Orthogonal Layouts of Planar Graphs

In the visualization of technical networks such as the structure of VLSI chips [Lei80], floor plans [SFK11], or UML diagrams [See97], there is a strong tendency to draw edges as rectilinear paths. The problem of laying out networks in such a way is called *orthogonal graph drawing* and has been studied extensively. For drawings of (planar) graphs to be readable, special care is needed to keep the number of bends small. In a seminal work, Tamassia [Tam87] showed that one can efficiently minimize the total number of bends in orthogonal layouts of *embedded maxdeg-4 planar graphs*, that is, planar graphs of maximum degree 4 whose combinatorial embedding (the cyclic order of the edges around each vertex) is given. In contrast to this, minimizing the number of bends over all embeddings of a maxdeg-4 planar graph is NP-hard [GT01]. Biedl and Kant [BK98] and Liu et al. [LMS98] have independently shown that any planar embedding of a maxdeg-4 planar graph admits a drawing on the $(n \times n)$-grid with at most $2n + 2$ bends in total and at most 2 bends per edge, with the exception of the octahedron, that requires one edge with 3 bends. Bläsius et al. [BKRW14] gave an algorithm that, given an embedded maxdeg-4 planar graph and a function flex: $E \to \mathbb{N}_{\geq 1}$, computes a drawing with at most flex(e) bends for every edge $e \in E$, if one exists.

In a so far unrelated line of research, circular-arc drawings of graphs have become popular in the last few years. Inspired by American artist Mark Lombardi (1951–2000), Duncan et al. [DEG$^+$12] introduced and studied *Lombardi drawings*, which are circular-arc drawings with the additional requirement of *perfect angular resolution*, that is, for each vertex, all pairs of consecutive edges form the same angle. Among others, Duncan et al. treat drawings of d-regular graphs, that is, graphs in which every vertex has degree d, where all vertices have to lie on one circle. They show that, under this restriction, Lombardi drawings can be constructed efficiently for some subclasses, whereas the problem is NP-hard for other subclasses. They also show that ordered trees can always be Lombardi drawn in polynomial area, whereas straight-line drawings with perfect resolution may need exponential area [DEG$^+$13].

Very recently, Bekos et al. [BKKS13] introduced the *smooth orthogonal* graph layout problem that combines the two worlds; the rigidity and clarity of orthogonal layouts with the artistic style and aesthetic appeal of Lombardi drawings. Formally, a smooth orthogonal drawing of a graph is a drawing on the plane where

(i) each vertex is drawn as a point

(ii) edges leave and enter vertices horizontally or vertically, and

(iii) each edge is drawn as an alternating sequence of axis-aligned line segments and circular-arc segments such that consecutive segments have a common *horizontal* or *vertical* tangent at their intersection point.

In the case of (maxdeg-4) planar graphs, it is additionally required that

(iv) there are no edge-crossings.

graph class	our contribution			Bekos et al. [BKKS13]
	complexity	area	Theorem	
biconnected maxdeg-4 planar	SC_2	super-poly	5.1	SC_3
maxdeg-4 planar	SC_2	super-poly	5.2	
maxdeg-3 planar	SC_2	$\lfloor n^2/4 \rfloor \times \lfloor n/2 \rfloor$	5.3	
biconnected maxdeg-4 outerplane	SC_1	exponential	5.4	
triconnected maxdeg-3 planar				SC_1
Hamiltonian maxdeg-3 planar				SC_1
maxdeg-4 planar, poly-area	$\not\sqsubseteq SC_1$		5.5	
OC_3, octahedron				$\not\subseteq SC_1$
OC_2	$\not\subseteq SC_1$		5.6	

**Table 5.1:** Comparison of our results to the results of Bekos et al. [BKKS13]

Note that, by construction, (smooth) orthogonal drawings of maxdeg-4 planar graphs have angular resolution within a factor of two of optimal.

The public transport map *Gmunden – Vöcklabruck – Salzkammergut*[1] of the Public Transportation Association Austria (OÖVG) shows a real-world example: a smooth orthogonal drawing of an Austrian regional bus and train map. Extending our model, the map has (multi-) edges that enter vertices diagonally.

For usability, it is important to keep the visual complexity of such drawings low. In a (smooth) orthogonal drawing, the *complexity* of an edge is the number of segments it consists of, that is, the number of inflection points plus one. Then, a natural optimization goal is to minimize, for a given (embedded) planar graph, the *edge complexity* of a drawing, which is defined as the maximum complexity over all edges. We say that a graph has *orthogonal complexity k* if it admits an orthogonal drawing of edge complexity at most k, for short, an OC_k*-layout*. Accordingly, we say that a graph has *smooth complexity k* if it admits a smooth orthogonal drawing of edge complexity at most k, for short, an SC_k*-layout*. We seek for drawings of maxdeg-4 planar graphs with low smooth complexity.

Our Contribution.   Known results and our contributions to smooth orthogonal drawings are shown in Table 5.1. The main result of this chapter is that any maxdeg-4 planar graph admits an SC_2-layout (see Sections 5.1 and 5.2). Our upper bound of 2 for the smooth complexity of maxdeg-4 planar graphs improves the previously known bound of 3 and matches the corresponding lower bound [BKKS13]. In contrast to the known algorithm for SC_3-layout [BKKS13], which is based on an algorithm for OC_3-layout of Biedl and Kant [BK98], we use an algorithm of Liu at al. [LMS98] for OC_3-layout, which avoids *S-shaped edges* (see

---

[1] http://almtal.salzkammergut.at/fileadmin/user_upload/almtal/pdf/Liniennetzplan_Salzkammergut_neu.pdf
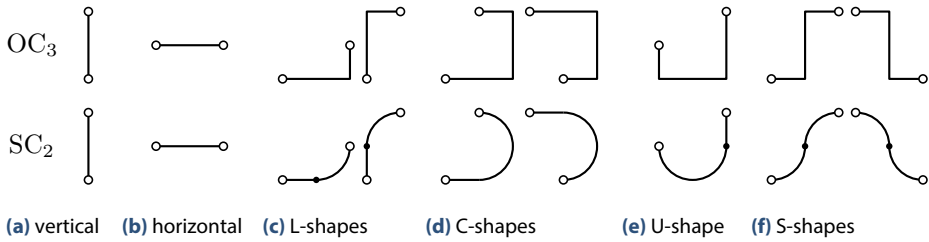
Figure 5.1: Converting shapes from the OC_3-layout to SC_2

Figure 5.1f, top). Such edges are undesirable since they force their endpoints to lie on a line of slope ±1 in a smooth orthogonal layout (see Figure 5.1f, bottom). Our construction requires super-polynomial area. Hence, we have made no effort in proving a concrete bound.

Further, we prove that every biconnected maxdeg-4 outerplane graph admits an SC_1-layout (see Section 5.3), expanding the class of graphs with SC_1-layout from triconnected or Hamiltonian maxdeg-3 planar graphs [BKKS13]. Note that in our result the outerplane embedding can be prescribed, while in the other results the algorithms need the freedom to choose an appropriate embedding.

We complement our positive results by two negative results. First, we show that there is an infinite family of graphs that admit SC_1-layouts but require exponential area; see Section 5.4. Second, we show that there is an infinite family of graphs that admit OC_2-layouts but do not admit SC_1-layouts; see Section 5.5.

5.1 Smooth Layouts for Biconnected Maxdeg-4 Planar Graphs

In this section, we prove that any biconnected maxdeg-4 planar graph admits an SC_2-layout. Given a biconnected maxdeg-4 planar graph, we first compute an OC_3-layout, using an algorithm of Liu et al. [LMS98]. Then, we turn the result of their algorithm into an SC_2-layout.

Liu et al. choose two vertices s and t and compute an st-ordering of the input graph. An st-ordering is an ordering $(s = 1, 2, \dots, n = t)$ of the vertices such that every vertex j $(2 < j < n - 1)$ has neighbors i and k with $i < j < k$. Then, they go through all vertices as prescribed by the st-ordering, placing vertex i in row i. Calling an edge of which exactly one endpoint is already drawn an *open edge*, they maintain the following invariant:

(I1) In each iteration, every open edge is associated with a *column* (a vertical grid line).

We define an *L-shaped edge* (for short, an *L-shape*) to be composed of a quarter-circle and a horizontal or vertical line-segment; see Figure 5.1c. A *C-shaped edge* (for short, a *C-shape*) is composed of a semicircle and (optionally) a vertical line segment; see Figure 5.1d. Similarly, a *U-shaped edge* (for short, a *U-shape*) is composed of a semicircle and (optionally) a horizontal line segment; see Figure 5.1e. Finally, an *S-shaped edge* (for short, an *S-shape*) is composed of
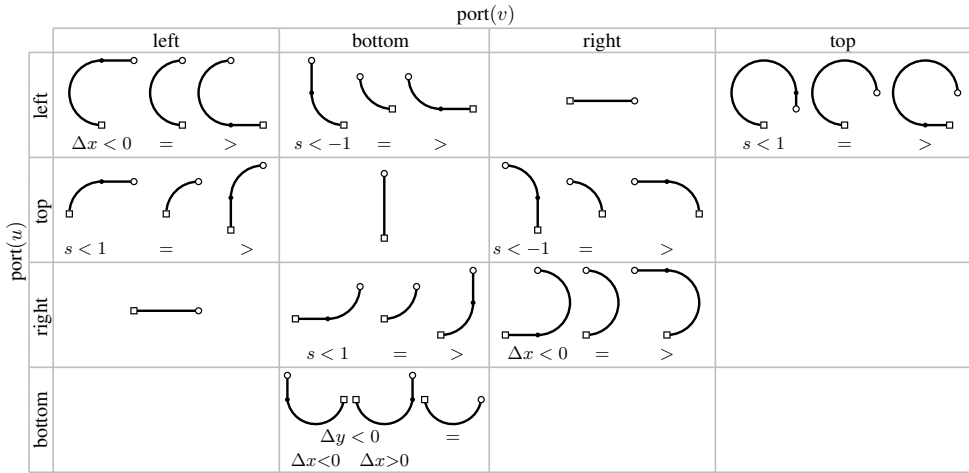
Figure 5.2: Cases for drawing the edge (u, v) based on the port assignment. In each case, u is the lower of the two vertices $(y(u) < y(v))$. As shorthand, we use $\Delta x = x(u) - x(v)$, $\Delta y = y(u) - y(v)$, and $s = \mathsf{slope}(u, v) = \Delta x / \Delta y$.

two quarter-circles; see Figure 5.1f (bottom); they are undesirable as they force their endpoints to lie on a line of slope ±1.

The algorithm of Biedl and Kant [BK98] yields an OC_3-layout similar to that of Liu et al. However, Liu et al. additionally show how to modify their algorithm such that it produces OC_3-layouts without the undesirable S-shapes.

In their modified algorithm, Liu et al. search for paths in the drawing that consist only of S-shapes; every vertex lies on at most one such path. They place all vertices on such a path in the same row, without changing their column. This essentially converts all S-shapes into horizontal edges. Now, every edge (except $(1, 2)$ and $(1, n)$) is drawn as a vertical segment, horizontal segment, L-shape, or C-shape; see Figure 5.1a–5.1d. The edge $(1, 2)$ is drawn as a U-shape and the edge $(1, n)$, if it exists, is either drawn as a C-shape or (only in the case of the octahedron) as a three-bend edge that uses the left port of vertex 1 and the top port of vertex n.

We convert the output of the algorithm of Liu et al. from OC_3 to SC_2. The coordinates of the vertices and the port assignment of their drawing define a (non-planar) SC_2-layout using the conversion table in Fig 5.2. In order to avoid crossings, we carefully determine new vertex positions scanning the drawing of Liu et al. from bottom to top.

We now introduce our main tool for the conversion: a *cut*, for us, is a y-monotone curve consisting of horizontal, vertical, and circular segments that divides the current drawing into a left and a right part, and only intersects horizontal segments and semicircles of the drawing. In the following, we describe how one can find such a cut from any starting point at the top of the drawing; see Figure 5.3. In spite of the fact that we define the cut going from top to bottom, "to its right" will, as usually, mean "with larger x-coordinate".

When such a cut encounters a vertex u to its right with an outgoing edge associated with its left port, then the cut continues by passing through the segment incident to u. On the other

(a) open edge   **(b)** L-shape   **(c)** C-shape   **(d)** maintaining (I2)
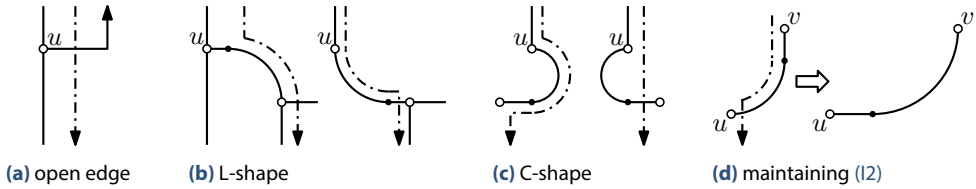
Figure 5.3: Finding a cut

hand, if the port has an incoming L-shaped or C-shaped edge, the cut just follows the edge. The case when the cut encounters a vertex to its left is handled symmetrically.

Let v be a vertex incident to two incoming C-shapes (u, v) and (w, v). If $y(w) \leq y(u)$ we call the C-shape (u, v) *protected* by (w, v); otherwise, we call it *unprotected*. In order to ensure that a cut passes only through horizontal segments and that our final drawing is planar, our algorithm will maintain the following new invariants:

(I2) An L-shape never contains a vertical segment (as in Figure 5.1c, right); it always contains a horizontal segment (as in Figure 5.1c, left) or a single quarter-circle.

(I3) An unprotected C-shape never contains a horizontal segment incident to its top vertex (as in Figure 5.1d, right); it always contains a horizontal segment incident to its bottom vertex (as in Figure 5.1d, left) or no straight-line segment.

(I4) The subgraph induced by the vertices that have already been drawn has the same embedding as in the drawing of Liu et al.

Below, we treat L- and C-shapes of complexity 1 as if they had a horizontal segment of length 0 incident to their bottom vertex. Note that we always cut around protected C-shapes, so we will never end up in their interior. Now, we are ready to state the main theorem of this section by presenting our algorithm for SC_2-layouts.

Theorem 5.1. *Every biconnected maxdeg-4 planar graph admits an SC_2-layout.*

Proof. In the drawing Γ of Liu et al., vertices are arranged in rows. Let $V_1, \ldots, V_r$ be the partition of the vertex set V in rows $1, \ldots, r$. Following Liu et al., the vertices in each such set induce a path in G. We place vertices in the order $V_1, \ldots, V_r$. In this process, we maintain a planar drawing Γ' and the invariants (I1) to (I4). As Liu et al., we place the vertices on the integer grid. We deal with the special edges $(1, 2)$ and $(1, n)$ at the end, leaving their ports, that is, the bottom and left port of vertex 1 and the top port of vertex n, open.

For invariant (I1), we associate each open edge with the column on which the algorithm of Liu et al. places it. If their algorithm draws the first segment of the open edge horizontally (from the source vertex to the column), we use the same segment for our drawing. We use the same ports for the edges as their algorithm. Thus, our drawing keeps the embedding of Liu et al., maintaining invariant (I4).

Assume that, for some $1 \leq i \leq n$, we have placed $V_1, \ldots, V_{i-1}$ and that the vertices in V_i are $v_1, \ldots, v_c$ in left-to-right order (the case $v_1 = v_c$ is possible; this is the only case in which a vertex can have incoming L- or C-shapes at both its left and right port). Vertex v_j $(1 \leq j \leq c)$ is

(a) OC$_3$-layout   **(b)** move v_1 up   **(c)** find a cut   **(d)** SC$_2$-layout   **(e)** protected C-shape
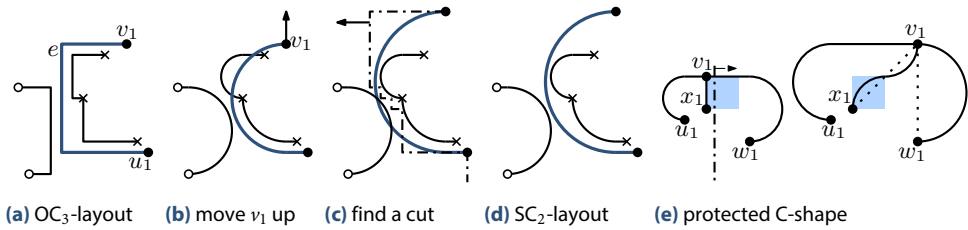
Figure 5.4: Handling C-shapes

placed in the column with which the edge entering the bottom port of v_j is associated. If the left port of v_1 is used by an incoming L- or C-shape $e = (u_1, v_1)$, we place v_1 (and the other vertices in V_i) on a row high enough such that a smooth drawing of e does not create any crossings with edges lying on the right side of e in Γ; see Figure 5.4b.

In order to make sure that the new drawing of e does not create crossings with edges on the left side of e in Γ, we need to "push" those edges to the left of e. We do this by computing a cut that starts from v_1, separates the vertices and edges that lie on the left side of e in Γ from those on the right side, passes u_1 slightly to the left, and continues downwards as described above; see Figure 5.4c. Since, by invariant (I4), our drawing so far is planar and each edge is drawn y-monotone, we can find a cut that is also y-monotone. We move everything on the left side of the cut further left such that e has no more crossings. Note that the cut intersects only horizontal edge segments. These will simply become longer by the move.

Let $\Delta x_i = x(v_i) - x(u_i)$ and $\Delta y_i = y(v_i) - y(u_i)$ for $i = 1, \ldots, c$. It is possible that the drawing of e violates invariant (I3)—if u_1 lies to the left of v_1. We consider two cases. First, assume that the edge (u_1, v_1) is the only incoming C-shape at v_1. Note that this is always the case if $c > 1$. In this case, we simply define a cut that starts slightly to the right of v_1, follows e, intersects e slightly to the left of u_1, and continues downwards. Then, we move everything on the left side of the cut by $\Delta x_1 + 1$ units to the left.

Next, assume that $c = 1$ and there is another C-shape (w_1, v_1) entering the right port of v_1; see Figure 5.4e (left). We assume, without loss of generality, that $y(w_1) \le y(u_1)$. Let (x_1, v_1) be the edge incident to the bottom port of v_1. In this case, we first find a cut that starts slightly to the right of v_1, follows (x_1, v_1), passes x_1 slightly to the right, and continues downwards. Then, we move everything on the right side of the cut by $y(v_1) - y(x_1)$ units to the right. Thus, there is an empty square to the right of (x_1, v_1) of side length $y(v_1) - y(x_1)$. Now, we place v_1 at the intersection of diagonal through x_1 with slope $+1$ and the vertical line through w_1. Due to this, we can draw (x_1, v_1) using two quarter-circles with a common horizontal tangent in the top right corner of the empty square; see Figure 5.4e (right). Note that the edge (u_1, v_1) is protected by (w_1, v_1), so it can have a horizontal segment incident to v_1. This establishes (I3).

It is also possible that the drawing of e violates invariant (I2)—if slope$(u_1, v_1) > 1$. In this case, we define a cut that starts slightly to the left of v_1, intersects e and continues downwards. Then, we move everything on the left side of the cut by Δy_1 units to the left. We treat v_c, the rightmost vertex in the current row, symmetrically to v_1. This establishes (I2).

For the case that v_1 does not have incoming C-shapes at both its left and right port, we still have to treat the edges entering vertices $v_1, \ldots, v_c$ from below. Note that these edges can

only be vertical or L-shaped. Vertical edges can be drawn without violating the invariants. However, invariant (I2) may be violated if an edge $e_i = (u_i, v_i)$ entering the bottom port of vertex v_i is L-shaped; see Figure 5.3d. Assume that $x(u_i) < x(v_i)$. In this case, we find a cut that starts slightly to the left of v_i, follows e_i, intersects e_i slightly to the right of u_i, and continues downwards. Then, we move everything on the left side of the cut by Δy_i units to the left. We handle the case $x(u_i) > x(v_i)$ symmetrically. This establishes (I2).

We thus place the vertices row by row from bottom to top and draw the incoming edges for the newly placed vertices, copying the embedding of the current subgraph from Γ. This completes the drawing of $G - \{(1, 2), (1, n)\}$. Note that vertex 1 has no incoming edge and vertex 2 has only one incoming edge, that is, $(1, 2)$. Thus, the bottom port of both vertices is still unused. We draw the edge $(1, 2)$ as a U-shape. Finally, we finish the layout by drawing the edge $(1, n)$, if it exists. By construction, the left port of vertex 1 is still unused. Note that vertex n has no outgoing edges, so the top port of n is still free. Hence, we can draw the edge $(1, n)$ as a horizontal or vertical segment followed by a three-quarter-circle. To avoid crossings, we may have to move vertex n upwards. This way, we will get a horizontal segment at vertex 1, and the three-quarter-circle will completely lie outside of the rest of the drawing. This completes the proof of Theorem 5.1. □

5.2 Smooth Layouts for Arbitrary Maxdeg-4 Planar Graphs

In this section, we describe how to create SC_2-layouts for arbitrary maxdeg-4 planar graphs. To achieve this, we decompose the graph into biconnected components, embed them separately, and then connect them. For the connection, it is important that one of the connector vertices lies on the outer face of its component. Within each component, the connector vertices have degree at most 3; if they have degree 2, we must make sure that their incident edges do not use opposite ports; otherwise, we cannot connect two components with a common connector vertex. Following Biedl and Kant [BK98], we say that a degree-2 vertex v is drawn *with right angle* if the edges incident to v use two neighboring ports.

Lemma 5.1. *Any biconnected maxdeg-4 planar graph admits an SC_2-layout such that all degree-2 vertices are drawn with right angle.*

Proof. Let v be a degree-2 vertex. We now show how to adjust the algorithm of Section 5.1 such that v is drawn with right angle. By construction, the top and the bottom ports of v are used. Let (u, v) be the edge entering v from below (we allow $v = 1$ and $u = 2$). We modify the algorithm such that (u, v) uses the left or right rather than the bottom port of v. We consider three cases; (u, v) is either L-shaped, U-shaped, or vertical. These cases are handled when v is inserted into the smooth orthogonal drawing.

First, we assume that (u, v) is L-shaped; see Figure 5.5a. Then, we can simply move v to the same row as u, making the edge horizontal.

Second, we assume that (u, v) is U-shaped; see Figures 5.5b, 5.5c. Then, $u = 1$ and $v = 2$ or vice versa. If both have degree 2, we move the higher vertex to the row of the lower vertex (if necessary) and replace the U-shaped edge by a horizontal edge. Otherwise, we move the
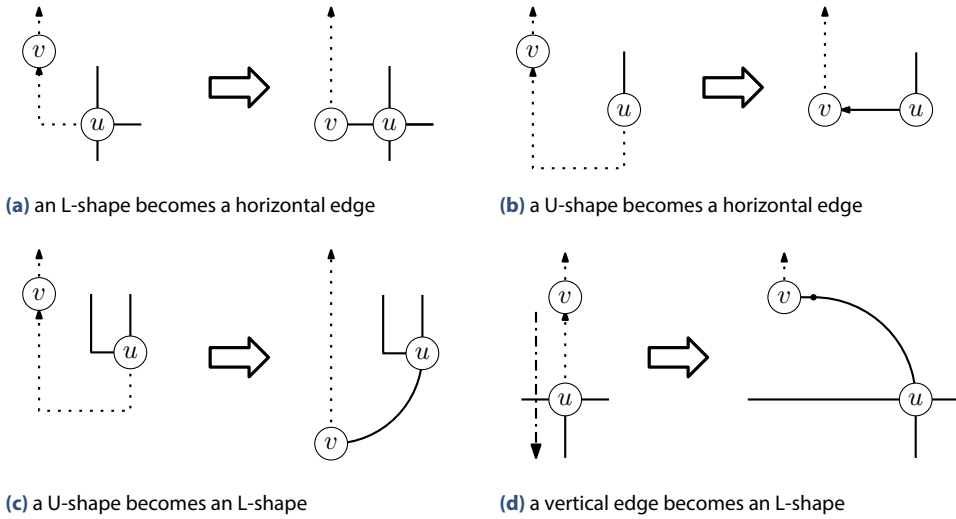
(a) an L-shape becomes a horizontal edge

(b) a U-shape becomes a horizontal edge

(c) a U-shape becomes an L-shape

(d) a vertical edge becomes an L-shape

Figure 5.5: Modification of the placement of degree-2 vertices

vertex with degree 2, say v, downwards to row $y(u) - \Delta x$ such that we can replace the U-shape by an L-shape.

Finally, we assume that (u, v) is vertical; see Figure 5.5d. Then, we compute a cut that starts slightly below v, follows (u, v) downwards, and passes u slightly to its left. We move all vertices (including u, but not v) that lie on the right side of this cut by at least Δy to the right. Then, we can draw (u, v) as an L-shape that uses the right port of v.

Observe that, in each of the three cases, we redraw all affected edges with SC_2. Hence, the modified algorithm still yields an SC_2-layout. At the same time, all degree-2 vertices are drawn with right angle as desired. □

Now, we describe how to connect the biconnected components. Recall that a *bridge* is an edge whose removal disconnects a graph G. We call the two endpoints of a bridge *bridge heads*. A *cut vertex* is a vertex whose removal disconnects the graph, but is not a bridge head.

Theorem 5.2. *Any maxdeg-4 planar graph admits an SC_2-layout.*

Proof. Let G_0 be some biconnected component of G, and let $v_1, \ldots, v_k$ be the cut vertices and bridge heads of G in G_0. For $i = 1, \ldots, k$, if v_i is a bridge head, let v_i' be the other head of the bridge; otherwise, let $v_i' = v_i$. Let G_i be the subgraph of G containing v_i' and the connected components of $G - v_i'$ not containing G_0. Following Lemma 5.1, G_0 can be drawn such that all degree-2 vertices are drawn with right angle.

The algorithm of Section 5.1 that we modified in the proof of Lemma 5.1 places the last vertex n at the top of the drawing and thus on the outer face. When drawing G_i, we choose v_i' as this vertex. By induction, G_i can be drawn such that all degree-2 vertices are drawn with right angle.

In order to connect G_i to G_0, we make G_0 large enough such that we can fit G_i into the face that contains the free ports of v_i. This face is unique because v_i is drawn with right angle.
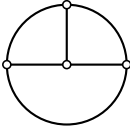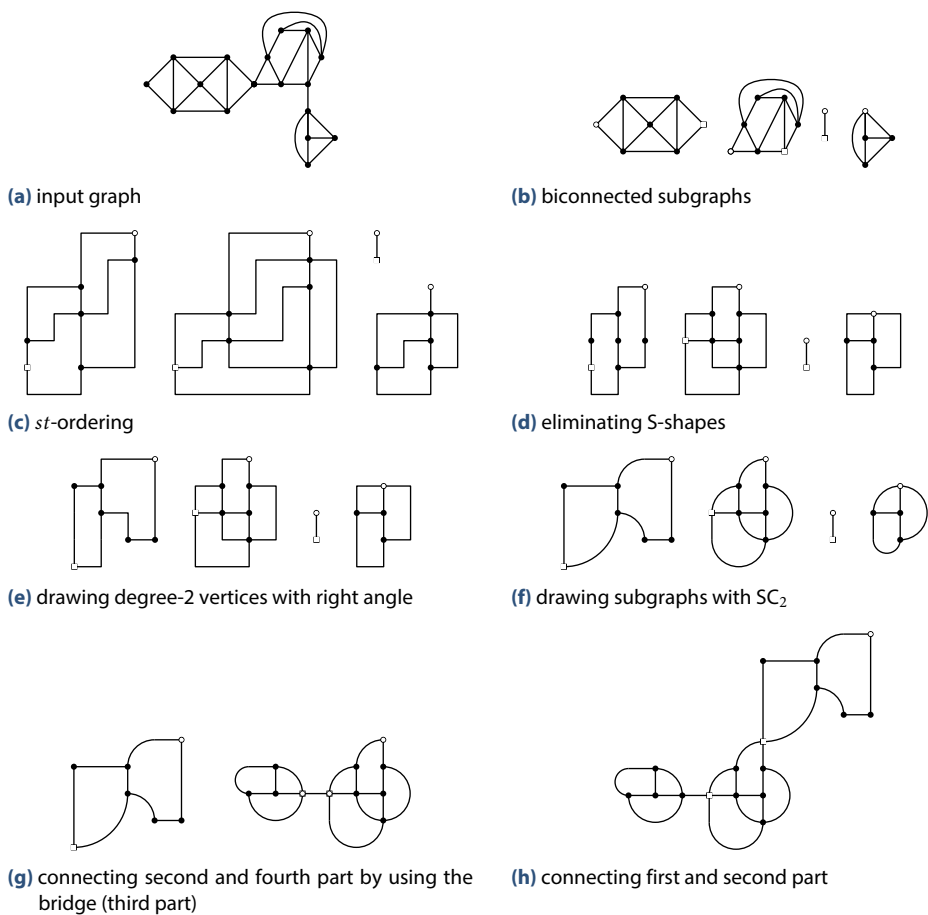
Figure 5.6: SC$_1$-layout of K_4.

We may have to rotate G_i by a multiple of 90° to achieve the following. If v_i is a cut vertex, we make sure that v_i' uses the ports of v_i that are free in G_0. Then, we identify v_i and v_i'. Otherwise, we make sure that a free port of v_i and a free port of v_i' are opposite. Then, we draw the bridge (v_i, v_i') horizontally or vertically. This completes our proof. $\qquad\square$

For an example run of our algorithm, see Figure 5.7.

For graphs of maximum degree 3, we can make our drawings more compact. This is due to the fact that we can avoid C-shaped edges (and hence cuts) completely. In the presence of L-shapes only, it suffices to stretch the orthogonal drawing by a factor of n.

Theorem 5.3. *Every biconnected maxdeg-3 planar graph with n vertices admits an SC$_2$-layout using area $\lfloor n^2/4 \rfloor \times \lfloor n/2 \rfloor$.*

Proof. It is known that every biconnected maxdeg-3 planar graph except K_4 has an OC$_2$-layout using area $\lfloor n/2 \rfloor \times \lfloor n/2 \rfloor$ from Kant [Kan96]. Now, we use the same global stretching as Bekos et al. [BKKS13, Theorem 2] when they showed that every OC$_2$-layout can be transformed into an SC$_2$-layout: we stretch the drawing horizontally by the height of the drawing, that is, by a factor of $\lfloor n/2 \rfloor$. This makes sure that we can replace every bend by a quarter-circle without introducing crossings. Figure 5.6 shows an SC$_1$-layout of K_4, completing our proof. $\qquad\square$

(a) input graph

(b) biconnected subgraphs

(c) st-ordering

(d) eliminating S-shapes

(e) drawing degree-2 vertices with right angle

(f) drawing subgraphs with SC_2

(g) connecting second and fourth part by using the bridge (third part)

(h) connecting first and second part

Figure 5.7: An example-run of our Algorithm for SC_2-layout. The circle vertices of component i correspond to the cut vertex v_i'. The square vertices correspond to cut vertices of other components.

5.3 SC$_1$-Layouts of Biconnected Maxdeg-4 Outerplane Graphs

In this section, we consider *maxdeg-4 outerplane* graphs, that is, maxdeg-4 outerplanar graphs with an outerplanar embedding. We prove that any biconnected maxdeg-4 outerplane graph admits an SC$_1$-layout. To do so, we first prove the result for a subclass of maxdeg-4 outerplane graphs, which we call $(2,3)$-*restricted* outerplane graphs; then, we generalize to maxdeg-4 outerplane graphs. We call a maxdeg-4 outerplane graph $(2,3)$-restricted if it contains a pair x and y of consecutive vertices on the outer face with $\deg(x) = 2$ and $\deg(y) \leq 3$. Recall that the *weak dual* of a plane graph is the subgraph of the dual graph whose vertices correspond to the bounded faces of the primal graph.

Lemma 5.2. *Any biconnected $(2,3)$-restricted maxdeg-4 outerplane graph admits an SC$_1$-layout.*

Proof. Let x and y be two consecutive vertices on the outer face of the given graph G with $\deg(x) = 2$ and $\deg(y) \leq 3$. Let also T be the weak dual tree of G rooted at the node v^* of T corresponding to the bounded face f^* that contains both x and y. We construct the SC$_1$-layout Γ of G by traversing T, starting with v^*. When we traverse a node of T, we draw the corresponding face of G with SC$_1$.

Consider the case that we have constructed a drawing $\Gamma(H)$ for a connected subgraph H of G and we want to add a new face f to $\Gamma(H)$. For each vertex u of H, we denote by $p_u = (x(u), y(u))$ the point at which u is drawn in $\Gamma(H)$. The *remaining degree* of u is the number of vertices adjacent to u in $G - H$. Since we construct $\Gamma(H)$ face by face, the remaining degree of each vertex in H is at most two. We call the ports of u that are not occupied by an edge of H in $\Gamma(H)$ *free ports*. During the construction of Γ, we maintain the following four invariants:

(J1) $\Gamma(H)$ is an SC$_1$-layout that preserves the planar embedding of G, and each edge is drawn either as an axis-parallel line segment or as a quarter-circle in $\Gamma(H)$. (Note that we do not use semi- and 3/4-circles.)

(J2) For each vertex u of H, the free ports of u in $\Gamma(H)$ are consecutive around u, and they point to the outer face of $\Gamma(H)$.

(J3) Vertices with remaining degree exactly 2 are incident to an edge drawn as a quarter-circle.

(J4) If an edge (u,v) is drawn as an axis-parallel segment, then at least one of u and v has remaining degree at most 1. If (u,v) is vertical and $y(u) < y(v)$, then u has remaining degree at most 1 and the free port of u in $\Gamma(H)$ is horizontal; see Figures. 5.8a, 5.8d and 5.8g. Symmetrically, if (u,v) is horizontal and $x(u) < x(v)$, then u has remaining degree at most 1 and the free port of u in $\Gamma(H)$ is vertical; see Figures 5.8b, 5.8e and 5.8h.

We now show how to add the drawing of the new face f to $\Gamma(H)$. Since G is biconnected and outerplanar, and due to the order in which we process the faces of G, f has exactly two vertices u and v which have already been drawn at p_u and p_v, respectively. The two vertices
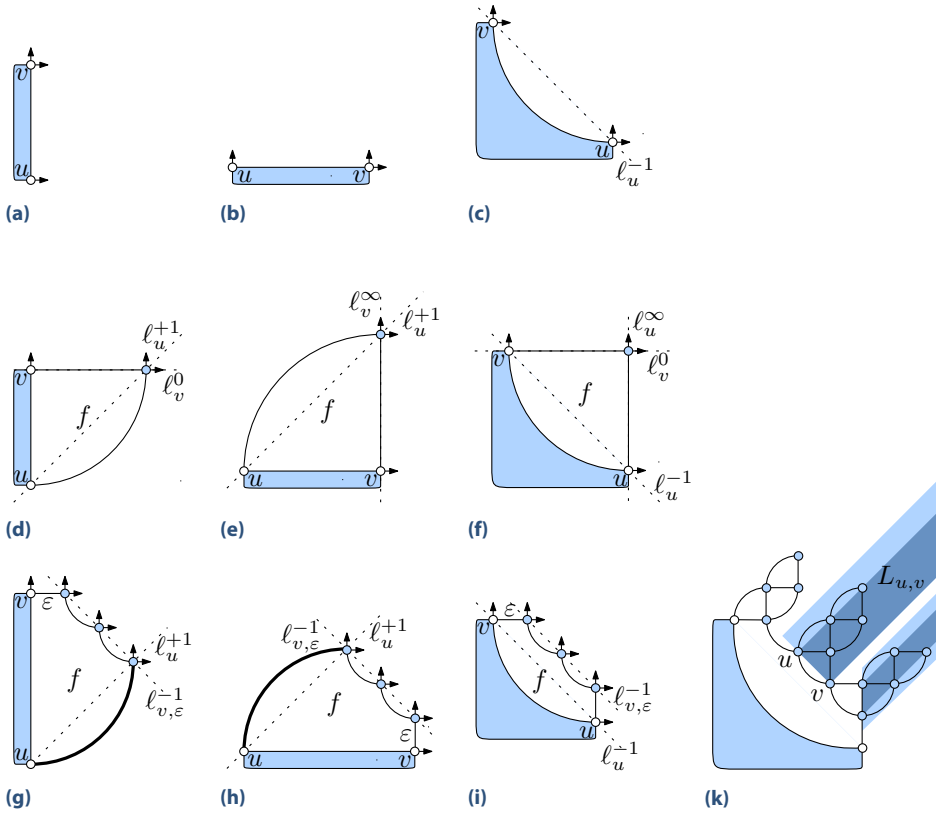
Figure 5.8: (a)–(i) Different cases that arise when drawing face f of G; (k) a sample drawing

are adjacent. Depending on how the edge (u, v) is drawn in $\Gamma(H)$, we draw the remaining vertices and edges of f, as follows.

Let $k \geq 3$ be the number of vertices on the boundary of f. The slope of the line segment $\overline{p_u p_v}$ is in $\{-1, 0, +1, \infty\}$, where ∞ means that $\overline{p_u p_v}$ is vertical. For $s \in \{-1, 0, +1, \infty\}$, we denote by ℓ_u^s the line with slope s through p_u. Similarly, we denote by $\ell_{u,\varepsilon}^s$ the line with slope s through the point $(x(u) + \varepsilon, y(u))$, for some $\varepsilon > 0$. Figures 5.8d–5.8f show the drawing of f for $k = 3$, and Figures 5.8g–5.8i for any $k \geq 4$.

Note that the lengths of the line segments and the radii of the quarter-circles that form f are equal (except for the radii of the bold-drawn quarter-circles of Figures 5.8g and 5.8h which are determined by the remaining edges of f). Hence, the lengths of the line segments and the radii of the quarter-circles that form any face that is descendant of face f in T are smaller than or equal to the lengths of the line segments and the radii of the quarter-circles that form f.

Our construction ensures that all vertices of the subgraph of G induced by the subtree of T rooted in f lie in the interior or on the boundary of the diagonal semi-strip L_{uv} delimited by ℓ_u^{+1}, ℓ_v^{+1}, and $\overline{p_u p_v}$; see Figure 5.8k. The only edges of this subgraph that are drawn in the complement of L_{uv}—and are potentially involved in crossings—are incident to two vertices

that both lie on the boundary of L_{uv}. In this particular case, however, the degree restriction implies that L_{uv} is surrounded from above and/or below by two empty diagonal semi-strips of at least half the width of the semi-strip L_{uv}, which suffices to ensure planarity for the following two reasons.

First, any face that is descendant of face f in T is formed by line segments and quarter-circles that are at most as long as the corresponding ones of face f. Second, due to the degree restrictions, if two neighboring children of f are triangles, the left one cannot have a right child, and vice versa.

Let us summarize. Figures 5.8d–5.8i show that the drawing of f ensures that the invariants (J1)–(J4) of our algorithm are satisfied for $H \cup \{f\}$. We begin by drawing the root face f^*. Since G is $(2,3)$-restricted, f^* has two vertices x and y consecutive on the outer face with $\deg(x) = 2$ and $\deg(y) \leq 3$. We draw the edge (x, y) as a vertical line segment. Then, the remaining degrees of x and y are 1 and 2, respectively, which satisfies the invariants for face f^*. Hence, we complete the drawing of f^* as in Figure 5.8d or 5.8g. Traversing T in pre-order, we complete the drawing of G. □

Suppose that the input graph G is not $(2,3)$-restricted. As the following lemma asserts, we can always construct a biconnected $(2,3)$-restricted maxdeg-4 outerplane graph by deleting a vertex of degree 2 from G.

Lemma 5.3. *Let $G = (V, E)$ be a non-$(2,3)$-restricted biconnected maxdeg-4 outerplane graph. Then, G has a degree-2 vertex whose removal yields a $(2,3)$-restricted biconnected maxdeg-4 outerplane graph.*

Proof. The proof is by induction on the number of vertices. The base case is a maximal biconnected outerplane graph on six vertices, which is the only non-$(2,3)$-restricted graph with six or less vertices. It is easy to see that in this case the removal of any degree-2 vertex yields a biconnected $(2,3)$-restricted maxdeg-4 outerplane graph. Now, assume that the hypothesis holds for any biconnected maxdeg-4 outerplane graph with $k \geq 6$ vertices.

Let G_{k+1} be a biconnected maxdeg-4 outerplane graph on $k + 1$ vertices which is not $(2,3)$-restricted. Let $\mathcal{F}$ be a face of G_{k+1} that is a leaf in its weak dual. Then, $\mathcal{F}$ contains only one internal edge and exactly two external edges since, if it contained more than two external edges, G_{k+1} would be $(2,3)$-restricted. Therefore, $\mathcal{F}$ consists of three consecutive vertices a, b and c on the outer face with $\deg(a) = \deg(c) = 4$ and $\deg(b)$, since, otherwise, G_{k+1} would be $(2,3)$-restricted. By removing b, we obtain a new graph G_k on k vertices. If a or c is incident to a degree-2 vertex in G_k, then G_k is $(2,3)$-restricted. Otherwise, by our induction hypothesis, G_k has a degree-2 vertex whose removal yields a $(2,3)$-restricted outerplanar graph. Since this vertex is neither adjacent to a nor c, the removal of this vertex makes G_{k+1} also $(2,3)$-restricted. □

Now, we are ready to deal with general biconnected maxdeg-4 outerplane graphs.

Theorem 5.4. *Any biconnected maxdeg-4 outerplane graph admits an SC_1-layout.*

Proof. If the given graph G is $(2,3)$-restricted, the result follows from Lemma 5.2. Thus, assume that G is not $(2,3)$-restricted. Then, G contains a degree-2 vertex b whose removal yields a biconnected $(2,3)$-restricted maxdeg-4 outerplane graph G'. Hence, we can apply
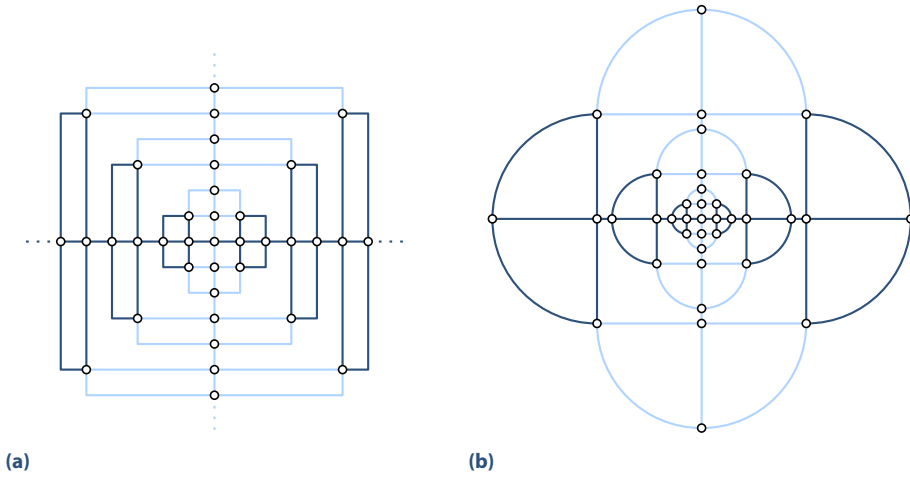
(a)   **(b)**

Figure 5.9: (a) A graph with an OC_2-layout using polynomial area and (b) an SC_1-layout using exponential area.

the algorithm of Lemma 5.2 to G' and obtain an outerplanar SC_1-layout $\Gamma(G')$ of G'. Since this algorithm always maintains consecutive free ports for each vertex, and the neighbors of b lie on the outer face of $\Gamma(G')$, we can insert b and its two incident edges to obtain an SC_1-layout $\Gamma(G)$ of G as follows. Let a and c be the neighbors of b and assume, without loss of generality, that c is drawn above a. If the edge (a, c) is drawn as a quarter-circle, then a 3/4-circle arc from p_c to p_b and a quarter-circle from p_b to p_a suffice. Otherwise, we can use the line segment $\overline{p_a p_b}$ and a quarter-circle from p_b to p_c. This concludes our proof. □

5.4 A Lower Bound for the Area Requirement of SC_1-Layouts

In this section, we demonstrate an infinite family of maxdeg-4 planar graphs that require exponential area if they are drawn with SC_1. Bekos et al. [BKKS13] presented such a family of graphs for the rather restricted setting where both the embedding of the graph and the port assignment of the edges are fixed. Here, we strengthen this result. Consider the graph depicted in Figure 5.9a. This graph consists of several layers. Each layer consists of a cycle of four pairs of adjacent triangles. The SC_1-layout of this graph in Figure 5.9b obviously requires exponential area since every layer uses more than twice the area of the previous layer. We will now show that this is the only SC_1-layout of the graph, up to translation, rotation and scaling.

First, we show that there are only two ways to draw one of the triangles of each layer. In Figure 5.10, we show all 16 possible ways to get an SC_1-layout of a triangle. However, in our graph all free ports have to lie on the outer face. There are only two SC_1-layouts of a triangle that have this property, marked by a dashed circle.

Next, we build a pair of adjacent triangles. In Figure 5.11, we show that there are three ways to combine two triangles that share an edge. Finally, we combine four pairs of adjacent
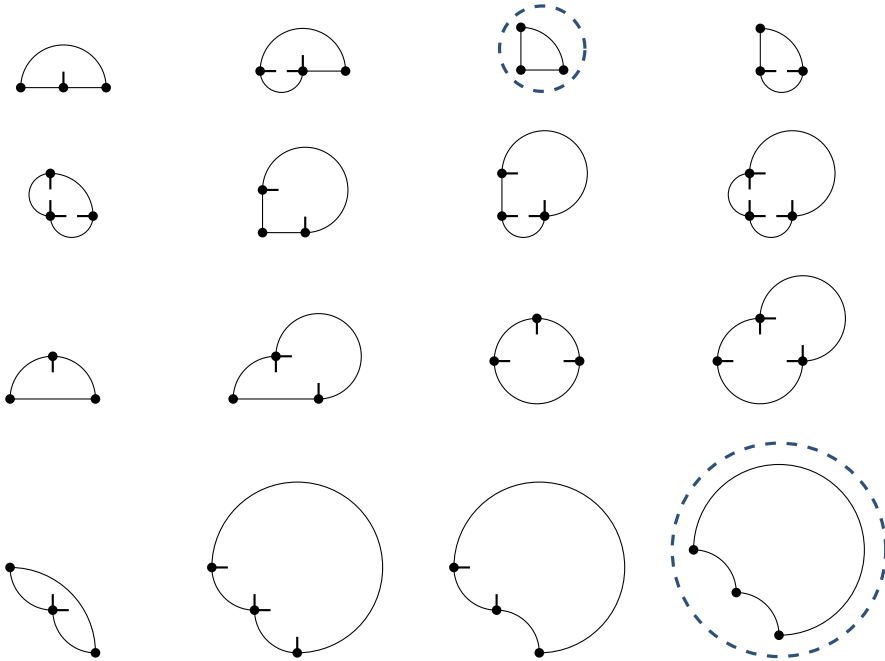
Figure 5.10: All possible ways to get an SC_1-layout of a triangle. Only two of these drawings (enclosed by dashed circles) have all their ports on the outer face.

triangles to one layer of the graph. Using careful case analysis, it can be shown that there are only two ways to draw one of the layers with SC_1; see Figure 5.12. However, it is easy to see that it is impossible to connect the drawing shown in Figure 5.12c to another layer. Thus, the SC_1-layout shown in Figure 5.9b is the only way to draw this graph, which proves the following theorem.

Theorem 5.5. *There is an infinite family of graphs that require exponential area if they are drawn with SC_1.*
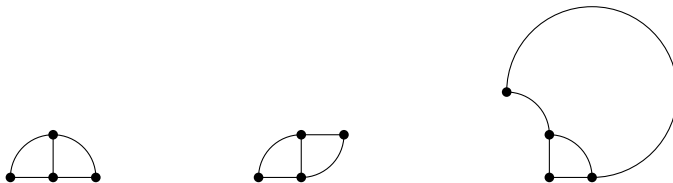
Figure 5.11: There are three ways to draw two adjacent triangles.

(a)   **(b)**   **(c)**
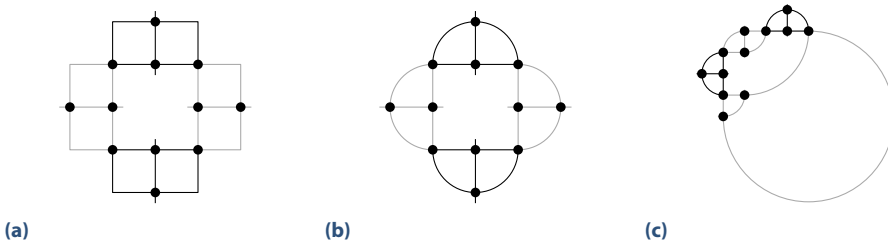
Figure 5.12: (a) One layer of the graph in Figure 5.9. (b) & (c) The only two ways to draw the subgraph depicted in (a) with SC$_1$.

5.5 Biconnected Graphs without SC$_1$-Layouts

In this section, we demonstrate an infinite family of biconnected maxdeg-4 planar graphs that admit OC$_2$-layouts, but do not admit SC$_1$-layouts. Bekos et al. [BKKS13] presented such a family of graphs assuming a rather restricted setting in which the choice of the outer face is fixed and always corresponds to a triangle. Here, we strengthen this result by providing an infinite family of biconnected maxdeg-4 planar graphs that admit no SC$_1$-layout in any embedding. We start with the following lemma.

Lemma 5.4. *There exists a biconnected maxdeg-4 planar graph that admits an OC$_2$-layout, but does not admit an SC$_1$-layout.*

Proof. Let G be the graph of Figure 5.13a. We prove that G has no SC$_1$-layout. First, note that G contains two copies of the graph depicted in Figure 5.13b. We denote this graph by H. We first prove that H has no SC$_1$-layout with the given embedding. In particular, we show that the subgraph of H induced by the vertices on or inside the black cycle cannot be drawn with SC$_1$.

(a) the graph G **(b)** the important subgraph H in detail

Figure 5.13: A graph G that admits an OC$_2$-layout, but does not admit an SC$_1$-layout
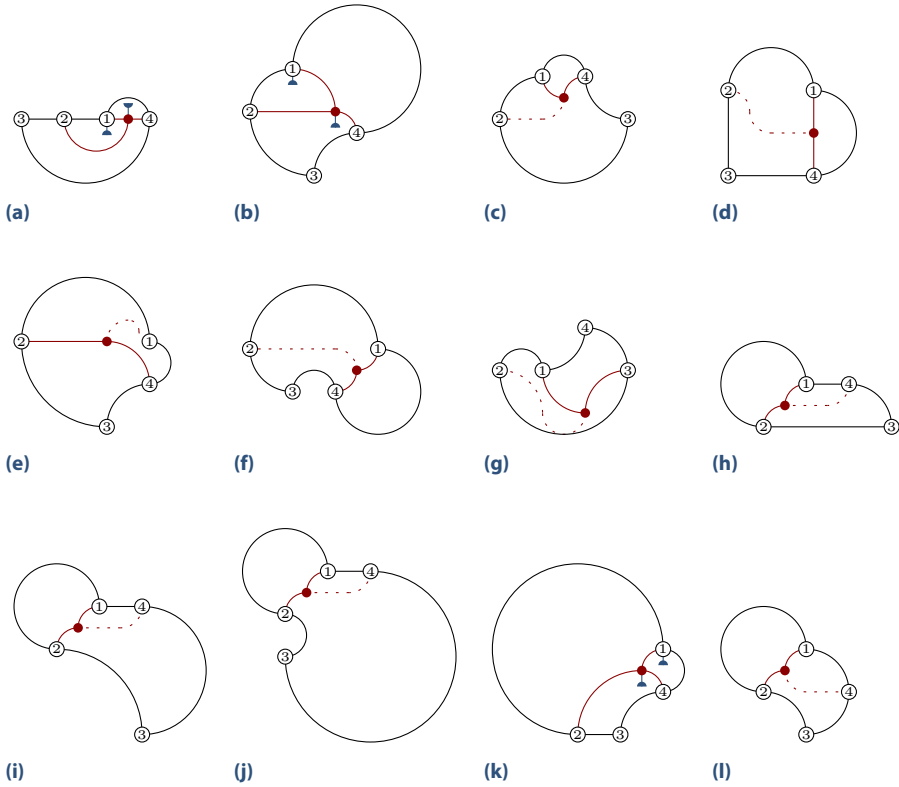
Figure 5.14: Illustration for the proof of Lemma 5.4

Consider edge $e = (1, 2)$ of H. This edge can be drawn as a straight-line segment, quarter-circle, semicircle or 3/4-circle. Figure 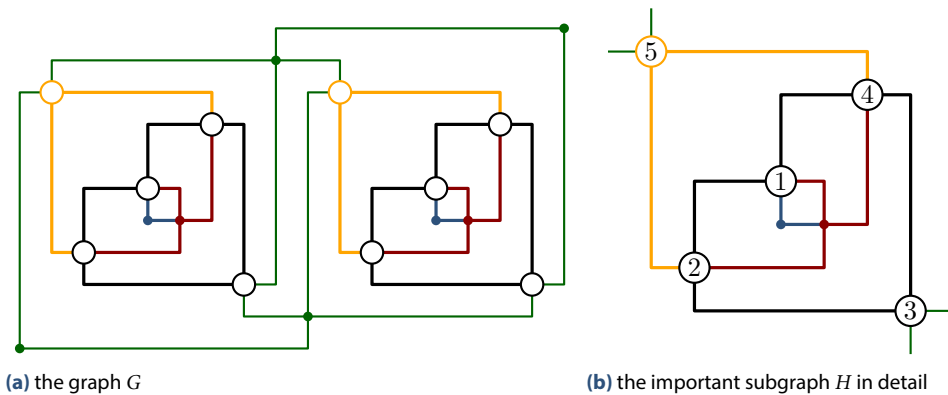5.14a illustrates the case that e is drawn as a horizontal line segment. In this case, the ports for the edges are fixed due to the given embedding and it is not possible to complete the drawing. The case that e is drawn as a vertical segment is analogous. Similarly, we show that there is no SC_1-layout for H if e is drawn as a quarter-circle in Figures 5.14b–5.14c, as a semicircle in Figures 5.14d–5.14g and as a 3/4-circle in Figures 5.14h–5.14l. Thus, there is no SC_1-layout for this fixed embedding of H.

Next, we claim that there is no SC_1-layout for any embedding of H where the vertices 2, 3, 4, and 5 define the outer cycle. Indeed, if the outer face is fixed, then the only way to find a different embedding is to find a separating pair $\{u, v\}$ in H and "flip" one of the components of $H - \{u, v\}$. There are two possible separating pairs in H: (i) vertex 1 and the red vertex; then, the flip with respect to this pair gives an isomorphic graph due to symmetry; and (ii) vertices 2 and 4; then, the flip with respect to this pair again gives an isomorphic graph by interchanging the role of 3 and 5. Thus, with the fixed outer cycle $(2, 3, 4, 5)$, all possible embeddings of H are isomorphic.

Since G contains two copies of H, in any embedding of G, at least one of the copies will retain its outer cycle. Hence, there is no SC_1-drawing for any embedding of G. □

Graph G of Figure 5.13a uses a few short paths to connect two copies of H. Obviously, we can add an arbitrary number of vertices to these paths such that the augmented graph remains biconnected and maxdeg-4 planar. This proves the following theorem.

Theorem 5.6. *There is an infinite family of biconnected maxdeg-4 planar graphs that admit OC_2-layouts but do not admit SC_1-layouts.*

5.6 Concluding Remarks

In this chapter, we have presented several new results about smooth orthogonal drawings of maxdeg-4 planar graphs. However, many problems remain open. We have shown that maxdeg-3 planar graphs can be drawn in polynomial area with SC_2, but it is unknown for maxdeg-4 planar graphs, as our algorithm requires exponential area. The graph classes admitting SC_1-layouts have not been fully characterized. It was already known that Hamiltonian maxdeg-3 planar and triconnected maxdeg-3 planar graphs always admit a SC_1-layout, and we extended these results to biconnected maxdeg-4 outerplane graphs. It remains open whether there are larger classes of graphs that admit SC_1-layouts, such as maxdeg-4 outerplanar or maxdeg-3 planar graphs. We strongly conjecture that it is NP-hard to decide whether a maxdeg-4 planar graph has an SC_1-layout, but we struggled with some details in our attempt for a proof.

6 Monotone Drawings of Trees

A natural requirement for the layout of a connected graph is that between any source vertex and any target vertex, there should be a source–target path that approaches the target according to some distance measure. A large body of literature deals with problems of this type; various measures have been studied. For example, in a *greedy drawing* you can find a path to a target vertex by iteratively selecting a neighbor that is closer to the target.

In a *monotone* drawing, the distance between vertices (on the desired source-target path) is measured with respect to their projections on *some* line, which may be different for any source–target pair. We say that a path P is *monotone with respect to* a vector $\vec{v}$ if the orthogonal projection of the vertices of P on every line with direction vector $\vec{v}$ appear on the line in the order induced by P. We also refer to $\vec{v}$ as a *direction*. In *strongly monotone* drawings, that line is always the line from source to target, and in *upward* drawings, the line is always the vertical line, directed upwards.

In this chapter, we focus on monotone and strongly monotone drawings of trees with additional aesthetic properties such as convexity or small area. Given a tree, we call the edges incident to the leaves *leaf edges* and all other edges *interior edges*. Given a straight-line drawing of a tree, we substitute each leaf edge by a ray whose initial part coincides with the edge. The embedding of the tree defines a combinatorial embedding of the tree, that is, the order of the edges around every vertex. The faces are then specified by this combinatorial embedding as leaf–leaf paths. If the faces of the augmented drawing are realized as convex non-overlapping (unbounded) polygonal regions, then we call the original drawing a *convex drawing*. If every region is *strictly convex* (that is, all interior angles are strictly less than π), we also call the drawing *strictly convex*.

Note that a convex drawing is also monotone [ACM89, ACB+12], but a monotone drawing is not necessarily convex. Strict convexity forbids vertices of degree 2. In this chapter, when we talk about (strongly) monotone drawings, this always includes the planarity requirement. Otherwise, as Angelini et al. [ACB+12] observed, drawing any spanning tree of the given graph in a (strongly) monotone way and inserting the remaining edges would yield a (strongly) monotone drawing of the graph.

Previous Work.　Rao et al. [RRP+03] introduced the concept of greedy drawings for a coordinate-based routing algorithm that does not rely on location information. While any 3-connected plane graph has a greedy drawing in the Euclidean plane [LM10] (even without crossing [Dha10]), this is, unfortunately, not true for trees. Nöllenburg and Prutkin [NP13] gave a complete characterization for the tree case, which shows that no tree with a vertex of degree 6 or more admits a greedy drawing.

Alamdari et al. [ACG+13] studied a subclass of greedy drawings, so-called *self-approaching drawings* which require that there always is a source–target path such that the distance de-

creases for any triplet of intermediate points on the *edges*, not only for the vertices on the path. These drawings are based on the concept of self-approaching curves [IKL95].

Carlson and Eppstein [CE07] studied convex drawings of trees. They give linear-time algorithms that optimize the angular resolution of the drawings, both for the fixed- and the variable-embedding case. They observe that convexity allows them to pick edge lengths arbitrarily, without introducing crossings.

For monotone drawings, Angelini et al. [ACB$^+$12] studied the variable-embedding case. They showed that any n-vertex tree admits a straight-line monotone drawing on a grid of size $O(n^{1.6}) \times O(n^{1.6})$ (using a BFS-based algorithm) or $O(n) \times O(n^2)$ (using a DFS-based algorithm). They also showed that any biconnected planar graph has a monotone drawing (using exponential area). Further, they observed that not every planar graph admits a monotone drawing if its embedding is fixed. They introduced the concept of *strong monotonicity* and showed that there is a drawing of a planar triangulation that is not strongly monotone. Hossain and Rahman [HR14] improve some of the results of Angelini et al. by showing that every connected planar graph admits a monotone drawing of size $O(n) \times O(n^2)$ and that such a drawing can be computed in linear time.

Both the BFS- and the DFS-based algorithms of Angelini et al. precompute a set of $n-1$ vectors in decreasing order of slope by using two different partial traversals of the so-called *Stern–Brocot tree*, an infinite tree whose vertices are in bijection with the irreducible positive rational numbers. Such numbers can be seen as *primitive* vectors in 2d, that is, vectors with pairwise different slopes. Then, both algorithms do a pre-order traversal of the input tree. Whenever they hit a new edge, they assign to it the steepest unused vector. They place the root of the input tree at the origin and draw each edge (u, v) by adding its assigned vector to the position of u. They call such tree drawings *slope-disjoint*. We will not formally define this notion here, but it is not hard to see that it implies monotonicity.

Angelini, with a different set of co-authors [ADK$^+$13], investigated the fixed-embedding case. They showed that, on the $O(n) \times O(n^2)$ grid, every connected plane graph admits a monotone drawing with two bends per edge and any outerplane graph admits a straight-line monotone drawing.

Our contribution.   We present two main results. First, we show that any n-vertex tree admits a strictly convex and, hence, monotone drawing on the $O(n^{1.5}) \times O(n^{1.5})$ grid (see Section 6.2). As the drawings of Angelini et al. [ACB$^+$12], our drawings are slope-disjoint, but we use a different set of primitive vectors (based on Farey sequences), which slightly decreases the grid size. (This also works for the BFS-based algorithm of Angelini et al.) Instead of pre-order, we use a kind of in-order traversal (first child – root – other children) of the input tree, which helps us to achieve convexity. Our ideas can be applied to modify the optimal angular resolution algorithm of Carlson and Eppstein [CE07] such that a drawing on an $O(n^{1.5}) \times O(n^{1.5})$ grid is constructed at the expense of missing the optimal angular resolution by a constant factor. Second, we show that any tree admits a *strongly* monotone drawing (see Section 6.3). So far, no positive results have been known for strongly monotone drawings.

In the case of proper binary trees, our drawings are additionally strictly convex. For biconnected outerplanar graphs, it is easy to construct strongly monotone drawings. On the other hand, we present a simply-connected planar graph that does not have a strongly monotone drawing in any embedding.
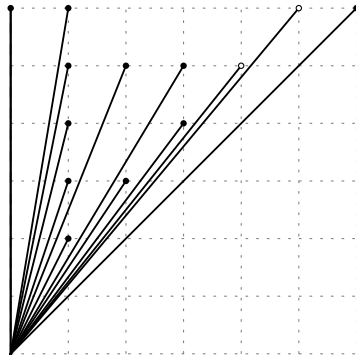
Figure 6.1: The 13 primitive vectors obtained from $\mathcal{F}_6$. The smallest angle of $\approx 1.14°$ is realized between the vectors $(4, 5)$ and $(5, 6)$ marked with white dots; the best possible angular resolution in this case is $45°/12 = 3.75°$.

6.1 Building Blocks: Primitive Vectors

The following algorithms require a set of integral vectors with distinct directed slopes and bounded length. In particular, we ask for a set of *primitive vectors* $P_d = \{(x, y) \mid \gcd(x, y) \in \{1, d\}, 0 \le x \le y \le d\}$. Our goal is to find the right value of d such that P_d contains at least k primitive vectors, where k is a number that we determine later. We can then use the reflections on the lines $x = y$, $y = 0$ and $x = 0$ to get a sufficiently large set of integer vectors with distinct directed slopes. The edges of the monotone drawings in Section 6.2 are translates of these vectors; each edge uses a different vector.

Assume that we have fixed d and want to generate the set P_d. If we consider each entry (x, y) of P_d to be a rational number x/y and order these numbers by value, we get the *Farey sequence* $\mathcal{F}_d$ (see, for example, Hardy and Wright's book [HW79]). The Farey sequence is well understood. In particular, it is known that $|\mathcal{F}_d| = 3d^2/\pi^2 + O(d \log d)$ [HW79, Theorem 331]. Furthermore, the entries of $\mathcal{F}_d$ can be computed in time $O(|\mathcal{F}_d|)$. We remark that the set $\bigcup_d \mathcal{F}_d$ coincides with the entries of the Stern–Brocot tree. However, collecting the latter level by level is not the most effective method to build a set of primitive vectors for our purpose.

To obtain a set of k primitive vectors, we use the first k entries of the Farey sequence $\mathcal{F}_d$, for $d := 4\lceil\sqrt{k}\rceil$, replacing each rational by its corresponding two-dimensional vector. We select exactly k primitive vectors from this set which we denote by V_k; see Figure 6.1.

If we wish to have more control over the aspect ratio in our final drawing, we can pick a set of primitive vectors contained inside a triangle spanned by the grid points $(0, 0)$, $(m_x, 0), (m_x, m_y)$. By stretching the triangle and keeping its area fixed, we may end up with fewer primitive vectors. This will result in an (only slightly) smaller constant compared to the case $m_x = m_y$. As proven by Bárány and Rote [BR06, Theorem 2], any such triangular domain contains at least $m_x m_y/4$ primitive vectors. This implies that we can adapt the algorithm easily to control the aspect ratio by selecting the box for the primitive vectors accordingly. For the sake of simplicity, we detail our algorithms only for the most interesting case $(m_x = m_y)$.

Lemma 6.1. *Let $P \subseteq P_d$ be a set of $k = |P_d|/c$ primitive vectors with no coordinate greater than d for some constant $c \geq 1$. Then, any two primitive vectors of P are separated by an angle of $\Omega(1/k)$.*

Proof. Since $|P_d| = 3d^2/\pi^2 + O(d \log d)$, we have that $2d^2 \approx 2\pi^2 ck/3$. Any line with slope m encloses an angle α with the x-axis, such that $\tan(\alpha) = m$. Let m_1 and m_2 be the slopes of two lines and let α_1 and α_2 be the corresponding angles with respect to the x-axis. By the trigonometric addition formulas we have that the separating angle of these two lines equals

$$\tan \phi := \tan(\alpha_1 - \alpha_2) = \frac{\tan \alpha_1 - \tan \alpha_2}{1 + \tan \alpha_1 \tan \alpha_2} = \frac{m_1 - m_2}{1 + m_1 m_2}.$$

For any two neighboring entries p/q and r/s in the Farey sequence, it holds that $qr - ps = 1$ [HW79, Theorem 3.1.2], and therefore p/q and r/s differ by exactly $(qr - ps)/(qs) = 1/(qs)$. As a consequence, $\tan \phi = 1/(pr + qs)$. The angle ϕ is minimized if $pr + qs$ is maximized. Clearly, we have that $pr + qs < 2d^2 \approx 2\pi^2 ck/3$. By the Taylor expansion, $\arctan(x) = x - x^2 \xi/(1 + \xi^2)^2$ for some value $0 \leq \xi \leq x$. Substituting x with $3/(2\pi^2 ck)$ yields, for $k \geq 2$, that

$$\phi \geq \frac{3}{2\pi^2 ck} - \frac{9\xi}{4\pi^4 c^2 k^2 (1 + \xi^2)^2} > \frac{3}{2\pi^2 ck} - \frac{9}{4\pi^4 c^2 k^2} \in \Omega(1/k).$$

$\square$

Since the best possible resolution for a set of k primitive vectors is $2\pi/k$, Lemma 6.1 shows that the resolution of our set differs from the optimum by at most a constant. To estimate this constant, let us assume we use $k = |P_d|$ primitive vectors (that is, $c = 1$ in Lemma 6.1). Then, the smallest angle ϕ spanned by these vectors is, according to the proof of the previous lemma, at least $3/(2\pi^2 k) - 9/(16\pi^4)$ for any $k > 1$. This value should be compared to opt $= \pi/(4k)$ since the primitive vectors span an angle of $\pi/4$ in total. We obtain that the ratio $\phi/$opt is smaller than 6.

6.2 Monotone Grid Drawings with Large Angles

In this section, we present a simple method for drawing a tree on a grid in a strictly convex, and therefore monotone way. Lemma 6.2 shows that this drawing is automatically crossing-free. We name our strategy the *inorder-algorithm*. We start by ensuring that convex tree drawings are crossing-free. This has already been stated by Carlson and Eppstein [CE07].

Lemma 6.2. *Any convex straight-line drawing of a tree is crossing-free.*

Proof. Let T be a tree and Γ a convex straight-line drawing of T. Assume that two edges $e = (a, b)$ and $e' = (a', b')$ are crossing in Γ in some point q, see Figure 6.2. Let w be the lowest common ancestor of b and b', let π_q be the path $w \to q$ via a, and let π'_q the path $w \to q$ via a'. Let us assume that the children in w are ordered such that π_q starts before π'_q. Let A_q be the region bounded by π_q and π'_q.

We can assume that A_q is of minimum area with respect to other crossings we may have chosen (and, hence, A_q has a connected interior). Now, we consider two paths starting from w.
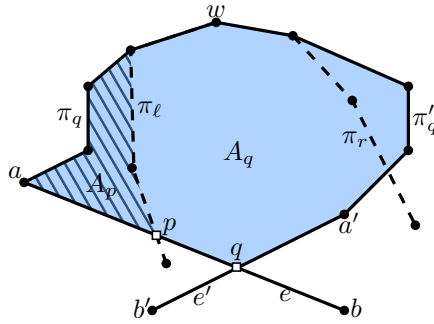
Figure 6.2: An illustration of the situation in the proof of Lemma 6.2

The first one, π_ℓ, starts with the first edge of π_q and then always continues via the last child until it reaches a leaf. The second path, π_r, starts with the first edge of π'_q and continues always using the first child. Note that the polygonal chain π_ℓ together with π_r forms a face f_q of the given convex drawing of the tree. Hence, the face is convex, which means that π_ℓ and π_r only meet in w. Furthermore, we either have $\pi_\ell \neq \pi_q$ or we have $\pi_r \neq \pi'_q$ since otherwise f_q is self-intersecting. As a consequence, at least one of the two paths, say π_ℓ, enters and leaves A_q. Let p be the point where π_ℓ crosses π_q for the first time, and let A_p be the polygon that is bounded by the parts of π_q and π_ℓ between w and p. Then A_p has smaller area than A_q, which contradicts our assumption that A_q has minimum area. $\qquad\square$

Our inorder-algorithm first computes a reasonable large set of primitive vectors, then selects a subset of these vectors, and finally assigns the slopes to the edges. The drawing is then generated by translating the selected primitive vectors. In the following, an *extended* subtree will refer to a subtree including the edge leading into the subtree (if the subtree is not the whole tree).

We will assign a number $s(e)$ to every edge e. This number will refer to the rank of the edge's slope (in circular order) in the final assignment. The rank assignment is done in a recursive fashion. At any time, let $\hat{s}$ be 1 plus the maximum rank $s(e)$ assigned so far. Initially, $\hat{s} = 1$. Let $e = (u, v)$ be an edge (directed away from the root), and let $v_1, v_2, \ldots, v_\ell$ be the children of v ordered from left to right. We recursively set the ranks of all edges in the extended subtree rooted at v_1. Then, we set $s(e) = \hat{s}$ (which increases $\hat{s}$ by one). Finally, for $i = 2, \ldots, \ell$, we set the ranks of the edges in the extended subtree rooted at v_i. For an example of a tree with its edge ranks, see Figure 6.3a.

Second, we assign actual slopes to the edges. Let e be an edge with $s(e) = j$. Then, we assign some vector $s_j \in \mathbb{Z}^2$ to e and draw e as a translate of s_j. We pick the vectors $s_1, s_2, \ldots, s_{n-1}$ by selecting a sufficiently large set of primitive vectors and their reflections in counterclockwise order; see Section 6.1. Our drawing algorithm has the following requirements:
(R1) Edges that are incident to the root and consecutive in circular order are assigned to vectors that together span an angle less than π.
(R2) In every extended subtree hanging off the root, the edges (including the edge incident to the root) are assigned to a set of vectors that spans an angle less than π.
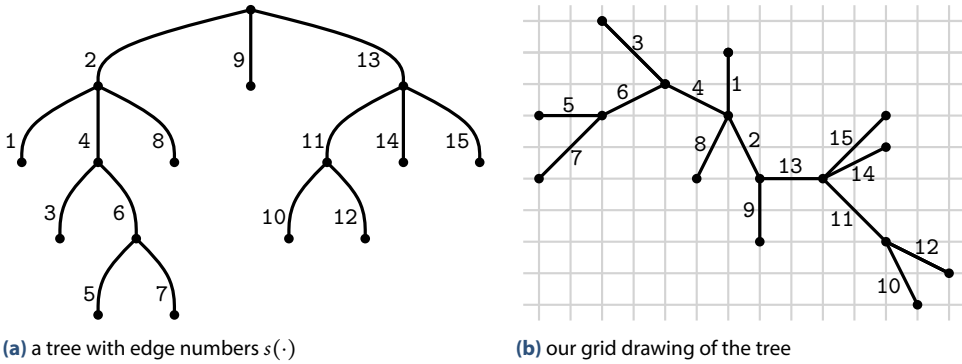These requirements can always be fulfilled, as the following lemma shows.

(a) a tree with edge numbers $s(\cdot)$

(b) our grid drawing of the tree

Figure 6.3: A strictly convex drawing of a tree

Lemma 6.3. *We can select $n-1$ vectors with distinct directed slopes from a $[-d,d] \times [-d,d]$ grid with $d = 4\lceil \sqrt{n} \rceil$ such that the requirements (R1) and (R2) are fulfilled.*

Proof. We first preprocess our tree by adding temporary edges at some leaves. These edges will receive slopes, but are immediately discarded after the assignment.

First, our objective is to ensure that the tree can be split up into three parts that all have n edges. In particular, we adjust the sizes of the extended subtrees hanging off the root by adding temporary edges such that we can partition them into three sets of consecutive extended subtrees which all contain n edges. Note that we have to add $2n + 1$ edges to achieve this.

Second, we define three cones C_1, C_2, and C_3; see Figure 6.4. Each cone has its apex at the origin and spans an angle of $\pi/4$. The angular ranges are $C_1 = [0, \pi/4]$, $C_2 = [3\pi/4, \pi]$, and $C_3 = [3\pi/2, 7\pi/4]$; angles are measured from the x-axis pointing in positive direction. Note that C_2 is separated from the two other cones by an angle of $\pi/2$. As mentioned in Section 6.1, the set V_n contains n primitive vectors in the $[0, d] \times [0, d]$ grid. When reflected on the $x = y$ line, these vectors lie in C_1. Reflecting the vectors in C_1, we further generate n vectors in C_2 and n vectors in C_3. In every cone, we "need" at most $n - 3$ edges. Hence, we can remove the vectors on the boundary of each cone. After removing the temporary edges, the number of vectors will drop from $3n$ to $n - 1$.

Now, we observe the following. Every two consecutive edges incident to the root lie in the *interiors* of our cones. Given the sizes and angular distances of the cones, this yields requirement (R1). Furthermore, any extended subtree is assigned slopes from a single cone. This yields (R2). □

For the example tree of Figure 6.3a, it suffices to pick the 16 vectors that one gets from reflecting the primitive vectors from the $[0, 2] \times [0, 2]$ grid. These vectors already fulfill requirements (R1) and (R2). Hence, we do not have to apply the more involved slope selection as described in Lemma 6.3. The resulting drawing is shown in Figure 6.3b.

Every face in the drawing contains two leaves. The leaves are ordered by their appearance in some DFS-sequence $\mathcal{D}$ respecting some rooted combinatorial embedding of T. For a face f, we call the leaf that comes first in $\mathcal{D}$ the *left leaf* and the other leaf of f the *right leaf* of f. The
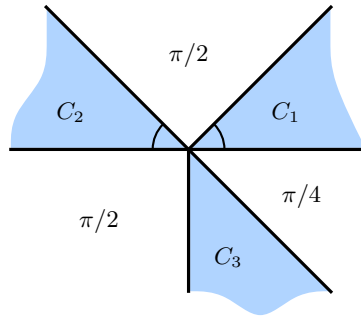
Figure 6.4: The cones that contain the slopes used in the algorithm

only exception is the face whose leaves are the first and last child of $\mathcal{D}$. Here, we call the first vertex in $\mathcal{D}$ the right leaf and the last vertex in $\mathcal{D}$ the left leaf.

Lemma 6.4. *Let u be the left leaf, and let v be the right leaf of a face of T. Further, let w be the lowest common ancestor of u and v. The above assignment of slope ranks s to the tree edges implies the following.*

- *(a) If edge e_1 is on the path $w \to u$ and edge e_2 is on the path $w \to v$, then $s(e_1) < s(e_2)$.*

- *(b) The ordered sequence of edges on the path $w \to u$ is increasing in $s(\cdot)$.*

- *(c) The ordered sequence of edges on the path $w \to v$ is decreasing in $s(\cdot)$.*

Proof. Let a be an edge that links the parent p to its child u, let b be the edge that links u to its leftmost child, and let c be the edge that links u to its rightmost child; see Figure 6.5b. In the assignment, we first picked the slope in the subtree rooted at the leftmost children of u, then we selected the slope for a, and later we picked the slopes for the subtree rooted at the rightmost children of u. Since we select the slopes in their radial order, we have $s(b) < s(a) < s(c)$.

Now, note that the slopes on the path $w \to u$ have been assigned before the slopes on the path $w \to v$, which proves (a). When traversing the path $w \to u$, we follow the rightmost children, except maybe for w's child; see Figure 6.5a. Hence, the sequence of slopes is increasing, and (b) follows. Statement (c) follows by a similar argument: We traverse the path $w \to v$ by taking the leftmost child, except maybe for w's child. Hence, the sequence of slopes is decreasing. $\qquad\square$

We now prove the correctness of our algorithm.

Theorem 6.1. *Given an embedded tree with n vertices (none of degree 2), the inorder-algorithm produces a strictly convex and crossing-free drawing with angular resolution $\Omega(1/n)$ on a grid of size $O(n^{1.5}) \times O(n^{1.5})$. The algorithm runs in $O(n)$ time.*

Proof. We first show that no face in the drawing is incident to an angle larger than π. Let f be a face, let e and e' be two consecutive edges on the boundary of f, and let α be the angle formed by e and e' in the interior of f. If e and e' are incident to the root, requirement (R1) implies $\alpha < \pi$. If both edges contain the lowest common ancestor of the leafs belonging to f,
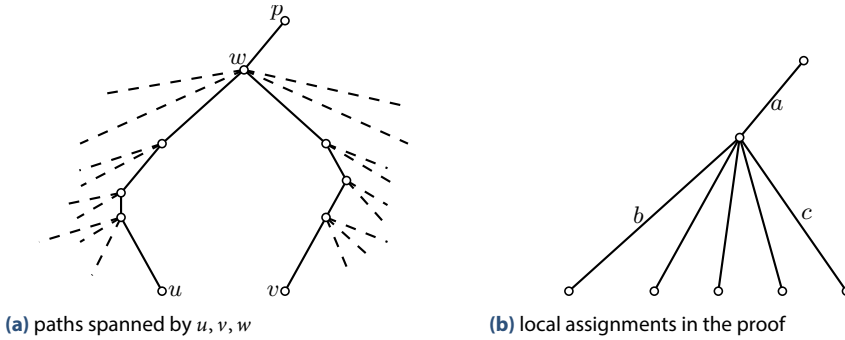
(a) paths spanned by u, v, w

(b) local assignments in the proof

Figure 6.5: Situation analyzed in Lemma 6.4

then, by requirement (R2), also $\alpha < \pi$. In the remaining case, e and e' both lie on a path to the left leaf of f, or both lie on a path to the right leaf of f. At vertex v, we have at least two outgoing edges. Let e_1 be the first outgoing edge and e_2 be the last outgoing edge at v—one of the edges is e'. By the selection of the slope ranks, we have $s(e_1) < s(e) < s(e_2)$. Consequently, the supporting line of e separates e_1 and e_2, and hence both faces containing e have an angle less than π at v. Therefore, it holds that $\alpha < \pi$.

Next, we show that the edges and rays of a face do not intersect. Then, by Lemma 6.2, no edges will cross. Assume that there are two edges/rays ℓ and r in a common face that intersect in some point x. Let t be the lowest common ancestor of ℓ and r, and assume that ℓ lies on the path to the left leaf and r on the path to the right leaf. We define a closed polygonal chain P as follows. The chain starts with the path $t \to \ell$, continues via x to r, and finally returns to t. We direct the edges according to this walk (for measuring the directed slopes) and call them $e_1, e_2, \ldots, e_k$. We may assume that P is simple; otherwise, we find another intersection point. By Lemma 6.4, the slopes are monotone when we traverse P. For $i = 1, \ldots, k-1$, let α_i be the difference between the directed slopes of the edges e_i and e_{i+1}. Then, the sum $\sum_{i<k} \alpha_i$ equals the angle between the slopes of e_1 and e_k. Due to requirement (R2), this angle is less than π. Let $\beta_i = \pi - \alpha_i$ be the angle between e_i and e_{i+1} in P, and let $\beta_0 > 0$ be the "interior" angle at t. We have that

$$\sum_{0 \le i < k} \beta_i = \beta_0 + \sum_{1 \le i < k} (\pi - \alpha_i) > 0 + (k-1)\pi - \pi = (k-2)\pi.$$

This, however, contradicts the fact that the angle sum of the polygon with boundary P is $(k-2)\pi$. Thus, our assumption that two edges/rays cross was wrong.

Since the drawing is assembled from $n - 1$ vectors whose absolute coordinates are at most $O(\sqrt{n})$, the complete drawing uses a grid of dimension $O(n^{1.5}) \times O(n^{1.5})$. Since all vectors are reflections of (a subset of) vectors defined by a Farey sequence with at most n entries, Lemma 6.1 yields that the angular resolution is bounded by $\Omega(1/n)$. □

We conclude this section with comparing our result with the drawing algorithm of Carlson and Eppstein [CE07]. Their algorithm produces a drawing with optimal angular resolution. It draws trees convex, but, in contrast to our algorithm, not necessarily strictly convex. Allowing

parallel leaf edges can have a great impact on the angular resolution. However, our ideas can be applied to modify the algorithm of Carlson and Eppstein. For the leaf edges, their algorithm uses a set of k slopes and picks the slopes such that they are separated by an angle of $2\pi/k$. The slopes of interior edges have either one of the slopes of the leaf edges, or are chosen such that they bisect the wedge spanned by their outermost child edges. However, it suffices to assure that the slope of an interior edge differs from the extreme slopes in the following subtree by at least $2\pi/(2k)$.

We can now modify the algorithm as follows. We pick $2k/8$ primitive vectors and reflect them such that they fill the whole angular space with $2k$ distinct integral vectors. We use every other vector of this set for the leaf edges. For an interior edge, we take any vector from our preselected set whose slope lies in between the extreme slopes of the edges in its subtree. Since we have sufficiently spaced out our set of primitive vectors, we can always find such a vector. Thus, we obtain a drawing on the $O(n^{1.5}) \times O(n^{1.5})$ grid. Clearly, the drawing does not have optimal angular resolution. However, since we use $2k$ integral vectors having, by Lemma 6.1, an angular resolution of $\Omega(1/k)$, we differ from the best possible angular resolution $2\pi/k$ only by a constant factor.

See Figure 6.6 for a comparison between our approach with that of Carlson and Eppstein [CE07] and that of Angelini et al. [ACB+12].

6.3 Strongly Monotone Drawings

In this section, we show how to draw trees in a strongly monotone fashion. We first show how to draw any proper binary tree, that is, any internal vertex has exactly two children. We name our strategy the *disk-algorithm*. Then, we generalize our result to arbitrary trees. Further, we show that connected planar graphs do not necessarily have a strongly monotone drawing. Finally, we show how to draw biconnected outerplanar graphs in a strongly monotone fashion. Note that the drawings computed by our algorithms require exponential area. However, Nöllenburg et al. [NPR14] have recently shown that exponential area is required for strongly monotone drawings of trees, which justifies this area bound.

Let T be a proper binary tree, let D be any disk with center c, and let C be the boundary of D. Recall that a strictly convex drawing cannot have a vertex of degree 2. Thus, we consider the root of T a dummy vertex and ensure that the angle at the root is π. We draw T inside D. We start by mapping the root of T to c. Then, we draw a horizontal line h through c and place the children of the root on $h \cap \mathrm{int}(D)$ such that they lie on opposite sites relative to c. We cut off two circular segments by dissecting D with two vertical lines running through points representing the children of the root. We inductively draw the right subtree of T into the right circular segment and the left subtree into the left circular segment.

In any step of the inductive process, we are given a vertex v of T, its position in D (which we also denote by v) and a circular segment D_v; see Figure 6.7a. The preconditions for our construction are that

(i) v lies in the relative interior of the chord s_v that delimits D_v, and

(ii) D_v is empty, that is, the interiors of D_v and D_u are disjoint for any vertex u that does not lie on a root–leaf path through v.

(a) tree with two dummy vertices

(b) drawing by Angelini et al. [ACB⁺12]

(c) drawing by Carlson and Eppstein [CE07]

(d) drawing by our algorithm
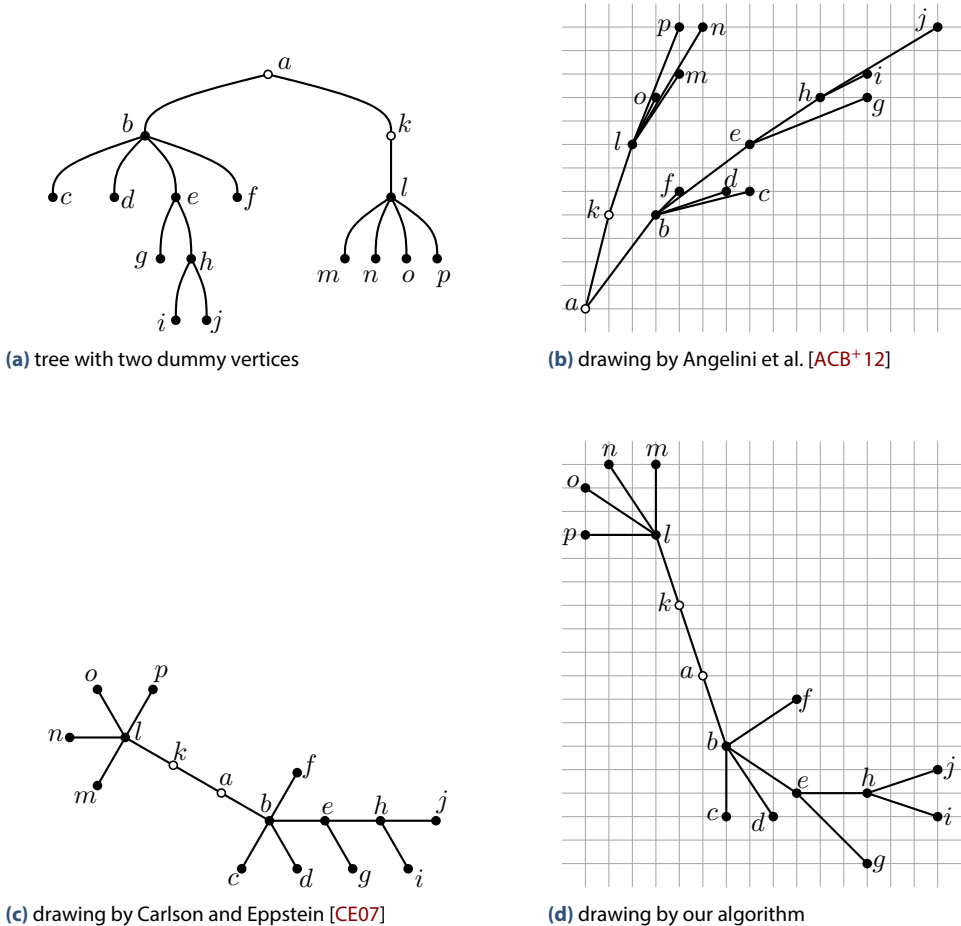
Figure 6.6: Example tree of Angelini et al. [ACB⁺12], drawn by various algorithms. We treat the degree-2 vertices as dummy vertices because of the degree restriction.

In order to place the two children l and r of v (if any), we shoot a ray $\vec{v}$ from v perpendicular to s_v into D_v. Let v' be the point where $\vec{v}$ hits C. Consider the chords that connect the endpoints of s_v to v'. The chords and s_v form a triangle with height vv'. The height is contained in the interior of the triangle and splits it into two right subtriangles. The chords are the hypotenuses of the subtriangles. We construct l and r by connecting v to these chords perpendicularly. Note that, since the subtriangles are right triangles, the heights lie inside the subtriangles. Hence, l and r lie in the relative interiors of the chords. Further, note that the circular segments D_l and D_r delimited by the two chords are disjoint and both are contained in D_v. Hence, D_l and D_r are empty, and the preconditions for applying the above inductive process to r and l with D_l and D_r are fulfilled. See Figure 6.7b for the output of our algorithm for a tree of height 3.
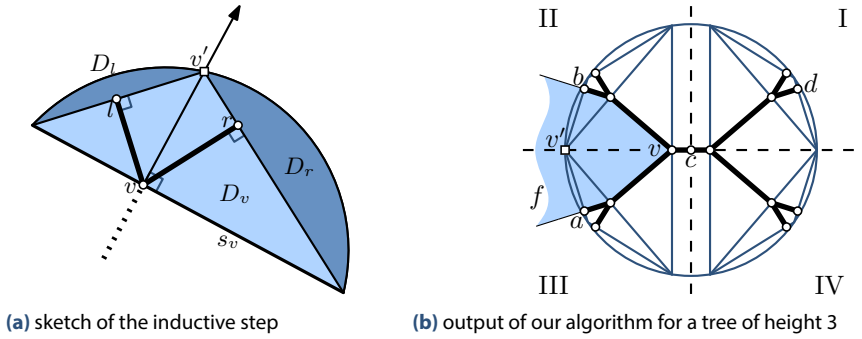
(a) sketch of the inductive step   **(b)** output of our algorithm for a tree of height 3

Figure 6.7: Strongly monotone drawings of proper binary trees

Lemma 6.5. *For a proper binary tree rooted in a dummy vertex, the disk-algorithm yields a strictly convex drawing.*

Proof. Let T be a proper binary tree and let f be a face of the drawing generated by the algorithm described above. Clearly, f is unbounded. Let a and b be the leaves of T that are incident to the two unbounded edges of f, and let v be the lowest common ancestor of a and b; see Figure 6.7b. Consider the two paths $v \to a$ and $v \to b$. We assume that the path from v through its left child ends in a and the path through its right child ends in b.

Due to our inductive construction that uses disjoint disk sections for different subtrees, the two paths clearly do not intersect. Moreover, each vertex on the two paths is convex, that is, the angle that such a vertex forms inside f is less than π. This is due to the fact that we always turn right when we go from v to a, and we always turn left when we go to b. Vertex v is also convex since the two edges from v to its children lie in the same half-plane (bounded by s_v).

It remains to show that the two rays $\vec{a}$ and $\vec{b}$ (defined analogously to $\vec{v}$ above) don't intersect. To this end, recall that $v' = \vec{v} \cap C$. By our construction, $\vec{a}$ and $\vec{b}$ are orthogonal to two chords of C that are both incident to v'. Clearly, the two chords form an angle of less than π in v'. Hence, the two rays diverge, and the face f is strictly convex. $\qquad\square$

For the proof that the algorithm described above yields a strongly monotone drawing, we need the following tools. Let $\vec{v}_1$ and $\vec{v}_2$ be two vectors. We say that $\vec{v}_3$ *lies between* $\vec{v}_1$ and $\vec{v}_2$ if $\vec{v}_3$ is a positive linear combination of $\vec{v}_1$ and $\vec{v}_2$. For two vectors $\vec{v}$ and $\vec{w}$, we define $\langle \vec{v}, \vec{w} \rangle = |\vec{v}||\vec{w}| \cos(\vec{v}, \vec{w})$ as the scalar product of $\vec{v}$ and $\vec{w}$.

Lemma 6.6. *If a path p is monotone with respect to two vectors $\vec{v}_1$ and $\vec{v}_2$, then it is monotone with respect to any vector $\vec{v}_3$ between $\vec{v}_1$ and $\vec{v}_2$.*

Proof. Let $\vec{v}_3 = \lambda_1 \vec{v}_1 + \lambda_2 \vec{v}_2$ with $\lambda_1, \lambda_2 > 0$. Assume that the path p is given by the sequence of vectors $\vec{w}_1, \vec{w}_2, \ldots, \vec{w}_k$. Since p is monotone with respect to vectors $\vec{v}_1$ and $\vec{v}_2$, we have that $\langle \vec{v}_1, \vec{w}_i \rangle > 0$ and $\langle \vec{v}_2, \vec{w}_i \rangle > 0$ for all $i \le k$. This yields, for all $i \le k$,

$$\langle \vec{v}_3, \vec{w}_i \rangle = \langle \lambda_1 \vec{v}_1 + \lambda_2 \vec{v}_2, \vec{w}_i \rangle = \lambda_1 \langle \vec{v}_1, \vec{w}_i \rangle + \lambda_2 \langle \vec{v}_2, \vec{w}_i \rangle > 0,$$

since $\lambda_1, \lambda_2 > 0$. It follows that p is monotone with respect to $\vec{v}_3$. $\qquad\square$
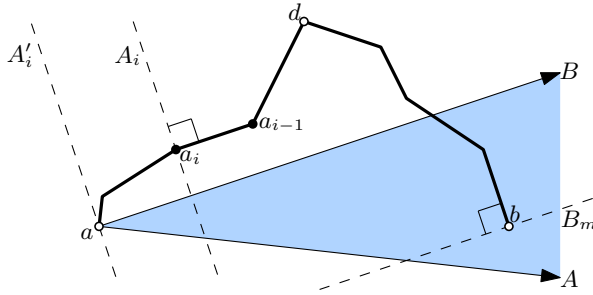
Figure 6.8: Illustration of case 3 in the proof of Lemma 6.7

Lemma 6.7. *For a proper binary tree rooted in a dummy vertex, the disk-algorithm yields a strongly monotone drawing.*

Proof. We split the drawing into four sectors: I, II, III and IV; see Figure 6.7b. Let a and b be two vertices in the graph. We will show that the path $a \to b$ in the output drawing of our algorithm is strongly monotone. Let c be the root of the tree. Without loss of generality, assume that a lies in sector III. Then, we distinguish three cases.

Case 1: a and b lie on a common root–leaf path; see a and v in Figure 6.7b. Obviously, b lies in sector III. Without loss of generality, assume that b lies on the path $a \to c$. By construction, all edges in sector III, seen as vectors directed towards c, lie between $\vec{x} = (0,1)$ and $\vec{y} = (1,0)$. Thus, all edges on the path $a \to b$, and in particular $\vec{ab}$, lie between $\vec{x}$ and $\vec{y}$. Since $\vec{x}$ is perpendicular to $\vec{y}$, the path $a \to b$ is monotone with respect to $\vec{x}$ and $\vec{y}$. Following Lemma 6.6, the path $a \to b$ is monotone with respect to $\vec{ab}$, and thus strongly monotone.

Case 2: b lies in sector I; see a and d in Figure 6.7b. In Case 1, we have shown that the all edges on the path $a \to c$ lie between $\vec{x} = (0,1)$ and $\vec{y} = (1,0)$. Analogously, the same holds for the path $c \to b$. Thus, the path $a \to b$ is monotone with respect to $\vec{x}$ and $\vec{y}$ and, following Lemma 6.6, strongly monotone.

Case 3: a and b do not lie on a common root–leaf path, and b does not lie in sector I; see a and b in Figure 6.7b. Let d be the lowest common ancestor of a and b. Let $\langle a = a_0, a_1, \ldots, a_k = d \rangle$ be the path $d \to a$ to a. Further, let $\langle d = b_0, b_1, \ldots, b_m = b \rangle$ be the path $d \to b$. Finally, let $p = \langle a_k, a_{k-1}, \ldots, a_0, b_1, \ldots, b_{m-1}, b_m \rangle$ be the path $a \to b$.

Below, we describe how to rotate and mirror the drawing such that any vector $\overrightarrow{a_i, a_{i-1}}$, $1 \le i \le k$ lies between $\vec{x} = (0,1)$ and $\vec{y} = (1,0)$, and any vector $\overrightarrow{b_{j-1}, b_j}$, $1 \le j \le m$ lies between $\vec{x}$ and $-\vec{y}$. This statement is equivalent to $x(a_i) < x(d) < x(b_j), 1 \le i \le k, 1 \le j \le m$ and $y(a_k) < \ldots < y(a_1) < y(d) > y(b_1) > \ldots > y(b_m)$; see Figure 6.8. If b lies in sector IV, then $d = c$ and this statement is true by construction. If b lies in sector II, then d is a child of c. We rotate the drawing by $\pi/2$ in counterclockwise direction and then mirror it horizontally. If b lies in sector III, let $p(d)$ be the parent of d. We rotate the drawing such that the edge $(p(d), d)$ is drawn vertically. Recall that, by construction, the ray from d in direction $\overrightarrow{p(d)d} = -\vec{y}$ separates the subtrees of the two children of d; see Figure 6.7a. Further, the angle between any edge (directed away from d) in the subtree of d and $\overrightarrow{p(d)d} = -\vec{y}$ is at most $\pi/2$, that is, they are directed downwards.

Let $A_i, 1 \le i \le k$ be the straight line through a_i perpendicular to $\overrightarrow{a_{i-1}a_i}$. Let A'_i be the parallel line to A_i that passes through a. Due to the x-monotonicity of p, the point a lies below A_i. During the construction of the tree, the line A_i defined a circular sector in which the subtree rooted at a_i including a was exclusively drawn. It follows that a and b lie on opposite sites of A_i. Thus, b lies above A_i and also above A'_i. Let $B_j, 1 \le j \le m$ be the straight line through b_j perpendicular to $\overrightarrow{b_{j-1}b_j}$. Let B'_j be the parallel line to B_j that passes through a. By construction, b lies below B_j and a lies above B_j. Thus, b lies below B'_j.

Let A be the line A'_i with maximum slope and let B be the line B'_j with minimum slope. First, we will show that the path is monotone with respect to the unit vector $\vec{A}$ on A directed to the right. By choice of A, the angle between any edge $(a_i, a_{i-1}), 1 \le i \le k$ and $\vec{A}$ is at most $\pi/2$. Recall that any vector $\overrightarrow{a_i, a_{i-1}}, 1 \le i \le k$, lies between $\vec{x}$ and $\vec{y}$. Since $\vec{A}$ is perpendicular to one of these edges and directed to the right, it lies between $\vec{x}$ and $-\vec{y}$. Since any vector $\overrightarrow{b_{j-1}, b_j}$, $1 \le j \le m$, also lies between $\vec{x}$ and $-\vec{y}$, the angle between $\vec{A}$ and these edges is at most $\pi/2$. Because the angle between $\vec{A}$ and any edge on the path $a \to b$ is at most $\pi/2$, the path is monotone with respect to $\vec{A}$.

Analogously, it can be shown that the path is monotone with respect to $\vec{B}$. Recall that b lies above A and below B. Hence, the vector $\overrightarrow{ab}$ lies between $\vec{A}$ and $\vec{B}$. Following Lemma 6.6, the path is monotone with respect to $\overrightarrow{ab}$ and, thus, strongly monotone. □

Lemmas 6.5 and 6.7 immediately imply the following.

Theorem 6.2. *Any proper binary tree rooted in a dummy vertex has a strongly monotone and strictly convex drawing.*

Next, we (partially) extend this result to arbitrary trees.

Theorem 6.3. *Any tree has a strongly monotone drawing.*

Proof. Let T be a tree. If T has a vertex of degree 2, we root T in this vertex. Otherwise, we subdivide any edge by creating a vertex of degree 2, which we pick as root. Then, we add a leaf to every vertex of degree 2, except the root. Now, let v be a vertex with out-degree $k > 2$. Let $(v, w_1), \ldots, (v, w_k)$ be the outgoing edges of v ordered from right to left. We substitute v by a path $\langle v = v_1, \ldots, v_{k+1} \rangle$, where v_{i+1} is the left child of v_i, for $i = 1, \ldots, k$. Then, we substitute the edges (v, w_i) by $(v_i, w_i), i = 2, \ldots, k$; see Figure 6.9.

Let T' be the resulting proper binary tree. Clearly, all vertices of T', except its root, have degree 1 or 3, so T' is a proper binary tree. We use Theorem 6.2 to get a strongly monotone drawing $\Gamma_{T'}$ of T'. Then, we remove the dummy vertices inserted above and draw the edges that have been subdivided by a path as a straight-line. This yields a drawing Γ_T of T that is crossing-free since the only new edges form a set of stars that are drawn in disjoint areas of the drawing.

Now, we show that Γ_T is strongly monotone. Let (v, w) be an edge in T. Let $p = \langle v = v_1, \ldots, v_m = w \rangle$ be the path $v \to w$ in T'. Suppose p is monotone with respect to some direction $\vec{d}$. Thus, $\angle\{\overrightarrow{v_i v_{i+1}}, \vec{d}\} < \pi/2$ for $1 \le i \le m-1$. Clearly, $\overrightarrow{vw} = \sum_{i=1}^{m-1} \overrightarrow{v_i v_{i+1}}$ is a positive linear combination of $\overrightarrow{v_i v_{i+1}}, i = 1, \ldots, m$ and, hence, $\angle\{\overrightarrow{vw}, \vec{d}\} < \pi/2$. It follows that the
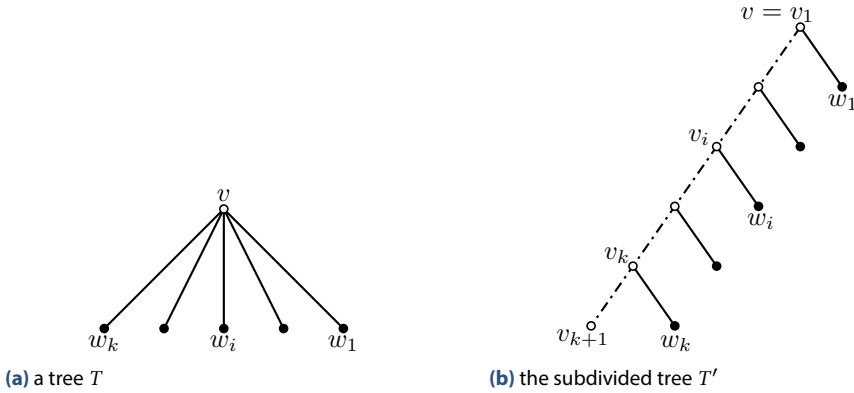
(a) a tree T **(b)** the subdivided tree T'

Figure 6.9: Subdivision of a vertex v with k outgoing edges

path $a \to b$ for some vertices a, b in T is monotone with respect to a direction $\vec{d}$ in Γ_T if the path $a \to b$ is monotone to $\vec{d}$ in $\Gamma_{T'}$. With $\vec{d} = \overrightarrow{ab}$, it follows that Γ_T is strongly monotone. □

We add to this another positive result concerning biconnected outerplanar graphs.

Theorem 6.4. *Any biconnected outerplanar graph has a strongly monotone and strictly convex drawing.*

Proof. Let G be a biconnected outerplanar graph with outer cycle $\langle v_1, \dots, v_n, v_1 \rangle$. We place the vertices $v_2, \dots, v_{n-1}$ in counterclockwise order on a quarter circle C that has $v_1 = (0,0)$ and $v_n = (1,1)$ as its endpoints; see Figure 6.10. Since the outer cycle is drawn strictly convex, the drawing is planar and strictly convex. Clearly, the path $\langle v_1, \dots, v_n \rangle$ is x- and y-monotone. Also, every vector $\overrightarrow{v_i v_j}$, $j > i$ lies between $\vec{x} = (0,1)$ and $\vec{y} = (1,0)$. Thus, by Lemma 6.6, the drawing is strongly monotone. □

We close with a negative result. Note that the graphs in the family that we construct are neither outerplanar nor biconnected.

Theorem 6.5. *There is an infinite family of connected planar graphs that do not have a strongly monotone drawing in any combinatorial embedding.*

Proof. Let C be the graph that arises by attaching to each vertex of K_4 a "leaf"; see Figure 6.11.
  Let $v_1, \dots, v_4$ be the vertices of K_4. For K_4 to be crossing-free, one of its vertices, say v_1, lies in the interior. Let w be the leaf incident to v_1. Because of planarity, w has to be placed inside a triangular face incident to v_1. Without loss of generality, assume that w is placed in the face (v_1, v_2, v_3). If the drawing is strongly monotone, then $\angle (\overrightarrow{wv_2}, \overrightarrow{wv_1}) < \pi/2$ and $\angle (\overrightarrow{wv_1}, \overrightarrow{wv_3}) < \pi/2$, and thus $\angle (\overrightarrow{wv_3}, \overrightarrow{wv_2}) > \pi$. However, this means that w does not lie inside the triangle (v_1, v_2, v_3), which is a contradiction to the assumption. Thus, C does not have a strongly monotone drawing in any combinatorial embedding. We create an infinite family from C by adding more leaves to the vertices of K_4. □
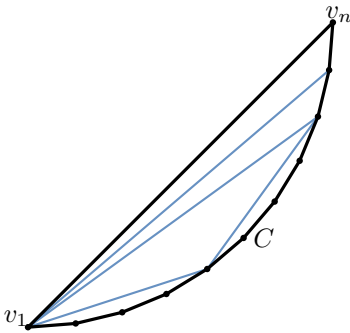
Figure 6.10: A strongly monotone drawing of a biconnected outerplanar graph
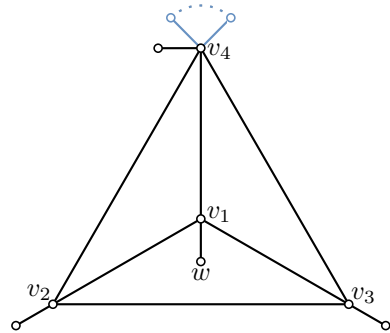
Figure 6.11: A planar graph without any strongly monotone drawing

6.4 Concluding Remarks

We have shown that any tree has a monotone drawings on a grid with area $O(n^3)$ and a strongly monotone drawing, but several problems remain open. It is a common question whether we can combine the features in our two algorithms, that is, whether any tree has a strongly monotone drawing on a grid of polynomial size. Angelini et al. [ACB+12, Figure 18(b)] have constructed a drawing of a triangulation that is not strongly monotone. It is still open whether there is also a triconnected (or biconnected) planar graph that does not have any strongly monotone drawing. If yes, it is interesting whether this can be tested efficiently. If we could show that our drawings for general trees are not just strongly monotone but also convex (as in the proper binary case), then all Halin graphs would automatically have convex and strictly monotone drawings, too.

Crossings with Large Angles

7 Simultaneous Drawing of Planar Graphs with Right-Angle Crossings

A simultaneous embedding of two planar graphs embeds each graph in a planar way—using the same vertex positions for both embeddings. Edges of one graph are allowed to intersect edges of the other graph. There are two versions of the problem: In the first version, called *Simultaneous Embedding with Fixed Edges* (Sefe), edges that occur in both graphs must be embedded in the same way in both graphs (and hence, cannot be crossed by any other edge). In the second version, simply called *Simultaneous Embedding*, these edges can be drawn differently for each of the graphs. Both versions of the problem have a geometric variant where edges must be drawn using straight-line segments.

Simultaneous embedding problems have been investigated extensively over the last few years, starting with the work of Brass et al. [BCD+07] on simultaneous straight-line drawing problems. Bläsius et al. [BKR13b] recently surveyed the area. For example, it is possible to decide in linear time whether a pair of graphs admits a Sefe or not, if the common graph is biconnected [ABF+12] or if it has a fixed planar embedding [ADF+15]. Furthermore, Sefe can be decided in polynomial time if each connected component of the common graph is biconnected or subcubic [Sch13], or if it is outerplanar with cutvertices of degree at most 3 [BKR13a].

When actually drawing these simultaneous embeddings, a natural choice is to use straight-line segments. It is NP-hard to decide whether two planar graphs admit a geometric simultaneous embedding [EBGJ+08]. This negative results holds even if one of the input graphs is a matching [CvL+11]. In fact, only very few graphs can be drawn in this way and some existing results require exponential area. For instance, there exist a tree and a path which cannot be drawn simultaneously with straight-line segments [AGKN12], and the algorithm for simultaneously drawing a tree and a matching [CvL+11] does not provide a polynomial area bound.

A way to overcome the restrictions of straight-line drawings is to allow edges to bend. The resulting drawings with polygonal edges are called *polyline drawings* for short. Such drawings have been investigated by Haeupler et al. [HJL13]. They showed that if the common graph is biconnected, then a drawing can be found in which one input graph has no bends at all and in the other input graph the number of bends per edge is bounded by the number of common vertices. Erten and Kobourov [EK05b] showed that three bends per edge and quadratic area suffice for any pair of planar graphs (without fixed edges), and that one bend per edge suffices for pairs of trees. Kammer [Kam06] reduced the required number of bends per edge to two for the general case of planar graphs. Grilli et al. [GHKR14] proved that every Sefe embedding of two graphs admits a drawing with no bends in the common edges and at most nine bends per exclusive edge. If the common graph is biconnected, they can reduce the number of bends to three per exclusive edge. In these results, however, the *crossing angles* can be very small.

In this chapter, we suggest a new approach that overcomes the aforementioned problems. We insist that crossings occur at right angles, thereby "taming" them. We do this while drawing all vertices and all bends on an integer grid of size $O(n) \times O(n)$ for any pair of planar n-vertex graphs. In a way, our results give a measure for the geometric complexity of simultaneous

(a) two graphs on the same vertex set   (b) a RacSim drawing   (c) a RacSefe drawing
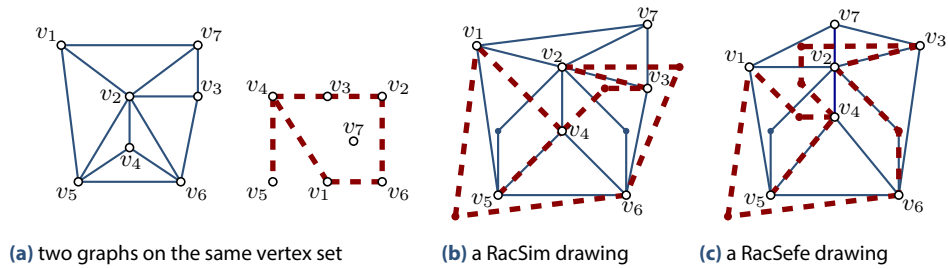
Figure 7.1: (a) Two planar graphs on the same vertex set. (b) A RacSim drawing of the two graphs with at most one bend per edge. The edge (v_2, v_6) is drawn differently in the two graphs, while the edges (v_3, v_4) and (v_2, v_3) are both represented by the same polyline in the two graphs. (c) A RacSefe drawing of the two graphs with up to two bends per edge.

embeddability for various combinations of graph classes, some of which can be combined more easily (that is, with fewer bends) and some not as easily (that is, with more bends).

Let $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ be two planar graphs defined on the same vertex set and let $n = |V|$. We say that G_1 and G_2 admit a *RAC simultaneous drawing* (or, RacSim *drawing* for short) if we can place the vertices on the plane such that:

(i) each edge is drawn as a polyline,

(ii) both graphs are drawn planar,

(iii) non-common edges are either disjoint or cross each other (one or several times) at right angles, and

(iv) common edges may be represented by the same polyline.

Note that non-common edges may not overlap. G_1 and G_2 admit a *RAC simultaneous drawing with fixed edges* (or, RacSefe *drawing* for short) if they admit a RacSim drawing with the adjusted condition (iv):

(iv') common edges *must* be represented by the same polyline.

In Figure 7.1, we give an example of two planar graphs on the same vertex set that admit a RacSim drawing with at most one bend per edge and a RacSefe drawing with at most two bends per edge.

Argyriou et al. [ABKS13] introduced and studied the geometric version of the RacSim drawing problem. In particular, they proved that any pair of a cycle and a matching admits a geometric RacSim drawing on an integer grid of quadratic size, while there exist a wheel and a cycle that do not admit a geometric RacSim drawing. The problem that we study in this chapter was left as an open problem.

Closely related to the RacSim drawing problem is the problem of simultaneously drawing a (primal) embedded graph and its dual, so that the primal-dual edge crossings form right angles. Brightwell and Scheinermann [BS93] proved that this is always possible if the input graph is triconnected planar. Erten and Kobourov [EK05a] presented an $O(n)$-time algorithm

Graph classes		Number of bends	Ref.	Model
planar	+ planar	6 + 6	Theorem 7.1	RacSim
subhamiltonian	+ subhamiltonian	4 + 4	Corollary 7.1	RacSim
outerplanar	+ outerplanar	3 + 3	Theorem 7.2	RacSim
cycle	+ cycle	1 + 1	Theorem 7.3	RacSefe
caterpillar	+ cycle	1 + 1	Theorem 7.4	RacSefe
four matchings		1 + 1 + 1 + 1	Theorem 7.5	RacSim
tree	+ matching	1 + 0	Theorem 7.6	RacSefe
wheel	+ matching	2 + 0	Theorem 7.7	RacSefe
outerpath	+ matching	2 + 1	Theorem 7.8	RacSefe

Table 7.1: A short summary of our results

that computes a simultaneous drawing of a triconnected planar graph and its dual on an integer grid of size $O(n) \times O(n)$, where n is the total number of vertices in the graph and its dual; their drawings, however, can have non-right angle crossings.

Our contribution.   Our main result is that any pair of planar graphs admits a RacSim drawing with at most six bends per edge. For pairs of subhamiltonian graphs and pairs of outerplanar graphs, we can reduce the required number of bends per edge to four and three, respectively; see Section 7.1. (Recall that a subhamiltonian graph is a subgraph of a planar Hamiltonian graph.) Then, we turn our attention to pairs of more restricted graph classes where we can guarantee RacSim and RacSefe drawings with one bend per edge or two bends per edge; see Sections 7.2 and 7.3, respectively. Table 7.1 summarizes our results. Note that all our algorithms run in linear time, with the exception of the algorithm for an outerpath and a matching (see Theorem 7.8), which runs in $O(n \log n)$ time. The produced drawings fit on integer grids of quadratic size. The main approach of all our algorithms is to find linear orders on the vertices of the two graphs and then to compute the exact coordinates of the vertices of both graphs based on these orders. These drawings may contain non-rectilinear segments (referred to as *slanted* segments, for short), but all crossings in our drawings appear between horizontal and vertical edge segments and are therefore at right angle.

7.1 RacSim Drawings of General Graphs

In this section, we study general planar graphs and show how to efficiently construct RacSim drawings in quadratic area, with few bends per edge. We prove that two planar graphs on a common set of vertices admit a RacSim drawing with six bends per edge (Theorem 7.1). We lower the required number of bends per edge to 4 for pairs of subhamiltonian graphs (Corollary 7.1), and to 3 for pairs of outerplanar graphs (Theorem 7.2). Note that the class of subhamiltonian graphs is equal to the class of 2-page book-embeddable graphs, and the class of outerplanar graphs is equal to the class of 1-page book-embeddable graphs [BK79].
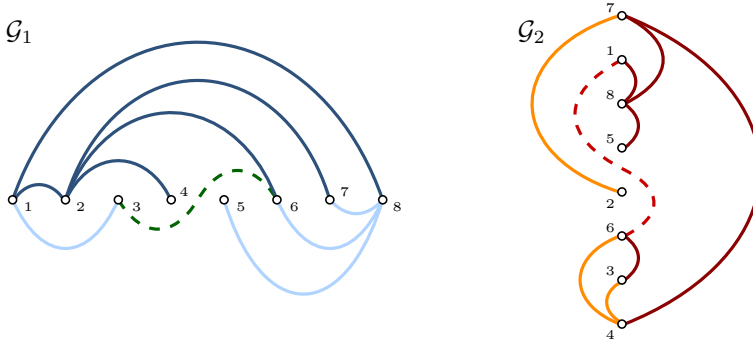
**Figure 7.2:** Drawings of two planar graphs by Kaufmann and Wiese [KW02]

Central to our approach is an algorithm by Kaufmann and Wiese [KW02] that embeds any planar graph such that vertices are mapped to points on a horizontal line (the so-called *spine*) and each edge crosses the spine at most once; see Figure 7.2. If one replaces each spine crossing with a dummy vertex, then a *linear order* of the vertices (both original and dummy) is obtained with the property that every edge is either completely above or completely below the spine. In order to determine the exact locations of the vertices of the two given graphs in our problem, we use the linear order induced by the first graph to compute the x-coordinates and the linear order induced by the second graph to compute the corresponding y-coordinates. (Note that this approach has been used for simultaneous drawing problems before [EK05b].) Then, we draw the edges of both graphs, so that all edges-crossings

   (i) are restricted between vertical and horizontal edge-segments, that is, slanted edge-segments are crossing-free, and

   (ii) appear in the interior of the smallest axis-aligned rectangle containing all vertices.

Theorem 7.1. *Two planar graphs on a common set of n vertices admit a RacSim drawing on an integer grid of size $(14n - 26) \times (14n - 26)$ with six bends per edge. The drawing can be computed in $O(n)$ time.*

Proof. Let $\mathcal{G}_1 = (V, E_1)$ and $\mathcal{G}_2 = (V, E_2)$ be planar graphs. For $m \in \{1, 2\}$, let ξ_m be an embedding of $\mathcal{G}_m$ according to the algorithm of Kaufmann and Wiese [KW02]; see Figure 7.2. As explained before, we subdivide all edges of $\mathcal{G}_m$ that cross the spine in ξ_m by introducing a dummy vertex at the point where it crosses the spine. Let $\mathcal{G}'_m = (V'_m, E'_m) = (V \cup V_m, A_m \cup B_m)$ be the resulting graph, where V_m is the set of dummy vertices in $\mathcal{G}'_m$, A_m is the set of edges that are drawn completely above the spine and B_m is the set of edges that are drawn completely below the spine. Let ξ'_m be the embedding of $\mathcal{G}'_m$.

   Now we show how to determine the x-coordinates of the vertices in V'_1 by assigning the vertices to the columns of a grid; the y-coordinates of the vertices in V'_2 are determined analogously. Let n'_1 be the number of vertices in V'_1 and let $v_1 \to v_2 \to \cdots \to v_{n'_1}$ be the linear order of the vertices of V'_1 along the spine in ξ'_1. We start by placing v_1. Between any two consecutive vertices v_i and v_{i+1}, we reserve several columns for the bends of the edges incident to v_i and v_{i+1}; see Figure 7.3.
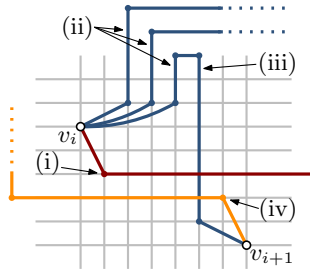
Figure 7.3: Reserving additional columns between v_i and v_{i+1}

The columns are used (in the given order) for the following purposes:

(i) for the first bend on all edges in A_2 leaving v_i,

(ii) for the first vertical segment of each edge $(v_i, v_j) \in E_1'$ with $j > i$,

(iii) for the last vertical segment of each edge $(v_k, v_{i+1}) \in E_1'$ with $k \leq i$, and

(iv) for the last bend on all edges in B_2 entering v_{i+1}.

Note that we can save some columns reserved for ((ii)) and ((iii)) because an edge in A_1 and an edge in B_1 can use the same column for their bend.

With this procedure, we fully specify the x-coordinates of the vertices in V_1'; the y-coordinates of the vertices in V_2' are determined analogously. Let R be the smallest axis-aligned rectangle that contains all vertices of the common vertex set of $\mathcal{G}_1'$ and $\mathcal{G}_2'$ (the gray rectangle in Figure 7.4a). Note that the y-coordinates of the dummy vertices of V_1' and the x-coordinates of the dummy vertices of V_2' have not been determined by the algorithm so far. They can be set arbitrarily, as long as the corresponding vertices are inside R.

We proceed to describe how to draw the edges of graph $\mathcal{G}_1'$ with at most four bends per edge such that all edge segments of $\mathcal{G}_1'$ in R are either vertical or of y-length exactly 1; see Figure 7.4b. The edges of graph $\mathcal{G}_2$ are drawn analogously (rotated by 90°). First, we draw the edges $(v_i, v_j) \in A_1$ with $i < j$ in a nested order: When we draw the edge (v_i, v_j), there is no edge $(v_k, v_l) \in A_1$ with $k \leq i$ and $l \geq j$ that has not already been drawn. Recall that the first column to the right and the first column to the left of every vertex is reserved for the edges in E_2, hence we assume that they are already used. We draw (v_i, v_j) with at most four bends as follows.

We start with a slanted segment incident to v_i that has its other endpoint in the row above v_i and in the first unused column that does not lie to the left of v_i. We follow with an upward vertical segment that leaves R. We add a horizontal segment above R; the row of this segment is determined by the nesting in the book embedding of $\mathcal{G}_1'$. In the last unused column that does not lie to the right of v_j, we add a vertical segment that ends one row above v_j. We finish the edge with a slanted segment that has its endpoint in v_j. We draw the edges in B_1 symmetrically, with the horizontal segment below R.

Note that this algorithm always uses the top and the bottom port of a vertex v if there is at least one edge incident to v in A_1 and B_1 respectively. The edges incident to a dummy vertex

(a) $\mathcal{G}_1$ **(b)** $\mathcal{G}_1$ and $\mathcal{G}_2$
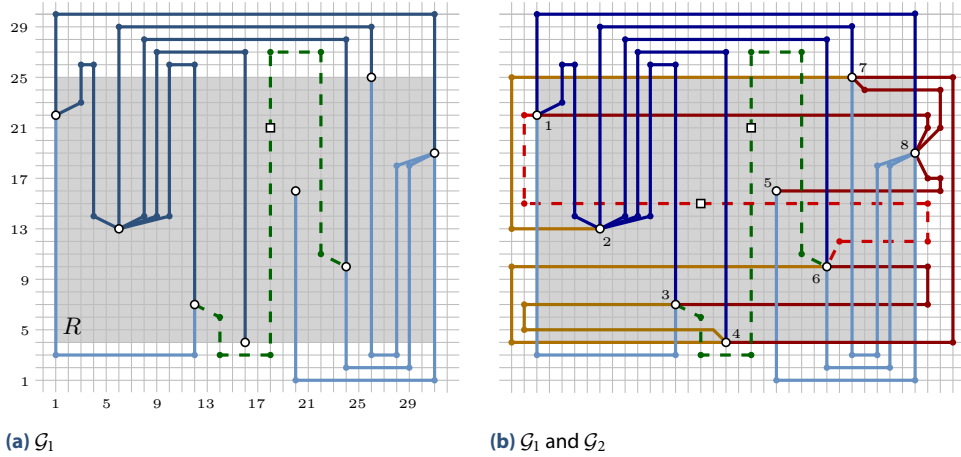
Figure 7.4: RacSim drawings of the graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ (see Figure 7.2) with at most six bends per edge, generated by our algorithm. The edges that cross the spine are drawn dashed; the dummy vertices on these edges are drawn as squares.

use precisely the top and the bottom port: Dummy vertices have exactly one incident edge in A_1 and one in B_1. We create a drawing of $\mathcal{G}_1$ and $\mathcal{G}_2$ with at most 6 bends per edge by bypassing (that is, smoothing out) the dummy vertices. This does not change the drawing.

By construction, all edge segments of E_1 inside R are either vertical segments or slanted segments of y-length 1. Symmetrically, all segments of E_2 inside R are either horizontal segments or slanted segments of x-length 1. Thus, the slanted segments cannot intersect. All crossings inside R occur between a horizontal and a vertical segment, and thus form right angles. Also, there are no segments in E_1 that lie to the left or to the right of R, and there are no segments in E_2 that lie above or below R. Hence, there are no crossings outside R, which guarantees that the constructed drawing of $\mathcal{G}_1$ and $\mathcal{G}_2$ is a RacSim drawing.

We now count the columns used by the drawing. For the leftmost and the rightmost vertex, we reserve one additional column for its incident edges in E_2; for the remaining vertices, we reserve two such columns. For each edge in E_1, we use up to three columns: one for each endpoint of the slanted segment at each vertex and one for the vertical segment that crosses the spine, if it exists. Recall that at least one edge per vertex does not need a slanted segment. For each edge in E_2, we need at most one column for the vertical segment to the side of R. Since there are at most $3n - 6$ edges, we need at most

$$3n - 2 + 3 \cdot (3n - 6) - n + 3n - 6 = 14n - 26$$

columns. By symmetry we need the same number of rows.

The algorithm of Kaufmann and Wiese computes a drawing with at most 3 bends per edge in $O(n)$ time. We can compute the nested order of the edges in linear time from the embedding, as the circular order of the edges around a vertex gives a hierarchical order on the edges that describes the nested order of the edges. Thus, our algorithm also runs in $O(n)$ total time. □

We can improve the result of Theorem 7.1 for subhamiltonian graphs, which admit 2-page book embeddings in which no edges cross the spine [BK79]. Since those edges are the only ones that need six bends, the number of bends per edge is reduced to four. The number of columns and rows are also reduced by one per edge. This is summarized in the following corollary. Note, however, that it is NP-hard to decide whether a given planar graph is subhamiltonian, even for maximal planar graphs [Wig82]. A 2-page book embedding can be found in linear time, if the ordering of the vertices along the spine is given [HS99]. Several classes of graphs are known to be (sub)hamiltonian, for example 4-connected planar graphs [NC08], planar graphs without separating triangles [KO07], Halin graphs [CNP83], planar graphs with maximum degree 3 or 4 [BGR14, Hea85].

Corollary 7.1. *Two subhamiltonian graphs on a common set of n vertices admit a RAcSim drawing on an integer grid of size $(11n - 32) \times (11n - 32)$ with four bends per edge. The algorithm runs in $O(n)$ time if the subhamiltonian cycles of both graphs are given.*

We use even fewer bends for a RacSim drawing of two outerplanar graphs. This is based on a decomposition of each of the graphs into two forests. The following lemma shows that we can do this in linear time.

Lemma 7.1. *Every outerplanar graph can be decomposed into two forests. This decomposition can be computed in linear time.*

Proof. It follows by Nash-Williams' formula [NW64] that every outerplanar graph has arboricity 2, that is, it can be decomposed into two forests. To prove the linear running time, we first assume biconnectivity and augment the input graph to a maximal outerplanar graph. Now, if we add a new vertex that is incident to all vertices of the graph, the result is a maximal planar graph which can be decomposed into three trees [Fel04], so that one of them is a star incident to the newly added vertex. Hence, the removal of this vertex yields a decomposition into two trees. The desired decomposition into two forests follows from the removal of the edges added to augment the graph to maximal outerplanar. For an outerplanar graph that is not biconnected, we have to compute the aforementioned decomposition for each of its biconnected components individually. This forms a tree (a so-called *BC-tree*) and therefore does not affect the structure of the overall decomposition: It still consists of two forests. The overall the decomposition can be done in linear time, since both the decomposition of a maximal planar graph into three trees, and the computation of the biconnected components of the input outerplanar graph can be done in linear time. □

With this lemma, we can further improve the required number of bends per edge to three for outerplanar graphs. (Recall that all outerplanar graphs are 1-page book embeddable.) We will use the order of the vertices on the spine of a 1-page book embedding to compute a 2-page book embedding in which every edge uses a rectilinear port at one of its endpoints, enabling us to omit one of its bends.

Theorem 7.2. *Two outerplanar graphs on a common set of n vertices admit a RAcSim drawing on an integer grid of size $(7n - 10) \times (7n - 10)$ with three bends per edge. The drawing can be computed in $O(n)$ time.*

Proof. Let $\mathcal{O}_1 = (V, E_1)$ and $\mathcal{O}_2 = (V, E_2)$ be the given outerplanar graphs. We will embed $\mathcal{O}_1$ and $\mathcal{O}_2$ on two pages with one forest per page.

To do so, we first create 1-page book embeddings for $\mathcal{O}_1$ and for $\mathcal{O}_2$ using the linear time algorithm of Heath [Hea87]. This gives the orders of the vertices of both graphs along the spine. It follows by Corollary 7.1 that, by using the algorithm described in the proof of Theorem 7.1, we can create a RacSim drawing of $\mathcal{O}_1$ and $\mathcal{O}_2$ with at most four bends per edge. We will now show how to adjust the algorithm to reduce the number of bends by one.

We decompose $\mathcal{O}_1$ into two forests A_1 and B_1 according to Lemma 7.1. We will draw the edges of A_1 above the spine and the edges B_1 below the spine. By rooting the trees in A_1 in arbitrary vertices, we can direct each edge such that every vertex has exactly one incoming edge. Recall that, in the drawing produced in Theorem 7.1, one edge per vertex can use its top port. We adjust the algorithm such that every directed edge (v, w) enters vertex w from its top port. To do so, we draw the edge as follows. We start with a slanted segment of y-length 1. We follow with a vertical segment to the top, a horizontal segment that ends directly above w, and finish the edge with a vertical segment that enters w in the top port. We use the same approach for the edges in B_1, using the bottom port and treat the second outerplanar graph $\mathcal{O}_2$ analogously.

Since every port of a vertex is only used once, the drawing has no overlaps. We now analyze the number of columns used. For every vertex except for the leftmost and rightmost, two additional columns are reserved for the edges in E_2; for the remaining two vertices, we reserve a single additional column. The edges in E_1 now only need one column for the bend of the single slanted segment. For every edge in E_2, we need up to one column for the vertical segment to the side of R. Since there are at most $2n - 4$ edges, our drawing needs $3n - 2 + 2n - 4 + 2n - 4 = 7n - 10$ columns. Analogously, we can show that the algorithm needs $7n - 10$ rows. Since the decomposition can be computed in $O(n)$ time, our algorithm also requires $O(n)$ time. □

7.2 RacSim and RacSefe Drawings with One Bend per Edge

In this section, we study simple classes of planar graphs and show how to efficiently construct RacSim and/or RacSefe drawings with one bend per edge in quadratic area. In particular, we prove that two cycles (four matchings, resp.) on a common set of n vertices admit a RacSefe (RacSim, resp.) drawing on an integer grid of size $2n \times 2n$; see Theorem 7.3 (Theorem 7.5, resp.). If the input to our problem is a caterpillar and a cycle, then we can compute a RacSefe drawing with one bend per edge on an integer grid of size $(2n - 1) \times 2n$; see Theorem 7.4. For a tree and a cycle, we can construct a RacSefe drawing with one bend per tree edge and no bends in the matching edges on an integer grid of size $n \times (n - 1)$; see Theorem 7.6.

In the next proof and in a few more places throughout this chapter, we use the following common notation. For any real x, let the *sign* of x, $\text{sgn}(x)$, be 0 if $x = 0$, 1 if $x > 0$, and -1 if $x < 0$.

Lemma 7.2. *Two paths on a common set of n vertices admit a RacSefe drawing on an integer grid of size $2n \times 2n$ with one bend per edge. The drawing can be computed in $O(n)$ time.*
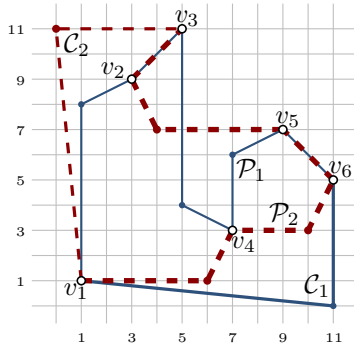
Figure 7.5: RacSefe drawings with one bend per edge: two paths $\mathcal{P}_1$ (thin solid) and $\mathcal{P}_2$ (bold dashed) and two cycles $\mathcal{C}_1 = \mathcal{P}_1 + (v_1, v_6)$ and $\mathcal{C}_2 = \mathcal{P}_2 + (v_1, v_3)$

Proof. Let $\mathcal{P}_1 = (V, E_1)$ and $\mathcal{P}_2 = (V, E_2)$ be the given paths. To keep the description simple, we first assume that $\mathcal{P}_1$ and $\mathcal{P}_2$ do not share edges. Following standard practices from the literature (see for example Brass et al. [BCD⁺07]), we draw $\mathcal{P}_1$ x-monotone and $\mathcal{P}_2$ y-monotone. This ensures that the drawing of the paths will individually be planar. We will now describe how to compute the exact coordinates of the vertices and how to draw the edges of $\mathcal{P}_1$ and $\mathcal{P}_2$, such that all crossings are at right angles and, more importantly, that no edge segments overlap.

For $m \in \{1, 2\}$ and any vertex $v \in V$, let $\pi_m(v)$ be the position of v in $\mathcal{P}_m$. Then, v is drawn at the point $p(v) = (2\pi_1(v) - 1, 2\pi_2(v) - 1)$; see Figure 7.5. It remains to determine, for each edge $e = (v, v')$, where to place its bend. First, assume that $e \in E_1$ and that e is directed from its left endpoint, say v, to its right endpoint, say v'. Then we place the bend to the left of v', and one row above or below it depending on which direction the edge is coming from. To be exact, we place it at $p(v') - (2, \text{sgn}(y(v') - y(v)))$. Second, assume that $e \in E_2$ and e is directed from its bottom endpoint, say v, to its top endpoint, say v'. Then, we place the bend at $p(v') - (\text{sgn}(x(v') - x(v)), 2)$.

The area required by the drawing is $(2n-1) \times (2n-1)$. An edge of $\mathcal{P}_1$ leaves its left endpoint vertically and enters its right endpoint with a slanted segment of x-length 1 and y-length 2. Similarly, an edge of $\mathcal{P}_2$ leaves its bottom endpoint horizontally and enters its top endpoint with a slanted segment of x-length 2 and y-length 1. Hence, the slanted segments cannot be involved in crossings or overlaps. Since $\mathcal{P}_1$ and $\mathcal{P}_2$ are x- and y-monotone, respectively, it follows that all crossings must involve a vertical edge segment of $\mathcal{P}_1$ and a horizontal edge segment of $\mathcal{P}_2$, which is at right angle. The runtime is clearly linear.

Finally, we show that our algorithm supports the Sefe model. Assume that $\mathcal{P}_1 = (V, E_1)$ and $\mathcal{P}_2 = (V, E_2)$ share edges, and let $e = (v, v')$ be an edge that belongs to both input paths. Our algorithm automatically places v and v' at consecutive x- and y-coordinates and draws e as a diagonal of x- and y-length 2 in both graphs, which cannot be involved in a crossing. □

We say that an edge uses the bottom/left/right/top *port* of a vertex if it enters the vertex from the bottom/left/right/top.

Theorem 7.3. *Two cycles on a common set of n vertices admit a RacSefe drawing on an integer grid of size $2n \times 2n$ with at most one bend per edge. The drawing can be computed in $O(n)$ time.*

Proof. Let $\mathcal{C}_1 = (V, E_1)$ and $\mathcal{C}_2 = (V, E_2)$ be the given cycles and assume first that $\mathcal{C}_1$ and $\mathcal{C}_2$ do not share edges. Let $v \in V$ be an arbitrary vertex. We temporarily delete one edge $(v, w_1) \in E_1$ from $\mathcal{C}_1$ and one edge $(v, w_2) \in E_2$ from $\mathcal{C}_2$ (refer to the edges (v_1, v_3) and (v_1, v_6) in Figure 7.5). This results into two paths $\mathcal{P}_1 = \langle v, \dots, w_1 \rangle$ and $\mathcal{P}_2 = \langle v, \dots, w_2 \rangle$. We use the algorithm of Lemma 7.2 to construct a RacSim drawing of $\mathcal{P}_1$ and $\mathcal{P}_2$ on an integer grid of size $(2n-1) \times (2n-1)$. Since v is the first vertex in both paths, it is placed at the bottom-left corner of the bounding box containing the drawing. Since w_1 and w_2 are the last vertices in $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively, w_1 is placed on the right side, and w_2 on the top of the bounding box. By construction, the bottom port of w_1 and the left port of w_2 are both unoccupied. Hence, the edges (v, w_1) and (v, w_2) that form $\mathcal{C}_1$ and $\mathcal{C}_2$ can be drawn with a single bend at points $(2n-1, 0)$ and $(0, 2n-1)$ respectively; see Figure 7.5. Since both edges are completely outside of the bounding box containing the drawing, neither is involved in crossings. The total area of the drawing gets larger by a single unit in each dimension. The runtime bound is unaffected.

It remains to consider the case that $\mathcal{C}_1$ and $\mathcal{C}_2$ share edges. Since $\mathcal{P}_1$ and $\mathcal{P}_2$ already support the Sefe model, it suffices to consider the case that a closing edge (v, w_1) or (v, w_2) is also contained in the other graph. Since the closing edges are drawn planar, we can simply use their drawing in both graphs and remove the corresponding edge from the path. Thus, the drawing supports the Sefe model. □

Theorem 7.4. *A caterpillar and a cycle on a common set of n vertices admit a RacSefe drawing on an integer grid of size $(2n-1) \times 2n$ with one bend per edge. The drawing can be computed in $O(n)$ time.*

Proof. Let $\mathcal{A} = (V, E_{\mathcal{A}})$ be the given caterpillar and $\mathcal{C} = (V, E_{\mathcal{C}})$ the given cycle. Similar to the previous proofs, we will first prove that $\mathcal{A}$ and $\mathcal{C}$ admit a RacSim drawing, assuming that they do not share edges. We postpone the case where $\mathcal{A}$ and $\mathcal{C}$ share edges for later. A caterpillar can be decomposed into a path, called its *spine*, and a set of leaves connected to the path, called its *legs*. Let $v_1, v_2, \dots, v_n$ be the vertex set of $\mathcal{A}$ ordered as follows (see Figure 7.6): Starting from an endpoint of the spine of $\mathcal{A}$, we traverse the caterpillar such that we visit all legs incident to a spine vertex before moving on to the next spine vertex. This order defines the x-order of the vertices in the output drawing.

As in the proof of Theorem 7.3, we temporarily delete an edge of $\mathcal{C}$ incident to v_1 (see the thin dashed edge in Figure 7.6) and obtain a path which we denote by $\mathcal{P} = (V, E_{\mathcal{P}})$. For any vertex $v_i \in V$, let $\pi(v_i)$ be the position of v_i in $\mathcal{P}$. The map π determines the y-order of the vertices in our drawing. For $i \in \{1, 2, \dots, n\}$, we draw vertex v_i at point $p(v_i) = (2i - 1, 2\pi(v_i) - 1)$. It remains to determine, for each edge $e = (v, v')$, where to place its bend. First, assume that $e \in E_{\mathcal{P}}$ and that e is directed from its bottom endpoint, say v, to its top endpoint, say v' (see the bold dashed edges in Figure 7.6). Then, we place the bend at $p(v) + (\text{sgn}(x(v') - x(v)), 2)$. Second, assume that $e \in E_{\mathcal{A}}$ and e is directed from its left endpoint, say v, to its right endpoint, say v' (see the solid edges in Figure 7.6). Then, we place the bend at $(x(v'), y(v) + \text{sgn}(y(v') - y(v))$.

The approach described above ensures that $\mathcal{P}$ is drawn y-monotone, hence planar. The spine of $\mathcal{A}$ is drawn x-monotone. The legs of a spine vertex of $\mathcal{A}$ are drawn to the right of
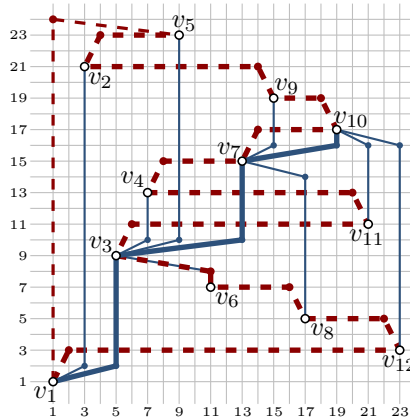
Figure 7.6: A RacSefe drawing of a caterpillar (solid; its spine is drawn bold) and a cycle (dashed)

their parent spine vertex and to the left of the next vertex along the spine. Hence, $\mathcal{A}$ is drawn planar as well. The slanted segments of $\mathcal{A}$ have y-length 1, while the slanted segments of $\mathcal{P}$ have x-length 1. Thus, they cannot be involved in crossings, which implies that all crossings form right angles.

It remains to draw the edge e in $E_{\mathcal{C}} \setminus E_{\mathcal{P}}$. Recall that e is incident to v_1, which lies at the bottom-left corner of the bounding box containing our drawing. Let v_j be the other endpoint of e. Since $\pi(v_j) = n$, vertex v_j lies at the top of the bounding box. As the top port of v_1 is not used, we can draw the first segment of e vertical, bending at $(1, 2n)$; see the thin dashed edge in Figure 7.6.

To complete the proof of this theorem, it remains to show that our algorithm supports the Sefe model. Suppose that there is an edge $e = (v, v')$ that belongs to both $\mathcal{A}$ and $\mathcal{C}$. Our algorithm places v and v' at consecutive y-coordinates. Thus, the drawing of e in $\mathcal{A}$ consists of a slanted and a vertical segment of y-length 1 each. This drawing of e in $\mathcal{A}$ is not crossed by any edge of $E_{\mathcal{C}}$, so e can be drawn in the same way for both graphs.

Clearly, the area of the drawing is $(2n - 1) \times 2n$, and the runtime is linear. □

Theorem 7.5. *Four matchings on a common set of n vertices admit a RacSim drawing on an integer grid of size $2n \times 2n$ with at most one bend per edge. The drawing can be computed in $O(n)$ time.*

Proof. Let $\mathcal{M}_1 = (V, E_1)$, $\mathcal{M}_2 = (V, E_2)$, $\mathcal{M}_3 = (V, E_3)$ and $\mathcal{M}_4 = (V, E_4)$ be the given matchings. Without loss of generality, we assume that all matchings are perfect; otherwise, we augment them to perfect matchings. Let $\mathcal{M}_{1,2} = (V, E_1 \cup E_2)$ and $\mathcal{M}_{3,4} = (V, E_3 \cup E_4)$. Since $\mathcal{M}_1$ and $\mathcal{M}_2$ are defined on the same vertex set, $\mathcal{M}_{1,2}$ is a 2-regular graph. Thus, each connected component of $\mathcal{M}_{1,2}$ corresponds to a cycle of even length which alternates between edges of $\mathcal{M}_1$ and $\mathcal{M}_2$; see Figure 7.7. The same holds for $\mathcal{M}_{3,4}$. We will determine the x-coordinates of the vertices from $\mathcal{M}_{1,2}$, and the y-coordinates from $\mathcal{M}_{3,4}$.

We start by choosing an arbitrary vertex $v \in V$. Let $\mathcal{C}$ be the cycle of $\mathcal{M}_{1,2}$ containing vertex v. We determine the x-coordinates of the vertices of $\mathcal{C}$ by traversing it in some direction,
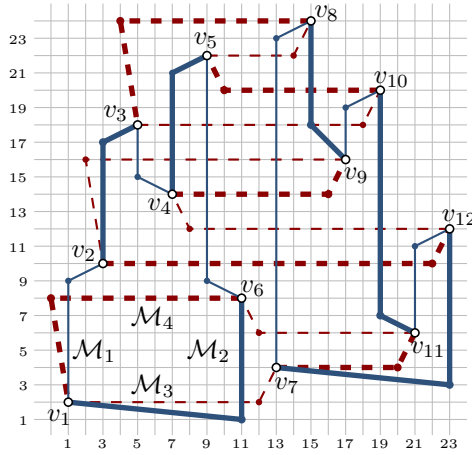
Figure 7.7: A RacSim drawing of four matchings: $\mathcal{M}_1$ (solid-plain), $\mathcal{M}_2$ (solid-bold), $\mathcal{M}_3$ (dashed-plain), and $\mathcal{M}_4$ (dashed-bold)

starting from vertex v. For each vertex u in $\mathcal{C}$, let $\pi_1(u)$ be the discovery time of u according to this traversal, with $\pi_1(v) = 0$. We set $x(u) = 2\pi_1(u) + 1$. After doing this, we determine the y-coordinates of all vertices that lie on cycles in $\mathcal{M}_{3,4}$ that contain at least one vertex of $\mathcal{C}$ (that is, cycles in $\mathcal{M}_{3,4}$ that contain a vertex for which the x-coordinate has been determined). Call these cycles $\mathcal{C}_1, \ldots, \mathcal{C}_k$, ordered as follows. For $i \in \{1, \ldots, k\}$, let a_i be the *anchor* of $\mathcal{C}_i$, that is, the vertex with the smallest determined x-coordinate of all vertices in $\mathcal{C}_i$. Then, $x(a_1) < \ldots < x(a_k)$. In what follows, we start with the first cycle $\mathcal{C}_1$ of the computed order and determine the y-coordinates of its vertices. To do so, we traverse $\mathcal{C}_1$ in some direction, starting from its anchor vertex a_1. For each vertex u in $\mathcal{C}_1$, let $\pi_2(u)$ be the discovery time of u according to this traversal, with $\pi_2(a_1) = 0$. Then, we set $y(u) = 2\pi_2(u) + 1$. We proceed analogously with the remaining cycles $\mathcal{C}_i$, $i = 2, \ldots, k$, setting $\pi_2(a_i) = \max_{u \in \mathcal{C}_{i-1}} \pi_2(u) + 1$.

After this step, there are no vertices for which only the x-coordinate has been determined. However, there might exist vertices where only the y-coordinate has been determined. If this is the case, we repeat the aforementioned procedure to determine the x-coordinates of the vertices of all cycles of $\mathcal{M}_{1,2} \smallsetminus \mathcal{C}$ that have at least one vertex with a determined y-coordinate, but without determined x-coordinates. If there are no vertices with only one determined coordinate left, either all coordinates are determined, or we restart this procedure with another arbitrary vertex that has no determined coordinates. Thus, our algorithm guarantees that the x- and y-coordinate of all vertices are eventually determined.

Note that for each cycle in $\mathcal{M}_{1,2}$ there is exactly one edge $e = (v, v')$ with $\pi_1(v') > \pi_1(v) + 1$. We call this the *closing edge*. Analogously, for each cycle in $\mathcal{M}_{3,4}$, there is exactly one closing edge $e = (u, u')$ with $\pi_2(u') > \pi_2(u) + 1$.

Finally we determine, for each edge $e = (v, v')$, where to place its bend. First, assume that $e \in E_1 \cup E_2$ and that e is directed from its left endpoint, say v, to its right endpoint, say v'. If e is not a closing edge, we place the bend at $(x(v') - 2, y(v') - \text{sgn}(y(v') - y(v)))$. Otherwise, we place the bend at $(x(v'), y(v) - 1)$. Second, assume that $e \in E_3 \cup E_4$ and e is directed from its bottom endpoint, say v, to its top endpoint, say v'. If e is not a closing edge,
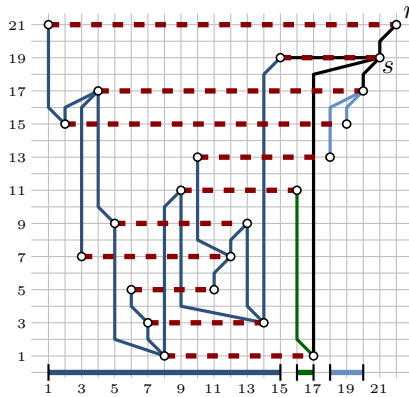
Figure 7.8: A RacSefe drawing of a tree (solid; rooted in r) and a matching (dashed). For each of the three subtrees hanging off vertex s we used a different color; the subtrees occupy disjoint x-intervals (marked below the drawing).

we place the bend at $(x(v') - \text{sgn}(x(v') - x(v)), x(v') - 2)$. Otherwise, we place the bend at $(x(v) - 1, y(v'))$; see Figure 7.7.

Our choice of coordinates guarantees that the x-coordinates of the cycles of $\mathcal{M}_{1,2}$ and the y-coordinates of the cycles of $\mathcal{M}_{3,4}$ form disjoint intervals. Thus, the area below a cycle of $\mathcal{M}_{1,2}$ and the area to the left of a cycle of $\mathcal{M}_{3,4}$ are free from vertices. Hence, the slanted segments of the closing edges cannot have a crossing that violates the RAC restriction. The total area required by the drawings is $2n \times 2n$. The running time is linear. □

We now detail how to compute a RacSefe drawing of a rooted tree and a matching with one bend per tree edge and no bends on matching edges. Figure 7.8 shows an example output of our algorithm. Our layout algorithm is inspired by an algorithm for drawing a geometric simultaneous embedding of a tree and a matching by Cabello et al. [CvL+11], which in turn goes back to an algorithm of Di Giacomo et al. [DDvK+09]. Cabello et al. draw edges straight (that is, no bends), but their crossing angles are not necessarily right. We recall some of their notation that we will then use for our purposes.

Cabello et al. draw the edges of the matching horizontally. They partition the matching into a *top group* and a *bottom group*. Within the top group, they place the edges from top to bottom; vice versa within the bottom group. Once a matching edge is assigned to one of the two groups, the edge and its endpoints are called *placed*.

The assignment of the edges to the groups is iterative. In each step, the placed vertices induce an edge partition of the tree into connected components (subtrees); each component up to and including the placed vertices is called a *rope*. If a rope with three placed vertices exists, the vertex that lies on all three paths between these placed vertices is called a *splitter*; see Figure 7.9a. Cabello et al. showed that placing a vertex creates at most one splitter and that placing a splitter does not create a new one.

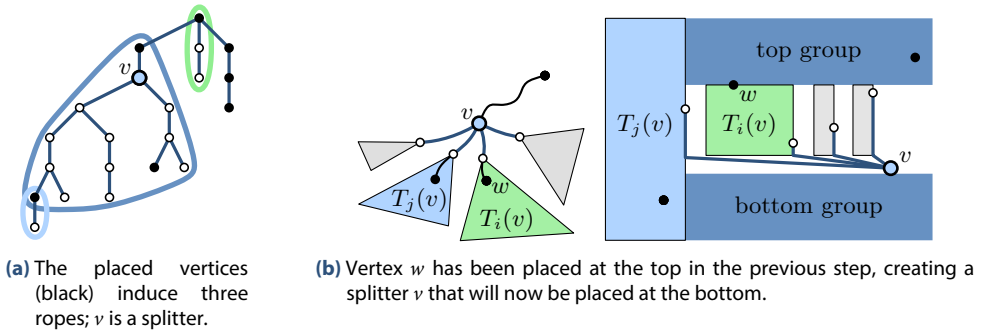Now, we are ready to present our algorithm in detail.

(a) The placed vertices (black) induce three ropes; v is a splitter.

(b) Vertex w has been placed at the top in the previous step, creating a splitter v that will now be placed at the bottom.

Figure 7.9: Definition and placement of a splitter

Theorem 7.6. *A tree and a matching on a common set of n vertices admit a RacSefe drawing on an integer grid of size $n \times (n - 1)$ with one bend per tree edge and no bends on matching edges. The drawing can be computed in $O(n)$ time.*

Proof. We first consider the case where the input graphs do not share edges. We root the given tree in an arbitrary leaf r, which yields a directed tree $\mathcal{T}$. We will obtain the x-coordinates of the vertices from a particular post-order traversal of this directed tree. As a consequence, all children of a vertex are placed to its left, and disjoint subtrees are placed in disjoint x-intervals. By adding dummy edges, we augment the given matching to a perfect matching if n is even, or to a near-perfect matching if n is odd. Let $\mathcal{M}$ be the augmented matching. After the algorithm terminates, these dummy edges can be safely removed. In order to compute a RacSefe drawing of $\mathcal{T}$ and $\mathcal{M}$, we follow an approach that is based on the one of Cabello et al. The difference is that (a) rather than placing ropes in disjoint parallelograms, we place subtrees into disjoint axis-parallel rectangles, and (b) the assignment of matching edges to the two groups is much simpler.

First, we show how to compute the y-coordinates of the vertices. Each edge of $\mathcal{M}$ is drawn horizontally at a unique odd y-coordinate between 1 and $n-1$. The coordinates are determined as follows. We start by putting the matching edge containing the root of $\mathcal{T}$ into the top group. Within the top group, the edges are assigned to odd rows, from top to bottom, starting from row $n - 2$ if n is odd, or $n - 1$ otherwise. In the bottom group, the edges are also assigned to odd rows, but from bottom to top, starting from row 1. In this way, a matching edge is always assigned to an odd row between 1 and $n - 1$. Thus, our drawing needs at most $n - 1$ rows. In the following we show how to determine the group into which a matching edge goes. There are two cases.

If there is no splitter, the algorithm finds an unplaced vertex that has a tree edge to a placed vertex, and adds its matching edge to the top group. This creates at most one splitter since one of the placed vertices is adjacent to a vertex that has already been placed. Hence, whenever a new matching edge is placed, there will be at most one splitter in the graph.

Now, assume that a splitter exists. Call it v. For any subtree, we call the unique vertex with no incoming edge its *local root*. Since we start by drawing the root of $\mathcal{T}$, every vertex lies in a rope that has its local root placed. Let $T_1(v), \ldots, T_k(v)$ be the subtrees of $\mathcal{T}$ hanging off v; see the left drawing of Figure 7.9b. Then, v is a splitter if and only if there are two

subtrees $T_i(v)$ and $T_j(v)$ with at least one placed vertex, one of which has been placed in the last step. Without loss of generality, let $w \in T_i(v)$ be this vertex. If w was added to the top group, we add v to the bottom group; otherwise, we add v to the top group. Thus, the vertices of the subtrees $T_l(v)$ with $l \neq j$ will be placed either all above or all below v; see Figure 7.9b.

Next, we describe how to determine the x-coordinates of the vertices. We proceed inductively. For each non-leaf vertex, we compute an order of the subtrees that hang off this vertex. This order determines the disjoint x-intervals into which we place the subtrees. Let v be a vertex that is placed in an induction step. We traverse the path from v to the root of the rope it lies on. For every vertex u on this path, we determine the order of the subtrees of its children as follows. Let $T_1(u), \ldots, T_\ell(u)$ be the subtrees hanging off u, assuming $v \in T_1(u)$. If v is the only placed successor of vertex u, then we assign the order $x(T_1(u)) < \ldots < x(T_\ell(u))$ to the x-intervals of the subtrees; otherwise, an order has already been determined. When the algorithm is done, we know the order of the subtrees for every vertex on this path. In particular, we know the x-interval of the subtree rooted in v. We assign the largest x-coordinate of this interval to v.

Now, we show how to draw the edges. Let (u, v) be a directed edge of $\mathcal{T}$. We draw (u, v) with a slanted segment of y-length 1 at vertex u and a vertical segment at vertex v. The bend of the edge lies at $(x(v), y(v) + \mathrm{sgn}(y(u) - y(v)))$. (Note that, if $(u, v) \in \mathcal{M}$, then the bend lies at v, that is, the edge is drawn horizontally.) Since we draw the edges of the matching horizontally without bends and since the slanted segments are drawn between two consecutive horizontal grid lines, there can only be crossings between vertical segments of the tree and horizontal segments of the matching. Thus, all crossings between the tree and the matching are at right angles.

It remains to show that the drawing of the tree itself is planar. Since the edges of the tree are drawn with a slanted segment and a vertical segment, each crossing must involve a slanted segment. Let v be a vertex of the tree. We will show that the slanted segments of the edges leaving v do not induce a crossing. Since the subtree rooted in v is assigned an x-interval that contains only vertices of the subtree, crossings can only occur with edges of this subtree. Recall that v lies on the right border of this x-interval, so all edges leaving v are directed to the left. Consider the step of the algorithm in which v is placed.

First, assume that v is not a splitter. If no successors of v have been placed so far, they will be placed all above or all below v. Therefore, the slanted segments of the edges leaving v will not induce any crossing. Otherwise, let $T_1(v), \ldots, T_k(v)$ be the subtrees hanging off v. By construction, all placed successors of v are located in the same subtree $T_1(v)$, and $T_1(v)$ is placed to the left of the other subtrees hanging off v. Thus, no edge leaving v is drawn inside the x-interval assigned to $T_1(v)$. The vertices in the other subtrees will be placed all above or all below v. Therefore, the slanted segments of the edges leaving v will induce no crossing.

Second, assume that v is a splitter. Then, there is a vertex w that was placed in the previous step and lies in the same rope as v; see Figure 7.9b. Further, all placed successors of v except w are located in the same subtree $T_1(v)$, and $T_1(v)$ is placed to the left of the other subtrees hanging off v. Hence, no edge leaving v is drawn inside the x-interval assigned to $T_1(v)$. Recall that v is placed in the group opposite of w. Thus, w and all unplaced successors of v (including all vertices in $T_2(v), \ldots, T_k(v)$), lie all above or all below v. Therefore, the slanted segments of the edges leaving v do not induce any crossing. This concludes the proof of planarity.

Note that our algorithm supports the SEFE model. Edges that lie both in $\mathcal{M}$ and $\mathcal{T}$ are drawn the same way; see the edge incident to vertex s in Figure 7.8.

Finally, we prove the area and running time bounds. Recall that each edge of $\mathcal{M}$ is drawn horizontally at a unique odd y-coordinate between 1 and $n - 1$. In every column, we place exactly one vertex, so the drawing needs n columns. As for the running time, the algorithm to place the vertices clearly requires only constant time per vertex, except when we traverse the tree upwards to determine the x-coordinate of a new vertex. As soon as we hit a vertex whose x-coordinate has already been determined, we are done, assuming that we have precomputed the sizes of all subtrees (which can be done in linear total time). During the traversal, we fix the x-coordinates of all vertices that we meet. Hence, in order to determine the x-coordinates of all vertices, we traverse every edge of $\mathcal{T}$ at most once. Thus, the running time of the algorithm is linear. □

## 7.3 RacSefe Drawings with Two Bends per Edge

In this section, we study slightly more complex classes of planar graphs, and show how to efficiently construct RACSEFE drawings with two bends per edge, in quadratic area. In particular, we prove that a wheel and a matching on a common set of n vertices admit a RACSEFE drawing on an integer grid of size $(1.5n - 1) \times (n + 2)$ with two bends per wheel edge and no bends on matching edges; see Theorem 7.7. An outerpath (that is, an outerplanar graph whose weak dual is a path) and a matching admit a RACSEFE drawing with two bends per outerpath edge and one bend per matching edge. They also need a slightly larger grid, namely one of size $(3n - 2) \times (3n - 2)$; see Theorem 7.8.

Theorem 7.7. *A wheel and a matching on a common set of n vertices admit a RACSEFE drawing on an integer grid of size $(1.5n - 1) \times (n + 2)$ with two bends per wheel edge and no bends on matching edges, respectively. The drawing can be computed in $O(n)$ time.*

Proof. Let $\mathcal{W} = (V, E_{\mathcal{W}})$ be the given wheel and $\mathcal{M} = (V, E_{\mathcal{M}})$ the given matching. A wheel can be decomposed into a cycle, called its *rim*, a center vertex, called its *hub*, and a set of edges that connect the hub to the rim, called its *spikes*. First, we consider the simpler case according to which $\mathcal{W}$ and $\mathcal{M}$ do not share edges (clearly, in this case the hub of the wheel cannot be incident to a matching edge). Let $V = \{v_1, v_2, \ldots, v_n\}$, such that v_1 is the hub of $\mathcal{W}$ and $\mathcal{C} = \langle v_2, v_3, \ldots, v_n, v_2 \rangle$ is the rim of $\mathcal{W}$ in this order. Thus, $E_{\mathcal{W}} = \{(v_i, v_{i+1}) \mid i = 1, \ldots, n-1\} \cup \{(v_n, v_2)\} \cup \{(v_1, v_i) \mid i = 2, \ldots, n\}$. Let $\mathcal{M}' = (V, E_{\mathcal{M}'})$ be the matching $\mathcal{M}$ without the edge incident to v_1.

We first compute the x-coordinates of the vertices. We do this such that $\mathcal{C} - \{(v_n, v_2)\}$ is x-monotone; see Figure 7.10. More precisely, for $i \in \{2, \ldots, n\}$, we set $x(v_i) = 2i - 3$. The y-coordinates of the vertices are computed based on the matching $\mathcal{M}'$ as follows. Let $E_{\mathcal{M}'} = \{e_1, \ldots, e_k\}$ be the matching edges, indexed such that v_2 is incident to e_1. For $j \in \{1, \ldots, k\}$, we assign the y-coordinate $2j - 1$ to the endpoints of e_j. Next, we assign the y-coordinate $2k + 1$ to the vertices incident to the rim without a matching edge in $\mathcal{M}'$. Finally, the hub v_1 of $\mathcal{W}$ is located at point $(1, 2k + 3)$.

It remains to place the bend of each edge $e \in E_{\mathcal{W}}$; the edges in $\mathcal{M}'$ are drawn without bends. First, let $e = (v_1, v_i)$, $i \in \{3, \ldots, n\}$ be a spike. Then, we place the bend at $(x(v_i), 2k + 2)$.
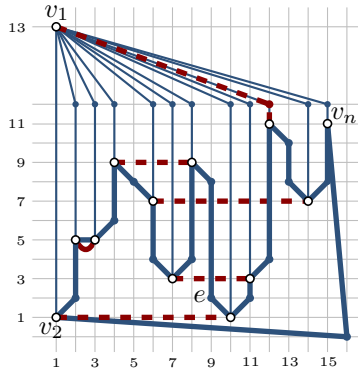
Figure 7.10: A RacSefe drawing of a wheel (solid; its rim is drawn in bold) and a matching (dashed)

Since both v_1 and v_2 are located in column 1, we can save the bend of the spike (v_1, v_2). Second, let $e = (v_i, v_{i+1})$, $i \in \{2, \ldots, n-1\}$ be an edge of the rim $\mathcal{C}$. We place the bend according to the following case distinction.

(i) If $y(v_{i+1}) > y(v_i)$, we place the bend at $(x(v_{i+1}), y(v_i) + 1)$.

(ii) If $y(v_{i-1}) > y(v_i) > y(v_{i+1})$, we place the bend at $(x(v_{i+1}), y(v_i) - 1)$.

(iii) If $y(v_i) > \max\{y(v_{i-1}), y(v_{i+1})\}$, by (i) the bottom port at v_i is already used; see the edge e in Figure 7.10. Thus, we draw e with two bends; at $(x(v_i) + 1, y(v_i) - 1)$ and $(x(v_i) + 1, y(v_{i+1}) + 1)$.

Now, let $e = (v_n, v_2)$ be the remaining edge of the rim. We place its bend at $(2n - 2, 0)$.

Our approach ensures that $\mathcal{C} - \{(v_n, v_2)\}$ is drawn x-monotone, hence planar. The last edge (v_n, v_2) of $\mathcal{C}$ is the only edge drawn outside of the bounding box that contains all vertices. Therefore, also the last edge is crossing-free. The spikes are not involved in crossings with the rim, as they are outside of the bounding box containing the rim edges. Hence, the drawing of $\mathcal{W}$ is planar. All edges of $\mathcal{M}'$ are drawn as horizontal, non-overlapping line segments, so $\mathcal{M}'$ is drawn planar as well. The slanted segments of $\mathcal{W} - (v_n, v_2)$ are of y-length 1, so they cannot be crossed by the edges of $\mathcal{M}'$. As the edge (v_n, v_2) is not involved in crossings, it follows that all crossings between $\mathcal{W}$ and $\mathcal{M}'$ form right angles.

Finally, we have to insert the matching edge (v_1, v_i) incident to the hub. Note that this edge also exists in $\mathcal{W}$ as a spike. Since v_i is not incident to a matching edge in $\mathcal{M}'$, it is placed above all matching edges. Therefore, the copy of (v_1, v_i) in $\mathcal{W}$ does not cross a matching edge, and we can use the same layout for the copy of (v_1, v_i) in $\mathcal{M}$.

We now show that our algorithm supports the SEFE model. Suppose that there is an edge $e = (v, v')$ that belongs to both the rim $\mathcal{C}$ and the matching $\mathcal{M}$. If e is the closing edge of the rim, then it is drawn planar in $\mathcal{C}$. Thus, this drawing can be used for both graphs. Otherwise, our algorithm places v and v' at consecutive x-coordinates and at the same y-coordinate. Then, we can draw e as a horizontal edge of length 1 in both graphs, and such an edge cannot be crossed.

We now prove the area bound of the drawing algorithm. To that end, we remove all columns that contain neither a vertex, nor a bend. First, we count the rows used. Since we remove the

(a) a non-planar drawing

(b) a RacSefe drawing
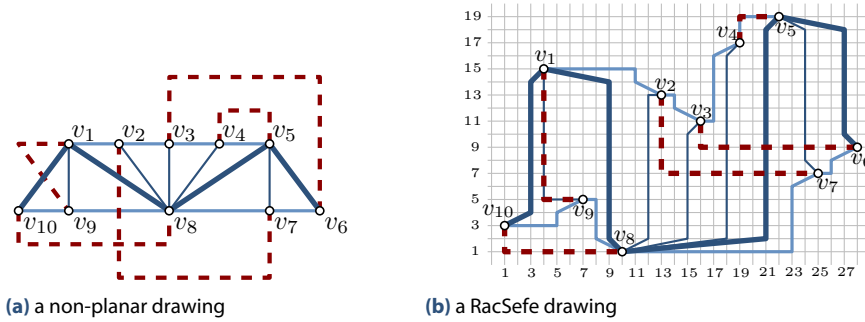
Figure 7.11: Two drawings of the same outerpath and matching. In both figures, the outerpath is drawn solid, the upper and the lower path are drawn in bold, the spine of the spanning caterpillar is drawn extra bold and the matching is drawn dashed.

matching edge incident to v_1, the matching $\mathcal{M}'$ has $k \le n/2 - 1$ matching edges. We place the bottommost vertex in row 1 and the topmost vertex (that is, vertex v_1) in row $2k + 3$. We add one extra bend in row 0 for the edge (v_n, v_2). Thus, our drawing uses $2k + 3 + 1 \le n + 2$ rows. Next, we count the columns used. The vertices $v_2, \ldots, v_n$ are each placed in their own column. Every spike has exactly one bend in the column of a vertex. An edge (v_i, v_{i+1}) of rim $\mathcal{W}$ has exactly one bend in a vertex column, except for the case that $y(v_i) > y(v_{i-1}), y(v_{i+1})$. In this case it needs an extra bend between v_i and v_{i+1}, $i = 1, \ldots, n-1$. Clearly, there can be at most $n/2 - 1$ vertices satisfying this condition. Since the edge (v_n, v_2) uses an extra column to the right of v_n, our drawing uses $(n - 1) + (n/2 - 1) + 1 = 1.5n - 1$ columns. $\qquad\square$

Theorem 7.8. *An outerpath and a matching on a common set of n vertices admit a RacSefe drawing on an integer grid of size $(3n - 2) \times (2n - 1)$ with two bends per outerpath edge and one bend per matching edge. The drawing can be computed in $O(n \log n)$ time.*

Proof. Let $\mathcal{Z} = (V, E_\mathcal{Z})$ be the given outerpath and $\mathcal{M} = (V, E_\mathcal{M})$ the given matching. Recall that an outerpath is a biconnected outerplanar graph whose weak dual is a path of length at least two; see Figure 7.11a. Furthermore, assume initially that $\mathcal{Z}$ and $\mathcal{M}$ do not share edges. Let $V = \{v_1, v_2, \ldots, v_n\}$, indexed such that $\langle v_1, v_2, \ldots, v_n, v_1 \rangle$ is the outer face of $\mathcal{Z}$.

We start by augmenting $\mathcal{Z}$ to a maximal outerpath $\mathcal{Z}' = (V, E_{\mathcal{Z}'})$ by triangulating its bounded faces. As $\mathcal{Z}'$ is internally-triangulated, it contains exactly two vertices of degree two, each of which lies on a face that is an endpoint of the dual path. We assume, without loss of generality, that $\deg(v_n) = \deg(v_j) = 2$ for some j with $1 < j < n$; see v_{10} and v_6 in Figure 7.11a. We call the path $\mathcal{P}_\mathrm{u} = (V_\mathrm{u}, E_\mathrm{u}) = \langle v_1, v_2, \ldots, v_{j-1} \rangle$ the *upper path* of $\mathcal{Z}'$, and $\mathcal{P}_\mathrm{l} = (V_\mathrm{l}, E_\mathrm{l}) = \langle v_j, v_{j+1}, \ldots, v_n \rangle$ the *lower path* of $\mathcal{Z}'$. Observe that $V = V_\mathrm{u} \cup V_\mathrm{l}$. If we remove $E_\mathrm{u} \cup E_\mathrm{l}$ from $E_{\mathcal{Z}'}$, then the resulting graph is a caterpillar $\mathcal{C}$ that spans V and whose spine alternates between vertices of V_u and V_l.

We first compute the left-to-right order of the vertices of caterpillar $\mathcal{C} = (V_\mathcal{C}, E_\mathcal{C})$ as described by the algorithm supporting Theorem 7.4. Then, the x-coordinate of the ith vertex in this order is $3i - 2$, $i = 1, 2, \ldots, n$; see Figure 7.11b.

In order to compute the y-coordinates of the vertices, we first partition $\mathcal{M}$ into three matchings $\mathcal{M}_{\mathrm{ll}} = (V_{\mathrm{ll}}, E_{\mathrm{ll}})$, $\mathcal{M}_{\mathrm{uu}} = (V_{\mathrm{uu}}, E_{\mathrm{uu}})$ and $\mathcal{M}_{\mathrm{ul}} = (V_{\mathrm{ul}}, E_{\mathrm{ul}})$ as follows. Let $(v, v') \in E_{\mathcal{M}}$. Then,

   (i) $(v, v') \in E_{\mathrm{ll}}$ if $v, v' \in V_{\mathrm{l}}$,

   (ii) $(v, v') \in E_{\mathrm{uu}}$ if $v, v' \in V_{\mathrm{u}}$,

   (iii) $(v, v') \in E_{\mathrm{ul}}$ if $v \in V_{\mathrm{u}}$ and $v' \in V_{\mathrm{l}}$.

Since $V = V_{\mathrm{u}} \cup V_{\mathrm{l}}$, it holds that $E_{\mathcal{M}} = E_{\mathrm{ll}} \cup E_{\mathrm{uu}} \cup E_{\mathrm{ul}}$. In the resulting layout, the edges in E_{ll} will be drawn below the edges in E_{ul}, which in turn will be drawn below the ones of E_{uu}; see Figure 7.11b. Thus, they will not cross each other.

Let $m_{\mathrm{ll}} = |E_{\mathrm{ll}}|$ and $E_{\mathrm{ll}} = \{e_1, \ldots, e_{m_{\mathrm{ll}}}\}$. We draw the edges from bottom to top, starting from row 1. For $i = 1, \ldots, m_{\mathrm{ll}}$, let $e_i = (u_i, u_i')$ with $x(u_i) < x(u_i')$. We set $y(u_i) = 4i - 1$ and $y(u_i') = 4i - 3$. Edge e_i is drawn with a bend at $(x(u_i), y(u_i'))$. Our approach ensures that there are no crossings between edges of $\mathcal{M}_{\mathrm{ll}}$, as they are drawn in different horizontal strips of the drawing. Similarly, we draw the edges in E_{uu} from top to bottom, starting from row $2n - 1$. Let $m_{\mathrm{uu}} = |E_{\mathrm{uu}}|$. By construction, the topmost vertex of V_{ll} is drawn in the row $4m_{\mathrm{ll}} - 1$ and the bottommost vertex of V_{uu} is drawn in the row $2n + 1 - 4m_{\mathrm{uu}}$. The vertices of V_{ul} will be drawn between these rows; see Figure 7.11b.

In order to draw the edges in E_{ul}, we process the vertices of the set V_{ul} from left to right and assign y-coordinates to both endpoints of the incident matching edge. Let $m_{\mathrm{ul}} = |E_{\mathrm{ul}}|$ and $E_{\mathrm{ul}} = \{\varepsilon_1, \ldots, \varepsilon_{m_{\mathrm{ul}}}\}$. For $k \in \{1, \ldots, \mu\}$, let $\varepsilon_k = (w_k, w_k')$ and assume without loss of generality that $x(w_k) < x(w_k')$ and $x(w_1) < \ldots < x(w_\mu)$. We place the vertices in V_{l} from bottom to top and the vertices in $V_{\mathrm{u}} \cap V_{\mathrm{ul}}$ from top to bottom. If $w_k \in V_{\mathrm{u}}$, we assign the y-coordinate $4m_{\mathrm{ll}} - 1 + 3k$ to w_k and the y-coordinate $2n + 1 - 4m_{\mathrm{uu}}$ to w_k'; if $w_k \in V_{\mathrm{l}}$, we switch the y-coordinates. Edge ε_k is drawn with a bend at $(x(w_k), y(w_k'))$. Further, every edge $\varepsilon_l \in E_{\mathrm{ul}}$ with $l > k$ has its endpoints to the right of w_k and in the horizontal strip defined by the lines $y = y(w_k)$ and $y = y(w_k')$. Hence, it will not be involved in crossings with (w_k, w_k'). This guarantees that the drawing of $\mathcal{M}$ is planar.

It remains to determine, for each edge $e = (v, v') \in \mathcal{Z}'$, where to place its bend. Without loss of generality, let e be directed from its left endpoint, say v, to its right endpoint, say v'. First, assume that $e \in E_{\mathrm{u}} \cup E_{\mathrm{l}}$ belongs to the outercycle. Then, we place its bends at $(x(v') - 2, y(v))$ and $(x(v') - 2, y(v') - \mathrm{sgn}(y(v') - y(v)))$. Second, assume that $e \in E_{\mathcal{C}}$ belongs to the inner caterpillar. Then, we place its bends at $(x(v') - 1, y(v) + \mathrm{sgn}(y(v) - y(v')))$ and $(x(v') - 1, y(v') - \mathrm{sgn}(y(v') - y(v)))$.

Since $\mathcal{P}_{\mathrm{u}}$ and $\mathcal{P}_{\mathrm{l}}$ are drawn x-monotone, both are drawn planar. Following similar arguments as in the proof of Theorem 7.4, we can show that $\mathcal{C}$ is drawn planar as well. Since $\mathcal{C}$ is drawn between $\mathcal{P}_{\mathrm{u}}$ and $\mathcal{P}_{\mathrm{l}}$, it follows that $\mathcal{Z}'$ is drawn planar, as desired. It now remains to prove that all (potential) crossings between $\mathcal{Z}'$ and $\mathcal{M}$ only involve rectilinear edge segments of $\mathcal{Z}'$, as $\mathcal{M}$ consists exclusively of rectilinear segments. As all slanted segments of $\mathcal{Z}'$ are of y-length 1, no horizontal segment of $\mathcal{M}$ can cross them. The same holds for vertical segments of $\mathcal{M}_{\mathrm{uu}} \cup \mathcal{M}_{\mathrm{ll}}$, as they are drawn above and below $\mathcal{P}_{\mathrm{l}}$ and $\mathcal{P}_{\mathrm{u}}$, respectively. The only possible non-rectilinear crossings are between a vertical segment of a matching edge $(u, v) \in E_{\mathrm{ul}}$ and a long slanted segment of $\mathcal{C}$ incident to a spine vertex w. This crossing can only occur if w

lies to the left of the vertical segment of (u, v). By construction, w is never drawn between u and v, with respect to the y-coordinate. Thus, such crossings cannot occur, which implies all crossings between $\mathcal{Z}'$ and $\mathcal{M}$ form right angles.

We now show that our algorithm supports the Sefe model. Assume that $\mathcal{M}$ and $\mathcal{Z}$ share edges, and let $e = (v, v')$ be an edge that belongs to both input graphs. If $e \in E_l$, then also $e \in E_{ll}$. Our algorithm places v and v' at consecutive y-coordinates and determines their x-coordinates such that no other vertex of V_l lies between them. Thus, edge e is drawn in the matching as a vertical segment of length 2 and a horizontal segment that has no vertex below it. Hence, the drawing of e in the matching is planar and can be used for both graphs. The case $e \in E_u$ is analogous.

If $e \in E_{\mathcal{C}}$, then also $e \in E_{ul}$. In this case, we observe that the topological routing of e is the same in both graphs with an offset of one unit to avoid overlaps. Thus, every other edge either crosses both or none of the drawings of e. Since each drawing of e forbids crossings by edges of one of the input graphs, actually none of the two drawings can have a crossing. Hence, we can draw e the same way in both graphs.

By the choice of the coordinates, the area requirement of our algorithm is $(3n-2) \times (2n-1)$. Since we have to sort the edges in E_{ul} by the x-coordinates of the incident vertices, our algorithm runs in $O(n \log n)$ time. To complete the proof of this theorem, observe that the extra edges that we introduced when augmenting $\mathcal{Z}$ to $\mathcal{Z}'$ can be safely removed from the constructed layout, affecting neither the crossing angles nor the area of the layout. $\square$

7.4 Conclusions and Open Problems

In this chapter, we have studied RacSefe and RacSim drawings with few bends per edge. We have proven that two planar graphs always admit a RacSim drawing with at most six bends per edge. For more restricted classes of graphs, we have reduced the number of bends per edge. Some of these specialized results also hold for the stronger RacSefe model. All drawings of our algorithms fit it quadratic area.

Our results raise several questions that remain open. First of all, can we strengthen any of our results from RacSim to RacSefe (see Table 7.1)? Are there other graph classes that admit RacSim drawings with few bends? Can we reduce the number of bends per edge by relaxing the strict constraint that edge intersections are at right angles and instead ask for drawings that have close to optimal crossing resolution? What about the computational complexity of the general problem, that is, given two or more planar graphs on the same set of vertices and a non-negative integer k, can we decide efficiently whether there is a RacSim drawing in which each graph is drawn with at most k bends per edge and the crossings are all at right angles? This seems unlikely. Finally, is it possible to achieve sub-quadratic area for RacSim drawings of subclasses of planar graphs when accepting more bends per edge?

8 Recognizing and Drawing IC-Planar Graphs

The study of graphs that are, in some sense, "nearly-planar", is an emerging topic in graph theory, graph algorithms, and network visualization. The general framework is to relax the planarity constraint by allowing edge crossings but still forbidding those configurations that would affect the readability of the drawing too much. Different types of forbidden edge-crossing configurations give rise to different families of nearly-planar graphs. For example, if the number of crossings per edge is bounded by a constant k, we have the family of *k-planar graphs* (see, for example, [ABK13, HELP12, PT97, Rin65, Tho88]). The *k-quasi-planar graphs* admit drawings with no k pairwise crossing edges (see, for example, [DDLM12, FPS13]). *RAC (Right Angle Crossing) graphs* can be drawn such that edges cross only at right angles (see, for example, [DEL11, EL13]). Generalizations of RAC drawings are ACE_α and ACL_α drawings, where the edges can cross only at an angle that is *exactly α* or *at least α*, respectively, where $\alpha \in (0, \pi/2]$; see [DL12] for a survey. Further families of nearly-planar graphs are *fan-crossing free graphs* [CHKK13] and *fan-planar graphs* [BCG+14, BDD+14, KU14]. Most of the existing literature on nearly-planar graphs can be classified according to the study of the following problems (see also [Lio14] for additional references).

Previous work.   Previous work on related graph classes can be categorized into different types of problems.

Coloring Problem: While the chromatic number of planar graphs is four, it is rather natural to ask what restrictions on the crossing configurations force the chromatic number of a graph to be relatively small. For example, Borodin [Bor84] proved that the chromatic number of 1-planar graphs is six. See also Borodin et al. [BKRS01] for more papers that study the coloring problems of nearly-planar graphs.

Turán-type Problem: The question here is to determine how many edges a nearly-planar graph can have. In particular, it is known that all the families of nearly-planar graphs mentioned above are rather sparse (see, for example, [Ack09, AT07, AAP+97, BEG+12, Did13, KU14, PT97, Val98, VBSW83]).

Recognition Problem: In contrast to testing for planarity, recognizing a nearly-planar graph has often been proved to be NP-hard. This is, for example, the case for 1-planar graphs [KM13], RAC graphs [ABS12], and fan-planar graphs [BCG+14, BDD+14]. For some constrained classes of nearly-planar graphs, polynomial-time tests exist (for example, [ABB+13, EHK+13, HEK+14]).

Drawing Algorithms: Some recent papers describe drawing algorithms for different families of nearly-planar graphs; the majority of them focuses on drawings with straight-line edges and often considers the interplay with other readability constraints, such as compact area. A limited list of examples includes [ABK13, DDEL14, DDLM13].
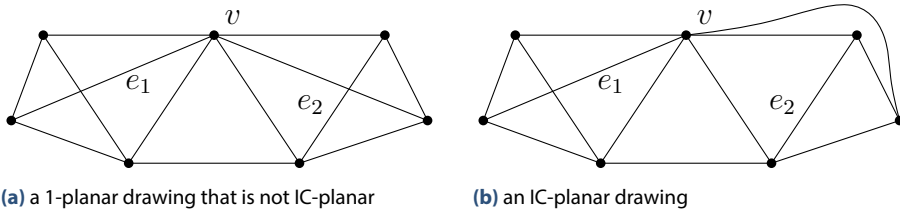
(a) a 1-planar drawing that is not IC-planar   **(b)** an IC-planar drawing

Figure 8.1: Two drawings of the same graph

Inclusion/intersection relationships: The relationships between different classes of nearly-planar graphs are also proved, as a fundamental step towards developing a comprehensive theory of graph drawing beyond planarity (see, for example, [BDD⁺14, EL13]), or between nearly-planar graphs and graphs that admit specific types of visual representations [Bra14, EKL⁺13].

This chapter studies *IC-planar graphs*, which stands for Independent Crossings graphs, i.e., graphs that admit a drawing where no two crossed edges share an end-vertex and each edge is crossed at most once. See Figure 8.1 for an example. For instance, the drawing of Figure 8.1a is not IC-planar because the two edges e_1 and e_2 are both crossed and they share vertex v; the drawing of Figure 8.1b is IC-planar. In other words, IC-planar graphs are 1-planar graphs with the additional property that all edge crossings are independent from one another. Král and Stacho [KS10] exploited the independence of the edge crossings to show that IC-planar graphs have chromatic number at most five. Albertson [Alb08] has showm that IC-planar graphs have chromatic number at most six; the chromatic number of IC-planar graphs was then proved to be at most five by Zhang and Liu [ZL13]. Zhang and Liu also study the Turán-type problem and prove that IC-planar graphs have at most $13n/4 - 6$ edges, which is a tight bound. Zhang [Zha14] studies so-called *plane graphs with near independent crossings* (*NIC-planar graphs*), that is, each pair of crossing edges shares at most one endpoint, and states the computational complexity of recognizing IC-planar graphs as an open problem.

Our contribution.   We extend the theory on IC-planarity beyond the already studied coloring and Turán-type problems. We investigate drawing algorithms, the complexity of the recognition problem, and the interplay between IC-planar graphs and other families of nearly-planar graphs. Our results are as follows.

(i) We present an $O(n)$-time algorithm that computes a straight-line drawing of an IC-planar graph with n vertices in $O(n^2)$ area, which is worst-case optimal (Theorem 8.1). It may be worth recalling that not all 1-planar graphs admit a straight-line drawing and that there are embedded 1-planar graphs that require $\Omega(2^n)$ area [HELP12].

(ii) We prove that IC-planarity testing is NP-complete both in the variable embedding setting (Theorem 8.2) and when the rotation system of the graph is fixed (Theorem 8.3). Note that 1-planarity testing is already known to be NP-complete in general [KM13], even if the rotation system is fixed [ABGR15]. In addition to the hardness result, we present a polynomial-time algorithm that tests whether a set of matching edges can
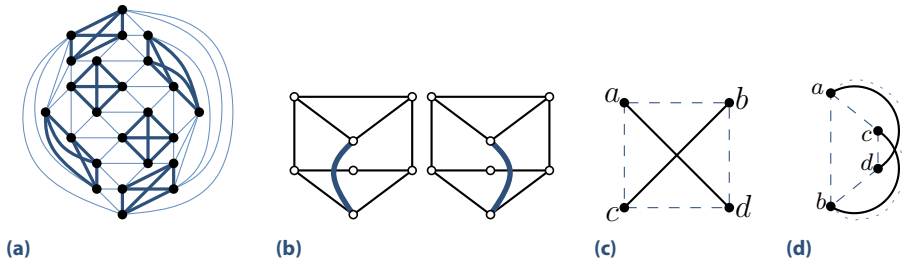
Figure 8.2: (a) An IC-planar drawing; (b) two different IC-planar embeddings of the same graph with the same rotation system; (c) an X-configuration; (d) a B-configuration.

be added to a triangulated plane graph such that the resulting graph is IC-planar (Theorem 8.4). We remark that in any IC-planar drawing the set of crossing edges form a matching.

(iii) We study the interplay between IC-planar graphs and RAC graphs. Namely, we show that every IC-planar graph is a RAC graph (Theorem 8.5), which sheds new light on an open problem about the relationship between 1-planar graphs and RAC graphs [EL13]. We also prove that a straight-line RAC drawing of an IC-planar graph may require $\Omega(q^n)$ area, for a suitable constant $q > 1$ (Theorem 8.6).

Notation.   We consider simple undirected graphs G. Recall that a drawing Γ of G is *planar* if no edges cross, and *1-planar* if each edge is crossed at most once. Γ is *IC-planar* if it is 1-planar and there are no crossing edges that share a vertex (see Figure 8.2a). Further, recall that a *rotation system* $\mathcal{R}(G)$ of a graph G describes a possible cyclic ordering of the edges around the vertices. $\mathcal{R}(G)$ is planar (1-planar, IC-planar) if G admits a planar (1-planar, IC-planar) embedding that preserves $\mathcal{R}(G)$. Observe that $\mathcal{R}(G)$ can directly be retrieved from a drawing or an embedding. The converse does not necessarily hold, as shown in Figure 8.2b.

A *kite K* is a graph isomorphic to K_4 with an embedding such that all the vertices are on the boundary of the outer face, the four edges on the boundary are planar, and the remaining two edges cross each other; see Figure 8.2c. Thomassen [Tho88] characterized the possible crossing configurations that occur in a 1-planar drawing. This characterization applied to IC-planar drawings gives rise to the following property, where an X-crossing is of the type described in Figure 8.2c, and a B-crossing is of the type described in Figure 8.2d (the bolds edges only).

Property 8.1. *Every crossing of an IC-planar drawing is either an X- or a B-crossing.*

8.1 Straight-Line Drawings of IC-planar graphs

We show that every IC-planar graph admits an IC-planar straight-line grid drawing in quadratic area, and this area is worst-case optimal (Theorem 8.1). The result is based on first using a new technique that possibly augments the input graph to a maximal IC-plane

graph (the resulting embedding might be different from the original one) with specific proper-
ties (Lemma 8.1), and then on suitably applying on the augmented graph a drawing algorithm
by Alam et al. for triconnected 1-plane graphs [ABK13] on the augmented graph. We say
that a kite (a, b, c, d) with crossing edges (a, d) and (b, c) is *empty* if it contains no other
vertices, that is, the edges (a, c), (a, d), and (a, b) are consecutive in the counterclockwise
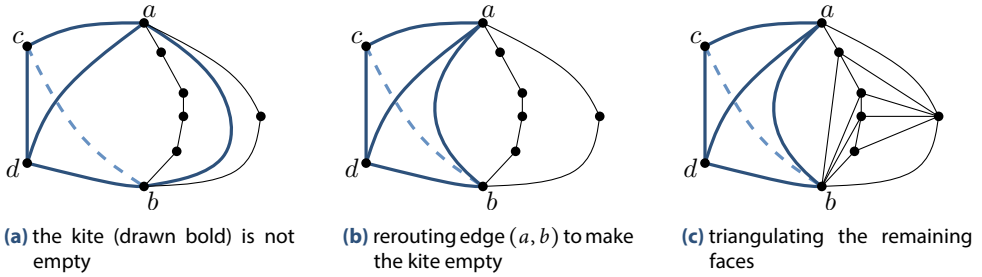order around a; see Figure 8.3b. The condition for the edges around b, c, and d is analogous.

Lemma 8.1. *Let $G = (V, E)$ be an IC-plane graph with n vertices. There exists an $O(n)$-time
algorithm that computes a planar-maximal IC-plane graph $G^+ = (V, E^+)$ with $E \subseteq E^+$ such
that the following conditions hold:*

  (c1) The four endvertices of each pair of crossing edges induce a kite.

  (c2) Each kite is empty.

  *(c3) Let C be the set of crossing edges in G^+. Let $C^* \subset C$ be a subset containing exactly one
        edge for each pair of crossing edges. Then $G^+ \smallsetminus C^*$ is plane and triangulated.*

  (c4) The outer face of G^+ is a 3-cycle of non-crossed edges.

Proof. Let G be an IC-plane graph; we augment G by adding edges such that for each pair of
crossing edges (a, d) and (b, c) the subgraph induced by vertices $\{a, b, c, d\}$ is isomorphic
to K_4; see the dashed edges in Figures 8.2c and 8.2d. Next, we want to make sure that this
subgraph forms an X-configuration and the resulting kite is empty. Since G is IC-planar,
it has no two B-configurations sharing an edge. Thus, we remove a B-configuration with
vertices $\{a, b, c, d\}$ by rerouting the edge (a, b) to follow the edge (a, d) from vertex a until
the crossing point, then edge (b, c) until vertex b, as shown by the dotted edge in Figure 8.2d.
This is always possible, because edges (a, c) and (b, d) only cross each other; hence, following
their curves, we do not introduce any new crossing. The resulting IC-plane graph satisfies
condition (c1) (recall that, by Property 8.1, only X- and B-configurations are possible). Now,
assume that a kite $\{a, b, c, d\}$ is not empty; see Figure 8.3a. Following the same argument as
above, we can reroute the edges (a, b), (b, d), (c, d) and (a, d) to follow the crossing edges
(a, d) and (b, c); see Figure 8.3b. The resulting IC-plane graph is denoted by G' and satisfies
condition (c2).

   We now augment G' to G^+, such that condition (c3) is satisfied. Let C be the set of all
pairs of crossing edges in G'. Let C^* be a subset constructed from C by keeping only one
(arbitrary) edge for each pair of crossing edges. The graph $G' \smallsetminus C^*$ is clearly plane. To
ensure condition (c3), graph $G^+ \smallsetminus C^*$ must be plane and triangulated. Because G' satisfies
condition (c2), each removed edge spans two triangular faces in $G' \smallsetminus C^*$. Thus, no face
incident to a crossing edge has to be triangulated. We internally triangulate the other faces by
picking any vertex on its boundary and connecting it to all other vertices (avoiding multiple
edges) of the boundary; see e.g. Figure 8.3c. Graph G^+ is then obtained by reinserting the
edges in C^* and satisfies condition (c3). To satisfy condition (c4), notice that G^+ is IC-plane,
hence, it has a face f whose boundary contains only non-crossed edges. Also, f is a 3-cycle by
construction. Thus, we can re-embed G^+ such that f is the outer face.

   It remains to prove that the described algorithm runs in $O(n)$ time. Namely, let m be the
number of edges of G, augmenting the graph such that for each pair of crossing edges their

(a) the kite (drawn bold) is not empty

(b) rerouting edge (a, b) to make the kite empty

(c) triangulating the remaining faces

Figure 8.3: Illustration for the proof of Lemma 8.1

end-vertices induce a subgraph isomorphic to K_4 can be done in $O(m)$ time (the number of added edges is $O(n)$). Similarly, rerouting some edges to remove all B-configurations requires $O(m)$ time. Also, triangulating graph $G' \smallsetminus C^*$ can be done in time proportional to the number of faces of $G' \smallsetminus C^*$, which is $O(n + m)$. Since IC-planar graphs are sparse [ZL13], the time complexity follows. □

Theorem 8.1. *There is an $O(n)$-time algorithm that takes an IC-plane graph G with n vertices as input and constructs an IC-planar straight-line grid drawing of G in $O(n) \times O(n)$ area. This area is worst-case optimal.*

Proof. Augment G into a planar-maximal IC-plane graph G^+ in $O(n)$ time using Lemma 8.1. Graph G^+ is triconnected, as it contains a triangulated plane subgraph. Draw G^+ with the algorithm by Alam et al. [ABK13] which takes as input a 1-plane triconnected graph with n vertices and computes a 1-planar drawing on the $(2n - 2) \times (2n - 3)$ grid in $O(n)$ time; this drawing is straight-line, but for the outer face, which may contain a bent edge if it has two crossing edges. Since by Lemma 8.1 the outer face of G^+ has no crossed edges, Γ is straight-line and IC-planar. Dummy edges are then removed from Γ.

It remains to prove that the area bound of the algorithm is worst-case optimal. To this aim, we show that for every $n \geq 2$ there exists an IC-planar graph G with $\Theta(n)$ vertices, such that every IC-planar straight-line grid drawing of G requires $\Omega(n^2)$ area. Dolev et al. [DLT84] described an infinite family of planar graphs, called nested triangle graphs, such that every planar straight-line drawing of an n-vertex graph G (for $n \geq 6$) of this family requires $\Omega(n^2)$ area. We augment G as follows. For every edge (u, v) of G, we add a vertex c_{uv}, and two edges (u, c_{uv}) and (c_{uv}, v). Denote by G^+ the resulting augmented graph, which clearly has $\Theta(n)$ vertices. We now show that in every possible IC-planar straight-line drawing of G^+ there are no two edges of G that cross each other. Suppose, by contradiction, that two edges (u, v) and (w, z) of G cross each other in an IC-planar straight-line drawing Γ of G^+. Observe that the subgraph induced by the vertices u, v, c_{uv} is a 3-cycle. Clearly, either w or z lies inside this 3-cycle, while the other one lies outside, as otherwise (u, v) would be crossed twice (or it would not be crossed). Thus, it immediately follows that the IC-planarity condition cannot be respected, since either an edge of the 3-cycle is crossed twice or two edges are crossed once, violating the IC-planarity condition. Hence, the subgraph G must be drawn planar and this implies that Γ requires $\Omega(n^2)$ area. □

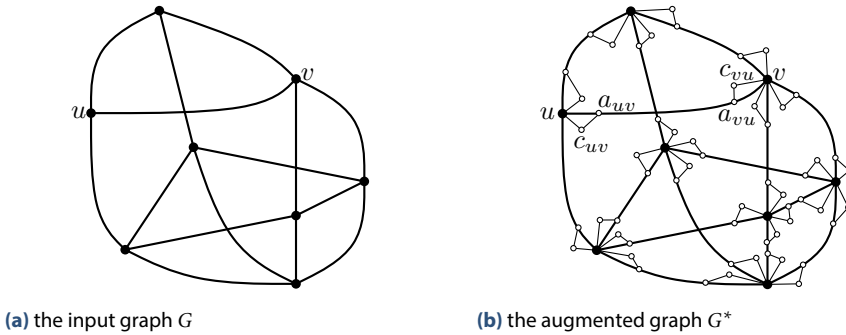**(a)** the input graph G **(b)** the augmented graph G^*

Figure 8.4: Illustration of the proof of Theorem 8.2.

8.2 Recognizing IC-planar graphs

The *IC-planarity testing* problem asks whether a given graph G admits an IC-planar embedding. We first show that this problem is NP-complete in the general case and even if the rotation system is given. Then, we give an $O(n^3)$-time algorithm that decides whether the union of a triangulated plane graph and a set of crossing edges is IC-planar and, if so, draws the graph.

8.2.1 Hardness of the Problem.

In the following, we give a simple reduction from the NP-hard problem *1-planarity testing* to IC-planarity testing, which shows that our problem is NP-complete.

Theorem 8.2. *IC-planarity testing is* NP-*complete.*

Proof. IC-planarity is in NP, as one can guess an embedding and check whether it is IC-planar [GJ83]. For the hardness proof, the reduction is from the 1-planarity testing problem, which asks whether a given graph is 1-planar or not. The reduction uses a 3-cycle gadget and exploits the fact that at most one edge of a 3-cycle is crossed in an IC-planar drawing. We transform an instance G of 1-planarity testing into an instance G^* of IC-planarity testing, by replacing each edge (u, v) of G with a graph G_{uv} consisting of two 3-cycles, T_{uv} and T_{vu}, with vertices $\{u, c_{uv}, a_{uv}\}$ and $\{v, c_{vu}, a_{vu}\}$, respectively, plus edge (a_{uv}, a_{vu}), called the *attaching edge* of u and v; see Figure 8.4.

Let Γ be a 1-planar drawing of G. An IC-planar drawing Γ^* of G^* can be easily constructed by replacing each curve representing an edge (u, v) in Γ with a drawing of G_{uv} where T_{uv} and T_{vu} are drawn planar and sufficiently small, such that the possible crossing that occurs on the edge (u, v) in Γ occurs on the attaching edge (a_{uv}, a_{vu}) in Γ^*. Hence, since all the attaching edges are independent, Γ^* is IC-planar.

Let Γ^* be an IC-planar drawing of G^*. We show that it is possible to transform the drawing in such a way that all crossings occur only between attaching edges. Once this condition is satisfied, in order to construct a 1-planar drawing Γ of G, it suffices to remove, for each edge (u, v), the vertices c_{uv} and c_{vu}, and to replace a_{uv} and a_{vu} with a bend point. Namely, as already observed, no more than one edge can be crossed for every gadget T_{uv} of G^*. Suppose

(a)   **(b)**
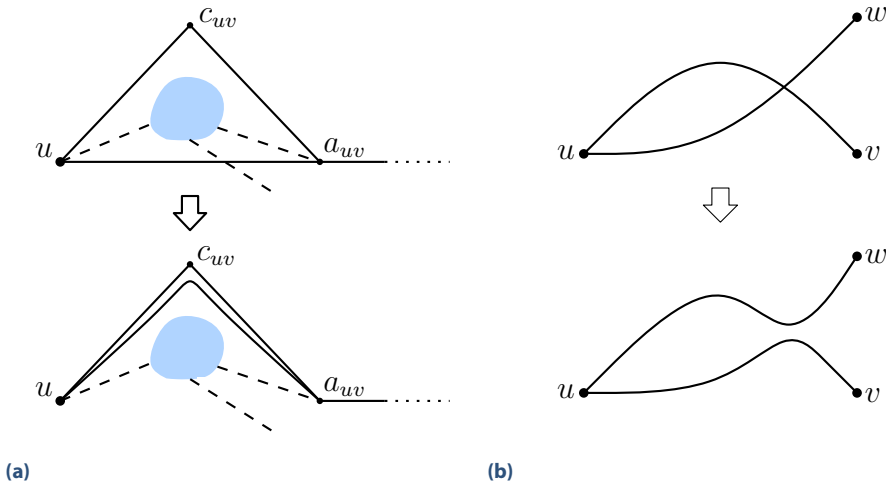
Figure 8.5: Illustration of the proof of Theorem 8.2. (a) Rerouting the crossed edge (u, a_{uv}) via c_{uv} to be planar. (b) Rerouting the crossing edges (u, v) and (u, w) to be planar.

now that the edge (u, a_{uv}) of T_{uv} is crossed. Since the other two edges of T_{uv} are not crossed, we can reroute (u, a_{uv}) such that it follows the curves that represent (u, c_{uv}) and (c_{uv}, a_{uv}); see Figure 8.5a.

In order to complete the proof, we need to take care of the following particular case. Two attaching edges a_{uv} and a_{uw} that cross and that are connected to two gadgets T_{uv} and T_{uw} with a common vertex u represent a valid configuration in Γ^*, while they give rise to a crossing between two adjacent edges in Γ, which is not allowed since Γ must be a simple drawing. However, this case can be easily solved by rerouting the two edges in Γ as shown in Figure 8.5b. $\qquad\square$

8.2.2 Hardness of the Problem With a Given Rotation System.

Note that the construction given above does not work for IC-planarity testing with a given rotation system since the rerouting step changes the rotation system. However, we now prove NP-hardness of IC-planarity testing for graphs with a given rotation system. We rely on the *membrane technique* introduced by Auer et al. [ABGR15] to prove the NP-hardness of 1-planarity testing for graphs with a given rotation system. In particular, we design particular gadgets that make it possible to use the membrane technique in the case of IC-planar graphs.

First, we replace the U-graphs [ABGR15] by M-graphs, called *mesh graphs*. These graphs have a unique embedding with a fixed rotation system. Namely, an M-graph is a mesh, where cells are filled with two crossing edges, following a checkerboard pattern to ensure independent crossings; see Figure 8.6. To see that with a given rotation system, an M-graph has a unique embedding, observe that each subgraph isomorphic to K_4 can be embedded planarly or as a kite, and this is determined by the rotation system [Kyn09]. The rotation system defined by

(a)   **(b)**
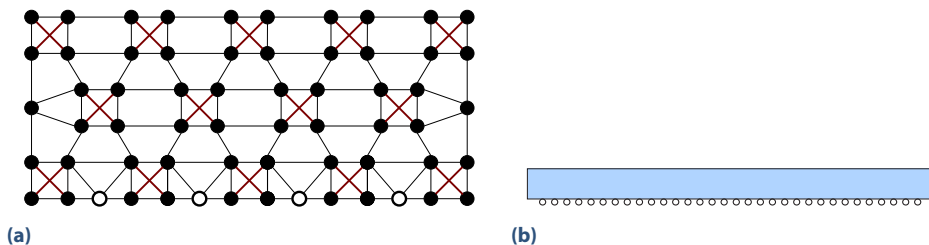
Figure 8.6: (a) The structure of an M-graph and (b) and its abbreviation.

the drawing in Figure 8.6 implies that each subgraph isomorphic to K_4 is embedded as a kite, and therefore the embedding of an M-graph is unique.

Let M be an M-graph with a given fixed embedding. At its bottom line, M has sufficiently many *free* vertices which are not incident with a crossing edge in M. These vertices are consecutively ordered, say from left to right. The edges on the bottom line are not crossed in any IC-planar embedding of M, so they are crossing-free in the given embedding. Finally, M cannot be crossed by any path from a free vertex. In what follows, we attach further gadgets to M by connecting these gadgets to consecutive free vertices. If there are several gadgets, then they are separated and placed next to each other.

General Construction.   Consider an instance α of Planar-3SAT with its corresponding plane graph G and its dual G^*. Recall that the vertices of G represent variables x and clauses c, also, there is an edge (x, c) if x or its negation occurs as a literal in c; see Figure 8.7a. We transform G^* into an *M-supergraph* G_α^* as follows.

Each vertex of G^*, corresponding to a face of the embedded graph G, is replaced by a sufficiently large M-graph. Further, each edge of G^* is replaced by a *barrier* of l parallel edges from/to l free consecutive vertices on the boundary of the M-graph. These edges will be crossed by paths that are called *ropes*. The size of l will be determined later. The M-graph must have sufficiently many free vertices for the edges from the gadgets and barriers. The smallest bound can easily be computed from the embedding of G and the attached gadgets; see Figure 8.7b.

For each variable x, we construct a *V-gadget* $\gamma(x)$, and similarly we build a *C-gadget* for each clause. These gadgets are described below.

Each vertex u of G lies in a face f^* of G, which corresponds to a vertex of G^*. We attach the gadget $\gamma(u)$ of u to a M-graph of a vertex of G^* on the boundary of f^* such that $\gamma(u)$ lies in f^*. To that end, it does not matter which vertex of f^* is chosen. Similarly, each edge e of G between a variable and a clause is replaced by a rope of length $2l + 3$. Since e is crossed by its dual edge, the rope is crossed by a barrier. A rope acts as a communication line that "passes" a crossing at a V-gadget across a barrier to a C-gadget. We denote by a *membrane* (similarly as in [ABGR15]) a path between free vertices on the boundary of a M-graph, or between particular vertices of a variable gadget. We call a vertex IN if it placed inside the region of a membrane and the boundary of the M-graph in an IC-drawing, and OUT otherwise. IN and OUT are exactly defined by edges which cross the membrane. Note that the framework is basically a simultaneous embedding of G and G^* by means of our gadgets, M-graphs, barriers
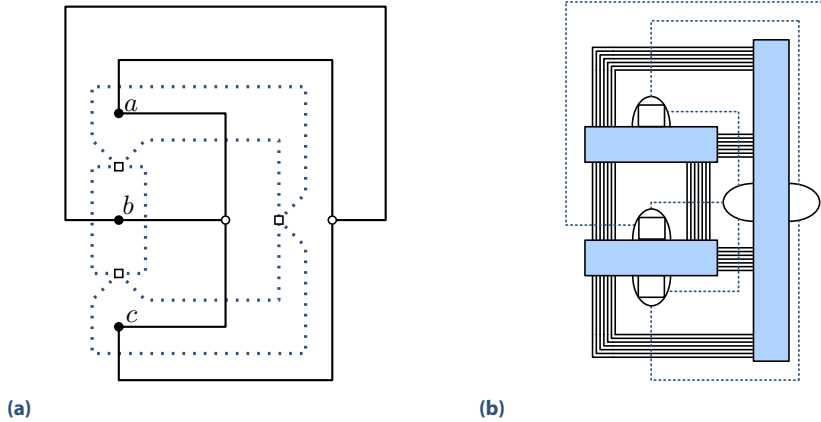
Figure 8.7: (a) The planar graph G (solid) and its dual G^* (dotted) corresponding to the Planar-3SAT formula $(a \vee b \vee \neg c) \wedge (a \vee \neg b \vee c)$; (b) the corresponding M-supergraph G^*_α with the clause gadgets (vertical) and the variable gadgets (horizontal)

and ropes. The subgraph without V- and C-gadgets is 3-connected, since the M-graphs are 3-connected and the barriers have size l for $l \geq 3$, and it has a unique planar embedding if one edge from each pair of crossing edges in each M-graph is removed.

Construction of the C-gadgets.   The C-gadget $c = (l_1, l_2, l_3)$ with three literals l_1, l_2 and l_3 is attached to eight consecutive free boundary vertices of an M-graph M, say $v_1, \dots, v_8$. For each literal l_i, there are three vertices u_i, a_i and b_i, and four edges (u_i, a_i), (u_i, b_i), (a_i, v_{2i}) and (b_i, v_{2i+1}), where u_i is the initial vertex of the rope towards the corresponding variable gadget. There is a membrane of nine edges from v_1 to v_8, see Figure 8.8a.

By construction, at most two vertices among u_1, u_2 and u_3 can be moved outside the membrane, and at least one initial vertex of a rope (and maybe all) must be IN. IN shall correspond to the value true of the literal and thus of the clause.

Construction of the V-gadgets.   Let x be a variable and let v be the vertex corresponding to x in G. Suppose that the literal x occurs in k clauses for some $k \geq 1$, which are ordered by the embedding of G. Denote this sequence by $x_1, \dots x_k$, where each x_i corresponds to x or $\neg x$. The V-gadget of x is $\gamma(x) = \gamma(t_l), \gamma(x_1), \dots, \gamma(x_k), \gamma(t_r)$. This gadget is connected to $7k + 14$ free consecutive vertices on the border of the M-graph M to which it is attached; see Figure 8.8b for an illustration. The gadgets $\gamma(t_l)$ and $\gamma(t_k)$ are called the (left and right) *terminal gadgets* and each $\gamma(x_i)$ is called a *literal gadget*. They are similar to clause gadgets and are each connected to seven consecutive free variables $v_1, \dots, v_7$ on the boundary of M.

The terminal gadget $\gamma(t_l)$ has two primary vertices x_0^+ and x_0^-. The primary vertex x_0^+ is connected to v_2, v_3 and v_4, and the other primary vertex x_0^- is connected to v_5, v_6 by paths of length two, respectively. Analogously, the terminal gadget has two primary vertices x_{k+1}^+ and x_{k+1}^-, with the same requirements.. There is a local membrane of seven edges from v_1 to v_7. The gadget $\gamma(x)$ has an outer membrane of length $2k + 1$ from x_0^+ to x_{k+1}^-.
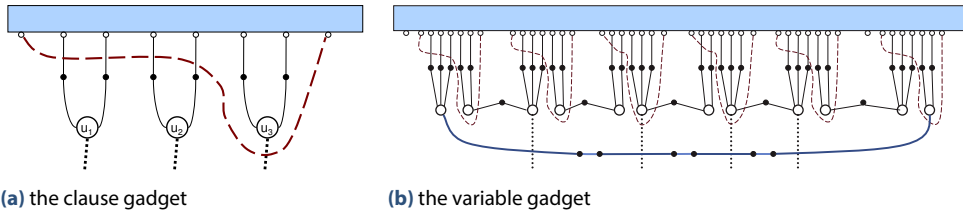
(a) the clause gadget   **(b)** the variable gadget

Figure 8.8: The two gadgets for the NP-hardness reduction

For each literal gadget $\gamma(x_i)$, consider two cases. If x_i is positive, then $\gamma(x_i)$ has two primary vertices x_i^+ and x_i^-, where x_i^+ is attached to three free vertices v_2, v_3 and v_4 of M, and x_i^- is attached to two free vertices v_5 and v_6 by two paths of length two, respectively. The rope to the literal begins at x_i^+. Otherwise, if x_i is negated, then the gadget is reflected, such that x_i^+ has two, and x_i^- has three paths of length two to the M-graph. The rope to the literal begins at x_i^+. In both cases, there is a local membrane of length seven from v_1 to v_7. The rope is a path of $2l + 3$ edges from vertex $x_i^\pm$ of the V-gadget to the vertex u_i of the clause gadget.

The rotation system of the gadgets it retrieved from the drawing and the ordering of the vertices on the border of M-graphs.

Correctness.   We will now prove several lemmas on the structure of our construction. With these lemmas, we will show that an IC-planar drawing to the resulting graph G_α immediately yields a valid solution to the underlying Planar-3SAT problem. First, we show that the M-graph is not crossed.

Lemma 8.2. *The boundary edges of a M-graph (with a fixed rotation system) are never crossed in an IC-planar drawing of G_α.*

Proof. This lemma follows directly from the construction. Each K_4 must be embedded as a kite, and further edge crossings violate IC-planarity. □

Consequently, the following corollary holds.

Corollary 8.1. *A path from a free boundary vertex cannot cross any M-graph.*

Now, we show that every clause, terminal and literal gadget has at least one primary vertex that is not OUT.

Lemma 8.3. *In every IC-planar drawing of G_α, at most two of the primary vertices u_1, u_2 and u_3 of a clause gadget can be OUT, and at most one of the primary vertices x_i^+, x_i^- of a terminal or a literal gadget can be OUT of the local membrane.*

Proof. Assume that u_1, u_2 and u_3 all are OUT. Then, the membrane must cross five edges. Thus, the membrane has a length of at least nine. However, from the construction, the membrane only has length seven, which is a contradiction. The proof for terminal and literal gadgets works analogously. □

Next, we show that the outer membrane crosses each rope.

Lemma 8.4. *In every IC-planar drawing of G_α, each rope is crossed by the outer membrane of the variable gadget.*

Proof. In order to avoid a crossing, the outer membrane must be drawn around the C-gadget. Then, the membrane must cross at least one barrier, which is impossible in an IC-planar drawing if the size of a barrier is chosen to be too large, that is,

$$l \geq \max\{k \mid \text{a variable } x \text{ occurs in at most } k \text{ clauses of } \alpha\} + 2. \qquad \square$$

The fact that a rope propagates a truth value is due to the fact that its length is tight, as the following lemma shows.

Lemma 8.5. *In every IC-planar drawing of G_α, respecting the rotation system, the endpoint of a rope at a gadget is OUT if the endpoint at the vertex is IN.*

Proof. By construction of M-graphs, they cannot be crossed by a rope. Thus, the rope must cross a barrier of l edges. In any IC-planar embedding, this costs (at least) $2l - 1$ edges. In addition, a rope is crossed by the outer membrane of the variable gadget. If the endpoint at the vertex is IN, then the rope is crossed by the membrane, which costs two edges. Hence, it cannot cross another membrane, since its length is $2k + 3$. $\qquad \square$

The consistency of the truth assignment of the variable is granted by the following lemma.

Lemma 8.6. *In every IC-planar drawing of G_α, and every variable x, if x_0^- is OUT, then each x_i^- is OUT and all x_j^+ are IN for $i = 1, \dots, k + 1$ and $j = 1, \dots, k + 1$. The reversed statement also holds.*

Proof. If x_0^- is OUT, then the local membrane crosses two edges of the paths of length two fro x_0^- to the border of M. Thus, by Lemma 8.3, x_0^- is OUT and the local membrane must cross an edge of the path of length two from x_0^+ to x_1^-. This implies that the local membrane of the first literal gadget cannot cross this path, and therefore must cross the paths from x_1^- to the M-graph. It follows by induction that all x_i^- are OUT and all x_i^+ are IN. Conversely, if x_{k+1}^+ is OUT, then we proceed from right to left. Now, all x_i^+ are OUT and all x_i^- are IN. $\qquad \square$

With these lemmas, we can finally prove the following theorem.

Theorem 8.3. *IC-planarity testing with given rotation system is* NP-*complete.*

Proof. We have already stated in the proof of Theorem 8.2 that IC-planarity is in NP. We reduce from Planar-3SAT and show that an expression α is satisfiable if and only if the constructed graph G_α^* has a IC-planar drawing. If α is satisfiable, then we draw the V- and C-gadgets according to the assignment, such that the initial vertex of each rope from the gadget of a variable x is IN at the C-gadget if the literal is assigned the value true. The resulting drawing is IC-planar by construction. If G_α^* has an IC-planar drawing, then we obtain the truth assignment of α from the drawing. Thus, IC-planarity with a given rotation system is NP-complete. $\qquad \square$

Note that the construction for the proof of Theorem 8.3 also holds in the variable embedding setting, since the used graphs have an almost fixed IC-planar embedding. From this, we can obtain an alternative NP-completeness proof of IC-planarity testing.

(a)   **(b)**
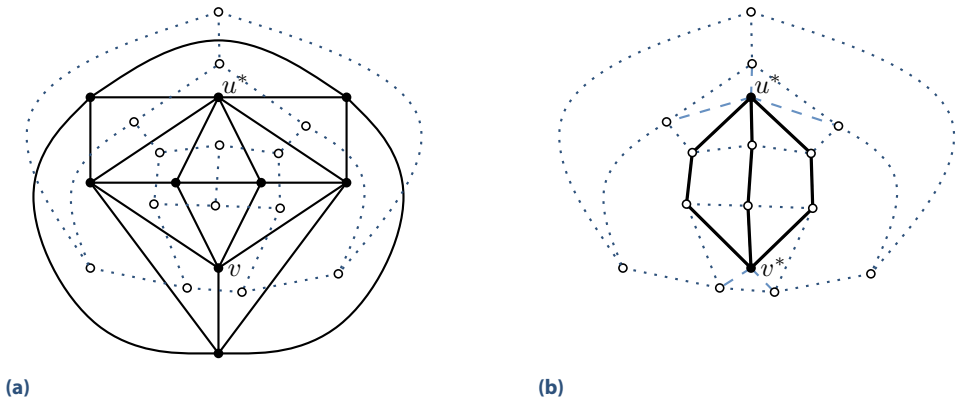
Figure 8.9: (a) A triconnected Graph T (solid) and its dual T^* (dotted), (b) The extended graph $T^* \cup \{u^*, v^*\}$ and the three length-3 paths between u^* and v^* (bold).

8.2.3 Polynomial-Time Test for a Triangulated Plane Graph Plus a Matching.

On the positive side, we now describe an algorithm to test whether a graph $G = (V, E_T \cup E_M)$ that consists of a triangulated plane graph $T = (V, E_T)$ and a matching $M = (V_M, E_M)$ with $V_M \subseteq V, E_M \cap E_T = \varnothing$ admits an IC-planar drawing that preserves the embedding of T. An outline of the algorithm is as follows.

(1) Check for every matching edge if there is a way to draw it such that it crosses only one edge of T.

(2) Split T into subgraphs that form a hierarchical tree structure.

(3) Traverse the 4-block tree bottom-up and solve a 2SAT formula for each tree node.

In order to check whether there is a valid placement for each matching edge $(u, v) \in M$, we have to find two adjacent faces, one of which is incident to u, while the other one is incident to v. To this end, we consider the dual T^* of T that contains a vertex for each face in T that is not incident to a vertex $w \in V_M \smallsetminus \{u, v\}$, and an edge for each edge in T that separates two faces. Further, we add two additional vertices u^* and v^* to T^* that are connected to all faces that are incident to u and v, respectively. In the resulting graph $T^* \cup \{u^*, v^*\}$, we look for all paths of length 3 from u^* to v^*. These paths are equivalent to routing (u, v) through two faces that are separated by a single edge. Note that no path of length 1 or 2 can exist, since (i) by construction u^* and v^* are not connected by an edge and (ii) if there was a path of length 2 between u^* and v^*, then u and v would lie on a common face in the triangulated graph T; thus, the edge (u, v) would exist both in E_T and in E_M, which is not possible since $E_T \cap E_M = \varnothing$. See Figure 8.9 for an illustration. If there is an edge that has no valid placement, then G is not IC-planar and the algorithm stops. Otherwise, we save each path that we found as a possible routing for the corresponding edge in M.
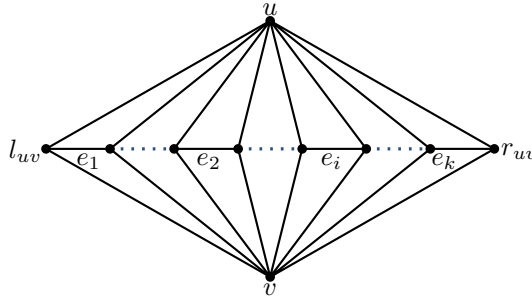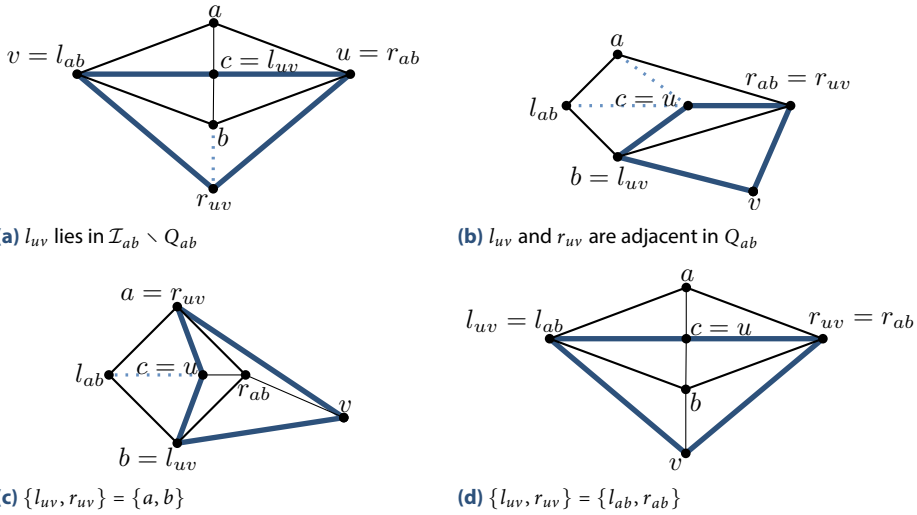
Figure 8.10: The ordered routing edges $e_1, \ldots, e_k$ lie inside the quadrangle (u, l_{uv}, v, r_{uv}).

Now, we make some observations on the structure of the possible routings in a drawing of an edge $(u, v) \in M$ that we can use to get a hierarchical tree structure of the graph T. Every routing is uniquely represented by an edge that separates a face incident to u and a face incident to v and that might be crossed by (u, v). We call these edges *routing edges*. Let there be k routing edges for the pair (u, v). Each of these edges forms a triangular face with u. From the embedding, we can enumerate the edges by the counterclockwise order of their corresponding faces at u. This gives an ordering $e_1, \ldots, e_k$ of the routing edges. Let $e_1 = (l_{uv}, l'_{uv})$ and $e_k = (r'_{uv}, r_{uv})$ such that the edge (u, l_{uv}) comes before the edge (u, l'_{uv}), and the edge (u, r'_{uv}) comes before (u, r_{uv}) in the counterclockwise order at u. Then, all edges $e_1, \ldots, e_k$ lie within the *routing quadrangle* (u, l_{uv}, v, r_{uv}); see Figure 8.10. Note that there may be more complicated structures between the edges, but they do not interfere with the ordering. Denote by $Q_{uv} = (u, l_{uv}, v, r_{uv})$ the routing quadrilateral of the matching edge $(u, v) \in M$. We define the *interior* $\mathcal{I}_{uv} = (\mathcal{V}_{uv}, \mathcal{E}_{uv})$ as the maximal subgraph of T such that, for all vertices $v \in \mathcal{V}_{uv}$, each path from v to a vertex on the outer face of T contains u, l_{uv}, v, or r_{uv}. Consequently, $Q_{uv} \in \mathcal{V}_{uv}$. We will now show that two interiors cannot overlap.

Lemma 8.7. *For each pair of interiors $\mathcal{I}_{uv}, \mathcal{I}_{ab}$, exactly one of the following conditions holds:*

(a) $\mathcal{I}_{uv} \cap \mathcal{I}_{ab} = \varnothing$

(b) $\mathcal{I}_{uv} \subset \mathcal{I}_{ab}$

(c) $\mathcal{I}_{ab} \subset \mathcal{I}_{uv}$

(d) $\mathcal{I}_{uv} \cap \mathcal{I}_{ab} = Q_{uv} \cap Q_{ab}$.

Proof. Assume that neither of the condition holds. Recall that Q_{uv} and Q_{ab} are the boundaries of the interiors. Note that $\mathcal{I}_{uv} \cap \mathcal{I}_{ab} = \varnothing$ corresponds to disjointness, $\mathcal{I}_{uv} \subset \mathcal{I}_{ab}$ and $\mathcal{I}_{ab} \subset \mathcal{I}_{uv}$ correspond to inclusion, and $\mathcal{I}_{uv} \cap \mathcal{I}_{ab} = Q_{uv} \cap Q_{ab}$ corresponds to touching in their boundary of the two interiors $\mathcal{I}_{uv}$ and $\mathcal{I}_{ab}$. Thus, if the conditions do not hold, the interiors must properly intersect, that is, without loss of generality, there is a vertex $c \in Q_{uv}$ that lies in $\mathcal{I}_{ab} \smallsetminus Q_{ab}$, and a vertex $d \in Q_{uv}$ that does not lie in $\mathcal{I}_{ab}$. Hence, the other two vertices of Q_{uv} lie in Q_{ab}. Clearly, c and d are opposite vertices in Q_{uv}. By definition of IC-planar graphs, it holds that $\{a, b\} \cap \{u, v\} = \varnothing$.

(a) l_{uv} lies in $\mathcal{I}_{ab} \setminus Q_{ab}$

(b) l_{uv} and r_{uv} are adjacent in Q_{ab}

(c) $\{l_{uv}, r_{uv}\} = \{a, b\}$

(d) $\{l_{uv}, r_{uv}\} = \{l_{ab}, r_{ab}\}$

Figure 8.11: Illustration of the proof of Lemma 8.7. The routing quadrilateral Q_{ab} is drawn bold, and Q_{uv} is drawn extra bold.

First, assume that $c = l_{uv}$. Then, u and v must lie in Q_{ab}. More specifically, by definition of IC-planar graphs $\{u, v\} = \{l_{ab}, r_{ab}\}$. Without loss of generality, assume that $u = r_{ab}$ and $v = l_{ab}$. Since the edges (u, c) and (v, c) have to lie in $\mathcal{I}_{ab}$, this leads to the situation depicted in Figure 8.11a. However, this implies that that there are only two routing edges for (a, b) with one of them incident to u, and the other one is incident to v. Thus, the routing edges are not valid. The case that $c = r_{uv}$ works analogously.

Second, assume that $c = u$. Then, l_{uv} and r_{uv} must lie in Q_{ab}. If l_{uv} and r_{uv} are adjacent on Q_{ab}, say $l_{uv} = b$ and $r_{uv} = r_{ab}$, then there is only a single routing edge for (u, v) that is incident to b and thus not valid; see Figure 8.11b. Otherwise, there are two cases. If $\{l_{uv}, r_{uv}\} = \{a, b\}$, say $r_{uv} = a$ and $l_{uv} = b$, then there are only two routing edges for (u, v) with one of them incident to a, and the other one incident to b; see Figure 8.11c. If $\{l_{uv}, r_{uv}\} = \{l_{ab}, r_{ab}\}$, say $l_{uv} = l_{ab}$ and $r_{uv} = r_{ab}$, then both routing edges of (u, v) are incident to b; see Figure 8.11d. The case that $c = v$ works analogously.

This proves that, if there is a proper intersection between two routing quadrilaterals, than at least one of the corresponding matching edges has no valid routing edge. Thus, one of the conditions must hold. $\qquad\square$

By using Lemma 8.7, we can find a hierarchical structure on the routing quadrilaterals. We construct a directed graph $H = (V_H, E_H)$ with $V_H = \{\mathcal{I}_{uv} \mid (u, v) \in M\} \cup \{G\}$. For each pair $\mathcal{I}_{uv}, \mathcal{I}_{xy}$, E_H contains a directed edge $(\mathcal{I}_{uv}, \mathcal{I}_{xy})$ if and only if $\mathcal{V}_{uv} \subset \mathcal{V}_{xy}$ and there is no matching edge (a, b) with $\mathcal{V}_{uv} \subset \mathcal{V}_{ab} \subset \mathcal{V}_{xy}$. Finally, we add an edge from each subgraph that has no outgoing edges to G. Each vertex but G only has one outgoing edge. Obviously, this graph contains no (undirected) cycles. Thus, H is a tree.

We will now show how to construct a drawing of G based on H in a bottom-up fashion. We will first look at the leaves of the graph. Let $\mathcal{I}_{uv}$ be a vertex of H whose children are all

leaves. Let $\mathcal{I}_{u_i v_i}, \ldots, \mathcal{I}_{u_k v_k}$ be these leaves. Since these interiors are all leaves in H, we can pick any of their routing edges. However, the interiors may touch on their boundary, so not every combination of routing edges can be used. Assume that a matching edge $(u_i, v_i), 1 \le i \le k$ has more than two valid routing edges. Then, we can always pick a *middle* one, that is, a routing edge that is not incident to $l_{u_i v_i}$ and $r_{u_i v_i}$, since this edge will not interfere with a routing edge of another matching edge.

Now, we can create a 2SAT formula to check whether there is a valid combination of routing edges as follows. For the sake of clarity, we will create several redundant variables and formulas. These can easily be removed or substituted by shorter structures to improve the running time. For each matching edge $(u_i, v_i), 1 \le i \le k$, we create two binary variables l_i and r_i, such that l_i is `true` if and only if the routing edge incident to $l_{u_i v_i}$ is picked, and r_i is `true` if and only if the routing edge incident to $r_{u_i v_i}$ is picked. If (u_i, v_i) has only one routing edge, then it is obviously incident to $l_{u_i v_i}$ and $r_{u_i v_i}$, so we set $l_{u_i v_i} = r_{u_i v_i} =$ `true` by adding the clauses $l_{u_i v_i} \vee$ `false` and $r_{u_i v_i} \vee$ `false`. If (u_i, v_i) has exactly two routing edges, the picked routing edge has to be incident to either $l_{u_i v_i}$ or $r_{u_i v_i}$, so we add the clauses $l_{u_i v_i} \vee r_{u_i v_i}$ and $\neg l_{u_i v_i} \vee \neg r_{u_i v_i}$. If (u_i, v_i) has more than two routing edges, we can pick a middle one, so we set $l_{u_i v_i} = r_{u_i v_i} =$ `false` by adding the clauses $\neg l_{u_i v_i} \vee$ `false` and $\neg r_{u_i v_i} \vee$ `false`. Next, we have to add clauses to forbid pairs of routing edges that can not be picked simultaneously, that is, they share a common vertex. Consider a pair of matching edges $(u_i, v_i), (u_j, v_j), 1 \le i, j \le k$. If $r_{u_i v_i} = l_{u_j v_j}$, we add the clause $\neg r_i \vee \neg l_j$. For the other three cases, we add an analogue clause.

Now, we use this 2SAT to decide whether the subgraph I_{uv} is IC-planar, and which routing edges can be used. For each routing edge (a, b) of I_{uv}, we solve the 2SAT formula given above with additional constraints that forbid the use of routing edges incident to a and b. To that end, add the following additional clauses: If $l_{u_i v_i} = a$, add the clause $\neg l_i \vee$ `false`. For the other three cases, we add an analogue clause. If this 2SAT formula has no solution, then the subgraph $\mathcal{I}_{uv}$ is not IC-planar. Otherwise, there is a solution where you pick the routing edges corresponding to the binary variables. To decide whether a subgraph I_{uv} whose children are not all leaves is IC-planar, we first compute which of their routing edges can be picked by recursively using the 2SAT formula above. Then, we use the 2SAT formula for I_{uv} to determine the valid routing edges of I_{uv}. Finally, we can decide whether G is IC-planar and, if yes, get a drawing by solving the 2SAT formula of all children of G.

Hence, we give the following theorem.

Theorem 8.4. *Let $T = (V, E_T)$ be a triangulated plane graph with n vertices and let $M = (V, E_M)$ be a matching. There exists an $O(n^3)$-time algorithm to test if $G = (V, E_T \cup E_M)$ admits an IC-planar drawing that preserves the embedding of T. If the test is positive, the algorithm computes a feasible drawing.*

Proof. We need to prove that the described algorithm runs in $O(n^3)$ time. Indeed, for each subgraph I_{uv}, we have to run a 2SAT formula for each routing edge. Once we have determined the valid routing edges, we do not have to look at the children anymore. Let c_{uv} be the number of children of I_{uv}. Each of these 2SAT formula contains $2c_{uv}$ variables and up to $2c_{uv} + 2c_{uv}^2$ clauses. Since every edge of G can only be a routing edge for exactly one matching edge, we have to solve at most n 2SAT formulas.

The tree H consists of at most $n/2 + 1$ vertices (one for each matching edge), so a very conservative estimation is that we have to solve $O(n)$ 2SAT formulas with $O(n)$ variables

and $O(n^2)$ clauses each. Aspvall et al. [APT79] showed how to solve 2SAT in time linear in the number of clauses. We can use the linear-time algorithm of Section 8.1 to draw the IC-planar graph corresponding to the IC-planar embedding by picking the routing edges corresponding to the binary variables. Thus, our algorithm runs in $O(n^3)$ time. □

8.3 IC-planarity and Rac Graphs

It is known that every maximally dense Rac graph is 1-planar, and that there exist both 1-planar graphs that are not Rac and Rac graphs that are not 1-planar [EL13]. Additionally, every 1-planar graph drawable with all vertices on the outer face is Rac [DE12]. Here, we further investigate the intersection between the classes of 1-planar and Rac graphs, showing that all IC-planar graphs are Rac graphs. To this aim, we describe a polynomial-time constructive algorithm. The computed drawings may require exponential area, which is however worst-case optimal; indeed, we exhibit IC-planar graphs that require exponential area in any possible IC-planar straight-line Rac drawing. Our construction is based on extending the linear-time algorithm by de Fraysseix et al. that computes a planar straight-line grid drawing of a maximal (that is, triangulated) plane graph in quadratic area [dFPP90]; we call this algorithm the dFPP algorithm. For completeness, we recall the idea behind dFPP before describing our extension.

Algorithm dFPP. Let G be a maximal plane graph with $n \geq 3$ vertices. The dFPP algorithm first computes a suitable linear ordering of the vertices of G, called a *canonical ordering* of G, and then incrementally constructs a drawing of G using a technique called *shift method*.

This method adds one vertex per time, following the computed canonical ordering and shifting vertices already in the drawing when needed. Namely, let $\sigma = (v_1, v_2, \ldots, v_n)$ be a linear ordering of the vertices of G. For each integer $k \in [3, n]$, denote by G_k the plane subgraph of G induced by the k vertices $v_1, v_2, \ldots, v_k$ ($G_n = G$) and by C_k the boundary of the outer face of G_k, called the *contour* of G_k. Ordering σ is a canonical ordering of G if the following conditions hold for each integer $k \in [3, n]$:

(i) G_k is biconnected and internally triangulated;

(ii) (v_1, v_2) is an outer edge of G_k; and

(iii) if $k + 1 \leq n$, vertex v_{k+1} is located in the outer face of G_k, and all neighbors of v_{k+1} in G_k appear on C_k consecutively.

We call *lower neighbors* of v_k all neighbors v_j of v_k for which $j < k$. Following the canonical ordering σ, the shift method constructs a drawing of G one vertex per time. The drawing Γ_k computed at step k is a drawing of G_k. Throughout the computation, the following invariants are maintained for each Γ_k, with $3 \leq k \leq n$:

(I1) $p_{v_1} = (0, 0)$ and $p_{v_2} = (2k - 4, 0)$;

(I2) $x(w_1) < x(w_2) < \cdots < x(w_t)$, where $w_1 = v_1, w_2, \ldots, w_t = v_2$ are the vertices that appear along C_k, going from v_1 to v_2.

(I3) Each edge (w_i, w_{i+1}) (for $i = 1, 2, \ldots, t - 1$) is drawn with slope either +1 or −1.

(a) placing v_{k+1} at $\mu(w_p, w_q)$ would create overlaps

**(b)** after the shift operation, v_{k+1} can be placed at $\mu(w_p, w_q)$ without overlaps
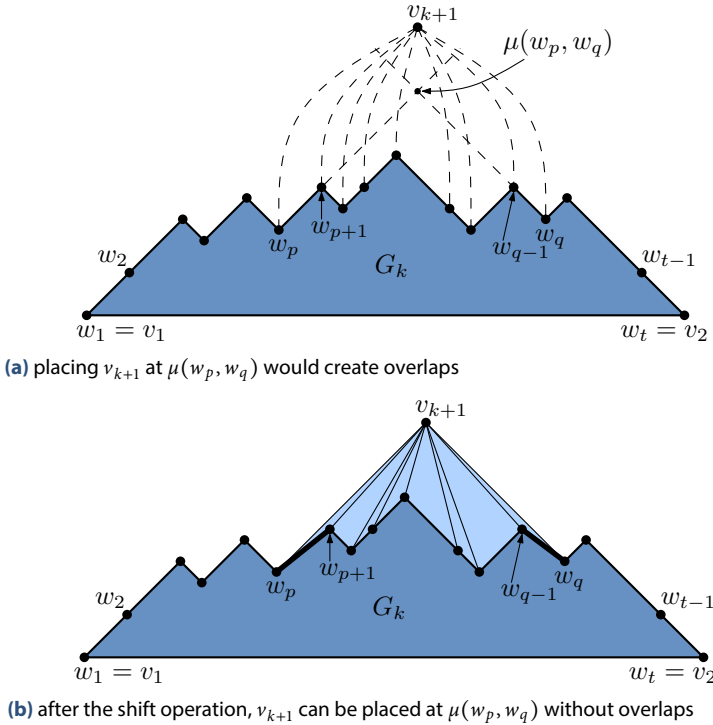
Figure 8.12: Illustration of the shift algorithm at the addition step of v_{k+1}. The shift operation changed the slopes of the edges drawn bold.

More precisely, Γ_3 is constructed placing v_1 at $(0,0)$, v_2 at $(2,0)$, and v_3 at $(1,1)$. The addition of v_{k+1} to Γ_k is executed as follows. Let $w_p, w_{p+1}, \ldots, w_q$ be the lower neighbors of v_{k+1} ordered from left to right. Denote by $\mu(w_p, w_q)$ the intersection point between the line with slope +1 passing through w_p and the line with slope −1 passing through w_q. Point $\mu(w_p, w_q)$ has integer coordinates and thus it is a valid placement for v_{k+1}. With this placement, however, (v_{k+1}, w_p) and (v_{k+1}, w_q) may overlap with (w_p, w_{p+1}) and (w_{q-1}, w_q), respectively; see Figure 8.12a. To avoid this, a *shift* operation is applied: $w_{p+1}, w_{p+2}, \ldots, w_{q-1}$ are shifted to the right by 1 unit, and $w_q, w_{q+1}, \ldots, w_t$ are shifted to the right by 2 units. Then v_{k+1} is placed at point $\mu(w_p, w_q)$ with no overlap; see Figure 8.12b. We recall that, to keep planarity, when the algorithm shifts a vertex w_i ($p+1 \le i \le t$) of C_k, it also shifts some of the inner vertices together with it; for more details on this point refer to [CP95, dFPP90]. By Invariants (I1) and (I3), the area of the final drawing is $(2n - 4) \times (n - 2)$.

Our extension. Let G be an IC-plane graph, and assume that G^+ is the planar-maximal IC-plane graph obtained from G by applying the technique of Lemma 8.1. Our drawing algorithm computes an IC-planar drawing of G^+ with right-angle crossings, by extending algorithm dFPP. It adds to the classical shift operation *move* and *lift* operations to guarantee that the one of the crossing edges of a kite is vertical and the other one is horizontal. We now
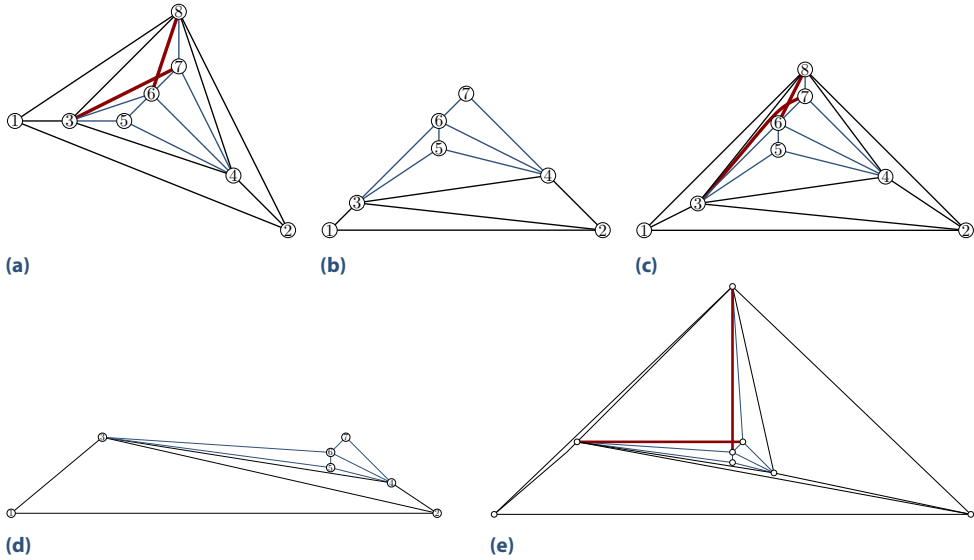
Figure 8.13: Example run of our algorithm on an IC-planar graph G with a separating triangle. The crossing edges are drawn bold, the edges inside the separating triangle are drawn light. (a) Input graph G; (b) output of dFPP after vertex 7; (c) output of dFPP after vertex 8; (d) lifting 3 to the level of 7; (e) moving 8 directly above 6.

give an idea of our technique, which we call RAC-drawer. Details are given in the proof of Theorem 8.5. Let σ be a canonical ordering constructed from the underlying maximal plane graph of G^+. Vertices are incrementally added to the drawing, according to σ, following the same approach as for dFPP.

However, suppose that $K = (a, b, c, d)$ is a kite of G^+, and that a and d are the first and the last vertex of σ among the vertices in K, respectively. Once d has been added to the drawing, the algorithm applies a suitable combination of move and lift operations to the vertices of the kite to rearrange their positions so to guarantee a right-angle crossing. Note that, following the dFPP technique, a was placed at a y-coordinate smaller than the y-coordinate of d. A move operation is then used to shift d horizontally to the same x-coordinate as a (that is, (a, d) becomes a vertical segment in the drawing); a lift operation is used to vertically shift the lower between b and c, such that these two vertices get the same y-coordinates. Both operations are applied so to preserve planarity and to maintain Invariant (I3) of dFPP; however, they do not maintain Invariant (I1), thus the area can increase more than in the dFPP algorithm and may be exponential. The application of move/lift operations on the vertices of two distinct kites do not interfere each other, as the kites do not share vertices in an IC-plane graph. Figure 8.13 shows a running example of our algorithm.

Theorem 8.5. *There is a $O(n^3)$-time algorithm that takes an IC-plane graph G with n vertices as input and constructs a straight-line IC-planar RAC grid drawing of G.*

Proof. Let G^+ be the augmented graph constructed from G by using Lemma 8.1. Call G' the subgraph obtained from G^+ by removing one edge from each pair of crossing edges; G' is a maximal plane graph (see condition (c3) of Lemma 8.1). We apply on G' the shelling procedure
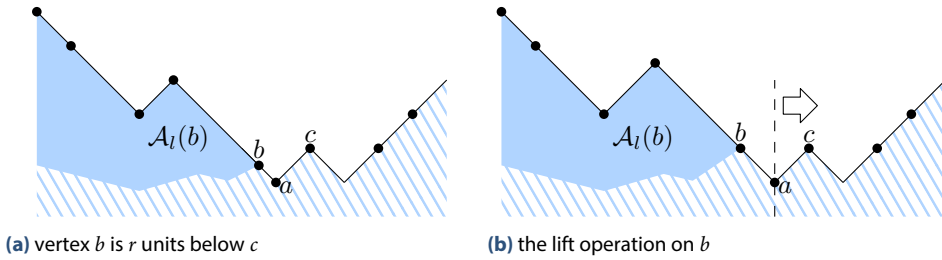
(a) vertex b is r units below c **(b)** the lift operation on b

Figure 8.14: The lift operation

used by de Fraysseix et al. to compute a canonical ordering σ of G' in $O(n)$ time [dFPP88]; it goes backwards, starting from a vertex on the outer face of G' and successively removing a vertex per time from the current contour. However, during this procedure, some edges of G' can be replaced with some other edges of G^+ that were previously excluded, although G' remains maximal planar. Namely, whenever the shelling procedure encounters the first vertex d of a kite $K = (a, b, c, d)$, it marks d as $\text{top}(K)$, and considers the edge e of K that is missing in G'. If e is incident to d in K, the procedure reinserts it and removes from G' the other edge of K that crosses e in G^+. If e is not incident to d, the procedure continues without varying G'.

We then compute a drawing of G^+ by using the RAC-drawer algorithm. Let vertex $v = v_{k+1}$ be the next vertex to be placed according to σ. Let $\mathcal{U}(v)$ be the set of lower neighbors of v, and let $\lambda(v)$ and $\rho(v)$ be the leftmost and the rightmost vertex in $\mathcal{U}(v)$, respectively. Also, denote by $\mathcal{A}_l(v)$ the vertices to the top-left of v, and by $\mathcal{A}_r(v)$ the vertices to the top-right of v. If v is not $\text{top}(K)$ for some kite K, then v is placed by following the rules of dFPP, that is, at the intersection of the ± 1 diagonals through $\lambda(v)$ and $\rho(v)$ after applying a suitable shift operation. If $v = \text{top}(K)$ for some kite K, the algorithm proceeds as follows. Let $K = (a, b, c, d)$ with $v = d = \text{top}(K)$. The remaining three vertices of K are in G_k and are consecutive along the contour C_k, as they all belong to $\mathcal{U}(d)$ (recall that, G' has been computed in such a way that it contains edge (a, d)). W.l.o.g., assume that they are encountered in the order $\{b, a, c\}$ from left to right. Two cases are now possible:

Case 1: $a \prec b$ and $a \prec c$ in σ. This implies that $a = \rho(b)$ and $a = \lambda(c)$. The edges (a, b) and (a, c) have slope -1 and $+1$, respectively, as they belong to C_k. We now aim at having b and c at the same y-coordinate, by applying a lift operation. Suppose first that $r = y(c) - y(b) > 0$; see Figure 8.14a. We apply the following steps:

(i) Temporarily undo the placement of b and of all vertices in $\mathcal{A}_l(b)$.

(ii) Apply the shift operation to vertex $\rho(b) = a$ by $2r$ units to the right, which implies that the intersection of the diagonals through $\lambda(v)$ and $\rho(v)$ is moved by r units to the right and by r units above their former intersection point. Hence, b and c are placed at the same y-coordinate; see also Figure 8.14b in.

(iii) Reinsert the vertices of $\mathcal{A}_l(b)$ and modify σ accordingly. Namely, by definition, each vertex in $\mathcal{A}_l(b)$ does not belong to $\mathcal{U}(b)$ and it is not an inner vertex below b; therefore,
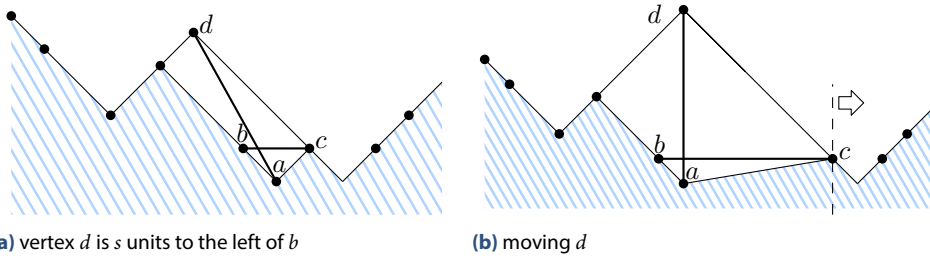
(a) vertex d is s units to the left of b **(b)** moving d

vertices in $\mathcal{A}_l(b)$ can be safely removed. Hence, σ can be modified by moving all of them after b.

If $r = y(c) - y(b) < 0$, a symmetric operation is applied:

(i) Undo the placement of c and of all vertices in $\mathcal{A}_r(c)$.

(ii) Apply the shift operation to vertex $\rho(c)$ by $2r$ additional units to the right.

(iii) Reinsert the vertices of $\mathcal{A}_r(c)$.

Finally, we place d vertically above a. To this aim, we may need to apply a move operation; see Figure 8.15a. If $s = x(d) - x(a) > 0$, then we apply the shift operation to vertex $\rho(d) = c$ by $2s$ units to the right and then place d (see Figure 8.15b). If $s = x(d) - x(a) < 0$, then a symmetric operation is applied. Namely, we apply the shift operation to vertex $\lambda(d) = b$ by $2s$ units to the left and then place d (clearly, the shift operation can be used to operate in the left direction with a procedure that is symmetric to the one that operates in the right direction).

Edges (a, d) and (b, c) are now vertical and horizontal, respectively. In the next steps, their slopes do not change, as their endvertices are shifted only horizontally (they do not belong to other kites); also, a is shifted along with d, as it belongs to $\mathcal{U}(d)$.

Case 2: $b < a < c < d$. This case occurs if there is a separating triangle $\tau = (u, b, d)$ that has a and c (and possibly other vertices) in its interior. A separating triangle is a 3-cycle such that by removing its three vertices we disconnect the graph in two components. One component embedded inside the triangle and one component embedded outside the triangle. Let $\{z_1 < \ldots < z_r\}$ be the sequence of $r \geq 1$ neighbors of b inside τ, with $a = z_r$ and $b = \lambda(z_i)$, with $1 \leq i \leq r$, as shown in Figure 8.16.

Suppose these vertices are in $\mathcal{A}_r(b)$, since the case when they are in $\mathcal{A}_l(b)$ is symmetric. Consider the slopes α_i of the edges (z_i, z_{i+1}) for $1 \leq i < r$, and let $\alpha_{\min}$ be the negative slope with the least absolute value among them; see the bold edge in Figure 8.16. Let s be the (negative) slope of the edge $(b, \rho(b))$. We aim at obtaining a drawing where $|s| \leq |\alpha_{min}|$. To this aim we apply the shift operation on $\rho(b)$ by x units to the right, which stretches and flattens the edge $(b, \rho(b))$. If $|\alpha_{min}| = h/w$, and $|s| = h'/w'$, then the value of x is the first even integer such that $x \geq (h'w - hw')/h$. The fact that x is even preserves the even length of the edges on the contour. This preliminary operation will be useful in the following part.
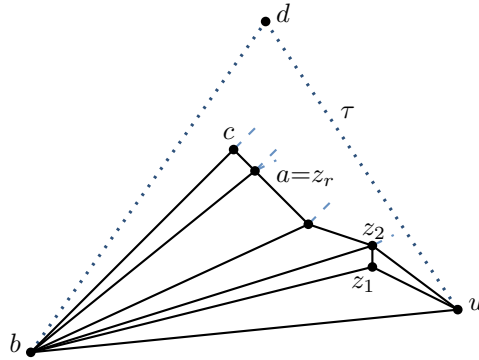
Figure 8.16: Illustration for the proof of Theorem 8.5. The edge with slope α_{min} is thicker.

Next, let $\Delta(b) = y(b) - y(\rho(b)) > 0$, and let $\Delta(c) = y(c) - y(b) > 0$, that is, b lies $\Delta(b)$ rows above $\rho(b)$, and c lies $\Delta(c)$ rows above b, where the edges (b, a) and (a, c) have slope +1. We apply the following procedure to lift b at the same y-coordinate of c.

  (i) We undo the placement of all vertices in $\mathcal{A}_l(b)$.

  (ii) If $\Delta(c)$ is not a multiple of $\Delta(b)$, say $\Delta(b) + \delta = q \cdot \Delta(c)$ for some integer q, then we shift $\rho(c)$ by 2δ units to the right. This implies that c moves by (δ, δ) above its former position.

  (iii) We set the y-coordinate of vertex b equal to the y-coordinate of c. To that end, we stretch the edge $(\rho(b), b)$ by the factor q. Let w' be the width and h' be the height of the edge $(\rho(b), b)$. The new edge has the same slope as before, and has width qw' and height qh'. This implies shifting all vertices $\rho(b), z_1, \ldots, z_{r-1}, a, c$ by $(q-1)w'$ units to the right. Vertex b may need a further adjustment by a single unit left shift if $b = \lambda(d)$ and the intersection point of the ± 1 diagonals through $\lambda(d)$ and $\rho(d)$ is not a grid point. Also, we apply the shift operation on $\lambda(b)$ by $(q-1)h'$ units to the left. This particular lifting operation applied on vertex b preserves planarity, which could be violated only by edges incident to b. Namely, if vertex w is a neighbor of b in $\mathcal{U}(b)$, then the edge (w, b) is vertically stretched by $(q-1)h'$ units. This cannot enforce a crossing, since it means a vertical shift of w. Clearly, b can see $\rho(b)$, since the edge was stretched. Consider the upper right neighbors $z_1, \ldots, z_r$ with $z_r = a$ of b. The edges (b, z_i) change direction from right upward to right downward. Since the absolute value of the slope of the edge $(b, \rho(b))$ is bounded from above by α_{min}, the new position of b is below or to the left of the edges (z_i, z_{i+1}) for $1 \le i < r$. Hence, b can see each such neighbor z_i, including $a = z_r$. The lifting of b has affected all vertices $v \in \mathcal{A}_l(b)$ with $y(v) < y(d)$.

  (iv) We re-insert the vertices in $\mathcal{A}_l(b)$ by changing σ accordingly, as already explained for **Case 1**.

Finally, we place $d = \text{top}(K)$. First, we place d at the intersection point of the ± 1 diagonals through $\lambda(d)$ and $\rho(d)$. Then, we adjust d such that it lies vertically above a. If the preliminary
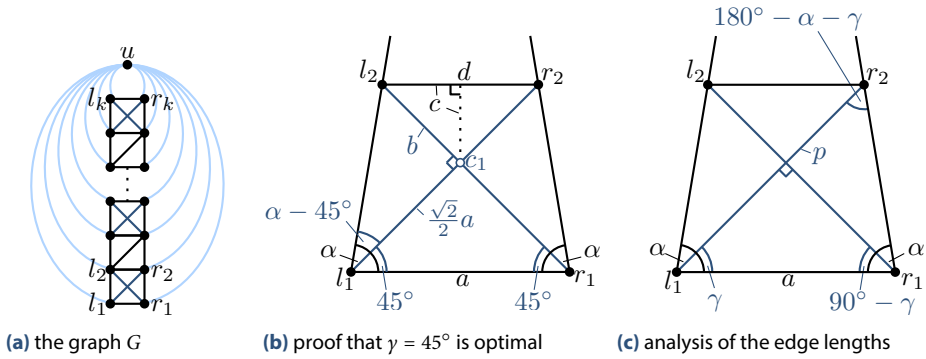
(a) the graph G **(b)** proof that $\gamma = 45°$ is optimal   **(c)** analysis of the edge lengths

Figure 8.17: An IC-planar graph G that requires exponential area if drawn Rac

position of d is t units to the left (right) of a, then we apply the shift operation on $\rho(d)$ by $2t$ units to the right (on $\lambda(d)$ by $2t$ units to the left).

To conclude the proof we need to consider the first edge of the construction, which is drawn horizontal. Since the lift operation requires an edge that does not have slope 0, we may need to introduce dummy vertices and edges. Namely, if there is a kite including the base edge (v_1, v_2), then we add two dummy vertices $1', 2'$ below it that form a new base edge (v_1', v_2'). We add the additional edges (v_1', v_1), (v_1', v_2), (v_1', v_n), (v_2', v_2) and (v_2', v_n) to make the graph maximal planar. These dummy vertices and edges will be removed once the last vertex v_n is placed.

In terms of time complexity, G^+ can be computed in $O(n)$ time, by Lemma 8.1. Furthermore, each shift, move and the lift operation can be implemented in $O(n)$ time, hence the placement of a single vertex costs $O(n)$ time. However, in some cases (in particular, when we are placing the top vertex of a kite), we may need to undo the placement of a set of vertices and re-insert them afterwards. Since when we undo and reinsert a set of vertices we also update σ accordingly, this guarantees that the placement of the same set of vertices will not be undone anymore. Thus, the reinsertion of a set of $O(n)$ vertices costs $O(n^2)$. Hence, we have $\sum_{i=1}^{n} O(n + n^2)$ which gives an $O(n^3)$ time complexity. □

Theorem 8.5 and the fact that there exist n-vertex Rac graphs with $4n - 10$ edges [DEL11] while an n-vertex IC-planar graph has at most $13n/4 - 6$ edges [ZL13] imply that IC-planar graphs are a proper subfamily of Rac graphs.

We now show that exponential area is required for Rac drawings of IC-planar graphs. Since the vertices are not drawn on the integer grid, the drawing area is measured as the proportion between the longest and the shortest edge.

Theorem 8.6. *There exists an infinite family $\mathcal{G}$ of graphs such that every IC-planar straight-line Rac drawing of an n-vertex graph $G \in \mathcal{G}$ requires area $\Omega(q^n)$, for some constant $q > 1$.*

Proof. Consider the family $\mathcal{G}$ depicted in Figure 8.17a. Let $G \in \mathcal{G}$ with n vertices. Then, G consists of $(n-1)/4$ kites that are sequentially connected by two triangles. More specifically, the graph consists of an augmented $(k + 1) = ln/2$-ladder. We denote the vertices on the left rail by $l_0, l_2, \ldots, l_k$ and the vertices on the right rail by $r_0, r_2, \ldots, r_k$. For every face $(l_i, r_i, r_{i+1}, l_{i+1})$ with odd i, we add the edges (l_i, r_{i+1}) and (r_i, l_{i+1}) to form a kite. For

every face $(l_i, r_i, r_{i+1}, l_{i+1})$ with even i, we add the edge (l_i, r_{i+1}) to triangulate it. Then, every vertex of this augmented ladder is connected to a universal vertex u.

This graph is constructed such that, if we remove one edge for each pair of crossing edges, the resulting planar graph is a triangulation. Thus, there is only one planar embedding (up to the choice of the outer face). Furthermore, the removed crossing edges can only be inserted at one position. Hence, up to the choice of the outer face, the embedding depicted in Figure 8.17a is the only IC-planar embedding. We consider the graph embedded such that vertices u, l_0, and r_0 are on the boundary of the outer face and the circular order of the vertices around u is $l_0, l_1, \ldots, l_k, r_k, r_{k-1}, \ldots, r_0$.

Without loss of generality, we assume that the edge (l_1, r_1) is drawn horizontally, otherwise we rotate the drawing to make the edge horizontal. For sake of simplicity, we assume that $x(u) = (x(l_1) + x(r_1))/2$, such that the triangle (l_1, r_1, u) is isosceles. Otherwise, the proof works analogously, but with more complicated formulas.

Assume that l_0, l_1, r_0, r_1, u have already been placed. Consider the first kite $K = (l_1, r_1, r_2, l_2)$ depicted in Figure 8.17b. Let a be the length of the edge (l_1, r_1), and let α be the angle at l_1 spanned by (u, l_1, r_1) which is equivalent to the angle at r_1 spanned by (l_1, r_1, u). Since we have to place a crossing pair with a pair of edges that form a Rac inside the kite, it holds that $\alpha > 45°$. In order to fit the whole drawing (except l_0 and r_0) into the triangle (l_1, r_1, u), we have to maximize the area of the triangle (l_2, r_2, u) that contains the rest of the drawing. Obviously, this triangle has maximum area if l_2 and r_2 are as close to the edges (l_1, u) and (r_1, u) as possible, respectively. Thus, we want to minimize the area of the kite K with l_2 on (l_1, u) and r_2 on (r_2, u).

We now argue that the area of K is minimal if the edge (l_2, r_2) is drawn horizontally. Let p be the length of the diagonal (l_1, r_2) and let q be the length of the diagonal (r_1, l_2). The area of a kite is known to be $(p \cdot q)/2$. Since $\alpha > 45°$, the length p and q increases with ascending y-coordinate of r_2 and l_2, respectively. Since the crossing has to be Rac, if we move r_2 further up, then we have to move l_2 further down, and vice versa. Let γ be the angle at l_1 spanned by (r_2, l_1, r_2). Then, the angle at r_2 spanned by (l_r, r_2, l_1) is $180° - \alpha - \gamma$. By the solution of triangles, it holds that

$$p = a \sin \alpha / \sin(180° - \alpha - \gamma) = a \sin \alpha / \sin(\alpha + \gamma)$$
$$= a \sin \alpha / (\sin \alpha \cos \gamma + \cos \alpha \sin \gamma).$$

Analogously, it holds that

$$q = a \sin \alpha / \sin(\alpha + 90° - \gamma) = a \sin \alpha / \cos(\gamma - \alpha)$$
$$= a \sin \alpha / (\cos \alpha \cos \gamma + \sin \alpha \sin \gamma).$$

Making use of trigonometric functions, we get that the product is

$$pq = a^2 \sin^2 \alpha / \left(\sin \alpha \cos \alpha \cos^2 \gamma \ + \ \cos^2 \alpha \sin \gamma \cos \gamma \right.$$
$$\left. + \ \sin^2 \alpha \sin \gamma \cos \gamma \ + \ \sin \alpha \cos \alpha \sin^2 \gamma \right)$$
$$= \frac{1}{2} a^2 \sin^2 \alpha / \left(\sin(2\alpha) \cos^2 \gamma \ + \ \cos^2 \alpha \sin(2\gamma) \right.$$
$$\left. + \ \sin^2 \alpha \sin(2\gamma) \ + \ \sin(2\alpha) \sin^2 \gamma \right)$$
$$= \frac{1}{4} a^2 \sin^2 \alpha / \left(\sin(2\alpha)(1 + \cos(2\gamma)) \ + \ (1 + \cos(2\alpha)) \sin(2\gamma) \right.$$
$$\left. + \ (1 - \cos(2\alpha)) \sin(2\gamma) \ + \ \sin(2\alpha)(1 - \cos(2\gamma)) \right)$$
$$= \frac{1}{8} a^2 \sin^2 \alpha / (\sin(2\alpha) \ + \ \sin(2\gamma))$$

Since a and α are given, the product is minimal if $\sin(2\gamma)$ is maximal. As the sine function has its maximum at $\sin(90°)$, the area of the kite is minimal for $\gamma = 45°$, and thus, the edge (l_2, r_2) is drawn horizontally.

Now, we show that the area of the triangle (l_2, r_2, u) is a fraction of the area of the triangle (l_1, r_1, u) that depends only on α and a. Consider the kite depicted in Figure 8.17c. Let c_1 be the crossing point of the edges (l_1, r_2) and (r_1, l_2). Since $\gamma = 45°$, the length of the edge (l_1, c_1) is $\sqrt{2}a/2$. Let b be the length of the edge (c_1, l_2). The angle at l_1 spanned by (c_1, l_1, l_2) is $\alpha - 45°$. The tangent of this angle is defined as

$$\tan(\alpha - 45°) = \frac{b}{\sqrt{2}a/2} = 2 \frac{b}{\sqrt{2}a}.$$

Further, from the subtraction theorem of tangent follows

$$\tan(\alpha - 45°) = \frac{\tan \alpha - \tan 45°}{1 + \tan \alpha \tan 45°} = \frac{\tan \alpha - 1}{\tan \alpha + 1}$$

Combining these equations, we get that

$$b = a \frac{\sqrt{2}}{2} \frac{\tan \alpha - 1}{\tan \alpha + 1}.$$

Let d be the length of the edge (l_2, r_2). Consider the triangle spanned by l_2, c_1 and the midpoint of the edge. This triangle has a hypotenuse of length b and two catheti of length $c = d/2$. Using the Pythagorean theorem, we obtain

$$2c^2 = b^2 \Rightarrow c = \frac{b}{\sqrt{2}} = \frac{a}{2} \frac{\tan \alpha - 1}{\tan \alpha + 1}$$

and thus

$$d = 2c = a \frac{\tan \alpha - 1}{\tan \alpha + 1}.$$

Since $\alpha > 45°$, this fraction is smaller than 1.

Now, consider another kite $K_i = (l_{2i-1}, r_{2i-1}, r_{2i}, l_{2i})$, $2 \le i \le k$. Letting d_i be the length of the edge (l_{2i}, r_{2i}), it holds that

$$d_i = d_{i-1} \frac{\tan \alpha - 1}{\tan \alpha + 1}.$$

This leads us to

$$d_k = a \left(\frac{\tan \alpha - 1}{\tan \alpha + 1} \right)^k.$$

If we impose $d_k = 1$, we have that $a = q^k$, with $q = \frac{\tan \alpha + 1}{\tan \alpha - 1} > 1$. Since $k \in O(n)$, we have that the ratio between the longest and the shortest segment of the drawing is $\Omega(q^n)$, which concludes our proof.   □

8.4 Conclucing Remarks

We have shown that every IC-planar graph can be drawn straight-line in quadratic area, although the angle formed by any two crossing edges can be small. Conversely, straight-line RAC drawings of IC-planar graphs may require exponential area. It would be interesting to design algorithms that draw IC-planar graphs in polynomial area by relaxing the strict constraint that edge intersections are at right-angles and instead asking for drawings that have close to and good crossing resolution. Also, although IC-planar graphs are both 1-planar and RAC, characterizing the intersection between these two classes is still an open problem. The question whether NIC-planar graphs (see Zhang [Zha14]) which lie between IC-planar graphs and 1-planar graphs are also RAC graphs is of particular interest.

Conclusion

In this book, we have investigated three aspects of angular schematization in graph drawing: placement of boxes, visual guidance, and crossings with large angles. We have presented several efficient algorithms that provided visually appealing drawings with large angles. For some problems, we have also given NP-hardness results and approximation algorithms. We will now browse through a short overview on the results stated in this book and state some problems that are still open.

In Chapter 3, we have given the first polynomial-time algorithms of testing and computing a solution to the BOUNDARY LABELING problem with two adjacent sides, three sides, and four sides. We have also presented an efficient algorithm to maximize the number of labeled sites and, for the two-sided case, to minimize the total leader length. The latter problem remains open for the three- and four-sided case.

Open Problem 1. Can a minimum-length solution of THREE-SIDED and FOUR-SIDED BOUND-ARY LABELING be computed in polynomial time?

In Chapter 4, we have presented approximation algorithms for representing planar graphs as contact graphs of boxes with given dimensions. We have given the first constant-factor approximations for general graphs. For several graph classes, we have improved the approximation factor. Further, we have shown that the problem is APX-complete even for bipartite graphs of bounded degree. Basically, we acquired these algorithms by reduction to our solution for stars. This leads us to the following open problems.

Open Problem 2. Is there any other graph class (except paths and cycles) that admit a direct approximation algorithm in MAX-CROWN?

Open Problem 3. Can we find constant-factor approximation algorithms if the total width and height of the representation is given?

In Chapter 5, we have given an algorithm to draw any planar of maximum degree 4 graph with smooth complexity 2. If the input graph is planar of maximum degree 3 , then the algorithm only requires polynomial area. Further, we have presented an algorithm that draws biconnected outerplane graphs of maximum degree 4 with smooth complexity 1. We have also shown that planar graphs of maximum degree 4 require exponential area if drawn with smooth complexity 1, and that there is an infinite family of graphs that cannot be drawn with smooth complexity 1, although they only require one bend in an orthogonal layout. The following problems remain open.

Open Problem 4. Can any planar graph of maximum degree 4 be drawn in polynomial area with smooth complexity 2?

Open Problem 5. Can we identify larger classes of graphs admitting smooth complexity 1 layouts, for example, does any (not necessarily biconnected) outerplanar graph of maximum degree 4 or any planar graph of maximum degree 3 admit a smooth complexity 1 layout?

Open Problem 6. Is it NP-hard to decide whether a given planar graph of maximum degree 4 admits a smooth complexity 1 layout?

In Chapter 6, we have introduced two algorithms for monotone drawings of trees. First, we have given a linear-time algorithm that draws any n-vertex tree on the $O(n^{1.5}) \times O(n^{1.5})$ grid in a monotone fashion and close-to-perfect angular resolution. Second, we have shown that any tree admits a strongly monotone drawing. For proper binary trees, the drawing is also strictly convex. Several questions remain open.

Open Problem 7. Does any tree admit a strongly monotone drawing on a grid of polynomial size?

Open Problem 8. Are there biconnected (or triconnected) planar graphs that do not admit a strongly monotone drawing? If yes, can this be tested efficiently?

Open Problem 9. Are our drawings for general trees not just strongly monotone but also convex?

In Chapter 7, we have considered the problem of simultaneously drawing planar graphs with right-angle crossings and a constant number of bends per edge. Our main result was that two planar graphs always admit a RAC simultaneous drawing with at most six bends per edge. Further, we have given betters bounds on the number of bends per edge for more restricted graph classes. All drawings can be computed efficiently and require quadratic area. Some interesting problems remain open.

Open Problem 10. What other non-trivial classes of graphs admit a RACSIM drawing with less than six bends per edge?

Open Problem 11. Can we reduce the number of bends per edge if we do not insist on right-angle crossings, but allow angles that are close to right angles?

Open Problem 12. Given two or more planar graphs on the same set of vertices and a non-negative integer k, can we test whether there is a RACSIM drawing in which each graph is drawn with at most k bends per edge?

Open Problem 13. Can we achieve sub-quadratic area for special subclasses of planar graphs if we increase the number of bends?

In Chapter 8, we have studied IC-planar graphs. We have shown that every IC-planar graph admits a drawing on the $O(n) \times O(n)$ grid that can be computed in linear time. If we restrict the crossings to RAC, then we can compute drawings in exponential area. Both area bounds are asymptotically tight. Deciding whether a given graph is IC-planar is NP-complete even if the rotation system is given. Given, however, a triangulated planar graph, we can efficiently test whether the graph can be made IC-planar by adding a matching. The following questions remain open.

Open Problem 14. Can we efficiently test whether a graph with given rotation system is maximal IC-planar?

Open Problem 15. Can we achieve polynomial-size IC-planar drawings by using crossing angles that are close to right angles?

Open Problem 16. Do larger graph classes that lie in the intersection of the class of 1-planar graphs and the class of RAC graphs?

Bibliography

[AAP+97]   Pankaj K. Agarwal, Boris Aronov, János Pach, Richard Pollack, and Micha Sharir. Quasi-planar graphs have a linear number of edges. *Combinatorica*, 17(1):1–9, 1997. [see page 129]

[ABB+13]   Christopher Auer, Christian Bachmaier, Franz J. Brandenburg, Andreas Gleißner, Kathrin Hanauer, Daniel Neuwirth, and Josef Reislhuber. Recognizing outer 1-planar graphs in linear time. In Stephen K. Wismath and Alexander Wolff, editors, *Proc. 21st Int. Symp. Graph Drawing (GD'13)*, volume 8242 of *Lecture Notes Comput. Sci.*, pages 107–118. Springer-Verlag, 2013. [see page 129]

[ABF+12]   Patrizio Angelini, Giuseppe Di Battista, Fabrizio Frati, Maurizio Patrignani, and Ignaz Rutter. Testing the simultaneous embeddability of two graphs whose intersection is a biconnected or a connected graph. *J. Discrete Algorithms*, 14:150–172, 2012. [see page 109]

[ABGR15]   Christopher Auer, Franz J. Brandenburg, Andreas Gleißner, and Josef Reislhuber. 1-planarity of graphs with a rotation system. *J. Graph Algorithms Appl.*, 19(1):67–86, 2015. [see pages 130, 135, and 136]

[ABK13]   Md. Jawaherul Alam, Franz J. Brandenburg, and Stephen G. Kobourov. Straight-line grid drawings of 3-connected 1-planar graphs. In Stephen K. Wismath and Alexander Wolff, editors, *Proc. 21st Int. Symp. Graph Drawing (GD'13)*, volume 8242 of *Lecture Notes Comput. Sci.*, pages 83–94. Springer-Verlag, 2013. [see pages 129, 132, and 133]

[ABK+14]   Md. Jawaherul Alam, Michael A. Bekos, Michael Kaufmann, Philipp Kindermann, Stephen G. Kobourov, and Alexander Wolff. Smooth orthogonal drawings of planar graphs. In A. Pardo and A. Viola, editors, *Proc. 13th Latin Am. Symp. Theor. Inform. (LATIN'14)*, volume 8392 of *Lecture Notes Comput. Sci.*, pages 144–155. Springer-Verlag, 2014. [see page 5]

[ABKS13]   Evmorfia N. Argyriou, Michael A. Bekos, Michael Kaufmann, and Antonios Symvonis. Geometric RAC simultaneous drawings of graphs. *J. Graph Algorithms Appl.*, 17(1):11–34, 2013. [see page 110]

[ABS12]   Evmorfia N. Argyriou, Michael A. Bekos, and Antonios Symvonis. The straight-line RAC drawing problem is NP-hard. *J. Graph Algorithms Appl.*, 16(2):569–597, 2012. [see page 129]

[ACB+12]   Patrizio Angelini, Enrico Colasante, Giuseppe Di Battista, Fabrizio Frati, and Maurizio Patrignani. Monotone drawings of graphs. *J. Graph Algorithms Appl.*, 16(1):5–35, 2012. [see pages 91, 92, 99, 100, and 105]

Bibliography

[ACG+13] Soroush Alamdari, Timothy M. Chan, Elyot Grant, Anna Lubiw, and Vinayak Pathak. Self-approaching graphs. In Walter Didimo and Maurizio Patrignani, editors, *Proc. 20th Int. Symp. Graph Drawing (GD'12)*, volume 7704 of *Lecture Notes Comput. Sci.*, pages 260–271. Springer-Verlag, 2013. [see page 91]

[Ack09] Eyal Ackerman. On the maximum number of edges in topological graphs with no four pairwise crossing edges. *Discrete Comput. Geom.*, 41(3):365–375, 2009. [see page 129]

[Ack14] Eyal Ackerman. A note on 1-planar graphs. *Discrete Appl. Math.*, 175:104–108, 2014. [see pages 59 and 63]

[ACM89] Esther M. Arkin, Robert Connelly, and Joseph S. B. Mitchell. On monotone paths among obstacles with applications to planning assemblies. In Kurt Mehlhorn, editor, *Proc. 5th Ann. ACM Symp. Comput. Geom. (SoCG'89)*, pages 334–343. ACM, 1989. [see page 91]

[ADF+15] Patrizio Angelini, Giuseppe Di Battista, Fabrizio Frati, Vít Jelínek, Jan Kratochvíl, Maurizio Patrignani, and Ignaz Rutter. Testing planarity of partially embedded graphs. *ACM Trans. Algorithms*, 11(4):32:1–32:42, 2015. [see page 109]

[ADK+13] Patrizio Angelini, Walter Didimo, Stephen Kobourov, Tamara Mchedlidze, Vincenzo Roselli, Antonios Symvonis, and Stephen Wismath. Monotone drawings of graphs with fixed embedding. *Algorithmica*, pages 1–25, 2013. [see page 92]

[AES99] Pankaj K. Agarwal, Alon Efrat, and Micha Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. *SIAM J. Comput.*, 29(3):912–953, 1999. [see page 23]

[AGKN12] Patrizio Angelini, Markus Geyer, Michael Kaufmann, and Daniel Neuwirth. On a tree and a path with no geometric simultaneous embedding. *J. Graph Algorithms Appl.*, 16(1):37–83, 2012. [see page 109]

[Alb08] Michael O. Albertson. Chromatic number, independence ratio, and crossing number. *Ars Math. Contemp.*, 1(1):1–6, 2008. [see page 130]

[APT79] Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Inform. Process. Lett.*, 8(3):121–123, 1979. [see page 144]

[AS92] Noga Alon and Joel Spencer. *The Probabilistic Method*. John Wiley, 1992. [see page 61]

[AT07] Eyal Ackerman and Gábor Tardos. On the maximum number of edges in quasi-planar graphs. *J. Comb. Theory, Series A*, 114(3):563–571, 2007. [see page 129]

[BCD⁺07]   Peter Brass, Eowyn Cenek, Cristian A. Duncan, Alon Efrat, Cesim Erten, Dan P. Ismailescu, Stephen G. Kobourov, Anna Lubiw, and Joseph S. B. Mitchell. On simultaneous planar graph embeddings. *Comput. Geom. Theory Appl.*, 36(2):117–130, 2007. [see pages 109 and 117]

[BCF⁺13]   Michael A. Bekos, Sabine Cornelsen, Martin Fink, Seokhee Hong, Michael Kaufmann, Martin Nöllenburg, Ignaz Rutter, and Antonios Symvonis. Many-to-one boundary labeling with backbones. In Stephen Wismath and Alexander Wolff, editors, *Proc. 21st Int. Sympos. Graph Drawing (GD'13)*, volume 8242 of *Lecture Notes Comput. Sci.*, pages 244–255. Springer-Verlag, 2013. [see page 23]

[BCG⁺14]   Michael A. Bekos, Sabine Cornelsen, Luca Grilli, Seok-Hee Hong, and Michael Kaufmann. On the recognition of fan-planar and maximal outer-fan-planar graphs. In Christian A. Duncan and Antonios Symvonis, editors, *Proc. 22nd Int. Symp. Graph Drawing (GD'14)*, volume 8871 of *Lecture Notes Comput. Sci.*, pages 198–209. Springer-Verlag, 2014. [see page 129]

[BDD⁺14]   Carla Binucci, Emilio Di Giacomo, Walter Didimo, Fabrizio Montecchiani, Maurizio Patrignani, and Ioannis G. Tollis. Fan-planar graphs: Combinatorial properties and complexity results. In Christian A. Duncan and Antonios Symvonis, editors, *Proc. 22nd Int. Symp. Graph Drawing (GD'14)*, volume 8871 of *Lecture Notes Comput. Sci.*, pages 186–197. Springer-Verlag, 2014. [see pages 129 and 130]

[BEG⁺12]   Franz-Josef Brandenburg, David Eppstein, Andreas Gleißner, Michael T. Goodrich, Kathrin Hanauer, and Josef Reislhuber. On the density of maximal 1-planar graphs. In Walter Didimo and Maurizio Patrignani, editors, *Proc. 20th Int. Symp. Graph Drawing (GD'12)*, volume 7704 of *Lecture Notes Comput. Sci.*, pages 327–338. Springer-Verlag, 2012. [see page 129]

[BETT99]   Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice Hall, 1999. [see page 11]

[BF96]   Oliver Bastert and Sándor P. Fekete. Geometrische Verdrahtungsprobleme. Technical Report 96–247, Universität zu Köln, 1996. [see page 23]

[BFK⁺14]   Lukas Barth, Sara Irina Fabrikant, Stephen Kobourov, Anna Lubiw, Martin Nöllenburg, Yoshio Okamoto, Sergey Pupyrev, Claudio Squarcella, Torsten Ueckerdt, and Alexander Wolff. Semantic word cloud representations: Hardness and approximation algorithms. In Alberto Pardo and Alfredo Viola, editors, *Proc. 11th Latin American Symp. Theor. Inform. (LATIN'14)*, volume 8392 of *Lecture Notes Comput. Sci.*, pages 514–525. Springer-Verlag, 2014. [see pages 54, 55, 56, 57, 64, and 67]

[BGPV08]   Adam L. Buchsbaum, Emden R. Gansner, Cecilia Magdalena Procopiuc, and Suresh Venkatasubramanian. Rectangular layouts and contact graphs. *ACM Trans. Algorithms*, 4(1):8:1–8:28, 2008. [see page 54]

Bibliography

[BGR14]   Michael A. Bekos, Martin Gronemann, and Chrysanthi N. Raftopoulou. Two-page book embeddings of 4-planar graphs. In Ernst W. Mayr and Natacha Portier, editors, *Proc. 31st Int. Symp. Theor. Aspects Comput. Sci. (STACS'14)*, volume 25 of *LIPIcs*, pages 137–148. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014. [see page 115]

[BHKN09]   Marc Benkert, Herman J. Haverkort, Moritz Kroll, and Martin Nöllenburg. Algorithms for multi-criteria boundary labeling. *J. Graph Algorithms Appl.*, 13(3):289–317, 2009. [see pages 22, 23, and 25]

[BK79]   Frank Bernhart and Paul C. Kainen. The book thickness of a graph. *J. Comb. Theory, Series B*, 27(3):320–331, 1979. [see pages 111 and 115]

[BK98]   Therese Biedl and Goos Kant. A better heuristic for orthogonal graph drawings. *Comput. Geom. Theory Appl.*, 9(3):159–180, 1998. [see pages 73, 74, 76, and 79]

[BKKS13]   Michael A. Bekos, Michael Kaufmann, Stephen G. Kobourov, and Antonis Symvonis. Smooth orthogonal layouts. In Walter Didimo and Maurizio Patrignani, editors, *Proc. 20th Int. Symp. Graph Drawing (GD'12)*, volume 7704 of *Lecture Notes Comput. Sci.*, pages 150–161. Springer-Verlag, 2013. [see pages 73, 74, 75, 81, 86, and 88]

[BKNS10]   Michael A. Bekos, Michael Kaufmann, Martin Nöllenburg, and Antonios Symvonis. Boundary labeling with octilinear leaders. *Algorithmica*, 57(3):436–461, 2010. [see page 23]

[BKP14]   Lukas Barth, Stephen Kobourov, and Sergey Pupyrev. Experimental comparison of semantic word clouds. In Joachim Gudmundsson and Jyrki Katajainen, editors, *Proc. 13th Int. Symp. Experimental Algorithms (SEA'14)*, volume 8504 of *Lecture Notes Comput. Sci.*, pages 247–258. Springer-Verlag, 2014. [see pages 54 and 55]

[BKPS10]   Michael A. Bekos, Michael Kaufmann, Katerina Potika, and Antonios Symvonis. Area-feature boundary labeling. *Comput. J.*, 53(6):827–841, 2010. [see pages 22 and 36]

[BKR13a]   Thomas Bläsius, Annette Karrer, and Ignaz Rutter. Simultaneous embedding: Edge orderings, relative positions, cutvertices. In Stephen Wismath and Alexander Wolff, editors, *Proc. 21st Int. Symp. Graph Drawing (GD'13)*, volume 8242 of *Lecture Notes Comput. Sci.*, pages 220–231. Springer-Verlag, 2013. [see page 109]

[BKR13b]   Thomas Bläsius, Stephen G. Kobourov, and Ignaz Rutter. Simultaneous embedding of planar graphs. In Roberto Tamassia, editor, *Handbook of Graph Drawing and Visualization*, chapter 11, pages 349–381. CRC Press, 2013. [see page 109]

[BKRS01]   Oleg V. Borodin, Alexandr V. Kostochka, André Raspaud, and Eric Sopena. Acyclic colouring of 1-planar graphs. *Discrete Appl. Math.*, 114(1-3):29–41, 2001. [see page 129]

[BKRW14] Thomas Bläsius, Marcus Krug, Ignaz Rutter, and Dorothea Wagner. Orthogonal graph drawing with flexibility constraints. *Algorithmica*, 68(4):859–885, 2014. [see page 73]

[BKSW07] Michael A. Bekos, Michael Kaufmann, Antonios Symvonis, and Alexander Wolff. Boundary labeling: Models and efficient algorithms for rectangular maps. *Comput. Geom. Theory Appl.*, 36(3):215–236, 2007. [see pages 21, 22, and 23]

[BKV11] Patrick Briest, Piotr Krysta, and Berthold Vöcking. Approximation techniques for utilitarian mechanism design. *SIAM J. Comput.*, 40(6):1587–1622, 2011. [see pages 55 and 57]

[Bor84] O. V. Borodin. Solution of the Ringel problem on vertex-face coloring of planar graphs and coloring of 1-planar graphs. *Metody Diskret. Analiz.*, 41:12–26, 108, 1984. [see page 129]

[BR06] Imre Bárány and Günter Rote. Strictly convex drawings of planar graphs. *Doc. Math.*, 11:369–391, 2006. [see page 93]

[Bra14] Franz J. Brandenburg. 1-visibility representations of 1-planar graphs. *J. Graph Algorithms Appl.*, 18(3):421–438, 2014. [see page 130]

[BS93] Graham Brightwell and Edward R. Scheinerman. Representations of planar graphs. *SIAM J. Discrete Math.*, 6(2):214–229, 1993. [see page 110]

[BvDF+14] Michael A. Bekos, Thomas C. van Dijk, Martin Fink, Philipp Kindermann, Stephen Kobourov, Sergey Pupyrev, Joachim Spoerhase, and Alexander Wolff. Improved approximation algorithms for box contact representations. In Andreas S. Schulz and Dorothea Wagner, editors, *Proc. 22nd Europ. Symp. Algorithms (ESA'14)*, volume 8737 of *Lecture Notes Comput. Sci.*, pages 87–99. Springer-Verlag, September 2014. [see page 4]

[BvDKW15] Michael A. Bekos, Thomas C. van Dijk, Philipp Kindermann, and Alexander Wolff. Simultaneous drawing of planar graphs with right-angle crossings and few bends. In M. Sohel Rahman and Etsuji Tojima, editors, *Proc. 9th Int. Workshop on Algorithms and Computation (WALCOM'15)*, volume 8973 of *Lecture Notes Comput. Sci.*, pages 222–233. Springer-Verlag, February 2015. [see page 7]

[Cc99] Bernard Chazelle and 36 co-authors. The computational geometry impact task force report. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Adv. Discrete and Comput. Geom.*, volume 223, pages 407–463. American Mathematical Society, Providence, RI, 1999. [see page 21]

[CE07] Josiah Carlson and David Eppstein. Trees with convex faces and optimal angles. In Michael Kaufmann and Dorothea Wagner, editors, *Proc. 14th Int. Symp. Graph Drawing (GD'06)*, volume 4372 of *Lecture Notes Comput. Sci.*, pages 77–88. Springer-Verlag, 2007. [see pages 92, 94, 98, 99, and 100]

Bibliography

[CHKK13]   Otfried Cheong, Sariel Har-Peled, Heuna Kim, and Hyo-Sil Kim. On the number of edges of fan-crossing free graphs. In Leizhen Cai, Siu-Wing Cheng, and Tak Wah Lam, editors, *Proc. 24th Int. Symp. Algorithms Comput. (ISAAC'13)*, volume 8283 of *Lecture Notes Comput. Sci.*, pages 163–173. Springer-Verlag, 2013. [see page 129]

[CHKL13]   Timothy M. Chan, Hella-Franziska Hoffmann, Stephen Kiazyk, and Anna Lubiw. Minimum length embedding of planar graphs at fixed vertex locations. In Stephen K. Wismath and Alexander Wolff, editors, *Proc. 21st Int. Symp. Graph Drawing (GD'13)*, volume 8242 of *Lecture Notes Comput. Sci.*, pages 376–387. Springer-Verlag, 2013. [see page 23]

[CK05]   Chandra Chekuri and Sanjeev Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM J. Comput.*, 35(3):713–728, 2005. [see pages 17 and 54]

[CKR06]   Reuven Cohen, Liran Katzir, and Danny Raz. An efficient approximation for the generalized assignment problem. *Inform. Process. Lett.*, 100(4):162–166, 2006. [see pages 56 and 62]

[CLRS09]   Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, third edition edition, 2009. [see page 11]

[CM13]   Sergio Cabello and Bojan Mohar. Adding one edge to planar graphs makes crossing number and 1-planarity hard. *SIAM J. Comput.*, 42(5):1803–1829, 2013. [see pages 15 and 16]

[CNP83]   Gérard Cornuéjols, Denis Naddef, and William R. Pulleyblank. Halin graphs and the travelling salesman problem. *Math. Program.*, 26(3):287–294, 1983. [see page 115]

[Coo71]   Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *Proc. 3rd Ann. ACM Symp. Theory Comput. (STOC'71)*, pages 151–158. ACM, 1971. [see page 16]

[CP95]   Marek Chrobak and Thomas H. Payne. A linear-time algorithm for drawing a planar graph on a grid. *Inform. Process. Lett.*, 54(4):241–246, 1995. [see page 145]

[CvL$^+$11]   Sergio Cabello, Marc van Kreveld, Giuseppe Liotta, Henk Meijer, Bettina Speckmann, and Kevin Verbeek. Geometric simultaneous embeddings of a graph and a matching. *J. Graph Algorithms Appl.*, 15(1):79–96, 2011. [see pages 109 and 121]

[CWL$^+$10]   Weiwei Cui, Yingcai Wu, Shixia Liu, Furu Wei, Michelle X. Zhou, and Huamin Qu. Context-preserving dynamic word cloud visualization. *Comput. Graph. Appl.*, 30(6):42–53, 2010. [see page 54]

[DDEL14]   Emilio Di Giacomo, Walter Didimo, Peter Eades, and Giuseppe Liotta. 2-layer right angle crossing drawings. *Algorithmica*, 68(4):954–997, 2014. [see page 129]

[DDLM12]   Emilio Di Giacomo, Walter Didimo, Giuseppe Liotta, and Fabrizio Montecchiani. *h*-quasi planar drawings of bounded treewidth graphs in linear area. In Martin Charles Golumbic, Michal Stern, Avivit Levy, and Gila Morgenstern, editors, *Proc. 38th Int. Workshop Graph-Theor. Concepts Comput. Sci. (WG'12)*, volume 7551 of *Lecture Notes Comput. Sci.*, pages 91–102. Springer-Verlag, 2012. [see page 129]

[DDLM13]   Emilio Di Giacomo, Walter Didimo, Giuseppe Liotta, and Fabrizio Montecchiani. Area requirement of graph drawings with few crossings per edge. *Comput. Geom.*, 46(8):909–916, 2013. [see page 129]

[DDvK⁺09]   Emilio Di Giacomo, Walter Didimo, Marc van Kreveld, Giuseppe Liotta, and Bettina Speckmann. Matched drawings of planar graphs. *J. Graph Algorithms Appl.*, 13(3):423–445, 2009. [see page 121]

[DE12]   Hooman Reisi Dehkordi and Peter Eades. Every outer-1-plane graph has a right angle crossing drawing. *Int. J. Comput. Geom. Appl.*, 22(06):543–557, 2012. [see page 144]

[DEG⁺12]   Christian A. Duncan, David Eppstein, Michael T. Goodrich, Stephen G. Kobourov, and Martin Nöllenburg. Lombardi drawings of graphs. *J. Graph Algorithms Appl.*, 16(1):85–108, 2012. [see page 73]

[DEG⁺13]   Christian A. Duncan, David Eppstein, Michael T. Goodrich, Stephen G. Kobourov, and Martin Nöllenburg. Drawing trees with perfect angular resolution and polynomial area. *Discrete Comput. Geom.*, 49(2):157–182, 2013. [see page 73]

[DEL11]   Walter Didimo, Peter Eades, and Giuseppe Liotta. Drawing graphs with right angle crossings. *Theoretical Computer Science*, 412(39):5156–5166, 2011. [see pages 129 and 150]

[dFPP88]   Hubert de Fraysseix, János Pach, and Richard Pollack. Small sets supporting fary embeddings of planar graphs. In Janos Simon, editor, *Proc. 20th Ann. ACM Symp. Theory Comput. (STOC'88)*, pages 426–433. ACM, 1988. [see page 147]

[dFPP90]   Hubert de Fraysseix, János Pach, and Richard Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990. [see pages 144 and 145]

[Dha10]   Raghavan Dhandapani. Greedy drawings of triangulations. *Discrete Comput. Geom.*, 43(2):375–392, 2010. [see page 91]

[Did13]   Walter Didimo. Density of straight-line 1-planar graph drawings. *Inform. Process. Lett.*, 113(7):236–240, 2013. [see page 129]

[DL12]   Walter Didimo and Giuseppe Liotta. The crossing angle resolution in graph drawing. In János Pach, editor, *Thirty Essays on Geometric Graph Theory*. Springer, 2012. [see page 129]

Bibliography

[DLT84]   Danny Dolev, Tom Leighton, and Howard Trickey. Planar embedding of planar graphs. *Adv. Comput. Res.*, 2:147–161, 1984. [see page 133]

[DMS05]   Tim Dwyer, Kim Marriott, and Peter J. Stuckey. Fast node overlap removal. In Patrick Healy and Nikola S. Nikolov, editors, *Proc. 13th Int. Symp. Graph Drawing (GD'05)*, volume 3843 of *Lecture Notes Comput. Sci.*, pages 153–164. Springer-Verlag, 2005. [see page 54]

[EBGJ+08]   Alejandro Estrella-Balderrama, Elisabeth Gassner, Michael Jünger, Merijam Percan, Marcus Schaefer, and Michael Schulz. Simultaneous geometric graph embeddings. In Seok-Hee Hong, Takao Nishizeki, and Wu Quan, editors, *Proc. 16th Int. Symp. Graph Drawing (GD'08)*, volume 4875 of *Lecture Notes Comput. Sci.*, pages 280–290. Springer-Verlag, 2008. [see page 109]

[EHK+13]   Peter Eades, Seok-Hee Hong, Naoki Katoh, Giuseppe Liotta, Pascal Schweitzer, and Yusuke Suzuki. A linear time algorithm for testing maximal 1-planarity of graphs with a rotation system. *Theoretical Computer Science*, 513:65–76, 2013. [see page 129]

[EK05a]   Cesim Erten and Stephen G. Kobourov. Simultaneous embedding of a planar graph and its dual on the grid. *Theory Comput. Syst.*, 38(3):313–327, 2005. [see page 110]

[EK05b]   Cesim Erten and Stephen G. Kobourov. Simultaneous embedding of planar graphs with few bends. *J. Graph Algorithms Appl.*, 9(3):347–364, 2005. [see pages 109 and 112]

[EKL+13]   William S. Evans, Michael Kaufmann, William Lenhart, Giuseppe Liotta, Tamara Mchedlidze, and Stephen K. Wismath. Bar 1-visibility graphs and their relation to other nearly planar graphs. Arxiv report, 2013. Available at http://arxiv.org/abs/1312.5520. [see page 130]

[EL13]   Peter Eades and Giuseppe Liotta. Right angle crossing graphs and 1-planarity. *Discrete Appl. Math.*, 161(7-8):961–969, 2013. [see pages 8, 129, 130, 131, and 144]

[EMSV12]   David Eppstein, Elena Mumford, Bettina Speckmann, and Kevin Verbeek. Area-universal and constrained rectangular layouts. *SIAM J. Comput.*, 41(3):537–564, 2012. [see page 54]

[ER04]   Günes Erkan and Dragomir R. Radev. Lexrank: Graph-based lexical centrality as salience in text summarization. *J. Artif. Int. Res.*, 22(1):457–479, 2004. [see page 55]

[Fel04]   Stefan Felsner. *Geometric Graphs and Arrangements*. Vieweg Verlag, 2004. [see page 115]

[Fel13]   Stefan Felsner. Rectangle and square representations of planar graphs. In János Pach, editor, *Thirty Essays on Geometric Graph Theory*, pages 213–248. Springer-Verlag, 2013. [see page 54]

[FGMS11]   Lisa Fleischer, Michel X. Goemans, Vahab Mirrokni, and Maxim Sviridenko. Tight approximation algorithms for maximum separable assignment problems. *Math. Oper. Res.*, 36(3):416–431, 2011. [see pages 17, 54, 55, and 68]

[FHH⁺93]   Michael Formann, Torben Hagerup, James Haralambides, Michael Kaufmann, Frank Thomson Leighton, Antonios Symvonis, Emo Welzl, and Gerhard J. Woeginger. Drawing graphs in the plane with high resolution. *SIAM J. Comput.*, 22(5):1035–1052, 1993. [see page 2]

[FHS⁺12]   Martin Fink, Jan-Henrik Haunert, André Schulz, Joachim Spoerhase, and Alexander Wolff. Algorithms for labeling focus regions. *IEEE Trans. Visual. Comput. Graphics*, 18(12):2583–2592, 2012. [see page 23]

[FMC96]   Herbert Freeman, Sean Marrinan, and Hitesh Chitalia. Automated labeling of soil survey maps. In *Proc. ASPRS-ACSM Ann. Conv., Baltimore*, volume 1, pages 51–59, 1996. [see page 21]

[FPS13]   Jacob Fox, János Pach, and Andrew Suk. The number of edges in *k*-quasi-planar graphs. *SIAM J. Discrete Math.*, 27(1):550–561, 2013. [see page 129]

[Fre87]   Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.*, 16(6):1004–1022, 1987. [see pages 55 and 65]

[GB07]   Alexander Grigoriev and Hans L. Bodlaender. Algorithms for graphs embeddable with few crossings per edge. *Algorithmica*, 49(1):1–11, 2007. [see page 16]

[GH10]   Emden R. Gansner and Yifan Hu. Efficient, proximity-preserving node overlap removal. *J. Graph Algorithms Appl.*, 14(1):53–74, 2010. [see page 54]

[GHKR14]   Luca Grilli, Seok-Hee Hong, Jan Kratochvíl, and Ignaz Rutter. Drawing simultaneously embedded graphs with few bends. In Christian Duncan and Antonios Symvonis, editors, *Proc. 22nd Int. Symp. Graph Drawing (GD'14)*, volume 8871 of *Lecture Notes Comput. Sci.*, pages 40–51. Springer-Verlag, 2014. [see page 109]

[GHN11]   Andreas Gemsa, Jan-Henrik Haunert, and Martin Nöllenburg. Boundary-labeling algorithms for panorama images. In Isabel F. Cruz, Divyakant Agrawal, Christian S. Jensen, Eyal Ofek, and Egemen Tanin, editors, *Proc. 19th ACM SIGSPATIAL Int. Conf. Adv. Geogr. Inform. Syst. (ACM-GIS'11)*, pages 289–298. ACM, 2011. [see page 23]

[Gib85]   Alan Gibbons. *Algorithmic Graph Theory*. Camb. Univ. Press, 1985. [see page 11]

[GJ78]   Michael R. Garey and David S. Johnson. "strong" np-completeness results: Motivation, examples, and implications. *J. ACM*, 25(3):499–508, 1978. [see page 15]

[GJ79]   Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979. [see page 15]

Bibliography

[GJ83]   Michael R. Garey and David S. Johnson. Crossing number is np-complete. *SIAM J. Algebraic and Discrete Methods*, 4(3):312–316, 1983. [see pages 14 and 134]

[GMPP91] P. Gritzmann, B. Mohar, J. Pach, and R. Pollack. Embedding a planar triangulation with vertices at specified positions. *Amer. Math. Mon.*, 98:165–166, 1991. [see page 23]

[GT94]   Ashim Garg and Roberto Tamassia. Planar drawings and angular resolution: Algorithms and bounds. In Jan van Leeuwen, editor, *Proc. 2nd Ann. Europ. Symp. Alg. (ESA'94)*, volume 855 of *Lecture Notes Comput. Sci.*, pages 12–23. Springer-Verlag, 1994. [see page 2]

[GT01]   Ashim Garg and Roberto Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM J. Comput.*, 31(2):601–625, 2001. [see page 73]

[Hea85]   Lenwood S. Heath. *Algorithms for Embedding Graphs in Books*. PhD thesis, University of North Carolina, Chapel Hill, 1985. [see page 115]

[Hea87]   Lenwood S. Heath. Embedding outerplanar graphs in small books. *SIAM J. Algebraic and Discrete Methods*, 8(2):198–218, 1987. [see page 116]

[HEK⁺14] Seok-Hee Hong, Peter Eades, Naoki Katoh, Giuseppe Liotta, Pascal Schweitzer, and Yusuke Suzuki. A linear-time algorithm for testing outer-1-planarity. *Algorithmica*, pages 1–22, 2014. [see page 129]

[HELP12] Seok-Hee Hong, Peter Eades, Giuseppe Liotta, and Sheung-Hung Poon. Fáry's theorem for 1-planar graphs. In Joachim Gudmundsson, Julián Mestre, and Taso Viglas, editors, *Proc. 18th Ann. Int. Conf. Comput. Combin. (COCOON'12)*, volume 7434 of *Lecture Notes Comput. Sci.*, pages 335–346. Springer-Verlag, 2012. [see pages 129 and 130]

[HHE08]   Weidong Huang, Seok-Hee Hong, and Peter Eades. Effects of crossing angles. In *Proc. IEEE Pacifc Visual. Symp. (PacifcVis'08)*, pages 41–46. IEEE Computer Society, 2008. [see page 2]

[Hir75]   Daniel S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, 18(6):341–343, 1975. [see pages 33 and 34]

[HJL13]   Bernhard Haeupler, Krishnam Raju Jampani, and Anna Lubiw. Testing simultaneous planarity when the common graph is 2-connected. *J. Graph Algorithms Appl.*, 17(3):147–171, 2013. [see page 109]

[HMS96]   S. Louis Hakimi, John Mitchem, and Edward F. Schmeichel. Star arboricity of graphs. *Discrete Math.*, 149(1–3):93–98, 1996. [see pages 57 and 63]

[HR14]   Md. Iqbal Hossain and Md. Saidur Rahman. Monotone grid drawings of planar graphs. In Jianer Chen, John E. Hopcroft, and Jianxin Wang, editors, *Proc. 8th Int. Workshop Front. Algorithmics (FAW'14)*, volume 8497 of *Lecture Notes Comput. Sci.*, pages 105–116. Springer-Verlag, 2014. [see page 92]

[HS99]   Christian Haslinger and Peter F. Stadler. Rna structures with pseudo-knots: Graph-theoretical, combinatorial, and statistical properties. *Bull. Math. Biol.*, 61(3):437–467, 1999. [see page 115]

[HW79]   Godfrey H. Hardy and Edward M. Wright. *An Introduction to the Theory of Numbers*. Oxford Univ. Press, 5th edition, 1979. [see pages 93 and 94]

[IKL95]   Christian Icking, Rolf Klein, and Elmar Langetepe. Self-approaching curves. In *Math. Proc. Camb. Philos. Soc.*, volume 125, pages 441–453. Camb. Univ. Press, 1995. [see page 92]

[Kam06]   Frank Kammer. Simultaneous embedding with two bends per edge in polynomial area. In Lars Arge and Rusins Freivalds, editors, *Proc. 10th Scand. Workshop Algorithm Theory (SWAT'06)*, volume 4059 of *Lecture Notes Comput. Sci.*, pages 255–267. Springer-Verlag, 2006. [see page 109]

[Kan96]   Goos Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16(1):4–32, 1996. [see page 81]

[Kar72]   Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Complexity of Computer Computations*, IBM Res. Symp. Series, pages 85–103. Springer-Verlag, 1972. [see page 16]

[KKRW10] B. Katz, M. Krug, I. Rutter, and A. Wolff. Manhattan-geodesic embedding of planar graphs. In David Eppstein and Emden R. Gansner, editors, *Proc. 17th Int. Symp. Graph Drawing (GD'09)*, volume 5849 of *Lecture Notes Comput. Sci.*, pages 207–218. Springer-Verlag, 2010. [see page 23]

[KM13]   Vladimir P. Korzhik and Bojan Mohar. Minimal obstructions for 1-immersions and hardness of 1-planarity testing. *J. Graph Theory*, 72(1):30–71, 2013. [see pages 129 and 130]

[KNR+13]   Philipp Kindermann, Benjamin Niedermann, Ignaz Rutter, Marcus Schaefer, André Schulz, and Alexander Wolff. Two-sided boundary labeling with adjacent sides. In F. Dehne, R. Solis-Oba, and J.-R. Sack, editors, *Proc. 13th Int. Algorithms Data Struct. Symp. (WADS'13)*, volume 8037 of *Lecture Notes Comput. Sci.*, pages 463–474. Springer-Verlag, 2013. [see page 4]

[KO07]   Paul C. Kainen and Shannon Overbay. Extension of a theorem of whitney. *Appl. Math. Lett.*, 20(7):835–837, 2007. [see page 115]

[Koe36]   Paul Koebe. Kontaktprobleme der konformen Abbildung. *Berichte d. math.-phys. Kl. d. Sächs. Akad. d. Wissenschaften zu Leipzig*, 88(2):141–164, 1936. [see page 2]

[Kro67]   Melven R. Krom. The decision problem for a class of first-order formulas in which all disjunctions are binary. *Math. Logic Quarterly*, 13(1-2):15–20, 1967. [see page 16]

Bibliography

[KS10]   Daniel Král and Ladislav Stacho.   Coloring plane graphs with independent crossings. *J. Graph Theory*, 64(3):184–205, 2010. [see page 130]

[KSSW14]   Philipp Kindermann, André Schulz, Joachim Spoerhase, and Alexander Wolff. On monotone drawings of trees. In C. Duncan and A. Symvonis, editors, *Proc. 22nd Int. Symp. Graph Drawing (GD'14)*, volume 8871 of *Lecture Notes Comput. Sci.*, pages 488–500. Springer-Verlag, 2014. [see page 6]

[KU14]   Michael Kaufmann and Torsten Ueckerdt. The density of fan-planar graphs. Arxiv report, 2014. Available at http://arxiv.org/abs/1403.6184. [see page 129]

[KW01]   Michael Kaufmann and Dorothea Wagner, editors. *Drawing Graphs: Methods and Models*, volume 2025 of *Lecture Notes Comput. Sci.* Springer-Verlag, 2001. [see page 11]

[KW02]   Michael Kaufmann and Roland Wiese. Embedding vertices at points: Few bends suffice for planar graphs. *J. Graph Algorithms Appl.*, 6(1):115–129, 2002. [see page 112]

[Kyn09]   Jan Kynčl. Enumeration of simple complete topological graphs. *Electr. J. Comb.*, 30(7):1676–1685, 2009. [see page 135]

[Lei80]   Charles E. Leiserson. Area-efficient graph layouts (for VLSI). In *Proc. 21st Annu. IEEE Symp. Foundat. Comput. Sci. (FOCS'80)*, pages 270–281. IEEE Computer Society, 1980. [see page 73]

[Li02]   Hang Li. Word clustering and disambiguation based on co-occurrence data. *J. Nat. Lang. Eng.*, 8(1):25–42, 2002. [see page 53]

[Lin10]   Chun-Cheng Lin. Crossing-free many-to-one boundary labeling with hyper-leaders. In *Proc. IEEE Pacific Visual. Symp. (PacificVis'10)*, pages 185–192. IEEE Computer Society, 2010. [see page 23]

[Lio14]   Giuseppe Liotta. Graph drawing beyond planarity: some results and open problems. In Stefano Bistarelli and Andrea Formisano, editors, *Proc. 15th Ital. Conf. Theoret. Comput. Sci. (ICTCS'14)*, volume 1231 of *CEUR Workshop Proc.*, pages 3–8. CEUR-WS.org, 2014. [see page 129]

[LKY08]   Chun-Cheng Lin, Hao-Jen Kao, and Hsu-Chun Yen. Many-to-one boundary labeling. *J. Graph Algorithms Appl.*, 12(3):319–356, 2008. [see page 23]

[LM10]   Tom Leighton and Ankur Moitra. Some results on greedy embeddings in metric spaces. *Discrete Comput. Geom.*, 44(3):686–705, 2010. [see page 91]

[LMM⁺95]   Thomas M. Liebling, François Margot, D. Müller, Alain Prodon, and L. Stauffer. Disjoint paths in the plane. *ORSA J. Comput.*, 7(1):84–88, 1995. [see page 23]

[LMS98]   Yanpei Liu, Aurora Morgana, and Bruno Simeone. A linear algorithm for 2-bend embeddings of planar graphs in the two-dimensional grid. *Discrete Appl. Math.*, 81(1–3):69–91, 1998. [see pages 73, 74, and 75]

[Mor80]   Joel L. Morrison. Computer technology and cartographic change. In D.R.F. Taylor, editor, *The Computer in Contemporary Cartography*. Johns Hopkins University Press, 1980. [see page 21]

[NB79]   Takao Nishizeki and Ilker Baybars. Lower bounds on the cardinality of the maximum matchings of planar graphs. *Discrete Math.*, 28(3):255–267, 1979. [see page 66]

[NC08]   Takao Nishizeki and Norishige Chiba. *Planar Graphs: Theory and Algorithms*, chapter Chapter 10. Hamiltonian Cycles, pages 171–184. Dover Books Math. Courier Dover Pub., 2008. [see page 115]

[NP13]   Martin Nöllenburg and Roman Prutkin. Euclidean greedy drawings of trees. In Hans L. Bodlaender and Giuseppe F. Italiano, editors, *Proc. 21st Europ. Symp. Algorithms (ESA'13)*, volume 8125 of *Lecture Notes Comput. Sci.*, pages 767–778. Springer-Verlag, 2013. [see page 91]

[NPR13]   Martin Nöllenburg, Roman Prutkin, and Ignaz Rutter. Edge-weighted contact representations of planar graphs. *J. Graph Algorithms Appl.*, 17(4):441–473, 2013. [see page 54]

[NPR14]   Martin Nöllenburg, Roman Prutkin, and Ignaz Rutter. On self-approaching and increasing-chord drawings of 3-connected planar graphs. Arxiv report, 2014. Available at http://arxiv.org/abs/1409.0315. [see page 99]

[NPS10]   Martin Nöllenburg, Valentin Polishchuk, and Mikko Sysikaski. Dynamic one-sided boundary labeling. In *Proc. 18th ACM SIGSPATIAL Int. Symp. Adv. Geogr. Inform. Syst. (ACM-GIS'10)*, pages 310–319. ACM, 2010. [see page 23]

[NR04]   Takao Nishizeki and Md. Saidur Rahman. *Planar Graph Drawing*, volume 12 of *Lecture Notes Comput. Sci.* World Sci. Pub., 2004. [see page 11]

[NW64]   Crispin St. John Alvah Nash-Williams. Decomposition of finite graphs into forests. *J. London Math. Soc.*, 1(1):12–12, 1964. [see pages 59 and 115]

[PT97]   János Pach and Gáza Tóth. Graphs drawn with few crossings per edge. *Combinatorica*, 17(3):427–439, 1997. [see page 129]

[PTT⁺12]   Fernando Vieira Paulovich, Franklina M. B. Toledo, Guilherme P. Telles, Rosane Minghim, and Luis Gustavo Nonato. Semantic wordification of document collections. *Comput. Graph. Forum*, 31(3):1145–1153, 2012. [see page 54]

[Rai34]   Erwin Raisz. The rectangular statistical cartogram. *Geogr. Review*, 24(3):292–296, 1934. [see page 54]

[RCS86]   Raghunath Raghavan, James Cohoon, and Sartaj Sahni. Single bend wiring. *J. Algorithms*, 7(2):232–257, 1986. [see page 23]

[Rin65]   Gerhard Ringel. Ein Sechsfarbenproblem auf der Kugel. *Abh. aus dem Math. Seminar der Univ. Hamburg*, 29(1-2):107–117, 1965. [see page 129]

Bibliography

[RRP$^+$03]   Ananth Rao, Sylvia Ratnasamy, Christos H. Papadimitriou, Scott Shenker, and Ion Stoica. Geographic routing without location information. In David B. Johnson, Anthony D. Joseph, and Nitin H. Vaidya, editors, *Proc. 9th Ann. Int. Conf. Mob. Comput. Netw. (MOBICOM'03)*, pages 96–108. ACM, 2003. [see page 91]

[Sch13]   Marcus Schaefer. Toward a theory of planarity: Hanani-tutte and planarity variants. *J. Graph Algorithms Appl.*, 17(4):367–440, 2013. [see page 109]

[See97]   Jochen Seemann. Extending the Sugiyama algorithm for drawing UML class diagrams: Towards automatic layout of object-oriented software diagrams. In Giuseppe Di Battista, editor, *Proc. 5th Int. Symp. Graph Drawing (GD'97)*, volume 1353 of *Lecture Notes Comput. Sci.*, pages 415–424. Springer-Verlag, 1997. [see page 73]

[SFK11]   Sven Schneider, Jan-Ruben Fischer, and Reinhard König. Rethinking automated layout design: Developing a creative evolutionary design method for the layout problems in architecture and urban design. In John S. Gero, editor, *Proc. 4th Int. Conf. Design Comput. Cognition (DCC'10)*, pages 367–386. Springer-Verlag, 2011. [see page 73]

[Syl78]   James J. Sylvester. On an application of the new atomic theory to the graphical representation of the invariants and covariants of binary quantics. *Amer. J. Math., Pure and Applied*, 1(1):64–90, 1878. [see page 1]

[Tam87]   R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987. [see page 73]

[Tam13]   Roberto Tamassia, editor. *Handbook of Graph Drawing and Visualization*, volume 81 of *Discrete Math. Appl.* Chapman & Hall/CRC, 2013. [see page 11]

[Tho88]   Carsten Thomassen. Rectilinear drawings of graphs. *J. Graph Theory*, 12(3):335–341, 1988. [see pages 129 and 131]

[Tru93]   Richard J. Trudeau. *Introduction to Graph Theory*. New York: Dover Pub, 1993. [see page 11]

[Val98]   Pavel Valtr. On geometric graphs with no k pairwise parallel edges. *Discrete Comput. Geom.*, 19(3):461–469, 1998. [see page 129]

[Vaz03]   Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2003. [see page 11]

[VBSW83]   R Von Bodendiek, H Schumacher, and K Wagner. Bemerkungen zu einem sechsfarbenproblem von g. ringel. *Abh. Math. Semin. Univ. Hambg.*, 53(1):41–52, 1983. [see page 129]

[vK11]   Marc van Kreveld. The quality ratio of rac drawings and planar drawings of planar graphs. In Ulrik Brandes and Sabine Cornelsen, editors, *Proc. 18th Int. Symp. Graph Drawing (GD'11)*, volume 6502 of *Lecture Notes Comput. Sci.*, pages 371–376. Springer-Verlag, 2011. [see page 2]

[vKSW99] Marc van Kreveld, Tycho Strijk, and Alexander Wolff. Point labeling with sliding labels. *Comput. Geom. Theory Appl.*, 13:21–47, 1999. [see page 21]

[VWF09]   Fernanda B. Viégas, Martin Wattenberg, and Jonathan Feinberg. Participatory visualization with Wordle. *IEEE Trans. Visual. Comput. Graphics*, 15(6):1137–1144, 2009. [see pages 53 and 54]

[Wei13]   Severin Weiland. Der Koalitionsvertrag im Schnellcheck (Quick overview of the [German] coalition agreement). Spiegel Online, www.spiegel.de/politik/deutschland/was-der-koalitionsvertrag-deutschland-bringt-a-935856.html. Click on "Fotos", 27 November 2013. [see page 53]

[Wig82]   Avi Wigderson. The complexity of the hamiltonian circuit problem for maximal planar graphs. Technical Report TR-298, EECS Department, Princeton University, 1982. [see page 115]

[WPW⁺11] Yingcai Wu, Thomas Provan, Furu Wei, Shixia Liu, and Kwan-Liu Ma. Semantic-preserving word clouds by seam carving. *Comput. Graph. Forum*, 30(3):741–750, 2011. [see page 54]

[WS11]   David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Camb. Univ. Pres, 2011. [see page 11]

[Zha14]   Xin Zhang. Drawing complete multipartite graphs on the plane with restrictions on crossings. *Acta Math. Sinica*, 30(12):2045–2053, 2014. [see pages 130 and 153]

[ZL13]   Xin Zhang and Guizhen Liu. The structure of plane graphs with independent crossings and its applications to coloring problems. *Central Europ. J. Math.*, 11(2):308–321, 2013. [see pages 130, 133, and 150]

[Zor97]   Steven Zoraster. Practical results using simulated annealing for point feature label placement. *Cartogr. GIS*, 24(4):228–238, 1997. [see page 21]

In the present time, increasing amounts of data becomes available to applications for all kinds. Data that describes relationships between objects is frequently modelled by a graph. For a human being to efficiently process the data, it is of common interest to visualize structural information. The field of graph drawing deals with the visualization of graphs and networks in a way to convey the information quickly and to be aesthetically appealing.

This book deals with angular schematization in graph drawing, that is, computing the layout of a graph under angular restrictions. It considers several different drawing styles and shows how to compute drawings that help the reader to comprehend the underlying data.