Schematics of Graphs and Hypergraphs

Dissertation

der Mathematisch- und Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Dipl.-Inform. Till Martin Bruckdorfer
aus Leer/Ostfriesland

Tübingen
2015

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	17.12.2015
Dekan:	Prof. Dr. Wolfgang Rosenstiel
1. Berichterstatter:	Prof. Dr. Michael Kaufmann
2. Berichterstatter:	Prof. Dr. Alexander Wolff

Acknowledgements

The first person I want to thank is Prof. Dr. Kaufmann for his support. He introduced me to graph drawing, gave me the opportunity to join his working group and to attend several workshops and conferences. He was available for all questions and discussions and gave me inspiration for my research. I also thank Prof. Dr. Alexander Wolff for being co-reviewer and his invitation to two great, helpful and inspiring PhD-workshops in Würzburg. For financial support my gratitude goes to the EuroGIGA project GraDR 10-EuroGIGA-OP-003.

I would also like to thank all my co-authors, i.e., Patrizio Angelini, Michael A. Bekos, Sabine Cornelsen, Stefan Felsner, Carsten Gutwenger, Michael Kaufmann, Stephen G. Kobourov, Tamara Mchedlidze, Fabrizio Montecchiani, Martin Nöllenburg, Sergey Pupyrev, Chrysanthi N. Raftopoulou and Alexander Wolff. I thank Christian Zielke for providing a small editor tool based on yED [193] that supports PEDs, Ferran Hurtado and Yoshio Okamoto for invaluable pointers to results in discrete geometry, Emilio Di Giacomo, Antonios Symvonis, Henk Meijer, Ulrik Brandes, and Gašper Fijavž for helpful hints and intense discussions, Jarek Byrka for the link between ink maximization and MIS, and Thomas van Dijk for figures and implementations. I also thank my bachelor students Andreas Lauer and Simon Leibssle for their contribution due to their bachelor theses.

For enjoyable coffee breaks and helpful discussions I thank my colleagues and visitors Patrizio Angelini, Michael A. Bekos, Philip Effinger, Andreas Gerasch, Niklas Heinsohn, Stephen G. Kobourov, Stephan Kottler, Robert Krug, Tamara Mchedlidze, Fabrizio Montecchiani, Sergey Pupyrev, Vincenzo Roselli, and Christian Zielke. Thanks also for joining the running activities and establishing the “Italian dinner” tradition in Tübingen. I also thank all proof-readers of this thesis.

I particularly thank Dunja for all her support, as well as my parents.

Zusammenfassung

Visualisierung von Informationen ist unerlässlich, wenn die zugrunde liegenden Daten komplex sind und das menschliche Gehirn trotzdem den Überblick behalten will. Daher gibt es viele Techniken der Visualisierung um wichtige Informationen der Daten in den Vordergrund zu stellen. In dieser Arbeit beschäftigen wir uns mit Visualisierungen, wenn der Datensatz der Informationen in Form eines Graphen gegeben ist, d.h. Objekte repräsentiert mit bilateralen Beziehungen. Dabei ignorieren wir kontext-spezifische Informationen und konzentrieren uns ausschließlich auf die strukturelle Visualisierung, also das Zeichnen von Graphen oder deren Verallgemeinerung: Hypergraphen.

Im Bereich des Graphenzeichnens ist das Ziel die Erstellung von ästhetischen Zeichnungen des Graphen. Man befasst sich mit der automatischen Erstellung von Zeichnungen ausgehend von einem Graphen als Eingabe für einen Algorithmus. Wir sind stets daran interessiert, welche Graphen als Eingabe erlaubt sind (Charakterisierung) und wie groß der Zeitaufwand des Testens eines Graphen bzw. des Erstellens einer Zeichnung ist (Komplexitätsklasse). Die durch Charakterisierung entstehenden Restriktionen an Graphen ergeben sich meist aus geometrischen Einschränkungen, die auf die Struktur der Graphen übertragen werden.

Wir untersuchen zwei Zeichenmodelle und bestimmen Graphenklassen, welche Zeichnungen in diesen Modellen erlauben. Dabei gehen wir auch auf Varianten ein, wo Knotenpositionen bereits festgelegt sind.

Im bekanntesten Modell, der traditionellen geradlinigen Zeichnung, werden wir Punkte vorab definieren, den Punkten Knoten zuweisen und trotz der Einschränkung auf eine kleine Menge von Punkten eine Zeichnung ohne Kantenkreuzungen erhalten.

In dieser Dissertation werden zwei unterschiedliche Zeichenmodelle zur Visualisierung von Graphen vorgestellt, sogenannte PED-Zeichnungen und Bus-Zeichnungen, die beide auch mit festen Knotenpositionen betrachtet werden, sowie ein Einbettungsproblem als Brücke zu beiden Modellen, welches sich damit befasst, ob sich Graphen auf einer vordefinierten Punktemenge zeichnen lassen. Beide Zeichenmodelle existierten bereits im Vorfeld dieser Arbeit, allerdings war kein formales Konzept bekannt, so wie eine Charakterisierung der Graphklasse, die solche Zeichnungen erlaubt, sowie Komplexitätsergebnisse für Test- und Konstruktionsalgorithmen. Wir extrahieren Eigenschaften für beide Zeichenmodelle, die eine aesthetische Zeichnung erlauben.

Für PED-Zeichnungen fokussieren wir uns auf 1/4-SHPEDs, also zugrunde liegende geradlinige Zeichnungen, in welchen bei jeder Kante nur ein viertel zu Beginn und ein viertel des Endes der Kante gezeichnet wird und die übrig gebliebenen Stummel kreuzungsfrei bleiben. Auch 1-bend SHOPEDs werden

untersucht, welche das Pandent von 1/4-SHPEDs bezüglich der orthogonalen Zeichenkonvention mit einem Knick pro Kante ist. Wir untersuchen ferner die Existenz und Konstruktion von PED-Zeichnungen basierend auf gradlinigen Zeichnungen bei denen Knotenpositionen festgelegt sind. Als ncPED werden PED-Zeichnungen auf Basis von geradlinigen Zeichnungen bezeichnet, wenn alle Kanten fast vollständig gezeichnet werden können, und maxSPED ist eine PED-Zeichnung auf Basis einer gradlinigen Zeichnung, wenn ein signifikanter Teil nicht gezeichnet werden konnte, aber die Länge aller Stummel maximal ist, bei gleicher Länge der beiden Stummel pro Kante.

Bei Bus-Zeichnungen konzentrieren wir uns ebenfalls auf planare (kreuzungsfreie) Zeichungen, allerdings für Hypergraphen, d.h. eine Verallgemeinerung von Graphen, bei der Hyperkanten Mengen entsprechen. Bus-Zeichnungen werden dadurch charakterisiert, dass Hyperkanten als fette Strecken gezeichnet werden und deren Elemente als orthogonal mit ihr verbundene Punkte. Wir bezeichnen Bus-Zeichnungen mit horizontalen Strecken als 1-dimensional und Bus-Zeichnungen mit horizontalen und vertikalen Strecken als 2-dimensional. Letztere charakterisieren wir für den planaren Fall und untersuchen die Komplexität der Konstruktion einer 2-dimensionalen planaren Bus-Zeichnung. Untersucht wird auch die Existenz und Konstruktion von 1-dimensionalen Bus-Zeichnungen, bei denen Hyperknotenpositionen festgelegt sind.

Insgesamt untersuchen wir drei verschiedene Fragestellungen für (1) das PED-Modell, zwei verschiedene Fragestellungen für (2) das BUS-Modell und eine Fragestellung zum (3) Einbettungsproblem.

(1.1) Wir bestimmen Graphenklassen, die ein 1/4-SHPED besitzen, beweisen, dass es Graphen gibt, die kein 1/4-SHPED besitzen, präsentieren einen kräftebasierten Algorithmus, der solche Zeichnungen so gut wie möglich erstellt und evaluieren dieses Konzept.

(1.2) Wir charakterisieren Graphen bezüglich der Existenz von PED-Varianten und bestimmen die Komplexität für Konstruktion innerhalb solcher Varianten bei festgelegten Knotenpositionen: ncPED existiert für alle 2-planaren Graphen und kann für alle Graphen effizient konstruiert werden, sofern sie der Charakterisierung entsprechen. Auch effizient konstruierbar ist maxSPED für 2-planare Graphen, allerdings ist maxSPED im Allgemeinen NP-hart.

(1.3) Wir führen 1-bend SHOPEDs ein und präsentieren einen konstruktiven Algorithmus für 1-bend SHOPEDs für Graphen mit Maximalgrad 3, sowie ein Beispiel, welches kein 1-bend SHOPEDs besitzt.

(2.1) Wir charakterisieren Hypergraphen, die eine kreuzungsfreie 2-dimensionale Bus-Zeichnung besitzen und stellen einen effizienten konstruktiven Algorithmus bereit.

(2.2) Wir diskutieren Existenz und Konstruktion von 1-dimensionalen Bus-Zeichnungen. Wenn Hyperknotenpositionen festgelegt sind, zeigen wir für kreuzungsfreie Zeichnungen, dass nur sehr eingeschränkte Varianten eine polynomielle Komplexität haben, dass allgemeine Problem allerdings sehr schnell schwierig wird.

(3.1) Wir beweisen die Existenz einer subquadratischen universellen Punktmenge für das Einbettungsproblem von 2-außenplanaren Graphen.

Contents

Chapter 1

Introduction

Information visualization is the field of visualizing representations of data in order to support human perception. The focus is on development and empirical analysis of methods for presenting abstract information in a visual form aiming at amplifying cognition [136]. Its indispensability is emphasized through applications in many areas like scientific research, digital libraries, financial data analysis, system engineering, information engineering, software engineering, specification and verification of software, workflow, manufacturing production control, public safety through crime or fraud detection [8, 59, 118, 167, 173] to name only a few. For instance, Matt Biddulph, a software designer, extracted Wikipedia pages of people illustrating related persons by colors [20], see Figure 1.1. Such visualizations of social relations are helpful for finding topics of interest or for a company's customer marketing research [181]. In many application scenarios we use information visualization for representing relational data in the form of graphs, that is, an abstract data structure representing objects, called *vertices* and binary relations between objects, called *edges*. Visualizing data in the form of graphs is the content of *graph drawing*, which is the central topic of this thesis.

We will focus only on the structure of graphs with the intent to visualize graphs with aesthetically pleasing drawings [49]. In general, good drawings of a graph depend on their application but also on the structure of the graph itself. For example, when searching for a shortest path between two cities, then the usage of a map that visualizes all possible roads is a natural choice for this kind of search. Not convenient in this case is a drawing of a graph with edges representing roads, where edges are routed with arcs for the sake of aesthetics, instead of representing the real distances [159]. In contrast, the question which city is central can probably be better answered by studying an aesthetic drawing of the graph rather than a map, since this information is hidden in the structure of the graph.

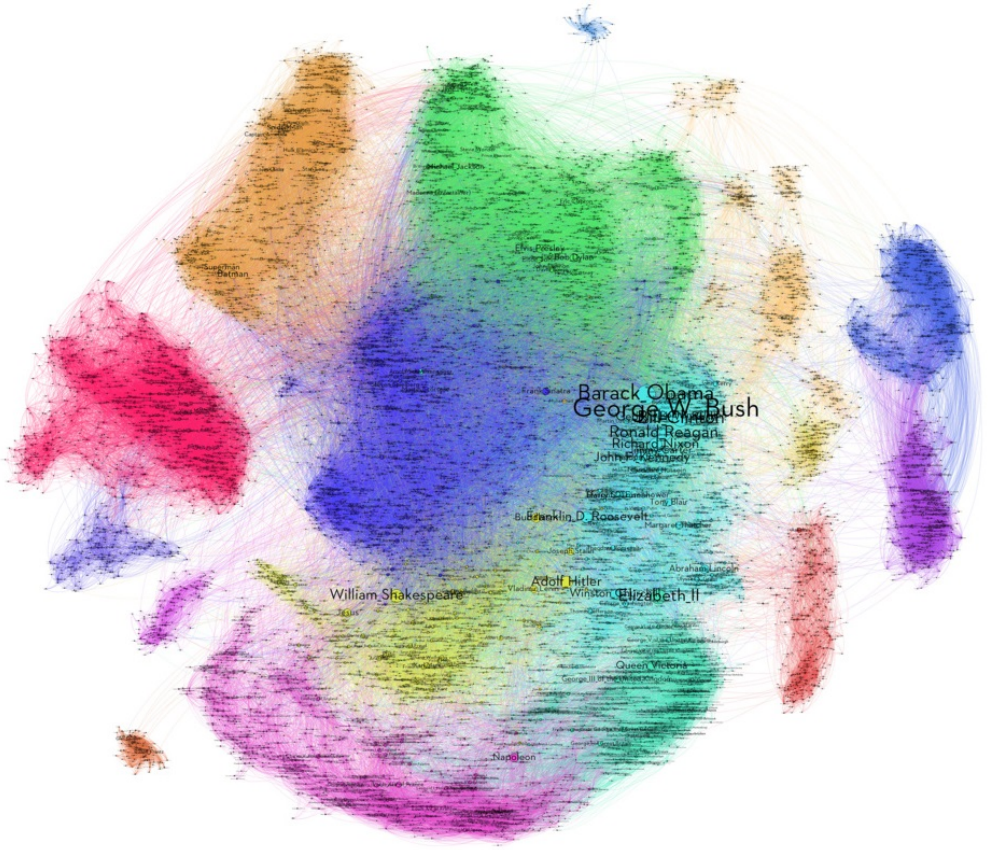


FIGURE 1.1: Visualization of Wikipedia pages of people colored with respect to social relations, taken from [20].

Apart from graphs we also consider data as a collection of sets, which can be seen as a hypergraph. Hypergraphs are abstractions of graphs, where objects, called *hypervertices*, arbitrarily connect many objects through a relation, called *hyperedge*. Classical examples of hypergraphs are color tables, where hyperedges represent colors and hypervertices are regions, which are influenced by the colors that overlap the region. Figure 1.2(a), for instance, illustrates a color table for visualization of the Young–Helmholtz theory of trichromatic color vision [192], which is a drawing of a hypergraph with three hyperedges and seven hypervertices, each representing a region. Other examples of hypergraphs are circuits, where hyperedges may represent connections between a power source and several resistances as in Figure 1.2(b).

Research in graph drawing recommends automatic graph drawing described by algorithms, first mentioned by Donald E. Knuth [121]. One possible reason for automatic generation of drawings is Figure 1.1, which won't be efficient when trying to draw this graph by hand. Additionally when the graph changes slightly, a hand-drawn graph cannot be changed easily, whereas running an algorithm with the modified graph is an easy and time-saving step. On the other hand algorithms may reject certain graphs as input. Let's assume for

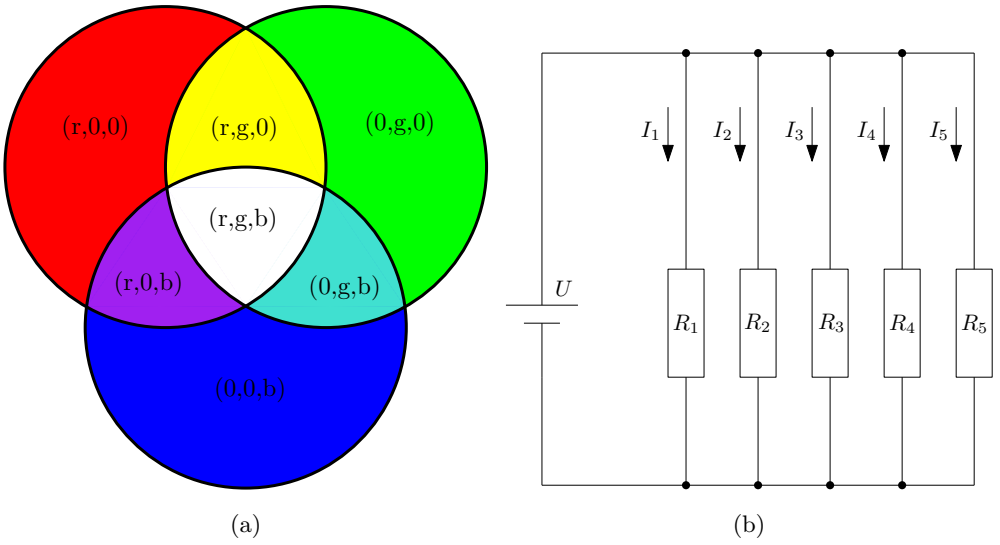
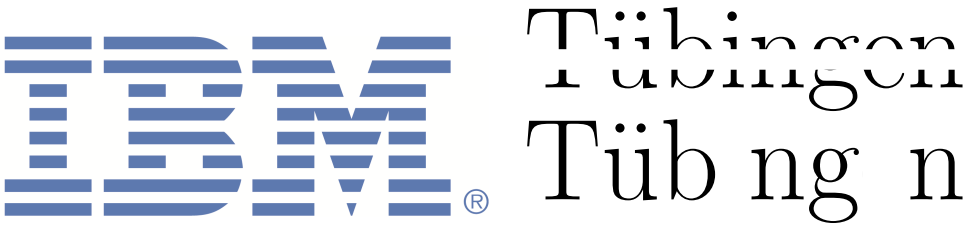


FIGURE 1.2: (a) Visualization of a color table according to the Young-Helmholtz-Theory, where triple (r,g,b) represents colors (red,green,blue) and 0 if no color of this type is used. (b) Visualization of a parallel circuit, where black points represent hyperedges connecting three wires from objects.

example a graph represents cities with train tracks connecting these cities. An algorithm drawing this graph should for example ensure that train tracks do not cross [138, 191]. Therefore it might be reasonable to first test the graph to determine whether its structure admits a crossing-free drawing, because otherwise the assumption was wrong. Thus for the algorithm that constructs a drawing, we are mainly interested in characterizing the graphs permitted as input, the complexity for testing if a graph is permitted as input, and the complexity of the construction.

Algorithms that produce drawings follow different rules for optimizing aesthetics in the drawing. There are algorithms producing drawings for metro maps [108], where edges mainly follow horizontal and vertical directions[1], which are used for so-called *orthogonal drawings*. There are also algorithms producing drawings for Unified Modeling Language diagrams, a modeling standard for specification, construction and documentation of software-parts [1], where edges are sequences of straight-line segments, so-called *poly-lines*. These rules for algorithms help to describe the algorithm and the ability to compare the drawings. Such a rule is called *drawing convention* and must be fulfilled when drawing a graph. According to [49], common conventions are planar (crossing-free) drawings, poly-line drawings, straight-line drawings, orthogonal drawings and recently invented Lombardi drawings [38, 64, 69, 133], latter respectively looking like pictures of the artist Mark Lombardi [171]. Drawing conventions optimize the aesthetics in drawings.

[1]Additionally to horizontal and vertical directions these algorithms for drawing metro maps produce diagonal directions as well.



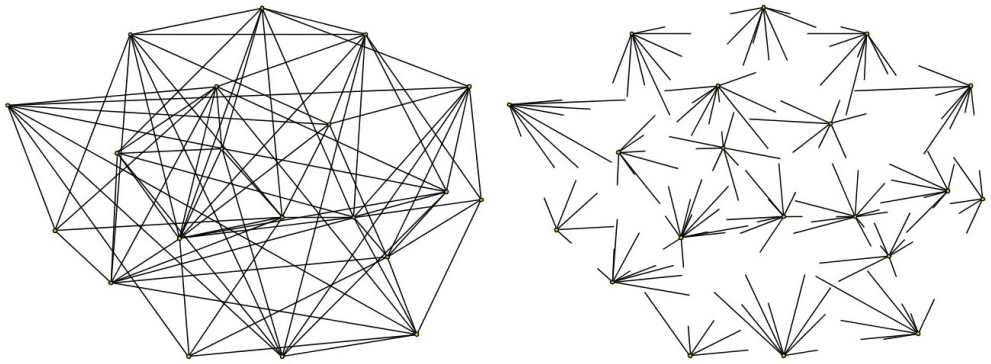
(a) (b)

FIGURE 1.3: (a) IBM uses gestalt principle in their logo [113]. The idea of (b) is taken from [164] and [97].

Algorithms follow drawing conventions in a specific drawing model, which is the geometric style according to which the drawing convention is performed. For instance, when drawing a graph using the straight-line convention, the traditional model is applied when vertices are drawn as discs with positive radius and edges are drawn entirely and thin, i.e., with positive width. This is the natural and intuitive way to draw straight-line edges. Other models for straight-line drawings aim at thickening of edges, called bold graph drawings [146, 185] or fat edge drawings [63]. Considering these models is useful for applications, when zooming into a drawing of a graph such that edges grow in length and width. Drawing models also exist for the orthogonal drawing convention, where the focus might be on modifying the vertices. If a vertex is drawn as a small disc, horizontal edge segments can only enter a vertex to the east or the west, and vertical edge segments can only enter a vertex from the north and the south. Thus only four edges can enter the vertex. In order to allow more than four edges, the shape of the vertex will be changed to a rectangle with appropriate side lengths, enabling many edges to enter the vertex on the same side in parallel. This model is called the Kandinsky model [67, 86]. Conclusively, the drawing model combines graph theory [24, 102] with geometry [155], which is known as a branch of the mathematical science dealing with sizes, shapes and positions of drawn objects. Our key goal is to identify structural constraints on graphs implied by the geometry of the drawing models.

This thesis contains three main parts. We introduce two new drawing models, the "PED" model for simple graphs in the first part and the "BUS" model for hypergraphs in the second part, while we deal with the traditional straight-line model in the third part. Both new models are defined later and aim for clarity by avoiding unnecessary information. This reduces potential confusion. Furthermore they achieve clarity by only drawing necessary parts, which avoids ambiguity [14, 158]. *Schematics* are visualizations following this concept. The PED model and BUS model support schematics that follow a common central idea:

"what is drawn is as much as necessary and as little as possible."



(a) (b)

FIGURE 1.4: (a) A traditional straight-line drawing of a graph G with 20 vertices and 80 edges. (b) A drawing of G with partially drawn edges, where the middle halves of the edges have been removed (1/4-SHPED). The drawings are generated with tools based on [87, 193].

For the PED model, the central idea is to use Gestalt theory [125]; see the first part of this thesis. One of the key ideas of this psychological theory is to exploit the mind, which tends to close holes in pictures. This aspect of Gestalt theory is the "principle of closure" [189]. Once the mind has captured the underlying rule according to which a picture is built, the brain automatically completes the picture, called "amodal completion" [48]. This Gestalt principle is used in many applications [35, 140, 189]. Even famous companies use the principle of closure within their logo [113], see Figure 1.3(a). The same idea is used when some unnecessary characters in words are missing or when they are not displayed completely [97, 164], see Figure 1.3(b).

We will continue the application of Gestalt theory on a graph drawing similar to [164], where the authors introduced gaps in edges. Since the authors provide no formal model or concept, we develop and introduce it by extracting aesthetic properties and presenting many results for several directions of application. In a high-level description, we define a *partial edge drawing* (or PED, for short) with respect to a straight-line drawing or orthogonal drawing as a drawing, where we remove a specific part of each edge in such a way that two crossing-free parts, called *stubs*, incident to the vertices remain [200, 201, 204, 207].

In the PED model we take advantage of Gestalt theory for edges in such a way that interrupted straight lines are completed by the mind in a natural way, i.e., without the change of direction during completing the interrupted straight-line. Using the straight-line drawing convention, the reader will reach the end vertex of the edge at some point. In this model we will introduce properties that support approximation of the end of the edge in the reader's mind, which are symmetry and homogeneity. Together the two properties imply that stub sizes are a specific fraction of the total edge length, referred to *stub-edge-ratio*. Our main focus is the stub-edge-ratio 1/4 as illustrated in Figure 1.4. Given

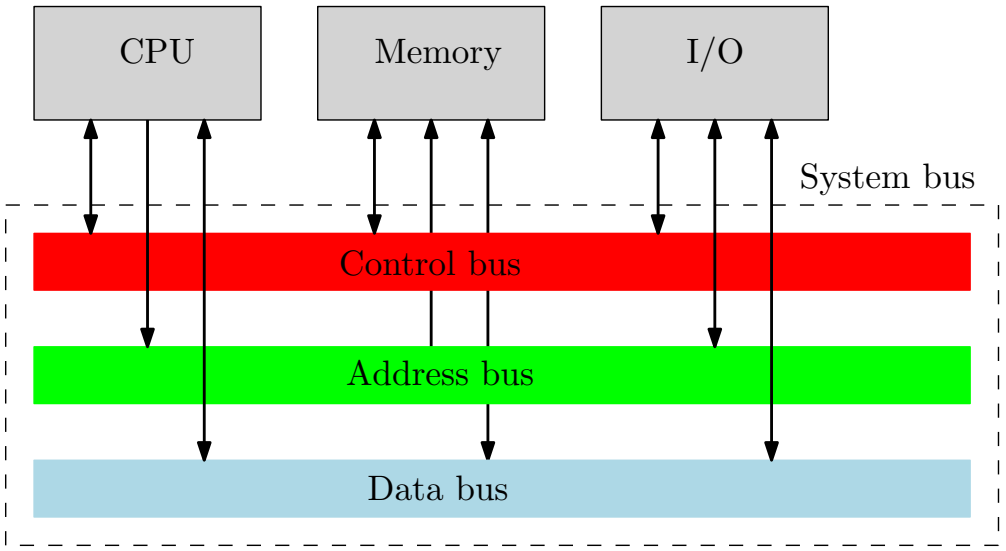


FIGURE 1.5: Buses are used in computer architecture to visualize the system bus component.

the length and the direction of a stub, this model helps to infer the position of the opposite stub. We refer to drawings in this model as 1/4-SHPEDs, where the letters S and H in this abbreviation emphasize the two properties.

In the BUS model, discussed in depth during the second part of this thesis, we follow the central idea by using buses. A BUS may be considered as backronym for Binary Unit System [84] and appears everywhere, for example as Universal Serial Bus stick, the USB-stick for short. A bus in computer architecture is a shared communication link for transferring data between several components by using one set of wires [149]. Buses visualize data transfer on common routes between input/output components, memory and Central Processing Unit (CPU) [161], as illustrated in Figure 1.5. Here buses may appear in different roles, i.e., as a "data bus" carrying the information, as an "address bus" carrying the information where to send data, or as a "control bus" carrying the information which operation to apply to the data, respectively, according to the "von Neumann architecture" [165, 187]. These three buses represent the "system bus". The advantage of buses is versatility and low cost, but in terms of data transfer they create a bottleneck [10]. This disadvantage, however, is irrelevant when buses are not specifically related to wires. In a more abstract setting, a bus is a visualization of a set containing several components. Transferring the situation to an underlying data structure, components are represented as hypervertices, the common route of components is represented as a hyperedge, and a *bus* is a drawing of this hyperedge from our point of view. Using buses is one possible approach to visualize hypergraphs.

Buses are also common in Very Large Scale Integration (VLSI) design [132, 183], which aims for the integration of millions of transistors on a single small chip. In order to compact the chip by rearranging components, it is helpful to consider

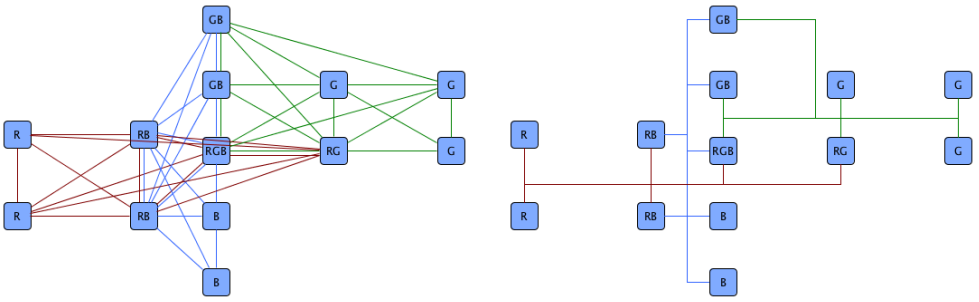


FIGURE 1.6: Three cliques represented in bus-style layout (taken from [193]).

a schematic drawing of the arrangement, represented by a hypergraph. In VLSI design, buses are used for bundling wire-routes that are used by several components or for visualization of components that are connected with many other components.

The bus-style layout can also be used for graphs, for instance, when representing cliques[2] in a compact and comprehensive way [56, 94]. Figure 1.6 illustrates the compactness, clearness and convenience of three cliques offered by the developer's guide[3] for the yFiles library [193].

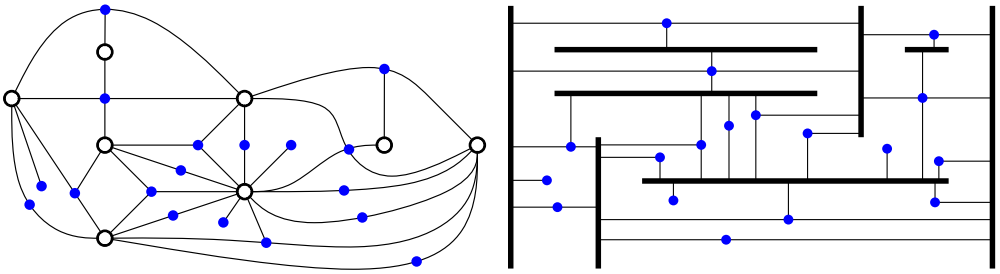
In our BUS model we will draw buses representing the hyperedges of a hypergraph. In order to apply methods from graph drawing, we transform the hypergraph into a graph such that a hypervertex v becomes a vertex, a hyperedge e becomes also a vertex and the edges represent incidences, i.e., v is connected with e, if the hyperedge e contains the hypervertex v. The resulting graph is called *bus graph* in order to emphasize the objective to represent hyperedges by buses. For an example, see Figure 1.7(a).

Ada et al. [3] introduced bus graphs in a very restricted way. In contrast, we define an abstract bus graph and investigate whether bus graphs as representatives of hypergraphs admit a drawing with buses. In a high-level-description we define *bus realizations* for hypergraphs in such a way that every hypervertex is drawn as a point and every hyperedge is drawn as bold segment (called *bus*) orthogonally connected to its hypervertices by a thin segment (called *connection*). For an example, see in Figure 1.7(b). Bus realizations may be specified by the number of slopes that appear for buses, referred to as *dimension*. If the dimension is one, by convention, we only use horizontal buses, while if the dimension is two we conventionally use horizontal and vertical buses. Higher dimensions are not considered in this thesis.

Bus realizations up to the second dimension resemble drawings in the orthogonal drawing convention since buses and their connections together use only

[2] A clique is a subgraph where every pair of vertices is adjacent.

[3] This diagram was created with the yFiles Java diagramming library, a product of yWorks GmbH, Tübingen (http://www.yworks.com).



(a) (b)

FIGURE 1.7: An example of a bus graph (a) with a realization (b).

horizontal and vertical segments. A difference is that not all vertices of the bus graph are drawn as points. All edges of the bus graph are drawn without any bend. Nevertheless, we will introduce basics of orthogonal drawings to be familiar with bus realizations and their related concepts.

Apart from the two new drawing models, we investigate, in the third part of this thesis, another topic of graph drawing in the traditional straight-line drawing model. This topic arises when we consider the space where we want to draw graphs. The graphs in the previous models were always drawn planar in the infinite Euclidean plane or on an infinite grid to support the orthogonal drawing convention. But what if we only permit a bounded region of the Euclidean plane for planar straight-line drawings, respectively just a finite grid? Another interesting question is, what if only some points of the Euclidean plane are permitted to place vertices, and how many points do we need? This is the key question for the third part of this thesis. This part builds a bridge to both the PED and the BUS model since in all parts we consider problem settings with point sets of predefined positions for points as input, on which vertices will be mapped. In contrast to the PED and BUS settings, the mapping of vertices to points is not part of the input for UPS.

It is well-known that any planar graph can be drawn straight-line and without crossings in the plane, as well as on an integer grid [45]. These authors bound the number of grid points asymptotically by $2n^2$ for a planar graph with n vertices, which was later improved to $(8/9)n^2$ [27]. On the other hand, outerplanar graphs, which are planar graphs admitting a planar drawing in which all vertices are on the outer boundary, can be drawn on any point set[4] of size n [25]. Therefore the question arises whether any planar graph can be drawn straight-line and planar on a point set of size n, which was negated by Chrobak and Karloff [39] and by Kurowski [128]. While n points are not always enough, it still makes sense to ask whether a point set of asymptotically linear size, or at least subquadratic size, exists that support straight-line planar drawings of every planar graph. This question, or more precisely the question on the

[4]These points must satisfy the assumption that no three points lie on the same line.

smallest asymptotic size of a point set with this property, is known as the *universal point set problem* (or UPS, for short) since the point set must be feasible for all planar graphs. As one step towards a full answer of UPS we give a positive answer for a class of graphs between planar graphs and outerplanar graphs, so called 2-outerplanar graphs. These graphs are a natural extension of outerplanar graphs in such a way that they admit a drawing in which we may remove all vertices from the outer boundary and an outerplanar graph remains. Using this incremental construction for k-outerplanar graphs we can say that any planar graph is k-outerplanar for some k, which is the reason why we consider the case $k = 2$. For the class of 1-outerplanar graphs we provide a universal point set of size $O(n \log n)$.

The main contribution of this thesis is the first formal concept for two new drawing models, the PED model and the BUS model (Ada et al. [3] introduce a very restricted version that is just two dimensional) including existential proofs, disproofs, and complexity calculations, respectively, as well as a big step towards a full answer of UPS. All results are new and provide a broad base for research in the direction of PED and BUS and UPS. The thesis is organized as follows:

In Chapter 2 we introduce basic definitions of graph drawing. This includes some important techniques for drawing graphs as well as necessary definitions for the remainder of this work.

Part I covers the PED layout model and starts with an introduction based on history and related work. This part has three main chapters. In the first main chapter, we investigate PEDs on the base of straight-line drawings. We demonstrate many classes of graphs admitting 1/4-SHPEDs and show that there exists an infinite class of graphs not admitting 1/4-SHPEDs. Furthermore we provide a force-directed algorithm for computing 1/4-SHPEDs if possible, or otherwise minimize the crossings on stubs. We empirically evaluate this 1/4-SHPED concept and intensively discuss the results. In the second main chapter, we adopt PED based on straight-line drawings on graphs with fixed vertex positions. We call PEDs with nearly completely drawn edges ncPED, and PEDs where all stubs have maximal length while fulfilling the requirement that both stubs of every edge have the same size, i.e., are symmetric, are called maxSPED. We prove the existence of ncPEDs for 1-planar, respectively 2-planar graphs, i.e., for graphs admitting a drawing with at most one, respectively two crossings per edge, and provide an efficient algorithm for constructing ncPEDs for these graphs. Furthermore, we prove NP-hardness for maxSPEDs of general graphs and present an efficient algorithm for 1-planar and 2-planar graphs. In the third main chapter, we investigate PED based on orthogonal drawings with one bend per edge. In this model for orthogonal drawings, symmetry and homogeneity are defined differently to the straight-line drawing in order to avoid ambiguity. We refer to drawings in this model as 1-bend SHOPEDs. We provide an algorithm producing a 1-bend SHOPED for all graphs with maximum degree three, while we prove that there is a graph with maximum

degree four that admits no 1-bend SHOPED. The results of this part have been published [200, 201, 204, 206, 207].

Part II covers the BUS model and also starts with an introduction based on history and related work. This part has two main chapters. In the first main chapter, we consider 2-dimensional bus realizations. We present a polynomial time algorithm that tests whether a planar bipartite graph representing a hypergraph admits a 2-dimensional planar bus realization and produces such a realization as byproduct in the affirmative case. Earlier it was just known that in general 2-dimensional bus realizations are NP-complete to test. In the second main chapter, we consider 1-dimensional bus realizations. First we consider the general setting and investigate connections to already well-known results. We consider bus graphs with fixed vertex positions and ask for a 1-dimensional planar bus realization. We provide relations to other NP-complete problems and present results via integer linear programming, which seem to confirm our impression that the problem is NP-complete. In one setting, where buses are strictly above or strictly below all their hypervertices, we prove NP-completeness for computing a 1-dimensional planar bus realization for a bus graph consisting of just a matching[5], but we additionally provide polynomial time algorithms for very restricted cases, e.g., when buses are only above all their hypervertices. All in all, this chapter provides a broad discussion on the limits for polynomial time solvable settings. Some results of this part have been published [202], some are under submission [203, 205].

Part III covers the UPS problem and also starts with an introduction based on history and related work. This part has only one main chapter and provides a solution for UPS with respect to 2-outerplanar graphs, which was an unsolved problem for many years [26]. We split the problem appropriately and refer for calculations on the size of the point set to a result of Bannister et al. [11]. The results of this part are under submission [198].

Finally we conclude with Part IV, summarizing the main results and pointing out further directions for research.

[5] A matching of a graph is a set of edges in which every vertex is incident to at most one matching edge.

Chapter 2

Basics of Graph Drawing

Graph drawing is a theory combining Computer Science with mathematics by using methods from information visualization, graph theory, topology, algebra and of course, geometry. The main objective is to find aesthetic drawings for graphs. In Section 2.1 we introduce the mathematical concept of a drawing in order to catch the geometric aspect of graph drawing. In Section 2.2 we then have a look at criteria, that judge the quality of a drawing with respect to its aesthetic. This section shows how to convert aesthetical aspects into criterias, which can be measured when drawing graphs according to certain drawing conventions. In Section 2.3 we have a short look at hypergraphs and show how drawing conventions and aesthetic criteria can be adopted there. After that we look at two standard techniques for getting drawings according to some specific criteria. One of them will be the force-directed technique in Section 2.4, while the other one will be the orthogonal graph drawing technique in Section 2.5.

2.1 Graphs, Drawings and Embeddings

We start this section with basic definitions of graph theory. For more details we refer the reader to the books of Di Battista et al. [49] or Diestel [60].

Graph: A *graph* $G = (V, E)$ is a pair of sets, where one set V, called *vertex set*, is the base set, and the other set E, called *edge set*, is a subset of the Cartesian product of the base set with itself, i.e., $E \subseteq V \times V$. This definition implies an order for the two elements of each edge, thus we call the graph *directed* and denote an edge by $(v, w) \in E$ for two vertices $v, w \in V$. We mostly deal with undirected graphs, that is, where the order of v, w is irrelevant, but we use the same notation for the edges, i.e., for an undirected graph, we have $(v, w) = (w, v)$. Whenever we consider directed edges, we will explicitly mention this.

An edge is called *multiedge* if it occurs more than once and $(v, w) \in E$ is called *loop* if $v = w$. We call a graph G *simple* if G has no multiedges and no loops. In general we only look at simple graphs. For an edge $e = (v, w)$ we call the vertices v, w *incident to* e and conversely for a vertex v we call any edge $e = (v, w)$ *incident to* v. For every edge $(v, w) \in E$ we call the two vertices v, w *adjacent* to each other. Directed graphs may have at most $|V|^2$ edges, directed simple graphs may have at most $|V|(|V| - 1)$ edges, while undirected simple graphs have at most $|V|(|V| - 1)/2$ edges. For a vertex $v \in V$ we define the *degree* $\deg(v)$ of v as $|\{e \in E \mid v \in e\}|$. Thus the equality $\sum_{v \in V} \deg(v) = 2|E|$ holds for any graph. Whenever we talk about a graph $G = (V, E)$, we immediately associate the numbers n with $|V|$ and m with $|E|$.

We call a graph $G' = (V', E')$ *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. The *induced subgraph* G' of G is then the subgraph such that $E' = E \cap (V' \times V')$. Furthermore a graph $G = (V, E)$ is called *bipartite* if we can partition its vertices into two disjoint sets V_1 and V_2 such that $V = V_1 \cup V_2, V_1 \cap V_2 = \emptyset$ and $E \subseteq V_1 \times V_2$. Similarly we call a graph $G = (V, E)$ *tripartite* if we can partition its vertices into three disjoint sets V_1, V_2 and V_3 such that $V = V_1 \cup V_2 \cup V_3, V_1 \cap V_2 = V_2 \cap V_3 = V_1 \cap V_3 = \emptyset$ and $E \subseteq V_1 \times V_2 \times V_3$.

Connectivity: Simple graphs may be categorized with respect to their connectivity, which is useful to partition graph algorithms into smaller well understandable parts. Let $G = (V, E)$ be a graph. A *path* between v and w of *length* $k + 1$ is a sequence of distinct vertices $v, u_1, \ldots, u_k, w, k \geq 0$ such that $(v, u_1), (u_1, u_2), \ldots, (u_k, w) \in E$. We call a pair v, w of vertices *connected* if there is a path between v and w. A simple undirected graph $G = (V, E)$ is called *connected* if every pair of vertices v, w is connected. We call a simple undirected graph *biconnected* if after removing any vertex (with its incident edges) G stays connected. Similarly we call a simple undirected graph G *triconnected* if after removing any vertex (with its incident edges) G stays biconnected. Every simple graph G can be decomposed into components according to the connectivity. A *connected component* is a maximal connected subgraph, a *biconnected component* is a maximal biconnected subgraph and a *triconnected component* is a maximal triconnected subgraph. The vertices separating a connected component are called *cut vertices*, while the pair of vertices separating a biconnected component is called *split pair*. These definitions are needed to introduce later two data structures called *block trees*, which are trees representing the biconnected components and *SPQR trees*, which are trees representing the triconnected components. In Section 9 we have a more detailed look how these data structures are built and how they are used.

In a graph G we call a path between v and w *cycle* if $v = w$. Notice that in non-simple graphs cycles may appear as self-loops or as multiple edges. In simple graphs a cycle contains at least three vertices. A graph that contains no cycles is called *forest* or *acyclic*. A forest that is connected is called *tree*. Thus trees are the smallest connected graphs in terms of the amount of edges. A tree $G = (V, E)$ can appear *rooted*, i.e., there is a designated vertex $r \in V$ which is

called *root*. The *depth* of a vertex v is the length of the path between v and r. Trees build an important sparse class of graphs. Another important dense class of graphs are the *complete graphs* which have edges between any pair of vertices, denoted by $K_n, n = |V|$. Similarly we denote by K_{n_1,n_2} the complete bipartite graphs and by K_{n_1,n_2,n_3} the complete tripartite graphs, where $n_i = |V_i|$ for $i = 1, 2, 3$.

Drawing: Next we give the concept of a drawing of a graph. We consider drawings only in 2 dimensions, i.e., we restrict the definitions to the Euclidean plane $\mathbb{R}^2$, together with the distance metric $d : \mathbb{R}^2 \to \mathbb{R}^2, (p, q) \mapsto d(p, q)$ for all points $p, q \in \mathbb{R}^2$. In $\mathbb{R}^2$ an *open Jordan curve* is the image of an injective continuous map $\varphi : [0, 1] \to \mathbb{R}^2$. Open Jordan curves φ connect two points p, q by a continuous curve, i.e., φ is a parametrization with $\varphi(0) = p, \varphi(1) = q$. An open Jordan curve is called *simple* if it is non-self intersecting. A *drawing* of a simple undirected graph $G = (V, E)$ is a map $\Gamma : (V, E) \to \mathbb{R}^2$ such that every vertex v is mapped to a distinct point $\Gamma(v)$ and every edge (v, w) is mapped to a simple open Jordan curve φ with $\varphi(0) = \Gamma(v)$ and $\varphi(1) = \Gamma(w)$. We will often use drawings of graphs to illustrate algorithms and properties of graphs. Drawings and graphs are different objects, although we often use the terms "edge" and "vertex" to refer to "the image of the edge" and "the image of the vertex".

Graphs can be categorized based on the drawings they admit. A drawing Γ is called *planar* if no two edges in Γ cross. We call a graph G planar if it admits a planar drawing. In this thesis planarity is of particular interest because planar graphs have some nice properties:

- Planar graphs admit better readable drawings (e.g., see [156, 158]).
- Planar graphs are well studied and have a long history (e.g., see [144]).
- Planar graphs are sparse (by Euler's formula [24], see below).

Note that planar graphs may also be drawn non-planar. Examples of planar drawings are given in Figure 2.1. Planarity of a graph can be decided independently of the drawings which the graph admits. A *subdivision* of graph $G = (V, E)$ is a graph $G' = (V', E')$ obtained from G by subsequently replacing edges $(v, w) \in E$ by a vertex u with incident edges $(u, v), (u, w)$.

Theorem 2.1 (Kuratowski's Theorem [24])**.** *A graph G is planar if and only if G does not contain a subdivision of K_5 or $K_{3,3}$.*

Planar drawings partition the plane into topologically connected regions, which we call *faces*. In a planar drawing there is one unbounded face, called *outer face*. The following theorem establishes a connection between the number of vertices, edges and faces of a planar graph.

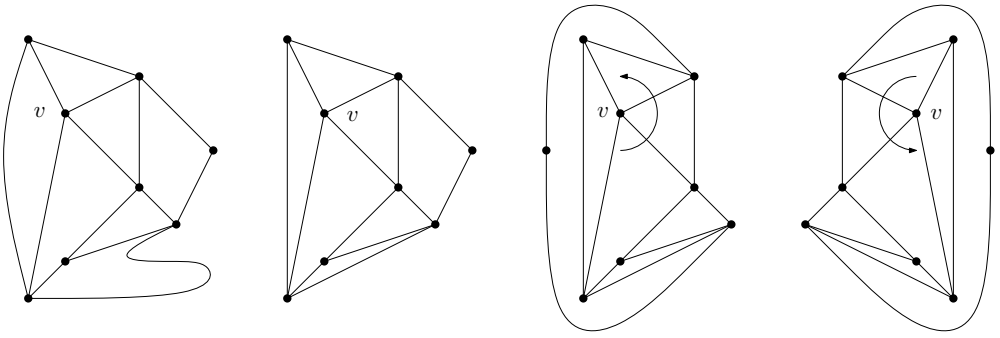


FIGURE 2.1: Four different planar drawings of a graph G. The left three of the drawings of G have the same planar embedding, while the rightmost drawing has a different planar embedding, which can be observed identifying the adjacent vertices of v along the circular arc. This figure is inspired by a graph from [49].

Theorem 2.2 (Euler's formula [24])**.** *Any planar graph $G = (V, E)$ with f faces satisfies $|V| + f = 2 + |E|$.*

Corollary 2.3. *Any planar graph $G = (V, E)$ has at most $|E| \leq 3|V| - 6$ edges.*

A special subclass of planar graphs is the class of *outerplanar graphs*, which are graphs admitting an *outerplanar drawing*, i.e., a planar drawing in which all vertices are incident to the unbounded face. The class of k-outerplanar graphs generalize the class of outerplanar graphs as follows. A *k-outerplanar graph*, with $k \geq 2$, is a graph admitting a *k-outerplanar drawing*, i.e., a planar drawing such that if removing the vertices of the outer face produces a $(k-1)$-outerplanar drawing, where 1-outerplanar stands for outerplanar.

Embedding: Sometimes it is not necessary to speak about planar drawings, but only about the topological structure of a drawing. A planar drawing Γ of a graph G determines a clockwise ordering of the edges incident to each vertex v of G, that we call *rotation at v*, denoted by $\phi_v \in V^k, k = \deg(v)$, regarding the adjacent vertices of v. The *rotation scheme* of G in Γ is the set of the rotations $\Phi = (\phi_{v_1}, \ldots, \phi_{v_n})$ at all the vertices $v_i \in V$ of G determined by Γ. An *embedding* of graph G is a rotation scheme Φ, together with the specification of the outer face.

We say a planar drawing Γ_G of graph $G = (V, E)$ is *equivalent* to a planar drawing Γ'_G of the same graph if they have the same embedding. In such a way an embedding of a graph G is an equivalence class of planar drawings with respect to the rotation scheme and the choice of the outer face. We call a graph *embedded* if its rotation scheme and the outer face are specified. A planar graph may have many embeddings, while a triconnected planar graph has only a single embedding up to the choice of the outer face. Examples of embeddings are given in Figure 2.1.

2.2 Aesthetic Criterias and Drawing Conventions

Drawings of graphs may look very different. Usually we describe a drawing by *points* and *curves*, which are the images of vertices and edges. The drawing conventions try to unify the drawings in terms of their geometric representation. The conventions are rules that must be satisfied in a drawing if the respective convention is chosen.

Conventions: Here we present a list of important drawing conventions, which is not complete, but sufficient to know as a base for this work taken from [49].

1. In a *planar* drawing the edges do not cross.
2. In a *straight-line* drawing every edge is drawn as a straight-line segment between its two incident vertices.
3. In a *orthogonal* drawing every edge is drawn as a chain of subsequent alternating horizontal and vertical straight-line segments. The attachment of two consecutive segments is called *bend*.
4. In a *visibility* drawing every vertex is drawn as a horizontal segment and every edge (v, w) is drawn as vertical segment with its endpoints at the segments representing v and w without crossing any other horizontal segment. The edges represent "lines of sight"[50, 53].

Graphs that admit visibility drawings[1] are called *visibility graphs* [53, 73, 179, 190]. The literature precisely distinguish between *weak visibility drawings*, where edges may be drawn if a line of sight between two horizontal segments is present, and *strong visibility drawings*, where edges must be drawn if a line of sight between two horizontal segments is present. Unless otherwise specified we refer to weak visibility drawings, whenever we talk about visibility drawings. Also we insist on crossing free visibility drawings, although there is already research on non-planar variants [46, 80].

The whole work deals with the planarity convention (1). Examples of planar drawings according to the above conventions are given in Figure 2.2. In Part I we first focus on a variant of straight-line drawings (2) and second on orthogonal drawings (3) with one bend per edge, both combined with the planar drawing convention (1). In Part II we combine a variant of orthogonal drawings (3) with a variant of visibility drawings (4) together with planarity (1). Finally in Chapter 12 we consider again straight-line drawings (2) together with the planar drawing convention (1).

[1] Some authors don't list visibility drawings as drawing convention, but as drawing "approach". For consistency we adopt visibility drawings to drawing conventions as [42].

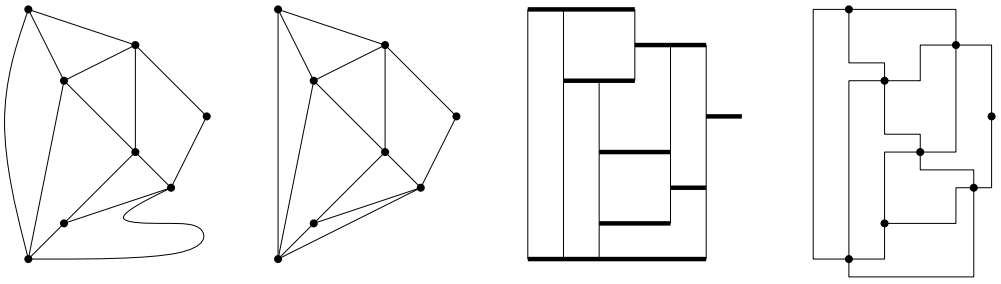


FIGURE 2.2: Four different drawings of a planar graph G. A planar drawing of G that is not a straight-line drawing, a planar straight-line drawing of G, a visibility drawing of G and an orthogonal drawing of G. This figure is inspired by a graph from [49].

The drawing conventions build a set of rules in order to fulfill aesthetic criteria, which are a set of properties that seem to support readability and interpretability of drawings.

Aesthetics: Drawings of a graph may look arbitrary complex. One of the main goal in the area of graph drawing is to draw the graphs as "nice" as possible. Unfortunately "nice" is subjective and depends also on the information, what we want to read from the graph. So researchers collected a list of criteria which improve the readability of graphs according to user studies, cf. [156]. These criteria have the advantage that they can be measured given a drawing of a graph. Some of the criteria are listed below, taken from [49] based on [14, 158, 174].

1. Minimization of the total number of crossings.
2. Minimization of the area of a drawing (the area is calculated as the area of the smallest enclosing rectangle, when the minimum edge length is one).
3. Minimization of the maximum number of bends per edge.
4. Minimization of the total number of bends.
5. Maximization of the angular resolution (which is the smallest angle between two incident edges of the same vertex).
6. Minimization of the total number of slopes of the edges.
7. Maximization of the symmetry.

Typically each drawing convention optimizes some aesthetic criteria. But some of the criteria may contradict each other. For example planar graphs can be drawn fulfilling the first criteria, while the resulting drawing is not always area minimal, see Figure 2.3. Thus we usually pick only one or a few of them and try to optimize them for a drawing.

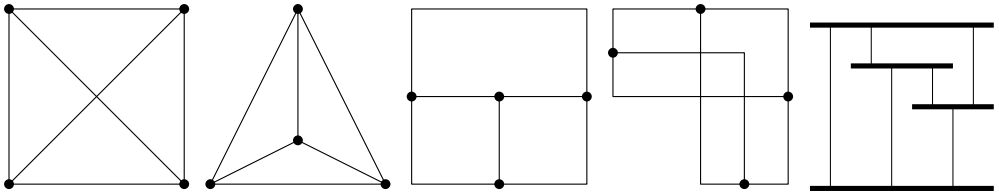


FIGURE 2.3: Five different drawings of the K_4. The visibility drawing (e) uses different "lines of sight" compared to Figure 2.2, which is permitted to avoid ambiguity of edges.

Figure 2.3 illustrates five drawings of the complete graph K_4. For these drawings the following **conventions** are chosen:

(1): (b), (c) and (e) obey the planar drawing convention.

(2): (a), (b) and (e) obey the straight-line drawing convention.

(3): (c), (d) and (e) obey the orthogonal drawing convention.

(4): only (e) obeys the visibility drawing convention.

We check now the **aesthetic** criteria that are fulfilled in the drawings of Figure 2.3:

(1): The drawings (b), (c) and (e) have no crossings.

(2): The drawing (a) is area minimal.

(3): The drawings (a), (b) and (e) have 0 bends, (c) has 2 bends and (d) has 1 bend as maximum number of bends per edge.

(4): The drawings (a), (b) and (e) have 0 bends in total, (c) and (d) have 4 bends in total.

(5): The drawing (a) has angular resolution 45 degree, (b) has 30 degree, (c) and (d) both have 90 degree, and (e) has even 180 degree.

(6): The drawing (a) has 4 slopes, (b) has 6 slopes, (c) and (d) have 2 slopes, (e) has 1 slope.

(7): The most symmetric drawing is (a) out of all drawings.

The aesthetic criteria build a set of properties that are aimed to be fulfilled by choosing an appropriate drawing convention. In the whole work we aim at satisfying the aesthetic criteria (1) together with (4). Using the orthogonal drawing convention we automatically optimized the aesthetic criteria (5) and (6). We also try to optimize (3) and (2). In the straight-line drawing convention, we mainly try to satisfy the aesthetic criteria (7), while (5) and (6) as well as (2) and (3) seem to contradict the symmetry in some cases.

2.3 Hypergraphs

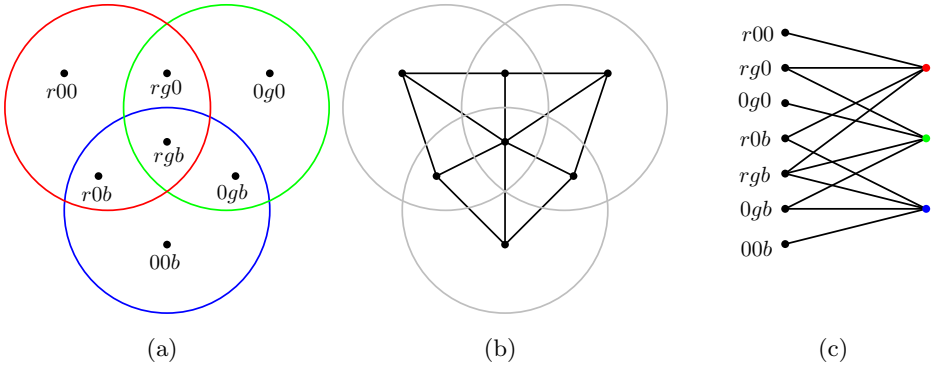


FIGURE 2.4: An Euler diagram (a), support (b) and incidence graph (c) of the hypergraph from Figure 1.2(a).

In this section we introduce in a generalization of graphs.

A *hypergraph* $H = (V, E)$ is a pair of sets, where one of the sets V, called *hypervertices*, is the base set, and the other set $E \subseteq \mathcal{P}(V) \setminus \emptyset$, called *hyperedges*, is a subset of the power set of V. The power set $\mathcal{P}(V)$ contains all possible subsets of V. A hypergraph can be seen as set system in a natural way.

There are several drawing conventions for hypergraphs, but we mention only a few of them in the following.

Euler diagram: In an *Euler diagram* the hypervertices V are drawn as points in the plane and the hyperedges E are simple closed Jordan curves, which enclose all points representing the hypervertices, which are contained in the hyperedge. This is a visualization of the hypergraph as set system. Figure 2.4(a) shows an example of an Euler diagram. Euler diagrams don't cover all possible subsets of the hypervertices in general in contrast to Venn diagrams [168]. The advantage of those diagrams is that for few hypervertices the diagrams look nice and readable and support understanding of logical formulas for instance. On the other hand they are not practical for many hypervertices, since the contours of hyperedges quickly become complicated [141]. In particular in case of Venn diagrams the contours of all hyperedges for n hypervertices partition the plane into 2^n regions although not all of them are really needed. Notice also that there are hypergraphs that cannot be visualized by Euler diagrams, or have no unique drawing [186]. Therefore several attempts to unify and structure Euler diagrams were investigated [81, 82, 162], as well as aesthetically pleasing Euler diagrams [83]. An Euler diagram is also a subdivision drawing [114, 117], whose dual graph is considered as "support".

Support: A *support* $G = (V_G, E_G)$ of a hypergraph $H = (V_H, E_H)$ is a graph on the same vertex set ($V_G = V_H$), where the induced subgraph $G_e = (e, E_G \cap e \times e)$ is connected for every $e \in E_H$. By drawing a support of a

hypergraph according to graph drawing conventions, we obtain a drawing of the hypergraph, see Figure 2.4(b). The advantage of supports is the usability of familiar graph drawing techniques. Also the hypergraph becomes clearer, since some information is omitted. On the other hand this omitted information leads to ambiguous drawings, i.e., from a support we may not necessarily distinguish hypergraphs. Also a support of a hypergraph is not unique. Often the visualization of support is underlined with a weak gray drawing of the respective hypergraph. The current research about supports of hypergraphs is restricted to planarity of the supports. In general the decision whether a hypergraph admits a planar support is NP-complete [114]. Even testing whether a hypergraph admits a 2-outerplanar support is NP-hard [32]. On the positive side the decision whether a hypergraph admits a path, cycle, tree or cactus[2] support is in P [29, 32, 120]. It remains open whether a hypergraph admits an outerplanar support. With some restrictions on the hypergraph H, that is if H is closed under intersection and difference, then the decision whether H admits an outerplanar support or planar support is in P [28].

We will tackle the problem to find an orthogonal-like support for hypergraphs by considering the incidence graph of a hypergraph in Part II.

Incidence graph: The *incidence graph* $I = (V_H \cup E_H, E)$ of a hypergraph $H = (V_H, E_H)$ is a bipartite graph with a vertex for every hypervertex and every hyperedge and edges between every hypervertex $v \in V_H$ and hyperedge $e \in E_H$, if $v \in e$. By drawing the incidence graph of a hypergraph according to graph drawing conventions, we obtain a drawing of the hypergraph, see Figure 2.4(c). In contrast to supports, the incidence graph is unique. In contrast to Euler diagrams, the incidence graph looks more familiar (since people are more familiar with graphs) and does not look as complicated as Euler diagrams when considering many hypervertices and many hyperedges. Incidence graphs may be drawn adopting any drawing convention and thus aesthetics of graph drawings may be transferred to aesthetics of hypergraph drawings.

Some research tackles the problem to visualize non-planar graphs, e.g., incidence graphs, in a planar way using techniques from confluent drawings [56, 70, 110], which provide very aesthetically pleasing drawings due to smooth bends in edges, in particular for complete and complete bipartite graphs [105]. Another similar approach is edge bundling [194], which provides pleasing visualizations connecting hierarchical entities [93, 106]. This technique can be combined with force-directed algorithms [107] improving aesthetics even more due to additional symmetry. Both techniques lead to high-quality drawings for many non-planar graphs and inspired us for the drawings in Part II.

When we talk about hypergraphs we focus mainly on the incidence graph in Part II. In such drawings it remains to visualize the difference of vertices representing hypervertices and vertices representing hyperedges. This can be done using colors or different shapes.

[2] A cactus is a specific outerplanar graph that is a tree consisting of edges and cycles.

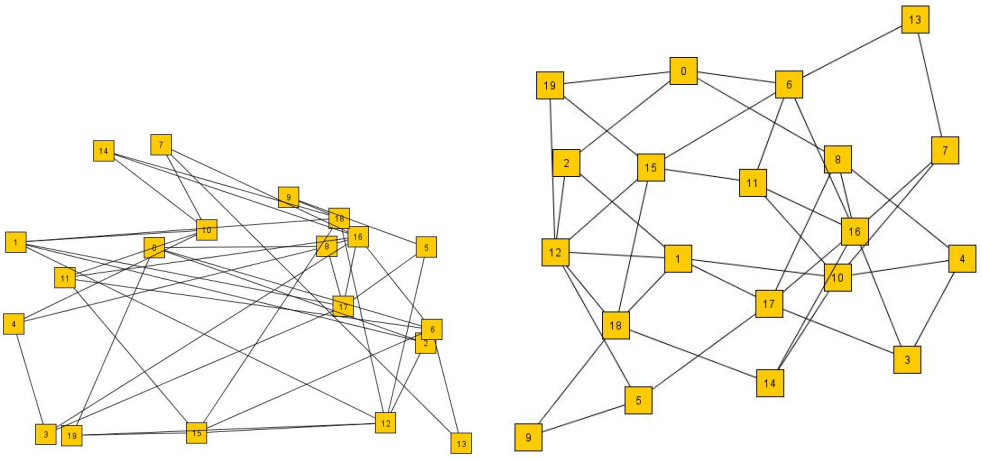


FIGURE 2.5: A randomly created drawing of a graph with 20 vertices and 40 edges (left) and the same graph drawn by the force-directed algorithm implemented in [193].

2.4 Force-Directed Algorithms

It may often happen that a graph as mathematical structure is given without any initial drawing. In this section we present a technique that produces drawings in the straight-line drawing convention, which are nice in terms of the aesthetic criterion "symmetry", see an example in Figure 2.5. The concept of force-directed algorithms is explained by physical analogies. For a more comprehensive survey we refer to [49, 123, 124].

The rough idea of force-directed algorithms is to treat the vertices as physical particles that are connected by a spring representing an edge. The springs attract the physical particles and try to minimize the forces acting on the particles using Hooke's law. If a configuration is found, where the particles are in an equilibrium state, i.e., the forces on every vertex sum up to zero or close to zero, then this configuration is taken to draw the graph with vertices at the positions defined by the positions of particles in the equilibrium state. Similarly to springs we can treat vertices as electrons that attract or repulse depending on whether there is an edge connecting them or not.

We will give a short overview of four important, but different methods for force-directed algorithms.

1963: Tutte [184] introduced a force-directed algorithm that used the barycentric method for computing forces. This algorithm guarantees a crossing free straight-line drawing with convex faces for every triconnected planar input graph. The algorithm fixes three vertices that form a convex face and places the remaining vertices by solving a linear equation system. The algorithm is summarized in Algorithm 1 according to [49].

Algorithm 1: Tutte's algorithm

Input : $G = (V, E)$ with partition $V = V_0 \cup V_1$, where V_0 is the set of *fixed* vertices and V_1 is the set of free vertices, $|V_0| \geq 3$, a strictly convex polygon P whose vertex set is in 1-to-1 correspondence to V_0.

Output: a position p_v for each $v \in V$ such that the fixed vertices form P.

Place each $u \in V_0$ at a corner of P, and each $v \in V_1$ at the origin;

repeat

  foreach *free vertex* v **do**

    $x_v = \frac{1}{deg(v)} \sum_{(u,v)\in E} x_v$;

    $y_v = \frac{1}{deg(v)} \sum_{(u,v)\in E} y_v$;

  end

until x_v *and* y_v *converge for all free vertices* v;

1984: Eades [65] introduced a force-directed algorithm that used springs for computing forces. This algorithm models the spring mechanism, i.e., vertices are represented by steel rings and edges by springs connecting the rings. When at least three rings are fixed in some position, then the spring forces move the remaining vertices to a minimum energy state, the equilibrium. There are only two practical aspects to adjust this system: (1) the strength of the springs is logarithmic, i.e., $c_1 \log(d/c_2)$, where d is the length of the spring, and c_1, c_2 form constants that can be adjusted according to experiments, (2) nonadjacent vertices repel each other by an inverse square root force $c_3/\sqrt{d}$, where c_3 is a constant again adjusted by experiments. Finally the algorithm uses two more constants c_4 and M, whose values were evaluated by experiments. The algorithm is summarized in Algorithm 2 according to [123], see Figure 2.6 for illustration.

Algorithm 2: Eades's algorithm

Input : Graph G

Output: Straight-line drawing of G

Initialize Positions: place vertices of G in random locations;

for $i = 0$ ***to*** M **do**

  calculate the force acting on each vertex;

  move the vertex c_4·(force on vertex);

end

draw a filled circle for each vertex;

draw a straight-line segment for each edge;

1989: Kamada and Kawai [115] introduced a force-directed algorithm using graph theoretic distance for computing forces, which is a completely different approach compared to the previous ones. They computed attractive and repulsing forces by the relation how far (in terms of Euclidean distance) are the vertices away from their graph theoretic distance (the length of a shortest path). This gives automatically attracting and repulsing forces.

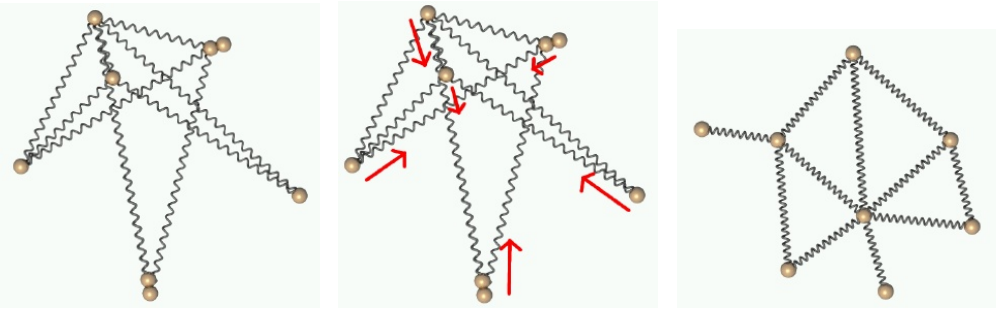


FIGURE 2.6: Illustration of a spring embedder, where edges are illustrated with springs aiming for a stable configuration taken from [123] based on [91].

Algorithm 3: Kamada–Kawai's algorithm

Input : Graph $G = (V, E)$
Output: Straight-line drawing of G
compute pairwise distances $d_{ij}, 1 \leq i \neq j \leq n$;
compute pairwise ideal lengths $l_{ij}, 1 \leq i \neq j \leq n$;
compute pairwise spring lengths $k_{ij}, 1 \leq i \neq j \leq n$;
initialize particle positions $p_1, \ldots, p_n$;
while $\max_i \Delta_i > \epsilon$ **do**
  let p_m be the particle satisfying $\Delta_m = \max_i \Delta_i$;
  while $\Delta_m > \epsilon$ **do**
    solve $\partial\partial E(\delta_x, \delta_y)^T(x_m, y_m) = -(\partial_x E, \partial_y E)^T(x_m, y_m)$;
    $(x_m, y_m) = (x_m, y_m) + (\delta_x, \delta_y)$;
  end
end

For the length d_{ij} of a shortest path between vertex v_i and v_j, the ideal Euclidean distance is set to $l_{ij} = L \cdot d_{ij}$, where L is the desired length of a single edge. The respective spring strength is then $k_{ij} = K/d_{ij}^2$ for a constant K. The energy with which the vertices are moved is described by $E = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 0.5 k_{ij}(|p_i - p_j| - l_{ij})^2$. At each local minimum p_m, the partial derivatives $\Delta_m = \sqrt{\left(\frac{\partial E}{\partial x_m}\right)^2 + \left(\frac{\partial E}{\partial y_m}\right)^2}$ are zero. So we choose in each step the point p_m with maximum Δ_m. By $\partial\partial E$ we denote the Hesse matrix and by ∂E we denote the gradient of E. The algorithm is summarized in Algorithm 3, following the presentation by Kobourov [123].

1991: Fruchterman and Reingold [87] introduced a force-directed algorithm that uses electrons for modeling forces. Based on the idea of Eades, they refined the forces such that they became more correlated to electrical forces. They adopted also the idea of attractive and repulsive forces, but treated them completely independent from the graph theoretic approach of Kamada and Kawai.

Algorithm 4: Fruchterman-Reingold's algorithm

Input : Graph $G = (V, E)$, $area = W \cdot L$, max. # I of iterations.
Output: Straight-line drawing of G

Place vertices at random into a rectangle of size $W \times L$;
$k := \sqrt{area/|V|}$;
function $f_r(x) = k^2/x$;
for $i = 0$ ***to*** I **do**
  for $v \in V$ **do**
    $v.disp := 0$;
    for $u \in V$ **do**
      if $u \neq v$ **then**
        $\Delta = v.pos - u.pos$;
        $v.disp = v.disp + sign(\Delta) \cdot f_r(|\Delta|)$;
      end
    end
  end
  function $f_a(x) = x^2/k$;
  for $e \in E$ **do**
    $\Delta = e.v.pos - e.u.pos$;
    $e.v.disp = e.v.disp - sign(\Delta) \cdot f_a(|\Delta|)$;
    $e.u.disp = e.u.disp + sign(\Delta) \cdot f_a(|\Delta|)$;
  end
  for $v \in V$ **do**
    $v.pos = v.pos + sign(v.disp) \cdot \min(v.disp, t)$;
    $v.pos.x = \min(W/2, \max(-W/2, v.pos.x))$;
    $v.pos.y = \min(L/2, \max(-L/2, v.pos.y))$;
  end
  $t = cool(t)$;
end

Here the attractive force $f_a(d)$ and repulsive force $f_r(d)$ are defined using a value $k = C\sqrt{area/|V|}$ for a constant C to stay within the area, i.e.,

$$f_a(d) = d^2/k \text{ and } f_r(d) = -k^2/d.$$

Additionally they introduced a variable simulating cooling down "temperature" for the effect that adjustments of vertices are greater in the beginning of the algorithm compared to the end of the algorithm. The algorithm is summarized in Algorithm 4, following the presentation by Kobourov [123].

Summarizing, we recalled four force-directed algorithms for producing an aesthetic and pleasing drawing of a graph. All algorithms together provide the main ideas when graphs are considered in a physical model. In Section 4.4 we will use the ideas of force-directed algorithms presented here, in particular of Algorithm 4, in order to adopt them partially for our drawing techniques.

2.5 Orthogonal Drawings

In this section we concentrate on orthogonal drawings of graphs. From the drawing conventions we know already that in orthogonal drawings the vertices are drawn as points and the edges are drawn as chains of subsequent alternating horizontal and vertical straight-line segments. This restricts algorithms that produce an orthogonal drawing also to a class of graphs in a natural way, namely all vertices must have a degree 4 at most.

We can easily imagine that any graph of maximum degree $\Delta = 4$ admits an orthogonal drawing, but these drawings don't necessarily look nice due to the number of bends (and crossings). While the aesthetic criteria (6) minimization of number of slopes and (5) maximization of angular resolution are naturally optimized in orthogonal drawings, we consider for these drawings also other aesthetic criteria in order to fulfill them. Some aesthetic criteria such as (2) minimizing the area or (4) minimizing the total number of bends drastically restrict the class of graphs admitting an orthogonal drawing with optimal aesthetic. In [148] the authors investigate the question how to optimize the area allowing a constant number of bends. Optimizing only the area [85, 127] or only the number of bends [172] is NP-hard. Aiming for aesthetic criterion (3), that is, minimization of the maximum number of bends, Felsner et al. [79] characterized the class of graphs admitting an orthogonal drawing with one bend per edge. In Chapter 6 we will use this characterization of non-planar graphs admitting an orthogonal drawing with one bend per edge.

Next we have a short look at planar orthogonal drawings with few bends [177]. This introduction to orthogonal drawings is based on visibility graphs. These graphs are closely related to the graphs that we consider in Part II. One of the main results for planar graphs is given by Tamassia and Tollis [178].

Theorem 2.4 (Tamassia and Tollis [178])**.** *Let $G = (V, E)$ be a planar graph with $\Delta = 4$. There is an orthogonal drawing of G on a grid with at most $2.4|V| + 2$ bends, which can be computed in $O(|V|)$ time.*

In particular, the construction of Tamassia and Tollis ensures at most $2|V| + 4$ bends if G is biconnected. The construction follows from a given visibility representation of the graph. The attachments of edge-segments to vertex-segments must be resolved by placing a point as vertex on the vertex-segment and translating vertex-segments to edge-segments, which imply bends. The position of the vertex is chosen such that the number of bends is controlled and some drawing invariants are maintained, see Figure 2.7. A final post-processing step for "relaxing" the faces leads to the above result. For details, see [118]. The needed visibility representation yields the following result by Rosenstiehl and Tarjan [163], and Tamassia and Tollis [178].

Theorem 2.5 (Rosenstiehl and Tarjan [163], Tamassia and Tollis [178])**.** *Let $G = (V, E)$ be a planar graph with $\Delta = 4$. There is a visibility representation Γ for G, which can be computed in $O(|V|)$ time.*

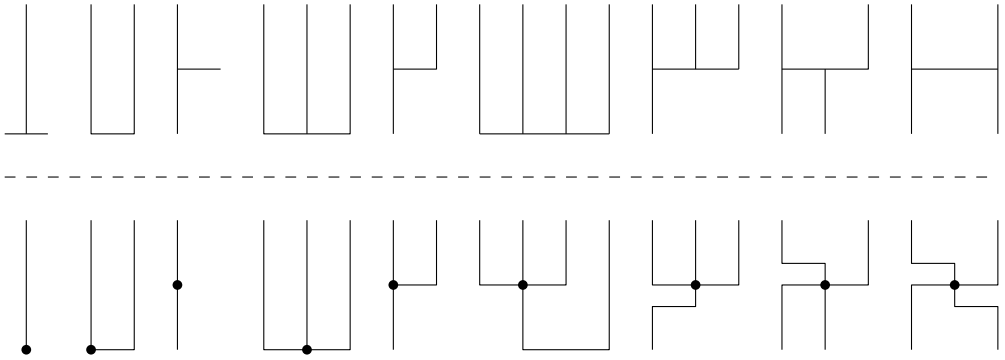


FIGURE 2.7: The above configurations in visibility representations are transformed to the orthogonal counterpart from below. Symmetric cases are omitted. The figure is taken from [118].

Furthermore this result implies some more properties on the representation Γ: If G is biconnected, then Γ can be constructed such that for any two vertices s, t incident to the same face, the two vertex segments $\Gamma(s), \Gamma(t)$ are the bottommost, the top most, respectively, and all remaining vertex segments $\Gamma(v), v \neq s, t$ have at least one incident edge segment from below and at least one incident edge segment from above. If G is connected, then Γ can be constructed such that a vertex segment $\Gamma(s)$ is bottommost, and all remaining vertex segments $\Gamma(v), v \neq s, t$ have at least one incident edge segment from below and at least one incident edge segment from above.

In a visibility representation Γ, the bottommost vertex segment $\Gamma(s)$ is called *source* and the topmost vertex segment $\Gamma(t)$ is called *sink*, if it exists. Note that in Part II the drawings we will consider are closely related to visibility graphs, which will become immediately clear in the respective section. Unfortunately we cannot use these techniques there, but this short introduction in visibility graphs aims to give some properties. It remains to find a visibility representation for a biconnected planar graph G, which can be directly obtained from an *st-order*, which is a ordering of the vertices $v_1, \ldots, v_n$ of a biconnected graph $G = (V, E), |V| = n$ such that every vertex $v_j, j \neq 1, n$ is adjacent to at least one $v_i, i < j$ and at least one $v_k, k > j$. The st-order determines the y-coordinate of the vertex-segments and the x-coordinates follow from a left to right pass over a planar embedding Γ of G. A visibility representation for a planar graph can be finally obtained by finding an st-order for each of its biconnected components and gluing them together. Details on computation of a st-order can be found in [74] by Even and Tarjan.

Theorem 2.6 (Even and Tarjan [74])**.** *Let $G = (V, E), |V| = n$ be a biconnected graph and let $s, t \in V$. There is an st-order with $v_1 = s, v_n = t$, which can be computed in $O(|E|)$ time.*

The procedure described above is illustrated in Figure 2.8. We will use an st-order also for orthogonal drawings in Chapter 6 in order to label the vertices

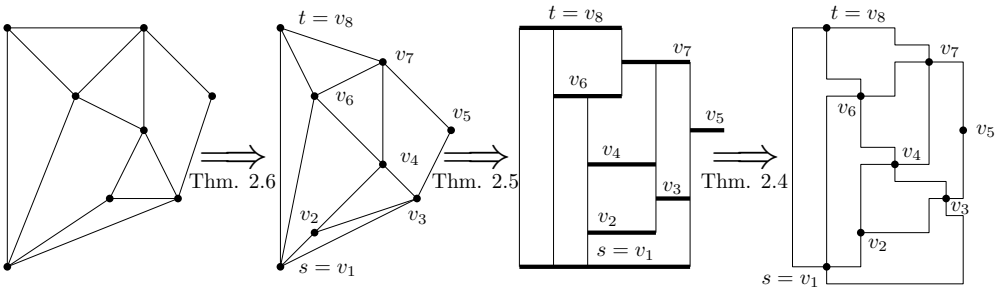


FIGURE 2.8: The graph from Figure 2.2 for which an st-order is computed by Theorem 2.6, a visibility representation by Theorem 2.5 and an orthogonal drawing by Theorem 2.4 before relaxing the faces. The orthogonal drawing after the relaxation is illustrated in Figure 2.2 [49].

with numbers according to their subscript, referred to as *st-numbering*. Whenever there is a non-biconnected graph, then biconnected components may be treated separately as we do in the respective section.

Part I

Partial Edge Drawings (PEDs)

Chapter 3

Introduction

One of the main principles for the effective visualization of graphs is the avoidance of edge crossings. Relating to this problem, very active research has been performed with works ranging from combinatorics, to algorithmics, visualization effects, to psychological user studies. Recently, the pragmatic approach has been proposed to avoid crossings by only drawing the edges partially. Unfortunately, no formal model and efficient algorithms have been formulated to this end.

The idea of *partial edge drawings* (PED) is to drop the middle part of the edges and rely on the remaining edge parts, called *stubs*. In Section 3.1 we point out related work that is part of the history of PED. Chapter 4 covers PEDs based on straight-line drawings, where we focus on some aesthetics (symmetry and homogeneity) within this model (1/4-SHPEDs). Using this model we show that some nontrivial graph classes admit 1/4-SHPED, while there are infinitely many graphs not admitting 1/4-SHPED. We present a force-directed layout algorithm that aims at producing a 1/4-SHPED for a given graph. We also evaluate this concept in the end of Chapter 4. In Chapter 5 we modify the concept to be applicable for graphs with fixed vertex positions, so-called *geometrically embedded graphs*. For those graphs our goal is to maximize the length of the stubs, therefore we partially disregard the symmetry and ignore the homogeneity completely, while insisting on non-crossing stubs. In Chapter 6 we adopt the PED model for orthogonal drawings. By doing so we restrict ourselves on orthogonal drawings where every edge has precisely one bend. We close this part with an overall conclusion about the PED model in Chapter 7. This part builds a comprehensive survey of PED based on published work [200, 201, 204, 206, 207].

3.1 History of PED

In the layout of graphs, diagrams, or maps, one of the central problems is to avoid the interference of elements such as crossing edges in graph drawings or overlapping labels on maps. This is a form of *visual clutter*. Avoiding visual clutter is one of the main objectives in cartography, information visualization, and graph drawing, which seriously affect the comprehensibility of a visualization [157, 188]. In graph drawing, a powerful method to achieve this is the avoidance and removal of edge crossings. Therefore the minimization of edge crossings became one of the most important research lines in the field [31, 49, 118]. Theoretical concepts such as planarity and crossing number form a sound mathematical basis.

Other papers study which non-planar graphs can be drawn such that the complexity of the edge crossings in the drawing is controlled. In the case of *k-planar drawings*, each edge is crossed at most k times (see, e.g., [55, 57, 109, 147]), while in *k-quasi planar drawings*, no k pairwise crossing edges exist (see, e.g., [2, 54, 175]), and finally in *large angle crossing drawings*, any two crossing edges form a large angle (refer to [58]).

In the last few years, more pragmatic approaches have been proposed for removing edge clutter when visualizing dense graphs. These approaches mostly stem from the field of information visualization. Edge bundling [43, 106, 107] (see [194] for a survey), confluent drawings [56, 110] and last, but not least partial edge drawings should be mentioned here. We will focus on *completely* removing edge crossings of non-planar graphs. Clearly, this is not possible in any of the traditional graph drawing styles that insist on connecting the geometric representations of two adjacent vertices (e.g., small disks) by a closed Jordan curve (e.g., segments of straight lines). In such drawings of non-planar graphs, some pairs of edge representations must cross (or overlap). This is a serious problem when displaying dense graphs.

All of the approaches above follow the same basic principle, but fail to provide a unified formal model as well as efficient algorithms to explore the power of this idea. As can be seen in Figure 3.1, the partially drawn edges improve the readability of drawings compared to completely drawn edges. In order to keep our drawing model as simple as possible, we do not allow any edge crossings at all and focus on optimizing the length of partially drawn edges. Our key questions are which graphs can be drawn with a prescribed length of partially drawn edges and which graphs cannot be drawn.

The beginning of PED

Becker et al. [15] have taken a rather radical approach to escape from this dilemma. They wanted to visualize network overload between the 110 switches of the AT&T long distance telephone network in the U.S. on October 17, 1989,

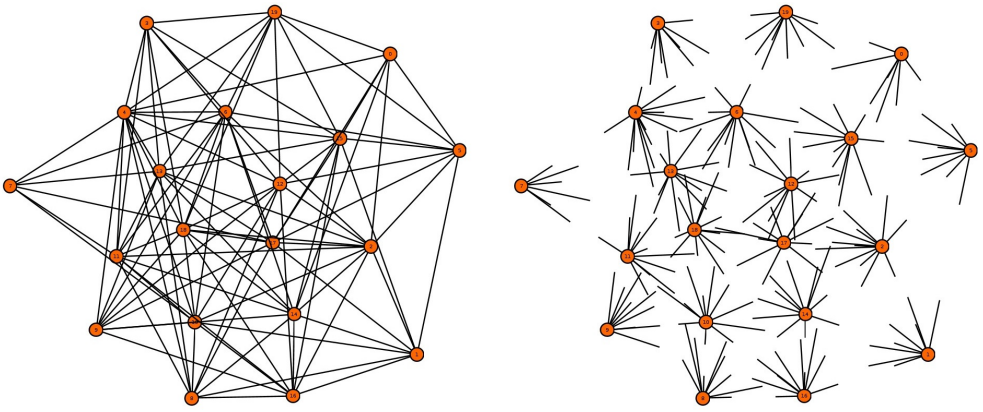


FIGURE 3.1: A randomly generated graph G with 20 vertices and 100 edges laid-out with a spring embedder [87, 193]. The drawing of G with partially drawn edges, where the middle halves of the edges have been removed (later called SHPED with stub-edge-ratio 1/4). Here the length of the stubs is proportional to the distance between corresponding vertices. The number of crossings shrinks from 413 to 23.

when the San Francisco Bay area was hit by an earthquake. They used straight-line segments to connect pairs of switches struck by overload; the width of the segments indicated the severeness of the overload. Due to the sheer number of edges of a certain width, the underlying map of the U.S. was barely visible. They solved this problem by drawing only a certain fraction (roughly 10%) of each edge; the part(s) incident to the switch(es) experiencing the overload. We call these parts the *stubs* of the edges. The resulting picture is much clearer; it shows a distinct east–west trend among the edges with overload.

Peng et al. [150] used splines to bundle edges, e.g., in the dense graph of all U.S. airline connections. In order to reduce clutter, they increased the transparency of edges towards the middle. They compared their method to other edge bundling techniques [93, 107], concluding that their method, by emphasizing the stubs, is better in revealing directional trends.

Burch et al. [33] recently investigated the usefulness of partial edge drawings of directed graphs. They used a single stub at the source vertex of each edge. They did a user study (with 42 subjects) which showed that, for one of the three tasks they investigated (identifying the vertex with highest out-degree), shorter stubs resulted in shorter completion times and lower error rates. For the two other tasks (deciding whether a highlighted pair of vertices is connected by a path of length one/two) the error rate went up with decreasing stub length; there was just a small dip in the completion time for a stub–edge length ratio of 75%.

A similar but less radical approach is the use of edge *casing*. Eppstein et al. [71] investigated how to optimize several criteria that encode the above–below behavior of edges in given graph drawings. They introduced three models (i.e.,

legal above–below patterns) and several objective functions such as minimizing the total number of above–below switches or the maximum number of switches per edge. For some combinations of models and objectives, they give efficient algorithms, for one they show NP-hardness; others are still open. Edge casings were re-invented by Rusu et al. [164] with reference to Gestalt principles.

Dickerson et al. [56] proposed confluent drawings to avoid edge crossings. In their approach, edges are drawn as locally monotone curves; edges may overlap but not cross.

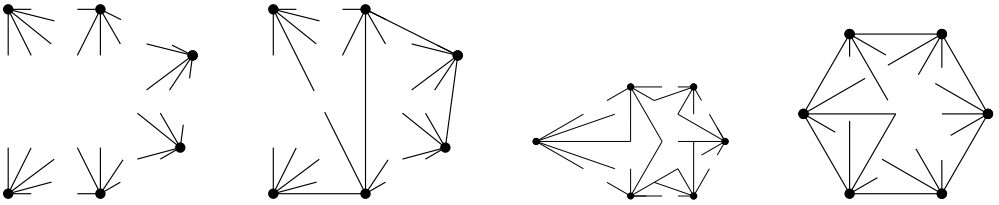
Chapter 4

PEDs for Graphs

In this section we introduce the PED model for straight-line drawings of graphs without fixed vertex positions. In Section 4.1 we define the PED model formally for drawings of graphs without given embedding and characterize PEDs with concepts like symmetry and homogeneity (1/4-SHPED). In Section 4.2 we identify nontrivial graph classes which admit a 1/4-SHPED and formulate a sufficient condition to guarantee a 1/4-SHPED. Most of the calculations can be generalized. In Section 4.3 we show that not all graphs admit a 1/4-SHPED. Specifically, we show that the complete graph K_{165} does not admit a 1/4-SHPED. In Section 4.4 we present a layout algorithm that aims at producing 1/4-SHPEDs for all graphs using a force-directed method. In Section 4.5 we evaluate this model and in Section 4.6 we summarize the results about graphs in the PED model for straight-line drawings without fixed vertex positions. Most of the results are published in [200, 201, 204, 206].

4.1 Formal Concept

In this section we formally define the model of PED for straight-line drawings. Let $G = (V, E)$ be an arbitrary undirected graph and let $\Gamma_G = (\Gamma(V), \Gamma(E))$ be a straight-line embedding of G, which maps the vertices to points in the Euclidean plane and the edges to segments connecting corresponding points. We will always identify the vertices of the given graph with the points in the plane to which we map the vertices. Note that we deal with undirected graphs in general. The directed version of edges is being considered when seen from a specific start/end vertex. Let $\gamma\colon [0,1] \times E \longrightarrow \Gamma(E)$ be a function with $\gamma([0,1], e) = \Gamma(e)$, continuous in the first parameter, describing points on an edge e. For $e = (v, w)$ we define $\gamma(\{0\}, e) := \Gamma(v)$ and $\gamma(\{1\}, e) := \Gamma(w)$. The formal definition of a PED_{Γ_G} is a drawing so that for every edge $e \in E$ there exists $A_e \subseteq [0,1]$ with $[0,1] \setminus A_e$ closed and connected and so that $\mathrm{PED}_{\Gamma_G} =$



(a) SHPED (b) SPED (c) HPED (d) PED

FIGURE 4.1: Drawings of K_6 in different types of PEDs.

$(\Gamma(V), \bigcup_{e\in E} \gamma(A_e, e))$ has no crossings of the drawn parts of an edge. So A_e describes the parts of the edge e that are actually drawn, consisting of two half-open intervals. For each $e \in E$ the length of $\Gamma(e)$ is denoted by the Euclidean distance $d(e) = d(v, w)$. We say $\gamma(A_e, e)$ is a *partial edge* and its length is denoted by $d(\gamma(A_e, e))$. There is one first segment e_f of $\gamma(A_e, e)$ incident to $\gamma(\{0\}, e)$ and one last segment e_l incident to $\gamma(\{1\}, e)$. Segments e_f and e_l are the *stubs* and can be assigned to an edge, which contains the stub, or to a vertex, which is incident to the stub. Its length is denoted by $d(e_f)$ or $d(e_l)$, respectively. Note, in every PED the set $[0, 1] \setminus A_e$ is connected. If $d(e_f) = d(e_l)$, the drawing is a *symmetric* PED (SPED). If $A_e = A_{e'}$ for all $e \neq e'$ with $e, e' \in E$, the PED is called *homogeneous* (HPED). That means the percentage of the drawn edge is equal for all edges. These PED-types are shown in Figure 4.1. Every SHPED has a *stub-edge-ratio* $\delta = d(e_f)/d(e)$, for $d(e_f) = d(\gamma(A_e, e))/2$ and an arbitrary edge $e \in E$.

It is easy to see that every planar graph G has a PED_{Γ_G} for an embedding Γ. For the following sections we assume G to be non-planar. We find the model for SHPEDs most appealing, when the stub-edge-ratio δ is being prescribed, denoted by δ-SHEPD. When considering an edge, the user can guess from the direction and the length of one of its stubs, where the other end of the edge is being placed. Therefore, we will mainly concentrate on δ-SHPEDs.

We introduced a formal model in a strict topological approach. In the next sections we consider the stubs not as images of parametrizations, but simply as set of points. Therefore we refer to stubs by an easier notation: in order to emphasize which of the two stubs of an edge $e = (v, w)$ is regarded, we denote the stub incident to v by s_v^e and the stub incident to w by s_w^e. We may even disregard the edge e in the notation, if the respective edge is unique from the context.

4.2 Graphs Admitting 1/4-SHPEDs

In this section we investigate graphs admitting 1/4-SHPEDs. Nevertheless we keep the stub-edge ratio δ as general as possible. We give bounds on the

number of vertices for specific graph classes, depending on δ. We start with complete graphs in Section 4.2.1 and continue with a sufficient condition that ensures 1/4-SHPEDs in Section 4.2.2. This condition provides a feeling for geometric behaviour and is further used in Section 4.2.3 to prove the existence of 1/4-SHEPDs for the class of powers of finite subgraphs of triangular tilings. Finally we prove the existence of 1/4-SHPEDs for complete bipartite graphs in Section 4.2.4 and for graphs of bounded bandwidth in Section 4.2.5. For all these classes the number of vertices of the graphs depends on the given stub-edge-ratio δ, but this bounds hold also for subgraphs of the mentioned graphs.

4.2.1 Complete Graphs

It is obvious that the complete graph K_n has a PED of any specific type, which can be constructed in a very simple way by drawing stubs sufficiently small. Thus a proof can be left out. As a next step, we consider the question, which complete graphs admit an SHPED when the stub-edge-ratio δ is given and compute a sufficient stub-edge-ratio for a given complete graph.

Lemma 4.1. *The graph K_n has an SHPED.*

Lemma 4.2. *For $\delta \in [0, 0.5]$, every subgraph G of K_n has an SHPED with stub-edge-ratio δ, with $n \leq n(\delta) = \sum_{i=1}^{m} n_i$ and $m = \lfloor \frac{1}{2\delta} \rfloor$, $n_i = \lfloor \pi / \arcsin(\frac{\delta}{1+\delta-2\delta i}) \rfloor$.*

Proof. First, we show that $n_1 = \lfloor \pi / \arcsin(\frac{\delta}{1-\delta}) \rfloor$ points can be placed on the circumference of a disk $\hat{c}$, so that K_{n_1} has an SHPED. Second we decompose $\hat{c}$ into layers and each layer contains $n_i = \lfloor \pi / \arcsin(\frac{\delta}{1+\delta-2\delta i}) \rfloor$ points, so that circles c_v, c_w for stubs do not intersect for each two vertices v, w in layer i.

Let $\delta \in [0, 0.5]$ be the stub-edge-ratio for an SHPED of $K_{n(\delta)}$. Let $\hat{c}(o, \hat{r})$ be the main disk, centered in the origin o with radius $\hat{r}$. The number n of uniformly distributed vertices on the circumference of $\hat{c}$ is maximal, if their distances is as follows, see Figure 4.2. The angle between each two neighbored vertices is $\alpha = 2\pi/n$. For each vertex $v \in V$ we find a small circle c_v touching v inside $\hat{c}$, which contains the stubs of v. The longest stub in every small circle has length $2r = 2\hat{r}\delta$, which is the diameter. We get a legal PED, if no two small circles intersect. So if two small circles c_v, c_w of v, w touch, the distance of the center points c_1 and c_2 is $l = 2\hat{r}(1-\delta)\sin(\alpha/2)$ and $l = 2r$ and thus we have $\frac{\delta}{1-\delta} = \sin(\pi/n) \Rightarrow n(\delta) := n = \lfloor \pi / \arcsin(\frac{\delta}{1-\delta}) \rfloor$.

Now we decompose $\hat{c}$ into *layers* depending on δ, see Figure 4.2. Let $\hat{c}(o, \hat{r}) - c(o, \hat{r} - 2\hat{r}\delta)$ be the first layer of the drawing and iteratively let $c(o, \hat{r} - 2\hat{r}\delta(i-1)) - c(o, \hat{r} - 2\hat{r}\delta i)$ be the i-th layer for $i = 1, ..., \lfloor \frac{1}{2\delta} \rfloor =: m$. So each layer is a disk of width $2\hat{r}\delta$ and if $\hat{r}$ is not a multiple of $2\hat{r}\delta$, a small disk $c(o, r')$ with $r' < 2\hat{r}\delta$ remains, which will not count as layer.

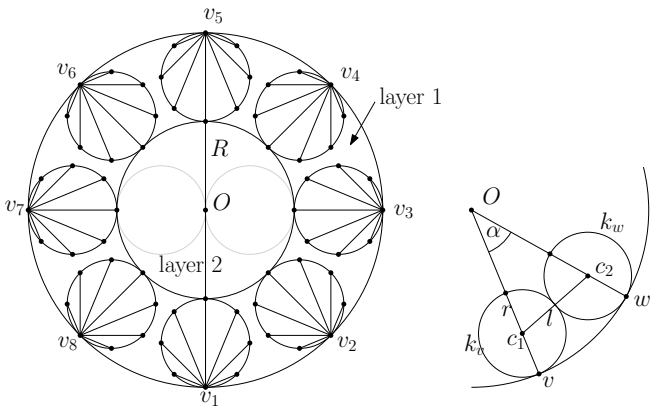


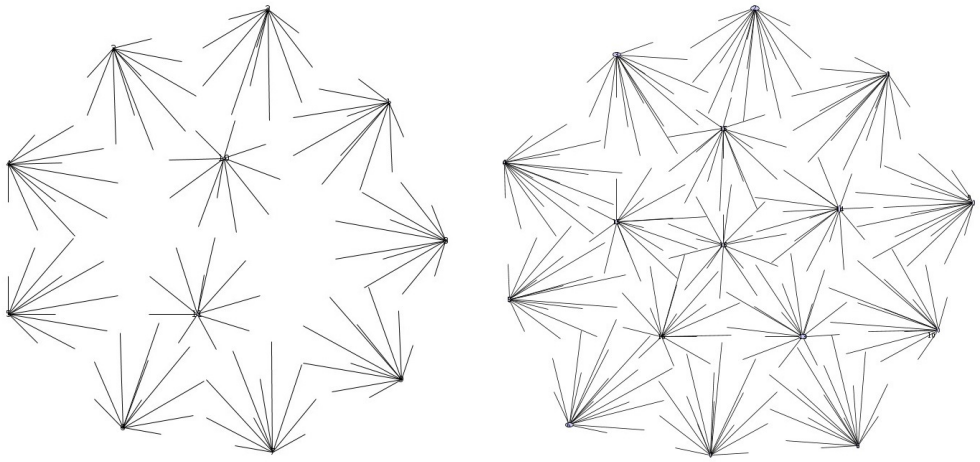
FIGURE 4.2: Computing the number of vertices for $\delta = 1/4$

A small circle c_v is described by δ and the position of v. We observe, that the radius r_v of the small circle c_v is the same for all $v \in V$, because our drawing is symmetric. So in general we have $r_v = \delta\hat{r}$ for all $v \in V$ and $l = 2\hat{r}\delta$ is the minimum distance of two neighbored vertices of the same layer. The center of each small circle of layer i is on the circumference of $c(o, \hat{r} - (\delta\hat{r} + 2\delta\hat{r}(i-1)))$, which is inside layer i. Thus we guarantee, that all small circles of layer i are completely inside layer i. We can compute the number n_i of vertices in layer i, with property $\frac{0.5l}{\hat{r}-(\delta\hat{r}+2\delta\hat{r}(i-1))} = \sin(\pi/n_i)$. This yields $n_i = \lfloor \pi / \arcsin(\frac{\delta}{1+\delta-2\delta i}) \rfloor$. □

If for example $\delta = 1/4$, then every subgraph of K_{11} has an SHPED with stub-edge-ratio δ, see Figure 4.3(a). If $\hat{r}$ is not multiple of $2\hat{r}\delta$, then a small circle c of radius r, which is not a layer, remains. If $\hat{r} < r < 2\hat{r}\delta$, then we can improve n by placing one more small circle inside c. There are two further ways to improve the result, namely to pack the small circles in a better way or to move the small circles such that they may intersect, but no stubs intersect. These improvements save a lot of space for more vertices, but it is more difficult to compute. By trial and error, we found a way to draw K_{16} for $\delta = 1/4$ in an SHPED, see Figure 4.3(b).

Lemma 4.3. *Let G be a subgraph of a complete graph K_n with n vertices. Then G has an SHPED with stub-edge-ratio $\delta \leq \frac{1}{\sqrt{4n/\pi}}$.*

Proof. We use the formula from Lemma 4.2. Assume $\delta = 1/x$ is the stub-edge-ratio of an SHPED of G for $x \in \mathbb{N}$. We distinguish between the odd and the even case of $1/\delta$ first and take the minimum of both values for δ's upper bound depending on n. With $\delta = 1/x$, we have $m = \lfloor x/2 \rfloor$ and the argument a of arcsin in the formula is $a = \frac{\delta}{1+\delta-2\delta i} = \frac{1}{x+1-2i}$. If x is even, we have $a_1 = \frac{1}{1}, a_2 = \frac{1}{3}, ..., a_m = \frac{1}{x-1}$ as arguments. Otherwise, we have $a_1 = \frac{1}{2}, a_2 = \frac{1}{4}, ..., a_m = \frac{1}{x-1}$ as arguments. So $n(\delta) = \sum_{i=1}^{\lfloor x/2 \rfloor} \lfloor \frac{\pi}{\arcsin(a_i)} \rfloor \leq$



(a) (b)

FIGURE 4.3: (a) SHPED of K_{11} with stub-edge-ratio $\delta = 1/4$ by construction. (b) SHPED of K_{16} with stub-edge-ratio $\delta = 1/4$ by trial and error.

$\sum_{i=1}^{x/2} \pi/a_i$. If x is even, then $n(\delta) \leq \sum_{i=1}^{x/2} \pi(2i-1) = \pi x^2/4$. Otherwise, we have $n(\delta) \leq \sum_{i=1}^{x/2} \pi 2i = \pi x + \pi x^2/2$. Thus we can compute $\delta_e(n) \leq \frac{1}{\sqrt{4n/\pi}}$, if x is even and $\delta_o(n) \leq \frac{1}{-1+\sqrt{1+2n/\pi}}$, if x is odd. The minimum $\min\{\delta_e, \delta_o\} = \delta_e$ defines a legal upper bound. □

4.2.2 A Sufficient Condition

Let $G = (V, E)$ be a graph, let $N(v)$ be the set of neighbors of $v \in V$, and let $\delta \in [0, 0.5]$ be the stub-edge-ratio of G's SHPED. For a vertex v, we choose an arbitrary radius $\hat{r}_v$ and set $r_v := \delta \hat{r}_v$. The following condition is shown in Figure 4.4, where the left drawing fulfills the condition and the right one does not. Note that this condition does not hold for the construction of K_n as above, because the vertices of K_n are not in the center of the small circles.

Condition 4.4. *There exists an embedding, such that for all vertices v all stubs of v are inside a small circle $c(v, r_v)$, all adjacent vertices of v are within a bigger circle $c(v, \hat{r}_v)$ and small circles $c(v, r_v), c(w, r_w)$ do not intersect for each v, w.*

Lemma 4.5. *If G satisfies Condition 4.4, then there exists an SPED. If additionally for all $v \in V$ and a constant $\delta \in [0, 0.5]$, the stubs of v have length $r_v = \delta \hat{r}_v$, then there exists an HPED.*

Proof. It is clear, that all stubs do not intersect, if the small circles do not intersect. For an SPED we draw the stubs with length $\min\{r_v, r_w\}$ for $e = (v, w)$. For an HPED we immediately have δ as stub-edge-ratio. □

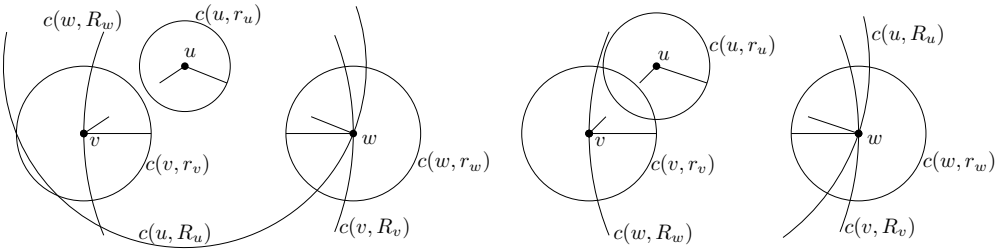


FIGURE 4.4: The embedding of K_3 to the left satisfies Condition 4.4, while the other one does not, although there are no "real" edge crossings.

4.2.3 Powers of Triangular Grids

We can identify an important graph class for which we can guarantee an SHPED by satisfying Condition 4.4. Let $T = (V_T, E_T)$ be a triangular tiling of the Euclidean plane with canonical embedding, see Figure 4.5. The dual T' of T is the so-called hexagonal tiling of the plane whose vertices lie in the centers of the bounded faces of T. For a connected graph $G \subset T$ with n vertices and $j \in \{1, \ldots, n-1\}$, we call G^j the j-th power of G if, for any path of length at most j from v to w in G, there is an edge (v, w) in G^j. Note that $G = G^1 \leq G^2 \leq \cdots \leq G^{n-1} = K_n$.

Theorem 4.6. *Let T be a triangular tiling and $G \subset T$ a connected subgraph with n vertices. For every $j \geq 1$ and any $\delta < \frac{1}{2j}$, G^j admits a δ-SHPED.*

Proof. Let T be a triangular tiling, let $G = (V, E)$ be a connected subgraph of T with n vertices, and let $j \geq 1$. Let T' be the dual of T as described. Every vertex $v \in V$ is inside a face $f(v)$ of T', called *comb*. More precisely v is the center of the hexagonal comb $f(v)$ in the embedding of T', see Figure 4.5. Choose $\delta < \frac{1}{2j}$ and call $\zeta := \frac{1}{2j} - \delta$ the *slack* for extensions of stubs in each comb. If $\zeta > 0.25$, then choose $\zeta < 0.25$ independent from j and δ. For movements of points we choose $\epsilon := \zeta / j$ as one unit of distance. For each $v \in V$ let $r = d(e) j \delta$ be the radius of a circle $c(v, r)$ containing all its stubs. Each comb $f(v)$ contains $c(v, r)$ completely and the distance of v to the boundary of $f(v)$, is at most half of the distance from v to a neighbor in G. Since δ is the percentage of the length between stubs and its edges, the maximum distance of v's neighbors in G^j is $\frac{d(e)}{2\delta}$ and thus all stubs in G^j can be drawn with $j = \frac{1}{2\delta}$. To avoid overlapping stubs, we move some of the vertices by multiples of ϵ.

Consider T as a grid with columns $1, \ldots, \hat{x}$ from left to right and rows $1, \ldots, \hat{y}$ from bottom left to the top right. Now we move the vertices of every i-th column downwards (*column-direction*) by $\epsilon(i \mod j)$ as well as all vertices of every i-th row downwards and rightwards along the row (*row-direction*) by $\epsilon(i \mod j)$ and get G'^j, which is shown in Figure 4.5. Now parallel edges in G^j do not overlap in G'^j. Next we prove that all stubs are inside their comb. Let $v, w \in V$ be two arbitrary vertices neighbored in G^j.

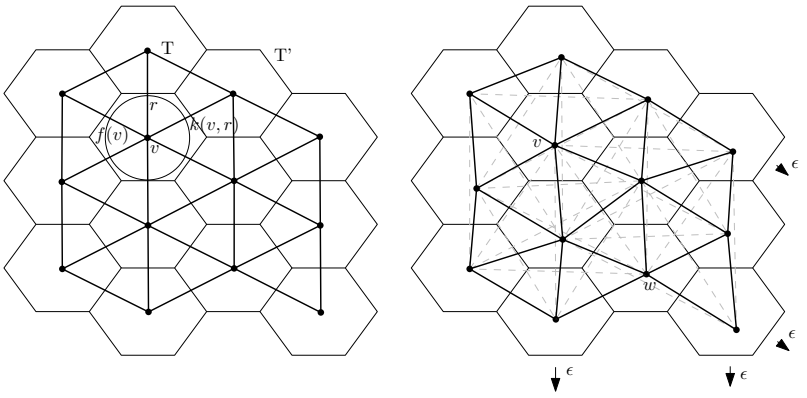


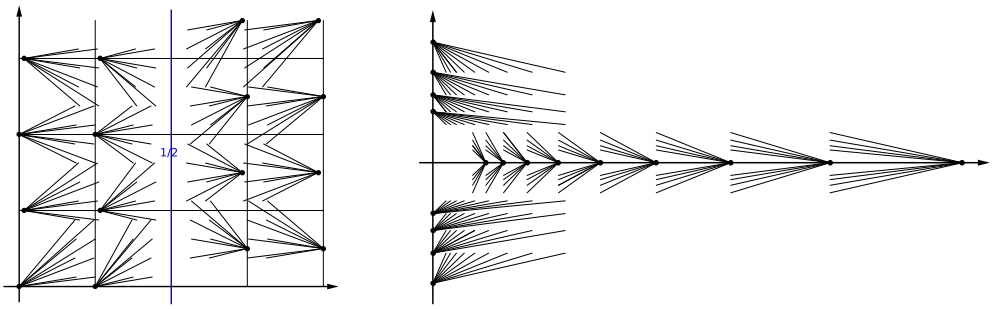
FIGURE 4.5: Triangular tiling T with its dual T' and a subgraph G of T and movement of vertices for $\delta < 0.25$ in G^2 (solid and dashed edges).

Case 1: Assume v, w are centers of combs in the same column of T. Then the positions of v, w have just a difference of at most $j'\epsilon$ in row-direction, $j' \leq j$. W.l.o.g. we can assume v is fixed and w is moved by $j'\epsilon$ to w', we just consider the differences of movements. By triangle inequality we can compute the distance $d(v, w') \leq d(v, w) + j\epsilon$. Thus with slack $\zeta = j\epsilon \geq j'\epsilon$ in each comb, there is no stub crossing its comb boundary, hence crossing no stubs.

Case 2: Assume v, w are centers of combs in the same row. Then we have just a movement in column-direction and we apply the same argument as in case 1.

Case 3: Assume v, w are centers of combs in different rows and columns of T. W.l.o.g. w is fixed and v is moved by $j_r\epsilon$ in row-direction ($0 \leq j_r \leq j$) and by $j_c\epsilon$ in column-direction ($0 \leq j_c \leq j$), which is v'. In G'^j we have the distance $d(v', w) \leq d(v, w) + j_r\epsilon + j_c\epsilon$ by triangle inequality. The distance $d(v, w)$ can be computed in G. Let $w, p_1, .., p_l, v$ be a shortest path from w to v in G. So $d(v, w) < 2r(l + 1)$, if we do not follow the path. Since v and w are not in the same row and column, there are 3 vertices p_{i-1}, p_i, p_{i+1} on the path such that p_{i-1} and p_{i+1} are not in the same row and column, $1 < i < l$. Then $d(p_{i-1}, p_{i+1}) = 2\sqrt{3}r$ and thus $d(v, w) \leq 2r(l - 1) + 2\sqrt{3}r$. In the end we have $d(v', w) \leq 2r((l - 1) + \sqrt{3}) + 2j\epsilon$ and since $\zeta = j\epsilon < 0.25$, we have $d(v', w) \leq 2r((l-1)+\sqrt{3})+0.5 < 2r(l+1)$ in G^j. Now G^j satisfies Condition 4.4 (apply Lemma 4.5) except of the separately considered vertices, but their stubs are inside their comb, as seen in the three cases. □

Corollary 4.7. *For $j = 1$, we have a planar graph $G = G^1$ and the movement of vertices is not necessary, while for $j = 2$, we have G^2 as the square of a triangular tiling with SHPED of stub-edge-ratio $\delta < 1/4$.*



(a) 1/4-SHPED of $K_{8,8}$ (b) 1/4-SHPED of $K_{8,9}$

FIGURE 4.6: Two methods for drawing complete bipartite graphs as SHPEDs.

4.2.4 Complete Bipartite Graphs

Our first construction is especially suitable if both sides of the bipartition have about the same size. The drawing is illustrated in Figure 4.6(a). Note that the figure scales x-axis and y-axis differently. In the following two results, there are fractions where both numerator and denominator are logarithmic expressions; therefore, we do not need to specify their bases.

Theorem 4.8. *The complete bipartite graph $K_{n,n}$ has a δ-SHPED if*

$$n \leq \left\lfloor \frac{1}{\delta} \right\rfloor \cdot \left\lfloor \left\lfloor \frac{\log 1/2}{\log(1-\delta)} \right\rfloor \right\rfloor,$$

where $\lfloor \lfloor r \rfloor \rfloor$ denotes the largest integer that is strictly less than r.

Proof. Let $k = \lfloor \frac{1}{\delta} \rfloor$ and $\ell = \left\lfloor \left\lfloor \frac{\log 1/2}{\log(1-\delta)} \right\rfloor \right\rfloor$. The latter implies that $(1-\delta)^\ell > \frac{1}{2}$.

Divide the plane at the vertical line $x = 1/2$ into two half planes, one for each side of the bipartition, to which we will refer as the right-hand side and the left-hand side. In each half plane draw the n vertices on a (perturbed) $k \times \ell$ grid. More precisely, for a horizontal line, let $\epsilon \geq 0$ such that $(1-\delta)^\ell > \frac{1}{2} + \epsilon$. Draw the vertices with x-coordinates

$$(1-\delta)^i - \epsilon \text{ and } 1 - (1-\delta)^i + \epsilon, i = 0, \ldots, \ell - 1.$$

Draw the vertices on the left-hand side with y-coordinates $0, \ldots, k-1$ and the vertices on the right-hand side with y-coordinates $0 + \sigma, \ldots, k - 1 + \sigma$ where $0 < \sigma < 1$ is chosen such that no two vertices on the right-hand side are collinear with a vertex on the left-hand side and vice versa. All edges are between a vertex on the left-hand side and a vertex on the right-hand side.

Then for any two vertices the bounding boxes of their incident stubs are disjoint up to their boundaries. Intersections of the stubs on the boundaries can be avoided by a suitable choice of ϵ.

1. If v is a vertex on the right-hand side with x-coordinate $(1-\delta)^i - \epsilon$, then the projection to the x-axis of the longest edge incident to v has length $(1-\delta)^i - \epsilon$. Hence all stubs incident to v are in the vertical strip bounded by $x = (1-\delta)^i - \epsilon$ and $x = (1-\delta)^i - \epsilon - \delta((1-\delta)^i - \epsilon) \geq (1-\delta)^{i+1} - \epsilon > 1/2$. The latter inequation follows since $i + 1 \leq \ell$.
2. Let v_i be a vertex with y-coordinate $\sigma + i$, $i = 0, \ldots, k-1$. Then the projection to the y-axis of the longest edge incident to v_i and above v has length $k - 1 - i - \sigma$ while the projection to the y-axis of the longest edge incident to v_i and below v has length $i + \sigma$. Hence the projection to the y-axis of the stubs incident to v_i and v_{i+1} do not intersect if $\delta(k - 1 - i - \sigma) + \delta(i + 1 + \sigma) < 1$ which is fulfilled if $k < 1/\delta$. If $k = 1/\delta$ then draw the vertices on the horizontal lines $y = i$ and $y = i + \sigma$ for even i with $\epsilon = 0$ and the vertices on the other horizontal lines with a slightly positive ϵ such that the end points of the stubs do not intersect.

A symmetric argument holds for the vertices on the left-hand side. □

Granacher [97] improved Theorem 4.8 by reducing the gaps between vertices and stubs in a more careful way.

Theorem 4.9 ([97])**.** *The complete bipartite graph $K_{2n,2n}$ has a δ-SHPED if*

$$n \leq \left\lfloor \left\lfloor \frac{\log(\frac{1-\delta}{\delta})}{\log(\frac{1-\delta-\delta^2}{(1-\delta)^2})} + 1 \right\rfloor \right\rfloor .$$

Our second construction is especially suitable if one side of the bipartition is much larger than the other. The drawing is illustrated in Figure 4.6(b).

Theorem 4.10. *For any integers $n > 0$ and $k < \log \delta / \log(1-\delta)$, the complete bipartite graph $K_{2k,n}$ has a δ-SHPED.*

Proof. Draw the n vertices on the x-axis with x-coordinate $x_i = 1/(1-\delta)^{i-1}, i = 1, \ldots, n$ and the $2k$ vertices on the y-axis with y-coordinate $y_i = 1/(1-\delta)^{i-1}, i = 1, \ldots, k$ and $-y_i, i = 1, \ldots, k$. All edges are between a vertex on the y-axis and a vertex on the x-axis. To show that no stubs intersect, we establish the following two properties on the regions that contain the stubs.

1. The stubs incident to $(0, \pm y_i), i = 2, \ldots, k$ are in the horizontal strip bounded by $y = \pm y_i$ and $y = \pm y_{i-1}$:

   The projection to the y-axis of any stub incident to $(0, y_i)$ has length $\delta \cdot y_i$, hence it stops at $y = (1-\delta) \cdot \left(\frac{1}{1-\delta}\right)^{i-1} = \left(\frac{1}{1-\delta}\right)^{i-2} = y_{i-1}$.

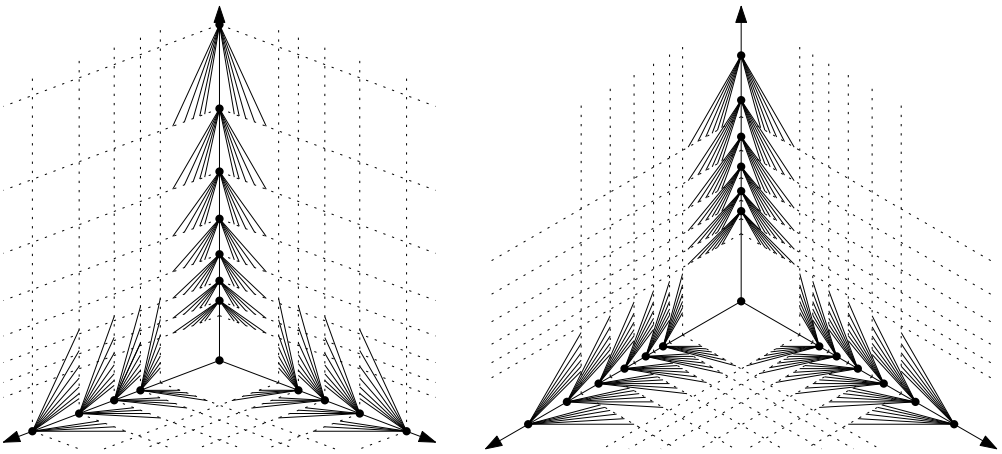


FIGURE 4.7: 1/4-SHPED of $K_{4,4,7}$ (left) and $K_{6,6,6}$ (right) with $\alpha = 2\pi/3$ from [97].

2. The stubs incident to $(0, x_i), i = 2, \ldots, n$ are in the rectangle bounded by $y = \pm(1-\delta)$, $x = x_{i-1}$, and $x = x_i$ (where $x_0 = 1-\delta$):

   As above, the projection of any stub incident to $(0, x_i)$ stops at $x = x_{i-1}$. The absolute value of the projection to the y-axis is bounded by $\delta \cdot y_k = \delta \cdot \left(\frac{1}{1-\delta}\right)^{k-1}$ which is less than $1-\delta$ if $k < \log\delta/\log(1-\delta)$.

Since the stubs incident to $(0, \pm y_1)$ lie in the horizontal strip bounded by $y = \pm 1$ and $y = \pm(1-\delta)$, it follows that any two stubs do not cross. □

Using a construction on only two axes (referred to as *rays*) Granacher [97] proved also results for tripartite graphs in a similar way. The results are summarized in the following two theorems. Again, $\lfloor\lfloor r \rfloor\rfloor$ denotes the largest integer that is strictly less than r. Figure 4.7 illustrates the construction according to both theorems.

Theorem 4.11 ([97])**.** *For any integers $n > 0$ the complete tripartite graph $K_{k,k,n}$ has a δ-SHPED if*

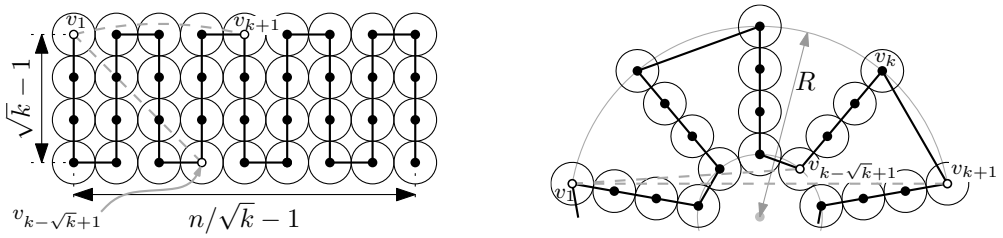
$$k \leq \lfloor\lfloor \log\delta/\log(1-\delta) \rfloor\rfloor,$$

and the angle α between every pair of rays fulfills $0 < \alpha < \pi$.

Theorem 4.12 ([97])**.** *For any integers $n > 0$ the complete tripartite graph $K_{n,n,n}$ has a δ-SHPED if*

$$n \leq \left\lfloor \left\lfloor \frac{\log(\frac{1-\delta}{\delta})}{\log(\frac{1-\delta-\delta^2}{(1-\delta)^2})} + 1 \right\rfloor \right\rfloor,$$

and the angle α between every pair of rays fulfills $0 < \alpha < \pi$.



(a) path case (b) circle case

FIGURE 4.8: SHPEDs for bandwidth-k and k-circulant graphs.

4.2.5 Graphs of Bounded Bandwidth

Tha *k-circulant* graph C_n^k with n vertices and $0 \leq k < n$ is the undirected simple graph whose vertex set is $\{v_0, \ldots, v_{n-1}\}$ and whose edge set is $\{(v_i, v_j) \colon |j - i| \leq k\}$. When we specify the index of a vertex, we implicitly assume computation modulo n. Note that $C_n^1 = C_n$ and $C_n^{n/2} = K_n$.

Granacher [97] showed how to use the Gosper curve to construct a $\frac{1}{4.7\sqrt{k}}$-SHPED for a bandwidth-k graph, i.e., the vertices of G can be ordered $v_1, \ldots, v_n$ and for each edge (v_i, v_j) it holds that $|j - i| \leq k$, even if k is not known.

For the case that k is known, we give drawings with a better constant. We provide δ-SHPED constructions for k-circulant and bandwidth-k graphs where $\delta = \Theta\big(1/\sqrt{k}\big)$, especially $\delta \geq \frac{1}{2.83\sqrt{k}}$ for bandwidth-k graphs. For ease of presentation, we assume that $\sqrt{k}$ and $n/\sqrt{k}$ are integers.

First, let G be a graph of bandwidth-k. We draw G as a δ-SHPED as follows. We map the vertices of G to the vertices of an integer grid of $\big(n/\sqrt{k} \times \sqrt{k}\big)$ points such that the sequence of vertices $v_1, \ldots, v_n$ traverses the grid column by column in a snake-like fashion, see Figure 4.8(a).

The distance from any vertex to its k-th successor is at most $\sqrt{\big(\sqrt{k}-1\big)^2 + k} < \sqrt{2k}$, see the two dashed line segments in Figure 4.8(a). Setting $\delta = 1/\big(2\sqrt{2k}\big)$ ensures that each stub is contained in the radius-$1/2$ disks centered at the vertex to which it is incident. Since the disks are pairwise disjoint, the stubs are disjoint.

For the k-circulant graph C_n^k, we modify this approach such that the start and the end of the snake coincide. In other words, we deform our rectangular section of the integer grid into an annulus; see Figure 4.8(b). We additionally assume that $n/\sqrt{k}$ is even.

The inner circle circumscribes a regular $\big(n/\sqrt{k}\big)$-gon Π of edge length 1.

We place the vertices of C_n^k on rays that go from the center of the annulus through the vertices of Π. On each ray, we place $\sqrt{k}$ vertices at distance 1 from

one another, starting from the inner circle and ending at the outer circle. The sequence again traverses the stacks of vertices in a snake-like fashion.

A vertex v can be reached from its j-th ($j < k$) successor s, by traversing at most $3\sqrt{k} - 2$ segments of length 1: at most $\sqrt{k} - 1$ segments from s to the inner circle, at most $\sqrt{k}$ segments on the inner circle, and at most $\sqrt{k} - 1$ segments from the inner circle to v. Hence, the maximum distance of two adjacent vertices is less than $3\sqrt{k}$, and we can choose $\delta = 1/(6\sqrt{k})$. The results presented above are summarized in the following theorem.

Theorem 4.13. *Let $2 \leq k \leq n$ and assume that $\sqrt{k}$ and $n/\sqrt{k}$ are integers. Then any graph of bandwidth k has a $1/\big(2\sqrt{2k}\big)$-SHPED. If additionally $n/\sqrt{k}$ is even, the k-circulant graph C_n^k has a $1/\big(6\sqrt{k}\big)$-SHPED.*

4.3 Graphs Not Admitting $1/4$-SHPEDs

In this section, we show that not any graph can be drawn as a 1/4-SHPED. Note that 1/4 is an interesting value since it balances the drawn and the erased parts of each edge. Yet, our proof techniques generalize to δ-SHPEDs for arbitrary but fixed $0 < \delta < 1/2$. We start with a simple proof for the scenario where we insist that vertices are mapped to specific point sets, namely point sets in convex or one-sided convex position. We say that a convex point set is *one-sided* if its convex hull contains an edge of a rectangle enclosing the point set.

Since we discuss about edges and stubs in this section extensively, we reduce notational complexity by the following notational conventions. We use uv as shorthand for the edge connecting u and v. If we refer to the stub uv then we mean the piece of the edge uv incident to u; the stub vu is incident to v.

The remainder of this section is structured accordingly for the decomposition of the proof into the subproblems, where we consider parts of the point set for a closer look.

4.3.1 The Main Argument

Theorem 4.14. *There is no set of 17 points in one-sided convex position on which the graph K_{17} can be embedded as 1/4-SHPED.*

Proof. We assume, to the contrary of the above statement, that there is a set P of 17 points in one-sided convex position that admits an embedding of K_{17} as a 1/4-SHPED. Consider the edge $e = uv$ that witnesses the one-sidedness of P. We can choose our coordinate system such that $u = (0, 0)$, $v = (1, 0)$ and all

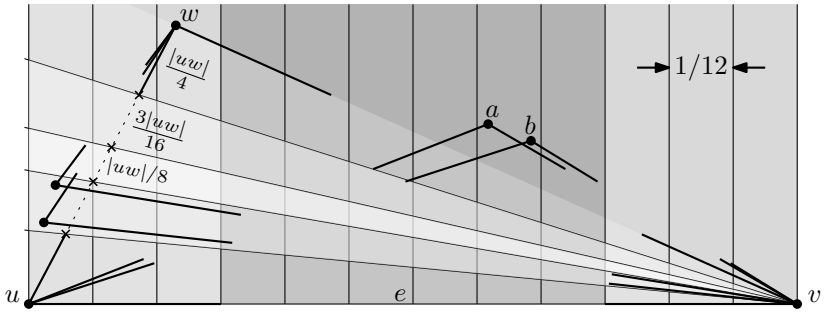


FIGURE 4.9: Sketch of the argument why no 17 points in one-sided convex position can be used to embed K_{17} as a 1/4-SHPED.

other points lie above e. We split the area above e into twelve interior-disjoint vertical strips of equal width, see Figure 4.9.

We first show that the union of the six innermost strips contains at most six points of P. Otherwise there would be a strip S that contains two points a and b of P. Let a be the one closer to u. Since S is one of the six innermost strips, the stub av intersects the right boundary of S (below the stub bv), and the stub bu intersects the left boundary of S (below the stub au). Point a lies above stub bu and point b lies above stub av. Hence, stubs av and bu intersect.

So at least eleven points of P must lie in the union of the three leftmost and the three rightmost strips. We may assume that the union S_{left} of the three leftmost strips contains at least six points. Let w be the rightmost point in $P \cap S_{\text{left}}$. We subdivide the edge uw into five pieces whose lengths are 1/4, 3/16, 1/8, 3/16, and 1/4 of the length of uw. Each piece contains its endpoint that is closer to one of the endpoints of uw. The innermost piece contains both of its endpoints. Now consider the cones with apex v spanned by the five pieces of uw. We claim that no cone contains more than one point.

Our main tool is the following. Let t be a point in $(P \cap S_{\text{left}}) \setminus \{u, w\}$. Then the stub tv intersects the right boundary of S_{left} and, hence, also the edge uw that separates $P \cap S_{\text{left}}$ from $P \setminus S_{\text{left}}$. It remains to note that in each cone, any point has a stub to u or w (whichever is further away from the cone) that intersects the boundary of the cone. □

Theorem 4.14 can be used to derive a first upper bound on general point sets as follows.

Corollary 4.15. *For any $n > \binom{30}{15}$, the graph K_n does not admit a 1/4-SHPED.*

Proof. By a result of Erdős and Szekeres [72], any set of more than $\binom{2k-4}{k-2}$ points in general position contains a subset of k points that form a one-sided convex set. Combining this with Theorem 4.14 and plugging in $k = 17$ yields the claimed bound. □

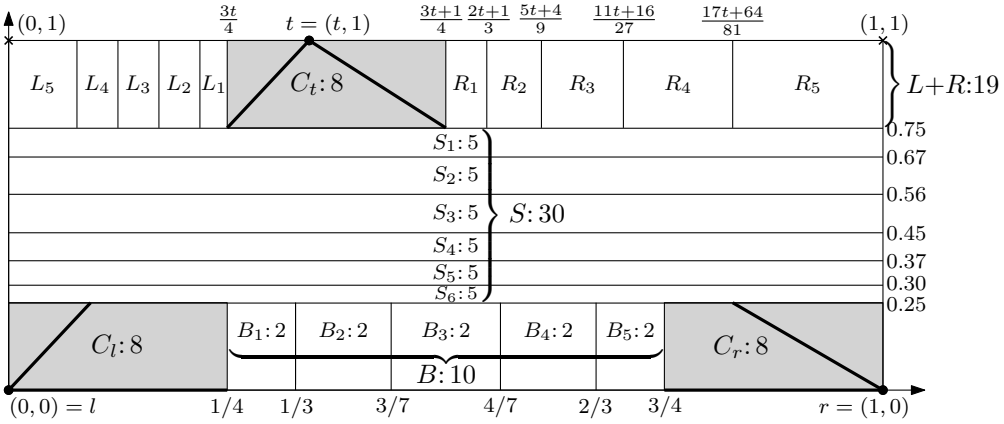


FIGURE 4.10: Partition of the enclosing rectangle A_t into cells. We have labeled each cell or group of cells with the maximum number of points that it can contain.

We now vastly improve upon the bound of Corollary 4.15. Let P be the point set in the plane, and let l and r be the two points on the convex hull that define the diameter of P, which is the largest distance between any two points. We rotate P such that the line lr is horizontal and l is on the left-hand side. Now let A be the smallest enclosing axis-aligned rectangle that contains P, and let t and b be the top- and bottommost points in A, respectively. Accordingly, let A_t be the part of A above (and including) lr and let $A_b = A \setminus A_t$. We consider the two rectangles separately and assume that the interior of A_t is not empty. (In our proof we argue, for any interior point, using only its stubs towards the three boundary points l, r, and t.)

We subdivide A_t into 26 cells such that for each point in a cell the three stubs to l, r, and t intersect the boundary of that cell; see Figure 4.10. For each cell, we prove, in the remainder of this section, an upper bound on the maximum number of points it can contain. Summing up these numbers (see again Figure 4.10), we get a bound of 83 points in total. Since we may have a symmetric subproblem below lr, we double this number, subtract 2 because of double-counting l and r, and finally get the following theorem.

Theorem 4.16. *For any $n > 164$, the graph K_n does not admit a 1/4-SHPED.*

We now prove Theorem 4.16 by upperbounding, for each cell in Figure 4.10, the number of points it contains.

For ease of presentation, we stretch A_t in y-direction to make it a square. Clearly, this operation does not change the crossing properties. We assume that the side length of A_t is 1. We further assume that the coordinates of l, r, and t, are $(0, 0)$, $(0, 1)$, and $(t, 1)$, respectively. Note that, by the choice of r and l, there are no other points on the left and right boundary of A_t (otherwise lr would not be the diameter of P). By symmetry, we may further

assume that $0 < t \leq 1/2$. For a point p, we call stub pt the *upper stub* of p, pr its *right stub*, pl its *left stub*, and both pr and pl its *lower stubs*.

For $p \in \{l, r, t\}$, let $C_p \subset A_t$ be the axis-parallel rectangle spanned by p and the endpoints of the two stubs that go from p to the two other boundary points. Note that C_l, C_r, and C_t (all shaded in Figure 4.10) are squares of size $1/4 \times 1/4$.

The squares C_l, C_r, and C_t subdivide the whole square into several parts, which will be considered in the next sections.

4.3.2 The Middle Strip

We first consider the middle strip $S = [0, 1] \times [1/4, 3/4]$. Here, we follow an improvement of Granacher [97]. In order to upperbound the number of points that S contains, we subdivide S into six horizontal strips, $S_1, \ldots, S_6$, from top to bottom. For $i = 1, \ldots, 6$, let a_i and b_i be (the y-coordinates of) the lower and upper boundaries of S_i. We will fix a_i and b_i such that, for any point p in S_i, each of pl, pr, and pt intersects either a_i or b_i.

Observe that, for any point in S_i, it holds that its lower stubs intersect a_i if

$$3/4 \cdot b_i \leq a_i, \tag{4.1}$$

whereas its upper stubs intersect b_i if $a_i + (1 - a_i)/4 = (3a_i + 1)/4 \geq b_i$., i.e., if

$$a_i \geq \frac{4b_i - 1}{3}. \tag{4.2}$$

In addition to the conditions in Equation 4.1 and 4.2, we want to determine the width $b_i - a_i$ in such a way that strip S_i contains at most 5 points. Let

$$c_i = \frac{3}{4} \cdot b_i$$

be the y-coordinate where the lower stubs of points on b_i end. We identify c_i with the line $y = c_i$. For any point p in S_i, let I_p be the part of c_i delimited by the lower stubs of p. Observe that, for $p, q \in S_i$ with $q \neq p$, it holds that I_p and I_q are disjoint. This is due to the fact that the upper stubs of p and q both intersect b_i.

Let δ_p be the length of I_p. We say that p *consumes* δ_p. By the intercept theorem, we obtain that $\delta_p/1 \geq (a_i - c_i)/a_i$ (which is what a point on a_i would consume).

Assume now that there were six points $q_1, \ldots, q_6$ in S_i from left to right. We may assume that there are at least three $j \in \{1, \ldots, 5\}$ such that q_j has a lower or equal y-coordinate than q_{j+1} (otherwise consider the points from right to left).

Consider now such a j. Consider also the parallel g of the right stub of q_{j+1} through the end point of the left stub of q_{j+1}. Since the right stub of q_{j+1} is steeper than the right stub of q_j, it follows that g intersects c_i to the right of I_{q_j}. Hence, we may assume that q_{j+1} consumes even the segment on c_i between the intersection point of g and c_i and the intersection point of the right stub with c_i. This segment has the same length as the distance between the end points of the two lower stubs of q_{j+1}, i.e., $1/4$. Hence, the six points together consume at least

$$3 \cdot \frac{1}{4} + 3 \cdot \frac{a_i - c_i}{a_i} = \frac{3}{4} + 3 \cdot \frac{a_i - 3/4 \cdot b_i}{a_i}$$

which has to be lower than one. Hence, if we choose a_i such that

$$\frac{3}{4} + 3 \cdot \frac{a_i - 3/4 \cdot b_i}{a_i} \geq 1$$

then there cannot be 6 points in S_i. Resolving this equation, we obtain

$$a_i \geq \frac{9}{11} \cdot b_i \tag{4.3}$$

Note that Equation 4.3 automatically implies Equation 4.1. Combining Equation 4.2 with Equation 4.3 yields the following values:

$$\begin{array}{rcccl} & & 3/4 & = & b_1, \\ a_1 & = & 2/3 & = & b_2, \\ a_2 & = & 5/9 & = & b_3, \\ a_3 & = & 5/11 & = & b_4, \\ a_4 & = & 45/121 & = & b_5, \\ a_5 & = & 405/1331 & = & b_6, \text{ and} \\ a_6 & = & 1/4. & & \end{array}$$

Hence, the middle part consists of 6 strips each with at most 5 points. Summarizing, we obtain the following lemma.

Lemma 4.17. *The middle strip S contains at most 30 points.*

4.3.3 The Middle Part of the Bottom Strip

We consider the rectangle $B = [1/4, 3/4] \times [0, 1/4]$ of length $1/2$ and height $1/4$ between the cells C_l and C_r. Similarly as for the middle strip, we construct five cells B_1 to B_5 such that all stubs to the extreme points r, l and t cross the cell boundaries. We denote the left and right boundaries of cell B_i by a_i and b_i and set $a_1 = 1/4$, $b_1 = a_2 = 1/3$, $b_2 = a_3 = 4/9$, $b_3 = a_4 = 5/9$, $b_4 = a_5 = 2/3$, $b_5 = 3/4$. Clearly, for every point p in any cell B_i, it holds that the stubs pl and pr intersect the left and right boundaries a_i and b_i, respectively. For the upper stub pt, we only know that it crosses the horizontal line $y = 1/4$, but

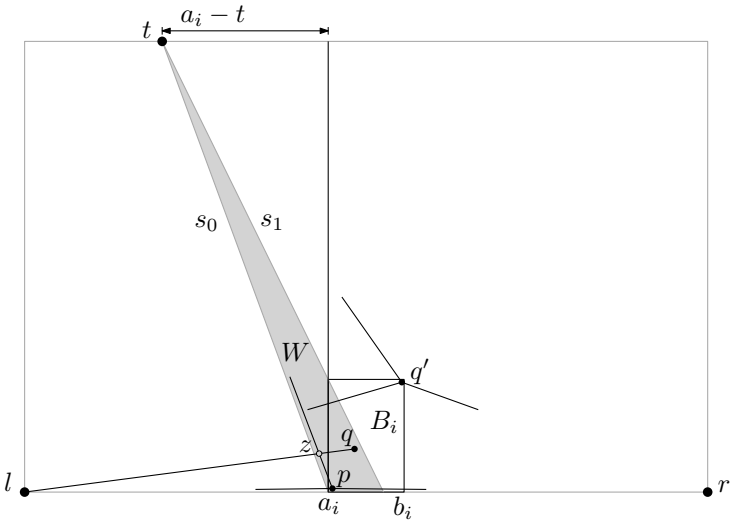


FIGURE 4.11: A cell B_i with a medial point q' and a lateral point p whose stubs do not intersect. The two lateral points p and q cannot both exist since the stubs pt and ql must intersect. (The vertical axis in this figure is scaled by 2/3.)

not necessarily the upper boundary of B_i. We say that a point p is a *medial* point in cell B_i if its stub pt does intersect the upper boundary of B_i. We observe that no two medial points can lie in the same cell B_i without causing stub intersections. Thus, for another point q in B_i, the stub qt must intersect either a_i or b_i. We call such a point a *lateral* point.

In the following, we show that there can be at most one lateral point in any cell B_i. Without loss of generality, we can assume that $t < a_i$. We consider the two rays s_0 and s_1 from t through the two corners $(a_i, 0)$ and $(a_i, 1/4)$ of B_i, see Figure 4.11. These two rays define a wedge W. Let p and q be two lateral points in B_i; then p and q must lie in $B_i \cap W$. Let p be the point whose ray from t is left of the ray from t through q. Then the two line segments $\overline{ql}$ and $\overline{pt}$ must intersect in a point z (otherwise the stubs pr and qt would necessarily cross). Let $\delta_q = |\overline{qz}|/|\overline{zl}|$. To avoid a crossing between the stubs pt and ql, we need that $|qz| \geq 1/4(|qz| + |zl|)$, or equivalently, that $\delta_q \geq 1/3$.

Using the intercept theorem, we observe that δ_q is maximized if p lies on s_0, q lies on s_1, and they both lie on the x-axis. We apply the intercept theorem once more for the line $x = a_i$ and the line supported by s_1 to show that in this case $|\overline{qz}| = (a_i - t)/3$. Using $|\overline{zl}| = a_i$, we get $\delta_q = 1/3 \cdot (1 - t/a_i) < 1/3$. This contradicts $\delta_q \geq 1/3$. Thus, each cell B_i contains at most one lateral and at most one medial point.

Summarizing, we obtain the following lemma.

Lemma 4.18. *The lower rectangle B contains at most 10 points.*

4.3.4 The Left and the Right Part of the Upper Strip

In the following, we consider the rectangles $L = [0, 3t/4] \times [3/4, 1]$ and $R = [(3t+1)/4, 1] \times [3/4, 1]$, separated by the upper central square C_t, which has size $1/4 \times 1/4$, is adjacent to t, and is defined by the stubs tl and tr.

We subdivide R into five height-1/4 rectangles $R_1, \ldots, R_5$, from left to right, and analyze how many points each rectangle can contain at most. The analysis for L is symmetric.

Note that the two lower stubs of any point in R intersects the horizontal line $y = 3/4$. To make sure that the upper stub of any point in cell R_i intersects the left boundary with x-coordinate a_i of R_i, the x-coordinate b_i of the right boundary of R_i has to fulfill $(b_i - t)/4 \geq b_i - a_i$ and, hence, $b_i \leq (4a_i - t)/3$. (Note that we assume that the right boundary of R_i is not part of R_i.) This yields the following boundaries.

$$\begin{array}{rcrcl} & & 3/4 \cdot t + 1/4 & = & a_1, \\ b_1 & = & 2/3 \cdot t + 1/3 & = & a_2, \\ b_2 & = & 5/9 \cdot t + 4/9 & = & a_3, \\ b_3 & = & 11/27 \cdot t + 16/27 & = & a_4, \\ b_4 & = & 17/81 \cdot t + 64/81 & = & a_5, \text{ and} \\ b_5 & = & 1. & & \end{array}$$

Claim 4.19. *The lower stubs of two points in the same cell are nested.*

Proof. Let p be a point in a cell R_i and consider the line s through l and p. Assume there is a point q above p such that the lower stubs of p and q are not nested. Then q has to be to the right of s. However, by the way the width of R_i is constructed, the left stub of q intersects the vertical line a_i. So, the left stub of q intersects the right stub of p. □

Now we analyze how many points can be stacked on top of each other in each subrectangle, depending on its width, its distance to t, and the *l-shadow* of the stub tr, i.e., the set of all points p such that the stubs tr and pl intersect. We first consider R_1.

Claim 4.20. *R_1 contains at most one point.*

Proof. Observe that the left stub of any point p in R_1 must leave R_1 through its bottom edge, see Figure 4.12. Otherwise p would lie in the l-shadow of tr. Hence, there are no two points with nested lower stubs in R_1. Otherwise the left stub of the upper point would intersect the upper stub of the lower point. There are also no two points side by side in R_1, because otherwise the right bottom stub of the left point crosses the left bottom stub of the right point. □

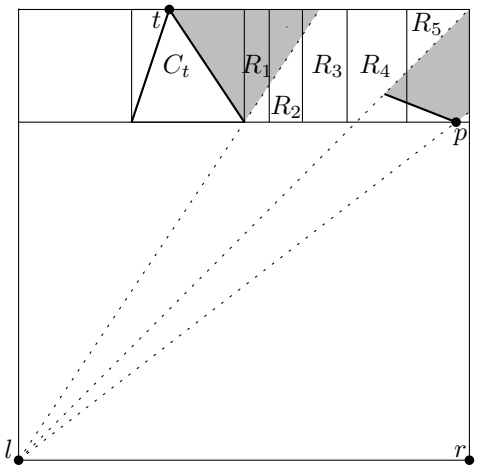


FIGURE 4.12: The lines from point l to the endpoints of the stubs tr and pt give rise to (gray) l-shadows where no point can be placed.

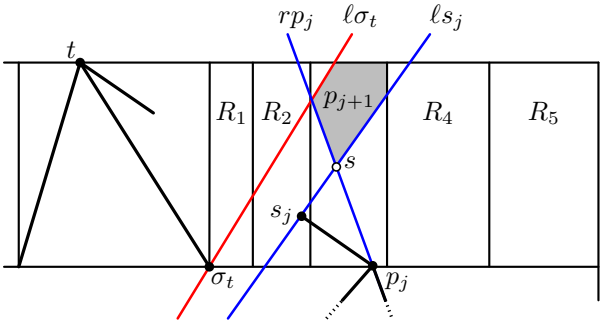


FIGURE 4.13: The shaded polygon indicates the possible area for p_{j+1}.

We now consider $R_2, \ldots, R_5$, see Figure 4.13 for an illustration. Let

$$\sigma_t = t + (r - t)/4 = \frac{1}{4}(3t + 1, 3)$$

be the endpoint of the stub tr. We use the following two observations to bound the number of vertices in R_i.

(O1) No vertex of R_i is above the line $\overline{\ell\sigma_t}$, since otherwise its left stub would intersect the stub tr.

(O2) The y-coordinate of any vertex in R_i is between 3/4 and 1.

Depending on t, these two observations yield upper bounds on the number of vertices in $R_2, \ldots, R_5$ and, symmetrically, in $L_2, \ldots, L_5$. In summary, we obtain the following.

Lemma 4.21. *Together, L and R contain at most 19 vertices.*

Proof. Let $i \in \{2,3,4\}$ and let $p_j = (x_j, y_j), j = 1, \ldots, m$ be the vertices in R_i with ascending y-coordinates. For $j = 1, \ldots, m-1$, let

$$s_j = p_j + (t - p_j)/4$$

be the endpoint of the top stub $p_j t$. To make sure that the left stub of p_{j+1} and the upper stub of p_j do not intersect, p_{j+1} has to be above the line ls_j. For large t, we exploit observation (O1); for small t, we exploit (O2). Putting things together, we will obtain, for each region R_i, upper bounds on the number of points that R_i can contain, for any $t \in [0,1]$.

Case I: t is large.

The condition that no vertex of R_i is above the line $\overline{\ell\sigma_t}$ means that

$$\text{slope}(\overline{\ell\sigma_t}) \geq \text{slope}(\overline{\ell p_j}) \text{ for } j = 1, \ldots, m. \tag{4.4}$$

Further, we know that

$$\text{slope}(\overline{\ell p_j}) < \text{slope}(\overline{\ell s_j}) \leq \text{slope}(\overline{\ell p_{j+1}}) \text{ for } j = 1, \ldots, m-1. \tag{4.5}$$

Observe that the slope of $\overline{\ell s_j}$ increases with the slope of $\overline{\ell p_j}$. Now we claim the following.

Claim 4.22. *If the slope of $\overline{\ell p_j}$ is fixed, then the slope of $\overline{\ell s_j}$ increases when the distance of p_j from ℓ decreases.*

Proof. We first show this claim. Let q_1 and q_2 be two points in R that are co-linear with $\ell = (0,0)$. Then, there is a $0 < c < 1/t$ (actually, we even have $c < 3/(3t+1)$) such that $q_i = (x_i, cx_i), i = 1, 2$. Let the endpoint of stub $q_i t$ be

$$s^i := q_i + (t - q_i)/4 = (3x_i + t, 3cx_i + 1)/4.$$

So the slope of $\overline{ls^i}$ is $(3cx_i + 1)/(3x_i + t)$. Hence, the slope of $\overline{ls^1}$ exceeds the slope of $\overline{ls^2}$ if and only if

$$\begin{aligned} \frac{3cx_1 + 1}{3x_1 + t} > \frac{3cx_2 + 1}{3x_2 + t} \quad &\Leftrightarrow \quad 3x_2 + 3cx_1 t > 3x_1 + 3cx_2 t \\ &\Leftrightarrow \quad ct(x_1 - x_2) > (x_1 - x_2) \\ &\Leftrightarrow \quad x_1 < x_2 \end{aligned}$$

The last equivalence holds since $ct < 1$ for any t. This finishes the proof of our claim. □

Now we continue the proof of Lemma 4.21. By Claim 4.22, the number of vertices $p_1, \ldots, p_m$ in R_i with the properties in Equations 4.4 and 4.5 is *maximized* if $p_1 = (b_i, 3/4)$ and p_{j+1} is the intersection point of the line $\overline{\ell s_j}$ with the vertical line at $x = b_i$; see Figure 4.14. Then, the x-coordinate of s_j is

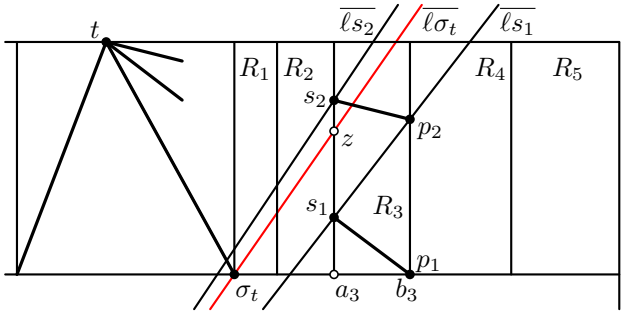


FIGURE 4.14: What is the smallest j for which the slope of $\overline{\ell s_j}$ exceeds that of $\overline{\ell \sigma_t}$?

$a'_i := b_i - (b_i - t)/4 = (3b_i + t)/4$. For $i = 1, \ldots, 4$, we have $a'_i = a_i$. We obtain the following y-coordinates:

$$\begin{aligned} y(s_j) &= y_j + (1 - y_j)/4 = (3y_j + 1)/4 \\ y(p_{j+1}) &= \frac{y_j + (1 - y_j)/4}{a'_i} \cdot b_i = \frac{b_i(3y_j + 1)}{3b_i + t} \\ y(s_1) &= 3/4 + \frac{1 - 3/4}{4} = \frac{13}{16} \end{aligned}$$

The slope of the line $\overline{\ell \sigma_t}$ is less than the slope of the line $\overline{\ell p_{j+1}}$ if the intersection point

$$z = \left(a'_i, \frac{3}{3t + 1} a'_i \right)$$

of the line $\overline{\ell \sigma_t}$ with the vertical line at $x = a'_i$ is below s_j. This means that, if z is below s_j, p_{j+1} cannot lie in R_i (due to Observation (O1)). Hence, depending on t, we obtain the following upper bounds on the number of points in R_i.

For $i = 2$, we plug in $b_2 = (5t+4)/9$ and $a'_2 = a_2 = (2t+1)/3$ into the equation above and obtain the following y-coordinates:

$$\begin{aligned} y(p_{j+1}) &= \frac{1/9(5t + 4)(3y_j + 1)}{1/3(5t + 4) + t} = \frac{(5t + 4)(3y_j + 1)}{24t + 12} \\ y(z) &= \frac{2t + 1}{3t + 1} \\ y(p_2) &= \frac{5t + 4}{24t + 12} \cdot 13/4 \\ y(s_2) &= \frac{1}{4} \left(\frac{5t + 4}{24t + 12} \cdot \frac{39}{4} + 1 \right) = \frac{1}{4} \cdot \frac{97t + 68}{32t + 16} \end{aligned}$$

Hence, R_2 contains at most one vertex if $13/16 = y(s_1) > y(z) = \frac{2t+1}{3t+1}$, that is, if $t > 3/7$, which holds if $t \geq 0.43$. There are at most two vertices in R_2 if $y(s_2) > y(z)$, that is, if

$$\frac{1}{4} \cdot \frac{97t + 68}{32t + 16} > \frac{2t + 1}{3t + 1},$$

which turns out to be true for any $t > 0$.

For $i = 3$, we get the following y-coordinates:

$$\begin{aligned} y(p_{j+1}) &= \frac{1/27(11t+16)(3y_j+1)}{1/9(11t+16)+t} = \frac{(11t+16)(3y_j+1)}{60t+48} \\ y(z) &= \frac{5t+4}{9t+3} \\ y(p_2) &= \frac{11t+16}{60t+48} \cdot 13/4 \\ y(s_2) &= \frac{1}{4}\left(\frac{11t+16}{60t+48} \cdot 39/4 + 1\right) = \frac{1}{4} \cdot \frac{223t+272}{80t+64} \end{aligned}$$

Hence, there is at most *one* vertex in R_3 if $13/16 = y(s_1) > y(z) = (5t + 4)/(9t + 3)$, that is, if $t > 25/37$, which holds if $t \geq 0.68$. There are at most *two* vertices in R_2 if

$$\frac{1}{4} \cdot \frac{223t+272}{80t+64} > \frac{5t+4}{9t+3}, \text{ that is, if } t > -\frac{557}{2 \cdot 407} + \sqrt{\left(\frac{557}{2 \cdot 407}\right)^2 + \frac{208}{407}},$$

which holds if $t \geq 0.31$.

For $i = 4$, we analogously obtain that R_4 contains at most *one* vertex if

$$\frac{13}{16} > \frac{11t+16}{27t+9}, \text{ that is, if } t > 139/175,$$

which holds, for example, if $t \geq 0.8$. The cell R_4 contains at most *two* vertices if $t \geq 0.55$.

For $i = 5$, we have that R_5 contains at most *one* vertex if $13/16 > (3t + 9)/(12t + 4)$, that is, if $t > 23/27$, which holds, for example, if $t \geq 0.86$. There are at most two vertices in R_5 if $t \geq 0.68$.

We summarize our upper bounds in the following table.

	$t \geq 0$	$t \geq 0.31$	$t \geq 0.43$	$t \geq 0.55$	$t \geq 0.68$	$t \geq 0.8$	$t \geq 0.86$
R_2	2	2	1	1	1	1	1
R_3		2	2	2	1	1	1
R_4				2	2	1	1
R_5					2	2	1

Case II: t is small.

It remains to compute upper bounds on the number of points in R_3, R_4, and R_5 for small values of t. For this, we use (O2) as mentioned earlier. To bound the y-coordinates of the vertices in R_i, observe that since the lower stubs of any

vertices in R_i have to be nested, p_{j+1} has to lie above the line $\overline{rp_j}$. Hence, p_{j+1} has to be above the intersection point s of $\overline{rp_j}$ and $\overline{ls_j}$; see Figure 4.13.

Recall that a line through two points (X_1, Y_1) and (X_2, Y_2) can be expressed by

$$y = \frac{Y_1 - Y_2}{X_1 - X_2} \cdot x + \frac{Y_2X_1 - Y_1X_2}{X_1 - X_2}$$

So, we obtain the following equations for the lines $\overline{rp_j}$ and $\overline{ls_j}$:

$$\overline{rp_j}: \qquad y = -\frac{y_j}{1 - x_j}x + \frac{y_j}{1 - x_j}$$
$$\overline{ls_j}: \qquad y = \frac{3y_j + 1}{3x_j + t}x$$

Intersecting these two lines yields the intersection point

$$s = \left(\frac{(3x_j + t)y_j}{y_j(t + 3) + 1 - x_j}, \frac{(3y_j + 1)y_j}{y_j(t + 3) + 1 - x_j}\right).$$

Given that p_{j+1} lies above s and given that $x_j \geq a_i$, we obtain the recursion

$$y_{j+1} > \frac{(3y_j + 1)y_j}{y_j(t + 3) + 1 - a_i}. \tag{4.6}$$

Recall now that $a_i = \lambda_i t + 1 - \lambda_i$ with $\lambda_i = 1 - 4^{i-2}/3^{i-1} < 3/4$ for $i = 2, \ldots, 5$ also depends on t. Hence, the lower bound

$$\frac{(3y_j + 1)y_j}{y_j(t + 3) + 1 - a_i} = \frac{3y_j + 1}{t + 3 + (1 - t)\lambda_i/y_j} = \frac{3y_j + 1}{t(1 - \lambda_i/y_j) + 3 + \lambda_i/y_j}$$

for y_{j+1} decreases with increasing t and decreasing y_j. This means that the number of points that fit into R_i such that Equation 4.6 is still fulfilled becomes larger if we choose the y-coordinates of the points as small as possible.

Hence, starting with $y_1 = 3/4$, we get lower bounds for $y_2, \ldots, y_m$ by assuming equation in (4.6). The largest i with $y_i \leq 1$ is an upper bound for the number of vertices in R_i.

Depending on t, observation (O2), that is, $y_j \leq 1$ for $j = 1, \ldots, m$, yields the following upper bounds on the number of vertices in R_i.

	$t \leq 0.31$	$t \leq 0.54$	$t \leq 0.55$	$t \leq 0.68$
R_3	3			
R_4	2	3	3	
R_5	2	2	3	3

This finishes the analysis of case II.

Now we can put things together. The resulting upper bounds for the regions of R and, symmetrically, for L are summarized in Table 4.1. Adding up the

$t \geq$	0.00	0.14	0.20	0.31	0.32	0.43	0.45	0.46	0.54	0.55	0.57	0.68	0.69	0.80	0.86
R_1	1														
L_1	1														
R_2	2					1									
L_2	1										2				
R_3	3			2								1			
L_3	1				2								3		
R_4	2			3					2				1		
L_4	1		2				3					2			
R_5	2							3			2			1	
L_5	1	2			3			2							
total	15	16	17	17	**19**	18	**19**	18	**19**	18	**19**	17	17	16	15

TABLE 4.1: Upper bounds for the number of vertices in the cells of L and R; $L \cup R$ contains at most 19 vertices.

bounds for each resulting t-subinterval of $[0, 1]$ yields that $L \cup R$ contains at most 19 points, namely if $0.32 < t < 0.43$, $0.45 < t < 0.46$, $0.54 < t < 0.55$, or $0.57 < t < 0.68$. □

4.3.5 The $(1/4 \times 1/4)$-Squares C_l, C_r, and C_t

Our approach for this part follows a suggestion of Gašper Fijavž. We consider the square C_l; for the two other squares C_r and C_t, we can argue analogously and get the same bound. Let $l, p_1, \ldots, p_k$ be the set of points contained in C_l.

First, we observe that the stubs from $p_1, \ldots, p_k$ to t and r intersect the upper and right boundary of C_l. Hence, the points $p_1, \ldots, p_k$ together with their stubs to r and t form a nested structure. This means that we can order the points such that, for $i = 2, \ldots, k$, the point p_i lies between the stubs of p_{i-1}, the point p_k being innermost. Now we define $\alpha_1, \alpha_2, \ldots, \alpha_k$ to be the angles at point r formed by the lines $\overline{rl}$ and $\overline{rp_i}$. Analogously, we have angles $\beta_1, \ldots, \beta_k$ at point t. We consider only the angles of type α_i. Analogous observations hold for the angles of type β_i, and the resulting bounds are the same.

From the nesting, we see that the sequence α_i, $i = 1, \ldots, k$ is monotonously increasing. Even stronger, we have the following claim.

Claim 4.23. *For $1 < i \leq k$ it holds that $\alpha_i \geq 1.3 \cdot \alpha_{i-1}$.*

Proof. Consider the segment from point l to p_i. We subdivide it into four segments of equal length; see Figure 4.15(a). This defines the four angles $\gamma_1, \ldots, \gamma_4$ by the connecting lines from point r. We have $\alpha_i = \gamma_1 + \cdots + \gamma_4$ and $\alpha_{i-1} \leq \gamma_1 + \gamma_2 + \gamma_3 =: \gamma$. It remains to prove that $\alpha_i \geq 1.3\gamma$.

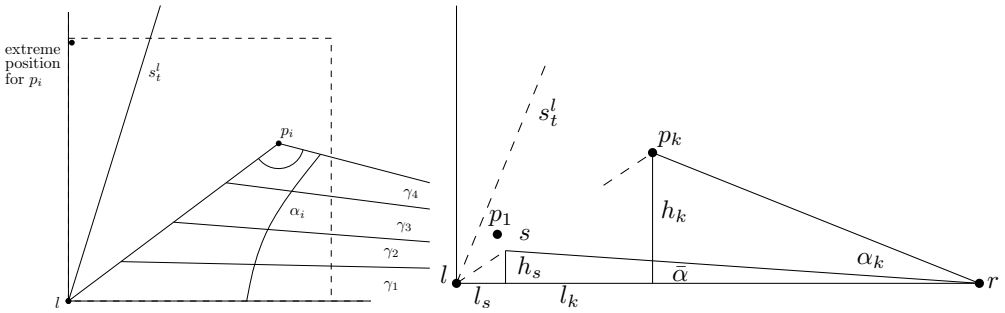


FIGURE 4.15: (a) Angles increase by a factor ≥ 1.3. (b) Ratio between the smallest and the largest angle.

The ratio of γ_4 to γ and, hence, the ratio of α_i to γ is smallest if the angle formed by r, p_i, l is minimized, i.e., if p_i lies in the upper left corner of C_l. Hence, $\alpha_i/\gamma \geq \arctan(1/4)/\arctan(3/16) > 1.3$. Note that in general p_i must be to the right of stub lt, so the ratio is even slightly better. □

Next, we restrict the range of the smallest and largest angle. Then we can easily compute the number k of points.

Let s be the endpoint of the stub lp_k. Consider the two lines $\overline{ts}$ and $\overline{rs}$. The point p_1, which defines the angle α_1 and β_1 respectively, has to lie either above $\overline{rs}$ or to the right of $\overline{ts}$ or both. We assume, without loss of generality, the first case, so the angle formed by $l, r, s := \bar{\alpha} \leq \alpha_1$. We set the length of the base line, which is the distance between l and r, to be $d = 1$.

We compute $\tan(\bar{\alpha}) = h_s/d - l_s = (h_k/4)/d - l_s$ and $\tan(\alpha_k) = h_k/(d - 4l_s) = h_k/(d - 4l_s)$; see Figure 4.15(b), where h_k and h_s are the minimum distances of p_k and s, respectively, to the base line $\overline{lr}$. This yields the ratio

$$\frac{\tan\bar{\alpha}}{\tan\alpha_k} = \frac{h_k}{4(d-l_s)}\frac{d-4l_s}{h_k} = 1 - \frac{3d}{4(d-l_s)} = 1 - \frac{3}{4(1-l_s/d)}.$$

Using $l_s \leq (1/4)^2$, this yields $\tan\bar{\alpha}/\tan\alpha_k \geq 1/5$. From the Taylor series expansion of the tangent function we know that $\tan\alpha > \alpha$, for all $0 < \alpha < \pi/2$, in particular $\tan\alpha = c_\alpha\alpha$ with $c_\alpha > 1$ monotonically increasing with α.

Hence, we conclude that

$$\frac{\alpha_1}{\alpha_k} = \frac{c_{\alpha_k}\tan\alpha_1}{c_{\alpha_1}\tan\alpha_k} > \frac{\tan\bar{\alpha}}{\tan\alpha_k} \geq 1/5.$$

This yields $\alpha_k \geq (1.3)^{k-1} \cdot \alpha_1 > (1.3)^{k-1} \cdot 1/5 \cdot \alpha_k$, which in turn implies $k < \log 5/\log 1.3 + 1 \leq 6.2$.

If p_1 lies to the right of $\overline{ts}$, we analogously obtain

$$\frac{\beta_1}{\beta_k} > 1 - \frac{3}{4(1 - l_s/d)}$$

where $d > 1$ is the distance of l and t and $l_s \leq 1/4 \cdot \sqrt{2}/4$ is the length of the projection of the segment $\overline{ls}$ to the line lt, hence $\beta_1 > 1/6 \cdot \beta_k$.

Arguing along the lines of the first case, we get $\beta_k > (1.3)^{k-1} \cdot 1/6 \cdot \beta_k$, and derive $k < 7.9$.

Lemma 4.24. *The squares C_l, C_t, and C_r contain at most eight points each.*

This finishes the proof of Theorem 4.16.

Granacher [97] showed that for $n > 96$ the complete bipartite graph $K_{n,n}$ has no axis-symmetric 1/4-SHPED drawing such that all edges cross the axis of symmetry. Also Granacher generalized our proof leading to a large expression $EXP(\delta)$ depending on the stub-edge-ratio δ, that is too lengthy to be noted here explicitly.

Theorem 4.25 ([97])**.** *For any $n > EXP(\delta)$ the graph K_n does not admit a δ-SHPED.*

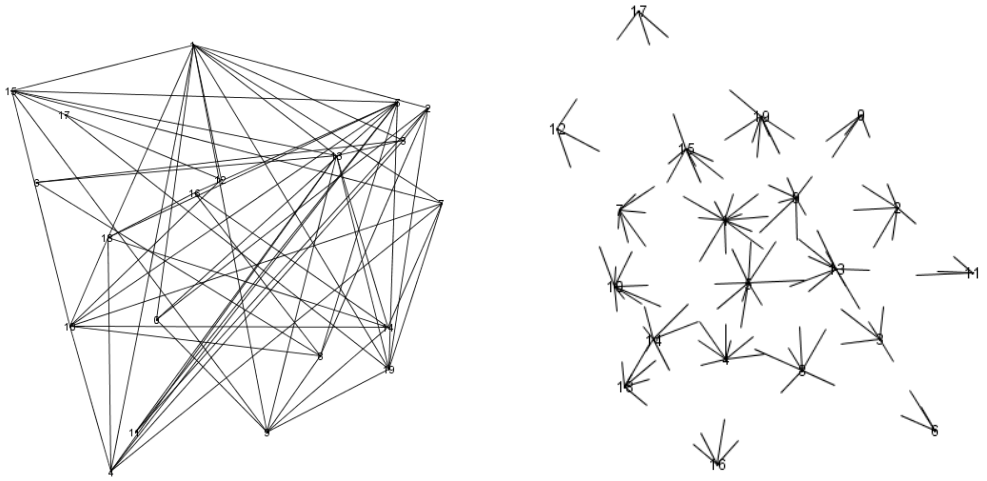
4.4 1/4-SHPED Spring Embedder

In this section we combine 1/4-SHPED with the pring embedder idea, see Figure 4.16. We start with an introduction in Section 4.4.1 motivating our approach. Because our model slightly differs from literature, we continue with necessary definitions in Section 4.4.2 and sketch computation of 1/4-SHPEDs with constraints similar to linear programming. The main algorithm is presented in Section 4.4.3 and is finally extensively evaluated in Section 4.4.4.

4.4.1 Introduction

Motivated by the fact that several classes of graphs admit a 1/4-SHPED (see Section 4.2) and the complete graphs $K_n, n > 164$ do not admit a 1/4-SHPED (see Section 4.3), we hope that our algorithm supports the search for a 1/4-SHPED of K_{17} if it exists.

Apart from theoretical results we want an algorithm for practical applications. The algorithm should produce a 1/4-SHPED for all graphs admitting a 1/4-SHPED in theory, but also for many other (randomly created) graphs if it exist. We expect from the algorithm ideas for further classes of graphs admitting a 1/4-SHPED. Therefore our first attempt was a computation similar to linear



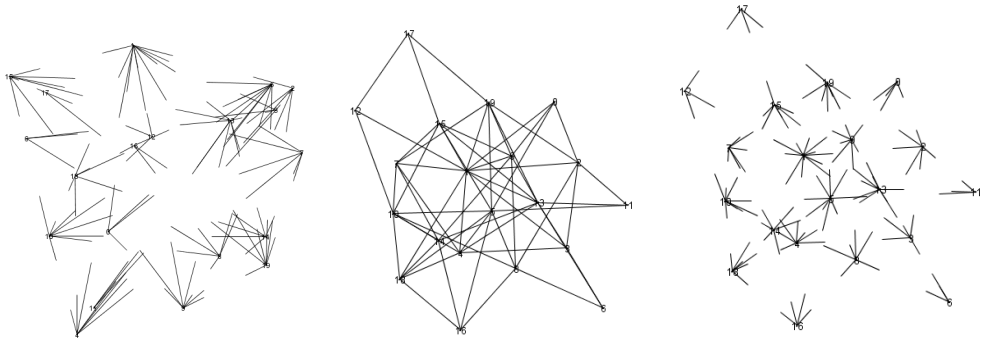
(a) Traditional straight-line drawing (b) 1/4-SHPED

FIGURE 4.16: (a) A randomly created graph G with 20 vertices and 60 edges has 383 crossings in the traditional straight-line model. (b) layout of G after application of our 1/4-SHPED spring embedder within 2 iterations and with a spring embedder [193] as preprocessing step.

programming, which failed in practical applications. So spring embedder [87] seem to be a useful concept, since they tend to produce symmetric drawings with few crossings. We develop our algorithm following the Fruchterman algorithm as long as possible. In the sequel of this paper we always use the yFiles Organic layouter [193] whenever we apply a spring embedder for the traditional model.

A force-directed drawing algorithm can be used to visualize any non-planar graph, in particular if no structural information is given in advance. Notice that even some spring embedder for the traditional straight-line model do not produce planar drawings for planar graphs. Since some graphs do not admit a 1/4-SHPED (see Section 4.3), we allow crossings while minimizing their amount and insist just on the stub-edge-ratio of 1/4. In order to be consistent with the definitions we call a 1/4-SHPED containing crossings a 1/4-nSHPED (n for nearly). In such a way we can directly turn any drawing into a 1/4-nSHPED. We describe a spring embedder algorithm that aims at producing a 1/4-SHPED by resolving crossing in the 1/4-nSHPED. By abuse of notation we call our algorithm a 1/4-SHPED spring embedder.

Spring embedder in general have a different goal compared to our goals. They aim for symmetry by spreading vertices uniformly for a drawing. When turning such a drawing into a 1/4-nSHPED, some crossings in the traditional model were good (those which disappear when an edge part is removed) and some crossings in the traditional model were bad (those which stay visible when an edge part is removed). Figure 4.17 shows that a spring embedder does not



(a) (b) (c)

FIGURE 4.17: (a) 1/4-nSHPED of Figure 4.16(a) having 71 crossings. (b) Traditional straight-line drawing of Figure 4.16(a) laid-out with spring embedder [193] having 116 crossings. (c) 1/4-nSHPED of (b) having 4 crossings.

resolve all crossings when turning a traditional straight-line drawing directly into a 1/4-nSHPED. Thus general spring embedder essentially do not support 1/4-SHPEDs. Our 1/4-SHPED spring embedder instead uses forces that push vertices away from stubs so that all crossings are possibly resolved. We redefine the forces and evaluate the algorithm finally. For optimal results the 1/4-SHPED spring embedder is applied after one application of spring embedder [193] as preprocessing step. Our embedder is based on a first implemented prototype [129].

4.4.2 Preliminaries

In this section we introduce and recall some necessary definitions, in particular the definitions needed to distinguish between drawings with or without crossings. Let $G{=}(V, E)$ be a graph and let Γ_G a straight-line drawing of G. A *nearly partial edge drawing* nPED is a drawing of G with respect to Γ_G that consists of only $\bigcup_{e=(v,w)\in E}(s_v^e \cup s_w^e)$, where s_v^e is the stub of e incident to v. A PED is an nPED without crossed stubs. Our focus is on 1/4-SHPEDs and 1/4-nSHPED.

As an intermediate result we provide a quadratically constrained quadratic program (QCQP) that produces a 1/4-SHPED if it exists[1]. In particular, it returns the positions of the vertices inside a square of size $L \times L$ for a given positive integer L. Our hope for this QCQP is that either we can prove or disprove the existence of a 1/4-SHPED for the complete graph with 17 vertices, or that we get at least ideas for geometric constraints and properties of 1/4-SHPEDs.

[1]The complexity of solving a QCQP is NP-hard.

Let $G{=}(V,E)$ be an arbitrary graph and let $L > 0$ be an integer arbitrarily chosen, but fixed. We denote with x_i (respectively y_i) the x-coordinate (respectively y-coordinate) of the vertex $v_i \in V{=}\{v_1,\ldots,v_n\}$ and introduce the constraints $0 \leq x_i \leq L$ and $0 \leq y_i \leq L$ for all $v_i \in V$. For simplicity we assume that any two vertices share neither a common x-coordinate $x_i \neq x_j$ nor a common y-coordinate $y_i \neq y_j$ for all $1 \leq i \neq j \leq n$. For every pair of edges $(v_i,v_j),(v_k,v_l)$ we want the intersection point of the segments $s_1 = \overline{v_iv_j}$ and $s_2 = \overline{v_kv_l}$ representing the edges to be either in the second or third quarter of s_1 or in the second or third quarter of s_2. Therefore we solve for every pair of edges $(v_i,v_j),(v_k,v_l)$ the equation system $v_i + t_{ijkl}(v_j - v_i) = v_k + s_{ijkl}(v_l - v_k)$, which describes the intersection of segments s_1 and s_2:

$$t_{ijkl} = \frac{(x_k - x_i)(y_l - y_k) + (x_l - x_k)(y_i - y_k)}{(x_l - x_k)(y_i - y_j) + (x_j - x_i)(y_l - y_k)} \in [0,1]$$
$$s_{ijkl} = \frac{(x_i - x_k)(y_j - y_i) + (x_j - x_i)(y_k - y_i)}{(x_j - x_i)(y_k - y_l) + (x_l - x_k)(y_j - y_i)} \in [0,1]$$

Notice that $t_{klij} = s_{ijkl}$ and $t_{ijkl} = t_{jikl} - 1 = s_{klji} - 1 = s_{lkji} = t_{jilk}$ (since $\overline{v_jv_i} = v_j + t_{jikl}(v_i - v_j) = v_i - (v_i - v_j) + t_{jikl}(v_i - v_j) = v_i + (-1 + t_{t_{jikl}})(v_i - v_j)$). We introduce the constraint $1/4 \leq t_{ijkl} \leq 3/4$ or $1/4 \leq t_{klij} \leq 3/4$, expressing the "or" with a binary variable, see [4]. Notice that, although t_{ijkl}, t_{klij} are fractions, in the last inequalities the fraction can be eliminated by multiplying with the denominator, while the quadratic terms still remain in the inequalities, see the reformulation for $1/4 \leq t_{ijkl} \leq 3/4$:

$$\begin{aligned} & (x_l - x_k)(y_i - y_j) + (x_j - x_i)(y_l - y_k) \\ \leq\ & 4((x_k - x_i)(y_l - y_k) + (x_l - x_k)(y_i - y_k)) \text{ and} \\ & 4((x_k - x_i)(y_l - y_k) + (x_l - x_k)(y_i - y_k)) \\ \leq\ & 3((x_l - x_k)(y_i - y_j) + (x_j - x_i)(y_l - y_k)) \end{aligned}$$

Finally the objective function is not important anymore, since any feasible solution fits into the square of size $L \times L$, because of the invariance of 1/4-SHPEDs under scaling.

We implemented this QCQP in Java and used the Gurobi solver [100] to solve the above problem. The Gurobi solver is not only applicable for LPs and ILPs, but also for QCQPs if the matrix of the problem is positive semidefinite in order to use the polynomial time interior point method. Unfortunately the Gurobi solver returned that the matrix of our problem is indeed not positive semidefinite. Thus this QCQP does not answer the question from the motivation part, whether the complete graph with 17 vertices admits a 1/4-SHPED. So we turn to 1/4-SHPED spring embedder aiming for finding further classes of graphs admitting a 1/4-SHPED, explained in the following.

The input of our algorithm is a graph G, which immediately is turned to a 1/4-nSHPED with respect to a straight-line drawing Γ_G produced with the spring embedder [193] as preprocessing step. The objective is now to minimize the number of crossings of the 1/4-nSHPED by obtaining an appropriate distribution of vertices. We will discuss later when a 1/4-nSHPED becomes a 1/4-SHPED. Figures 4.17(a) and 4.17(c) illustrate 1/4-nSHPEDs, while Figure 4.16(b) illustrates a 1/4-SHPED.

4.4.3 The Algorithm

A spring embedder places vertices adequately distributed in general, but there is still space for improvement, see Figure 4.17(c) in comparison to Figure 4.16(b).

A spring embedder iterates several times and in each iteration every vertex is considered once. When a vertex is considered, the algorithm computes the forces on this vertex and moves the vertex at its calculated place. We adopt this base of the spring embedder algorithm and focus now on how to build the forces that push or pull vertices according to the 1/4-SHPED model.

Let v be a vertex at point p and let w be adjacent to v at point q. The edge (v, w) consists of two stubs s_v, s_w in the 1/4-SHPED model, where $p \in s_v$ and $q \in s_w$. Suppose that s_v is crossed by an edge e at point c. Locally this crossing can be resolved by moving either p away from e or moving q closer to e. In our concept we just consider the first variant, i.e., we move p away from e and call it to *repair* v. The crossing c is not resolved as long as $|s_v| \geq d(p,q)/4$, where $d(p,q)$ denotes the distance from p to q. At the moment when s_v just touches e, the equation $|s_v| = \frac{1}{4}d(p,q)$ holds. Let p' denote the new position of v. Then we want to know the distance d'_{vw} to move v from p to p' (refer to Figure 4.18) such that the crossing becomes a touching point, i.e., $d'_{vw} + d(p,c) = \frac{1}{4}(d'_{vw} + d(p,q)) \Leftrightarrow d'_{vw} = \frac{1}{3}(d(p,q) - 4d(p,c))$. The direction d''_{vw} to move v is clearly $d''_{vw} = \overrightarrow{qp}/d(p,q)$, which will be weighted with the distance d'_{vw} to obtain the force $f_{vw} = d''_{vw}d'_{vw}$ of edge (v, w). We compute the force $f_v = \sum_{(v,w)\in E} f_{vw}$ on vertex v by summing up the forces over all incident edges and repair v with force f_v. We summarize the algorithm in pseudo-code with a constant $I = 200$ of iterations.

This naive algorithm contains some problems to be resolved. We address these problems step by step next.

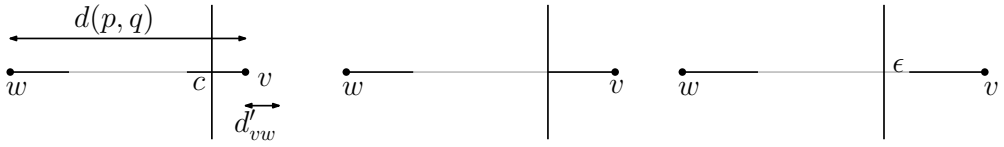


FIGURE 4.18: Shift of v with d'_{vw} to the right, first without ϵ, second with ϵ.

Algorithm 5: 1/4-SHPED spring embedder

Data: graph $G = (V, E)$, max. # I of iterations
Result: 1/4-nSHPED with few crossings
produce drawing Γ_G with spring embedder;
convert Γ_G into 1/4-nSHPED;
for $i = 1$ ***to*** I **do**
  forall the $v \in V$ **do**
    compute force f_v on v;
    move v from p to $p' = p + f_v$;
  end
end

Shifts: Assume that the movement has converted a crossing point to a touching point by shifting a vertex. In theory the crossing is resolved, since stubs are half-open sets. In practice instead the drawing may still look non-planar. To make the drawing looking planar, it suffices to shift the vertex slightly to resolve the crossing also optically. Therefore we introduce an $\epsilon > 0$ to shift the vertex. Precisely we modify the weight of the force to $d'_{vw} = \frac{1}{3}(d(p,q) - 4d(p,c)) + \epsilon$, see Figure 4.18. In practice it turns out that $\epsilon = 0.4$ is an appropriate value.

Small Movements: Sometimes one stub incident to vertex v participates in more than one crossing. In this case all crossings induce a force on v, all of them pointing in the same direction and all of them with possibly a huge value, while it suffices to just resolve the closest crossing. All other crossings farther away are then implicitly resolved. This is done as follows.

Let $c_1, \ldots, c_t$ be the crossings on a stub incident to v such that $d(v, c_i) < d(v, c_{i+1})$ for $i = 1, \ldots, t-1$. We ignore $c_2, \ldots, c_t$ and compute the weight of the force $d'_{vw} = \frac{1}{3}(d(p,q) - 4d(p,c_1)) + \epsilon$ just with respect to c_1, see Figure 4.19.

Shrinks: It may happen that the sum of forces reaches a huge value and the direction of movement causes a vertex to be moved very far away. In practice it turns out that the drawing stretches a lot and the resolution is not good enough to display the drawing properly anymore. To avoid this, the drawing is shrunk to its original size after every iteration.

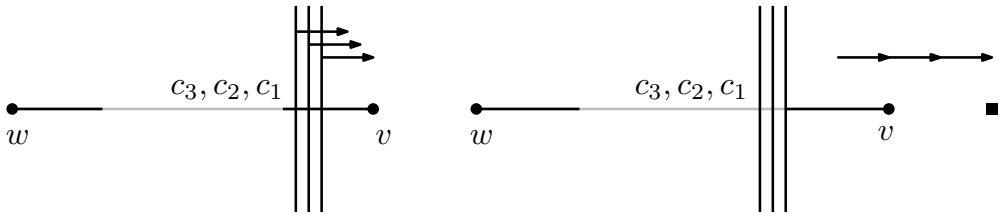


FIGURE 4.19: A shift by only one force defined by c_1 to the right, while ignoring the forces defined by c_2 and c_3, which would result in a position for v marked with the square.

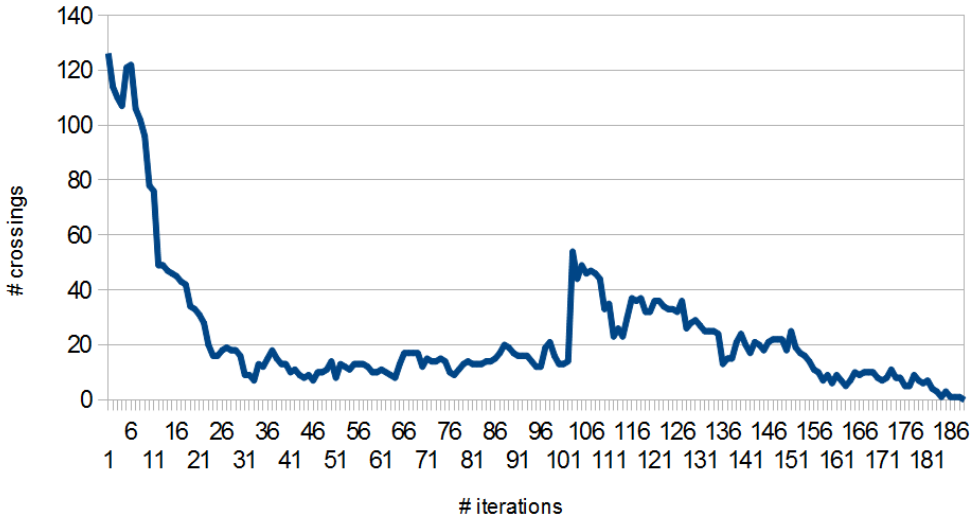


FIGURE 4.20: Illustration of the development of the number of crossings for a complete graph with 14 vertices in circular layout as input. The number of crossings decreases non-monotonically with the number of iterations until the algorithm finishes after 188 iterations without preprocessing.

Let D_i be the drawing after iteration i and let $d(v, w)$ be the Euclidean distance between vertex v and w. The *diameter* $d(D)$ of a drawing D is defined as $\max\{d(v, w) \mid v, w \in D\}$. We aim to shrink the drawing D_{i+1} to the diameter of D_i, i.e., we simply multiply the coordinates of all points with the shrinking-factor $k = d(D_i)/d(D_{i+1})$.

Simply zooming the drawing had bad resolution effects. In practice some resolved crossing became crossing again due to rounding.

Iterations: The number of iterations I is set to 200, which seems to be an appropriate value. If the algorithm produces a drawing D_i with $cr(D_i) = 0$ in one of the iterations $1 \leq i \leq I$, then the algorithm stops and returns D_i as the final 1/4-SHPED. Otherwise the algorithm returns D_i with $cr(D_i) = \min_{1 \leq j \leq I} cr(D_j)$ as final 1/4-nSHPED. The reason that the output is not D_I is explained next.

Non-monotonicity: A movement may increase the number of crossings, but this move might be helpful to decrease this number drastically in a later step. Thus we focus not on decreasing the number of crossings, but on placing the vertices appropriately. Figure 4.20 illustrates that the number of crossings does not decrease monotonically with respect to the number of iterations.

We choose as the final result not the drawing D_I after I iterations, but the one with the smallest number of crossings.

Blockings: It may happen that $f_v \approx 0$ for a vertex v whose incident stubs have no crossings (then $f_v = 0$), or whose incident stubs have crossings such

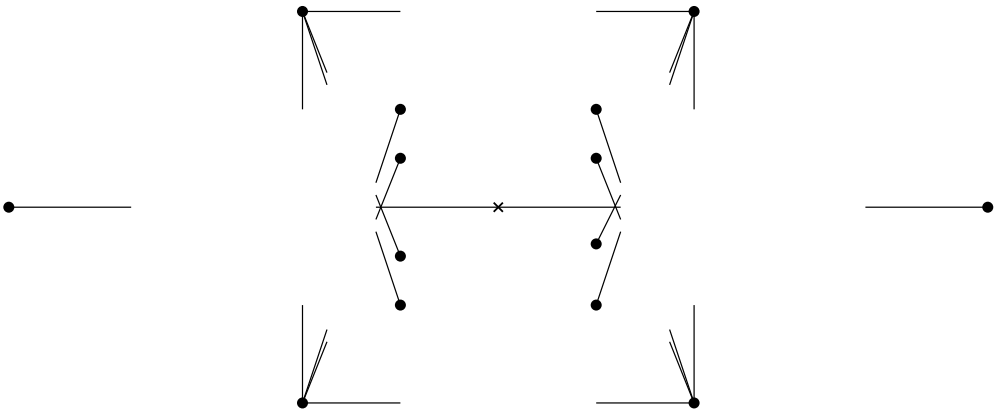


FIGURE 4.21: The vertex marked by a cross can neither be moved to the left, nor to the right. Moving first outer vertices resolves this deadlock for the cross.

that opposite stubs cancel the force. In the latter case a vertex is blocked by surrounding stubs and a movement of v yields no profit or is even impossible, see Figure 4.21 for illustration.

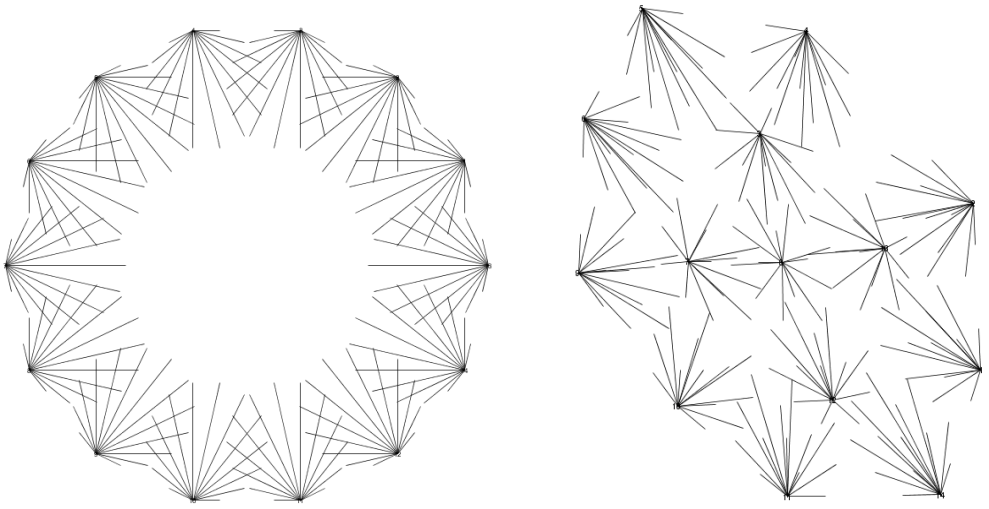
The strategy to tackle this problem is to first focus on the outer vertices. More precisely, we first repair the vertices of the convex hull C_1 of the input drawing D_0 and receive D_{i_1}. Second we repair the vertices of the convex hull C_2 of $D_{i_1} - C_1$ and receive D_{i_2}. Iteratively we repair in round j the convex hull C_j of $D_{i_{j-1}} - C_{j-1}$ and receive D_{i_j} until D_{i_j} is empty ($j \leq m$). We call $C_1, \ldots, C_m$ the levels of the graph. In such a way we repair the drawing in a spiral way such that the vertices of level j build a frame in which vertices of level $j+1$ can be moved. In such a way one iteration consist of m rounds.

This procedure has the advantage that vertices may move from level j to level $j-1$ if level $j-1$ provides some space. In the next iteration they will be considered earlier. We experimented with several orders of the vertices, but this approach was the most reasonable choice.

To conclude, Algorithm 5 produces a 1/4-nSHPED with drastically reduced number of crossings. Very often it produces even a 1/4-SHPED. We refer to Algorithm 5 as our 1/4-SHPED spring embedder.

4.4.4 Experimental Evaluation

First we consider the theoretical complexity of this 1/4-SHPED spring embedder. Afterwards we analyze the real running time and the quality of produced drawings. The final 1/4-SHPED spring embedder is tested using several classes of graphs. Here we present the evaluation with complete graphs K_n, squares of triangular grids T^2 (theoretically analyzed in Section 4.2) and randomly created graphs R_d (respectively R_s) with high (respectively low) number of edges.



(a) (b)

FIGURE 4.22: (a) The complete graph with 14 vertices in circular layout with 126 crossings and (b) the same graph after 188 iterations of our 1/4-SHPED spring embedder without preprocessing with 0 crossings.

All classes were tested on a four-core i5-4750 processor with 8GB RAM. Finally we tested all 11,534 graphs from the Rome graphs and all 1,277 graphs of the North graphs [98] and classified the results into 1/4-SHPED and 1/4-nSHPED.

Complexity: Since the iterations I are bounded by a constant, we focus on a single iteration. In every iteration we consider n vertices for which we compute the force. After moving a vertex by its calculated force, the whole drawing changes. Thus forces cannot be calculated simultaneously and must be re-calculated after every movement. For every vertex v, we consider its incident edges and search for crossings with the remaining edges in $O(m \log m)$ time with a sweep-line algorithm[2]. With the knowledge of the crossings we may compute d''_{vw} directly in constant time as well as d'_{vw} through the closest crossing. In total the algorithm runs in $O(nm \log m)$ time. Note that at iteration i the algorithm shrinks the drawing D_i in $O(n)$ time and computes $cr(D_i)$ in $O(m \log m)$ time. Both complexities are covered by $O(nm \log m)$ time.

Quality of Results: For complete graphs K_n with $n \leq 14$ the algorithm produced a 1/4-SHPED within a few seconds. Even after 1,000 iterations complete graphs with $15 \leq n \leq 18$ vertices could not be converted to a 1/4-SHPED without crossings. This took up to one minute, but the number of crossings has been dramatically reduced compared to the initial state. The following table shows the result ($i\#cr$ = initial number of crossings, $f\#cr$ = final number of crossings) computed from an initial circular layout of the complete graph with

[2]Since our instances had few vertices, we dind't implement a sweep-line algorithm but a naive $O(m^2)$ time approach.

n vertices. We stopped the algorithm after 1,000 iterations. Figure 4.22 illustrates the complete graph with 14 vertices before and after the 1/4-SHPED spring embedder without preprocessing.

n	$i\#cr$	$f\#cr$	n	$i\#cr$	$f\#cr$	n	$i\#cr$	$f\#cr$
10	10	0	13	56	0	16	256	41
11	22	0	14	126	0	17	422	128
12	24	0	15	180	13	18	648	134

TABLE 4.2: The 1/4-SHPED spring embedder (without preprocessing step) decreased the number $i\#cr$ of crossings in the initial circular layout of the complete graph with n vertices to the number $f\#cr$ of crossings in the final drawing.

For squares of triangular grids T^2 with $|T| = n \in \{9, 16, 25, 36\}$ the algorithm produced a 1/4-SHPED within a few seconds. Squares of triangular grids T^2 with $|T| = n \in \{49, 64\}$ were still converted to a 1/4-SHPED, but the algorithm needed up to half a minute. We were not able to test larger problem sizes.

The running time of the algorithm for randomly created graphs R_d, R_s was up to 100 seconds (until 1,000 iterations were accomplished). The graphs had 15 to 30 vertices and 30 to 100 edges. The algorithm reduced the amount of crossings on average from about 100 to 20.

Rome Library: Our 1/4-SHPED spring embedder produced (after 200 iterations) a 1/4-SHPED for 11,507 of the 11,534 graphs of the Rome graphs, the remaining drawings were just 1/4-nSHPED. We started the algorithm with applying a spring embedder algorithm [193] on the graph to obtain a first initial drawing. In the initial drawing we had just 1,110 drawings without crossings, but from all drawings transformed into 1/4-nSHPED without any further modification, we had 9,009 drawings without any crossing. After this preprocessing step we applied the 1/4-SHPED spring embedder with at most 200 iterations for every graph. The average time the algorithm needed (including the preprocessing step) was 0.5 seconds. The average number of crossings in the initial drawings was 26.1 in the traditional straight-line model, and the average number of crossings decreased from 0.4 to ≈ 0 in the 1/4-nSHPED model due to the 1/4-SHPED spring embedder.

From the 2,525 input 1/4-nSHPEDs for the 1/4-SHPED spring embedder, we obtained for 2,498 graphs a 1/4-SHPED, while only for 27, we still had some crossings. See Figure 4.23 for illustration. The average time needed to process these graphs (including the initial preprocessing step) was 2.1 seconds. Here we decreased the average number of crossings from 2.0 to ≈ 0.

Summarizing, we point out that our 1/4-SHPED spring embedder converted 99% of the initial 1/4-nSHPED of the Rome graphs into 1/4-SHPED. On average, we decreased the number of crossings to <1% of the initial 1/4-nSHPED.

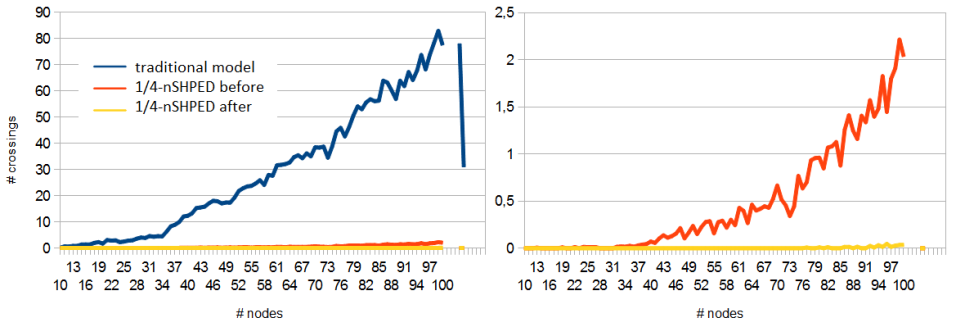


FIGURE 4.23: The number of crossings averaged over all graphs of the Rome graphs with n vertices. The blue line refers to the traditional model after the spring embedder preprocessing step, the red line refers to the 1/4-nSHPED model before application of the 1/4-SHPED spring embedder, and the yellow line refers to the result after application of our 1/4-SHPED spring embedder.

North Graphs: Our 1/4-SHPED spring embedder produced (after 200 iterations) a 1/4-SHPED for 1,239 of the 1,277 graphs of the North graphs, the remaining drawings were just 1/4-nSHPED. Again we started the algorithm with applying a spring embedder algorithm [193] on the graph to obtain a first initial drawing. In the initial drawing we had just 432 drawings without crossings, but from all drawings transformed into 1/4-nSHPED without any further modification we had 1,092 drawings without any crossing. After this preprocessing step we applied the 1/4-SHPED spring embedder with at most 200 iterations for every graph. The average time the algorithm needed (including the preprocessing step) was 10.5 seconds. The average number of crossings in the initial drawings was 33.3 in the traditional straight-line model, and the average number of crossings decreased from 2.9 to 1.6 in the 1/4-nSHPED model due to the 1/4-SHPED spring embedder.

From the 185 input 1/4-nSHPED for the 1/4-SHPED spring embedder, we obtained for 147 graphs a 1/4-SHPED, while only for 38, we still had some crossings. See Figure 4.24 for illustration. The average time needed to process these graphs (including the initial preprocessing step) was 72.5 seconds. Here we decreased the average number of crossings from 20.0 to 11.0.

Summarizing, our 1/4-SHPED spring embedder converted 79% of the initial 1/4-nSHPED of the North graphs into 1/4-SHPED. In average we decreased the number of crossings to 55% of the initial 1/4-nSHPED. Even for the North graphs, that are denser and thus have many more crossings, our embedder improved the drawings significantly.

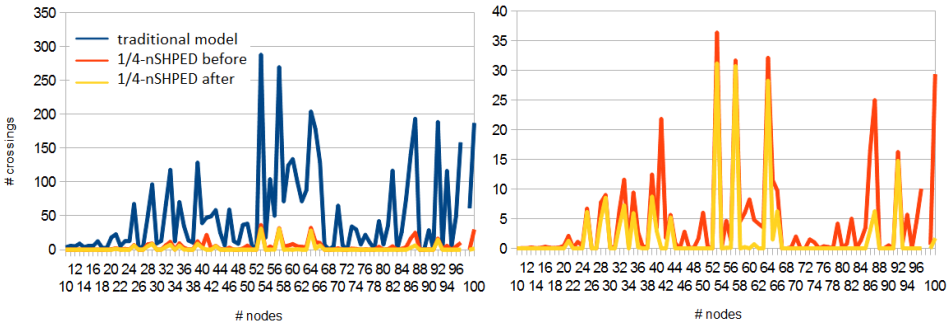


FIGURE 4.24: The number of crossings averaged over all of the North graphs with n vertices. The blue line refers to the traditional model after the spring embedder preprocessing step, the red line refers to the 1/4-nSHPED model before application of the 1/4-SHPED spring embedder, and the yellow line refers to the result after application of our 1/4-SHPED spring embedder.

4.5 1/4-SHPED User Study

In this section we will evaluate the partial edge drawing model. We will aim for objective judgement by measuring the error rate and task completion time for several tasks. The study will follow the cross-over design as a kind of longitudinal survey, i.e., participants receive a sequence of treatments. We also control the study, since every task for a configuration is also given in the base setting, which we use for comparison.

We want to evaluate 1/4-SHPEDs in terms of usability and readability. We will start with an introduction in Section 4.5.1 motivating the design and the setting of the study. We then present the whole design in Section 4.5.2 and present results, evaluation, and their interpretation in Section 4.5.3. After that, we discuss in Section 4.5.4 again the design of the study, in terms of which settings may have distorbed the results. We conclude with pointing out directions for future work.

4.5.1 Introduction

In 2011 Burch et al. [33] investigated the usefulness of partially drawn links for directed graphs. Their study confirms the impression that partially drawn links lead to shorter task completion times. Their main result is that optimal link length is either around 75% or 12.5% dependent on the task. However there was one task where the error rate significantly improved. Since his results were strongly depending on the fact that edges were directed, theory on partial edge drawings for undirected graphs focused on 25% link length. The reason was that given 25% of an edge the estimated distance of the adjacent vertex, which is four times the link length, was more accurate in some examples than choosing

12.5%, leading to eight times the link length. Clearly the link length must be smaller than 50% for undirected graphs.

We develop a user study that aims at a direct comparison of partial edge drawings with the traditional straight-lines in particular for undirected graphs. To do so we take straight-line drawings in a layout that is appropriate to supports the beauty of straight-line drawings according to relevant graph drawing aesthetic criteria, i.e., layouts produced by a force-directed method [87]. This method displays symmetries and aims for a well-distribution of the vertices. When adopting the 1/4-SHPED drawing model to these drawings, the resulting drawing is not a 1/4-SHPED in general, since not all crossings on stubs may disappear. Thus we produce only 1/4-nSHPEDs, as introduced in Section 4.4.2, which is a 1/4-SHPED disregarding the crossings of stubs, but insisting on the stub-edge-ratio of 1/4. Although the drawings are 1/4-nSHPEDs we refer to them as 1/4-SHPEDs by abuse of notation.

The use of our 1/4-SHPED spring embedder for laying out graphs as 1/4-SHPEDs may distort the study, since when choosing for any layout the best distribution of vertices makes the drawings incomparable. The base of the study is the layout that supports not the drawing we claim to be the best according to our hypothesis:

The 1/4-SHPED model is more readable, understandable and interpretable than the traditional straight-line model.

In our user study we want to evaluate whether partial edge drawings significantly support readability in contrast to the traditional straight-line drawing approach. We will measure this by analyzing the error rate and the task completion time for every participant in each trial. The results will be taken to analyze differences in accuracy and completion time with respect to study variables. The user study is based on the bachelor's thesis of Leibßle [131].

4.5.2 Design

In this section we present and defend our choices for the design so that we minimize the parameters that affect a decision in a particular direction. Clearly we want to have a neutral setting and if not avoidable, then the advantage should be at the traditional straight-line model, so that criticism in advantageous setting for PED can be excluded.

The Graph: We have chosen a graph according to the Barabási Albert graph model [13]. This graph model statistically ensures a specific vertex connectivity in large networks that is following a scale-free power-law distribution. This graph model seems to be appropriate when generating graphs with similar statistical properties.

The Model: We will produce drawings in the traditional straight-line drawing model, referred to as *TRA*, and in the 1/4-SHPED model, referred to as *PED*. For both drawing models we picked randomly a graph from the pool of graphs with similar statistical properties. Since our user study will not take much time (up to 10 minutes), there is no time to make the participants forgetting previously seen drawings. Thus we avoid using the same graph in different models.

The Layout: Graphs with equal statistical properties were taken and laid out by an implementation of the force-directed layout algorithm of Fruchterman and Reingold [87]. We produced drawings according to the layouts with a Java based tool supported by the yFiles library [193]. In order to avoid advantageous layouts for PED, we took as base a layout that supports the traditional straight-line layout. We also avoided using the same graph in different layouts (independently of the model) since we suspected that participants may remember or recognize the graph, when seen them before.

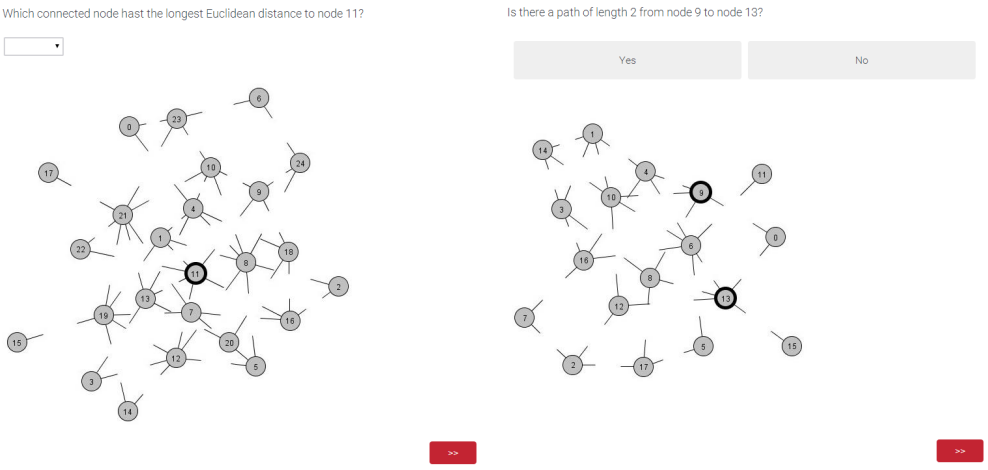
Turning the 1/4-nSHPEDs into 1/4-SHPEDs may affect the aesthetic properties of the PED drawing positive as well as a possibly negative influence on the traditional straight-line drawing. Since we cannot control the grade of aesthetic with respect to crossings, we fix the vertex positions given by the force-directed algorithm.

The Tools: For creating the graph and its layout we used yEd, a yFiles-based tool [193], together with a separate yFiles-based tool for just shortening the edges appropriately, i.e., for turning a straight-line drawing into a 1/4-nSHPED. The evaluation was then created on the base of these drawings using Qualtrics, a commercial research software as online platform [36, 170]. The commercial part of this software basically concerns a full evaluation of the data. Thus we received just the raw data in tables and evaluated the data according to our standards, which is covered by the next section.

The Graph Size: For both models we have chosen graphs of two different sizes, i.e., *small* graphs with $n = 18$ and *large* graphs with $n = 25$ vertices. We have chosen the number of vertices of both, small and large graphs, such that tasks become not trivial and graphs can be drawn within a specific frame even for small screen resolution. The number of edges was then determined by the expected value of the power-law distribution [13] through the density $m/n \in [1.6, 1.9]$. We randomly chose one value for the small graphs, which was 1.66 and led to $m = 30$ edges, and one value for the large graphs, which was 1.88 and led to $m = 47$ edges. In such a way we avoided to distinguish between dense and sparse graphs.

The Tasks: A difficult decision was to chose adequate tasks. Lee et al. [130] suggested a list of relevant tasks for experimental results. We picked two of them of different types for our study. We asked the following two questions:

1. Which adjacent node has the longest Euclidean distance to node v?



(a) A screenshot of task 1.

(b) A screenshot of task 2.

FIGURE 4.25: Screenshots of different tasks with highlighted vertices.

2. Is there a path of length 2 from node v to node w?

According to Lee et al. [130] Task 1 is a representative question for evaluating the "adjacency" of a node, while Task 2 is a representative question for evaluating the "accessibility" of a node. Each question formed a block to reduce the cognitive load. The first question had to be answered by choosing the correct node from a drop-down menu, while the second question had to be answered by pressing a "yes" or "no" button. Figure 4.25 shows screenshots for both tasks.

The Tasks Generation: For each of the two questions we have chosen a node v randomly out of a pool of candidates satisfying three properties:

- v is not in the convex hull of the set of vertices.
- v has no two neighbors with the same longest Euclidean distance.
- the probability of choosing v increases with the degree of v

We have collected the candidates for every layout and randomly picked one vertex as v and one vertex as w, in case of the second block. Thus all picked vertices were "inside" the drawing. They had an average degree of 4.85.

With the three properties we wanted to avoid trivial answers because of boundary cases and ambiguity in the second block. Choosing vertices from the interior provides a comparable statistical difficulty for answering the questions. Another aspect we wanted to avoid is to spend time on searching for the respective vertex, which is given in the question. We supported finding the vertex quickly by marking the vertex fat.

level	description	# participants
1	No experience at all - This is the first time I ever heard of graphs.	1
2	I've heard of the concept of graphs before but have never really seen much of them.	3
3	I have seen a few graphs before and know what they look like.	4
4	I am sometimes confronted with graphs due to my field of study.	16
5	I have seen many graphs before and would consider myself well-informed.	15
6	I deal with graphs almost daily (due to my job and/or hobby).	46

TABLE 4.3: The participants were chategorized into *levels* due to the *description* they assigned. The final amount of participant per level is listed in the last column.

The Participants: In total we had 85 participants. We started our controlled user experiment with some introductory questions in order to categorize the participants. First we wanted to know their knowledge about graphs, which separated the participants into experts and non-experts. We intended this split by inviting members of the graph drawing community via mailing list and students of computer science from the second year of studying to the survey.

The participants were classified into groups according to their knowledge about graphs. Table 4.3 shows which categories we offered (levels 1–6) with description and how many participants were classified to the respective category.

We also asked for their age, in particular to possibly classify the experts once more with respect to their experiences in terms of age.

The Information to the Participants: Every participant was introduced in the partial edge drawing model by the definition, the intuition behind and an example. The example showed a graph in the traditional straight-line model and the same graph laid out in the 1/4-SHPED model. The users had to confirm they understand the model.

Also the whole study was split into two blocks. Each block was about the same question in order to reduce cognitive load from task switching. The two blocks were clearly separated such that the users could adjust themselves on the next question. This followed the continue-on-demand study design.

Each block was introduced with an example consisting of the question of this block and a layout of a graph in the traditional straight-line drawing model. By confirming the correctness of the answer of the question of this particular layout we assumed familiarity with the question for the whole block. This

also prevented the user to focus on reading the question since it didn't change during the block.

The Variables: Our study variables were (1) the model (two options), (2) the graph size (two options) and (3) the tasks (two options). Thus in total we had (2 tasks) $\times$ (2 models) $\times$ (2 graph sizes) $= 8$ configurations. In order to get significant results according to the repeated-measures design, we asked two times for each of the configurations resulting in 16 trials per participant. Within each block of a question we randomly permuted the eight trials, consisting of (2 models) $\times$ (2 graph sizes) $\times$ (2 repeats). With 85 participants we had in total 1,360 trials to evaluate.

In total our evaluation design provides four parameters, three of them stem from variables of the design:

- *level* of experience $\in \{1, \ldots, 6\}$
- *model* of drawing $\in \{\text{PED}, \text{TRA}\}$
- *size* of graphs $\in \{18, 25\}$
- *task* of block $\in \{1, 2\}$

The Study Procedure: For each trial from the design we stored the start-time, the first-click-time, the last-click-time and the given answer. The first-click-time indicates the moment when the participant found her answer, i.e., the participant started to select the answer in the first block, while in the second block the first-click-time is already the confirmation of her answer. The last-click-time indicates when the participant was ready to leave the page of this trial, i.e., we can check here the difference between the first-click-time and the last-click-time to see if the participant was maybe undecided or needed time for selecting his answer.

4.5.3 Results

In this section we will present the results in different formats. We will first present a total overview of the result and after that we focus on interesting parts to be broken down in more details. Along the way, we will interpret reasons for the results.

In order to evaluate the results with a meaningful amount of participants, we have chosen only levels 4 to 6 of the participants for the following evaluation. The reason is that we define meaningful by at least 50% of "possible participants" in a level. For 85 participants we expect 14.16 possible participants in each level, such that 50% of them gives the threshold of 7.08 participants, which is fulfilled only for levels 4 to 6.

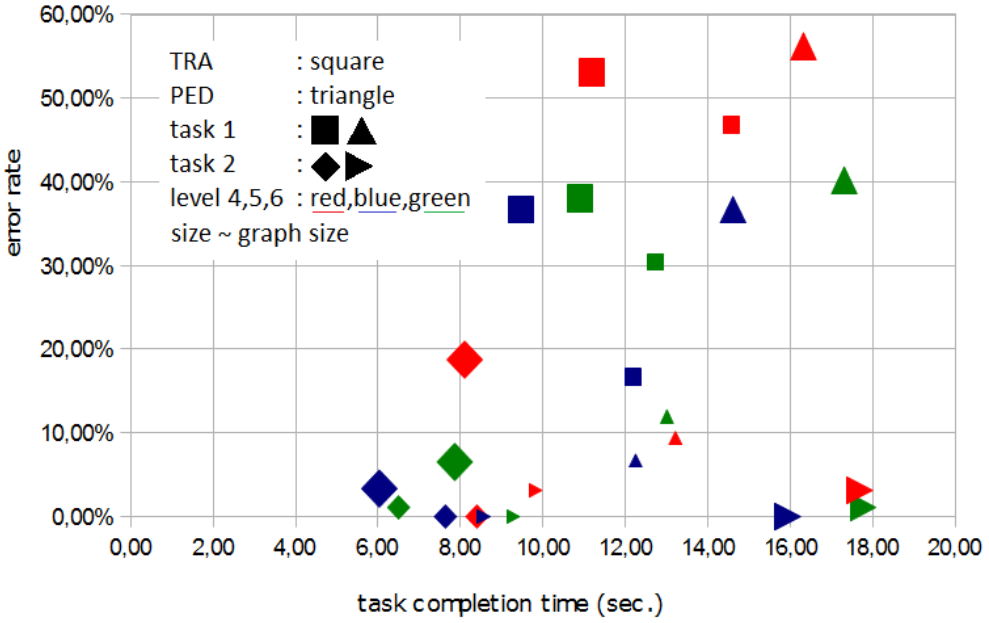


FIGURE 4.26: A scatterplot of error rates vs. task completion time of the averaged results. Triangles represent PED, squares represent TRA, horizontal alignment represents task 1, vertical alignment represents task 2, size of shapes represent size of graphs and colors red, blue, and green, respectively, represent level 4, 5, and 6, respectively.

We first averaged the error rates and the task completion times over the two repeats and the number of participant in each level to get an overview. The scatter plot in Figure 4.26 illustrates the results of the study with respect to the parameters (model, task, size, level) in different attributes of graphical visualization (shape, orientation, size, color). The scatter plot shows clusters with respect to some parameters.

Results of Task 1 are mainly above the 30% error rate (significant exception is PED for small graphs), while results of Task 2 are mainly below the 10% error rate. Also most of the answers concerning Task 1 are located with more than 10 seconds task completion time, while most of the answers concerning Task 2 are located below 10 seconds for task completion time (significant esception is PED for large graphs). These results confirm our impression that Task 1 was more difficult to read from the graphs in general.

Results for TRA are located mostly to the left of the corresponding results for PED. So answers for the traditional model seem to be given faster compared to results for the PED model.

When considering only Task 1 and fixing the error rate, then for large graphs the answers for TRA were given faster compared to PED. When considering only Task 1 and fixing the task completion time, then for small graphs the

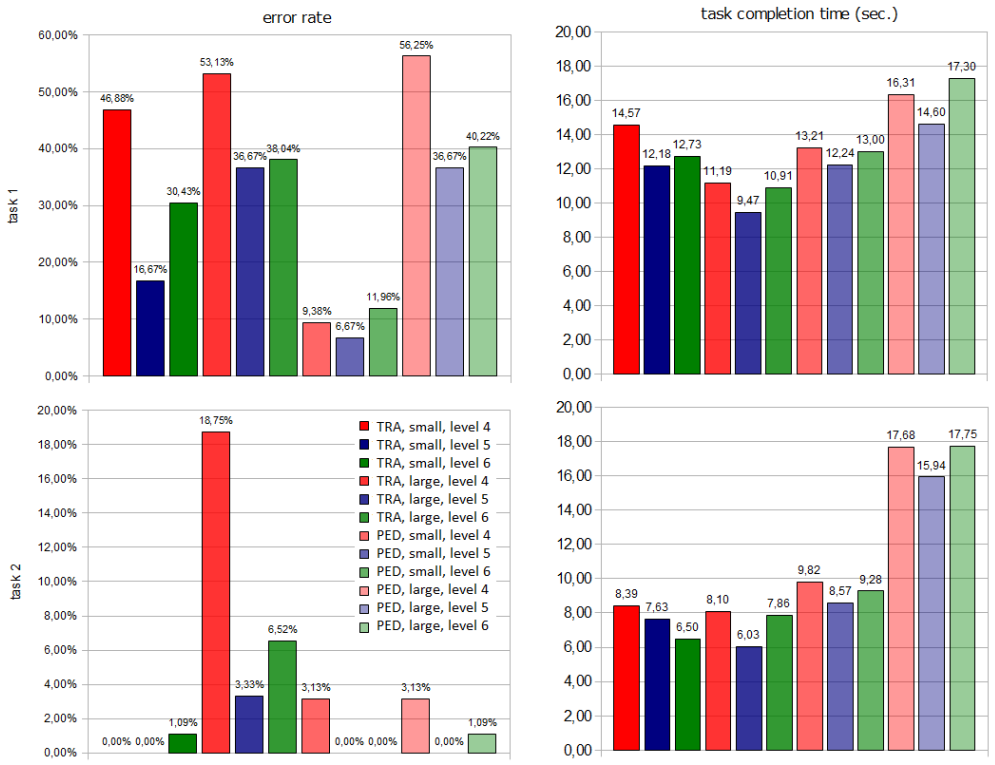


FIGURE 4.27: The left graphics illustrate error rate, the right graphics the task completion time. The top row represents task 1, while the bottom row represents task 2. The results are separated through model, graph size and level.

answers for PED were more accurate compared to TRA. It seems that for the "adjacency check" that large graphs should be drawn in the traditional model, mostly because of familiarity with this model, while small graphs should be drawn in the PED model. This result surprised us since we thought that PED supports Task 1 in general, because of simply comparing the rays of a star.

When considering only Task 2 and fixing the error rate, then answers for TRA were given faster compared to PED, even in half the time of PED for large graphs. On the base of the same error rate TRA seems to be more familiar for "accessibility checks", in particular for following paths. When considering only Task 2 and fixing the task completion time, then for TRA the answers for small graphs were given more accurate compared to large graphs. This sounds very natural for us, since the general assumption is that large graphs require a longer and more intensive perception.

Results for sizes of graphs are difficult to generalize. Also results distinguishing between the levels are not obvious in this scatter plot. For almost every configuration there was no significant difference between the levels. Thus in the next graphics we place the three levels as block close together.

We turn to bar charts separated by task, illustrating the error rates and task completion time separately, see Figure 4.27. We will compare both the error rate and the task completion time, with respect to level of knowledge, graph size and model of the drawing. We observe that differences between levels are mostly not significant, which means the knowledge and expertise on graphs is not crucial.

For PEDs the error rate is very low in general, except for large graphs in Task 1. On the other hand for TRA the error rate is not very low in general, except of small graphs in Task 2. For PED the task completion time is comparable with TRA, except of the large graphs in Task 2. Thus from the bar charts we extract the following two interpretations.

- PED supports small graphs for searching the farthest neighbor.
- PED does not support large graphs for checking paths of length two.

In Task 1 the task completion time seems to be in a reasonable fluctuation for PED as well as for TRA. We interprete the difference of task completion times for large graphs between TRA and PED with approximately 5 seconds as not highly significant. A significant difference in task completion time can be observed for Task 2. Large graphs in PED required much more time. But in general, the error rate for PED is very low or in worst case similar to the error rate for TRA in the same configuration. At this point we have the impression that in general PED leads to lower error rate. On the other hand this might be a consequence of spending more time on the PED model, at least in Task 2 results of the large graphs provide evidence of this reason.

Some results differ significantly in terms of task completion time. The reason might also be located in the design or in the fact that it was an online evaluation. When participant do the evaluation we can not exclude disturbing factors from environment. Also we had no control of the screen resolution, e.g., leading to scrolling, or other handling problems. Thus the task completion time of the participants may differ from person to person even in the same configuration.

The next step of deeper evaluation is to consider discrete task completion times, see Figure 4.28. This graphic confirms the interpretations from the previous results in general. There is one additional interesting aspect to be pointed out. In Task 2 the level–4 candidates were more accurate with PED for large graphs compared to TRA for large graphs. This indicates that experts are used to the traditional model, while non-experts may prefer the PED model. Regarding this aspect further studies must follow with non-experts in order to reduce prejudice.

Another aspect we ignored is the age of participants. In Figure 4.28 we see no significant difference between different levels in Task 1, which we claim is

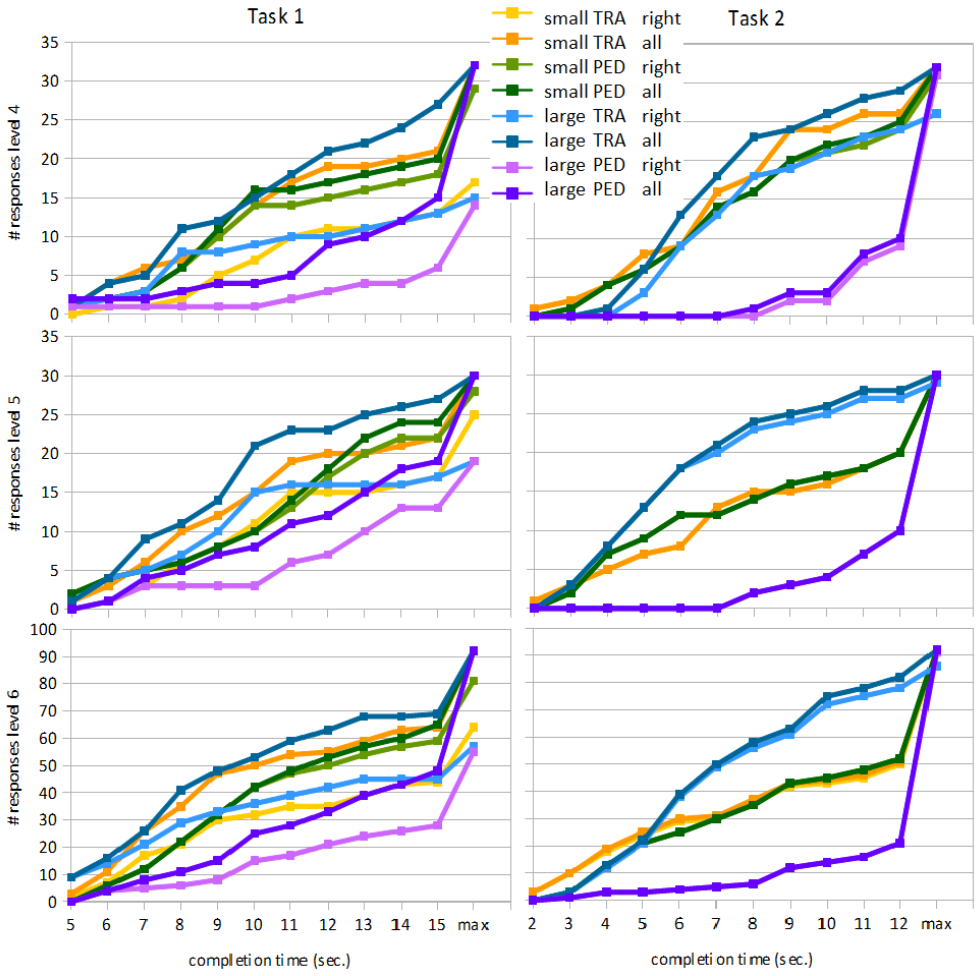


FIGURE 4.28: This graphic shows the absolute number of responses. The left column represents Task 1, the right column represents Task 2. The first, second, third row, respectively result illustrate participants with experience levels 4,5,6, respectively. In each graphic part we see the number of correct results, as well as the total number of results acummulated up to every time stamp and for every configuration regarding size and model.

also true for the age. Indeed Figure 4.29 confirms this claim. The age is well distributed and no accumulation point can be identified.

We next combine results of participants with comparable task completion time in order to extract more information about error rate. Due to the observation that differences between levels are not significant, we combine the results from different levels within the same settings.

Figure 4.30 shows the relative time, which participants needed for a configuration in PED compared to TRA. This graphic was expected to show us whether there are significant differences among reaction times of participants. However

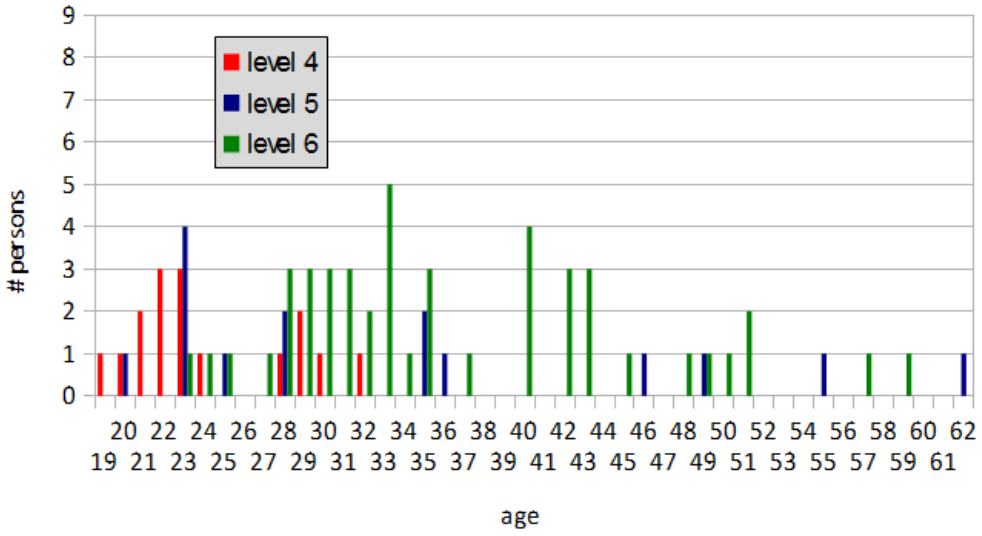


FIGURE 4.29: The age of participants separated due to level of knowledge.

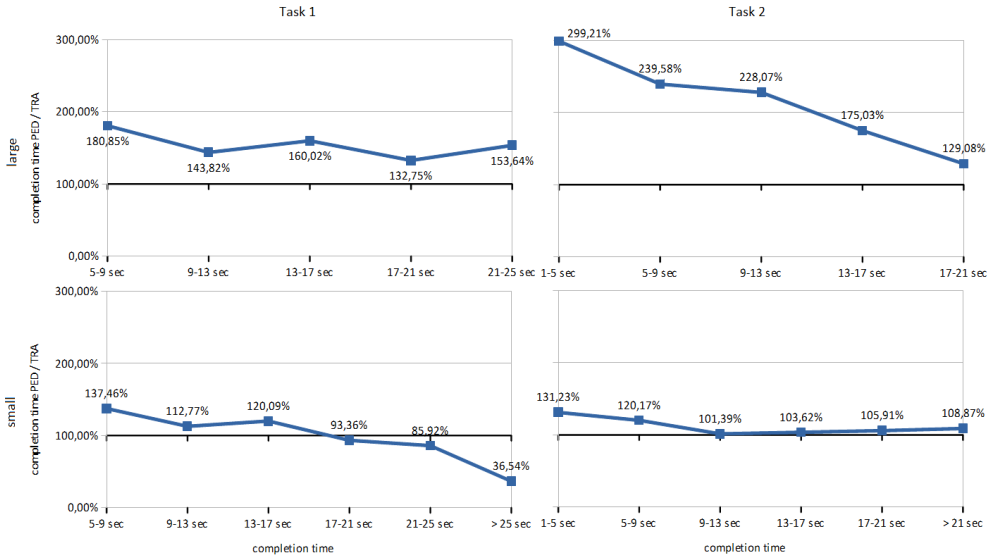


FIGURE 4.30: Illustration of relative task completion time of PED compared to TRA (100%). The left column represents Task 1, the right column represents Task 2. The first row represents large graphs, while the second row small graphs.

we still can only see that TRA is better for large graphs in general, while PED seems to be better for small graphs - at least for Task 1.

Summarizing we can say that Task 2 (accessibility check), for which we assumed to support the traditional straight-line model, was answered in the PED model more accurately, but not significantly faster. Task 1 (adjacency check), for which we assumed the PED model to be superior, was answered more accurate

only for small graphs in PED model and only slightly faster in the traditional model. Thus PED seem to support questions of accessibility (if there is a path connecting them) in general and distance comparisons for small graphs. For large graphs in terms of distance comparisons, we cannot recommend a preferable model. Since for some configurations PED required higher task completion time, there is no significant result that supports PED undisputedly. The same holds for the traditional model since we picked a layout advantageous for the traditional model. At least a general tendency to PED can be identified.

4.5.4 Discussion

In the design of this evaluation we tried to minimize the influences on the participants. However this was not always possible in a study of this small size. We list here several aspects that may have influenced results due to the design. We also give some hints for settings of further evaluations in future work.

- Due to the small study size we could not make the participants forget previously seen graphs. Thus we took graphs with comparable statistical properties; we could not avoid accidentally picking more difficult graphs.
- For the same reason we draw graphs as 1/4-nSHPED laid out with a traditional force-directed algorithm for straight-lines, not with a 1/4-SHPED supporting force-directed algorithm.
- We couldn't control participants with respect to distractions in the environment.
- We mainly asked for participant with some experience on graphs, which means that many of them were possibly used to the traditional straight-line model and thus possibly prejudiced.
- Due to screen resolution we could not control, whether the participants had to scroll or search for buttons. Nevertheless we picked a small resolution to minimize the probability of scrolling.
- We were not totally convinced whether participants understood the model. A longer study would offer the opportunity to check the participants more accurately for their understanding of the model.
- Some participants seem to have used browsers that prevented submitting time-stamps. Due to this fact we had to sort out some results.
- We did not implement a deeper check whether the experience of the participant really correspond to the level they checked. So participants could have over- or underestimated their experience.

In the beginning we motivated why we used a traditional spring embedder for the layout of graphs. For future work we may turn the page and produce drawings with our 1/4-SHPED spring embedder plus creating drawings in the traditional straight-line layout directly from the 1/4-SHPED spring embedder layout. Another approach is to use the 1/4-SHPED spring embedder for PED layout and a spring embedder for the traditional straight-line layout by using the same graph in both layouts. This may require a relabeling of the graph or a short "break" between serving the same graph. We also like to investigate in a longer study what happens to the results of Task 1 vs. Task 2 if we use the same configurations in terms of model and size for both tasks.

In future work we like to evaluate another stub-edge ratio. Also the graph sizes were chosen arbitrarily. Thus other graph sizes should be evaluated as well. We selected a specific density for graphs. Since 1/4-SHPEDs support highly connected graphs, like the graphs from Section 4.2, we would like to evaluate also denser graphs.

Unfortunately the average of task completion times differed a lot between all participants. In order to control this fact better, another study should provide every figure just for a fixed number of seconds, e.g., 15 seconds for a configuration in Task 1 and 10 seconds for a configuration in Task 2.

4.6 Summary and Future Work

Avoiding edge crossings by breaking edges of drawing only the stubs incident to corresponding vertices is a pragmatic approach which seems to make not much sense on the first sight. After some work on it, we found the topic appealing from the intellectual and fun point of view but also from the practical side. Restricted versions of the approach might be very valid and effective for some applications.

We provide the first formal step for a more structural way of research on PED topics. We found several classes of graphs admitting 1/4-SHPED and proved also that some graphs do not admit a 1/4-SHPED. For general graphs we provided a force-directed embedder and evaluated the new drawing model by a user study.

It would be especially great to find further classes of graphs admitting a δ-SHPED. Here the Condition 4.4 might be of some help. Candidates are l-planar graphs among others. We would like to further reduce the bound on n for which the complete graph K_n admits no 1/4-SHPED. Finally it would be nice to improve the 1/4-SHPED spring embedder to be applicable for larger graphs and to evaluate practical aspects of 1/4-SHPEDs more extensively.

Chapter 5

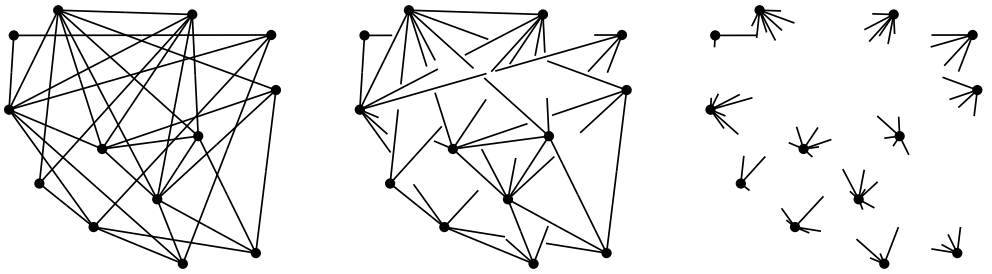
PEDs for Graphs with Fixed Vertex Positions

In this chapter we focus on the key question, which is the optimal length of the partially drawn edges, provided that the geometric embedding is given. Geometric embedding means, that the position of the vertices is fixed and the edges are drawn as straight lines [30, 68]. Since our focus is on straight-line drawings, embeddings will always mean geometric embeddings in this chapter. A graph together with a geometric embedding is called *geometric graph*. In this chapter we will discuss geometric graphs according to the type of PED they admit.

It will turn out that considering only SHPEDs is not very interesting, so we focus more on non-homogeneous and non-symmetric drawings and optimize the length of the stubs. PEDs where for every edge the sum of both stubs is approximately the same as the edge length are called *ncPEDs*. In drawings not admitting a ncPED we aim at maximizing the sum of the stub lengths and call those drawings *maxPED*, respectively *maxSPED* if symmetry is preserved. We will consider maxSPEDs for the sake of finding techniques to tackle maxPEDs.

Figure 5.1(b) depicts a maxSPED of the straight-line drawing in Figure 5.1(a). We have slightly shrunken the stubs in the maxSPED so that they do not touch. For comparison, Figure 5.1(c) depicts a SHPED with maximum ratio δ.

We start with an introduction including the definition of ncPED, maxSPED and maxPED, respectively, in Section 5.1. Here we also provide some basic results, which are mentioned just for completeness but involve no complex calculations. In Section 5.2 we continue with ncPEDs and provide a sufficient and necessary condition for geometric graphs admitting ncPEDs. We continue with maxSPEDs in Section 5.3, where we show how to compute maxSPEDs efficiently for some specific geometric graphs, while we show NP-hardness in



(a) with crossings (b) as a maxSPED (c) as a maxSHPED

FIGURE 5.1: Various drawings of a 13-vertex graph.

general and provide a 2-approximation for minSPEDs, the dual of maxSPEDs. Finally we provide an ILP for maxPED and point out differences between maxSPEDs and maxPEDs geometrically, which are the reason that maxPED is much more difficult to access for us. In Section 5.4 we summarize the main results on PEDs for geometric graphs. Most of the results of this chapter are published in [200, 201, 204].

5.1 Definitions and Basic Results

Let $G = (V, E)$ be an arbitrary graph with fixed embedding Γ, which in particular means that the vertex positions are fixed, say by a preprocessing layout step, where in a traditional straight-line representation some of the edges intersect each other. Instead of considering the graph G, we always consider the geometric embedding Γ, or *embedding* for short. In particular we focus on the geometric embedding, whenever a geometric graph is given. For an embedding Γ, we introduce three types of PEDs. Let $w : E \longrightarrow \mathbb{R}$ be a mapping, which sums up the length of edges, so $w(E) = \sum_{e \in E} d(e)$. For a given graph $G = (V, E)$ with embedding Γ, the *weight* $w(\Gamma)$ is defined by $w(E)$. We denote by s_v^e, s_w^e the stub of the edge $e = (v, w) \in E$ incident to v, respectively incident to w, as well as their length. Weight $w(PED_\Gamma) = \sum_{e=(v,w)\in E}(s_v^e + s_w^e)$ is computed for a PED_Γ. We say a PED_Γ is a *nearly complete PED* (ncPED), if for every $\epsilon > 0 : w(\Gamma) - w(PED_\Gamma) < \epsilon$. Additionally we demand on ncPEDs to have precisely as many gaps as intersection points in the underlying traditional straight-line embedding. A $(S)PED_\Gamma$ is *maximal* denoted by max(S)PED, if there is no other $(S)PED'_\Gamma$ with $w((S)PED'_\Gamma) > w((S)PED_\Gamma)$. We will again identify vertices with points and edges with segments.

At this point we recall that stubs are considered as half-open sets. Otherwise the corresponding terms are not well-defined. An ncPED requires just at most one cut for every edge e of the underlying embedding. So edge $e = (v, w)$ is split into two stubs s_v^e, s_w^e and the equality $s_v^e + s_w^e = d(e)$ holds, if there is just the intersection point missing. For illustrations the gap is drawn significantly

larger than in theory. This is emphasized by defining an ncPED through an infinitesimal ϵ. Notice here that for length of edges we use the distance function d, while for the length of stubs we omit this function.

We may classify the embeddings in the traditional straight-line model by the maximal number of crossings on each edge. We call a geometric embedding k-planar, if every edge is crossed at most k times [147]. A geometric graph is called k-planar, if id admits a k-planar geometric embedding. Whenever we refer to a k-planar geometric graph, we implicitly assume that the corresponding embedding is k-planar as well. In the following we first state some trivial results to get an impression of the PED variants.

Lemma 5.1. *Every embedding of a given graph can be modified to an SHPED.*

Proof. We easily erase all intersection points and all segments cut off by this operation. By shorten the remaining edges sufficiently, we get an SHPED. □

We give an efficient algorithm to test whether a given graph G with embedding Γ and a prescribed stub-edge-ratio δ has an SHPED. Let P be the set of intersection points of Γ and each $q \in P$ separates its incident edges e_q^1, e_q^2 into $e_q^{11}, e_q^{12}, e_q^{21}, e_q^{22}$. For stub-edge-ratio $\delta \in [0, 0.5]$, the algorithm is simply as follows:

For each intersection point $q \in P$ **do**
If $\frac{e_q^{11}}{e_q^{12}} \notin \left(\frac{1}{\left(\frac{1}{\delta}\right)-1}, \left(\frac{1}{\delta}\right) - 1 \right)$ and $\frac{e_q^{21}}{e_q^{22}} \notin \left(\frac{1}{\left(\frac{1}{\delta}\right)-1}, \left(\frac{1}{\delta}\right) - 1 \right)$,
then return 'no SHPED with stub-edge-ratio δ'.

This algorithm tests whether an intersection point q is located on a broken part of the edges. Since $|P| = O(n^2)$, the algorithm requires $O(n^2)$ time.

5.2 Nearly Complete PEDs

In this section we are interested in whether k-planar geometric graphs admit an ncPED. Recall, we focus on the k-planar geometric embedding, whenever a geometric k-planar graph is given. Our objective is to modify drawings at crossings aiming for ncPEDs. Whenever we remove one ϵ-part before and one ϵ-part after the crossing, then this removal counts as one gap for this particular edge.

Lemma 5.2. *Every* 1*-planar geometric graph admits an ncPED.*

Proof. Let $\epsilon > 0$. In every 1-planar embedding Γ every edge has at most one intersection point. So we can avoid an intersection point p in a PED, by removing 2ϵ-parts on one of the two edges - one before p and one after p. Proceeding this for every intersection point we obtain a PED Γ'. Its weight is $w(\Gamma') = \sum_{e \in E} d(e) - 2\epsilon|P|$, where P is the set of intersection points in Γ. The difference between $w(\Gamma) = \sum_{e \in E} d(e)$ and $w(\Gamma')$ can be made arbitrarily small by selecting a small ϵ. In particular, there are precisely $|P|$ gaps in Γ'. □

Lemma 5.3. *Every 2-planar geometric graph admits an ncPED.*

Proof. Let Γ be a 2-planar embedding. To guarantee an ncPED, every edge is cut at one intersection point at most, which we call *responsibility* of an edge e for an intersection point q. In this case a length-2ϵ piece of e is removed – one before q and one after q. To find right responsibility for edges, we compute a bipartite intersection graph $S = (P \cup E, E')$, where P is the set of intersection points in Γ and $(q, e) \in E'$, if and only if q is incident to e. Each intersection point has exactly one responsible edge, which is indicated by a maximal matching in S. In Γ each intersection point comes from exactly two edges and each edge has at most two intersection points. We prove by contradiction using Halls's Marriage Theorem that a maximal matching exists and covers P.

Assume for contradiction that in the bipartite intersection graph $S = (P \cup E, E')$ there is a subset $P' \subseteq P$ with $|N(P')| < |P'|$, where $N(P')$ denotes the neighbors of P'. Since for every crossing exactly two edges are involved, we have exactly $2|P'|$ outgoing edges for P'. Thus $N(T)$ has at least $2|P'| > 2|N(P')|$ outgoing edges and by the pigeonhole principle there is an edge $e \in N(P')$ which is involved in at least three crossings, contradicting the 2-planarity of Γ. □

For graphs with 3-planar embedding it is not possible to compute an ncPED in general, see Figure 5.2. If we consider the bipartite intersection graph of G, there are two intersection points p_9, p_6, which are not covered by the matching and thus have no responsible edge. Given the responsibilities, we can draw ncPEDs as in Lemma 5.3.

Lemma 5.4. *Let $G = (V, E)$ be an undirected graph with embedding Γ and intersection points P and $S = (E \cup P, E')$ its bipartite intersection graph. G has an ncPED concerning Γ, if and only if S has a maximal matching covering P.*

Proof. The proof of this lemma follows directly from the assignment of crossings to edges. □

Notice, according to the definition of ncPED, i.e., that in every ncPED the crossings are resolved by one local broken and one local entirely drawn edge,

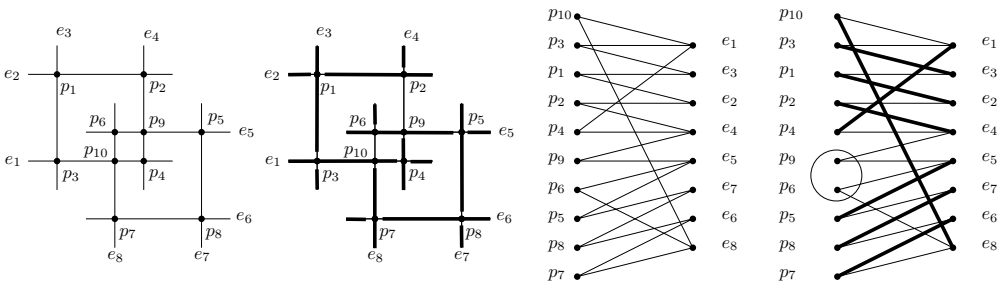


FIGURE 5.2: Example of embedded graph G with no ncPED, because any matching in its intersection graph does not cover P.

Lemma 5.4 provides a sufficient and necessary condition on the existence of ncPEDs. We can not ensure ncPEDs for embeddings of graphs with edges that are crossed more than twice. Therefore we turn to PEDs of maximal weight in the next section.

5.3 Maximal SPEDs

The geometric graphs can be partitioned into those admitting an ncPED and the remaining graphs, which admit (at least one) maxPED. Clearly there is no graph admitting no maxPED, since an ncPED is in particular a maxPED and from all finitely many PED candidates, that have cuts only at intersection points, there will be some of maximum weight. So we are interested in investigating maxPEDs for the sake of characterizing all geometric graphs, whether they admit a ncPED or just a maxPED.

We start all considerations on maxSPEDs instead of maxPEDs to get a feeling for the complexity of this problem. The symmetry makes the problem easier accessible for us. Our hope is to extend techniques applicable for maxSPED also to be applicable for maxPED with slight modifications. We finish this section with a short discussion on maxPED.

In the following we provide an ILP for computing maxSPEDs in general and a dynamic program to efficiently compute a maxSPED for the special case of geometric graphs that admit 2-planar embeddings. Then we prove the NP-hardness for computing a maxSPED in general. Finally we turn to the dual problem minSPED of minimizing the ink that has to be erased in order to turn a given embedding into a SPED and provide a 2-approximation. We conclude with an ILP formulation for maxPED.

Lemma 5.5. *A maxSPED can be computed by an ILP for every geometric graph.*

Proof. Let $e \in E$ be an edge and s^e be the length of one single stub of e. We maximize the total length of the stubs while avoiding crossings in PED.

Let P be the set of real crossing points, which exclude the vertices, and let $M = \max_{e \in E} d(e)$ be a constant. We may compute in advance for every intersection point $p \in P$ the length of the smallest segment for both of the involved edges $e = (v, w)$ and $e' = (a, b)$, i.e., $l_p^e = \min\{d(v, p), d(w, p)\}$ and $l_p^{e'} = \min\{d(a, p), d(b, p)\}$. We use again a binary variable $x_{ee'}$ to express an "or" [4]. So, the ILP is as follows.

$$\begin{aligned} \max \quad & \sum_{e \in E} s^e \\ s.t. \quad & 0 < s^e \leq l_p^e + x_{ee'} M \\ & 0 < s^{e'} \leq l_p^{e'} + (1 - x_{ee'}) M \\ & x_{ee'} \in \{0, 1\} \quad \forall\, p \in P \end{aligned}$$

□

Note that for a 1-planar geometric graph maxSPEDs can be obtained greedily, since its intersection graph consist of components of size two. Moreover we have $O(n)$ intersection points.

Theorem 5.6. *Let G be a 1-planar geometric graph with n vertices and with 1-planar embedding Γ. A maxSPED of Γ can be computed in $O(n)$ time.*

For a 2-planar geometric graph the components of the intersection graph consists of paths and cycles. For that case dynamic programming for maxSPED can be applied successfully, as we will see next.

Let G be a 2-planar geometric graph and let Γ be the corresponding 2-planar embedding. Given G and Γ, we define a simple undirected graph C as follows. C has a vertex v_e for each edge e of G. Two vertices v_e and $v_{e'}$ of C are connected by an edge if and only if e and e' form a crossing in Γ. Such a graph is in general non-connected. Furthermore, since the maximum degree of C is 2, a connected component of C is either a path (possibly formed by only one edge) or a cycle.

Let C_i be a connected component of C. We define a total ordering of the vertices of C_i. Namely, if C_i is a path such an ordering is directly defined by the order of its vertices along the path (rooted at an arbitrary end vertex). If C_i is a cycle, we simply delete an arbitrary edge of the cycle, obtaining again a path and the related order. That means, if we consider the subdrawing Γ_i of Γ induced by the vertices of C_i (edges of G_i), such a drawing is formed by an ordered sequence of edges (according to the ordering of the vertices of C_i), $e_1, \ldots, e_{n_i}$, such that e_j crosses $e_{(j+1) \bmod n_i}$ for $j = 1, \ldots, n_i - 1$ in case of a path, and $j = 1, \ldots, n_i$ in case of a cycle.

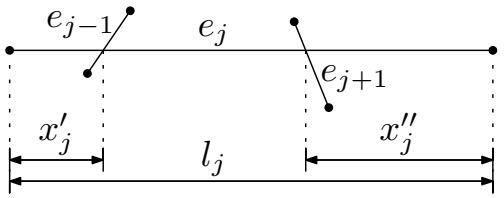


FIGURE 5.3: Notation used in the DP.

We will use the following notation: l_j is the total length of the edge e_j; x'_j is the length of the shortest stub of e_j defined by the crossing between e_{j-1} and e_j, called the *backward stub*; x''_j is the length of the shortest stub of e_j defined by the crossing between e_j and e_{j+1}, called the *forward stub*. See also Figure 5.3.

Consider now the subdrawing Γ_i, and assume that $e_1, \ldots, e_{n_i}$ form a path in C_i. If $n_i = 2$, the maximum total length of the stubs is $k_{\text{opt}} = \max\{l_1 + 2x'_2, l_2 + 2x''_1\}$.

In the general case, we can process the path edge by edge, having at most three choices for each edge: (*i*) we can draw it entirely, (*ii*) we can draw only its backward stubs, or (*iii*) we can draw only its forward stubs. The number of choices we have at any step is influenced only by the previous step, while the best choice is determined only by the rest of the path. Following this approach, let γ_i be a maxSPED for Γ_i and consider the choice done for the first edge e_1 of the path.

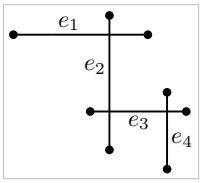
The total length of the stubs in γ_i, minus the length of the stubs assigned to e_1, represents an optimal solution for $\Gamma_i \setminus e_1$, under the initial condition defined by the first step, otherwise, γ_i could be improved, a contradiction. In other words, the optimality principle holds for our problem. Thus, we can exploit the following dynamic programming (DP) formulation, where $O_{\text{in}}(e_j)$ describes the maximum total length of the stubs of $e_j, \ldots, e_{n_i}$ under the choice (*i*) for e_j, $O'_{\text{out}}(e_j)$ describes the choice (*ii*) and $O''_{\text{out}}(e_j)$ describes the choice (*iii*).

$$O_{\text{in}}(e_j) = \begin{cases} l_j + \max\{O'_{\text{out}}(e_{j+1}), O''_{\text{out}}(e_{j+1})\} & \text{if } x'_{j+1} \geq x''_{j+1}, \\ l_j + O'_{\text{out}}(e_{j+1}) & \text{if } x'_{j+1} < x''_{j+1}. \end{cases} \tag{5.1a}$$

$$O'_{\text{out}}(e_j) = \begin{cases} 2x'_j + \max\{O'_{\text{out}}(e_{j+1}), O''_{\text{out}}(e_{j+1})\} & \text{if } x'_j > x''_j \text{ and } x'_{j+1} \geq x''_{j+1}, \\ 2x'_j + O'_{\text{out}}(e_{j+1}) & \text{if } x'_j > x''_j \text{ and } x'_{j+1} < x''_{j+1}, \\ 2x'_j + \max\{O_{\text{in}}(e_{j+1}), O'_{\text{out}}(e_{j+1}), O''_{\text{out}}(e_{j+1})\} & \text{if } x'_j \leq x''_j. \end{cases} \tag{5.1b}$$

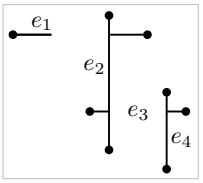
$$O''_{\text{out}}(e_j) = 2x''_j + \max\{O_{\text{in}}(e_{j+1}), O'_{\text{out}}(e_{j+1}), O''_{\text{out}}(e_{j+1})\} \tag{5.1c}$$

In case of a path, we store in a table the values of $O_{\text{in}}(e_j)$, $O'_{\text{out}}(e_j)$ and $O''_{\text{out}}(e_j)$, for $j = 1, \ldots, n_i$, through a bottom-up traversal of the path (from e_{n_i} to e_1). Since e_1 and e_{n_i} do not cross, we have $x'_1 = l_1/2$ and $x''_{n_i} = l_{n_i}/2$. Then, the



	l	x'	x''
e_1	7	3.5	2
e_2	7	1	2
e_3	5	1	1
e_4	4	1	2

(a) with crossings



	O_{in}	O'_{out}	O''_{out}
e_4	4	2	4
e_3	7	6	6
e_2	13	9	11
e_1	16	16	17

(b) as a maxSPED

FIGURE 5.4: (a) A 2-planar drawing Γ and (b) a maxSPED of Γ computed by the DP algorithm.

maximal value of ink is given by $k_{\text{opt}} = \max\{O_{\text{in}}(e_1), O'_{\text{out}}(e_1), O''_{\text{out}}(e_1)\}$. See Figure 5.4 for an example.

In case of a cycle, we have that e_1 and e_{n_i} cross each other, thus, in order to compute the table of values we must assume an initial condition for e_{n_i}. Namely, we perform the bottom-up visit from e_{n_i} to e_1 three times. The first time we consider as initial condition that e_{n_i} is entirely drawn (choice $O_{\text{in}}(e_{n_i})$), the second time we consider only the backward stubs drawn (choice $O'_{\text{out}}(e_{n_i})$), and the third time we consider only the forward stubs drawn (choice $O''_{\text{out}}(e_{n_i})$). Every initial condition will lead to a table where, in general, we do not have all the three possible choices for e_1 (i.e., some choices are forbidden due to the initial condition). Performing the algorithm for every possible initial condition and choosing the best value yields the optimal solution k_{opt}.

The algorithm described above leads to the following result.

Theorem 5.7. *Let G be a 2-planar geometric graph with n vertices and with 2-planar embedding Γ. A maxSPED of Γ can be computed in $O(n \log n)$ time.*

Proof. Consider the algorithm described above, based on the DP formulation defined by the set of equations (5.1). We already showed how this algorithm computes a maxSPED of Γ. The construction of the graph C requires time $O(m \log m)$ with a standard sweep-line algorithm for computing the $O(m)$ line-segment intersections [17]. Once C has been constructed, ordering its vertices requires $O(n_C)$ time, where $n_C \in O(m)$ is the number of vertices of C. Performing a bottom-up visit and up to three top-down visits of every path or cycle takes $O(m)$ time. Thus, the overall time complexity is $O(n \log n)$, since for 2-planar graphs $m \in O(n)$ [147]. □

We finally observe that the restricted 0/1-maxSPED problem for 2-planar drawings, where each edge is either drawn or erased completely, may be solved through a different approach. Indeed, we can exploit a maximum-weight SAT formulation in the CNF+($\leq$2) model, where each variable can appear at most twice and only with positive values [154]. Roughly speaking, we map each edge to a variable, with the weight of the variable equal to the length of its edge, and define a clause for each crossing. Applying an algorithm of Porschen

and Speckenmeyer [154] for CNF+($\leq$2) solves 0/1-maxSPED in $O(n^3)$ time. However, our algorithm solves a more general problem in less time.

5.3.1 NP-hardness

Next we consider the complexity of computing maxSPED in general. We will give a reduction from the NP-hard positive planar 1-in-3SAT [143] to maxSPED according to [119]:

Let φ be a boolean formula in 3-CNF. In the positive planar 1-in-3SAT there is a graph $G(\varphi)$ associated with φ in such a way that there is a *variable vertex* v_x for every variable x and a *clause vertex* v_c for every clause c and edges between v_x and v_c, if and only if x or $\overline{x}$ appears in c. The formula φ is called *planar*, if $G(\varphi)$ is planar. *Positive planar 1-in-3SAT* is the problem of deciding whether a planar φ is satisfiable so that in each clause is precisely one variable satisfied and all variables appear not negated.

An instance of positive planar 1-in-3SAT consists of a formula φ and a planar embedding Γ of $G(\varphi)$. Let c be a clause in φ and let x be a variable in c. We will associate with x a segment s_x, which will be entirely drawn, if x is false, and it is partially drawn, if x is true. For the variables $x_1, \ldots, x_n$ of φ we draw the segments $s_{x_1}, \ldots, s_{x_n}$ replacing the variable vertices $v_{x_1}, \ldots, v_{x_n}$ in Γ. For every clause vertex v_c in Γ we draw three segments s'_x, s'_y, s'_z intersecting in v_c, if x, y, z appear in c. For every edge (v_c, v_x) we draw a sequence of segments, referred to *chain of segments* in the following. The segment $s'_i, i \in \{x, y, z\}$ is connected with s_i by a chain of segments $s_i = s_i^1, \ldots, s_i^k = s'_i$, where k is an even integer. More precisely every segment $s_i^j, 1 \leq j \leq k$ has length 4 and consists of four subsegments $s_i^{j,1}, s_i^{j,2}, s_i^{j,3}, s_i^{j,4}$, each of length 1. Every crossing on segment $s_i^j, 1 \leq j \leq k$ lies between $s_i^{j,1}$ and $s_i^{j,2}$, or between $s_i^{j,3}$ and $s_i^{j,4}$. So in a chain of segments, segment s_i^{j-1} crosses s_i^j between $s_i^{j,1}$ and $s_i^{j,2}$ for $1 < j \leq k$, while segment s_i^{j+1} crosses s_i^j between $s_i^{j,3}$ and $s_i^{j,4}$ for $1 \leq j < k$. Thus the length of the entirely drawn s_i^j is 4, while the length of the partially drawn s_i^j is 2, when $s_i^{j,2}$ and $s_i^{j,3}$ are removed. In every chain from a segment of a variable vertex to a segment of a clause vertex $s_i^1, \ldots, s_i^k$, the segments alternate between entirely drawn and partially drawn. The total length of $s_i^1, \ldots, s_i^k$ is $k/2 * 2 + k/2 * 4 = 3k$. Figure 5.5 shows an example of this transformation. For this reduction k is chosen so that there are sufficiently enough segments to connect three segments of variable vertices in the clause vertex by intersection. All chains must have the same even length k so that the existence of a maxSPED is independent from the value of maxSPED computed by the length of chains. The above construction can be done in $O(m)$ time.

Theorem 5.8. *Computing a maxSPED is NP-hard.*

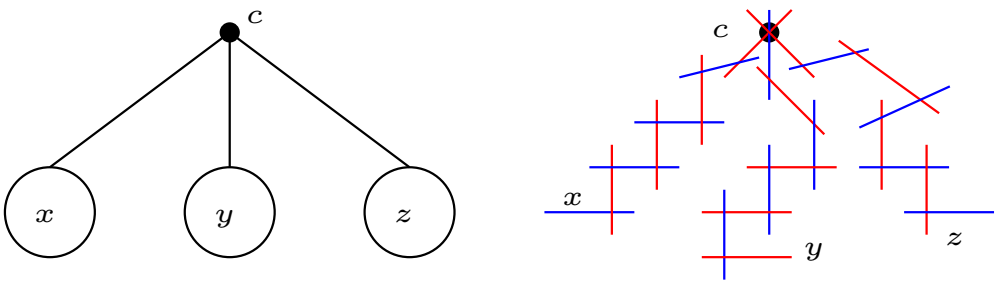


FIGURE 5.5: A clause c on the left with positive variables x, y, z and valid truth assignment with blue $\hat{=}$ false and red $\hat{=}$ true. The transformed clause contains 3 chains of segments, where red edges are partially drawn, while the blue edges are entirely drawn.

Proof. Let φ, Γ be a Boolean formula with a planar embedding of an instance of positive planar 1-in-3SAT and let Γ' be the drawing constructed from Γ as above.

We will show that there is a valid truth assignment for φ, if and only if Γ' admits a maxSPED with (1) chains of segments alternating between entirely drawn and partially drawn, and where (2) every clause vertex is incident to exactly one entirely drawn segment in Γ'.

"$\Rightarrow$": Let x be a variable, c be a clause containing x and $e = (v_x, v_c)$ be an edge connecting the vertex representing x with the vertex representing c. We draw segment s_x entirely, if the value of x is false, otherwise we draw s_x partially. We alternate between partially drawn segments and entirely drawn segments for the chain of segments representing e. Since chains have even length and precisely one variable in c was true, there is only one segment at v_c entirely drawn, while the two others are partially drawn. Due to the alternation the value of the PED is maximal and the PED is symmetric by construction. Thus a valid truth assignment for the variables of φ creates a valid maxSPED satisfying (1) and (2).

"$\Leftarrow$": Consider a maxSPED of Γ'. Suppose φ has no valid truth assignment, then there exists a clause c containing variables x, y and z, respectively, such that the value of x, y and z is false, respectively. Thus the segment s_v representing the variable $v \in \{x, y, z\}$ is drawn entirely for all $v = x, y, z$. Since chains of segments have even length and together with (1), we conclude that the segments s'_v are entirely drawn, when they meet in v_c at the end of chains of segments for all $v = x, y, z$, contradiction to the validity of the maxSPED of Γ'. Assume (2) is satisfied in the maxSPED of Γ', that is, there are precisely two segments s'_v, s'_w that are partially drawn for $v, w \in \{x, y, z\}$. Since chains of segments have even length and since segments s_v, s_w are also partially drawn due to the invalid truth assignment, we conclude that either (1) is not satisfied or the maxSPED was not valid. Summarizing in every maxSPED not both (1) and (2) can be fulfilled if φ has no valid truth assignment. □

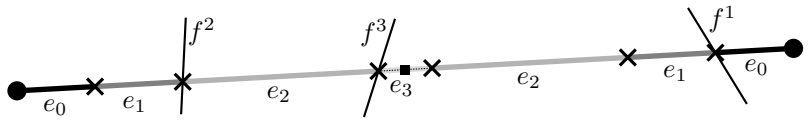


FIGURE 5.6: Edge e is split into four pairs of edge segments; pairs are labeled equally.

5.3.2 Erasing Ink in Arbitrary Graph Drawings

In this section, we consider the problem minSPED, which is dual to maxSPED. In minSPED, we are given a geometric graph and the task is to erase as little of the edges as possible in order to make the embedding a SPED.

We will exploit a connection between the NP-hard minimum-weight 2-SAT problem (MINW2SAT) and minSPED. Recall that the MINW2SAT problem, given a 2-SAT formula, asks for a satisfying variable assignment that minimizes the total weight of the true variables. There is a 2-approximation algorithm for MINW2SAT that runs in $O(vc)$ time and uses $O(c)$ space, where v is the number of variables and c is the number of clauses of the given 2-SAT formula [12].

Theorem 5.9. *MinSPED can be 2-approximated in time quadratic in the number of crossings of the given geometric graph.*

Proof. Let G be an instance of minSPED. We construct an instance φ of the MINW2SAT problem as follows. Let e be an edge of G with k crossings. Then e is split into $k+1$ pairs of edge segments $e_0, \ldots, e_k$ as shown in Figure 5.6. If we order the edges that cross e in increasing order of the distance of their crossing point to the closer endpoint of e, we can assign each segment pair e_i for $i \geq 1$ to the ith edge f^i crossing e, in this order. We also say that edge f^i *induces* segment pair e_i. Any valid maximal (non-extensible) partial edge drawing of e is the union $\bigcup_{i=0}^{j} e_i$ of all pairs of edge segments up to some index $j \leq k$.

We model all pairs of (induced) edge segments as truth variables $\hat{e}_1, \ldots, \hat{e}_k$ with the interpretation that the pair e_i is *not* drawn if $\hat{e}_i = \mathit{true}$. The pair e_0 is always drawn. For $i = 1, \ldots, k$, we introduce the clause $(\neg\hat{e}_{i+1} \Rightarrow \neg\hat{e}_i) \equiv (\hat{e}_{i+1} \vee \neg\hat{e}_i)$. This models that e_{i+1} can only be drawn if e_i is drawn. Moreover, for every crossing between two edges e and f, we introduce the clause $(e_i \vee f_j)$, where e_i is the segment pair of e induced by f and f_j is the segment pair of f induced by e. This simply means that at least one of the two induced segment pairs is not drawn and thus the crossing is avoided.

Now we assign a weight $w_{e,i}$ to each variable $\hat{e}_i$, which is either the absolute length $|e_i|$ of e_i if we are interested in ink, or the relative length $|e_i|/(2|e|)$ if we are interested in relative stub lengths (δ). Then minimizing the value $\sum_{\hat{e}_i \in \mathrm{Var}(\varphi)} w_{e,i}\hat{e}_i$ over all valid variable assignments minimizes the weight of the erased parts of the edges in the given geometric graph.

The 2-approximation algorithm for MinW2Sat yields a 2-approximation for the problem to erase the minimum ink from the given straight-line drawing of G. It runs in $O(vc) = O(I^2) \subseteq O(m^4)$ time since our 2-SAT formula has $O(I) \subseteq O(m^2)$ variables and clauses, where m is the number of edges of G and I is the number of intersections in the drawing of G. □

If we encode the primal problem (maximize ink) using 2SAT, we cannot hope for a similar positive result. The reason is that the tool that we would need, namely an algorithm for the problem MaxW2Sat dual to MinW2Sat would also solve maximum independent set (MIS). For MIS, however, no $(n^{1-\varepsilon})$-approximation exists unless $\mathcal{NP} = \mathcal{ZPP}$ [103].

To see that MaxW2Sat can be used to encode MIS, use a variable $\hat{v}$ for each vertex v of the given (graph) instance G of MIS and, for each edge (u, v) of G, the clause $(\hat{u} \vee \hat{v})$. Let φ be the conjunction of all such clauses. Then finding a satisfying truth assignment for φ that maximizes the number of false variables (i.e., all variable weights are 1) is equivalent to finding a maximum independent set in G. Note that this does *not* mean that maximizing ink is as hard to approximate as MIS.

5.3.3 maxPEDs

Next we give some remarks on maxPEDs. Clearly 1-planar and 2-planar geometric graphs even admit an ncPED, thus it suffices to consider k-planar geometric graphs with $k > 2$.

Since maxSPED is NP-hard, we have the impression that also maxPED is NP-hard. Unfortunately we didn't find a proof for this claim. Also we couldn't adopt the proof for maxSPED, since in the non-symmetric case a stub at one end of the edge is completely independent from the corresponding other stub. To achieve a polynomial-time algorithm for the construction of maxPEDs, we tried to generalize the concept of a matching based on the intersection graph. Unfortunately we were not able to prove optimality for an efficient algorithm in general. At least we can formulate the problem of computing a maxPED for a general geometric graph G as ILP.

We introduce $2m$ variables $s_v^e, s_w^e, e = (v, w) \in E$, which will be the length of the stubs. Also we have at most m^2 binary variables $x_{ee'}$ for every pair of edges $e, e' \in E$ that cross each other [4]. The crossing induced by edge $e = (v, w)$ with edge $e' = (a, b)$ is denoted by p. The set of crossing points is referred to P. Furthermore we use a big constant $M = \max_{e \in E} d(e)$, where $d(e)$ denotes the length of edge e. The exact formulation follows.

$$
\begin{aligned}
\max \quad & \sum_{e=(v,w)\in E} (s_v^e + s_w^e) \\
s.t. \quad & 0 < s_a^{e'} \leq d(a,p) + x_{ee'}M \\
& 0 < s_b^{e'} \leq d(b,p) + x_{ee'}M \\
& 0 < s_v^e \leq d(v,p) + (1 - x_{ee'})M \\
& 0 < s_w^e \leq d(w,p) + (1 - x_{ee'})M \\
& x_{ee'} \in \{0,1\} \quad \forall\, p \in P
\end{aligned}
$$

5.4 Summary and Future Work

We have proved for 1- and 2-planar geometric graphs the existence of ncPEDs. For k-planar geometric graphs we provided a necessary and sufficient condition whether they admit an ncPED. A maxSPED for 1-planar and 2-planar geometric graphs can be computed efficiently. On the other hand we proved NP-hardness for computing a maxSPED in general and presented an approximation algorithm for the dual of maxSPED. We finally conjecture that the problem of computing maxPEDs for geometric graphs is NP-hard. It would be interesting to have at least a good approximation algorithm for maxPED.

Chapter 6

PEDs for Orthogonal 1-bend Drawings

In this chapter we extend PEDs to orthogonal drawings in two dimensions with exactly one bend per edge, called *1-bend drawings*. Indeed, orthogonal drawing is a central concept in graph drawing (see [49, 118] as a reference), and we find it natural to ask how the techniques and results for PEDs carry over to orthogonal drawings. Also, restricting to 1-bend drawings is the first step before extending the model to orthogonal drawings with more than one bend per edge. In a 1-bend drawing, vertices are represented as points with integer coordinates and edges as chains of two orthogonal (axis-aligned) segments. Hence, each vertex can have degree at most 4, ($\Delta_G \leq 4$). A characterization of the graphs that can be drawn orthogonally with one bend per edge has been recently presented by Felsner et al. [79], where they adopted the *general position model*, i.e., no two points can share a coordinate. Indeed, these drawings are also bend minimal, since each edge is represented exactly by one segment parallel to each coordinate axis. In this chapter we also adopt the general position model.

Observe that we cannot directly extend the SHPED model from straight-line drawings to 1-bend drawings, since bent edges might lead to ambiguous interpretations. In particular, it would be unclear how long one should follow a vertical stub or a horizontal stub to reach the bend, see Figure 6.1. Therefore, we introduce the new model *1-bend Orthogonal Partial Edge Drawing*, or simply *1-bend OPED*, where for every edge in a 1-bend drawing we erase the longer segment (we actually draw it as a thin dotted segment). Figure 6.2(a) illustrates a 1-bend drawing of the 4-dimensional hypercube, while Figure 6.2(b) illustrates a 1-bend OPED of the same graph. Similarly as for straight-line PEDs, we define two further properties, *homogeneity* and *symmetry*. Formal definitions for *1-bend homogeneous orthogonal PEDs* (*1-bend HOPED*) and for *1-bend symmetric and homogeneous orthogonal PEDs* (*1-bend SHOPED*) are given in Section 6.1.

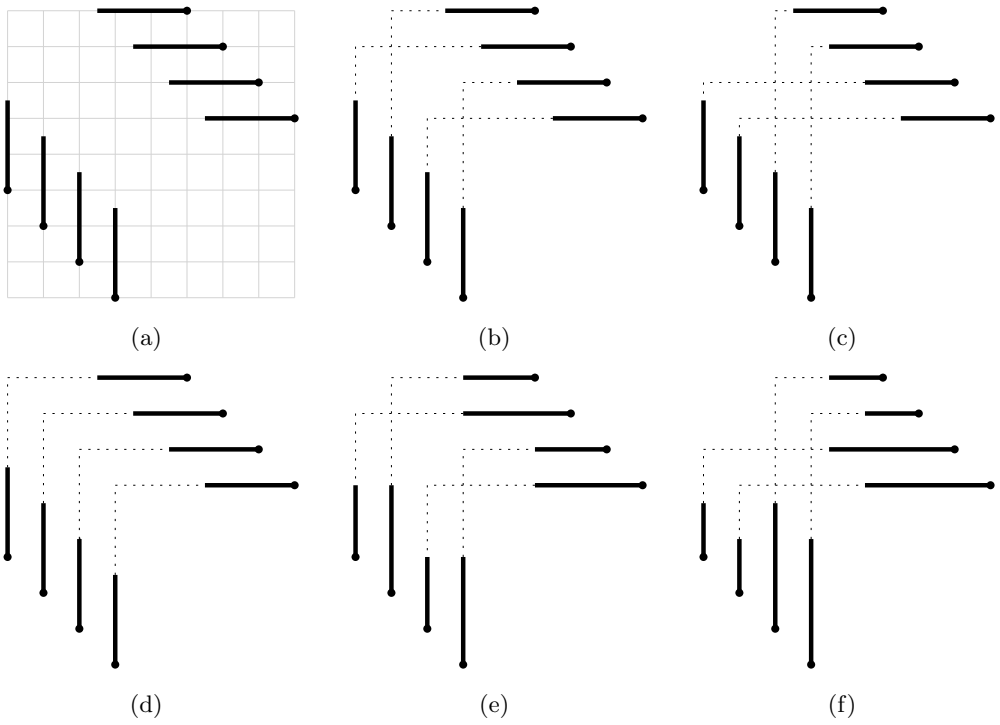
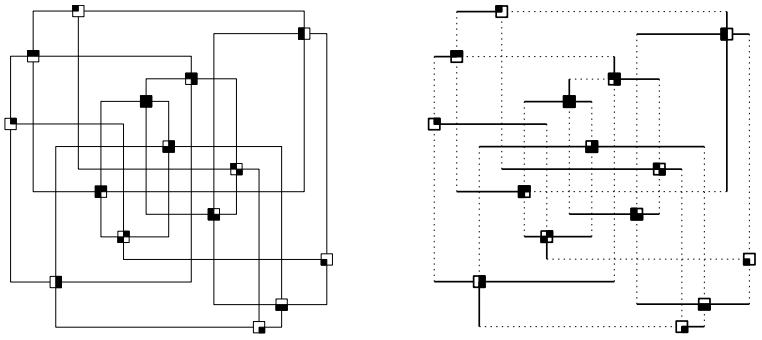


FIGURE 6.1: (a) A 1-bend drawing of a graph with four edges when applying the SHPED model of straight-line drawings, i.e., half of the edge is removed in such a way that a quarter of the edge remains incident to an end-vertex and a quarter to the other end-vertex. It is ambiguous which vertices are connected, since (b), (c) and (d) are different graphs that can be extracted from (a) according to the SHPED model of straight-line drawings. Such an ambiguity can be resolved by using the new definitions for 1-bend SHOPEDs introduced in Section 6.1. Indeed, (d), (e) and (f) are unique 1-bend SHOPEDs of the graphs (d), (b) and (c), resp.

The remainder of this section is organized as follows. Similarly to the straight-line case, we study those graphs that admit 1-bend OPEDs when homogeneity and symmetry are required, where these two properties are defined so to support readability and avoid ambiguities, see Section 6.1. In order to understand the more careful treatment of PED in the orthogonal case as in the straight-line case, we will explore the guidance to PEDs for orthogonal drawings and the involved ambiguity in this part. According to this new model, we show in Section 6.2 that every graph that admits a 1-bend drawing also admits a 1-bend OPED as well as a 1-bend homogeneous orthogonal PED, i.e., a *1-bend HOPED*. Furthermore, we prove in Section 6.3 that all graphs with maximum degree 3 admit a 1-bend symmetric and homogeneous orthogonal PED, i.e., a *1-bend SHOPED*. Concerning graphs with maximum degree 4, we prove in Section 6.4 that the 2-circulant graphs that admit 1-bend drawings also admit 1-bend SHOPEDs, while there is a graph with maximum degree 4 that does not admit such a representation. We finish with a conclusion on 1-bend OPEDs in



(a) (b)

FIGURE 6.2: (a) A 1-bend drawing of the 4-dimensional cube where one vertex has been removed, taken from [79]. (b) A 1-bend OPED drawing of the same graph. Omitted segments are drawn by thin dotted lines.

Section 6.5. These results were published in [207].

6.1 Definitions

Felsner et al. [79] showed that any graph $G = (V, E)$ with $\Delta_G \leq 4$ has a 1-bend drawing if and only if $|E(S)| \leq 2|S| - 2$ for all $S \subset V$. The necessity of this density condition for each induced subgraph of G comes from the clear observation that in any 1-bend drawing there must be a topmost vertex without any vertex towards the top, a bottommost vertex without any vertex towards the bottom, and similarly for the left and right directions. Notice that, 4-regular graphs cannot guarantee such a condition. Hence, Felsner et al. [79] placed one vertex at a point at ∞. Since we consider only the Euclidean plane, we will remove this vertex from the graph.

Let Γ be a 1-bend drawing of a graph G. The *bounding box* R of Γ is the smallest axis-aligned rectangle containing the drawing. Let v be a vertex of G. We denote the four possible anchor points for an edge incident to v by *north*, *south*, *west* and *east ports* of v. Also, we denote the x- and y-coordinates of v in Γ by $x(v)$ and $y(v)$, respectively. Let e be an edge of G, the segments parallel to the x-axis and to the y-axis in the chain of segments representing e in Γ are called *horizontal* and *vertical segments*, respectively, and denoted by e^h and e^v, respectively. The length of a segment s is denoted by $|s|$.

Let $s_e^v \subseteq e^v$, respectively $s_e^h \subseteq e^h$ be subsegments (the *stubs* of e), so that $(e^v \cup e^h) - (s_e^v \cup s_e^h)$ is connected. In what follows, we introduce three variations of our model for orthogonal partial edge drawings. Each variation is defined as an orthogonal 1-bend drawing (short: *1-bend drawing*) in general position, where for every edge e, the two segments e^v and e^h are replaced by their stubs s_e^v and s_e^h (defined above) and no two stubs cross. Furthermore, instead of completely

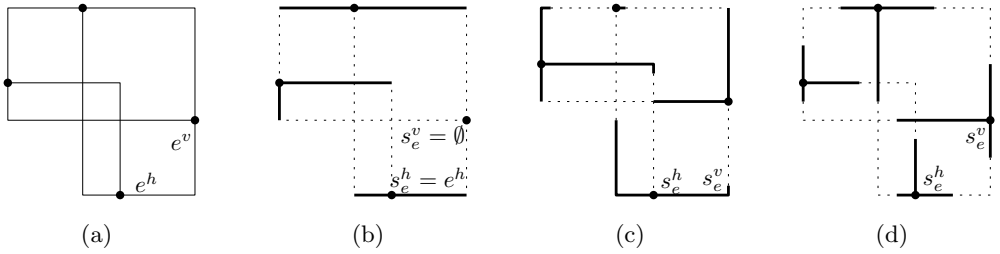


FIGURE 6.3: (a) K_4 has no planar 1-bend drawing, (b) but admits a 1-bend OPED, (c) a 1-bend HOPED and also (d) a 1-bend SHOPED. Observe that in (b) and (c) the bend point is part of the stubs, while in (d) it is not.

erasing the non-drawn parts, we draw them by thin dotted segments, which help the user to follow the edge correctly. This is just a visualization tool to support the reader, since the graph extracted from the drawing is already unique.

We define a *1-bend Orthogonal Partial Edge Drawing*, or simply *1-bend OPED*, as a 1-bend drawing in general position where for every edge e, we remove the longer segment and the remaining (shorter) segments do not cross. That is, if $|e^h| > |e^v|$, then we have $s_e^h = \emptyset$ and $s_e^v = e^v$, and otherwise in case of $|e^h| \leq |e^v|$, we have $s_e^h = e^h$ and $s_e^v = \emptyset$. Since we adopt the general position model, each vertical (horizontal) stub uniquely determines a horizontal (vertical) line on which we can find its end-vertex. An illustration of a 1-bend OPED is given in Figure 6.3(b).

A *1-bend HOPED* is a 1-bend drawing in general position where half of each edge is dropped, while the shorter segment is always entirely drawn. More precisely $|s_e^v| + |s_e^h| = (|e^v| + |e^h|)/2$ is true for every edge e of G and, in addition, we always draw the shorter segment of an edge completely (as in the 1-bend OPED) and draw on the remaining segment (starting from the bend point or from the end-vertex arbitrarily) as long as we need to reach half of the total edge length (see Figure 6.3(c)). Therefore, the two stubs may be continuous, forming a unique bent stub.

A *1-bend SHOPED* is a 1-bend drawing in general position where we symmetrically remove half of the horizontal segment and half of the vertical segment for each edge, i.e., $2|s_e^v| = |e^v|$ and $2|s_e^h| = |e^h|$ for every edge e of G. The dropped parts of e^v and e^h are connected by definition, i.e., they meet at the bend point and the stubs s_e^v and s_e^h are incident to the end vertices of e. An illustration of the 1-bend SHOPED is given in Figure 6.3(d).

6.2 1-bend OPEDs and 1-bend HOPEDs

First we consider 1-bend OPEDs and begin with a simple observation. Every graph that admits a 1-bend drawing also admits a 1-bend OPED: it is sufficient to drop all the horizontal (vertical) segments of a 1-bend drawing after stretching the drawing horizontally (vertically) by a factor equal to the length of the longest vertical (horizontal) segment. If homogeneity is required, the technique we use is slightly more difficult and it is presented in the following.

In the remainder of this section we consider 1-bend HOPEDs. We prove that every graph that admits a 1-bend drawing also admits a 1-bend HOPED. Namely, let G be a graph and let Γ be a 1-bend drawing of G produced by the algorithm of Felsner et al. [79]. We modify the x-coordinates of the vertices of Γ so that for each edge e of G, e^v is always shorter than e^h, and therefore $s_e^v = e^v$. Then, we draw s_e^h always from right to left, as much as we need to reach half of the edge length. Observe that s_e^v and s_e^h might be continuous, forming a unique bent stub. If necessary, we further modify the x-coordinates of the vertices of Γ so that each crossing involving s_e^h is repaired.

Theorem 6.1. *Every n-vertex graph G with $\Delta_G \leq 4$ that admits a 1-bend drawing also admits a 1-bend HOPED.*

Proof. We start by constructing a 1-bend drawing Γ of $G = (V, E)$ by using the technique of Felsner et al. [79]. Recall that we adopt the general position model and consider the total ordering of the (to the right oriented) edges of G defined as follows: $e = (u, v) \prec e' = (w, z)$ if and only if $\max\{x(u), x(v)\} < \max\{x(w), x(z)\}$ and $v \neq z$. The only incomparable edges are those with common rightmost end-vertex. Then we break the ties as follows. Let $e = (u, v)$ and $e' = (w, v)$ be two edges with common rightmost end-vertex. Then $e \prec e'$ if and only if $y(u) < y(w)$. We scan the edges following the order $\prec$ described above. Namely, let $e = (u, v) \in E$ and assume that v is the end-vertex of e placed at the point with largest x-coordinate. Consider the point p_v (if any) defined by the crossing involving e^h with largest x-coordinate, denoted as x_{p_v}. Then, we shift v and all the vertices to the right of v in the following way. For all $w \in V$ with $x(w) \geq x(v)$:

$$\begin{aligned} x(w) &= x(w) + \max\{\delta, \delta'\}, (\text{iff} \max\{\delta, \delta'\} > 0) \text{ , where} \\ \delta &= x(u) - x(v) + |e^v| + 1, \\ \delta' &= 2x_{p_v} - x(u) - x(v) - |e^v| + 1, (\delta' = 0, \text{ if } p_v \text{ not exists}). \end{aligned}$$

Clearly $\delta > 0$ if the vertical segment is at least as long as the horizontal segment of an edge. Thus, the shift operation ensures that the vertical segment is always shorter than the horizontal one (i.e., the vertical segment will be

drawn entirely). Similarly $\delta' > 0$ if the horizontal distance between u and p_v is at least as large as the horizontal distance between p_v and v plus the length of the vertical segment. In this case, drawing the vertical stub entirely and the horizontal stub from right to left as much as necessary to reach half of the total edge length would cross p_v. Hence, the shift operation ensures that all the crossings involving e^h lie on its non-drawn part. It is easy to see that each crossing has been considered exactly once (due to the total order of the edges) and it has been repaired, i.e., it lies on the non-drawn part of the involved horizontal segment. Also, each shift operation does not affect previously repaired crossings. □

Conversely, in Section 6.4 we show that, if symmetry is also required, constructing a 1-bend SHOPED is not always possible.

6.3 1-bend SHOPEDs for Graphs of Maximum Degree 3

We prove that all graphs with maximum degree 3 admit 1-bend SHOPEDs. Namely, we first present an efficient technique to construct 1-bend drawings for biconnected graphs with maximum degree 3, which is of independent interest since it is easy to implement and it has a better time complexity compared to the technique in [79]. Then we show how to turn a 1-bend drawing constructed by this technique into a 1-bend SHOPED. In the end, we extend our results to connected graphs with maximum degree 3.

Lemma 6.2. *Let G be a biconnected n-vertex graph with $\Delta_G \leq 3$. We can construct a 1-bend drawing Γ of G in $O(n)$ time.*

Proof. Given two vertices of G, s and t connected by an edge in G, an *st-numbering* of G is a bijective function, $V \to \{1, \ldots, n\}$, such that s receives number 1 (i.e., $s = v_1$), t receives number n (i.e., $t = v_n$) and every other vertex, except for s and t, is adjacent to at least one lower-numbered and at least one higher-numbered vertex [74]. Let $\{v_1, \ldots, v_n\}$ be an st-numbering of G (with $s = v_1$ and $t = v_n$).

We first construct a 1-bend drawing Γ' of the subgraph $G' = (V' = V \setminus \{s, t\}, E(V'))$ and then add s and t in a proper way.

Construction 1.

We assign to vertex v_i the coordinates $x(v_i) = i$ and $y(v_i) = i$, $i = 2, \ldots, n-1$. We orient the edges $e = (v_i, v_j)$ so that e goes from v_i to v_j, when $2 \leq i < j \leq n-1$. Notice that there are only two possible shapes for the edges in Γ'. Namely, let $e = (v_i, v_j)$, $2 \leq i < j \leq n-1$, be a directed edge. It can either

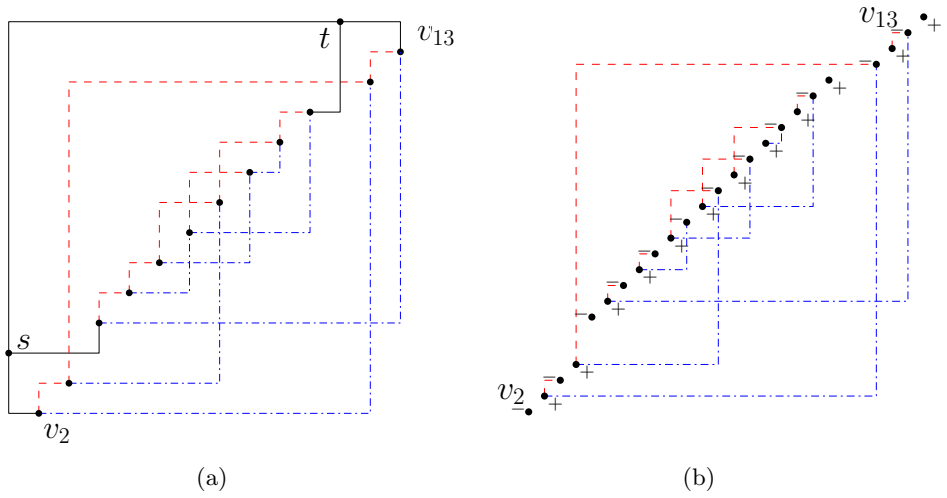


FIGURE 6.4: (a) A 1-bend drawing of a 14-vertex biconnected cubic graph G described in Construction 1 and 2. (b) The graph G^* taken from (a) already colored as described in Construction 1 (red edges are represented by dashed segments, while blue edges by dashed-dotted segments).

leave the east port of v_i and enter the south port of v_j or leave the north port of v_i and enter the west port of v_j. In the first case we call e a *blue edge*, while in the second case we say that e is a *red edge*. We denote by E_B (respectively E_R) the set of blue (respectively red) edges. Due to the *st*-numbering, each vertex has at most two incoming edges and at most two outgoing edges. Furthermore, v_2 has only one incoming edge from s and v_{n-1} has only one outgoing edge to t. We want to find a 2-coloring of the edges $E(V') = E_R \cup E_B$ such that the following two properties hold:

1. $\forall v \in V'$, the (at most) two incoming edges receive different colors.
2. $\forall v \in V'$, the (at most) two outgoing edges receive different colors.

To this aim, we construct the following undirected graph G^* from G'. For each vertex v in G' there will be two vertices v^- and v^+ in G^*. For each edge $e = (w, z)$ in G' (oriented from w to z) there will be an edge $e^* = (w^+, z^-)$ in G^*. See Figure 6.4(b) for an illustration. Thus, G^* is a (possibly not connected) bipartite graph (clearly there are no cycles with odd length) with maximum degree 2. Thus, each connected component is either a path or a cycle. It follows that the edges of G^* can be colored with two colors in a straightforward way. Namely, let C be a component of G^* with m_C edges, we define a total ordering of the edges of C, i.e., $e_1 \prec e_2 \prec \cdots \prec e_{m_C}$. If C is a path such an ordering is directly defined by the order of its edges along the path (rooted at an arbitrary end-vertex). If C is a cycle, we simply choose an arbitrary edge to be the first one (e_1) and remove it from C, the rest of the order is defined

by the remaining path. Finally, we color the edges as follows, e_i receives color $c_{e_i} = i \mod 2, i = 1, \ldots, m_C$. Since there is a clear one-to-one correspondence between edges in G' and edges in G^*, we can directly color the edges of G' with the colors assigned in G^*. Let e' be an edge of G' and let e^* be the respective edge in G^*, we assign to e' the red color if $c_{e^*} = 0$ and the blue color if $c_{e^*} = 1$.

We prove now that such a coloring of the edges of G' respects the properties 1 and 2 defined above. Let e' and e'' be two incoming (outgoing) edges with respect to the same vertex v. By construction they will belong to the same component C of G^*. If C is a path, then e' and e'' will always appear consecutive in any possible order of the edges of C, thus, they will be assigned different colors. If C is a cycle, they may not be consecutive only in such an order where e' is the first (last) one and e'' is the last (first) one. In this case, since m_C is even, they will again receive two different colors.

Γ' is now defined and we only need to place s and t to construct Γ.

Construction 2. (adding s and t)

We recall that s and t are connected by an edge, v_2 has coordinates $(2, 2)$ and just one incoming edge from s, as well as v_{n-1} has coordinates $(n-1, n-1)$ and just one outgoing edge to t. Hence, they can be easily connected to s and t, respectively, without causing crossings. Let v_i, $2 < i < n$, be a possible third vertex connected to s and let v_j, $1 < j < n-1$, be a possible third vertex connected to t. We can skip the following consideration for v_i (respectively v_j), if s (respectively t) has degree 2. We consider the two free ports of v_i and we can assume that they are two consecutive ports. If not, we can just toggle the color of one of the two edges incident to v_i to match this situation (the colors of the edges in the same component of this edge in G^* must be toggled accordingly). Thus, either the north or the south port of v_i is free, as well as either the west or the east port of v_i is free. We choose which port of v_i to use after considering the free ports of v_j. Consider the two free ports of v_j. Either one between the east or the west port is free or, if both are occupied, then both the north and the south port will be free. In total we have 4 possible cases.

1. If the east port of v_j is free, then we set $x(t) = x(v_j) + 0.5$ and $y(t) = n$. While if the west port of v_j is free, $x(t) = x(v_j) - 0.5$ and $y(t) = n$. Also, we can always assign to s the x-coordinate $x(s) = 1$.
   - 1.1 If the north port of v_i is free, then we set $y(s)=y(v_i) + 0.5$, see Figure 6.5(a).
   - 1.2 If the south port of v_i is free, then we set $y(s) = y(v_i) - 0.5$.
2. If the north port of v_j is free, then we set $x(t) = n$ and $y(t) = y(v_j) + 0.5$. While if the south port of v_j is free, $x(t) = n$ and $y(t) = y(v_j) - 0.5$. Also, we can always assign to s the y-coordinate $y(s) = 1$.

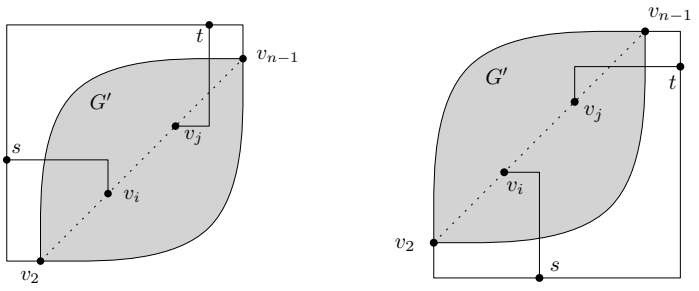


FIGURE 6.5: (a) A 1-bend drawing according to the case 1a of the proof of Lemma 6.2. (b) A 1-bend drawing according to the case 2a of the proof of Lemma 6.2.

2.1 If the east port of v_i is free, then we set $x(s)$=$x(v_i) + 0.5$, see Figure 6.5(b).

2.2 If the west port of v_i is free, then we set $x(s) = x(v_i) - 0.5$.

Notice that, before adding s and t, the vertices were placed in general position, thus there could not be overlaps among edges and vertices. After adding s and t, this property is still maintained due to the introduced fractional coordinates (the grid unit must be halved to get integer coordinates). An example of a 1-bend drawing constructed with this technique is presented in Figure 6.4(a).

Finally, we observe that constructing an st-numbering of G takes $O(n + m)$ time [74], as well as placing vertices (including s and t), constructing G^* and coloring its edges. Thus, since $m \leq 1.5n$, the algorithm runs in $O(n)$ time. □

Theorem 6.3. *Every biconnected n-vertex graph G with $\Delta_G \leq 3$ admits a 1-bend SHOPED. Furthermore, such a drawing can be constructed in $O(n)$ time.*

Proof. Let Γ be a 1-bend drawing of G constructed by Construction 1 and Construction 2. We adopt the notation used in the proof of Lemma 6.2. Consider again the subgraph $G' = (V' = V \setminus \{s, t\}, E(V'))$ and the induced subdrawing Γ'. In Γ' a red edge can be crossed only by red edges, as well as a blue edge can be crossed only by blue edges. Indeed, red edges are all above the diagonal formed by the vertices, while blue edges are all below this diagonal. If a crossing is caused by two red edges, it can be repaired by shifting the rightmost endpoint of the horizontal segment involved in the crossing, so that such a crossing will lie (in a SHOPED) on the non-drawn part of this horizontal segment. In a similar way, if the crossing is caused by two blue edges, it can be repaired by shifting the topmost endpoint of the vertical segment involved in the crossing, so that such a crossing will lie (in a SHOPED) on the non-drawn part of this vertical segment. We will repair the crossings by assigning the vertices new coordinates.

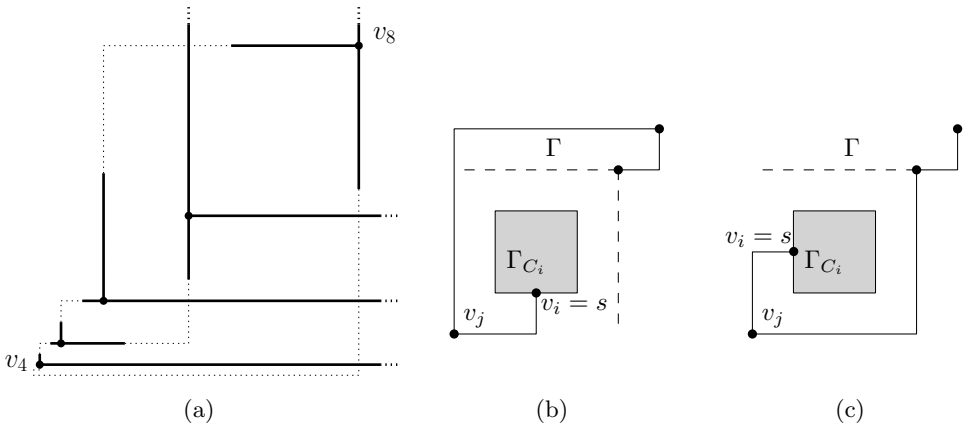


FIGURE 6.6: (a) A 1-bend SHOPED constructed from the drawing in Figure 6.4(a) (for the sake of readability only part of the drawing is shown). (b)-(c) Illustration of the technique described in the proof of Lemma 6.4 to attach the drawing Γ_{C_i} to Γ when the east port (b) or the north port (c) of v_j is free.

Repair 3.

We assign new coordinates to the vertices v_2 to v_{n-1} in the following way. Let $(x(v_i), y(v_i)) = (2^i, 2^i)$ be the new coordinates of $v_i, 2 \leq i \leq n-1$. After that s, t are again placed according to Construction 2. To prove that all crossings are repaired by the assignment of the new coordinates, we consider the vertical segment of the edge between v_i and $v_j, j < i$. The length of the stub on this segment is

$$\frac{y(v_i) - y(v_j)}{2} \leq \frac{y(v_i) - y(v_2)}{2} = y(v_{i-1}) - 2 = y(v_i) - y(v_{i-1}) - 2.$$

Thus the horizontal segments incident to the vertices $v_j, j \leq i-1$ are never crossed by a vertical stub. To prove that vertical segments are never crossed by horizontal stubs we use the same argument for the x-coordinates.

Finally, we need to repair the crossings caused by the outgoing edges of s and by the incoming edges of t. We observe that edges (s,t), (s,v_2),(v_{n-1},t) are not crossed due to the placement of s, t, v_2, v_{n-1} on the bounding box of the drawing, see also Figure 6.5(a) and Figure 6.5(b). Thus, only the crossings affecting (s, v_i), $2 < i < n$, and (v_j, t), $1 < j < n-1$, must be repaired.

Repair 4.

In case 1 of Construction 2, s is always placed 0.5 grid units above or below v_i, thus the vertical segment of the edge (s, v_i) cannot be crossed. In order to fix the crossings on the horizontal segment of (s, v_i) it is enough to shift s to the left, i.e., $x(s) = x(s) - (x(v_i) - x(v_2)) - 1$. Similarly in case 2 of Construction 2, s is always placed 0.5 grid units to the left or to the right of

v_i, thus the horizontal segment of the edge (s, v_i) cannot be crossed. In order to fix the crossings on the vertical segment of (s, v_i) it is enough to shift s to the bottom, i.e., $y(s) = y(s) - (y(v_i) - y(v_2)) - 1$. A symmetric argument can be applied to fix the crossings on the edge (v_j, t). □

A 1-bend SHOPED constructed from the 1-bend drawing in Figure 6.4(a) is shown in Figure 6.6(a).

Next we explain how to extend the previous result to any connected graph G with $\Delta_G \leq 3$. Recall that a *cut vertex* is a vertex whose removal disconnects G, while a *bridge* is an edge whose removal disconnects G. We observe for a graph G with $\Delta_G \leq 3$ that cut vertices are absent in G, if and only if bridges are absent in G. Indeed using the fact that the graphs have maximum degree at most 3, any bridge is incident to at least one cut vertex (and vice versa), justifying the observation.

Lemma 6.4. *Let G be a connected n-vertex graph with $\Delta_G \leq 3$. We can construct a 1-bend drawing Γ of G in $O(n^2)$ time.*

Proof. We start by removing all the bridges of G, obtaining a set of k biconnected components $\mathcal{C} = \{C_1, \ldots, C_k\}$, where each component $C_i \in \mathcal{C}$, $1 \leq i \leq k$, is either a single vertex or a graph such that $\Delta_{C_i} \leq 3$. Next, we define a graph $\mathcal{T}$ having one vertex n_i for each component C_i of G and an edge (n_i, n_j), iff C_i and C_j are connected by a bridge in G. Clearly $\mathcal{T}$ is a tree, since a cycle in $\mathcal{T}$ would imply a biconnected component comprised by the cycle, inferring a contradiction to the decomposition. Also there is at least one vertex n_r in $\mathcal{T}$ that represents a component C_r, which is either a single vertex or a biconnected graph having two adjacent vertices not incident to any bridge. We choose n_r to be the root of $\mathcal{T}$. In the following we describe an algorithm that takes as input a graph G with tree $\mathcal{T}$ and computes a 1-bend drawing Γ of G. We assume that a 1-bend drawing Γ_{C_i} of a component $C_i \in \mathcal{C}$, $1 \leq i \leq k$, is constructed as follows:

(a) If C_i is a biconnected graph with $\Delta_{C_i} \leq 3$, then Γ_{C_i} is always constructed by Construction 1 and 2, where the two poles of the st-numbering will be defined by this construction.

(b) If C_i is composed by a single vertex v, then Γ_{C_i} is defined by placing v in the origin and the definition of the two poles can be ignored.

We visit $\mathcal{T}$ from the root n_r following a Breadth First Search (BFS) order as follows.

Construction 5.

Root n_r: Consider C_r, we construct a drawing Γ_{C_r} of C_r, where in case (a) (i.e., C_r is a biconnected graph with $\Delta_{C_r} \leq 3$) the st-numbering is defined so that s and t are two adjacent vertices in C_r that are not incident to any bridge. Then we set $\Gamma = \Gamma_{C_r}$.

Node $n_i, i \neq r$: Assume n_i is the next vertex of $\mathcal{T}$ according to the bfs order. Let Γ be the drawing constructed so far. We first compute a 1-bend drawing Γ_{C_i} of C_i, where $s = v_i$ and t is an adjacent vertex of s if C_i is a biconnected graph with $\Delta_{C_r} \leq 3$ (case (a)). If C_i is composed by a single vertex v, then $v = v_i$ (case (b)). Drawing Γ_{C_i} is now attached to drawing Γ as follows.

Attachment: Let n_j be the parent of n_i in $\mathcal{T}$ and let (v_j, v_i) be the bridge in G that corresponds to the edge (n_j, n_i) in $\mathcal{T}$. Consider the vertex v_j of C_j. If C_j is a biconnected graph with $\Delta_{C_j} \leq 3$, then v_j cannot be the s pole in Γ_{C_j}. Furthermore, in this case, since the degree of v_j is 2 in C_j, v_j has either the east port or the north port free in Γ. If C_j consists of a single vertex $v = v_j$, then again v_j has either the east port or the north port free in Γ. We use this free port of v_j to place the bridge $e = (v_j, v_i)$ and connect Γ_{C_i} to Γ. In the first case we rotate Γ_{C_i} such that v_i is the southernmost vertex and it can be connected by its south port, see Figure 6.6(b), while in the second case we rotate Γ_{C_i} such that v_i is the westernmost vertex and it can be connected by its west port, see Figure 6.6(c).

Before attaching the drawing Γ_{C_i}, we modify the current drawing Γ in the following way. We assign new x-coordinates to all vertices v with $x(v) > x(v_j)$ by setting $x(v) = x(v) + |C_i| + 1$, and we assign new y-coordinates to all vertices v with $y(v) > y(v_j)$ by setting $y(v) = y(v) + |C_i| + 1$. Now we place Γ_{C_i} in this free area, i.e., for each vertex $v_{C_i} \in C_i$, $x(v_{C_i}) = x(v_j) + x_{\Gamma_{C_i}}(v_{C_i})$, and $y(v_{C_i}) = y(v_j) + y_{\Gamma_{C_i}}(v_{C_i})$, where $x_{\Gamma_{C_i}}(v_{C_i})$ and $y_{\Gamma_{C_i}}(v_{C_i})$ are the x- and y-coordinates of v_{C_i} in Γ_{C_i}, respectively. Then we connect v_j with v_i. Notice that, by placing Γ_{C_i} in this free area, no edges of $\Gamma \setminus \Gamma_{C_i}$ can cross edges of Γ_{C_i}.

Finally, we observe that finding the bridges of G can be done in linear time [180]. Furthermore, constructing a 1-bend drawing Γ_{C_i} for each component $C_i \in \mathcal{C}$ takes $O(|C_i|)$ time, thus, constructing Γ takes $\sum_{i=1}^{k} O(|C_i|) = O(n)$ time. However, the time complexity of the technique is dominated by the shift operation required to add the drawing of each component to the current drawing, which takes $O(n^2)$. □

By iteratively applying Repair 3 and 4 to drawings constructed by Construction 5 we can prove the next theorem.

Theorem 6.5. *Every n-vertex graph G with $\Delta_G \leq 3$ admits a 1-bend SHOPED. Furthermore, such a drawing can be constructed in $O(n^2)$ time.*

Proof. Let Γ be a 1-bend drawing of G constructed by Construction 5. We adopt the notation used in the proof of Lemma 6.4 and we traverse $\mathcal{T}$ in the reversed bfs order defined in that proof. Also, we assume that a 1-bend drawing Γ_{C_i} of a component $C_i \in \mathcal{C}$, $1 \leq i \leq k$, can be transformed into a 1-bend SHOPED as follows. If C_i is a biconnected graph with $\Delta_{C_i} \leq 3$, then Γ_{C_i} is repaired by Repair 3 and 4. If C_i is composed by a single vertex v, then Γ_{C_i} does not need to be transformed into a 1-bend SHOPED.

Let Γ_{C_i} be the 1-bend drawing of C_i and let Γ_{C_j} be the 1-bend drawing of C_j, where n_j is the parent of n_i in $\mathcal{T}$. Recall that no edges of C_j can cross edges of C_i in Γ. First, we transform Γ_{C_i} into a 1-bend SHOPED. Then, let $s_i = \max\{width(\Gamma_{C_i}), height(\Gamma_{C_i})\}$, we assign new x-coordinates to all vertices v of Γ_{C_j} with $x(v) > x(v_j)$ by setting $x(v) = x(v)+s_i+1$, and new y-coordinates to all vertices v with $y(v) \geq y(v_j)$ by setting $y(v) = y(v) + s_i + 1$.

The time complexity is dominated by the construction of Γ, which takes $O(n^2)$ time, while repairing the components takes $\sum_{i=1}^{k} O(|C_i|) = O(n)$ time. □

6.4 1-bend SHOPEDs for Graphs of Maximum Degree 4

We first present a class of graphs with maximum degree 4, the 2-circulant graphs, that admit a 1-bend SHOPED, and show afterwards that there is a graph that does not admit a 1-bend SHOPED.

Recall that the k-circulant graph C_n^k with $n > 2k$ vertices is the simple graph whose vertex set is $V = \{v_0, \ldots, v_{n-1}\}$ and whose edge set is $E = \{(v_i, v_j) : |j - i| \leq k\}$. The specified index of a vertex is calculated modulo n. Notice that, $\Delta_{C_n^k} = 4$ implies $k = 2$, hence, each vertex has exactly two neighbors with smaller indices and two neighbors with larger indices. Greater values of k are not realizable when $\Delta(C_n^k) = 4$. Extending the techniques in Section 6.3, we can prove the next theorem.

Theorem 6.6. *Every 2-circulant n-vertex graph that admits a 1-bend drawing also admits a 1-bend SHOPED. Furthermore, such a drawing can be constructed in $O(n)$ time.*

Proof. We start by observing that one vertex of a 2-circulant n-vertex graph, $v_t \in V$ $(0 \leq t \leq n-1)$ has to be removed in order to match the necessary density condition for 1-bend drawings, i.e., $|E(S)| \leq 2|S| - 2$ for all $S \subset V$. W.l.o.g., let $t = 0$, otherwise the vertices of C_n^2 can be easily renumbered so to match this condition.

We first construct a 1-bend drawing Γ' of $G' = (V' = V \setminus \{v_t{=}v_0, v_1, v_{n-1}\}, E(V'))$ adopting a similar strategy as in Construction 1. Namely, we assign to

vertex $v_i \in V'$ coordinates $x(v_i) = i$ and $y(v_i) = i$, $2 \leq i \leq n-2$. Again, there can be only two possible shapes of edges in Γ' according to such a placement of the vertices. Indeed, let $e = (v_i, v_j)$, so that $j = i+1, i = 2, \ldots, n-3$, we call e a red edge, $e \in E_R \subset E(V')$, and we draw it so that it leaves the north port of v_i and enters the west port of v_j. While, if $e = (v_i, v_j)$, so that $j = i+2, i = 2, \ldots, n-4$, we call e a blue edge, $e \in E_B \subset E(V')$, and we draw it so that it leaves the east port of v_i and enters the south port of v_j. Thus, red edges are never crossed, while each blue edge receives at most two crossings (one involving the vertical segment and one involving the horizontal segment).

To construct a 1-bend drawing Γ of G, the addition of v_1 and v_{n-1} to Γ' can be managed by adopting the same strategy used to add s and t in Construction 2 Finally, we can apply Repair 3 and 4 (restricted to blue edges) to turn Γ into a 1-bend SHOPED. □

Now, we show that there exists a graph with maximum degree 4 that admits a 1-bend drawing but not a 1-bend SHOPED. To this end, we need to introduce some additional notation. Consider a 1-bend SHOPED Γ. We say that two horizontal (vertical) stubs s^h_e, $s^h_{e'}$ (s^v_e, $s^v_{e'}$) *overlap*, if there exists a vertical (horizontal) line l such that $s^h_e \cap l \neq \emptyset$ and $s^h_{e'} \cap l \neq \emptyset$ ($s^v_e \cap l \neq \emptyset$ and $s^v_{e'} \cap l \neq \emptyset$). Two horizontal (vertical) stubs overlap by u units if there are two vertical (horizontal) lines l,l' with the above property and such that their horizontal (vertical) distance is u. Also, we call $\prec_x$ and $\prec_y$ the total vertex orderings induced by the projection of the x- and y-coordinates of the vertices in Γ, respectively (we remark that we adopt the general position model).

Recall from [79] (see also Section 6.1) that any graph $G = (V, E)$ with $\Delta_G \leq 4$ has a 1-bend drawing if and only if $|E(S)| \leq 2|S| - 2$ for all $S \subset V$. This necessary and sufficient condition is satisfied by any of the following three cases: (*i*) there are four vertices of degree at most three; (*ii*) there are two vertices of degree at most two; (*iii*) there are one vertex of degree at most two and two vertices of degree at most three. Since a 4-regular graph cannot guarantee such a condition, one vertex must be removed. Furthermore, if we are in case (*i*) each side of the bounding box of any possible 1-bend drawing contains one of the four vertices of degree three. In case (*ii*) the two vertices of degree two must be placed at the opposite corners of the bounding box. While in case (*iii*) the degree two vertex must be placed at one corner of the bounding box and the other two vertices of degree three must be placed on the two opposite sides. We call these vertices lying on the sides of the bounding box by *external vertices*. All the other vertices must lie in the interior of the bounding box and we refer to them as *internal vertices*.

Consider a graph $G' = (V', E')$ so that $V' = \{v_1, \ldots, v_6\}$ and $E' = \{(v_1, v_2), (v_2, v_3), (v_4, v_5), (v_5, v_6)\}$, see Figure 6.7(a). We prove that there exist two total vertex orderings $\prec_x, \prec_y$ such that in every 1-bend drawing of G' at least two stubs cross each other. Recall that $\prec_x$ and $\prec_y$ are orders, where the x- and y-order of the vertices coincide with these two orderings.

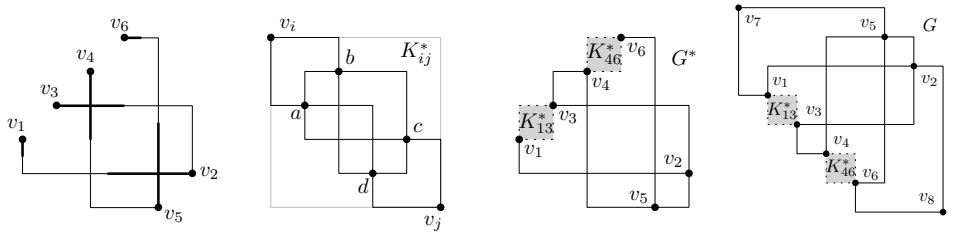


FIGURE 6.7: (a) A drawing Γ' of G' that cannot be redrawn as a 1-bend SHOPED if the vertical and horizontal order of the vertices cannot be changed. The relative order of each pair of vertices in G' can be constrained by the gadget in (b). (c) An illustration of the graph G^* as defined in the proof of Theorem 6.8. (d) A graph G that admits no 1-bend SHOPED.

Lemma 6.7. *Graph G' does not admit a 1-bend SHOPED so that $v_1 \prec_x v_3 \prec_x v_4 \prec_x v_6 \prec_x v_5 \prec_x v_2$ and $v_5 \prec_y v_2 \prec_y v_1 \prec_y v_3 \prec_y v_4 \prec_y v_6$.*

Proof. Let Γ' be a 1-bend drawing of G' respecting the properties defined in the statement of this lemma (see Figure 6.7(a)), and consider the stubs (in the 1-bend SHOPED model) of the segments representing the edges of G' in Γ'. First, we observe that the two horizontal stubs $s^h_{v_1v_2}$ and $s^h_{v_2v_3}$ overlap, because of $v_1 \prec_x v_3 \prec_x v_2$, that is:

$$|s^h_{v_1v_2}| + |s^h_{v_2v_3}| = |s^h_{v_2v_3}| + |s^h_{v_2v_3}| + \frac{1}{2}[x(v_3) - x(v_1)]$$

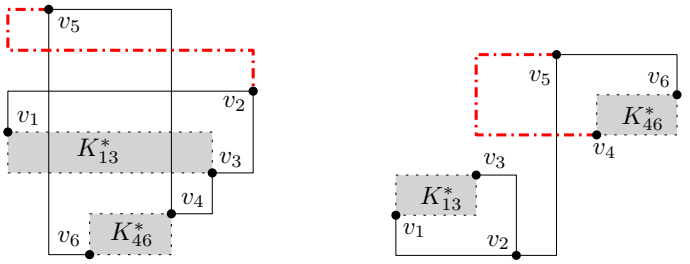
By definition $x(v_2) - x(v_3) = 2|s^h_{v_2v_3}|$, which implies:

$$|s^h_{v_1v_2}| + |s^h_{v_2v_3}| = x(v_2) - x(v_3) + \frac{1}{2}[x(v_3) - x(v_1)]$$

Since $x(v_3) - x(v_1) \geq 1$, the two stubs must overlap by at least half of a grid unit. In a similar way, the two vertical stubs $s^v_{v_4v_5}, s^v_{v_5v_6}$ overlap, because of $v_5 \prec_y v_4 \prec_y v_6$. Assume the stub $s^v_{v_5v_6}$ does not cross the stub $s^h_{v_1v_2}$. Due to the overlap between $s^v_{v_5v_6}$ and $s^v_{v_4v_5}$, and since $v_3 \prec_x v_4 \prec_x v_5$, it follows that $s^v_{v_4v_5}$ will cross either $s^h_{v_1v_2}$ or $s^h_{v_2v_3}$, or $s^v_{v_5v_6}$ will cross $s^h_{v_2v_3}$. □

In order to remove the assumption that the x-order and the y-order of the vertices cannot be changed, we augment G' adding new edges and vertices obtaining a new graph G, such that in every 1-bend drawing of G the x-order and the y-order of the vertices in V' is the same up to rotation. Let K_4 be the complete graph with 4 vertices. Let K'_4 be a copy of K_4 and let a, b, c, d (in clockwise order traversing the bounding box) be its 4 vertices. We connect v_1 to a and b and v_3 to c and d. See also Figure 6.7(b). In a similar way we add one more copy of K_4 between v_4 and v_6. We call the subgraph induced by the pair v_1, v_3 and its copy of K_4 as K^*_{13}, analogously we call K^*_{46} the subgraph induced

by the pair v_4, v_6 and its copy of K_4. Furthermore, we add the edge (v_3, v_4) and (v_2, v_5). Finally, we add the vertices v_7 and v_8 and the edges $(v_1, v_7), (v_5, v_7)$ and $(v_2, v_8), (v_6, v_8)$. The final graph G is shown in Figure 6.7(d), it clearly admits a 1-bend drawing.



(a) (b)

FIGURE 6.8: An illustration of the case 1 (a) and 2 (b) in the proof of Theorem 6.8. The red dash dotted edge indicates an edge, which cannot be drawn with 1 bend.

Theorem 6.8. *There exists a graph that does not admit a 1-bend SHOPED.*

Proof. Consider the graph G shown in Figure 6.7(d). We will evaluate all the possible configurations in terms of x-order and y-order for the vertices of G. For every feasible configuration we will then apply Lemma 6.7 to prove that there is no 1-bend SHOPED.

Let Γ be a 1-bend SHOPED of G. Consider the vertices v_7 and v_8, they must be the external vertices of Γ and they must be placed at the corners of one of the two diagonals of the bounding box R of Γ. Consider the bounding-box R^* of the subdrawing Γ^* induced by the graph $G^* = (V^* = V \setminus \{v_7, v_8\}, E(V^*))$ (see Figure 6.7(c)). Vertices v_1, v_2, v_5, v_6 must be the external vertices of Γ^* and they must lie one on each side of R^*. For the same reason, vertices v_3, v_4 are internal vertices (as well as all the vertices in the two copies of K_4) of Γ^*. Also, consider the bounding-box R_{13} of the subdrawing Γ_{13} induced by the graph K^*_{13}. Vertices v_1 and v_3 must be the external vertices of Γ_{13} and must be placed at the corners of one of the two diagonals of R_{13}. W.l.o.g., let v_7 be at the top-left corner of R. Hence, either v_1 is the westernmost vertex and v_5 is the northernmost vertex in Γ^* or vice versa. Assume to be in the former case, since the latter case can be proved with symmetric arguments. It follows that v_2 is either the southernmost or the easternmost vertex, while v_6 is either the easternmost or the southernmost vertex in Γ^*.

We start the analysis looking at the vertex v_1. We already know that $v_1 \prec_x v_3$ and consider two possible cases for the relative y-order of these two vertices.

1. Assume $v_3 \prec_y v_1$. In this case, since the only free port of v_1 is the north one, we have $v_1 \prec_y v_2$. This implies that v_2 cannot be the southernmost vertex, but it must be the easternmost vertex and v_6 the southernmost instead. Hence, the edge (v_2, v_3) must leave the south port of v_2 and enter the east port of v_3. It follows that the only free port of v_3 is the south one, and, due to the edge (v_3, v_4), $v_4 \prec_y v_3$. Now consider two further subcases, either $v_4 \prec_x v_3$ or $v_3 \prec_x v_4$. We prove by contradiction that $v_3 \prec_x v_4$.

   1.1 Let $v_4 \prec_x v_3$ (see Figure 6.8(a)), this implies that $v_6 \prec_x v_4$. Consider the edge (v_5, v_6). It must leave the west port of v_6 and enter the south port of v_5. Also, consider the edge (v_4, v_5), it must leave the north port of v_4 and enter the east port of v_5. Finally, consider the edge (v_2, v_5), again it must leave the north port of v_2 and enter the west port of v_5, thus, it must be bent at least three times, a contradiction.

   1.2 Let $v_3 \prec_x v_4$, thus $v_4 \prec_x v_6$. Consider the edge (v_5, v_6), it must leave the east port of v_6 and enter the south port of v_5, thus $v_6 \prec_x v_5$. Also, consider the edge (v_4, v_5), it must leave the north port of v_4 and enter the west port of v_5. Finally, consider the edge (v_2, v_5), it can be placed so that it leaves the east port of v_5 and enters the north port of v_2. Notice that, $v_6 \prec_y v_4 \prec_y v_3 \prec_y v_1 \prec_y v_2 \prec_y v_5$, as well as $v_1 \prec_x v_3 \prec_x v_4 \prec_x v_6 \prec_x v_5 \prec_x v_2$. Thus, by Lemma 6.7, there is no 1-bend SHOPED for the subdrawing Γ' induced by the subgraph G'.

2. Assume $v_1 \prec_y v_3$, as depicted in Figure 6.8(b). In this case, since the only free port of v_1 is the south one, we have $v_2 \prec_y v_1$. Now consider the edge (v_2, v_3), it must leave the north port of v_2 and enter the east port of v_3. Also, consider the edge (v_2, v_5), it must leave the east port of v_2 and enter the south port of v_5. This implies that v_2 is the southernmost vertex and v_6 is the easternmost vertex. Thus, consider the edge (v_5, v_6), it must leave the east port of v_5 and enter and north port of v_6. Now, since v_4 must lie on the opposite extreme of the diagonal of R_{46} with respect to v_6, we have that $v_4 \prec_y v_6$. Thus, since the edge (v_4, v_5) must use the west port of v_5 and the west or south port of v_4, it must have at least two bends, which implies that this configuration is not feasible and that $v_3 \prec_y v_1$.

□

A natural question which is still open, regards the complexity of deciding whether a graph with maximum degree 4 admits a 1-bend SHOPED. In what follows, we give an ILP formulation for a very related problem. Given a graph G such that $\Delta_G = 4$, a 1-bend drawing Γ of G, and a parameter L, we ask whether it is possible to transform Γ into a 1-bend SHOPED without modifying the x- and y-order of its vertices such that the 1-bend SHOPED fits into

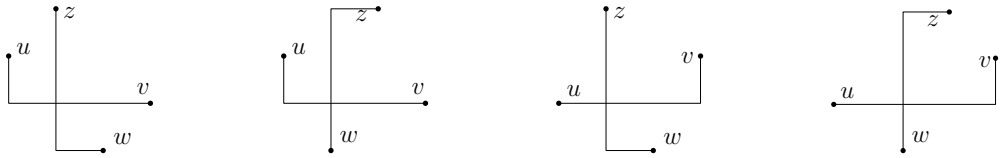


FIGURE 6.9: (a) An illustration of the case $b_i^x = 1$ and $b_i^y = 1$. (b) An illustration of the case $b_i^x = 1$ and $b_i^y = 0$. (c) An illustration of the case $b_i^x = 0$ and $b_i^y = 1$. (d) An illustration of the case $b_i^x = 0$ and $b_i^y = 0$.

a square of size L^2. We call this problem the *FixedSizeFixedOrderSHOPED* problem.

Theorem 6.9. *The FixedSizeFixedOrderSHOPED problem can be formulated as an ILP.*

Proof. Let $G = (V, E)$ be an n-vertex graph, such that $\Delta_G = 4$, and let Γ be a 1-bend drawing of G (in the general position model). Also, let $N \in O(n^2)$ be the number of edge crossings in Γ. We construct an ILP formulation with $2n$ integer variables, i.e., the x- and y-coordinates of each vertex, and $2N + 4n$ constraints.

We encode each crossing c_i, $i = 1, \ldots, N$, as $\{(u, v), (w, z), b_i^x, b_i^y\}$, where (u, v) and (w, z) are the two edges causing the crossing. Without loss of generality, we assume that $x(u) < x(v)$ and $y(w) < y(z)$ in Γ, and that the crossing lies on the horizontal segment of (u, v) and on the vertical segment of (w, z). Also, b_i^x is a constant equal to 1 if the bend of (u, v) has coordinates $(x(u), y(v))$ or it is equal to 0 if such a bend has coordinates $(x(v), y(u))$. Similarly, b_i^y is a constant equal to 1 if the bend of (w, z) has coordinates $(x(z), y(w))$ or it is equal to 0 if such a bend has coordinates $(x(w), y(z))$. See Figure 6.9 for an illustration of the above notation.

In order to repair the crossing c_i, one of the following two inequalities must be satisfied. Let $\delta^x_{(u,v)} = x(v) - x(u)$ and $\delta^y_{(u,v)} = y(v) - y(u)$:

$$
\begin{aligned}
& b_i^x\{b_i^y\delta^x_{(z,v)} + (1-b_i^y)\delta^x_{(w,v)}\} + (1-b_i^x)\{b_i^y\delta^x_{(u,z)} + (1-b_i^y)\delta^x_{(u,w)}\} \\
> \quad & b_i^x\{b_i^y\delta^x_{(u,z)} + (1-b_i^y)\delta^x_{(u,w)}\} + (1-b_i^x)\{b_i^y\delta^x_{(z,v)} + (1-b_i^y)\delta^x_{(w,v)}\} \\
& \text{or} \\
& b_i^x\{b_i^y\delta^y_{(v,z)} + (1-b_i^y)\delta^y_{(w,v)}\} + (1-b_i^x)\{b_i^y\delta^y_{(u,z)} + (1-b_i^y)\delta^y_{(w,u)}\} \\
> \quad & b_i^x\{b_i^y\delta^y_{(w,v)} + (1-b_i^y)\delta^y_{(v,z)}\} + (1-b_i^x)\{b_i^y\delta^y_{(w,u)} + (1-b_i^y)\delta^y_{(u,z)}\}
\end{aligned}
$$

Hence, for each crossing we have the following two constraints.

$$
\begin{aligned}
& b_i^x\{b_i^y\delta^x_{(z,v)} + (1-b_i^y)\delta^x_{(w,v)}\} + (1-b_i^x)\{b_i^y\delta^x_{(u,z)} + (1-b_i^y)\delta^x_{(u,w)}\} \\
- \quad & b_i^x\{b_i^y\delta^x_{(u,z)} + (1-b_i^y)\delta^x_{(u,w)}\} + (1-b_i^x)\{b_i^y\delta^x_{(z,v)} + (1-b_i^y)\delta^x_{(w,v)}\} > e_i M \\
& \text{and} \\
& b_i^x\{b_i^y\delta^y_{(v,z)} + (1-b_i^y)\delta^y_{(w,v)}\} + (1-b_i^x)\{b_i^y\delta^y_{(u,z)} + (1-b_i^y)\delta^y_{(w,u)}\} \\
- \quad & b_i^x\{b_i^y\delta^y_{(w,v)} + (1-b_i^y)\delta^y_{(v,z)}\} + (1-b_i^x)\{b_i^y\delta^y_{(w,u)} + (1-b_i^y)\delta^y_{(u,z)}\} > (1-e_i)M
\end{aligned}
$$

Where $e_i \in \{0,1\}$ is a binary variable [4] that is $e_i = 0$, if the vertical part of the crossing segments can be drawn arbitrarily long and just the horizontal stub has a restricted length, while for $e_i = 1$ the horizontal part of the crossing segments can be drawn arbitrarily long and just the vertical stub has restricted length. The transformation from "or" to "and" uses also a big constant $M = 4L$ that can not be reached by summing up four stubs.

Also, let Π_x and Π_y be the functions defining the x- and y-order of the vertices in Γ, respectively. We add the following $2n$ constraints, which ensure the preservation of the x- and y-orders of the vertices.

$$
\begin{aligned}
x(u) &= 1, u \in V : \Pi_x(u) = 1; \quad x(u) < x(v), \forall u, v \in V : \Pi_x(v) = \Pi_x(u) + 1 \\
y(u) &= 1, u \in V : \Pi_y(u) = 1; \quad y(u) < y(v), \forall u, v \in V : \Pi_y(v) = \Pi_y(u) + 1
\end{aligned}
$$

Finally we add the following $2n$ constraints that force the x- and y-coordinates into a square of size L^2.

$$0 < x(v) \leq L \text{ and } 0 < y(v) \leq L \; \forall \; v \in V$$

The cost function is irrelevant and thus can be set to a constant, e.g., $\min 1$. Notice that this ILP cannot test the overall existence of a 1-bend SHOPED, but we can test the existence of a 1-bend SHOPED of a certain size. □

6.5 Summary and Future Work

We defined a new drawing model for orthogonal drawings with one bend per edge, called 1-bend Orthogonal Partial Edge Drawing, extending the already existent PED model for straight-line drawings. We studied those graphs that admit such a representation when homogeneity or both symmetry and homogeneity are required. In the former case, we proved that every graph that admits a 1-bend drawing also admits a 1-bend HOPED. In the latter case, we proved that all graphs with maximum degree 3 and the 2-circulant graphs that admit a 1-bend drawing, also admit a 1-bend SHOPED. Furthermore we

proved that there is a graph with maximum degree 4 that does not admit a 1-bend SHOPED.

The complexity of the decision problem is still open, i.e., deciding whether a graph with maximum degree 4 admits a 1-bend SHOPED. We formalized a related question as an integer linear program (ILP), i.e., a test of a 1-bend drawing, whether it admits a 1-bend SHOPED inside a square of given size without changing the relative horizontal and vertical order of the vertices, but the complexity of this question is also still open. Also, it would be of interest to study 1-bend SHOPEDs where a few crossings among stubs are allowed, so to enlarge the family of graphs that admit these representations. To extend our model, one may consider graphs with degree greater than 4, by following similar approaches as in [86] for representing vertices, or orthogonal drawings with more than one bend per edge.

Chapter 7

Short Conclusion on PEDs

Partial edge drawing is a new drawing model and may appear in different settings. The most common and intuitive one is the 1/4-SHPED, which is applicable for many classes of graphs. We have also seen that not every graph admits a 1/4-SHPED. This drawing model is also appropriate to combine with other drawing conventions apart from the straight-line drawing convention, i.e., orthogonal variants were considered. Another way of extension is to apply this drawing model on Lombardi drawings [38, 64, 69, 133], which uses circular arcs instead of straight-lines. We also present a 1/4-SHPED spring embedder which is applicable for many graphs. In a user study we found out that partial edge drawings are not worse than the traditional straight-line drawing style. In contrast some tasks were completed faster and more accurately using partial edge drawings. In the geometric embedding setting, many questions on PEDs are difficult. This is especially true for k-planar drawings, where $k > 2$.

Part II

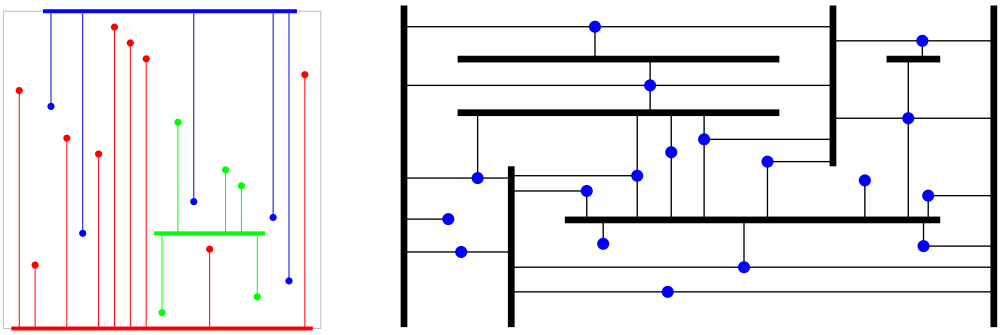
Bus Realizations

Chapter 8

Introduction

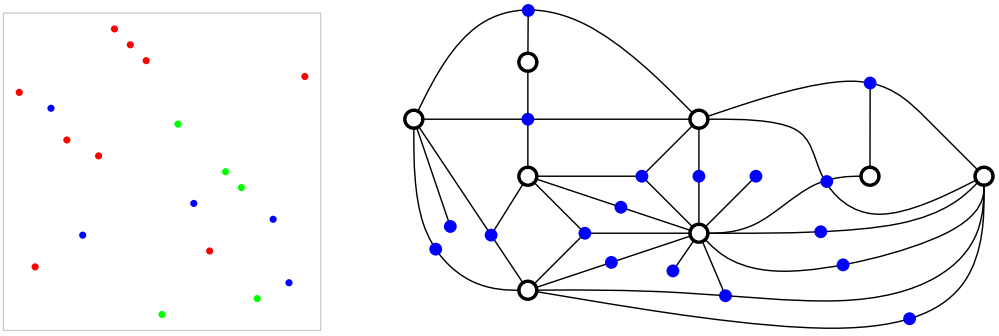
A *bus realization* of a hypergraph is a drawing with hypervertices as points and hyperedges as bold segments such that a point is orthogonally connected to a segment if and only if the respective hypervertex is incident to the respective hyperedge. We define the *dimension* as the number of slopes for the segments in a bus realization. One-dimensional bus realizations use only horizontal segments, while two-dimensional bus realizations use horizontal and vertical segments. In both dimensions the realizations resemble orthogonal drawings since the slopes of the segments and the edges together are horizontal and vertical. Extensions with more than two dimensions are mentioned as open problems. In a bus realization the bold segments are called *buses* or *bus segments*, the points are called *connectors* and the thin segments connecting a bus with a connector are called *connections*. A *planar realization* is a realization without crossings. We will mainly focus on planar realizations. To distinguish between buses and connections in a bus realization, the bus segments are drawn with bold lines. Figure 8.1(a) shows a 1-dimensional planar bus realization, and Figure 8.1(b) shows a 2-dimensional planar bus realization.

Instead of considering a hypergraph $H = (V, B)$, we may consider its incidence graph $G = (V \cup B, E)$ as defined in Section 2.3. This graph contains a vertex for every hypervertex v and every hyperedge b and an edge (v, b), if $v \in b$. In the remainder we will call this incidence graph *bus graph* in order to emphasize our objective, which is producing a bus realization from the incidence graph. For a bus graph $G = (V \cup B, E)$ we will refer to vertices in B as *bus vertices* and vertices in V as *connector vertices*. A *planar bus graph* is a bus graph that admits a drawing without crossings, which is not necessarily a planar bus realization. We will focus our attention on planar bus graphs, since planar realizations imply planarity for the bus graph. To distinguish between a connector vertex v and a bus vertex b of a bus graph we illustrate v as point and b as circle, see Figure 8.2(b). Figure 8.2(a) shows the bus graph of Figure 8.1(a), while Figure 8.2(b) shows the bus graph of Figure 8.1(b).



(a) (b)

FIGURE 8.1: (a) A 1-dimensional planar bus realization, where points and buses are colored. (b) A 2-dimensional planar bus realization. In both realizations the buses are bold segments, connectors are points, and connections are orthogonally thin connected from buses to connectors.



(a) (b)

FIGURE 8.2: (a) The bus graph of Figure 8.1(a) with fixed vertex positions, where for convenience bus vertices are left out. Instead, the points representing connectors incident to the same bus vertex are colored the same. (b) The bus graph of Figure 8.1(b) with circles representing bus vertices and points representing connector vertices.

The structure of this part is as follows. We will point out related work in Section 8.1. Deciding whether a bus graph admits a 2-dimensional bus realization is NP-complete [3]. In Chapter 9 we show that on the other hand the question whether a planar bus graph admits a planar 2-dimensional bus realization can be answered in polynomial time. We will also remark differences to non-planar 2-dimensional bus realizations at the end of this section. In Chapter 10 we consider planar 1-dimensional bus realizations. There we provide a short argument that this problem can be easily transformed into finding a visibility drawing using Section 2.5. Therefore we restrict our research on planar bus graphs with fixed vertex positions and investigate whether they admit a planar 1-dimensional bus realization. We discuss restricted versions in particular to uncover the limits of efficient computability. We conclude this part in

Chapter 11.

8.1 Related Work

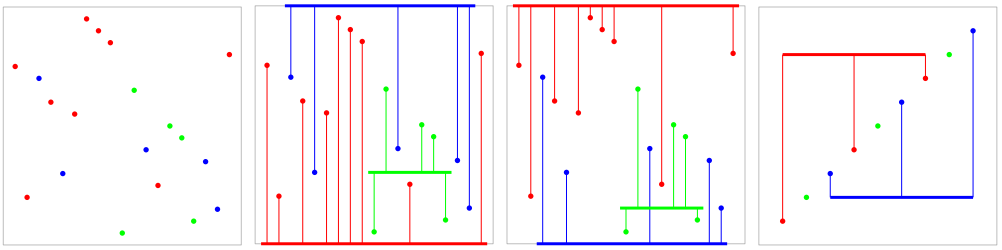
A classical topic in the area of graph visualization is *orthogonal graph drawing*, which is covered in many books on graph drawing [49, 118, 145], or surveys [195]. In this drawing model each edge consists of a series of alternating horizontal or vertical line segments. Applications can be found in, e.g., VLSI design [132, 183]. In this application it may also be necessary to use hypergraphs as models. For example, power buses on VLSI chips are often modeled as hyperedges, as well as Local Area Networks in computer network visualization. Bus graphs are a possible approach to modeling hyperedges since bus realizations can provide a good visualization of hypergraphs. A bus-style representation can also be used when facing the visualization of highly interconnected parts of a given graph, see Figure 1.6. In this way, cliques can be represented in a compact and comprehensive way using a bus-style model as an alternative model to edge bundling and confluent drawings [56, 94].

Visualization of hypergraphs is an important topic in the areaa of graph drawing and information visualization. The traditional approach relies on representing overlapping sets via Venn diagrams and Euler diagrams [168, 169]. When more than a handful sets are present, however, such diagrams become difficult to interpret and alternative approaches, such as compact rectangular Euler diagrams are needed [160].

Rather than subset-based approaches [19, 117] or incidence graphs, researchers have recently focused on planar support [32], path-based support [29] or tree-support of hypergraphs [120]. Here the approaches are to connect the hypervertices by a graph, called *support*, so that a single hyperedge induces a connected subgraph.

Ada et al. [3] used horizontal and vertical buses in a bus realization, implying to be 2-dimensional, and thus restricted bus graphs to be the incidence graphs of hypergraphs with hypervertex degree at most four. They considered the problem to decide whether a bus graph has a 2-dimensional bus realization and showed NP-completeness. In Chapter 9 we consider the problem of deciding whether a planar bus graph has a planar 2-dimensional bus realization. We show that a planar 2-dimensional bus realization can be constructed on a $O(n)\times O(n)$ grid in $O(n^{5/2})$ time if it exists. This bus graph approach is related to rectangular drawings, rectangular duals, and visibility graphs since edges in bus graphs enforce visibility constraints in realizations, see [104, 145, 163, 179] for key concepts.

When visualizing a hypergraph, considered as visualization of set membership, then often the geometric position of the elements of the sets are prescribed as points in the plane. The task is to emphasize the sets where the elements



(a) (b) (c) (d)

FIGURE 8.3: (a) Fixed positions of points, where points with the same color belong to the same set. (b) A planar 1-dimensional bus realization for this setting, while (c) is a non-planar 1-dimensional bus realization. (d) A point set without any planar 1-dimensional bus realization.

belong to. In visualization approaches for set memberships of items on maps, this is done by connecting points from the same set by corresponding lines (LineSets [5]), tree structures (KelpFusion [137]), and enclosing polygons (BubbleSet [41] or MapSets [66]).

In 1-dimensional bus realizations we consider a unified version of the tree-structure approach using the bus model known from VLSI design. Our goal in Chapter 10 is a membership visualization of points in sets by a tree-structure that consists of a single horizontal bus to which all the points from the same set are connected by vertical segments; see Figure 8.3 for planar and non-planar versions. We assume the sets to be given by single-colored points such that, in the final bus realization, all points of the same color are connected to exactly one bus associated with this color. The objective is to find a position for each bus such that crossings of buses with connections are avoided. We call this the *bus embeddability problem* (BEP). Such a simple visualization scheme makes it very easy to recognize the sets and label them, by placing a label inside each bus (if the bus is drawn thick enough), or directly above/next to the bus.

A solution to the BEP problem can be viewed as planar tree support for hypergraphs, and this problem is related to Steiner trees [112], where the goal is to connect a set of points in the plane while minimizing the sum of edge lengths in the resulting tree; this is a classic NP-complete problem [95, 126].

Hurtado et al. [111] considered planar supports for hypergraphs with two hyperedges such that the induced subgraph for every hyperedge as well as for their intersection is a Steiner tree. Their objective was to minimize the sum of edge lengths, while allowing degree one or two for the hypervertices.

BEP is even more closely related to rectilinear Steiner trees [92], where the Euclidean distance is replaced by the rectilinear distance; constructing rectilinear Steiner trees is also NP-complete [96]. A *single-trunk* Steiner tree [37] is a path that contains all vertices of degree greater than one. This is a variant that is solvable in linear time. BEP for a single set is the single trunk rectilinear Steiner tree problem, where we ignore the minimization of the sum of the

edge lengths. Thus BEP can be seen as a simultaneous single-trunk rectilinear Steiner tree problem. The fact that a bus placement influences the placement of other buses makes the problem hard.

Consider the input to BEP along with a box that encloses all the points. If in BEP the buses extend to the right boundary of this box, or both to the left and right boundary of this box, then this problem corresponds to backbone boundary labeling [16] and can be solved efficiently. In backbone boundary labeling, the problem is to orthogonally connect points by a horizontal backbone segment leading to a label placed at the boundary. In this setting it is always possible to split the problem into two independent subproblems, which is impossible in our case.

BEP is also related to the classical *point set embeddability problem*, where we are given a set of points along with a planar graph, and need to determine whether there exists a mapping of vertices to points such that the resulting straight-line drawing is planar. The general decision problem is NP-hard [34]. Several variants of this problem already exist such as what is the smallest size of a point set where any planar graph can be embedded, but only results for smaller graph classes are known, like outerplanar graphs [25], simply-nested graphs [7], and planar three-trees [88]. In Part III we add 2-outerplanar graphs to this list. In the variant of orthogeodesic point set embedding, Katz et al. proved that deciding whether a planar graph can be embedded using only orthogonal edge routing is NP-hard [116] on the grid – even for matchings.

In BEP the bus graph represents a hypergraph with hypervertex degree one and disjoint hyperedges. Thus we consider the problem from the point of view that every point has a color and every bus connects all points of the same color. The answer of BEP builds the base for extensions of this problem such as considering higher degree of hypervertices and therefore more directions of buses.

The 2-dimensional bus realization considered as the single trunk rectilinear Steiner tree implies a horizontal or vertical slope for the single trunk. When asking for bus graphs admitting a bus realization of dimension d, we can directly ignore all bus graphs that have connector vertices with a degree of more than $2d$. In our abstract definition of bus graph, the degree restriction on the hypervertices is directly implied by the dimension of the bus realization. Since the testing of a bus graph for maximum degree can be done efficiently, we will adopt the degree restriction on the connector vertices for the bus graph in the respective section. Thus in Chapter 9, when considering horizontal and vertical segments, the bus graph is the one defined by Ada et al. [3], while in Chapter 10, when considering only horizontal segments, the bus graph is defined by hypergraphs, where hypervertices are part of at most two hyperedges, respectively exactly one hyperedge.

Chapter 9

Bus Graphs in Two Dimensions

In this chapter we consider planar bus realizations, where the bus segments will be drawn either vertically or horizontally. Thus we are considering the bus graph model introduced by Ada et al. [3], summarized as follows: A *bus graph* is a bipartite graph $G = (V \cup B, E)$, where $E \subseteq V \times B$ and $deg(v) \leq 4$ for all $v \in V$. They considered the problem of deciding whether a bus graph has a realization and showed NP-completeness. In this chapter we consider the problem of deciding whether a planar bus graph has a planar realization. We show that this question, in contrast to the previous result, can be decided in polynomial time.

Recall that a planar bus graph is a bus graph that admits a drawing without crossings, while a *plane bus graph* is a planar bus graph together with a planar embedding. We will first focus on plane bus graphs and then extend the results to planar bus graphs. We always assume to have a connected bus graph, since components can be considered separately. We will assign labels to the bus vertices that determine whether they are to be realized vertically or horizontally and study properties that have to be obeyed by this labeling so that it corresponds to a planar realization. In Section 9.1 we identify necessary conditions for the labeling – if they are satisfied, we speak of a *good partition*. In Section 9.2, we present an algorithm to test whether a maximal plane graph admits a good partition. Subsequently, in Section 9.3, we give a linear-time algorithm that produces a planar realization on a grid of size $O(n) \times O(n)$ from a maximal plane bus graph together with a good partition. The approach is based on techniques from [44] and [78]. In the next section, Section 9.4, we extend the test for a good partition to non-maximal plane bus graphs. For an extension to (not-embedded) planar bus graphs, we first recall main definitions for SPQR-trees in Section 9.5 and extend the results from Section 9.4 to biconnected planar bus graphs in an easy form (Section 9.6) then to biconnected planar graphs without simplifications (Section 9.6.1) and finally to general planar bus graphs in Section 9.6.2. We also consider in Section 9.7 some non-planar

cases of realizations. We summarize this chapter in Section 9.8. The results are based on [202] and [203].

9.1 Necessary Properties

Clearly non-planar bus graphs cannot be drawn in a planar way. So we focus on finding planar structures in a planar bus graph. First we assume the bus graph $G = (V \cup B, E)$ to be *plane*, i.e., a planar bus graphs together with a fixed planar embedding. Later we concentrate on planar bus graphs without any given embedding. We assume any bus graph to be *simple*, i.e., there are no multiple edges (multiple edges would necessarily be on top of each other).

A planar realization of a plane bus graph $G = (V \cup B, E)$, induces labels H or V for the bus vertices where H indicates that a bus is represented by a horizontal segment in the realization, while V indicates that the bus is represented by a vertical segment. Let $\Pi = (B_{\mathrm{V}}, B_{\mathrm{H}})$ denote the *partition* of B according to the labels of the bus vertices. We observe two natural properties of a partition $\Pi = (B_{\mathrm{V}}, B_{\mathrm{H}})$ corresponding to a planar realization:

(P1) Every connector vertex of degree at least 3 has neighbors in both classes.

(P2) A connector vertex of degree 4 has cyclically alternating neighbors in both classes.

Properties (P1) and (P2) for a partition of the bus vertices are not sufficient to ensure a planar representation. Also a third property for what we call 'diamonds' must be true. A *diamond* in a plane bus graph is a cycle $z = (b, v, b', v')$ of length four with bus vertices b, b' and connector vertices v, v' such that both v, v' have a third bus neighbor in the bounded region defined by z. Note that the outer cycle of G may be a diamond.

Lemma 9.1. *Let G be a plane bus graph that has a realization inducing the partition $(B_{\mathrm{V}}, B_{\mathrm{H}})$ of the set of buses B. Then, for any diamond $z = (b, v, b', v')$ the two bus vertices b and b' belong to the same class (B_{V} or B_{H}).*

Proof. Suppose for contradiction that $b \in B_{\mathrm{H}}$ and $b' \in B_{\mathrm{V}}$. The interior of z in the planar bus realization is a polygon with six corners. Four of the corners are at contacts of connector edges and buses and two corners are at the connector vertices. We account for four corners of size $\pi/2$ each, where the edges meet the buses. The other two corners are at v and v'. Since b is horizontal and b' vertical, the angles at v and v' have to be either $\pi/2$ or $3\pi/2$. Because v and v' have an additional bus neighbor in the interior, the angle at each of v and v' is at least π. Hence, both these angles are of size $3\pi/2$. Therefore, the sum of interior angles is at least $4 \cdot \pi/2 + 2 \cdot 3\pi/2 = 5\pi$. A six-gon, however, has a

sum of angles of 4π. The contradiction shows that b and b' belong to the same class of the partition $(B_{\mathrm{V}}, B_{\mathrm{H}})$. □

An *oversaturated diamond* of G is a diamond $z = (b, v, b', v')$ with the property that one of the connector vertices has degree four and all its neighbors lie in the closed bounded region defined by z. From the lemma we see that a plane bus graph with an oversaturated diamond cannot admit a planar bus representation. Since our algorithm will identify all diamonds, it can reject instances that contain oversaturated diamonds right away. Therefore we will henceforth assume that the plane bus graph that we consider contains no oversaturated diamond.

As a consequence of the lemma we state a third property to ensure a planar realization.

(P3) A diamond has both bus vertices in the same class.

Definition 9.2. A partition $\Pi = ((B_{\mathrm{V}}, B_{\mathrm{H}}))$ of the buses of a plane bus graph G is called a *good partition* if it obeys properties (P1), (P2), and (P3).

In the next section we consider maximal plane bus graphs and test efficiently whether they admit a good partition. The test is constructive, i.e., if the answer is yes, then a good partition is constructed.

9.2 Maximal Plane Bus Graphs

A *maximal plane bus graph* G is a plane bipartite bus graph that is a quadrangulation, i.e., all faces have cardinality 4. Let $G = (V \cup B, E)$ be a maximal plane bus graph. We assume that G has no connector vertices of degree 1 or 2, since they have no influence on the existence of a good partition. Let $\Delta = \Delta(G)$ denote the *degree of a plane bus graph* G which is defined as the maximum degree among the connector vertices of G.

In this section we first assume that G has $\Delta = 3$ and give an algorithm to test if G admits a good partition and if so the algorithm returns a good partition. After that we allow $\Delta = 4$ and reduce this case with a simple modification to the case $\Delta = 3$.

Let $G = (V \cup B, E)$ be a plane maximal bus graph with $\Delta = 3$. The *connector graph* $C_G = (V_C, E_C)$ of G consists of all the connector vertices $V_C = V$ and edges $(v, v') \in E_C$, if v and v' are both incident to the same face of the plane embedding of G. The connector graph is helpful because it allows the translation of the problem of finding a good partition for G to the problem of

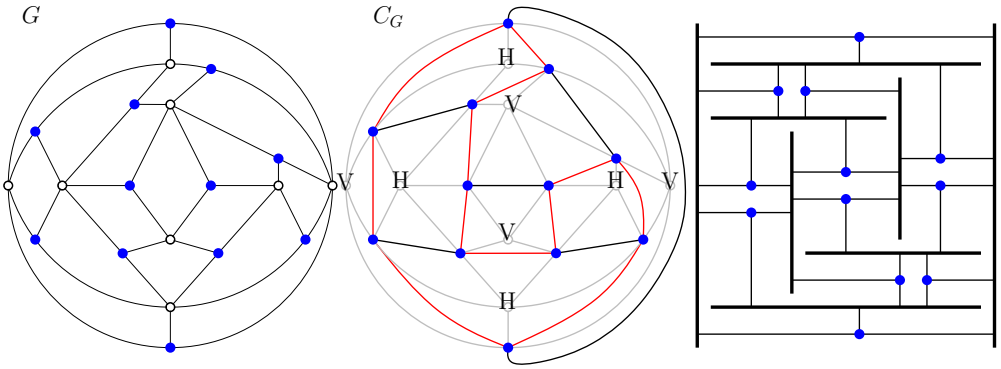


FIGURE 9.1: A maximal plane bus graph, the connector graph with a matching (fat black edges) and its good partition, and a corresponding bus representation.

finding an appropriate perfect matching in C_G, summarized in Proposition 9.3 and Proposition 9.4 and illustrated in Figure 9.1.

The first property (P1) of a good partition Π of G requires that every connector vertex v has two adjacent bus vertices in one partition class and one in the other. If b and b' are neighbors of v in G with the same label, then there is a connector vertex v' sharing a face with v, b, and b', since every face has cardinality 4. When looking at v' the two neighbors b and b' are again the two in a common partition class. Hence, property (P1) of a good partition of G induces a perfect matching on C_G.

Conversely a perfect matching M of C_G induces a labeling of the bus vertices. Removing the matching edges from C_G leaves a 2-regular graph, i.e., a disjoint collection of cycles. The regions defined by this collection of cycles can be 2-colored with colors V and H such that each cycle has regions of different colors on its sides. Let B_{V} be the set of bus vertices in faces colored with V, and let B_{H} be the set of bus vertices in faces colored with H. This yields a partition satisfying (P1) because every connector vertex is on a cycle and has a bus neighbor in each of the two faces bounded by the cycle. Since $\Delta = 3$, the second property (P2) is void.

Consider a diamond $z = (b, v, b', v')$ in G. Each of v, v', has exactly one edge e, e' in C_G that corresponds to a face of the outside of z. Since (P3) forces equal labels for b, b', the faces represented by e, e' have equally labeled incident bus vertices (b, b') and thus e, e' must be in a matching of C_G. We define the set E_d of *edges forced by diamonds* as the set of edges consisting of the two outside edges e, e' in C_G for each diamond z of G. We have thus shown that a perfect matching M induced by a good partition contains E_d.

Conversely, if E_d is contained in a perfect matching M of C_G, then the bus vertices b, b' of each diamond are in the same partition class and thus G has

a partition that satisfies property (P3). The findings are summarized in the following proposition.

Proposition 9.3. *Let G be a maximal plane bus graph with $\Delta = 3$, let C_G be its connector graph, and let E_d be the set of edges of C_G forced by diamonds. Then G admits a good partition iff C_G has a perfect matching M, with $E_d \subseteq M$.*

Now we allow $\Delta = 4$ for a maximal plane bus graph G. To transform G into a plane bus graph G' with $\Delta = 3$, we split every connector vertex v of degree 4 into two connector vertices v', v'', both of degree 3 in the following way: let b_1, b_2, b_3, b_4 be the adjacent bus vertices of v in consecutive cyclic order around v. Remove v and its incident edges and introduce new vertices v', v'' with edges (b_1, v'), (b_2, v'), (b_3, v'), (b_3, v''), (b_4, v''), (b_1, v''). The connector graph C_G is obtained from $C_{G'}$ by contracting the edges (v', v'') corresponding to the pairs v', v'' that have been obtained by splitting a vertex of degree 4. Define the set E_s of edges *forced by splits* of $C_{G'}$ as the set of these edges (v', v'').

If G has a partition satisfying property (P2), then we have to ensure alternating labels for the neighbors of v in G. This forces $(v', v'') \in E_s$ to be a matching edge in $C_{G'}$. Conversely if $(v', v'') \in E_s$ is a matching edge, then the common neighbors b_1, b_3 have the same label. Since v', v'' have both degree 3 and two of their neighbors have equal label, the third neighbor (for each of v', v'') has a different label, i.e. b_2 and b_4 have both different label compared to the label of b_1, b_3, hence, v obeys property (P2). So in total a partition Π of G satisfies property (P2), iff the edges E_s are contained in a matching of $C_{G'}$. For notational simplicity we denote the connector graph $C_{G'}$ of the transformed graph G' by C_G. An example for a maximal plane bus graph with its connector graph showing the edges of $E_d \cup E_s$ is shown in Figure 9.2.

Proposition 9.4. *Let G be a maximal plane bus graph with $\Delta = 4$ and C_G its connector graph and let E_d, E_s be the sets of edges of C_G that are forced by diamonds and splits, respectively. The graph G admits a good partition iff C_G has a perfect matching M, with $(E_d \cup E_s) \subseteq M$.*

Proof. The proof almost follows from Proposition 9.3 and the above considerations. Splitting connector vertices of degree 4, however, may separate diamonds. We claim that this is no problem. Let v be split into v', v'' as above. Any diamond containing v has a cycle z with bus vertices b_1 and b_3. Condition (P3) for this diamond requires that b_1 and b_3 belong to the same class of a good partition. This requirement, however, is already implied by condition (P2) for the original connector vertex v. That is, separated diamonds do not impose additional conditions on the matching. □

Theorem 9.5. *Let G be a maximal plane bus graph. A good partition for G can be computed in $O(n^{3/2})$ time if it exists.*

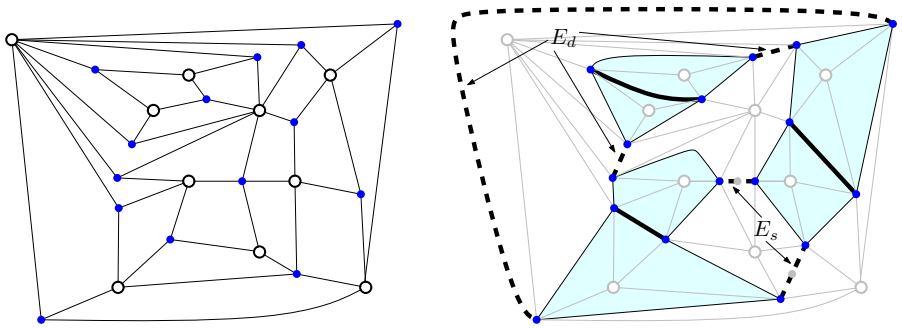


FIGURE 9.2: A maximal plane bus graph and its connector graph with the modifications, where the dotted edges are forced and the fat edges complete the perfect matching.

Proof. By Proposition 9.4 it suffices to test the connector graph C_G for a perfect matching M that contains $(E_d \cup E_s)$. The extraction of the connector graph C_G from G requires linear time. The set E_s can be computed while constructing C_G. To identify diamonds we consider the dual D_G of the connector graph C_G. The vertices of D_G are the bus vertices of G and edges correspond to faces of G'. Diamonds of G' correspond to double edges of D_G. The only exception is the diamond bounding the outer face. Double edges of D_G can be found and sorted so that the set E_d can be constructed in $O(n \operatorname{polylog}(n))$ time. To force $E_d \cup E_s$, we simply delete all vertices incident to these edges from the graph. If a vertex is incident to two edges from the set, then there is no matching. For constructing a perfect matching in the remaining graph, there exist several $O(\sqrt{n}m)$-time algorithms. For planar graphs this yields the claimed time complexity[1] of $O(n^{3/2})$.

Given the perfect matching, the corresponding good partition can again be computed in linear time. □

Note that in general a perfect matching in a bridgeless planar 3-regular graph exists by Peterson's theorem [151] and can be computed in linear time [21]. Unfortunately this efficient algorithm doesn't support the requirement that edges of E_s, E_d have to participate in the matching. This is because the algorithm from [21] splits the graphs into two bridgeless biconnected components by a "suitable reduction" and iterates on these components. This "suitable reduction" stands for the choice of an edge to be in the matching, respectively to be not in the matching, so that the resulting components have those properties. In our case we don't have the freedom of this choice in general because of the enforced matching edges.

[1] In [142] a slightly faster randomized algorithm for planar graphs has been proposed.

9.3 Planar Realizations

In this section we show how to construct a planar realization from a maximal plane bus graph G with a good partition. Our main result, which we are going to prove in the remainder of this section, is as follows.

Theorem 9.6. *Let G be a maximal plane bus graph admitting a good partition. Then G has a planar realization on a grid of size $O(n) \times O(n)$. If the good partition is given, the realization can be computed in $O(n)$ time.*

Let G be a maximal plane bus graph admitting a good partition. We start with some simplifications. First we recursively remove all connector and bus vertices of degree 1 and all connector vertices of degree 2. Second we split all connector vertices of degree 4 into two connector vertices of degree 3. After these two modifications the new graph G' is still a quadrangulation.

The *reduced bus graph* $R' = (B', E_R)$ of G' is the graph on the bus vertices of G' with edges (b, b'), iff b, b' are incident to a common face and have different labels. Diamonds with different labeled bus vertices are the only substructure that would create double edges in R' but diamonds have identically labeled bus vertices in a good partition. Hence, there are no double edges in R'. From the three faces incident to a connector vertex exactly two contribute an edge to R'. It follows that R' is a quadrangulation. Another approach to derive this is by observing that the edges of the matching M of Proposition 9.4 are in bijection with the faces of R'.

Let Q be a quadrangulation, which is a bipartite graph since all faces have cardinality 4. We call the color classes of the bipartition white and black and name the two black vertices on the outer face s and t. A *separating decomposition* of Q is an orientation and coloring of the edges of Q with colors red and blue such that:

- All edges incident to s are ingoing red and all edges incident to t are ingoing blue.
- Every vertex $v \neq s, t$ is incident to a non-empty interval of red edges and a non-empty interval of blue edges. If v is white, then, in clockwise order, the first edge in the interval of a color is outgoing and all the other edges of the interval are incoming. If v is black, the outgoing edge is the last one in its color in clockwise order (see Figure 9.3).

Separating decompositions have been studied in [44], [78] and [77]. In particular it is known that every plane quadrangulation admits a separating decomposition. To us separating decompositions are of interest because of their connection with segment contact representations of the underlying quadrangulation. The following lemma has been shown by Felsner in [76]; an illustration of the lemma is given in Figure 9.4.

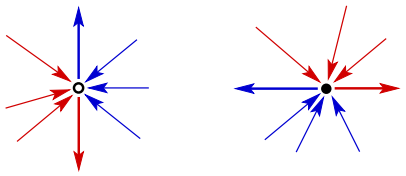


FIGURE 9.3: Edge orientations and colors at white and black vertices.

Lemma 9.7 (Felsner [76]). *A separation decomposition of Q can be used to construct a segment contact representation of Q with vertical and horizontal segments such that an edge $v \to w$ of the separating decomposition corresponds to a contact of the segments S_v and S_w where an endpoint of S_v is in the interior of S_w.*

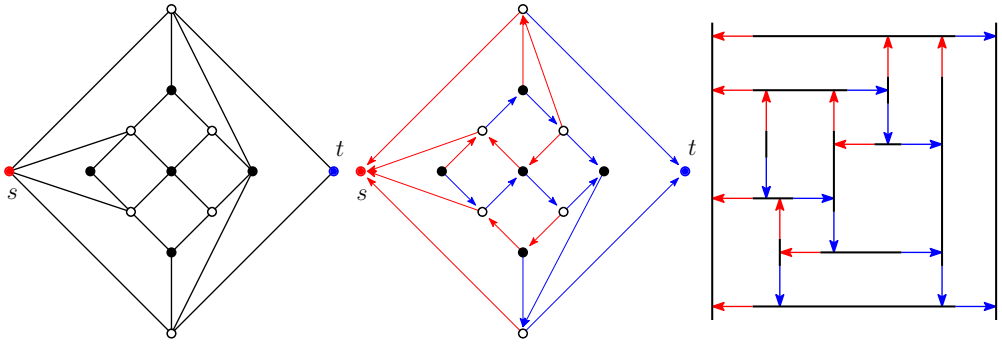


FIGURE 9.4: A quadrangulation Q, a separating decomposition of Q and a corresponding segment contact representation of Q.

The construction of a planar realization is as follows. First we identify the two classes V and H of the bipartition of the reduced bus graph R' with black and white. Next we construct a separating decomposition of R' and a corresponding segment contact representation. Later the following observation will be important:

($\star$) The rectangles in the segment contact representation correspond bijectively to the faces of R'. Moreover, vertex b is incident to face f in R' if and only if segment S_b contributes a side of the rectangle R_f corresponding to f.

From the segment contact representation of R' we obtain a representation of the bus graph G' in two steps. First we clip the endpoints of all segments of the representation so that a disjoint collection of segments remains. These segments serve as the bus segments for the representation of the bus graph G'. It remains to insert the connector vertices and the edges of G' into the picture. To this end recall that each connector vertex belongs to a unique face of R' and each face of R' contains exactly two connector vertices. The two connector vertices contained in a face f can easily be accommodated in the rectangle R_f, because of ($\star$). Figure 9.5 shows the picture.

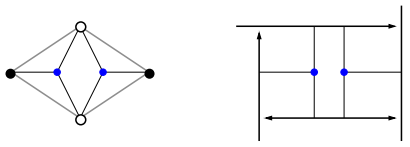


FIGURE 9.5: A face f of R' with its two connector vertices and the placement of the two vertices in R_f.

At this point we have a representation of the slightly modified maximal plane bus graph G'. It remains to transform the planar representation of G' into a planar representation of the original input graph G. These are the steps that have to be done:

- Merge pairs of connector vertices that have been created by splitting a connector vertex of degree 4.
- Insert all connector and bus vertices of degree 1 and all connector vertices of degree 2 that had been deleted in the reverse order of the deletion.

This yields a representation of the input graph G.

To complete the proof of Theorem 9.6 it remains to argue about the complexity. Let $G = (V \cup B, E)$ be the maximal plane input bus graph with $n = |V| + |B|$. The slightly modified bus graph G' is obtained by removing or splitting some vertices, which can be done in linear time. The reduced bus graph R' can be computed from the plane G' in $O(n)$ time. A separating decomposition of R' can also be computed in linear time, details can be found in the PhD thesis of É. Fusy [90]. The segment contact representation of R' associated with the separation decomposition is computable in linear time with standard techniques, c.f. [49] or [76]. The number of grid lines needed for the representation of R' is bounded by the number of vertices n' of R', i.e., the size of the grid is at most $n' \times n'$. The compression of grid lines that don't contain any connector vertex or endpoint of a bus segment, as well as the reinsertion of connector vertices and bus vertices and the merging of vertices that were created by splitting a connector vertex of degree 4, leads to an $O(n) \times O(n)$ grid. Finally each grid line is occupied by either a connector vertex or a bus segment along the grid line or the start point, respectively the end point, of a bus segment, i.e., the total number of occupied grid lines is $3|B| + 2|V|$. This is true, even if we extend the graph at the beginning to match maximality (see the next section) and remove the inserted vertices at the end.

Now we finished the construction from a given maximal plane bus graph to a planar realization through the computation of a good partition for the bus vertices. In the next sections we generalize the result from Section 9.2, i.e., as input graph we permit non-maximal plane bus graphs (Section 9.4), then we permit biconnected non-embedded planar bus graphs (Sections 9.5, 9.6, 9.6.1)

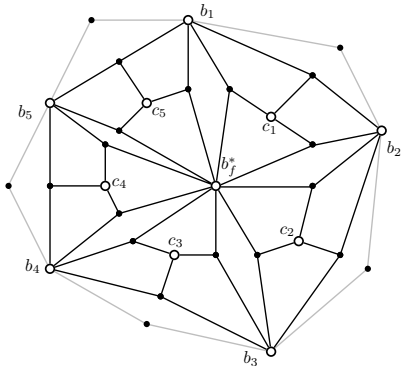


FIGURE 9.6: New vertices and edges added to quadrangulate a face f of cardinality 10.

and finally we permit general non-embedded planar connected bus graphs (Section 9.6.2). All extensions offer as a final result a maximal plane bus graphs together with a good partition, if it exists, for which Theorem 9.6 from this section can be applied as a black box.

9.4 Non-Maximal Plane Bus Graphs

In this section we consider a plane bus graph G that is not necessarily maximal. In a first preprocessing step we remove all connector vertices of degree 1 as well as their incident edge. These objects can easily be integrated in a realization of the remaining graph.

In the following we describe how to extend G to a maximal plane bus graph G^+ containing G as induced subgraph such that G^+ has a good partition iff G has a good partition (Lemma 9.8). The graph G^+ will be called *the quadrangulation* of G.

Let f be a face with cardinality $2k$ in G and let $b_1, \ldots, b_k$ be the bus vertices of f in clockwise order. To *quadrangulate* f we first place a new bus vertex b_f^* in the inside. The bus vertex b_f^* is then connected to the boundary of f by adding a triangular structure for every consecutive pair b_i, b_{i+1} of bus vertices including the pair b_k, b_1. The triangular structure for b_i, b_{i+1} consists of another new bus vertex c_i and three connector vertices v_i^1, v_i^2, v_i^3 such that $N(v_i^1) = \{b_i, c_i, b_{i+1}\}$, $N(v_i^2) = \{b_{i+1}, c_i, b_f^*\}$, and $N(v_i^1) = \{b_f^*, c_i, b_i\}$. Figure 9.6 shows an example.

The graph G^+ is obtained from G by quadrangulating away every face f with cardinality > 4 including, if necessary, the outer face. The following properties of the quadrangulation G^+ of G are obvious:

- G^+ is planar and has $O(n)$ vertices.
- All diamonds of G^+ are diamonds of G.

Note that the outer face of G^+ has cardinality 4. If the outer face of G has cardinality > 4 this is an additional diamond of G^+. We ignore this diamond and the condition imposed by it on good partitions of G^+.

In addition we have the following important lemma:

Lemma 9.8. *Let G be a plane bus graph, and let G^+ be its quadrangulation. Then G has a good partition iff G^+ has a good partition.*

Proof. The three defining properties (P1), (P2), and (P3) are stable under taking induced subgraphs. Hence, a good partition of G^+ immediately yields a good partition of G.

Now assume that G has a good partition. We aim for a partition of the bus vertices of G^+ that extends the given partition of the bus vertices of G. Since all bus vertices of degree 4 and all diamonds of G^+ already belong to G we do not have to care of (P2) and (P3). The following rules define the labels for the new bus vertices

- Label all central bus vertices b_f^* with V.
- If b_i and b_{i+1} are both labeled H, then the label of c_i is defined to be V. Otherwise the label of c_i is H.

It is straightforward to check that (P1) is fulfilled for all new connector vertices, i.e., the construction yields a good partition of G^+.

If vertices have been added to the outer face f^* in the quadrangulation process, then we can choose the outer face of G^+ such that it contains $b_{f^*}^*$ and both bus vertices of the (new) outer face are labeled V. This change in the outer face does not affect the plane embedding of G. These considerations imply that when looking for a good partition of G^+ we do not fail because of the condition implied by the diamond defined by the outer face of G^+ if this was not already a diamond of G. □

Theorem 9.9. *Let G be a plane bus graph. A good partition for G can be computed in $O(n^{3/2})$ time if it exists.*

Proof. By Lemma 9.8 it suffices to test if the quadrangulation G^+ of $G = (V \cup B, E)$ has a good partition. Hence we first compute G^+ in linear time. Simple estimates on the basis of Euler's formula show that, in going from G to G^+, at most $O(|B|)$ new vertices have been introduced. Our analysis led to a constant ≤ 13. Hence, G^+ has $n^+ \in O(n)$ vertices and since G^+ is a maximal plane bus graph we can use the $O(n^{3/2})$ time algorithm from Theorem 9.5 to check whether G^+ has a good partition. The algorithm returns a good partition if it exists. Notice that, we have to remove all connector vertices of degree 2 from G^+ to match the requirement of Section 9.2. □

9.5 Embedding Missing – SPQR-Trees

A planar bus graph can have many planar embeddings. Some of them may allow a planar realization of this bus graph and some may not. From Theorem 9.6 we know that a plane embedding has a planar realization if and only if the plane embedding admits a good partition. Therefore, we now face the problem of deciding whether a planar bus graph has a planar embedding that admits a good partition.

If the input graph has cut vertices, we look at the blocks separately. In Section 9.6.2 we give the details on how to merge solutions for the blocks. The main problem is the problem for the blocks:

Problem 1. Given a planar biconnected bus graph, find a planar embedding that admits a good partition if such an embedding exists.

To tackle this problem, we use SPQR-trees, a tool developed by Di Battista and Tamassia [51, 52] to describe all combinatorially different planar embeddings of a biconnected graph in one structure. Another important reference about SPQR-trees is [101]. As sources about SPQR-trees we also used the Wikipedia article and a brief description of Eppstein [69].

An SPQR-tree for a graph G is a tree $\mathcal{T}$ in which each node $n \in \mathcal{T}$ represents a graph G_n. The vertex set of G_n is a subset of the vertices of G. Some edges of G_n are labeled as *virtual* the others are *real*. Removing the virtual edges from G_n yields a subgraph of G and each edge of G appears as a real edge in exactly one of the graphs G_n. If (n, n') is an edge of the SPQR-tree, then G_n and $G_{n'}$ share exactly one edge which is virtual in both, this virtual edge is associated with the tree edge. Each virtual edge of G_n is associated with exactly one tree edge. The given graph G can, hence, be reconstructed by repeatedly taking the union of graphs G_n and $G_{n'}$ belonging to adjacent tree-nodes and deleting the virtual edge associated with the tree-edge (n, n').

The nodes of an SPQR-tree have three types[2]:

[**S**] If n is a S-node, then G_n is a cycle of length at least three (the S represents "series").

[**P**] If n is a P-node, then G_n is a multigraph with two vertices and three or more edges (the P represents "parallel").

[**R**] If n is a R-node, then G_n is a 3-connected graph with more than three vertices (the R represents "rigid").

If we require that no two S-nodes and no two P-nodes are adjacent in $\mathcal{T}$, then the SPQR-tree of G is unique. Moreover the size of $\mathcal{T}$ is linear in the size of G. Algorithmically the SPQR-tree of a graph can be constructed in linear time.

[2]Some descriptions also use type Q nodes, so that SPQR represents the types.

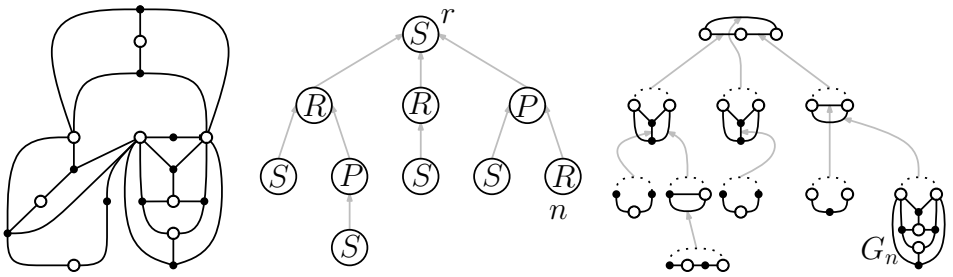


FIGURE 9.7: A plane bus graph G with a rooted SPQR-tree $\mathcal{T}$ and the associated graphs for each node. In $\mathcal{T}$ one node r is designated to be the root. The associated graph G_n of each node $n \neq r$ is drawn with one of its feasible embeddings, while the virtual up-edge of n is the dotted edge.

We continue with some definitions for rooted SPQR-trees. A *rooted SPQR-tree* is an SPQR-tree with a designated root node r. We denote such a tree by $\mathcal{T}_r$. Since every node $n \neq r$ has a unique parent n^+ with respect to the root r, there is also a special *virtual* edge in G_n. This is the virtual edge associated with (n, n^+). In the sequel we will refer to this virtual edge as *the virtual up-edge* of G_n. The two vertices of the virtual up-edge are the *poles* of G_n. An example of a rooted SPQR-tree for a planar bus graph is shown in Figure 9.7.

For $n \neq r$ we orient the virtual up-edge of G_n arbitrarily and consider the set $G_n^1, \ldots, G_n^k$ of all embeddings of G_n that have the outer face to the left of the oriented virtual up-edge. This set will be called the *set of feasible embeddings*. For S-nodes we have $k = 1$, for R-nodes $k = 2$, and for a P-node with $l + 1$ parallel edges we have $k = l!$.

Lemma 9.10. *Let G be a biconnected plane graph with SPQR-tree $\mathcal{T}$. Consider the induced plane embeddings of the graphs G_n represented by the nodes of $\mathcal{T}$. All but at most one of these plane graphs G_n have a virtual edge on the outer face.*

Proof. Each of the plane graphs G_n has an outer face. There is at most one node r such that the outer face of G_r contains no vertex from the set $V(G) \setminus V(G_r)$. Now let n be a node with $n \neq r$ and consider a vertex v_r in the outer face. In G there are two disjoint paths from v_r to G_n. These paths can be traced through $\mathcal{T}$ where they both reach n across the same tree edge (n', n). The virtual edge associated with (n', n) is on the outer face of G_n. □

A planar graph G may have exponentially many embeddings. Lemma 9.10 will be useful to bound the set of embeddings that have to be explored by the algorithm. Indeed, if we correctly guess the root, then we can restrict attention to the set of feasible embeddings of each G_n.

In the next section we make some simplifying assumptions and describe the key ideas for the algorithm in the simplified setting.

9.6 The Algorithm

In the first part of this section we describe the *basic algorithm* for solving Problem 1 with a simplifying assumptions about the input graph G and its SPQR-tree $\mathcal{T}$.

- At every node n of $\mathcal{T}$ all virtual edges of G_n are only incident to bus vertices.

In Section 9.6.1 we add the details for dealing with virtual edges incident to connector vertices. Finally, in Section 9.6.2 we remove the assumption the input graph is biconnected.

We are going to work with rooted SPQR-trees but since we don't know which root is a good one we initialize the set R^* of potential roots with the set of all nodes of $\mathcal{T}$. Then we guess a root by picking arbitrarily an $r \in R^*$.

The tree $\mathcal{T}_r$ will be traversed bottom up. In the traversal each node has to be processed. However, n can only be processed if all its child nodes have been processed before. To control this condition we use the notion of *eligibility*. At the very beginning all leave nodes of $\mathcal{T}_r$ are eligible. During execution of the algorithm a node n of $\mathcal{T}_r$ becomes eligible as soon as all its children have been processed.

When a node n is being processed we want to decide whether the bus graph G_n^* represented by the subtree $\mathcal{T}_r(n)$ of $\mathcal{T}_r$ rooted at n admits a good partition. In the affirmative case we have to decide in addition whether there is a good partition with the two poles *in the same class* and whether there is a good partition with the two poles *in different classes*. The information is then passed on from n to the father node n^+ as a letter σ_n from the set $\{s, d, w\}$ where the meaning is $s =$*same*, $d =$*different*, and $w =$*whatever*.

To decide whether a good partition for G_n^* exists we replace each of the virtual edges of a *feasible embedding* (the definition was given before Lemma 9.10) of G_n by one of the gadgets[3] shown in Figure 9.8. The gadget for the case s is a simple diamond and forces the two golden bus vertices to belong to the same class (Lemma 9.1). The gadget for the case d also contains a diamond but in addition a connector vertex of degree 3 that forces the two golden bus vertices to belong to different classes (property **(P1)**). The gadget for case w allows any assignment to the classes. For the replacement of a virtual edge by gadget d we have orientation options that lead to different graphs, respectively to different embeddings. The next lemma shows that the choices don't matter.

Lemma 9.11. *Let G_n^+ be obtained from G_n by replacing virtual edges by gadgets of the three types shown in Figure 9.8. Either the cycle bounding the outer face*

[3]Note that some of the gadgets are multisubgraphs, i.e., there are two connector vertices with the same adjacency. This, however, can easily be resolved.

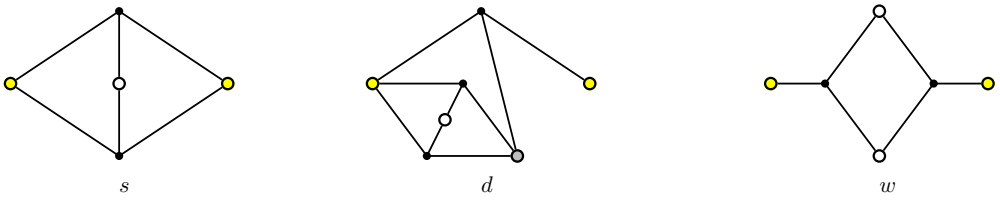


FIGURE 9.8: The gadget replacing the virtual edges of G_n. The yellow bus vertices are the vertices of the virtual edge.

of G_n is a diamond or every diamond of G_n^+ is the diamond of a gadget of type s or d. In particular the existence of a good partition for G_n^+ is independent of the orientation of d gadgets.

Proof. Suppose the cycle bounding the outer face of G_n is a diamond, as shown in Figure 9.7, then the existence of a good partition for G_n^+ follows directly by Theorem 9.9. Suppose now G_n is diamond free. A diamond with one bus vertex a white vertex of a gadget and the other outside of the gadget is impossible. Hence all diamonds of G_n^+ are diamond of s gadgets or d gadgets. The statement about the existence of a good partition follows directly from properties (P1), (P2), and (P3), i.e., from the definition. □

Processing an eligible node n:

(1) Choose a feasible embedding of G_n.

(2) Replace the virtual edges that connect to the children of n by the gadgets corresponding to their letter.

(3) Run the algorithm to compute a good partition twice: first with the virtual up-edge of G_n, i.e, the virtual edge connecting to the father, replaced by the s gadget and secondly replaced by the d gadget.

(4) If successful, pass σ_n to the father, otherwise call for a new root.

We now come to a more detailed look at the parts of this algorithm. Depending on the type of the node n in the SPQR-tree the number of feasible embeddings of G_n varies.

In the case of an S node there is no choice. Actually the letter that has to be passed to the father of an S node n can be computed directly from the letters obtained from the children. If one of the children submitted a w, then $\sigma_n = w$. Otherwise it depends on the parity of the number of children that submitted a d. If this number is even, then $\sigma_n = s$, otherwise $\sigma_n = d$.

In the case of an R node we have two feasible embeddings G_n^1 and G_n^2. Figure 9.9 indicates how the two feasible embeddings are related. With respect

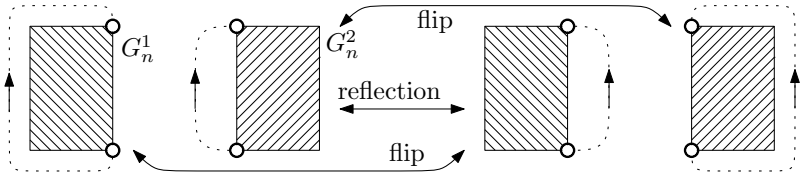


FIGURE 9.9: Four possible embeddings for G_n when n is an R node. The two left embeddings G_n^1, G_n^2 have the unbounded face to the left of the directed virtual edge, i.e., they are the feasible embeddings.

to properties (P1) and (P2) instances G_n^1 and G_n^2 are equivalent. Since we know from Lemma 9.11 that either G_n^1 and G_n^2 both have an identical global diamond, or all the diamonds of G_n^1 and G_n^2 are local to the gadgets, they are also equivalent with respect to (P3). In other words G_n^1 and G_n^2 admit the same good partitions.

In the case of a P node we may have many feasible embeddings. The answer to the question whether there is a good partition, however, is the same for all of them. It only depends on the set of letters passed from the children. Let $\Sigma_n = \{\sigma_{n'} : n' \text{ child of } n\}$. If $\{s, d\} \subseteq \Sigma_n$, then there is no good partition. If $\Sigma_n \subseteq \{s, w\}$, then the graph $G_{n'}^*$ of each child n' has a good partition where the poles are labeled the same, these partitions yield a good partition for G_n^* where the poles are labeled the same. If $\Sigma_n \subseteq \{d, w\}$, each child has a good partition where the poles are labeled differently, these partitions yield a good partition for G_n^* where the poles are labeled differently. Hence, for P nodes we can skip the run of the procedure with gadgets replacing the virtual edges and directly compute σ_n in the case where $\{s, d\} \not\subseteq \Sigma_n$:

$$\sigma_n = \begin{cases} s, & s \in \Sigma_n \\ d, & d \in \Sigma_n \\ w, & \Sigma_n = \{w\} \end{cases}$$

It remains to deal with the "otherwise" in step 4 of the procedure, i.e., with the case where the computation shows that there is no good partition for G_n^* with the chosen feasible embedding. From the discussion above it follows that in this situation there is no good partition for G_n^* with a feasible embedding, i.e., with an embedding that has the virtual up-edge of G_n on the outer face. If there is an embedding of G that admits a good partition, then either n is the root or by Lemma 9.10 the virtual up-edge of node n has to be a different one. This implies that the root node for such an embedding belongs to the subtree $\mathcal{T}_r(n)$ of $\mathcal{T}_r$ rooted at n. Therefore, R^* is redefined to be $R^* \cap V(\mathcal{T}_r(n))$ and a new root r from the new set R^* is chosen.

If R^* becomes empty, then there is no good partition. Otherwise the root r of the tree becomes eligible. Processing the root node is similar to processing any other node. The difference is in the choice of the embedding. In the case of

an S node there is only one embedding. In the case of a P node we can argue as before that the embedding doesn't matter. The interesting case is when the root is an R node. An R node is a 3-connected component and by Whitney's theorem specifying the outer face determines the embedding. Therefore, we can afford to check independently for each embedding.

We have completed the description of the algorithm. We now come to the discussion of the running time.

Due to the change of root a node n may have to be processed several times.

Lemma 9.12. *The number of calls to the procedure that processes the nodes is* $\leq 2|V(\mathcal{T})|$.

Proof. Consider a tree edge (n_1, n_2). If the root is on the side of n_2 after processing n_1, the information σ_{n_1} is passed to n_2. If the root is on the side of n_1 after processing n_2, the information σ_{n_2} is passed to n_1. This is all the exchange along this edge. Since each processing of a node is followed by passing information over a tree edge the number of calls to the procedure that processes the nodes is bounded by twice the number of edges of $\mathcal{T}$. □

If the input graph has n vertices, then in linear time we get an SPQR-tree $\mathcal{T}$ whose size is linear in n. Processing a node can be done in $O(n^{3/2})$ (Theorem 9.9). The root has at most $2n$ faces, hence processing the root be done in $O(n^{5/2})$. Actually we may have to test more that one root but if $f_1, f_2, \ldots, f_k$ are the numbers of faces of k roots that are checked, then still $\sum f_i \in O(n)$.

Proposition 9.13. *If a given planar biconnected bus graph G has the property that all the poles in the SPQR-tree are bus vertices, then we can solve Problem 1, i.e., decide whether G admits a good partition in $O(n^{5/2})$ time.*

9.6.1 Connector Vertices as Poles

We now come to the details for dealing with virtual edges incident to connector vertices, i.e., with situations where the poles of a tree node n are connector vertices. The neighbors of such a connector vertex v in G are called *v-exposed*, note that exposed vertices are bus vertices.

Let (n, n') be an edge in the SPQR-tree $\mathcal{T}$ and let (v, w) be the corresponding virtual edge. We assume that v is a connector vertex and if w is also a connector vertex then $\deg(w) \leq \deg(v)$.

There are several cases that are treated differently. First we consider connector poles of degree three.

The degree of v is 3. There are several subcases.

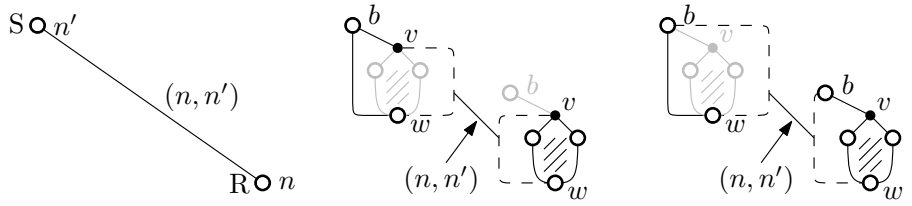


FIGURE 9.10: The movement of the virtual edge when n is an R-node and n' an S-node.

Neither n nor n' is a P-node. The types of one of n and n' has to be R. Assuming that n is of type R we know that G_n contains two v-exposed bus vertices. Let b be the exposed neighbor of v in $G_{n'}$. Since $\{b, w\}$ is a separating set of size two we can conclude that n' is a node of type S and (v, b) is one of the edges of the cycle $G_{n'}$.

We deal with this case by moving the edge (v, b) from $G_{n'}$ to G_n so that the new virtual edge corresponding to the tree edge (n, n') is (b, w), cf. Figure 9.10. After that movement we treat n, n' as R-, S-node in the base algorithm, respectively, since their new poles with respect to (n, n') are only bus vertices.

Node n is a P-node. In $G_{n'}$ there is only one v-exposed vertex. If there were two, then the third exposed neighbor b and the separator $\{b, w\}$ would reveal that n is of type S. Hence, n has three tree neighbors n_1, n_2, and n_3. Let b_i be the neighbor of v in n_i. Since $\{b_i, w\}$ is separating each n_i is of type S.

Let b_i' be the bus vertex closest to w on G_{n_i}, i.e, either $b_i' = w$ or it is the unique w-exposed neighbor (recall $\deg(w) \leq \deg(v)$). The relevant information needed from G_{n_i} is the classifying letter from $\{s, d, w\}$ for the pair b_i, b_i'. This is obtained by running the procedure for eligible nodes.

When n becomes eligible and n is the root, then we can use one of the standard gadgets from Figure 9.8 for each child together with v and w to check the existence of a good partition (this can even be decided just on the basis of the three letters).

When n becomes eligible and n is not the root. Then we skip n, however, when it comes to processing the parent n_1 (a node of type S) we use two of the standard gadgets, one for each of n_2 and n_3 together with v and w. Figure 9.11 indicates how this is done.

The degree of v is 4. The critical issue in this case is that we have to care of property (P2) of a good partition, i.e., the cyclic alternation of labels around v.

In the basic case and in the case where connector vertices of degree 3 are poles the existence of a good partition for some embedding of G_n^* could be checked by replacing virtual edges of G_n with some system of simple gadgets

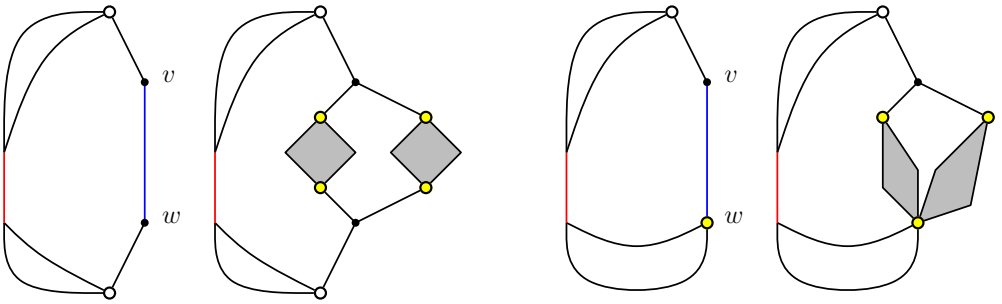


FIGURE 9.11: Replacing the virtual edge in G_{n_1} by two gadgets, two cases. The red edge is the virtual up-edge of n_1.

that represent conditions enforced by the descendant nodes and then check for a good partition of the thus constructed graph G_n^+.

The increase of complexity in the present case can be seen by observing that now we may have to be cautious about the embedding. Here is an example illustrating this: Let w be a bus node, let b_1, b_2, b_3 and b_4 be v-exposed, and assume that each $\{b_i, w\}$ is separating. The graph G_{n_i} has two bus vertices as poles, hence, there is a letter $\sigma_i = \sigma_{n_i}$ encoding conditions on good embeddings. Now suppose that $(\sigma_1, \sigma_2, \sigma_3, \sigma_4) = (s, s, d, d)$, then there is a good partition if and only if the G_{n_i} are arranged in the embedding so that the letters alternate between s and d.

We now go through several subcases depending on the number of v-exposed vertices in G_n. Notice that v (if n is not root) is incident to precisely one virtual edge in G_n, since when considering node n all children must have been processed before.

Node n contains 3 v-exposed vertices. Consider the fourth v-exposed vertex b together with w to conclude that n' is a node of type S and (v, b) is one of the edges of the cycle $G_{n'}$. We deal with this case by moving the edge (v, b) from $G_{n'}$ to G_n so that the new virtual edge corresponding to the tree edge (n, n') is (b, w), cf Figure 9.10.

Node n contains 1 v-exposed vertex. Let b be this v-exposed vertex together with w to conclude that n is a node of type S and (v, b) is one of the edges of the cycle G_n. We deal with this case by moving the edge (v, b) from G_n to $G_{n'}$ so that the new virtual edge corresponding to the tree edge (n, n') is (b, w), cf Figure 9.10 with interchanged roles of n and n'.

Node n contains 2 v-exposed vertices. We consider two subssubcases depending on whether w is a bus vertex or not.

Node n contains the bus vertex w. Suppose that G_n contains two v-exposed vertices b_1, b_2. In this case it is necessary that there is a good partition of G_n^* where b_1 and b_2 have different labels. This can be checked when

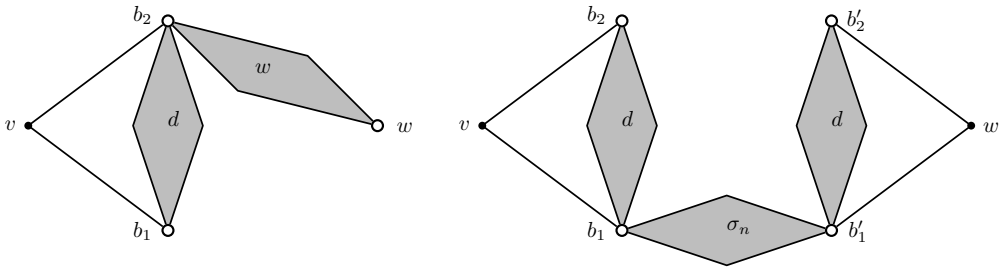


FIGURE 9.12: Replacement for a node n with 2 v-exposed vertices and 2 w-exposed vertices.

processing node n by inserting a d gadget between b_1 and b_2. If such a test fails, then we have to try with a new root from the subtree rooted at n.

Otherwise note that G_n^* can be reflected, therefore the alternation condition (P2) for v can be satisfied as soon as the remaining two v-exposed vertices are labeled differently. In the parent node of n it is then enough to replace the virtual edge (v, w) with the gadget shown in the left part of Figure 9.12. The w gadget encodes the fact that we can use reflections.

Node n contains 2 w-exposed vertices. Let b_1, b_2 be the v-exposed vertices and b_1', b_2' be the w-exposed vertices of G_n. Insert d gadgets between the pairs b_1, b_2 and b_1', b_2'. Then check the existence of a good partition twice, first with b_1, b_1' connected by an s gadget and then by a d gadget. If both fail we have to try with a new root, otherwise let $\sigma_n \in \{s, d, w\}$ be the outcome of the tests.

In the parent node of n it is then enough to replace the virtual edge (v, w) with the gadget shown in the right part of Figure 9.12. Since the degree of v and w remains unaffected the alternation condition (P2) is taken care of.

When n becomes eligible and n is the root, then we can use one of the standard gadgets from Figure 9.8 for each child together with v and w to check the existence of a good partition.

In the above cases for n with 3 v-exposed vertices and with 1 v-exposed vertex, the insertion of gadgets was not necessary, while in the case of 2 v-exposed vertices, it was necessary. Nevertheless all cases disregard the type of n. If n is of type S or R, we treat n as in the base algorithm, while if n is of type P, we have to take care of (P2). Therefore we consider this as last case.

Node n is a P-node. Consider the vertex v of degree four, which was a pole before moving the virtual edge.

Suppose first that n is the root. Then there are up to four components providing information that can be captured by one of the gadgets from Figure 9.8 or 9.12 and thus up to six possible orders of these components cyclically arranged around v in G_n. It is an easy combinatorial sorting problem to find a good partition if it exists.

If n is an inner node then there are up to three children providing information that can be captured by appropriate gadgets. Again the provided information from the children can be captured by one of the gadgets from Figure 9.8 or 9.12. We skip considering n and move the gadgets to the parent n' of n, where we have again up to six possible orders of these components cyclically arranged around v. In $G_{n'}$ it is the same combinatorial sorting problem (as if n were root) to find a good partition if it exists.

All in all we have to check a constant number of sortings, i.e., different embeddings, and the result can be encoded by a gadget in the parent node.

Lemma 9.14. *Let G_n^* be the bus graph represented by $\mathcal{T}_r(n)$ computed according to the above cases. If G admits a plane embedding with good partition, then G_n^* has a good partition.*

Proof. We consider a connector vertex v and distinguish the cases whether v is a pole or not. In the latter case v is a vertex of G_n for a unique node n in $\mathcal{T}$ with no incident virtual edge. Thus properties (P1), (P2), (P3) are satisfied by Theorem 9.9 as black-box. Consider the pole vertex v of degree 3. After the movement of the virtual edge v becomes a vertex in a unique G_n without incident virtual edges and thus again Theorem 9.9 ensures properties (P1), (P3), while (P2) is irrelevant for v. Consider the pole vertex v of degree 4. We aim at ensuring property (P2), since (P1) is irrelevant and (P3) is implicitly ensured, as soon as (P2) is ensured and the graph contains no oversaturated diamonds. Since v is a pole it is in at least two components.

Suppose v is in precisely two components of vertices n', n in $\mathcal{T}$. Then n', n are adjacent and of type S,R or R,S or R,R. The first case is covered by "n has 3 v-exposed vertices" (and n' has 1 v-exposed vertex), the second case is covered by "n has 1 v-exposed vertex" (and n' has 3 v-exposed vertices), and the last case is covered by "n has 2 v-exposed vertices" (the 2 v-exposed vertices in n' are fixed due to the embedding and (P2) is ensured due to the black-box after inserting a gadget for the 2 v-exposed vertices of n).

Suppose v is in more than two components of vertices in $\mathcal{T}$. Then there is precisely one P-node n with pole v. The adjacent nodes $n_1, \ldots, n_k$ are of type S or R. We will distinguish whether $k = 4$ or $k = 3$ (notice that $k \geq 3$ because of the definition of P-node and $k \leq 4$ because of the degree of v).

If $k = 4$, then $n_1, \ldots, n_4$ all have the same type S. This case is covered, when 3 of the nodes were processed and n becomes eligible, i.e., then n contains 3 v-exposed vertices and the fourth v-exposed vertex is moved to n during the movement of the virtual edge. Still n is a P-node and (P2) is ensured after finding a feasible combinatorial sorting of $n_1, \ldots, n_4$ in the case "Node n is a P-node".

If $k = 3$, then w.l.o.g. n_1 is an R-node and n_2, n_3 are S-nodes.

Assume n has no parent, i.e., n is the root, then the d gadget transfered from n_1 together with the remaining 2 v-exposed vertices from n_2, n_3 transfered when moving the virtual edge, are arranged in "Node n is a P-node" when solving the combinatorial problem for the feasible sorting of the v-exposed vertices around v.

Assume n has a parent n_i; we consider the case $i = 1$ and $i \neq 1$. If $i = 1$, i.e., n_1 is the parent of n, then n_2, n_3 were processed and n becomes eligible with 2 v-exposed vertices transfered when moving the virtual edge. These 2 v-exposed vertices are linked by a d gadget from Figure 9.12 which is directly moved to n_1. In n_1 (P2) is ensured by the black-box.

Otherwise if $i \neq 1$, then n_1 and one of n_2, n_3 were processed and n becomes eligible with 3 v-exposed vertices, while two of them (those from n_1) are linked by a d gadget from Figure 9.12. Since the parent n_i is an S-node, its v-exposed vertex is moved to n due to the movement of the virtual edge. It is again just a combinatorial problem to find a feasible sorting of the v-exposed vertices ensuring (P2), which is covered in "Node n is a P-node". In the end we considered all possible cases for a connector vertex pole. □

Proposition 9.15. *Problem 1 can be solved in $O(n^{5/2})$ time, i.e., a good partition for a planar biconnected bus graph can be computed efficiently if it exists.*

9.6.2 Simply Connected Inputs

Now suppose that the input graph G fails to be biconnected.

Let G_1 and G_2 be biconnected bus graphs possibly with connector vertices of degree one. For vertices a_1 and a_2 of the respective graphs of the same type let $G_1 \,{}_{a_1}\!\oplus_{a_2} G_2$ denote the graph obtained by gluing G_1 and G_2 together by identifying a_1 and a_2.

Lemma 9.16. *If G_1 and G_2 both have a good partition and if a_1, a_2 are bus vertices or a_1, a_2 are connector vertices with $\deg_{G_1}(a_1) + \deg_{G_2}(a_2) \leq 3$, then the graph $G_1 \,{}_{a_1}\!\oplus_{a_2} G_2$ also has a good partition.*

Proof. Compute good partitions of G_1 and G_2. Either they combine to a good partition of $G_1 \,{}_{a_1}\!\oplus_{a_2} G_2$ or exchanging all labels in G_2 yields a labeling that can be combined with the labeling of G_1. □

The remaining case where $G = G_1 \,{}_{a_1}\!\oplus_{a_2} G_2$ is a bus graph is when a_1, a_2 are connector vertices with $\deg_{G_1}(a_1) + \deg_{G_2}(a_2) = 4$.

If $\deg_{G_1}(a_1) = 2 = \deg_{G_2}(a_2)$, then a good partition of G exists if and only if the neighbors of a_1 in G_1 and the neighbors of a_2 in G_2 are labeled differently. This can be checked by inserting d-gadgets before computing good partitions.

If $\deg_{G_1}(a_1) = 3$ and $\deg_{G_2}(a_2) = 1$, then it is again sufficient to have good partitions for G_1 and G_2. Condition (P1) at a_1 requires that both labels appear in the neighborhood of a_1. Embedding G_2 in the face with the two identical labels allows to satisfy (P2) in the full graph G.

These considerations can be used on the *block-tree* B of G, i.e., on the tree that represents the maximal biconnected components of G.

The results are summarized in the following theorem; a planar embedding will be produced as a byproduct of a good partition if it exists.

Theorem 9.17. *A good partition for a planar bus graph can be computed in* $O(n^{5/2})$ *time if it exists.*

9.7 Non-Planar Bus Graphs

In the previous sections we proved that recognition of planar realizations can be done in polynomial time. We say such a bus graph is in class C_0, which are bus graphs admitting a planar realization. The corresponding realization is referred to as C_0 realization. Now we have characterized the whole C_0 class, which are precisely the planar bus graphs with good partition. A good partition of a planar bus graph forms a sufficient condition for the construction of a C_0 realization due to Theorem 9.17. The necessity of this condition follows by extracting the properties of a good partition and the planarity for the bus graph from a given C_0 realization.

In contrast to the result for C_0 realizations, Ada et al.[3] proved that the decision whether a bus graph admits a non-planar realization is NP-complete. They used non-planarity only in terms of connections. This class of bus graphs admitting a realization where only connections are allowed to cross is referred to as C_1. The corresponding realization is referred to as C_1 realization. Ada et al. didn't characterize this class of bus graphs, but proved the complexity for the decision problem.

Theorem 9.18 ([3])**.** *The decision whether a bus graph is in* C_1 *is NP-complete.*

The are two more non-planarity types we are interested in: a bus graph is in class C_2 if it admits a realization where connections may cross each other or connections may cross bus segments (referred to as C_2 realization), while a bus graph is in class C_3 if it admits a realization where any crossing is allowed (referred to as C_3 realization). In order to characterize the bus graphs in C_3, we combine property (P1) and (P2) of a good partition as follows:

(P1') Every connector vertex of degree ≥ 3 has at most 2 neighbors in the same partition class.

The bus graphs in C_3 are characterized by all bus graphs admitting a partition satisfying property (P1') (short (P1')-partition). The necessity of this condition follows directly by the property of orthogonal drawings, where at most two vertical segments and at most two horizontal segments can enter a vertex. This condition is also sufficient to construct a C_3 realization from a bus graph G by the following argument. Assume G admits a (P1')-partition. Place all k V-labeled buses and all l H-labeled buses on a $l \times k$ grid. Each vertical (horizontal) bus crosses every horizontal (vertical) bus. The connector vertices can be placed inside the region bounded by the buses representing its neighbors.

The decision whether a bus graph is in C_3 is NP-complete[4] since the question whether a bus graph admits a (P1')-partition is equivalent to the question whether a 3-uniform hypergraph has a 2-coloring [61, 135]. Consider a bus graph $G = (B \cup V, E)$ representing the hypergraph $H = (V, B)$. We can skip the connector vertices of degree ≤ 2 since they have no influence on a (P1')-partition. Also we substitute every connector vertex v of degree 4 with neighbors $N(v)$ by four connector vertices of degree 3, each of them connected to one of the four triple of $N(v)$. Now the modified bus graph $G' = (B \cup V', E')$ is 3-regular in V' and the dual $H'^* = (B, V')$ of the represented hypergraph $H' = (V', B)$ is 3-uniform[5]. A coloring of a hypergraph is a coloring of the hypervertices such that no hyperedge is monochromatic. A 2-coloring of H'^* is a coloring of B with V and H such that no hyperedge in V' is monochromatic. This is precisely a (P1')-partition for G' and, hence, also for G.

Theorem 9.19. *The decision whether a bus graph is in C_3 is NP-complete.*

For some bus graphs in C_2 we know an efficient way to produce a C_2 realization. We consider the concept of the reduced bus graph $R = (B, E_R)$ of $G = (V \cup B, E)$, where $(b, b') \in E_R \subseteq B \times B$ if and only if there is a $v \in V$ and edges $(b, v), (b', v) \in E$. Assume that the reduced bus graph R has a 3-coloring (B_1, B_2, B_3). It implies that G has $\Delta = 3$. It also implies that G admits a (P1')-partition $(B_1 \cup B_2, B_3)$. From this partition we construct a C_2 realization as follows. We place three blocks on top of each other: the B_1-buses are horizontal in the topmost block, B_3-buses are vertical in the middle block and B_2-buses are horizontal in the bottommost block. Connector vertices can be placed in the region bounded by the buses representing its neighbors.

Nevertheless we neither characterized the bus graph in class C_2, nor we know the complexity of the decision problem. To close the discussion about realizability of bus graph we conjecture the following.

Conjecture 9.20. *The decision whether a bus graph is in C_2 is NP-complete.*

[4] Ada et al. [3] called this problem "Partition-by-Orientation" and gave a proof using gadgets from the C_1 case. We present a simpler argument.

[5] For more details on uniform and regular hypergraphs we refer to [18].

9.8 Summary and Future Work

We have considered the class of planar bus graphs that admit a planar 2-dimensional bus realization and have characterized this class by the existence of a good partition of bus vertices. To test for the existence of a good partition, we have given an $O(n^{5/2})$-time algorithm based on planar matching and SPQR-trees. Given a good partition, the representation can be computed in linear time.

It is still open to characterize the class of graphs that admit realizations, where connections are allowed to cross (C_1-realizations) apart from the knowledge, that the decision is NP-complete. The decision problem and the characterization of the class of graphs admitting a realization, where connections are allowed to cross as well as connections with buses (C_2-realization), remains as open problem. On the positive side we characterized the class of C_3-realizations and uncovered the complexity of its decision problem. Table 9.1 summarizes the results for the realization types with respect to the decision problem and the characterization problem.

	C_0	C_1	C_2	C_3
decision	P	NP	?	NP
characterization	yes^*	?	?	yes^{**}

TABLE 9.1: This table summarizes for which realization type the decision problem and characterization problem is known. NP is short for "NP-complete" and P is short for "polynomial time solvable". yes^* is a placeholder for Definition 9.2 for a good partition, while yes^{**} replaces Property (P1').

Apart from the question if the decision is NP-complete, we could ask here for a given C_j-representation, if there exists a C_i-representation for $i < j$.

Furthermore it is interesting to consider the realizability question in $d > 2$ dimensions. Here we may consider bus graphs, where the connector vertices have degree at most six, respectively eight, regarding a 3-dimensional, respectively 4-dimensional bus realization with vertical and two types of diagonal segments, respectively vertical, horizontal and diagonal segments.

The other direction, i.e., bipartite graphs, where the connector vertices have degree at most two, is considered next.

Chapter 10

Bus Graphs in One Dimension

In this chapter we consider planar bus realizations, where the bus segments will be drawn only horizontally. Thus we are considering the bus graph model as follows: A *bus graph* is a bipartite graph $G = (V \cup B, E)$, where $E \subseteq V \times B$ and $deg(v) \leq 2$ for all $v \in V$. In this section the key question in this model is, whether a planar bus graph admits a planar bus realization.

In order to produce a planar bus realization with only horizontal segments, we may ignore connector vertices of degree one and replace every connector vertex of degree two by an edge connecting its adjacent vertices resulting in a graph G that consists of only bus vertices. If G is planar, then a visibility representation exists by Theorem 2.5, which is precisely a planar bus realization with only horizontal segments. If G is non-planar, then drawing the buses on top of each other leads to a non-planar bus realization. Here an interesting objective might be to reduce the number of crossings or minimizing the sum of edge length for the buses and connections. We will leave this for future work.

Instead we change the setting of our key question to one in which connector vertices are already placed in the plane. Thus finding a feasible placement for the bus segments is the main problem in this section, assuming the existence of a planar bus realization.

Due to the difficulty of this problem we will restrict the bus graph once more by the degree of connector vertices. The *bus graph* we are considering in the remainder of this section is a bipartite graph $G = (V \cup B, E)$, where $E \subseteq V \times B$ and $deg(v) = 1$ for all $v \in V$. The degree restriction on the connector vertices allows the reformulation of our problem as follows. Assume every bus vertex has a color and the connector vertices are colored according to the adjacent bus vertex. Suppose the connector vertices are given as set of colored points in general position[1], namely that no two points have the same x- or y-coordinate.

[1]This is not the standard general position assumption, but the same as in Section 6.

Using this point of view we study whether there is a planar 1-dimensional bus realization such that all points of the same color are orthogonally connected to its bus, called the *bus embeddability problem* (BEP). This changes the point of view from visualizing a hypergraph with fixed vertex positions to visualization of set membership of colored points in the plane. In the sequel of this chapter we assume a bus realization always to be 1-dimensional without mention it explicitly.

In Section 10.1 we formalize the problem by introducing related terminology and basic results on BEP. We solve BEP when the relative order of the buses is prescribed; we also show that BEP is fixed-parameter tractable with respect to the number of colors. In Section 10.2 we formulate an ILP that solves BEP and show some experimental results. In Section 10.3 we restrict BEP (when a bus must be above all its points, or a bus must be either at its topmost or bottommost point) and describe efficient algorithms for these settings. Another restricted version of the problem can be solved using 2-stack pushall sorting. Finally in Section 10.4 we prove that BEP is NP-complete, even for just two points per color, if points may not lie on buses. We summarize the results in Section 10.5 and point out directions for future work. The results are based on [205].

10.1 Definitions and Basic Results

We begin with some definitions. Supose we are given a set of points $\mathcal{P} = \{p_1, \ldots, p_n\}$ and colors $\mathcal{C} = \{c_1, \ldots, c_k\}$ together with a function $f\colon \mathcal{P} \to \mathcal{C}, f(p) = c$. For simplicity, we assume that no two points share a coordinate in the input pointset, although in some illustrations the input points might violate this assumption. The bus embeddability problem (BEP) asks whether there is a planar bus realization with one horizontal bus per color. BEP is a decision problem, but in our descriptions whenever the answer is affirmative we also compute a drawing. We refer to such a drawing as a *solution of BEP*. In the negative case we say that BEP has no solution.

A point p has x-coordinate $x(p)$ and a y-coordinate $y(p)$, as well as a color $f(p)$. In a bus realization we have connections only between a point p and a bus c of the same color, that is, $c = f(p)$. We denote by $f^{-1}(c)$ the set of points with color c. Bus c naturally extends from the x-coordinate $x_l(c) = \min\{x(p)|p \in f^{-1}(c)\}$ of the leftmost point to the x-coordinate $x_r(c) = \max\{x(p)|p \in f^{-1}(c)\}$ of the rightmost point of $f^{-1}(c)$. We call $[x_l(c), x_r(c)]$ the *span* of c, which is predefined by the input points. The y-coordinate of a bus c is denoted by $y(c)$, which is the only parameter to be determined for a solution for BEP.

Note that BEP is trivial when there are at most two colors: it is always possible to place one bus at the top and the other (if it exists) at the bottom of the drawing. Thus in the following we assume $k > 2$. For more than two colors, the

relative order of the buses is important; see Figure 8.3. Suppose the y-order of the buses is prescribed. The next lemma shows that one can check an existence of a solution for BEP respecting the order.

Lemma 10.1. *There is a $O(n \log n)$-time algorithm that, given an order of buses, tests whether there exists a solution for BEP respecting the order.*

Proof. Suppose we are given an order $c_1 < \cdots < c_k$ of the buses from bottom to top. We use discrete values for the y-coordinates increasing from bottom to top, where a unit is $1/n$ of the y-distance of two consecutive points. We first present a simpler $O(n^2)$-time algorithm, and then describe how to speed it up.

Recall that the span of every bus is defined by an input point set; hence, we only show how to choose y-coordinates of the buses. The first bus, c_1, is placed at y-coordinate $y(c_1) = 0$, and all the points of color c_1 are connected to the bus. Assume that bus c_{i-1} is placed at y-coordinate $y(c_{i-1})$ and is connected to all its points. We place c_i at $y(c_i) = y(c_{i-1}) + 1$ unit and check if the bus crosses a previously drawn (vertical) segment. If it does cross a segment, then we shift c_i one unit upwards by increasing $y(c_i)$ and repeat the procedure. Once the bus is placed without crossings, we connect it to the corresponding points. Consider the vertical segment of a point p of color c_i. It is easy to see that if $y(p) \geq y(c_i)$, then the segment cannot cross a previously placed bus c_j for $j < i$. If $y(p) < y(c_i)$ and the vertical segment crosses a bus, then such a crossing is unavoidable in any solution respecting the given order. Hence, we may stop the algorithm reporting that no solution exists. Otherwise, we proceed with the next color.

The above algorithm can easily be implemented in quadratic time. However, we can do better using the following observation: Every bus is placed at its bottommost "valid" y-coordinate, that is, the one that does not produce crossings with previously placed buses. To find such a y-coordinate efficiently for each color, we store all points of the already processed colors in a data structure D that supports range operations such as "extracting minimum/maximum on a given range". For every color c_i, we extract a point with the maximum y-coordinate in the range corresponding to the span of c_i. The bus of c_i is placed at the maximum of the extracted y-coordinate and the y-coordinate of bus $y(c_{i-1})$. Then all the points of color c_i are added to D. A balanced tree (e.g., a segment tree) providing logarithmic complexity for insert and extract operations is sufficient for our needs. □

In general the correct order of the buses for a planar bus realization is not known. One can apply Lemma 10.1 for each of the $k!$ possible bus orders, which yields an $\widetilde{O}(k!)$-time[2] algorithm for BEP. In the following algorithm, we improve the running time.

[2] $\widetilde{O}$ hides polynomial factors.

Lemma 10.2. *There is a $\widetilde{O}(2^k)$-time algorithm for BEP.*

Proof. We solve a given instance of BEP using dynamic programming. Let us call a *state* a pair (h, B), where $0 \leq h \leq n+1$ is an integer and B is a subset of $\mathcal{C} = \{c_1, \ldots, c_k\}$. By a solution for a state (h, B) we mean a (planar) bus realization consisting of buses for every color $c \in B$ such that the topmost bus has y-coordinate h. If such a solution exists, we write $F(h, B) = \text{true}$, and otherwise $F(h, B) = \text{false}$. It is easy to see that a solution for the original BEP problem exists if and only if $F(h, \mathcal{C}) = \text{true}$ for some $0 \leq h \leq n+1$.

We reduce the problem to solving it for "smaller" states, that is, the states with fewer elements in B. As a base case, we set $F(h, B) = \text{true}$ for all $0 \leq h \leq n+1$ and $|B| = 1$. To compute a value for a state $F(h, B)$ with $|B| > 1$, we consider a color $c^* \in B$. Let $h^* = \max\{y(p) | f(p) \in B \setminus \{c^*\} \text{ and } x_l(c^*) \leq x(p) \leq x_r(c^*)\}$, that is, the largest (topmost) y-coordinate of a point of color $B \setminus \{c^*\}$ laying in the span of c^*. It follows from the proof of Lemma 10.1 that the bus for c^* should be placed at y-coordinate h^*. Thus, $F(h, B)$ is set to true if (a) $h \geq h^*$ and (b) there exists a solution for a state $(h', B \setminus \{c^*\})$ for some $h' < h$. We stress here that in order to compute $F(h, B)$, one needs to consider every color of B as a potential c^*.

There are $n2^k$ different states, and a computation for a single state clearly takes a polynomial number of steps. □

The above result shows that the BEP problem is fixed-parameter tractable with respect to k, that is, it can be efficiently solved for a small number of buses. Note that in Section 10.4 we prove that BEP is NP-complete; hence, it is unlikely that a polynomial-time (in terms of k) algorithm exists.

10.2 An ILP

In this section we present an integer linear programming (ILP) formulation for BEP that produces a planar bus realization if one exists. Here we also minimize the amount of ink, which is the sum of all segment lengths.

Lemma 10.3. *A solution for BEP can be computed by an ILP.*

Proof. In a preprocessing step we compute the span of every bus $c \in \mathcal{C}$. It remains to compute an y-coordinate variable $y(c)$ for every bus c. To this end, we introduce a planarity constraint for every point $p \in \mathcal{P}$ within the span of bus c having a different color. These pairs (p, c) with $c \neq f(p)$ and with $x(p)$ in the span of c are called *conflicting*. Conflicting pairs (p, c) are stored in a matrix $\mathcal{J}$ and induce the constraint $(y(p) < y(c) \text{ and } y(f(p)) < y(c))$ or $(y(p) > y(c) \text{ and } y(f(p)) > y(c))$. The matrix $\mathcal{J}$ can be computed in $O(kn)$ time, where $n = |\mathcal{P}|$

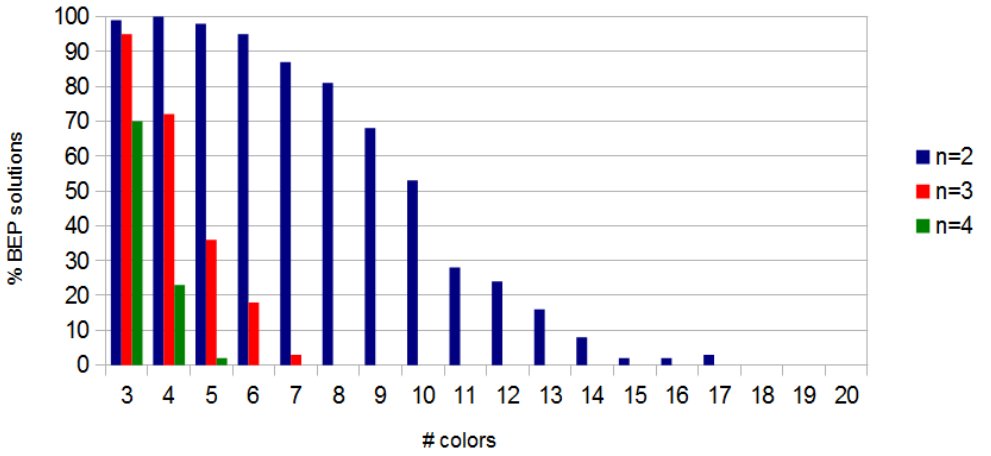


FIGURE 10.1: The percentage of solutions for BEP for a random point set of size kn with $n = 2, 3, 4$ points per color out of $k = 3, \ldots, 20$ colors.

and $k = |\mathcal{C}|$. In order to minimize the amount of ink, we sum up the lengths of all connections and ignore the lengths of the buses, as those are determined by the input. Since an absolute value requires one more variable and 3 constraints for every point, and since the "or" also requires an additional variable and 3 constraints for every conflicting pair, the final ILP has $n + k + 2|\mathcal{J}|$ variables and $3n + k + 6|\mathcal{J}|$ constraints.

$$\begin{aligned} \min \quad & \sum_{c \in \mathcal{C}} \sum_{f(p)=c} |y(c) - y(p)| \\ s.t. \quad & (y(p) < y(c) \text{ or } y(f(p)) > y(c)) \text{ for each } (p, c) \in \mathcal{J} \\ & (y(p) > y(c) \text{ or } y(f(p)) < y(c)) \text{ for each } (p, c) \in \mathcal{J} \\ & 0 \leq y(c) \leq \max_{p \in \mathcal{P}} \{y(p)\} + 1 \end{aligned}$$

□

In order to get a feeling about the probability that a point set admits a solution of BEP, we ran a small experiment with the ILP, implemented with the Gurobi solver [100]. We considered point sets with $k = 3, \ldots, 20$ colors and with $n = 2, 3, 4$ points per color. We randomly placed the points on a 1024×768 area. For each pair (n, k) we counted the number of BEP solutions out of 100 instances; see Figure 10.1.

For a fixed number of points n, the number of solutions for BEP decreases with increasing number of colors k. It decreases faster the larger n is. On the other hand, for a fixed number of colors k, the number of solutions for BEP also decreases with increasing number of points n. Thus, even studying two points per color promises to be sufficiently interesting. Thus, as base case for further analysis, we initially consider two points per color, before dealing with the general case.

FIGURE 10.2: Illustration of (a) $\sqcap$-bus, (b) $\sqcup$-bus, (c) Γ-bus, and (d) L-bus.

10.3 Efficiently Solvable Variants

In this section we consider three variants of BEP that can be solved in polynomial time. A bus c is called *top* (resp., *bottom*) if all of its points are below (resp., above) the bus, that is, $y(c) \geq y(p)$ (resp., $y(c) \leq y(p)$) for all $p \in f^{-1}(c)$. We distinguish between buses that are above (below) of their points and buses that pass through one of their points. A top bus is a $\sqcap$-*bus* if $y(c) > y(p)$ for all $p \in f^{-1}(c)$ (Figure 10.2(a)), while it is a Γ-*bus* if $y(c) = y(p)$ for a point p with $y(p) = \max\{y(q) | q \in f^{-1}(c)\}$ (Figure 10.2(c)). Similarly we define a $\sqcup$-*bus* and a L-*bus*; see Figs. 10.2(b) and 10.2(d). A bus, whose type is none of the four types from above, is called a *center bus*.

In Section 10.3.1 we study $\sqcap$-buses and provide an algorithm for a variant of BEP, called $\sqcap$-BEP, which is restricted to the buses of the type. The same algorithm obviously solves the $\sqcup$-BEP variant. Next, in Section 10.3.2, we consider Γ-buses and L-buses. Note that Γ-BEP and L-BEP are trivial since every Γ-bus (resp., L-bus) is uniquely defined by its span and the topmost (bottommost) point. Hence, we investigate and design an efficient algorithm for the (Γ, L)-BEP variant. Finally in Section 10.3.3, we examine the general BEP for a specific point set where all points lie on a diagonal. We show that this variant of the problem is equivalent to a longstanding open problem (resolved very recently) of sorting a permutation with a series of two stacks.

10.3.1 $\sqcap$-BEP

We present an algorithm that decides in polynomial time whether a drawing with $\sqcap$-buses exists for a given input, and constructs such a drawing if one exists.

Theorem 10.4. *There exists an $O(n \log n)$-time algorithm for $\sqcap$-BEP.*

Proof. For ease of presentation, we first assume that the input consists of two points per color, that is, $k = n/2$, and provide a simple quadratic-time implementation. Later we generalize the algorithm and improve the running time. Intuitively, the algorithm sweeps a line from bottom to top and processes the points in increasing order of y-coordinates. At every step, we keep all the vertical segments of the "active" colors (the ones without a bus) in the correct

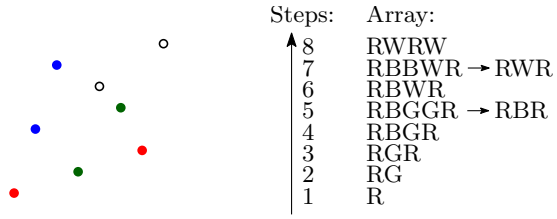


FIGURE 10.3: Running the algorithm from Lemma 10.4 on a given point set with red (R), green (G), blue (B), and white (W) pairs of points. Since the resulting array is not empty, there is no solution for the instance. Notice that removing any of the colors yields an instance with a solution.

left-to-right order. If two vertical segments of the same color are adjacent in the order, then we can draw the corresponding bus and remove the color and its vertical segments. Otherwise, all the active vertical segments have to be "grown" until we reach the next point. It is easy to see that a solution exists if and only if the set of active colors is empty after processing all the points.

More formally, the points are processed one-by-one in increasing order of their y-coordinates. The points are stored in an array sorted by x-coordinate, that is, we have $(p_1, \ldots, p_n)$ with $x(p_1) < \cdots < x(p_n)$. At each iteration, a new point is inserted into the array in the position determined by its x-coordinate. Then the array is modified (or simplified) so that the pairs of points of the same color that are adjacent in the array are removed. That is, if $f(p_i) = f(p_{i+1})$ for some $1 \leq i < n$, then we get a new array $(p_1, \ldots, p_{i-1}, p_{i+2}, \ldots, p_n)$. The simplification is performed as long as the array contains monochromatic adjacent points. After this step the algorithm proceeds with the next point. For every color c, we keep the value $y^*(c)$, which is equal to the y-coordinate $y(p), p \in f^{-1}(c')$ of the point of color c', whose insertion into the array induced the removal of points $f^{-1}(c)$ from the array. If the algorithm ends up with a non-empty array, then we report that no solution exists. Otherwise, the y-coordinate of the resulting bus of color c is $y^*(c) + \varepsilon$, where $\varepsilon > 0$ is sufficiently small to avoid overlaps between the buses. An example of the algorithm is illustrated in Figure 10.3.

Correctness. The correctness follows from the observation that the algorithm chooses the lowest "available" y-coordinate for every bus, that is, the one that does not induce a crossing between the bus and vertical segments of other colors. Indeed, if at any step of the algorithm we get a color pattern $R, \ldots, B, \ldots, R$ in the array formed by red (R) and blue (B) points and the second blue points p has not been processed yet, then clearly in any solution the red vertical segments reach y-coordinate of p. Hence, it is safe to "grow" the segments. On the other hand, if processed points form a color pattern RR (that is, two consecutive points of the same color), then there is a solution connecting the corresponding vertical segments at the current y-coordinate. The two points can be removed from consideration, as they cannot create crossings with the subsequent buses. It is also easy to see that the algorithm minimizes ink of the resulting drawing.

Running time. At every iteration of the algorithm, we need to insert a new point into the sorted array and then run the simplification procedure. Point insertion takes $O(n)$ time and the removal of a pair of points from the array can also be done in $O(n)$ time. Since every pair is removed only once, the total running time is $O(n^2)$.

To get down to $O(n \log n)$ time, we use a balanced binary tree instead of an array to store the points. The tree is sorted by the x-coordinates of the points; hence, insertion/removal of a point takes $O(\log n)$ time. Note that after inserting/removing a point, the only potential candidate pairs for simplification are the point's neighbors that can be found in $O(\log n)$ time. Again, every point is inserted/removed only once; thus, the total running time is $O(n \log n)$.

Finally, we observe that the algorithm can be generalized to handle multiple points per color. To this end, we change the simplification step so that the points are removed only if they form a contiguous subsequence in the array (tree), containing all points of this color. Hence we need to know the number of points for each color, which can be done with a linear-time scan of the input. It is easy to see that the proof of correctness can be appropriately modified and the running time remains the same. □

10.3.2 (Γ, L)-BEP

We present an algorithm that decides in polynomial time whether a drawing with Γ-buses or L-buses exists for a given input, and constructs such a drawing if one exists.

Theorem 10.5. *There exists an $O(n^2)$-time algorithm for (Γ, L)-BEP.*

Proof. The span of every bus is predefined by the input, while for the y-coordinate there are precisely two options. We show that (Γ, L)-BEP can be modeled by 2-SAT, and thus is efficiently solvable. For ease of presentation, we assume that the input consists of two points per color and describe a simple quadratic-time algorithm.

The algorithm creates a variable x_c for every color $c \in \mathcal{C}$. The value of x_c is true if c is a Γ-bus, and it is false if c is a L-bus. After that for every pair of colors c, c', the algorithm creates a clause for the 2-SAT instance when the corresponding buses induce a crossing. Building the clauses with respect to the relative position of points is a straight-forward procedure, where three examples are illustrated in Figure 10.4.

Specifically we create clauses according to the following case analysis. Let $R(c)$ be the smallest enclosing rectangle of the points p_c, q_c of color c. W.l.o.g. we may assume that p_c appears in the left bottom corner, while q_c appears in the right top corner of $R(c)$.

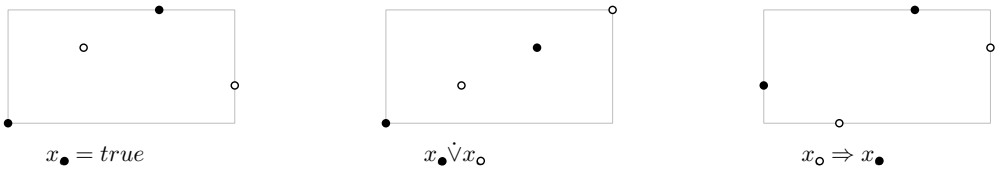


FIGURE 10.4: Three examples for creating clauses for two colors black and white.

We distinguish the cases when

(1) points $p_{c'}, q_{c'}$ are in the top left, bottom right corner of $R(c')$ or whether

(2) points $p_{c'}, q_{c'}$ are in the bottom left, top right corner of $R(c')$.

In each of the two cases we consider the 8 subcases, which are

(a) $R(c')$ intersects only the top boundary of $R(c)$,

(b) $R(c')$ intersects only the bottom boundary of $R(c)$,

(c) $R(c')$ intersects only the right boundary of $R(c)$,

(d) $R(c')$ intersects only the left boundary of $R(c)$,

(e) $R(c')$ contains the top right corner of $R(c)$,

(f) $R(c')$ contains the bottom right corner of $R(c)$,

(g) $R(c')$ contains the top left corner of $R(c)$,

(h) $R(c')$ contains the bottom left corner of $R(c)$.

cases	a	b	c	d	e	f	g	h
1	$x_c = f$	$x_c = t$	$x_c = t$	$x_c = f$	$x_{c'} = t$	$x_c = t$	$x_c = f$	$x_{c'} = f$
2	$x_c = f$	$x_c = t$	$x_c = t$	$x_c = f$	$x_c \dot{\vee} x_{c'}$	$x_{c'} \Rightarrow x_c$	$\overline{x_{c'}} \Rightarrow \overline{x_c}$	$x_c \dot{\vee} x_{c'}$

TABLE 10.1: For each of the cases (a)-(h) from above we build a clause depending on the configuration (1)-(2) from above, where t stands for true and f for false.

Correctness. The correctness follows from the complete case analysis by the rules of Table 10.1.

Running time. We remark that for the $n^2/4$ pairs of colors, we create $O(n^2)$ clauses, each clause in constant time by a case analysis. This results in a 2-SAT instance with k variables $x_c, c \in \mathcal{C}$ and $O(n^2)$ clauses. We solve this instance in linear time [9] and the solution determines the drawing: c is drawn as a Γ-bus, if the value of x_c is *true*, otherwise c is drawn as a L-bus.

We can generalize this idea in a straight-forward manner to the case of more points per color. □

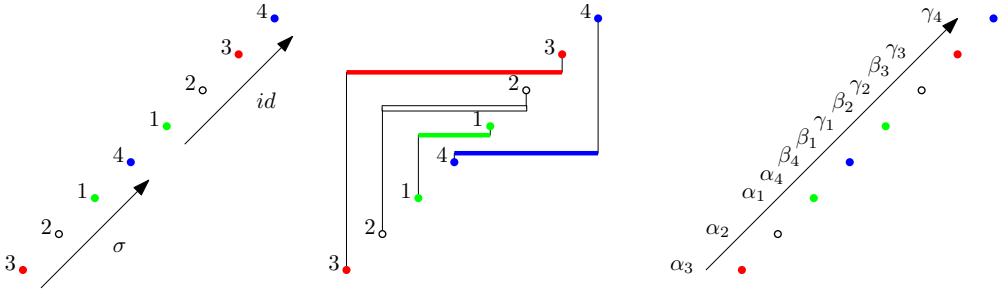


FIGURE 10.5: A diagonal point set with a solution for BEP and the corresponding sorting sequence.

10.3.3 Diagonal BEP

Here we consider a *diagonal* point set in which all points lie on a single diagonal line and there are two points per color. We assume that the point set is *separable*, that is, there is a straight line separating every pair of points having the same color; see Figure 10.5. This specific arrangement can be naturally described in terms of permutations. Assuming that the colors are numbered from 1 to k in the order along the diagonal from bottom to top, the input is described by a permutation $\pi = [\pi(1), \ldots, \pi(k)]$ on $\{1, \ldots, k\}$. We call such an instance of BEP by *diagonal* π-BEP.

It turns out that this variant of BEP is closely related to the well-studied topic of sorting a permutation with stacks introduced by Knuth in the 1960's [122]. We next show that diagonal π-BEP has a solution if and only if π can be sorted with 2 stacks in series. The problem of deciding whether a permutation is sortable with 2 stacks in series is a longstanding open problem and it has been conjectured to be NP-complete several times [23]. Only very recently a polynomial-time algorithm has been developed [152, 153]. It is an indication that even our restricted variant of BEP is highly non-trivial. For more details on stack sorting we refer to [62].

Theorem 10.6. *Diagonal π-BEP has a solution if and only if π is 2-stack pushall sortable. This can be checked in $O(n^2)$ time.*

Proof. First observe that for a diagonal point set with two points per color, a top-bus (bottom-bus) can be transformed to a center-bus. For every color c, there are no points of different color within the span of c above the topmost point of c. Hence, we may only consider center buses in the variant of BEP.

For the 2-stack sorting problem, we are given a permutation π and want to sort the numbers to the identity permutation $[1, \ldots, k]$ with two stacks S_I, S_{II} using the following operations:

- α_i : read the next element i from input π and push it on the first stack S_I;

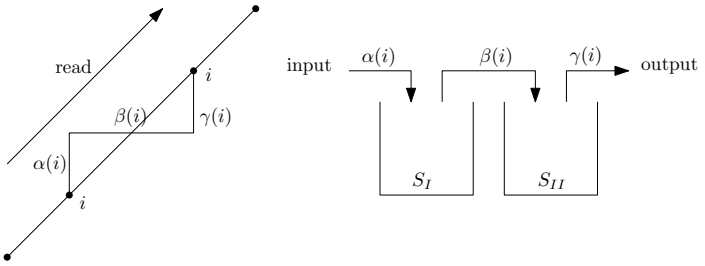


FIGURE 10.6: A correspondence between 2-stack sorting and a planar bus realization.

- β_i : pop the topmost element i from S_I and push it on S_{II};
- γ_i : pop the topmost element i from S_{II} and print it to the output.

To sketch the proof of the equivalence between 2-stack sorting and bus embeddability, we note that the first operation, α_i, corresponds to the left vertical segment of color i, the second one, β_i, is the bus of i, while γ_i corresponds to the right vertical segment of the color; see Figure 10.5 and Figure 10.6. A crossing in the drawing correspond to an "invalid" sorting operation in which either a non-topmost element is moved from S_I to S_{II} (a crossing to the "left" of the diagonal), or a non-topmost element is moved from S_{II} to the output (a crossing to the "right" of the diagonal). Hence, sorting sequences of the operations for π are in one-to-one correspondence with planar bus realization for the point set. Since the point set is separable, all the elements of π will be pushed to S_I before any of the elements is popped to the output. This is called 2-stack *pushall* sorting and is considered next in more detail.

A sequence of operations can be described by a word $w \in \{\alpha, \beta, \gamma\}^{3n}$, where every operation appears n times.

For example $w = \alpha_3\alpha_2\alpha_1\alpha_4\beta_4\beta_1\gamma_1\beta_2\gamma_2\beta_3\gamma_3\gamma_4$ is a sorting word for $\pi_1 = 3214$ to $\pi_2 = 1234$ with two stacks, see Table 10.2.

A word w also encodes the input and output of a sequence by subscripts, when disregarding the subscripts of the β-operation. For example $s(w) = 32141234$ is the sequence of subscripts for w.

We may restrict this sequence of operations to only α- and γ-operations, denoted by $w|\{\alpha, \gamma\}$. We say w is a *pushall word*, if $s(w|\{\alpha, \gamma\}) = \pi_1\pi_2$. The word $w' = \alpha_3\alpha_2\alpha_1\beta_1\gamma_1\alpha_4\beta_4\beta_2\gamma_2\beta_3\gamma_3\gamma_4$ also sorts π_1 to π_2 with two stacks, but w' is not a pushall word, since $s(w') = 32114234 \neq \pi_1\pi_2$.

Now we assume we are given $2n$ points on a diagonal respecting the orders π_1, π_2. We denote by $\pi_1(\pi_2)$ the order of the first (second) appearance of the elements. Every output word w of the 2-stack-pushall-sorting algorithm describes the sorting from π_1 to π_2.

operation	input	S_I	S_{II}	output
	3214			
α_3	214	3		
α_2	14	23		
α_1	4	123		
α_4		4123		
β_4		123	4	
β_1		23	14	
γ_1		23	4	1
β_2		3	24	1
γ_2		3	4	12
β_3			34	12
γ_3			4	123
γ_4				1234

TABLE 10.2: Permutation $[3, 2, 1, 4]$ is sortable with two stacks.

The 2-stack-sorting algorithm takes as input π_1 and π_2 and returns in $O(n^2)$ time one sorting word of $E = \{w : w \text{ sorts } \pi_1 \text{ to } \pi_2\}$. If such a word w exists, then we can construct a planar bus realization with center buses of the embedded points $\pi_1\pi_2$ according to w as follows. We apply one of the following three rules to the letters of w. We process w letter by letter and read along $3n$ imaginary slots on the diagonal.

α_i the next slot of the diagonal is point i with a connection going up.

β_i the next slot of the diagonal is taken by the horizontal segment from the end of the connection of point i, then crossing the diagonal.

γ_i the next slot of the diagonal is point i with a connection down to its horizontal segment, extended such that this connection meets perpendicular.

This drawing is planar:

- any crossing of two edges incident to i, j to the left of the diagonal comes from the sequence $\ldots, \alpha(i), \ldots, \alpha(j), \ldots, \beta(i), \ldots$, which means push i on S_I, then push j on S_I and then pop i from S_I, which is impossible since i is not the topmost element of S_I.
- any crossing of two edges incident to i, j to the right of the diagonal comes from the sequence $\ldots, \beta(i), \ldots, \beta(j), \ldots, \gamma(i), \ldots$, which means push i on S_{II}, then push j on S_{II} and then pop i from S_{II} (and print i to the output), which is impossible since i is not the topmost element of S_{II}.

The construction from a planar bus realization with center buses of a diagonal point $\pi_1\pi_2$ set to a sorting word w for $\pi_1\pi_2$ is just traversing the diagonal from bottom to top and simultaneously building incrementally the sorting word w. We start with $w = \lambda$, where λ is the empty word. If the next item on the diagonal is the first appearance of a letter i, we set $w = w \circ \alpha_i$, where $\circ$ is the concatenation of two words. If the next item on the diagonal is a crossing of the edge connecting the two points of i, we set $w = w \circ \beta_i$. If the next item on the diagonal is the second appearance of a letter i, we set $w = w \circ \gamma_i$. It is easy to see that this word w sorts π_1 to π_2. □

10.4 NP-Completeness

In this section we prove that BEP$^\varepsilon$ for two points per color is NP-complete, where BEP$^\varepsilon$ is BEP with the additional input ε, which is the minimum distance of points to their bus. We show first that $(\sqcap, \sqcup)$-BEP$^\varepsilon$ for 2 points per color is NP-complete. To prove the hardness of $(\sqcap, \sqcup)$-BEP$^\varepsilon$ we show a reduction from planar 3-SAT [134] and give a clause gadget, variable gadget and a chain-gadget, cf. Figure 10.7. We use only $\sqcap$-buses and $\sqcup$-buses with distance at least ε to all their points. We use the parameter ε to build the gadgets. Also we first drop the "no points share a coordinate" restriction. In the end we transform the construction back into the "no points share a coordinate" setting and allow also center buses.

A **variable gadget** consists of two points a_1, a_2 of the same color on the same y-coordinate. The value of the variable is true if the two points are connected with a $\sqcap$-bus and the value of the variable is false if the two points are connected with a $\sqcup$-bus. We use a variable gadget, referred to as a *chain link*, also as elements of chain gadgets.

A **chain gadget** propagates the value of a chain link, which is actually a variable gadget, to another chain link. Let a_1, a_2 (respectively b_1, b_2) be the two points of the chain link at the beginning (respectively end) of the chain. A chain gadget consists of k chain links.

In a *horizontal* chain gadget we place the points on a single horizontal line in the order a_1, b_1, a_2, b_2 (respectively b_1, a_1, b_2, a_2) for propagating to the right (respectively to the left). If a_1, a_2 are connected with a $\sqcap$-bus, then b_1, b_2 must be connected with a $\sqcup$-bus and the other way round. This construction can be repeated until the chain consists of k chain links. The *sign* of a horizontal chain is defined by $(-1)^{(1+k)}$. Clearly if the sign is positive, then the first bus and the last bus are of the same type. If the sign is negative, then the first bus and the last bus are different.

In a *vertical* chain gadget we place b_1 below (respectively above) a_1 and b_2 below (respectively above) a_2 on the same x-coordinate with a distance of 2ε for propagating to the top (respectively to the bottom). It is easy to check that

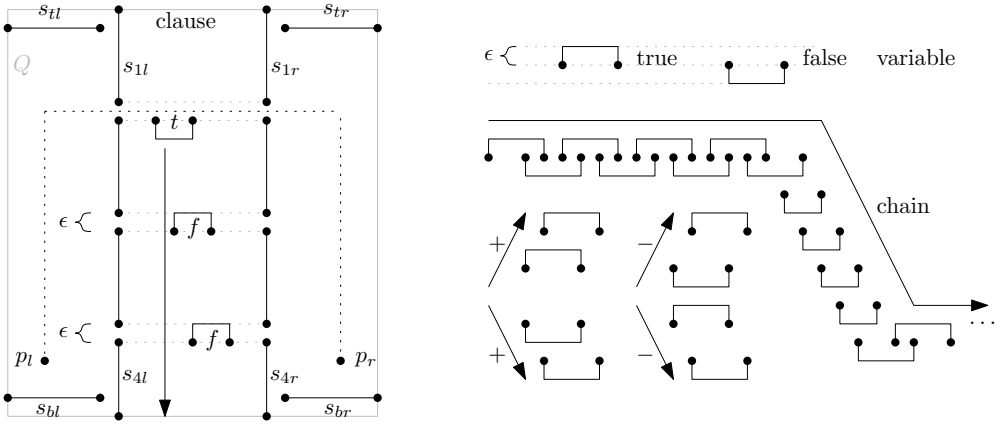


FIGURE 10.7: A clause, variable and chain gadget for reduction from planar 3-SAT. Vertical propagation of true and false are unique, but ⊓-buses are just uniquely propagated in the top direction and ⊔-buses are just uniquely propagated in the bottom direction.

in such a way we can only *uniquely propagate* a ⊓-bus to the top and a ⊔-bus to the bottom. It may happen that the type of buses change during a vertical propagation. The *sign* of a vertical chain is defined as $+1$.

The sign of two chains, which are connected, will be multiplied. If a literal in a clause appears positive, then the corresponding chain has sign -1, otherwise $+1$.

A **clause gadget** consists of two main points p_l, p_r, four horizontal bounding segments $s_{tl}, s_{tr}, s_{bl}, s_{br}$, eight vertical bounding segments $s_{1l}, s_{1r}, \ldots, s_{4l}, s_{4r}$, and 18 chain links, see a schematic illustration in Figure 10.7. We aim at satisfying the clause if and only if a bus connecting the main points can be drawn.

Within a bounding square Q we place horizontal bounding segment s_{tl} (s_{tr}) in the top left (right) corner, and horizontal bounding segment s_{bl} (s_{br}) in the bottom left (right) corner. Above s_{bl} (s_{br}) we place main point p_l (p_r) such that there is a normal to s_{bl} (s_{br}) through p_l (p_r) that is also crossing s_{tl} (s_{tr}). This construction prevents the bus connecting the main points to be in the exterior of Q.

In Q there are two vertical lines l and r that both separate p_l from p_r. We place the vertical bounding segments $s_{1x}, s_{2x}, s_{3x}, s_{4x}, x \in \{l, r\}$ in this order from bottom to top on line x with ε distance between every pair of consecutive segments. The resulting horizontal space between segment s_{ix} and $s_{(i+1)x}$ is called i-th *gap*, $i = 1, 2, 3$. The gaps represent the literals in the clause. This construction restricts the choices for the bus connecting the main points to be precisely three.

Finally we place 9 chain links below the first gap, 6 chain links between the first and second gap and 3 chain links between the second and the third gap. More specifically let $v_1, \ldots, v_6$ be 6 vertical lines between l and r in this order from left to right. We place 3 chain links on lines v_1, v_2 such that the first chain link has its points on the boundary of Q, the last chain link has its points on the bottom boundary of the first gap and the distance between every pair of chain link points is at most 2ε. Similarly we place 6 chain links on lines v_3, v_4 such that the first chain link has its points on the boundary of Q, the 4th chain link has its points on the top boundary of the first gap, the last chain link has its points on the bottom boundary of the second gap, and the distance between every pair of chain link points is at most 2ε. In the same way we place 9 chain links on lines v_5, v_6 such that the first chain link has its points on the boundary of Q, the 4th (7th) chain link has its points on the top boundary of the first (second) gap, the last chain link has its points on the bottom boundary of the third gap, and the distance between every pair of chain link points is at most 2ε. We refer to the last chain link of lines v_1, v_2 (respectively lines v_3, v_4 and v_5, v_6) as *the chain link of the first gap* (respectively second and third gap). This construction allows to block or open gaps from the bottom of Q.

Notice that it is easy to simulate vertical or horizontal segments with points as demonstrated in Figure 10.8.

The construction of an instance of $(\sqcap, \sqcup)$-BEP$^{\varepsilon}$ from an instance I of planar 3-SAT is done according to a planar drawing of the graph G_I. We may assume that all variable vertices of G_I are on a single horizontal line. We use this line and place the variable gadgets according to this order. The clause vertices above the variable vertices (top clauses) are replaced by clause gadgets and the clause vertices below the variable vertices (bottom clauses) are replaced by horizontally mirrored clause gadgets. Finally we replace the edges of G_I with chain gadgets.

The number of points needed to construct an instance of $(\sqcap, \sqcup)$-BEP$^{\varepsilon}$ is polynomial in n and m. Given a planar 3-SAT instance with n variables and m clauses, the corresponding $(\sqcap, \sqcup)$-BEP$^{\varepsilon}$ instance has at most $O(nm)$ points. For a clause gadget we need precisely 118 points, for a variable gadget we need 2 points and for chain gadgets we need 10 points plus the points needed to surround other clause gadgets. If an edge from a clause to variables vertically passes k other clauses, then we need $18k$ points for this construction. Since we have $O(n+m)$ edges and m clauses, we might need $O(nm+m^2)$ points for the edges.

Figure 10.8 shows how to use a variable several times: we stretch one chain link for horizontal propagation and add a vertical chain gadget for vertical propagation.

Theorem 10.7. $(\sqcap, \sqcup)$*-BEP*$^{\varepsilon}$ *for 2 points per color is NP-complete.*

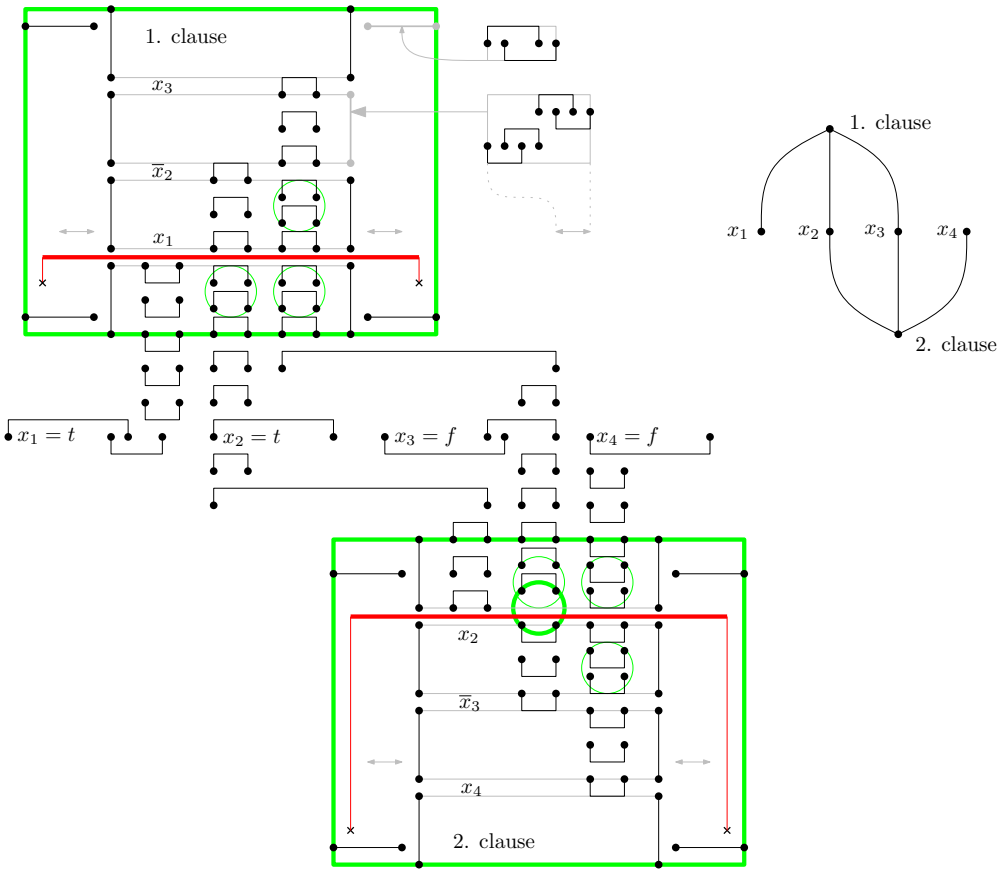


FIGURE 10.8: Point set instance constructed via gadgets for $I = (x_1 \vee \overline{x}_2 \vee x_3) \wedge (x_2 \vee \overline{x}_3 \vee x_4)$. The red buses indicate the truth assignment $x_3 = x_4 = f, x_1 = x_2 = t$ (t stands for true and f stands for false). Clause gadgets are enclosed by a green rectangle. The thin green circles indicate that distances are less than 2ε, while the thick green circle indicates a change of bus types during a not-unique vertical propagation of top buses to the bottom, that is, x_3 is meaningless for the second clause.

Proof. To show the membership of $(\sqcap, \sqcup)$-BEP$^\varepsilon$ in the class NP we observe that we have n points and between every pair of consecutive points we have a gap. In every gap there can be possibly n buses, that is, we have $(n-1)n$ slots, where to place buses. So every slot represents a possibility to place a bus. We can guess a drawing by choosing an order of the buses: all the drawings where buses move within their gap are equivalent. To check if the order leads to a feasible solution of $(\sqcap, \sqcup)$-BEP$^\varepsilon$, we apply the algorithm of Lemma 10.1.

We prove the hardness of $(\sqcap, \sqcup)$-BEP$^\varepsilon$ by a reduction from planar 3-SAT [134]. Let I be an instance of the planar 3-SAT problem and let P_I be the point set constructed from the gadgets, that is, we replace in the planar graph representing I every clause vertex by a clause gadget, every variable vertex by a variable gadget and every edge by a chain gadget. We prove next that P_I admits a solution of $(\sqcap, \sqcup)$-BEP$^\varepsilon$ if and only if I has a satisfying truth assignment.

"⇒:" If P_I admits a solution, then in particular every pair of main points is connected. Consider w.l.o.g. a top clause c with literal y corresponding to the gap through which the main points are connected. We associate with y the gap of c. If the chain link of y is a $\sqcup$-bus, then this bus is uniquely propagated to the bottom and does not change its type. If y is a positive variable x, then by construction the chain has sign -1 and the chain ends in a $\sqcap$-bus at the variable gadget, corresponding to x being true. If y is a negated variable $\overline{x}$, then by construction the chain has sign $+1$ and the chain ends in a $\sqcup$-bus at the variable gadget, corresponding to x being false. For bottom clauses the same argument holds horizontally mirrored.

"⇐:" Assume we have a satisfying truth assignment for I. We explain how to construct a solution for $(\sqcap, \sqcup)$-BEP$^\varepsilon$. First for every variable being true we draw a $\sqcap$-bus, while for every variable being false we draw a $\sqcup$-bus. We propagate $\sqcap$-buses with $\sqcap$-buses to the top and to the bottom, while we propagate $\sqcup$-buses with $\sqcup$-buses to the top and to the bottom. A $\sqcap$-bus ($\sqcup$-bus) ends in a $\sqcap$-bus ($\sqcup$-bus) if the variable in the top clause is negated (positive) or the variable in the bottom clause is positive (negated).

We keep the type of buses in a vertical propagation as long as possible, which can only be interrupted by a main bus. Then we change the type of buses and the gap becomes blocked, although the variable is true and appears positive, or the variable is false and appears negative in the clause. Additionally this interrupting main bus indicates that this clause is already satisfied and thus the variable of the interrupted chain is irrelevant for the satisfiability of this particular clause.

By construction we get a feasible (planar) solution for the buses in P_I.

Finally we translate the construction into the assumption "no two points share a coordinate". We may assume an underlying grid with grid unit $\varepsilon/2$ in the plane $\mathbb{R}^2$ so that all points have integer coordinates. Let $p_1, \ldots, p_n$ be the points ordered first by x-coordinate, then by y-coordinate. We modify the x-coordinates by a shift $x(l) = x(l) + 1$ for all $l \geq j$, if $x(p_i) = x(p_j)$, as long as two points share the same x-coordinate. We apply the same modification in y-direction with respect to the same order of points $p_1, \ldots, p_n$. Finally all points satisfy our assumption.

The properties of depending colors stay the same since the topological operation is just a stretch. Clearly chain links are dependent before the stretch if and only if they are dependent after the stretch. □

We consider as an example the instance $I = (x_1 \vee \overline{x}_2 \vee x_3) \wedge (x_2 \vee \overline{x}_3 \vee x_4)$ of planar 3-SAT. Clearly the clause-variable graph is planar. Figure 10.8 illustrates the point set created from the instance I.

We can adopt the same construction when additionally using center buses. Now some vertical segments can be modeled by using just two points. In a clause

gadget, we move one of the main points from bottom to top such that the bus connecting the main points is necessarily a center bus. The remaining parts are the same. Also for center buses we need ε as input for the minimum distance of buses to their points. Notice that a bus c and a point p of different color $c \neq c(p)$ may be closer than ε, as well as two buses c, c' may be closer than ε.

Theorem 10.8. *BEP$^{\varepsilon}$ for 2 points per color is NP-complete.*

10.5 Summary and Future Work

We studied bus embeddability, where a set of colored points is covered by a set of horizontal buses, one per color and without crossings. We described an ILP and an FPT-algorithm for the general problem and presented efficient algorithms for several restricted versions. The general problem is shown to be NP-complete even for two points per color when points may not lie on buses.

It is still open to determine the complexity of the problem for the following cases:

- BEP using only center buses;
- $(\sqcap, \sqcup)$-BEP, that is, BEP without center buses;
- general diagonal BEP, with more than two points per color;
- general BEP (since we used an extra ϵ as parameter).

A natural generalization would be to allow both horizontal and vertical buses, as in Chapter 9. Another variant might be to consider multi-colored points, where a point has to be connected either to all the buses of its corresponding colors, or to at least one of them. For point sets that have no solution for BEP with only one bus per color, we may allow more than one bus. Possible objectives in this scenario are

- to minimize the total number of buses over all colors,
- to minimize the total number of buses, and
- to minimize the total number of buses if each tree can connect at most l unicolored points.

Chapter 11

Short Conclusion on Bus Realizations

Bus realizations are one aesthetically pleasing way to visualize hypergraphs. We proved that it can be tested efficiently whether planar bus graphs admit a planar 2-dimensional bus realization. The non-planar bus graphs seem to admit no efficient test for admitting a 2-dimensional bus realization. Bus graphs with fixed vertex positions are related to many other problems such as Steiner trees, planar supports of hypergraphs, and boundary labeling. They also provide aesthetically pleasing visualizations. There are efficient tests for the existence for some very restricted versions, but most of the questions seem to be NP-hard.

Part III

Universal Point Set (UPS)

Chapter 12

Introduction

In the previous sections we have seen many problems that where solved with placing vertices on points of the Euclidean plane, or where vertices were already placed on fixed points of the Euclidean plane. The Euclidean plane is one possible space, where graphs can be drawn. Other spaces like the 3-dimensional Euclidean space or the 2-dimensional integer-grid provide also possible spaces for drawing graphs.

A fascinating problem in graph drawing is, which graphs admit a planar straight-line drawing, when the space is predefined. Clearly this is not challenging for the 3-dimensional Euclidean space [118] or the well-known 2-dimensional space, where we don't need to distinguish between straight-lines and curves by the Jordan-Schönflies-Theorem [182]. This result is used for the famous Fáry's Theorem [75] that any planar graph admits a planar straight-line drawing in the Euclidean plane. This theorem still holds, when restricting the plane to a 2-dimensional integer grid [45] of quadratic size.

The problem, which graphs admit a planar straight-line drawing on a given point set, becomes challenging, when considering point sets of subquadratic size. Let S be a set of m points on the plane. A *planar straight-line embedding* of an n-vertex planar graph G, with $n \leq m$, on point set S is a mapping of each vertex of G to a distinct point of S and of each edge of G to the straight-line segment between its corresponding end-points so that no two edges cross. Let $\mathcal{G}$ be a class of n-vertex planar graphs. Point set S is *universal* for $\mathcal{G}$ if for every graph $G \in \mathcal{G}$, G has a planar straight-line embedding on S. The problem of finding a universal point set of small size for the class of planar graphs is denoted by *UPS*. We will discuss previous work on UPS in Section 12.1. After that we consider the class of 2-outerplanar graphs and provide a point set of size $O(n \log n)$. To do so we split the problem in appropriate parts in Chapter 13. We conclude this part by giving final remarks and open problems in Chapter 14.

12.1 Related Work

We start with introducing in the universal point set problem with related work.

Asymptotically, the smallest universal point set for general planar graphs is known to have size at least $1.235n$ [39, 128], while the upper bound is still $O(n^2)$ [11, 40, 45, 166]. All the upper bounds are based on drawing the graphs on an integer grid, except for the one by Bannister et al. [11], who use super-patterns to obtain a universal point set of size $n^2/4 - \Theta(n)$, which is currently the best result for planar graphs. Characterizing the asymptotic size of the smallest universal point set, closing the gap between the lower and the upper bound, is a challenging open problems in graph drawing [34, 47, 139].

Angelini et al. [6] have provided a *universal point subset* of size $\sqrt{n}$, namely a point set S of size $\sqrt{n}$ such that every n-vertex planar graph admits a planar straight-line embedding in which $\sqrt{n}$ of its vertices are placed on the points of S.

A subclass of planar graphs for which the "smallest possible" universal point set is known is the class of *outerplanar* graphs, that is, the graphs that admit a straight-line planar drawing in which all vertices are incident to the outer face. Namely, Gritzmann et al. [99] and Bose [25] proved that any point set of size n in general position is universal for n-vertex outerplanar graphs. Gritzmann et al [99] also noted that outerplanar graphs are the largest class of graphs with this property.

Motivated by this result, we consider the class of *k-outerplanar* graphs, with $k \geq 2$, that is a generalization of outerplanar graphs. A planar drawing of a graph is k-outerplanar if removing the vertices of the outer face, called k-th *level*, produces a $(k-1)$-outerplanar drawing, where 1-outerplanar stands for outerplanar. A graph is k-outerplanar if it admits a k-outerplanar drawing.

Note that every planar graph is a k-outerplanar graph, for some value of $k \in O(n)$. Hence, in order to tackle a meaningful subproblem of the general one, it makes sense to study the existence of subquadratic universal point sets when the value of k is bounded by a constant or by a $o(n)$ function. However, while the case $k = 1$ is trivially solved by selecting any n points in the plane, as observed above [25, 99], the case $k = 2$ already eluded several attempts of solution and resulted far from being trivial. In the next chapter, we finally solve the case $k = 2$ by providing a universal point set for 2-outerplanar graphs of size $O(n \log n)$.

A different subclass of k-outerplanar graphs has also been recently considered, namely the *simply nested* graphs, that are k-outerplanar graphs in which the value of k is unbounded but every level is restricted to be a chordless simple cycle. For this class of graphs, a universal point set of size $O(n(\frac{\log n}{\log \log n})^2)$ has been proved by Angelini et al. [7], subsequently improved to $O(n \log n)$ by

Bannister et al. [11], again using super-patterns. For our calculations on the size of the universal point set we adopt a strategy that is analogous to the one of Bannister et al. [11].

Another very relevant subclass of planar graphs, not defined in terms of k-outerplanarity, for which a subquadratic universal point set is known is the class of *planar 3-trees*. A planar 3-tree is an incrementally constructed maximal planar graph that is obtained by starting with a triangle and by successively placing a new vertex inside one of its triangular faces connected to the three vertices incident to such face. For this class, Fulek et al. [89] recently described a universal point set of size $O(n^{5/3})$.

In order to construct our point set for 2-outerplanar graphs we first consider a specific subclass, called *cycle-tree* graphs, and then prove that the general case can be reduced to this case. A cycle-tree graph is a 2-outerplanar graph where the inner level is a tree and the outer level is a chordless simple cycle. Note that, cycle-tree graphs are a generalization (minimal superclass) of Halin graphs [176], as vertices of the outer level are not restricted to have degree 3 and leaves of the inner tree are not necessarily connected to vertices of the outer cycle.

Chapter 13

UPS for 2-Outerplanar Graphs

In this chapter we prove the existence of a universal point set of size $O(n \log n)$ for the class of 2-outerplanar graphs. This chapter is structured as follows. After some preliminaries and definitions in Section 13.1, we consider in Section 13.2 2-outerplanar graphs where the inner level is a forest and all the internal faces are triangulated. We prove that this class of graphs admits a universal point set of size $O(n^{3/2})$ by considering each of the faces of the outer level together with the tree lying inside it, which determine a cycle-tree graph. We then extend the result in Section 13.3 to 2-outerplanar graphs in which the inner level is still a forest but the faces are allowed to have larger size. Finally, we present in Section 13.4 the main result of this chapter, that is obtained by transforming any 2-outerplanar graph into one whose inner level is a forest, apply the algorithm described in Section 13.3, and revert the transformation while maintaining planarity. Also, we show in this section how to apply the calculations in [11] to reduce the size of the point set to $O(n \log n)$. The results are based on [198].

13.1 Preliminaries and Definitions

In this section we introduce and recall some necessary basic terminology used in the whole chapter. In the sequel we will require the graph to be undirected, simple and connected.

For construction of a universal point set, we need the following geometric notation. A straight-line segment with endpoints p and q is denoted by $s(pq)$. A circular arc with endpoints p and q (clockwise) is denoted by $a(pq)$.

A straight-line planar drawing of a graph G is obtained by placing each vertex u of G on a distinct point in the plane and representing each edge (u, v) of G

as the straight-line segment between the points where u and v are placed. A *planar straight-line embedding* of G on a point set S is a straight-line planar drawing of G in which the vertices are restricted to be placed only on the points of S. Recall from Section 2.1 that a straight-line planar drawing Γ of G determines a clockwise ordering of the edges incident to each vertex u of G, that we call rotation at u. The rotation scheme of G in Γ is the set of the rotations at all the vertices of G determined by Γ. Observe that, if G is connected, in all the straight-line planar drawings of G determining the same rotation scheme, the faces of the drawing are delimited by the same edges.

Let $[G, \mathcal{H}]$ be an arbitrary 2-outerplanar graph, where the outer level is an outerplanar graph G and the inner level is a set $\mathcal{H} = \{G_1, \ldots, G_k\}$ of outerplanar graphs. We assume that $[G, \mathcal{H}]$ is given together with a rotation scheme, and the goal is to construct a planar straight-line embedding of $[G, \mathcal{H}]$ on a point set determining this rotation scheme. Since $[G, \mathcal{H}]$ can be assumed to be connected, as otherwise we can add a minimal set of dummy edges to make it connected, this is equivalent to assuming that a straight-line planar drawing Γ of $[G, \mathcal{H}]$ is given. We finally rename the faces of G as $F_1, \ldots, F_k$ in such a way that each graph G_h, which can also be assumed connected, lies inside face F_h.

Note that, for each face F_h of G, the graph $[F_h, G_h]$ is again a 2-outerplanar graph; however, in contrast to $[G, \mathcal{H}]$, its outer level F_h is a simple chordless cycle and its inner level G_h consists of only one connected component. In the special case in which G_h is a tree we say that graph $[F_h, G_h]$ is a *cycle-tree* graph.

Also, we say that a 2-outerplanar graph is *inner-triangulated* if all its internal faces are 3-cycles. Note that, not every 2-outerplanar graph can be augmented to be inner-triangulated without introducing multiple edges.

13.2 Inner-Triangulated 2-Outerplanar Graphs with Forest

In this section we prove that there exists a universal point set containing $O(n^{3/2})$ points for the class of n-vertex inner-triangulated 2-outerplanar graphs $[G, \mathcal{H}]$ in which the inner level $\mathcal{H}$ is a forest. In order to obtain this result, we show in the following subsections how to:

- construct a universal point set S (Subsection 13.2.1),
- label the vertices of $[G, \mathcal{H}]$ (Subsection 13.2.2),
- embed $[G, \mathcal{H}]$ on S according to this labeling (Subsection 13.2.3).

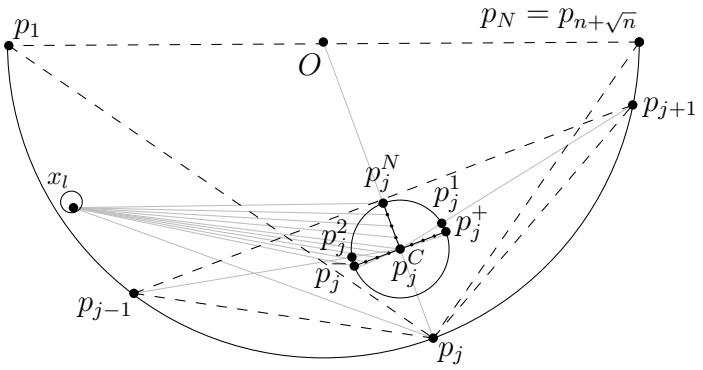


FIGURE 13.1: Illustration of the universal point set S, focused on the point set S_j of a point p_j.

13.2.1 Construction of the Universal Point Set

In the following we describe the structure of the point set S of size $O(n^{3/2})$ which will be later on proved to be universal for the class of n-vertex inner-triangulated 2-outerplanar graphs where the inner level is a forest. Refer to Figure 13.1 for an illustration of the construction.

Let π be a half circle with center point O and define $N := n + \sqrt{n}$. Uniformly distribute points $S_{\mathcal{M}} = \{p_1, \ldots, p_N\}$ on π. The points $S_{\mathcal{D}} = \{p_{i\sqrt{n}+i} : 1 \leq i \leq \sqrt{n}\}$ are called *dense points*, while the other points $S_{\mathcal{M}} \setminus S_{\mathcal{D}}$ are called *sparse points*.

For each $j = 2, \ldots, N-1$ consider the line segment $s(p_jO)$. Place a circle π_j with its center p_j^C on $s(p_jO)$, so that it lies completely inside the triangle $\triangle p_{j-1}, p_j, p_{j+1}$ and inside the triangle $\triangle p_1, p_j, p_N$. Note that the angle $\angle p_j p_j^C p_N$ (resp. $\angle p_j p_j^C p_1$) is smaller than 180°. Let p_j^N be the intersection point between $s(p_jO)$ and π_j that is closer to O. Also, let p_j^1 (resp. p_j^2) be the intersection point of $s(p_j^C p_{j+1})$ (resp. $s(p_j^C p_{j-1})$) with π_j. Finally let p_j^3 (resp. p_j^4) be the intersection point of π_j with its diameter that is orthogonal to $s(p_jO)$, such that $a(p_j^3 p_j^4)$ does not contain p_j^N.

Now, choose a point p_j^+ on the arc $a(p_j^1 p_j^3)$, and a point p_j^- on the arc $a(p_j^4 p_j^2)$. Observe that the angle $\angle p_j^- p_j^C p_j^+$ containing O is smaller than 180°. To complete the construction of S, evenly distribute $\overline{n} - 1$ points on each of the three segments $s_j^N := s(p_j^C p_j^N)$, $s_j^+ := s(p_j^C p_j^+)$, and $s_j^- := s(p_j^C p_j^-)$, where $\overline{n} = n$ if p_j is dense and $\overline{n} = \sqrt{n}$ if it is sparse. We refer to the points on s^N, s^+, s^-, including the points $p_j^N, p_j^C, p_j^+, p_j^-$, as *the point set of* p_j, and we denote it by S_j. Vertex p_j^C is the *center vertex of* S_j.

Note that the above described construction ensures the following properties for the point set.

Property 13.1. *For each $j = 1, \ldots, N$, the following visibility properties hold:*

(A) The straight-line segments connecting point p_j to point p_j^-, to the points on s_j^-, to p_j^C, to the points on s_j^+, and to p_j^+ appear in this clockwise order around p_j.

(B) For all $l < j$, consider any point $x_l \in \{p_l\} \cup R_l$ (see Figure 13.1); then, the straight-line segments connecting x_l to p_j^N, to the points on s_j^N, to p_j^C, to the points on s_j^-, to p_j^-, and to p_j appear in this clockwise order around x_l. Also, consider the line passing through x_l and any point in $\{p_j\} \cup S_j$; then, every point in $\{p_q\} \cup S_q$, with $l < q < j$, lies in the half-plane delimited by such line that does not contain the center point of π.

(C) For all $l > j$, consider any point $x_l \in \{p_l\} \cup R_l$; then, the straight-line segments connecting x_l to p_j^N, to the points on s_j^N, to p_j^C, to the points on s_j^+, to p_j^+, and to p_j appear in this counterclockwise order around x_l. Also, consider the line passing through x_l and any point in $\{p_j\} \cup S_j$; then, every point in $\{p_q\} \cup S_q$, with $j < q < l$, lies in the half-plane delimited by such line that does not contain the center point of π.

Proof. Item (A) follows from the fact that p_j^- and p_j^+ lie on different sides of segment $s(p_jO)$. In order to prove item (B), consider the intersection point p_x between π_j and segment $s(p_j^C x_l)$; then, the first statement of item (B) follows from the fact that points p_j^-, p_x, and p_j^N appear in this clockwise order along π_j. This is true since, by the construction of S, point p_x lies between p_j^2 and p_j^N, and point p_j^- precedes p_j^2 in this clockwise order. As for the second statement, this depends on the fact that each point set S_q, with $l < q < j$, is entirely contained inside triangle $\triangle p_{q-1}, p_q, p_{q+1}$. The proof for item (C) is symmetrical to the one for item (B). □

Property 13.2. *Point set S has $(\sqrt{n}-1)(3n+1)+(n-1)(3\sqrt{n}+1) = O(n^{3/2})$ points.*

13.2.2 Labeling the Graph

Let $[G, \mathcal{H}]$ be an inner-triangulated 2-outerplanar graph where G is an outerplanar graph and $\mathcal{H} = \{T_1, \ldots, T_k\}$ is a forest such that tree T_h lies inside face F_h of G, for each $1 \leq h \leq k$.

The idea behind the labeling is the following: in our embedding strategy, G will be embedded on the outer half-circle π of the point set S, while the tree $T_h \in \mathcal{H}$ lying inside each face F_h of G will be embedded on the point sets S_j of some of the points p_j on which vertices of F_h are placed. Note that, since π is a half-circle, the drawing of F_h will always be a convex polygon in which two vertices have *small* (acute) internal angles, while all the other vertices have

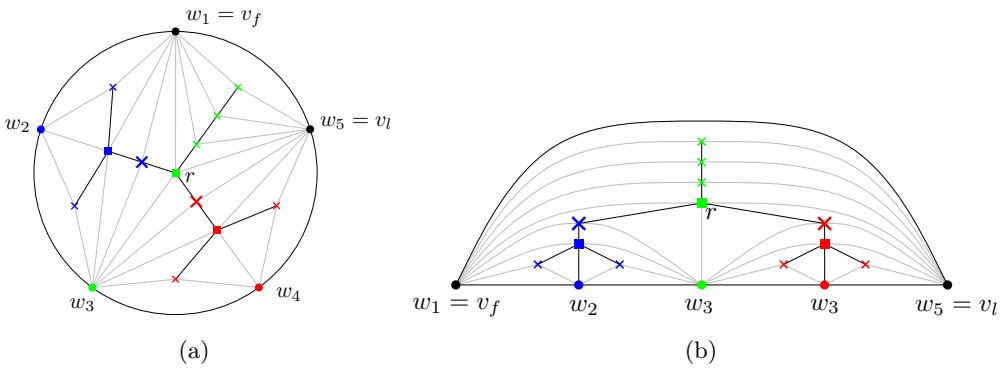
large (obtuse) internal angles. In particular, the vertices with the small angle are the first and the last vertices of F_h in the order they appear along the outer face of Γ. Since, by construction, a point p_j of F_h has its point set S_j in the interior of F_h if and only if it has a large angle, we aim at assigning each vertex of T_h to a vertex of F_h that is neither the first nor the last, and that hence will have a large angle. We describe this assignment by means of a labeling $\ell : [G, \mathcal{H}] \to 1, \ldots, |G|$; namely, we will assign a distinct label $\ell(v)$ to each vertex $v \in G$ and then assign to each vertex of T_h the same label as one of the vertices of F_h that is not the first or the last. Then, the number of vertices that are assigned the same label as a vertex of G will determine whether this vertex will be placed on a sparse or a dense point. This means that we cannot know in advance on which exact point a vertex of G will be placed, but we know whether the point will have a small or a large angle, and this information is enough to perform the labeling. We formalize this idea in the following.

First, we label the vertices of G. Rename the vertices of G as $v_1, \ldots, v_{|G|}$ in the order they appear along the outer face of Γ, and label them such that $\ell(v_i) = i$ for all $i = 1, \ldots, |G|$. Note that, any placement of vertices $v_1, \ldots, v_{|G|}$ on the points of π in the same order as they appear in Γ results in a planar embedding of G.

Next, we label the vertices of the trees $T_h \in \mathcal{H}$. Since the faces of G are independent, that is T_h and $T_{h'}$ are vertex disjoint for $h \neq h'$, we focus our description on a single face $F = F_h$ of G and on the tree $T = T_h \in \mathcal{H}$ lying inside it. Note that the induced subgraph $[F, T]$ is an inner-triangulated cycle-tree graph.

Rename the vertices of F as $w_1, \ldots, w_m$ in such a way that for each two vertices $w_x = v_p$ and $w_{x+1} = v_q$, where $p, q \in \{1, \ldots, |G|\}$, it holds $p < q$. The goal is to label each vertex of T with the same label $\ell(w_x)$ as a vertex w_x with $2 \leq x \leq m - 1$, since we know that w_1 and w_m will be the only vertices with small internal angles in any drawing of G on π. We say that a vertex of T is a *fork vertex* if it is adjacent to more than two vertices of F (square vertices in Figure 13.2(a)), otherwise it is a *non-fork vertex* (cross vertices in Figure 13.2(a)). Since $[F, T]$ is inner-triangulated, every vertex of T is adjacent to at least two vertices of F, and hence non-fork-vertices are adjacent to exactly two vertices of F.

We will now label the vertices of T starting from the fork-vertices. In order to do so, we first construct a tree T' composed only of the fork-vertices, as follows. Initialize $T' = T$. Then, as long as there exists a non-fork vertex of degree 3 (namely, with two neighbors in F and one in T'), remove it and its incident edges from T'. We call *foliage-vertices* the non-fork-vertices removed in this step (small crosses in Figure 13.2(a)). Then, all the other non-fork-vertices have degree 4 (namely 2 in F and 2 in T'); for each of them, remove it and its incident edges from T' and add an edge between the two vertices of



(a) (b)

FIGURE 13.2: (a) A cycle-tree graph $[F, T]$ with $F = \{w_1, w_2, w_3, w_4, w_5\}$, where $\ell(w_2)$ is blue, $\ell(w_3)$ is green and $\ell(w_4)$ is red. Fork-vertices are squares; foliage-vertices are small crosses, while branch-vertices are large crosses. Tree T' is composed of the root r (the green square vertex) with two children (the red and the blue square vertices). Vertices of T got color red, green, blue according to the labeling algorithm. (b) The placement of vertices in this drawing will be the same as on the point set.

T' that were connected to it before its removal. We call *branch-vertices* the non-fork-vertices removed in this step (large crosses in Figure 13.2(a)).

During the labeling, we say that a vertex $w_x \in F$ is *free* if there is no vertex of T' with label $\ell(w_x)$. In order to perform the labeling, we traverse T' bottom-up with respect to a root r that is the vertex of T' adjacent to both w_1 and w_m. Since $[F, T]$ is inner-triangulated this vertex is unique. During the visit of T' we maintain the following invariant.

Invariant: The vertices of T' are incident to only free vertices of F.

At the first step, the invariant is trivially satisfied, since all the vertices of F are free. Let a be the fork vertex considered in a step of the traversal of T', and let $w_{a_1}, \ldots, w_{a_k}$ be the vertices of F adjacent to a, with $1 \leq a_1 < \cdots < a_k \leq m$ and $k \geq 3$. By the invariant, all of $w_{a_1}, \ldots, w_{a_k}$ are free. Choose any vertex w_{a_i} such that $2 \leq i \leq k-1$, and assign $\ell(a) = \ell(w_{a_i})$. For example, the blue fork vertex in Figure 13.2(a), that is adjacent to w_3, w_4, and w_5 in F, gets label $\ell(w_4)$ (see Figure 13.2(b)). Since vertices $w_{a_2}, \ldots, w_{a_{k-1}}$ cannot be adjacent to any vertex of T' that is visited after a in the bottom-up traversal, the invariant is maintained at the end of each step.

Since at the last step of the traversal, when a coincides with the root r of T', we have that $w_{a_1} = w_1$ and $w_{a_k} = w_k$, and we have the following property.

Property 13.3. *Let $w_1, \ldots, w_m$ be the vertices of F with $\ell(w_1) < \cdots < \ell(w_m)$. The labeling of the fork-vertices of T computed according to the traversal of T' is such that both w_1 and w_m remain free.*

Now we label the non-fork-vertices of T based on the labeling of T'. Let b be a non-fork vertex. If b is a branch vertex, then consider the first fork vertex a encountered on a path from b to a leaf of T; set $\ell(b) = \ell(a)$. Otherwise, b is a foliage vertex. In this case, consider the first fork vertex a' encountered on a path from b to the root r of T. Also, let $v, w \in F$ be the two vertices of F adjacent to b; assume $\ell(v) < \ell(w)$. If $\ell(a') \leq \ell(v)$, then set $\ell(b) = \ell(v)$; if $\ell(a') \geq \ell(w)$, then set $\ell(b) = \ell(w)$; and if $\ell(v) < \ell(a') < \ell(w)$, then set $\ell(b) = \ell(a')$ (note that this latter case only happens when a' is the root and b is adjacent to w_1 and w_m). Figure 13.2(b) describes the labeling of the graph in Figure 13.2(a). The following property holds.

Property 13.4. *Adjacent non-fork-vertices have the same label.*

We perform the labeling procedure for every $T_h \in \mathcal{H}$ and obtain a labeling for $[G, \mathcal{H}]$.

For each $i = 1, \ldots, |G|$, we say that the subgraph of $\mathcal{H}$ induced by all the vertices of $\mathcal{H}$ with label i is the *restricted subgraph* H_i of $\mathcal{H}$ for i, see Figure 13.2. We prove an important lemma concerning restricted subgraphs.

Lemma 13.5. *Let H_i be the restricted subgraph of $\mathcal{H}$ for some $1 \leq i \leq |G|$. Then H_i is a tree whose all the vertices have degree at most 2, except for at most one that might have degree 3.*

Proof. First observe that, due to the procedure used to label the vertices of T', graph H_i contains at most one fork vertex a, which is hence the only one that might have degree larger than 2. Since, by Property 13.4, adjacent non-fork-vertices got the same label, H_i is connected and only contains paths of non-fork-vertices incident to a. We prove that there exist at most three of such paths. First, H_i contains at most one path of branch-vertices incident to a, namely the one connecting it to its unique parent in T'. Further, H_i contains at most two paths of foliage-vertices incident to a, namely one composed of the foliage-vertices adjacent to w_x and to w_{x-1}, and one composed of the foliage-vertices adjacent to w_x and to w_{x+1}, where $w_{x-1}, w_x, w_{x+1} \in G$ and $\ell(w_x) = i$. Note that, if a coincides with the root r of T, there might exist three paths of foliage-vertices incident to a, namely the two that are incident to w_x, w_{x-1}, and w_{x+1}, as before, plus one composed of the foliage-vertices that are incident to both w_1 and w_m; however, since r has no parent in T', there is no path of branch-vertices incident to a in this case. This concludes the proof of the lemma. □

13.2.3 Embedding on the Point Set

In this subsection we prove the following lemma.

Lemma 13.6. *Let $[G, \mathcal{H}]$ be an n-vertex inner-triangulated 2-outerplanar graph where $\mathcal{H}$ is a forest. Then, $[G, \mathcal{H}]$ has a planar embedding on S.*

We describe an embedding algorithm consisting of three steps:

a) Assignment of weight to each vertex v of G according to how many vertices have the label $\ell(v)$.

b) Embedding of G on point set S.

c) Embedding of the trees $\mathcal{H} = \{T_1, \ldots, T_k\}$ on S.

Step a: This step is straightforward after the labeling of $[G, \mathcal{H}]$ described in Subsection 13.2.2. Namely, let $\omega : G \to \mathbb{N}$ be the weight function such that $\omega(v_i) = |\{v \in [G, \mathcal{H}] | \ell(v) = i\}|$ for every $v_i \in G$. Note that $\sum_{v_i \in G} \omega(v_i) = n$.

We categorize each vertex $v_i \in G$ as *sparse* or *dense*, depending on whether $1 \leq \omega(v_i) \leq \sqrt{n}$ or $\omega(v_i) > \sqrt{n}$. Since $\sum_{v_i \in G} \omega(v_i) = n$, there are at most $\sqrt{n}$ dense vertices.

Step b: In this step we draw the vertices $v_1, \ldots, v_{|G|}$ of G on the $N := n + \sqrt{n}$ points of π in the same order as they appear along the outer face of Γ, in such a way that dense (sparse, resp.) vertices are placed on dense (sparse, resp.) points, as follows.

Since vertex v_1 is the first vertex in Γ along π, it is the first vertex in every face of G it is incident to; hence, by Property 13.3, there is no other vertex with the same label as v_1. Therefore, $w(v_1) = 1$ and thus vertex v_1 is sparse. We map v_1 to the sparse point p_1. During the following procedure, the points $p_2, \ldots, p_N$ to which no vertex has been mapped yet are called *free* points. We next look at every vertex v_i, for $i = 2, \ldots, |G|$, assuming that vertex v_{i-1} was mapped to a point p_j. Then, vertex v_i is mapped to the first free sparse (resp. dense) point after p_j, if v_i is sparse (resp. dense).

Note that, for each subsequence of points in π composed of $\sqrt{n}$ sparse points plus the dense point following them, either we use all the $\sqrt{n}$ sparse points or we use the dense point (possibly plus some sparse points). In both cases, the sum of the weights of the vertices that are placed on this sequence of points is at least $\sqrt{n}$, as sparse vertices have weight at least 1 and dense vertices have weight at least $\sqrt{n}$. Since π contains $\sqrt{n}$ of such subsequences, and since the total weight of the vertices is n, this procedure places all the vertices of G on points of π. Also, as observed before, the resulting embedding $\widetilde{\Gamma}$ of G is planar, since vertices are placed on π in the same order as in the planar drawing Γ. Finally, we observe that $\widetilde{\Gamma}$ has the following property (see Figure 13.3(a)) which directly follows from the construction of S:

Property 13.7. *Consider a face F of G and let Q be the polygon representing F in $\widetilde{\Gamma}$. Let $p_{j_1}, \ldots, p_{j_m}$, with $1 \leq j_1 < \cdots < j_m \leq N$ be the points of π where the vertices of F have been placed. Then, Q contains in its interior all the point sets $S_{j_2}, \ldots, S_{j_{m-1}}$.*

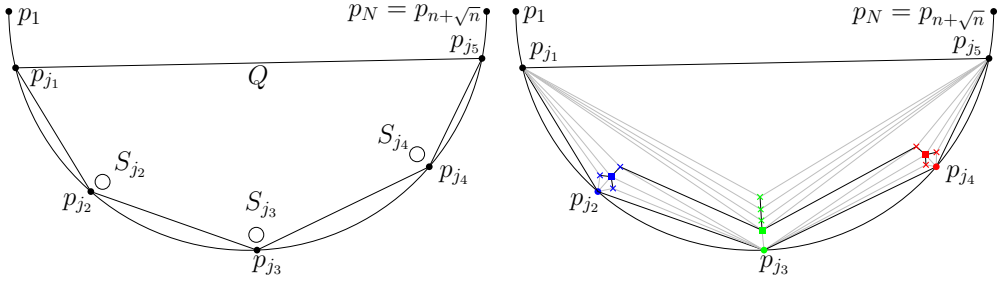


FIGURE 13.3: (a) Polygon Q contains all point sets of points incident to large angles. (b) An embedding of the graph in Figure 13.2(b) according to Steps a,b, and c.

Step c: Finally, we consider forest $\mathcal{H} = \{T_1, \ldots, T_k\}$. As in the labeling subsection, we describe the embedding algorithm for a single cycle-tree graph $[F, T]$, where $F = w_1, \ldots, w_m$ is a face of G and $T \in \mathcal{H}$ is the tree lying inside F. In particular, we show how to embed the restricted subgraph H_i, for each vertex w_x of F with label $\ell(w_x) = i$, on the point set S_j of the point p_j where w_x is placed. We remark that the labeling procedure ensures that $|H_i| + 1 = \omega(w_x) \leq |S_j|$; also, by Property 13.7, point set S_j lies inside the polygon representing F, except for the two points where vertices w_1 and w_m have been placed

By Lemma 13.5, H_i has at most one vertex a of degree 3, while all other vertices have smaller degree. Recall that, if a exists, then it is a fork vertex. We place a on the center point p_j^C of p_j, if it exists. The at most three paths of non-fork-vertices are placed on segments s_j^+, s_j^-, s_j^N; namely, the unique path of branch-vertices is placed on s_j^N, while the two paths of foliage-vertices are placed on s_j^+ or s_j^- based on whether the vertex of G different from w_x they are incident to is w_{x+1} or w_{x-1}, respectively. If $a = r$, then the path of foliage-vertices incident to w_1 and w_m is placed on s_j^N. The vertices of the paths are placed on segments s_j^+, s_j^-, s_j^N starting from a and using the points from p_j^C to p_j^+, p_j^-, p_j^N, respectively.

We show that the described placement of the vertices of H_i, for all $w_x \in F$, results in a planar drawing of T.

First, observe that the ordering of the internal fork-vertices and of the leaves of T is such that, for every two fork-vertices $a \in H_p$ and $a' \in H_q$, with $p < q$, all the leaves of the subtree of T rooted at a have smaller label than all the leaves of the subtree of T rooted at a'.

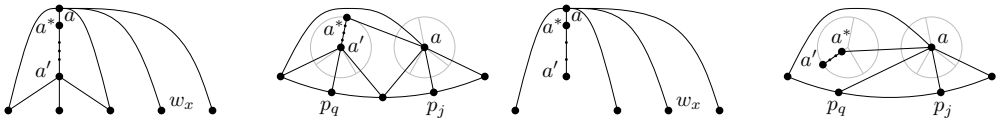


FIGURE 13.4: (a) Embedding of path P if P contains a fork vertex a' other than a. (b) Embedding of path P if P contains no other fork vertex than a. In this case a' is a leaf of T.

Then, for each $w_x \in F$, with $\ell(w_x) = i$, consider the fork vertex $a \in H_i$; recall that a has been placed on p_j^C. Let P be any path connecting a to a leaf of T and let a^* be the neighbor of a in P.

If P contains a fork vertex other than a (Figure 13.4(a)), then let a' be the fork vertex in P that is closest to a (possibly $a' = a^*$) and let p_q^C be the point where a' has been placed. Assume $q < j$, the case in which $q > j$ being analogous (note that, $q \neq j$, as no two fork-vertices can belong to the same restricted subgraph). By definition, the non-fork-vertices in the path between a and a' (if any) are branch-vertices, and hence they have been placed on the points of s_q^N. Then, Property 13.1 ensures that the straight-line edge connecting a and a^* separates all the point sets S_p with $q < p < j$ from the center of π. Since, by construction, the vertices that are placed on these point sets S_p are only connected either to each other or to the vertices on s_j^-, respectively on s_q^+, edge (a, a^*) is not involved in any crossing.

If P does not contain any fork vertex other than a (Figure 13.4(b)), then all the vertices of P other than a are foliage-vertices and are placed on a segment s_q^+ or s_q^-, for some q. In particular, if $q < j$, then they are on s_q^-; if $q > j$, then they are on s_q^+; while if $q = j$, then they are either on s_q^+ or on s_q^-. In all the cases, Property 13.1 ensures that the straight-line edge connecting a to a^* does not cross any edge.

Finally, observe that any path of T containing only non-fork-vertices is placed on the same segment of the point set (either s_j^N or s_j^+ or s_j^-, for some j), and hence its edges do not cross. As for the edges connecting vertices in one of these paths to the two leaves of T they are connected to, note that by item (A) of Property 13.1 the edges between each of these leaves and these vertices appear in the rotation at the leaf in the same order as they appear in the path.

Figure 13.3(b) shows a planar embedding of the graph from Figure 13.2(b) according to steps a,b, and c. All the above discussion leads to the following result:

Lemma 13.8. *There exists a universal point set of size $O(n^{3/2})$ for the class of n-vertex inner-triangulated 2-outerplanar graphs $[G, \mathcal{H}]$ where $\mathcal{H}$ is a forest.*

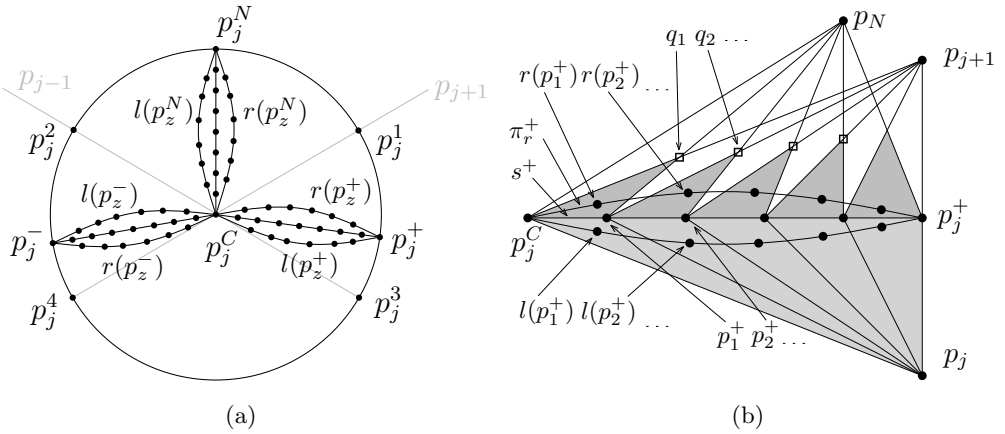


FIGURE 13.5: Construction of petal points for s^+. Dark-gray triangles are used for petal points $r(p_z^+)$ while light-gray triangles for $l(p_z^+)$.

13.3 2-Outerplanar Graphs with Forest

In this section we consider 2-outerplanar graphs $[G, \mathcal{H}]$ where $\mathcal{H}$ is a forest. Contrary to the previous section, we do not assume $[G, \mathcal{H}]$ to be inner-triangulated. As observed before, augmenting it to be inner-triangulated might be not possible without introducing multiple edges.

The main idea to overcome this problem is to first identify the parts of the graph that do not allow for the required augmentation, remove them, and augment the resulting graph with dummy edges to inner-triangulated (Section 13.3.2); then, apply Lemma 13.8 to embed the inner-triangulated graph on the point set S; and finally remove the dummy edges and embed the parts of the graph that had been previously removed on the remaining points (Section 13.3.3). However, in order to ensure that this last step can always be performed without introducing any crossing, we first need to extend the point set S with some additional points. We denote the augmented point set by S^* and we describe it in Section 13.3.1.

13.3.1 Extending the Universal Point Set

Let S be the universal point set constructed in Section 13.2.1. We construct S^* from S by adding *petal points* to the point sets S_j, for every $j = 2, \ldots, N-1$ (see Figure 13.5(a)). For simplicity of notation, we skip the subscript j in the following description whenever possible.

Let s^+, s^N, s^-, respectively, be the segments of S_j as defined in Section 13.2.1. We denote by p_z^σ the z-th point on segment s^σ, with $\sigma \in \{+, -, N\}$ and $z = 1, \ldots, \overline{n}$ (where $\overline{n} = \sqrt{n}$ or $\overline{n} = n$, depending on whether p_j is sparse or dense), in such a way that p_1^σ is the point following p^C along s^σ and $p_{\overline{n}}^\sigma = p_j^\sigma$. For each point p_z^σ of s^σ we add two *petal points* $l(p_z^\sigma)$ and $r(p_z^\sigma)$ to S^*, as follows.

We first describe the procedure for s^+, as illustrated in Figure 13.5(b). For each $z = 1, \ldots, \overline{n}$, consider the intersection point q_z between segments $s(p^+_{z-1}p_{j+1})$ and $s(p^+_z p_N)$, where $p^+_{z-1} = p^C_j$ when $z = 1$. Note that, by construction, all triangles $\triangle p^+_{z-1}p^+_z q_z$ have two corners on s^+, have the other corner in the same half-plane delimited by the line through s^+, and do not intersect each other except at common corners. Hence, it is possible to construct a convex arc π^+_r passing through points p^C_j and $p^+_{\overline{n}} = p^+_j$, and intersecting the interior of every such triangle. For each $z = 1, \ldots, \overline{n}$, we place the petal point $r(p^+_z)$ on the arc of π^+_r lying inside triangle $\triangle p^+_{z-1}p^+_z q_z$. In order to place the other petal points $l(p^+_z)$, for each $z = 1, \ldots, \overline{n}$, we use the same procedure by considering triangles $\triangle p^+_{z-1}p^+_z p_j$ instead of $\triangle p^+_{z-1}p^+_z q_z$.

In the symmetric way we construct the petal points for s^-, using points p_{j-1} and p_1 to place $l(p^-_z)$ and point p_j to place $r(p^-_z)$, and for s^N, using points p_{j-1} and p_1 to place $l(p^N_z)$ and points p_{j+1} and p_N to place $r(p^N_z)$.

Recall that we have $N = n + \sqrt{n}$ points p_j on the outer half circle π of S, and $N - 2$ of them have their point set S_j. For each dense p_j we added $6n$ points to S^*, while for every sparse p_j we added $6\sqrt{n}$ points.

Property 13.9. *Point set S^* has $(\sqrt{n}-1)(9n+1)+(n-1)(9\sqrt{n}+1) = O(n^{3/2})$ points.*

13.3.2 Modifying and Labeling the Graph

We now aim at modifying $[G, \mathcal{H}]$ to obtain an inner-triangulated graph that can be embedded on the original point set S (**Part A** and **Part B**); we defer to the following section the description of how to exploit this embedding on S to obtain an embedding of the original graph $[G, \mathcal{H}]$ on the extended point set S^* (**Part C**).

As in Sections 13.2.2 and 13.2.3, we describe the procedure just for a simple cycle-tree graph $[F, T]$ composed of a face F of G and of the tree T inside it.

We first summarize the operations performed in the different Parts and then give more details in the following.

1. **Part A:**
   - We delete some edges from $[F, T]$ connecting F with T to identify "tree components", resulting in a new graph $[F, T' = T]$; note that the set of edges connecting T' to F might be different from the set of edges connecting T to F.
   - We delete from $[F, T']$ the "tree components", to be defined later, and obtain a new graph $[F, T'' \subseteq T']$ which has the property that it admits an augmentation to inner-triangulated without multiple edges.

- We augment $[F, T'']$ to an inner-triangulated graph $[F, T^{\Delta} = T'']$; again, instance $[F, T^{\Delta}]$ might differ from $[F, T'']$ only on the set of edges connecting the two levels.

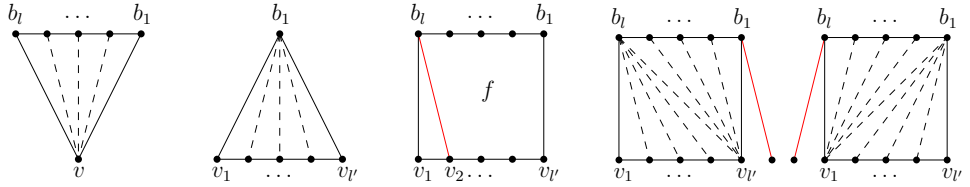
2. We label $[F, T^{\Delta}]$ with the algorithm described in Section 13.2.2.
3. **Part B:**
   - We insert vertices in $[F, T^{\Delta}]$ representing the previously removed tree components and give suitable labels to these vertices, hence obtaining a new instance $[F, T^{\mathcal{A}} \supseteq T^{\Delta}]$. By adding appropriate edges we keep the instance triangulated.
4. We embed $[F, T^{\mathcal{A}}]$ on point set S with the algorithm described in Section 13.2.3.
5. **Part C:**
   - We obtain a planar embedding of $[F, T]$ on point set S^* by removing all the vertices and edges added during these steps and by suitably adding back the removed edges and tree components.

Part A: Let $[F, T]$ be a simple cycle-tree graph. We categorize each face f of $[F, T]$ based on the number of vertices of F and of T that are incident to it. Note that, since T is a tree, f has at least a vertex of F and a vertex of T incident to it. Refer to Figure 13.6 for an illustration.

If f contains exactly one vertex of F, then it is a *petal face*. The reason for this name is that the "tree components" lying inside petal faces will be placed on petal points later. If f contains exactly one vertex of T, then it is a *small face*. If f is neither a petal nor a small face, then it is a *big face*. Suppose f is a big face, and let $b_1, \ldots, b_l$ be the vertices of T in the clockwise order they appear along the boundary of f. If either b_1 or b_l, say b_1, has more than one adjacent vertex in F (namely one in f and at least one not in f), then we say that f is *protected* by b_1. If f is a big face with exactly two vertices incident to F and is not protected by any vertex, then we say that f is a *bad face*.

We now prove a lemma that gives sufficient conditions to triangulate G without introducing multiple edges; we will later use this lemma to identify the "tree components" of T whose removal allows for a triangulation.

Lemma 13.10. *Let $[F, T]$ be a biconnected simple cycle-tree graph, such that (1) each vertex of F has degree at most four, and (2) there exists no bad face in $[F, T]$. It is possible to augment $[F, T]$ to an inner-triangulated simple cycle-tree graph by only adding dummy edges.*



(a) (b) (c) (d) (e)

FIGURE 13.6: Insertion of triangulation edges in a (a) petal face, (b) small face, (c) a non-protected big face, and a big face protected by vertex (d) b_1 and (e) b_l.

Proof. Let f be any face of $[F, T]$. We describe how to triangulate f without creating multiple edges.

Suppose f is a petal face (see Figure 13.6(a)); let $v, b_1, ..., b_l$ (with $l > 2$) be the vertices on its boundary, where $v \in F$ and $b_i \in T$ for $1 \leq i \leq l$. We triangulate f by adding an edge (v, b_i), for each $2 \leq i \leq l-1$. Since $[F, T]$ is biconnected, there exists no multiple edge inside f. Also, since condition (1) ensures that $v \in F$ has degree at most four, there is no petal face incident to v other than f, and thus no multiple edge is created outside f.

Suppose f is a small face (see Figure 13.6(b)); let $v_1, \ldots, v_{l'}, b$ (with $l > 2$) be the vertices on its boundary, where $v_i \in F$ for $1 \leq i \leq l'$ and $b \in T$. We triangulate f by adding an edge (b, v_i), for each $2 \leq i \leq l'-1$. Note that, before introducing these edges, vertices $v_2, \ldots, v_{l'-1} \in F$ were not connected to any vertex of T (and in particular to b); thus, no multiple edge is created.

Suppose f is a big face that is not a bad face; let $v_1, ..., v_{l'}, b_1, ..., b_l$ (with $l, l' > 1$) be the vertices along the boundary of f, where $v_1, ..., v_{l'} \in F$ and $b_1, ..., b_l \in T$. If f is not protected by any vertex (see Figure 13.6(c)), then $l' \geq 3$, as otherwise it would be a bad face. This implies that vertex $v_2 \in F$ is not connected to any vertex of T. Hence, it is possible to add edge (b_l, v_2) without creating multiple edges. Face f is hence split into a triangular face v_1, v_2, b_l and a big face that is protected by b_l, which we cover in the next case. Otherwise, f is protected by a vertex. If f is protected by b_1 (see Figure 13.6(d)), then we triangulate f by adding edges $(b_i, v_{l'})$, for $2 \leq i \leq l$ and (b_l, v_i), for $2 \leq i \leq l'-1$. If f is protected by b_l (see Figure 13.6(e)), then we triangulate f by adding edges (b_i, v_1), for $1 \leq i \leq l-1$ and (b_1, v_i), for $2 \leq i \leq l'-1$. Note that, before introducing these edges, vertices $v_2, \ldots, v_{l'-1} \in F$ were not connected to any vertex of T (and in particular to b_1 and b_l); also, vertices $b_2, \ldots, b_l$ (vertices $b_1, \ldots, b_{l-1}$) were not connected to $v_{l'}$ (resp. to v_1), f was protected by b_1 (resp. b_l). Thus, no multiple edge is created.

Since by condition (2) there exists no bad face in $[F, T]$, all the possible cases have been considered; this concludes the proof of the lemma. □

We now describe a procedure to transform the given cycle-tree graph $[F, T]$ into another one $[F, T'']$ that is biconnected and satisfies the conditions of Lemma 13.10. We do this in two steps: first, we remove some edges connecting a vertex of F and a vertex of T in order to transform $[F, T]$ into a cycle-tree graph $[F, T' = T]$ that is not biconnected but that satisfies the two conditions of the lemma; then, we remove the "tree components" of T' that are not connected to vertices of F in order to obtain a cycle-tree graph $[F, T'' \subseteq T']$ that is also biconnected.

In order to satisfy condition (1) of Lemma 13.10, we perform the following operation. As long as there exist two petal faces in $[F, T]$ sharing an edge $e = (v, b)$, we remove e from $[F, T]$. Note that the removal of e merges the two petal faces into a single petal face, also incident to v. Also, when this operation cannot be applied any longer, all the vertices of F have degree at most 4. We refer to the set of edges removed in this step as *petal edges*, denoted by E_P.

In order to satisfy condition (2) of Lemma 13.10, we repeatedly perform the operation described in the following. As long as there exists a bad face $f = v_1, v_2, b_1, \ldots, b_l$, where $v_1, v_2 \in F$ and $b_1, \ldots, b_l \in T$, let g be the face incident to v_1 that shares edge $e = (v_1, b_l)$ with f. We remove e, hence merging f and g into a single face f', that we subsequently split again by adding dummy edges, based on the type of face g, in such a way to ensure that no new bad face is created, as follows; see Figure 13.7.

First note that, since f is a bad face, it is not protected by b_l, and hence g is not a small face. If g is a petal face, then f' is still a big face with two vertices of F incident to it, namely v_1 and v_2; see Figure 13.7(a). We add an edge (v_1, b_1); this corresponds to splitting f' into a petal face $v_1, b_1, \ldots, b_l$ and a triangular face v_1, v_2, b_1. If g is a big face, then f' is a big face; see Figure 13.7(b). Let $w_1, \ldots, w_q, c_1, \ldots, c_h$ be the vertices incident to g, where $w_1, \ldots, w_q \in F$, with $w_q = v_1$, and $c_1, \ldots, c_h \in T$, with $c_1 = b_l$. We add two dummy edges (v_1, c_h) and (v_1, b_1). This corresponds to splitting f' into a small face $w_1, \ldots, w_q, c_h$, a petal face $v_1, b_1, \ldots, b_l = c_1, \ldots, c_h$, and a triangular face v_1, v_2, b_1.

We refer to the set of edges removed in this step as *big face edges*, denoted by E_B, and to the set of dummy edges added in this step as *triangulation edges*.

Since in both cases the application of this operation reduces the total number of bad faces by at least one and since the addition of a single dummy edge (of two dummy edges) incident to v_1 is performed only when v_1 has degree 3 (resp., degree 2), after a linear number of operations we obtain an instance $[F, T']$ such that every vertex of F has degree at most 4 and there exists no bad face. Note that, each edge that has been removed in the two steps connects a vertex of F to a vertex of T, and thus $T' = T$.

In order to transform $[F, T']$ into a biconnected graph, we note that $[F, T']$ is comprised by a biconnected component which contains F, called *block-component*, and a set $\mathcal{T}_B$ of subtrees of T', called *tree components*, each sharing

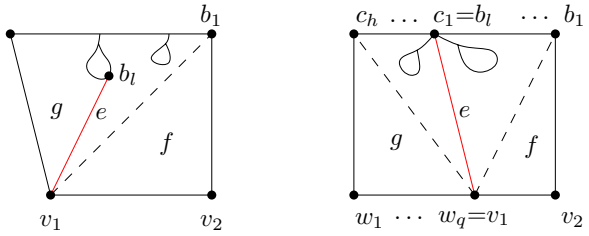


FIGURE 13.7: Illustration of the two cases for removing bad faces. Face g is a petal face in (a) and a big face in (b). Dummy edges are dashed, while the removed edge e is red.

a cut-vertex with the block component. A tree component sharing a cut-vertex c with the block component is denoted by T_c.

We remove the tree components $\mathcal{T}_B$ from $[F, T']$ and obtain an instance $[F, T'' \subseteq T']$, that is actually the block component of $[F, T']$. Note that the removal of $\mathcal{T}_B$ does not change the degree of the vertices of F and does not create any bad face. Hence, $[F, T'']$ is indeed a biconnected instance that satisfies the two conditions of Lemma 13.10. Thus, we can augment it to an inner-triangulated instance $[F, T^\Delta]$, with $T^\Delta = T''$, without introducing multiple edges. The edges introduced during the triangulation are also called *triangulation edges*.

In the following we prove some properties of $[F, T^\Delta]$.

Lemma 13.11. *Let $e = (b, v)$ be an edge of $E_P \cup E_B$, where $b \in T$ and $v \in F$. Then, either e is a triangulation edge in $[F, T^\Delta]$ or $b \notin T''$, that is, b belongs to a tree component T_c of $\mathcal{T}_B$ sharing a cut-vertex c with the block-component $[F, T'']$. In the latter case, edge (v, c) is a triangulation edge in $[F, T^\Delta]$.*

Proof. Suppose that $b \in T''$; we prove that e is a triangulation edge in $[F, T^\Delta]$.

If $e \in E_P$, this directly descends from the fact that the algorithm to triangulate a petal face f described in Lemma 13.10 adds a triangulation edge between every vertex of T incident to f, including b, and the only vertex of F incident to f, namely v.

If $e \in E_B$, this depends again on the triangulation algorithm of Lemma 13.10 and on the addition of the one or two dummy edges incident to v that is performed when merging the two faces sharing edge e. In fact, these dummy edges ensure that there exists a petal face in which v is the only vertex of F; then, the same argument as above applies to prove that v is connected to b by a triangulation edge.

Suppose that $b \notin T''$ and let T_c be the tree component such that $b \in T_c$; the fact that there exists a triangulation edge connecting v to c follows from the

same arguments as above, since in both cases v is connected by triangulation edges to all the vertices of T, including c, incident to the same face it is incident to. □

Lemma 13.12. *Let $T_c \in \mathcal{T}_B$ be a tree component such that there exists at least an edge $e = (b, v) \in E_P \cup E_B$, with $b \in T_c$ and $v \in F$. Then, for each edge in $E_P \cup E_B$ with an endvertex belonging to T_c, the other endvertex is v.*

Proof. First suppose that all the edges in $E_P \cup E_B$ connecting a vertex of T_c to a vertex of F, including e, belong to E_P. Consider the two edges e_1 and e_2 such that e_1 and e_2 connect v to vertices of T, and all the other edges that connect v to vertices of T lie between e_1 and e_2 in the circular order of the edges around v in $[F, T]$. Note that, all the edges between e_1 and e_2 belong to E_P, while e_1 and e_2 do not, as one of the two faces they are incident to is not a petal face. Let f be the face both e_1 and e_2 are incident to after the removal of all the edges between them. Since all the vertices of T_c are incident to f, and since v is the only vertex of F incident to f, all the edges of E_P connecting a vertex of T_c to a vertex of F are incident to v.

Suppose now that there exists at least an edge of E_B connecting a vertex of T_c to a vertex of F. Hence, we can assume that $e \in E_B$. This implies that e is incident to a bad face f and a face g that can be either a petal or a big face.

If g is a petal face, then let $e' = (v, b')$ be the other edge incident to g and to v. Since g is a petal face, edge e' belongs neither to E_P nor to E_B. Also, let $e'' = (v, b'')$ be the dummy edge incident to v added when removing e (the dashed edge in Figure 13.7(a)). Since, by construction, e'' is incident to a small face, it belongs neither to E_P nor to E_B, as well. Hence, both e' and e'' are edges of $[F, T']$ (and hence of $[F, T'']$) incident to v. This implies that all the vertices of T_c are incident to the unique face g of $[F, T']$ to which e' and e'' are incident. Since v is the only vertex of F incident to this face, all the edges of $E_P \cup E_B$ connecting a vertex of T_c to a vertex of F are incident to v.

If g is a big face, then let $e' = (v, b')$ and $e'' = (v, b'')$ be the two edges incident to v added when removing e (the dashed edges in Figure 13.7(b)). Again, e' and e'' belong to neither E_P nor E_B, since by construction they are both incident to small faces. The statement follows by the same argument as above. □

Performing the above transformation for every cycle-tree graph $[F, T]$ yields an inner-triangulated 2-outerplanar graph $[G, \mathcal{H}^{\Delta}]$, that is the outcome of **Part A** of the algorithm.

Part B: We now extend the labeling of $[G, \mathcal{H}^{\Delta}]$, computed as in Section 13.2.2, to the vertices of the tree components. Simultaneously, we augment $[G, \mathcal{H}^{\Delta}]$ by adding dummy vertices and edges in such a way that the augmented graph

FIGURE 13.8: Inserting dummy vertices for a tree-component in a triangular face (c, v, w) with $v, w \in F$ and $c \in T^{\Delta}$. (a) $\ell(c) \leq \ell(v)$ and (b) $\ell(c) \geq \ell(w)$.

is still inner-triangulated (and hence can be embedded according to Theorem 13.8) and that the tree components can be later reinserted by placing them on the petal points of the points where the dummy vertices have been placed.

As in previous sections, we describe the procedure for a single face F of $[G, \mathcal{H}^{\Delta}]$; let $[F, T^{\Delta}]$ be the corresponding inner-triangulated cycle-tree graph, and let $\mathcal{T}_B$ be the set of tree components, where each $T_c \in \mathcal{T}_B$ is rooted at the cut-vertex c separating it from $[F, T^{\Delta}]$.

Note that, the face of $[F, T'']$ to which a tree component T_c belongs might have been split into several faces of $[F, T^{\Delta}]$ by the addition of triangulation edges. We assign T_c to any of such faces f that is incident to c. Then, we label the vertices T_c based on the shape of f; we distinguish two cases.

Suppose f is a triangular face (c, v, w) with $v, w \in F$ and $c \in T^{\Delta}$, refer to Figure 13.8; assume $\ell(v) < \ell(w)$. We create a path P_c containing $|T_c| - 1$ dummy vertices and append this path at c. Then, we connect every dummy vertex of P_c with both v and w. If $\ell(c) \leq \ell(v)$, as in Figure 13.8(a), then we label the vertices of P_c with $\ell(P_c) = \ell(v)$. If $\ell(c) \geq \ell(w)$, as in Figure 13.8(b), then we label them with $\ell(P_c) = \ell(w)$. Note that, since (c, v, w) is a triangular face, v and w are consecutive along F, and hence c cannot have been labeled in such a way that $\ell(v) < \ell(c) < \ell(w)$. Also note that path P_c augments T^{Δ} without closing any cycle.

Suppose f is a triangular face (a, b, v) with $v \in F$ and $a, b \in T^{\Delta}$, refer to Figure 13.9; assume $\ell(a) \leq \ell(b)$. Then, either $c = a$ or $c = b$.

We replace edge (a, b) with a path P_c between a and b with $|T_c| - 1$ internal dummy vertices, and connect each of these dummy vertices to v and to w, where w is the other vertex adjacent to both a and b; note that, w exists since $[F, T^{\Delta}]$ is inner-triangulated, and $w \in F$ as otherwise there would be a cycle in T^{Δ}.

For each dummy vertex x of P_c, we assign $\ell(x) = \ell(a)$ if $\ell(v) \leq \ell(a)$; we assign $\ell(x) = \ell(b)$ if $\ell(v) \geq \ell(b)$; and we assign $\ell(x) = \ell(v)$ if $\ell(a) < \ell(v) < \ell(b)$.

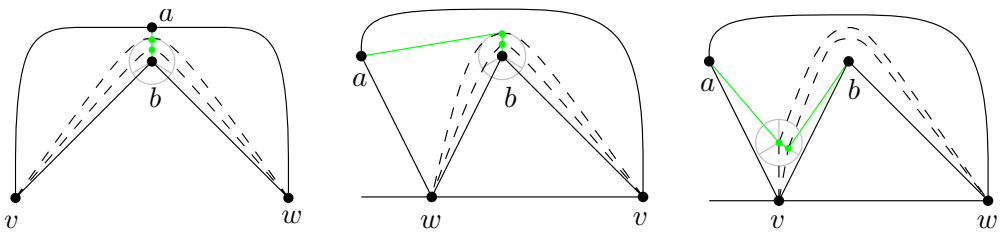


FIGURE 13.9: Inserting dummy vertices for a tree-component in a triangular face (a, b, v) with $v \in F$ and $a, b \in T^{\Delta}$, when (a) $\ell(a) = \ell(b)$, (b) $\ell(a) \neq \ell(b)$ and $\ell(w) < \ell(v)$, and (c) $\ell(a) \neq \ell(b)$ and $\ell(w) > \ell(v)$.

Note that the existence of edge $(a, b) \in T^{\Delta}$ implies that either a is the parent of b in T^{Δ} or vice versa. Suppose the former, the other case being analogous with a and b switching their role. Then, v and w are the extremal neighbors of b in F, and thus the label of b is between those of v and w, that is, either $\ell(v) \leq \ell(b) \leq \ell(w)$ or $\ell(w) \leq \ell(b) \leq \ell(v)$; if b is a fork vertex, then the inequalities are strict. Also note that, if $\ell(a) \neq \ell(b)$, then the label of a does not lie strictly between those of v and w. In fact, this can only happen if the label of b strictly lies between those of v and w, and $\ell(a) = \ell(b)$ (which happens only if a is a non-fork vertex). Since $\ell(a) \leq \ell(b)$, by assumption, this implies that $\ell(a) \leq \ell(v), \ell(w)$ (we remark that, when considering the symmetric case in which b is the parent of a, here we will have $\ell(b) \geq \ell(v), \ell(w)$). The two observations before can be combined to conclude that, if $\ell(a) = \ell(b)$, then all the tree components lying inside faces (a, b, v) and (a, b, w) have the same label as a and b (Figure 13.9(a)). Otherwise, either the tree components inside (a, b, v) have label $\ell(b)$ and those inside (a, b, w) have label $\ell(w)$ (Figure 13.9(b)), or the tree components inside (a, b, v) have label $\ell(v)$ and those inside (a, b, w) have label $\ell(b)$ (Figure 13.9(c)).

All the edges added in this step connecting a dummy vertex to v and w are again called *triangulation edges*.

We apply the modification of **Part B** for every cycle-tree graph $[F, T^{\Delta}]$ of $[G, \mathcal{H}^{\Delta}]$, hence creating a graph $[G, \mathcal{H}^{\mathcal{A}}]$ that is still an inner-triangulated 2-outerplanar graph where $\mathcal{H}^{\mathcal{A}}$ is a forest.

Since, by construction, all the dummy vertices added in the last step of the augmentation are connected to exactly two vertices $v, w \in F^{\Delta}$, all such dummy vertices are non-fork-vertices.

We also remark that the labeling of the dummy vertices is the same as the one that the algorithm described in Subsection 13.2.2 would have assigned to them, except for one single case, namely the one in which the face f containing the tree component T_c is a triangular face (a, b, v) with $v \in F$ and $a, b \in T^{\Delta}$, and $\ell(a) < \ell(v) < \ell(b)$. In this case, indeed, the algorithm would have assigned to

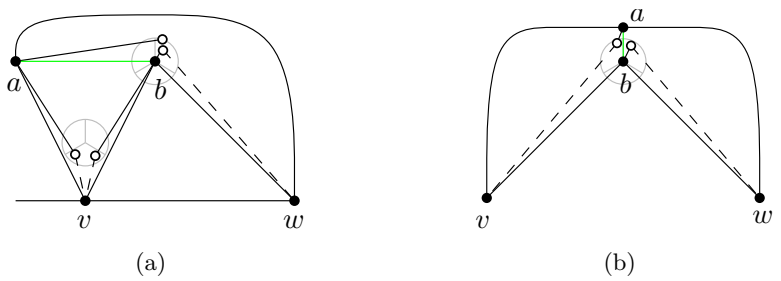


FIGURE 13.10: Moving vertices of tree components to petal points if (a) $\ell(a) \neq \ell(b)$ and if (b) $\ell(a) = \ell(b)$.

the dummy vertices the same label as either a or b, depending on whether b is the parent of a or vice versa. However, the fact that the labels assigned to a, b, v when applying the algorithm on $[F, T^{\Delta}]$ are such that $\ell(a) < \ell(v) < \ell(b)$, and the fact that (a, b, v) is a triangular face of $[F, T^{\Delta}]$ imply that no vertex of $[F, T^{\Delta}]$ different from v has been assigned the same label as v.

The two observations above lead to conclude that the labeling of $[G, \mathcal{H}^{\mathcal{A}}]$ still satisfies the main property that allows to construct a planar embedding on the point set S, which we formalize as follows.

Lemma 13.13. *Let $H_i^{\mathcal{A}}$ be the restricted subgraph of $\mathcal{H}^{\mathcal{A}}$ for some $1 \leq i \leq |G|$. Then $H_i^{\mathcal{A}}$ is a tree whose all the vertices have degree at most 2, except for at most one that might have degree 3.*

Hence, we can still apply Theorem 13.8 to obtain a planar embedding $\Gamma^{\mathcal{A}}$ of $[G, \mathcal{H}^{\mathcal{A}}]$ on point set S. Note that $[G, \mathcal{H}^{\mathcal{A}}]$ has the same number of vertices as $[G, \mathcal{H}]$.

In the following subsection we describe **Part C** of our algorithm, namely we show that $\Gamma^{\mathcal{A}}$ can be transformed into a planar embedding Γ of $[G, \mathcal{H}]$ on S^*.

13.3.3 Transformation of the Embedding

The transformation is performed along the following steps.

Removing triangulation edges. First, remove all the triangulation edges added in the different steps of the augmentation of $[G, \mathcal{H}]$. This results in a planar embedding of a graph that is composed of a subdivision of the block component $[G, \mathcal{H}'']$ plus a set of paths, each attached to a vertex of $[G, \mathcal{H}'']$; such paths, together with the subdivided edges, represent the tree components in $\mathcal{T}_B$.

Recovering the tree components We transform the drawings of such paths and of the subdivided edges in order to obtain a drawing Γ' of a graph that

is composed of $[G, \mathcal{H}']$ (that is, the original graph $[G, \mathcal{H}]$ without the edges in $E_P \cup E_B$) plus some edges of $E_P \cup E_B$ (namely those connecting vertices of G to vertices of the tree components in $\mathcal{T}_B$), as follows. Refer to Figure 13.10.

Let $T_c \in \mathcal{T}_B$ be a tree component and let P_c be the path representing it (either a path attached to a vertex c of $[G, \mathcal{H}'']$ or a path induced by some subdivision vertices of a subdivided edge). As observed above, path P_c consists of only non-fork-vertices. Also, all the vertices of P_c have the same label i, by construction. Hence, all of such vertices are placed on the points of a segment $s \in \{s^+, s^N, s^-\}$ of S_j, where p_j is the point vertex v_i is placed on.

We remove all the internal edges of P_c and move each vertex x of P_c from the point p of s it lies on to one of the corresponding petal points, either $l(p)$ or $r(p)$, as follows. Let v be a vertex of G connected to a vertex of T_c by an edge in $E_P \cup E_B$, if any; recall that, by Lemma 13.12, all the edges of $E_P \cup E_B$ connecting T_c to G are incident to v. If $\ell(x) > \ell(v)$, then move x to $l(p)$; see the tree component connected to v in Figure 13.10(b). If $\ell(x) < \ell(v)$, then move x to $r(p)$; see the tree components connected to w in Figure 13.10(a) and (b). Otherwise, $\ell(x) = \ell(v)$; we note that in this case $s \neq s^N$, by construction, and hence we have to distinguish the following two cases: If $s = s^+$, then move x to $l(p)$ (see the tree components attached to $c = b$ connected to v in Figure 13.10(a)); while if $s = s^-$, then move x to $r(p)$ (see the tree components attached to $c = a$ connected to v in Figure 13.10(a)). If no vertex $v \in G$ is connected to T_c, then we move x to $r(p)$ if $\ell(c) < \ell(x)$ (see the tree component attached to $c = a$ in Figure 13.10(a)), and to $l(p)$ otherwise.

We prove that this operation allows for a planar drawing of each edge incident to a vertex of T_c. As for the internal edges of T_c, this is due to the fact that the petal points, together with the point on which vertex c has been placed, form a convex point set; hence, it is possible to construct a planar embedding of T_c on such points [22]. As for the edges connecting vertices of T_c to v recall that, by Lemma 13.11, v has visibility to the root c of T_c, since (v, c) is a triangulation edge; also, by Property 13.1, this visibility from v extends to all the points of S the vertices of P_c had been placed on; finally, by the construction of S^*, we have that v has also visibility to all the petal points where the vertices have been moved. In fact, consider the triangle $\triangle p_i p_{i-1} q_i$ that was used to place the petal points, where p_i and p_{i-1} are two consecutive points on the segment $s \in \{s^-, s^+, s^N\}$ where the dummy vertices are placed, and q_i is the intersection point between segments $s(p_i p_N)$ and $s(p_{i-1} p_{j+1})$, assuming $s = s^+$, the other cases being analogous. Since $s = s^+$ implies that v lies on the half-circle π on between p_{j+1} and p_N, and since v has visibility on both p_i and p_{i-1}, as observed before, it follows that v has visibility to all the points of $\triangle p_i p_{i-1} q_i$, and hence to the petal point inside it.

To conclude the construction of Γ', we reintroduce all the edges (a, b) such that there existed a subdivided edge between a and b. We prove that this does not introduce any crossing. Let v and w be the two vertices of G that

are connected to both a and b. Recall that all the subdivision vertices of (a, b) correspond to vertices of tree components belonging to faces (a, b, v) and (a, b, w). If $\ell(a) = \ell(b)$ (see Figure 13.10(b)), then for each tree component T_c belonging to face either (a, b, v) or (a, b, w), the vertices of P_c lie on the segment s^N corresponding to $\ell(a) = \ell(b)$, by construction, since they are non-fork-vertices on the path between a and b and have label $\ell(a) = \ell(b)$. Also, both a and b lie on s^N, possibly at its extremal points. Since, by construction, all the tree components that are connected to v (to w) through edges of $E_P \cup E_B$ are moved to petal points lying inside triangle $\triangle(a, b, v)$ (triangle $\triangle(a, b, w)$), and since no tree component stays on s^N, edge (a, b) does not cross any edge. If $\ell(a) \neq \ell(b)$, the fact that edge (a, b) does not cross any edge again depends on the labels we assigned to the tree components belonging to faces (a, b, v) and (a, b, w). Namely, assume that $\ell(a) < \ell(b)$ and that a is the parent of b (see Figure 13.10(a)), the other cases being analogous. As observed above, either the tree components belonging to (a, b, v) have label $\ell(b)$ and those belonging to (a, b, w) have label $\ell(w)$, or the tree components belonging to (a, b, v) have label $\ell(v)$ and those belonging to (a, b, w) have label either $\ell(b)$. We prove the claim in the latter case (as in the figure), the other being analogous. Note that, for each tree component T_c belonging to face (a, b, w), all the vertices of P_c lie on the segment s^N corresponding to $\ell(b)$, by construction, since they are non-fork-vertices on the path between a and b and have label $\ell(b)$. Hence, Property 13.1 ensures that they lie inside triangle $\triangle(a, b, w)$, which implies that the corresponding petal points lie inside $\triangle(a, b, w)$, as well. The fact that the tree components T_c lying inside face (a, b, v) are also placed on petal points lying inside triangle $\triangle(a, b, v)$ trivially follows from the fact that the vertices of P_c have label $\ell(v)$.

Inserting Petal Edges and Big Face Edges To complete the transformation it remains to insert the edges of $E_P \cup E_B$ which were not inserted in the previous step, namely those connecting a vertex of G to a vertex of the block-component $[G, \mathcal{H}'']$. However, since by Lemma 13.11 all of such edges were also triangulation edges, their insertion does not produce any crossing.

As a result of the above transformations we obtain a planar embedding Γ of $[G, \mathcal{H}]$ on point set S^*.

All the discussion in this section leads to the following result.

Theorem 13.14. *There exists a universal point set of size $O(n^{3/2})$ for the class of n-vertex 2-outerplanar graphs $[G, \mathcal{H}]$ where $\mathcal{H}$ is a forest.*

13.4 General 2-Outerplanar Graphs

This section is devoted to extend the result presented in Theorem 13.14 to any arbitrary 2-outerplanar graph $[G, \mathcal{H}]$. The idea is to convert every graph $G_h \in \mathcal{H}$ lying in a face F_h of G into a tree T_h, where T_h and G_h have the same

vertex set; embed the resulting graph on S^*; and finally revert the conversion from each T_h to G_h. Similar to the previous sections, we describe all the steps for a single subgraph $[F = F_h, G_h]$.

We say that a cut-vertex of G_h is a *c-vertex*, and that the vertices and the edges of a block B of G_h are its *block vertices*, denoted by N_B, and its *block edges*, denoted by $E_{BL} \subseteq N_B \times N_B$, respectively. Now we transform graph $[F, G_h]$ into a cycle-tree graph $[F, T]$ as follows: For each block B of G_h, we remove all its block edges E_{BL} and insert a *b-vertex* b representing B; also, we insert edges (b, b') for every vertex $b' \in N_B$. In other words, we replace each block B with a star whose center is a new vertex b and whose leaves are the vertices in N_B. This results in transforming G_h into a tree T obtained by attaching the stars through the identification of leaves corresponding to c-vertices. When performing the transformation, we start from the given planar embedding Γ of $[G, \mathcal{H}]$, which naturally induces a planar embedding Γ' of each resulting cycle-tree graph $[F, T]$.

We apply the operations described in **Part A** of Section 13.3.2 (delete petal and big-face edges, remove tree components, and triangulate) to make $[F, T]$ inner-triangulated, and then label it as in Section 13.2.2. We then relabel some of the c-vertices and perform the merging of the tree components in a special way, slightly different from the one described in **Part B**, so that the embedding of the resulting graph will satisfy some additional geometric properties that will allow us to restore the original blocks of G_h when performing **Part C**.

Let $w_1, \ldots, w_m$ be the vertices of F in the order defined by the labeling, and let r be the root of T; recall that, since the root is a fork vertex, it is independent of where the tree components, which become non-fork-vertices, are merged. We give some additional definition. For a b-vertex b we define two particular vertices, called its *opener* and the *closer*, that will play a special role in the merging of the tree components incident to b. If $b \neq r$ and b is not adjacent to r, then the opener of b is the c-vertex c that is the parent of b in T. If $b = r$ (see Figure 13.11(a)), then the opener of b is the c-vertex c adjacent to b, w_1, and w_m, such that 3-cycle (c, w_1, w_m) does not contain in its interior any c-vertex with the same property as c in Γ'. If b is adjacent to r, then the opener of b is r; note that, in this way we treat r as a c-vertex even when it is not a cut-vertex of G_h. For a b-vertex b with opener c, the *closer* of b is the first block vertex following (the last preceding) c in the rotation at b in Γ', if $\ell(c) < \ell(b)$ (if $\ell(c) \geq \ell(b)$); note that, the closer always exists since b has at least two neighbors that are not incident to F.

Some blocks of G_h, and the corresponding b-vertices of T, have to be treated in a special way because of their relationship with the root r of T. Let c be the opener of a b-vertex b such that $N_B \cup \{b\}$ contains r, where B is the block of G_h corresponding to b. We call *root-blocks* the set of blocks lying in the interior of 3-cycle (c, w_1, w_m) in Γ'. If c is a non-fork vertex, the presence of root-blocks might create problems in the algorithm we are going to describe later; hence,

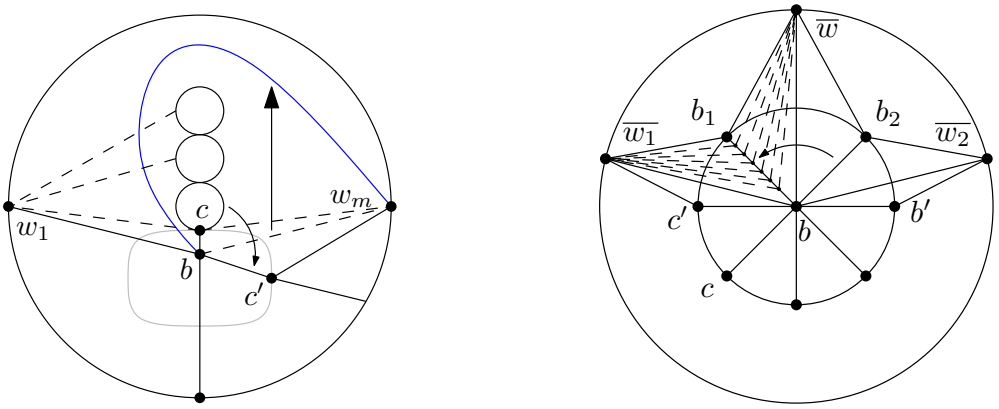


FIGURE 13.11: (a) Rerouting edge (w_m, b) to eliminate root-blocks when the opener c of the block containing the root is a non-fork vertex. (b) Illustration for the rule "choice of Faces".

in this case, we change the embedding Γ' slightly (cf. Figure 13.11(a)) by rerouting edge (w_m, b) so that root-blocks do not exist any longer. This change of embedding consists of swapping edges (b, c) and (w_m, b) in the rotation at b. Note that edge (w_m, b) does not belong to $[F, G_h]$, which implies that embedding Γ has not been changed. In order to maintain planarity, we have to remove all the edges connecting w_1 to root-blocks, as otherwise they would cross edge (w_m, b); however, the fact that (w_m, b) does not belong to $[F, G_h]$, together with a visibility property between w_1 and the root-blocks that we will prove in Lemma 13.16, will make it possible to add the removed edges at the end of the construction without introducing any crossing.

We now describe the part of the algorithm that differs from the one described in Section 13.3.

First, we change the labeling of each c-vertex c that is a branch vertex of T. Namely, consider the two fork-vertices a and d such that the subpath of T between a and d contains c and does not contain any other fork vertex, with a being closer to the root than d. Let v and w be the two neighbors of c in F; assume $\ell(w) < \ell(v)$. Note that, as described in **Part B** of Section 13.3.2, we have either $\ell(w) < \ell(d) < \ell(v) \leq \ell(a)$ or $\ell(a) \leq \ell(w) < \ell(d) < \ell(v)$. In the first case, we relabel c by setting $\ell(c) = \ell(v)$, otherwise we set $\ell(c) = \ell(w)$. Observe that this is analogous to considering c as a tree component and applying for it the labeling algorithm in Section 13.3.2. This observation allows us to state that the same arguments as in Lemma 13.13 can be used to prove that the restricted subgraph H_i of G_h, for each $i = 1, \ldots, |G|$, maintains the same property even after the relabeling of c.

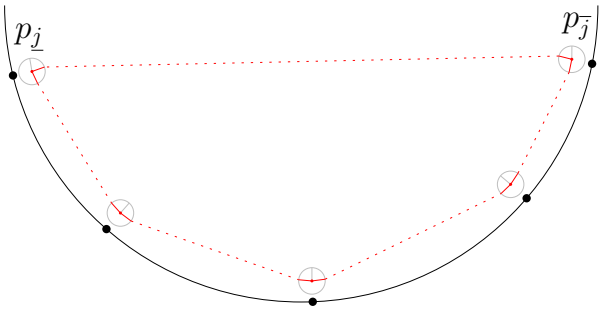


FIGURE 13.12: Illustration for Property 13.15.

Then, we describe a procedure, that we call **Part B'** as it coincides with **Part B** of Section 13.3.2, except for the choice of the face where the tree components are placed and of the edge they are merged to. This choice, that we describe later, is done in such a way that applying **Part C** of the embedding algorithm described in Theorem 13.14 yields an embedding Γ^* of $[F,T]$ on S^* that satisfies the following two properties, which will then allow us to redraw all the blocks of G_h:

- the block vertices of every block form a convex region and
- the clockwise order in which the block vertices of every block appear along this convex region coincides with the clockwise order in which they appear along the outer face of the block in the drawing Γ of G.

For ensuring the first item, the following important property derived from Property 13.1 is of particular help. Refer to Figure 13.12.

Property 13.15. *Let $\underline{j}$ and $\overline{j}$ be two integers such that $1 \leq \underline{j} < \overline{j} \leq N$. Then the points of $\bigcup_{j=\underline{j},\ldots,\overline{j}}[s_j^- \cup \{p_j^C\} \cup s_j^+]$ determine a convex point set. This is also true if we replace $s_{\underline{j}}^-$ by $s_{\underline{j}}^N$ and $s_{\overline{j}}^+$ by $s_{\overline{j}}^N$.*

Proof. First observe that the center points p_j^C of all the point sets between $\underline{j}$ and $\overline{j}$, that is, $\bigcup_{j=\underline{j},\ldots,\overline{j}}\{p_j^C\}$ are in convex position by construction.

Then, for each $j = \underline{j},\ldots,\overline{j}-1$, segments s_j^+ and s_{j+1}^- lie below the segment $s(p_j^C, p_{j+1}^C)$, due to the fact that points p_j^+ and p_{j+1}^- lie below points p_j^1 on π_j and p_{j+1}^2 on π_{j+1}, respectively; see Figure 13.1. This implies that the internal angles at p_j^+ and p_{j+1}^- are smaller than 180°. As for the internal angle at each center point p_j^C, this is still smaller than 180° due to the fact that p_j^+ and p_j^- lie above points p_j^3 and p_j^4 on π_j, respectively, which lie on a diameter of π_j.

The fact that segments either $s^-_{\underline{j}}$ or $s^N_{\underline{j}}$, and either $s^+_{\overline{j}}$ or $s^N_{\overline{j}}$ do not destroy the convexity of the point set again descends from the fact that the internal angles at $p^C_{\underline{j}}$ and at $p^C_{\overline{j}}$ are always smaller than 180°. □

The second item can be mostly ensured by choosing an appropriate face for the tree components. In fact, as already noted in Section 13.3, the triangulation step performed after the removal of tree components splits the face where each tree component used to lie into several faces; while in Section 13.3 the choice among these faces was arbitrary, in this case we have to make a suitable choice, which will be based on the opener and the closer of the block the tree component belongs to.

Rule "choice of Faces":

Let b be a b-vertex of a block B, and let c and c' be the opener and the closer of b, respectively. Also, let b' be the last counterclockwise neighbor of b different from c such that $b' \in N_B$ and $\ell(b') = \ell(b)$ (possibly, $b' = c'$).

Consider any two neighbors b_1 and b_2 of b such that $b_1, b_2 \in N_B$ and there exists no vertex $b_3 \in N_B$ of b between b_1 and b_2 in the rotation at b. Since $[F, T]$ is inner-triangulated, there exists a vertex $\overline{w} \in F$ that is adjacent to both b_1 and b_2; also, there exists edge $(b, \overline{w})$, which is a triangulation edge. Hence, each tree component $T_{1,2}$ that used to lie between b_1 and b_2 has to be placed either inside face $(b, b_1, \overline{w})$ or inside $(b, b_2, \overline{w})$ in order to maintain the embedding of the graph before the triangulation. Finally, let $\overline{w}_1$ and $\overline{w}_2$ be the two vertices of F preceding b_1 and following b_2 in the rotation at b, respectively.

If both b_1 and b_2 are between b' and c' in the rotation at b, then place $T_{1,2}$ inside face $(b, b_2, \overline{w})$ and merge it to edge (b, b_2), that is, subdivide this edge with $|T_{1,2}|$ dummy edges, each connected to $\overline{w}$ and to $\overline{w}_2$; otherwise, place $T_{1,2}$ inside face $(b, b_1, \overline{w})$ and merge it to edge (b, b_1), connecting the subdivision edges to $\overline{w}$ and to $\overline{w}_1$; see Figure 13.11(b).

Let $[F, T^*]$ be the cycle-tree graph obtained after all the tree components have been merged. In the following lemma we prove that $[F, T^*]$ admits an embedding on S^* satisfying the required geometric properties.

Lemma 13.16. *There exists an embedding Γ^* of $[F, T^*]$ on S^* in which, for each b-vertex b corresponding to a block B of G_h, the vertices of N_B are in convex position and appear along this convex region in the same clockwise order as they appear along the outer face of B in the given planar drawing Γ of G.*

Proof. First, construct a straight-line planar embedding Γ'' of $[F, T^*]$ on S^* by applying Theorem 13.14.

We will now consider each block B represented by a b-vertex b in T^* and analyze where the vertices N_B are placed in Γ'' due to **Part C** of Theorem 13.14 and

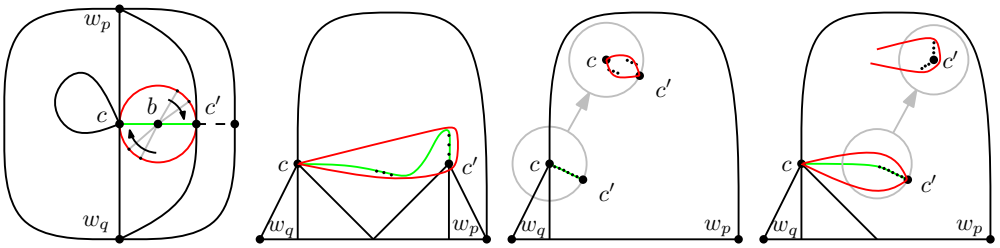


FIGURE 13.13: Illustration when b is non-fork vertex. The red circle indicates the block, tiny black vertices are from tree-components, green edges are tree-edges and the dashed edge is present if c' is fork vertex, otherwise c' is non-fork vertex. The block illustrated in (a) is placed as in (b), if c' is fork vertex. (c) illustrates the case $\ell(c) \neq \ell(c')$ when a promotion of c' is not necessary, while in (d) a promotion is necessary.

to the rule "choice of faces" described in **Part B'**, proving that the vertices in N_B either already satisfy the required properties or can do so by performing some local changes to Γ''.

The block vertices N_B consist of the fork-vertices N_f, of the non-fork-vertices N_{tc} obtained by merging tree components, and of the other non-fork-vertices N_{nf}, which are also non-fork-vertices of $[F, T]$. Note that sets N_f, N_{nf}, and N_{tc} are disjoint, if we consider the root of a tree component not in N_{tc}.

We start with removing b and its incident edges. Note that, in the local changes we possibly perform, the position of b might be reused by another vertex. As orientation help we sometimes keep b on its point, in particular in illustrations, until all its block vertices have been considered.

First suppose that B is a root-block. Recall that the c-vertex c^* separating the root-blocks from the block containing the root r is a fork vertex, since in the case it was a non-fork vertex we rerouted edge (w_m, b), hence eliminating the root-blocks. Thus, all the vertices of the root-blocks have the same label as c^* and are placed on the s_j^N segment of the point set S_j where c^* is placed. Since each vertex x of B in N_{tc} is moved to a petal point of s_j^N by the algorithm described in **Part C**, and since the petal points of the same segment are in convex position, by construction of S^*, the vertices of B satisfy the required properties.

Assume now that B is not a root-block. We distinguish two cases, based on whether b is a fork vertex or not. Let c and c' be the opener and the closer of b, respectively, and assume $\ell(c) \geq \ell(b)$ (the other case is symmetric). Refer to Figure 13.13. Let j and k be the indexes such that c is placed on point set S_j and c' is placed on point set S_k.

Suppose b is a non-fork vertex, and let w_p, w_q (with $p < q$) be the neighbors of b in F. Refer to Figure 13.13.

First note that, in this case, c' is the only vertex of $N_B \setminus \{c\}$ belonging to $N_f \cup N_{nf}$, that is, all the vertices in N_B different from c and c' belong to some tree components. Also, we have $\ell(c) \geq \ell(x) \geq \ell(c')$ for all $x \in N_B$. See Figure 13.13(a).

If $c' \in N_f$, then c' is placed on the center point p_k^C of S_k, as in Figure 13.13(b). We have that the vertices of N_B that have been merged to edge (b, c') are placed on the s_k^N segment of S_k, since the algorithm described in **Part C** moved the vertices adjacent to w_p inside triangle (c, c', w_p); also, the vertices of N_B that have been merged to edge (b, c) are placed on the s_l^- segment of a point set S_l such that $k < l \leq j$, since the vertices adjacent to w_q were moved inside triangle (c, c', w_q). Hence, Property 13.15 ensures that the vertices of N_B are in convex position. The fact that they appear in the correct order along this convex region depends on the fact that the vertices merged to (b, c'), as well as those merged to (b, c), are consecutive along the boundary of B.

If $c' \in N_{nf}$, then c' is placed on the s_k^- segment of S_k. If $j = k$, as in Figure 13.13(c), then c is either on s_k^- or on p_k^C; in both cases, the vertices in N_B are on the same segment, and the proof that they satisfy the required properties, after they have been moved to petal points, is the same as for the case of the root-blocks. If $j > k$, as in Figure 13.13(d), which can only happen if c is a fork vertex, then all the points of N_B, except for c, lie on s_k^-, while c lies on p_j^C. This implies that the region defined by the points of N_B is not convex. We thus need to perform a local change in the placement of these vertices, that we call a *promotion* of c' at S_k. This operation places c' on p_k^C, and places on s_k^N the vertices of N_B that were merged to (b, c'), and on s_k^+ the vertices of N_B that were merged to (b, c). Intuitively, this corresponds to "promoting" c' to become a fork vertex. Note that, no vertex lies on p_k^C before the promotion of c', since there is no fork vertex between c and c' in T^*, and this implies that no vertex lies on s_k^N and s_k^+, as well. By Property 13.15, the vertices of N_B are now in convex position and in the correct order, as in the case in which c' is a fork vertex.

Suppose b is a fork vertex, and let w_p, w_q (with $p < q$) be the two extremal neighbors of b in F. Refer to Figure 13.14.

Let a be the ancestor of b in T such that a is a fork vertex and there exists no fork vertex in the path of T^* between a and b. Note that, a might either coincide with c or it might be the b-vertex or the opener of an ancestor block $\overline{B}$ of B. In any case, vertex a always exists, as the root r is a fork vertex, except for the case in which b itself is the root. This special case $b = r$ will be considered at the end of the proof. Also note that a is adjacent to both w_p and w_q, and we have $\ell(a) \geq \ell(x) \geq \ell(c')$ for all $x \in N_B$.

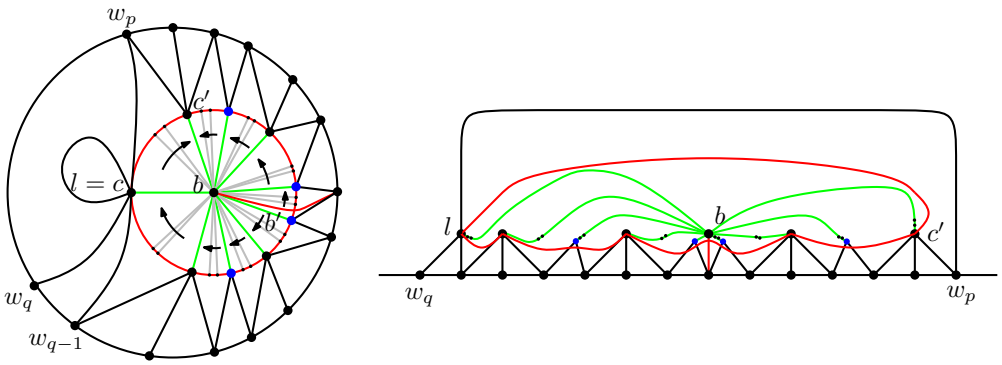


FIGURE 13.14: Illustration when c is fork vertex. The red circle indicates the block, black(blue) vertices on this circle are fork(non-fork)-vertices. Arrows indicate to which edge tree-components are assigned. The right drawing simulates the placement on the point set.

We claim that $\ell(c) \geq \ell(x) \geq \ell(c')$ for all $x \in N_B$. Namely, if c is a fork vertex, then $c = a$ and the claim trivially follows; while if c is a non-fork vertex, then it is a branch vertex (since it has at least a fork vertex descendant, namely b), and hence it has been relabeled so that $\ell(c) = \ell(w_q)$.

We then claim that, for each point set S_l with $k < l \leq j$, there exists no vertex of N_B lying on segment s_l^N. Namely, the embedding algorithm places a vertex z on the s_l^N segment only if z is a branch vertex of T; however, this implies that there exists at least a child block of B attached to z, and hence z is the opener of this block. Thus, z has been relabeled and does not lie on s_l^N.

Finally, we consider the placement of c' and of the tree components merged to edge (b, c'). If c' is a fork vertex, then c' lies on p_k^C, the vertices of N_{tc} adjacent to w_p are on s_k^N, and the other vertices of N_{tc} are either on s_k^+ or on a segment $s_{k'}^-$, for some $k' > k$, by the algorithm described in **Part C**. If c' is a non-fork vertex, then c' lies on s_k^-, together with all the vertices of N_{tc} that have been merged to (b, c'). We hence perform a promotion of c' at S_k, moving c' to p_k^C, the vertices of N_{tc} adjacent to w_p to s_k^N, and the other vertices of N_{tc} to s_k^+. As in the previous case, there was no vertex of N_B placed on p_k^C before promoting c'; in this case, however, we have to consider the possibility that vertex b was placed on p_k^C. Since b has been removed, p_k^C is again free, but a vertex of N_B might still lie on s_k^+, namely b'. This does not affect the possibility of performing the promotion of c', as we have only to ensure that b' is moved on s_k^+ far enough from p_k^C so that the other vertices of N_B that are moved to that segment can fit. This is always possible since s_k^+ contains $\overline{n}$ points, where either $\overline{n} = \sqrt{n}$ or $\overline{n} = n$, and there exist at most $\overline{n}$ vertices in total on S_k.

The two claims above, together with the discussion about c', make it possible to apply Property 13.15 to prove that the vertices of N_B are in convex position.

In the following we prove that they appear along this convex region in the correct order. First note that the vertices in $N_f \cup N_{nf}$ are in the correct order, by construction. As for the vertices in N_{tc}, the algorithm in **Part C** places each set of vertices belonging to the same tree component T_b between the two vertices of $N_f \cup N_{nf}$ incident to the face to which the vertices of T_b have been assigned by the rule "choice of faces" in **Part B'**. The only exception concerns the vertices merged to (b, c') that are adjacent to w_p, as these vertices are on s_k^N; however, this is still consistent with the order in which the vertices of N_B appear along the boundary of B.

This concludes the proof of the lemma. □

By Lemma 13.16 the block vertices of every block are in convex position. Since every convex point of size n set is universal for n-vertex outerplanar graphs [25, 99], we can now insert all block edges E_{BL} in Γ'' without introducing any crossing. The resulting drawing is a planar embedding of $[F, G_h]$ on S^*, which proves the following.

Theorem 13.17. *Any 2-outerplanar graph admits a planar straight-line embedding on a point set of size $O(n^{3/2})$.*

Using the technique of Angelini et al. [7] we can reduce the size of S^* to $O(n(\frac{\log n}{\log\log n})^2)$, and using the technique of Bannister et al. [11] we can even reduce it to $O(n \log n)$. This is done by considering a larger number of classes of point sets, not only dense and sparse ones, to fit the restricted subgraphs of our graph. We formalize the result obtained with the technique of Bannister et al. [11] in the following theorem, which states the final result of this chapter.

Theorem 13.18. *There exists a universal point set of size $O(n \log n)$ for the class of n-vertex 2-outerplanar graphs.*

Proof. Bannister et al. [11] proved that there exists a sequence ξ of integers ξ_j, with $\sum_{j=1,\ldots,n} \xi_j = O(n \log n)$, that satisfies the following property. For each finite sequence $\alpha_1, \ldots, \alpha_k$ of integers such that $\sum_{i=1,\ldots,k} \alpha_i = n$, there exists a subsequence $\beta_1, \ldots, \beta_k$ of the first k elements of ξ such that, for each $i = 1, \ldots, k$, we have $\alpha_i \leq \beta_i$.

Bannister et al. [11] used this sequence to construct a universal point set of size a $O(n \log n)$ for simply-nested graphs [7]. We use the same technique to construct our universal point set S^*. Namely, for each $j = 1, \ldots, n$, we place ξ_j points on each of segments s_j^-, s_j^+, and s_j^N of S_j, which hence results in a point set of total size $O(n \log n)$. Then, when each vertex $v_i \in G$ has to be placed on a point of the outer half-circle π according to its weight $\omega(v_i)$, we place it on the first free point p_j such that $\omega(v_i) \leq \xi_j$. Since the sum of the weights of the vertices of G is equal to n, by the property of the sequence ξ, we have that all the vertices of G can be placed on S^*. This concludes the proof of the theorem. □

Chapter 14

Short Conclusion on UPS

In this part we proved that there exists a universal point set S^* of size $O(n \log n)$ for the class of 2-outerplanar graphs. It remains open whether our techniques can be extended such that S^* is universal for the class of planar graphs or at least for 3-outerplanar graphs. We are also interested in the required area of a universal point set for 2-outerplanar graphs. In particular since a grid of size $O(n^2)$ is universal for the class of planar graphs, we would like to know whether universal point sets of asymptotically subquadratic size require polynomial or exponential area.

Part IV

Conclusions

Chapter 15

Main Results

We have introduced a new drawing model for graphs based on straight-line drawings and based on orthogonal drawings. We also generalized a drawing model for hypergraphs. For the new drawing models we investigated their usability and focused on the characterization of graphs admitting these drawing models and presented several algorithms for testing and construction. In both models we considered variants with fixed vertex positions building a bridge to the point set embeddability problem, which we solved for 2-outerplanar graphs.

We present a detailed list of all results separated by the parts of the thesis.

Avoiding edge crossings is an important topic in graph drawing. In this work we presented a new approach to avoid edge crossings coming from information visualization and by using the Gestalt principle, called PED. This drawing model provides a large body of research for which we introduced some directions with different settings: we considered

(1) PED for graphs with respect to straight-line drawings and

(2) PED for geometrically embedded graphs, as well as

(3) PED for graphs with respect to orthogonal drawings with one bend per edge.

The following list summarizes these results.

- We have introduced a formal model for each of the three settings, which avoids ambiguity and extracts helpful properties for human understanding, i.e., symmetry and homogeneity.
- We have found many graph classes admitting a 1/4-SHPED and proved for an infinite class of graphs that they do not admit a 1/4-SHPED.

- We have provided a force-directed layout algorithm for producing 1/4-SHPEDs in many cases, applicable for all graphs up to a certain number of vertices.
- We have investigated the human understanding of 1/4-SHPEDs and found out that the new model is not worse than the traditional straight-line drawing model.
- We have shown that geometrically embedded 1- and 2-planar graphs admit a ncPED and provided a sufficient and necessary condition for k-planar geometric graphs. We have also showed with an example that a ncPED does not always exist.
- We have proved the NP-hardness of computing a maxSPED in general, while for geometrically embedded 1- and 2-planar graphs the computation can efficiently be done.
- We have shown that any orthogonal 1-bend drawing can be transformed into a 1-bend OPED or 1-bend HOPED, while for 1-bend SHOPED this is only true for a maximum degree of 3. We presented a constructive algorithm for this case and presented an example of maximum degree 4, not admitting a 1-bend SHOPED.

The second part of this thesis dealt with bus realizations, a new drawing model for hypergraphs also providing reduction of visual clutter. This is a model coming from the VLSI design and was investigated in terms of classification and construction in the following two directions: we have considered

(1) planar 2-dimensional bus realizations for hypergraphs

(2) planar 1-dimensional bus realizations for hypergraphs with fixed hypervertex positions.

The following list summarizes these results.

- We have characterized the class of graphs admitting a planar 2-dimensional bus realization. The decision and implicit construction can be done efficiently in contrast to the general case. This work points out the threshold when this problem becomes hard to solve.
- We have further looked at variants of 2-dimensional bus realization, classified by the type of crossings. Only the extremal class, where all types of crossings are permitted, could be characterized and their decision problem is NP-hard.
- For planar 1-dimensional bus realizations we have pointed out the equality to visibility drawings, whereas the existence of non-planar 1-dimensional bus realizations is trivial.

- We have investigated planar 1-dimensional bus realizations with fixed hypervertex positions and proved NP-completeness even for testing a matching.
- In this setting we have just found a few very restricted variants that can be decided efficiently. Thereby we have extensively discussed the bounds of polynomial time limits for the decision problem.

The final part of this thesis is focused on a universal point set of 2-outerplanar graphs. Our main result is that

- any 2-outerplanar graph admits a planar straight-line embedding on a point set of size $O(n \log n)$.

Chapter 16

Future Work

A lot of open questions evolved during this whole thesis by closing other gaps. Different unanswered questions arise which invite further investigation of the topics. A list of the main interesting questions is given next.

- It would be nice to find criteria or a characterization which graphs admit a 1/4-SHPEDs, i.e., is it possible to see it in the structure of the graph if it admits a 1/4-SHPED?
- On the positive side we would like to find further classes of graphs admitting 1/4-SHPEDs.
- On the negative side we would like to reduce the n such that K_n admits no 1/4-SHPED. In particular we like to see a proof or disproof that K_{17} does not admit a 1/4-SHPED.
- Is there another infinite class of graphs admitting no 1/4-SHPED?
- We would like to improve the 1/4-SHPED spring embedder to be applicable for more, especially larger graphs.
- We would like to investigate in further questions on 1/4-SHPEDs compared to traditional straight-line drawings in order to find a class of questions, that point out the advantages of 1/4-SHPED more strongly, as well as the advantages of the traditional drawing model.
- Another claim that has to be proven or disproved is that computing maxPED is NP-hard.
- In terms of orthogonal drawings we would like to know, whether the approach can be extended to k-bend drawings, where $k > 1$? What about the ambiguity in this case?

- We would like to characterize the orthogonal 1-bend drawings admitting a 1-bend SHOPED.
- Furthermore we want to extend this concept for orthogonal 1-bend drawings to graphs without any degree restriction using the Kandinsky model.
- Additionally we would like to see a formal concept of SHPED for Lombardi drawings or other drawing conventions.

In the topic of bus graphs we are interested in the following research questions.

- We would like to see a full characterization of bus graphs according to the realization they admit.
- What is the complexity of the decision, whether a bus graph with C_j-realization admits a C_i-realization, $i < j$?
- For the generalized bus graph, i.e., with realizations of buses in more than two directions, we would like to know if the decision whether they admit a planar realization lies also in P. Furthermore a characterization of those graphs would be nice as well as a proof or disproof that the decision in the general case is NP-complete.
- When considering BEP it would be nice to state results using only center buses. Additionally the complexity of the general case with more than two points per color and no global minimum distance is of interest.
- We would like to see extensions in terms of points per color and in terms of number of buses per color: what are the properties a point set must fulfill in order to admit a solution for BEP?
- Variants with minimizing the crossings while insisting on one bus per color are interesting in terms of complexity.

In the final topic of UPS we have three main open problems.

- It remains as open problem whether our techniques can be extended, such that our universal point set is also universal for the class of 3-planar graphs or even for all planar graphs.
- In general we are interested whether or whether not there exist a universal point set of subquadratic size for the class of planar graphs.
- What is the minimum area requirement for a universal point set of the class of 2-outerplanar graphs? And what is the minimum area requirement for a universal point set of the class of planar graphs?

Bibliography

[1] Unified Modelling Language. Website. http://www.uml.org/ (accessed February 2015).

[2] E. Ackerman and G. Tardos. On the maximum number of edges in quasi-planar graphs. *Journal of Combinatorial Theory, Series A*, 114(3):563–571, 2007.

[3] A. Ada, M. Coggan, P. D. Marco, A. Doyon, L. Flookes, S. Heilala, E. Kim, J. L. O. Wing, L.-F. Préville-Ratelle, S. Whitesides, and N. Yu. On bus graph realizability. *CCCG*, abs/cs/0609127:229–232, 2007.

[4] AIMMS. Aimms modeling guide - integer linear programming tricks, chapter 7. Website. http://www.aimms.com/ aimms/ download/ manuals /aimms3om_integerprogrammingtricks.pdf (accessed April 2015).

[5] B. Alper, N. H. Riche, G. Ramos, and M. Czerwinski. Design study of LineSets, a novel set visualization technique. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2259–2267, 2011.

[6] P. Angelini, C. Binucci, W. Evans, F. Hurtado, G. Liotta, T. Mchedlidze, H. Meijer, and Y. Okamoto. Universal point subsets for planar graphs. In *Procedings of the 23rd International Symposium on Algorithms and Computation*, volume 7676 of *Lecture Notes in Computer Science*, pages 423–432, 2012.

[7] P. Angelini, G. Di Battista, M. Kaufmann, T. Mchedlidze, V. Roselli, and C. Squarcella. Small point sets for simply-nested planar graphs. In *Graph Drawing - 19th International Symposium*, volume 7034 of *Lecture Notes in Computer Science*, pages 75–85. Springer, 2011.

[8] E. N. Argyriou, A. Symvonis, and V. Vassiliou. A fraud detection visualization system utilizing radial drawings and heat-maps. In *Proceedings of the 5th International Conference on Information Visualization Theory and Applications*, pages 153–160. SciTePress, 2014.

[9] B. Aspvall, M. F. Plass, and R. E. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979.

[10] J. Backus. Can programming be liberated from the von neumann style? a functional style and its algebra of programs. *Communications on the ACM*, 21(8):613–641, 1978.

[11] M. J. Bannister, Z. Cheng, W. E. Devanny, and D. Eppstein. Superpatterns and universal point sets. *Journal of Graph Algorithms and Applications*, 18(2):177–209, 2014.

[12] R. Bar-Yehuda and D. Rawitz. Efficient algorithms for integer programs with two variables per constraint. *Algorithmica*, 29:595–609, 2001.

[13] A.-L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[14] C. Batini, L. Furlani, and E. Nardelli. What is a good diagram? A pragmatic approach. In *Proceedings of the 5th International Conference on Entity-Relationship Approach*, pages 312–319. IEEE Computer Society, 1985.

[15] R. A. Becker, S. G. Eick, and A. R. Wilks. Visualizing network data. *IEEE Transactions on Visualization and Computational Graphics*, 1(1):16–28, 1995.

[16] M. A. Bekos, S. Cornelsen, M. Fink, S. Hong, M. Kaufmann, M. Nöllenburg, I. Rutter, and A. Symvonis. Many-to-one boundary labeling with backbones. In *Graph Drawing - 21st International Symposium*, pages 244–255. Springer, 2013.

[17] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers*, 28(9):643–647, 1979.

[18] C. Berge. *Hypergraphs (Combinatorics of Finite Sets)*, volume 45 of *North-Holland Mathematical Library*. North-Holland, 1989.

[19] F. Bertault and P. Eades. Drawing hypergraphs in the subset standard (short demo paper). In *Graph Drawing - 8th International Symposium*, pages 164–169. Springer, 2000.

[20] M. Biddulph. Extracting a social graph from Wikipedia people pages. Website. http://www.hackdiary.com/2012 (accessed February 2015).

[21] T. C. Biedl, P. Bose, E. D. Demaine, and A. Lubiw. Efficient algorithms for Petersen's matching theorem. *Journal of Algorithms*, 38(1):110–134, 2001.

[22] C. Binucci, E. Di Giacomo, W. Didimo, A. Estrella-Balderrama, F. Frati, S. Kobourov, and G. Liotta. Upward straight-line embeddings of directed graphs into point sets. *Computational Geometry: Theory and Applications*, 43:219–232, 2010.

[23] M. Bóna. A survey of stack-sorting disciplines. *Electronic Journal of Combinatorics*, on(2), 2002.

[24] J. A. Bondy. *Graph Theory With Applications*. Elsevier Science Ltd., 1976.

[25] P. Bose. On embedding an outer-planar graph in a point set. *Computational Geometry: Theory and Application*, 23(3):303–312, 2002.

[26] F. Brandenburg, D. Eppstein, M. T. Goodrich, S. G. Kobourov, G. Liotta, and P. Mutzel. Selected open problems in graph drawing. In *Graph Drawing - 11th International Symposium*, volume 2912 of *Lecture Notes in Computer Science*, pages 515–539. Springer, 2003.

[27] F.-J. Brandenburg. Drawing planar graphs on $(8/9)*n^2$ area. *Electronic Notes in Discrete Mathematics*, 31:37–40, 2008.

[28] U. Brandes, S. Cornelsen, B. Pampel, and A. Sallaberry. Blocks of hypergraphs - applied to hypergraphs and outerplanarity. In *Combinatorial Algorithms - 21st International Workshop*, pages 201–211. Springer, 2010.

[29] U. Brandes, S. Cornelsen, B. Pampel, and A. Sallaberry. Path-based supports for hypergraphs. *Journal of Discrete Algorithms*, 14:248–261, 2012.

[30] U. Brandes, C. Erten, A. Estrella-Balderrama, J. J. Fowler, F. Frati, M. Geyer, C. Gutwenger, S.-H. Hong, M. Kaufmann, S. G. Kobourov, G. Liotta, P. Mutzel, and A. Symvonis. Colored simultaneous geometric embeddings and universal pointsets. *Algorithmica*, 60(3):569–592, 2011.

[31] C. Buchheim, M. Chimani, C. Gutwenger, M. Jünger, and P. Mutzel. *Handbook of Graph Drawing and Visualization*, chapter Crossings and Planarization. CRC Press, 2013.

[32] K. Buchin, M. J. van Kreveld, H. Meijer, B. Speckmann, and K. Verbeek. On planar supports for hypergraphs. *Journal of Graph Algorithms and Applications*, 15(4):533–549, 2011.

[33] M. Burch, C. Vehlow, N. Konevtsova, and D. Weiskopf. Evaluating partially drawn links for directed graph edges. In *Graph Drawing - 19th International Symposium*, volume 7034 of *Lecture Notes in Computer Science*, pages 226–237. Springer, 2012.

[34] S. Cabello. Planar embeddability of the vertices of a graph using a fixed point set is NP-hard. *Journal of Graph Algorithms and Applications*, 10(2):353–366, 2006.

[35] D. Chang, L. Dooley, and J. E. Tuovinen. Gestalt theory in visual screen design – a new look at an old subject. In *WCCE2001 Australian Topics: Selected Papers from the Seventh World Conference on Computers in Education*, volume 8 of *CRPIT*, pages 5–12. ACS, 2002.

[36] L. Chapman. Qualtrics taps accel, sequoia for first-ever vc round, wall street journal. Website, 2012. http://blogs.wsj.com/venturecapital/2012/05/15/qualtrics-taps-accel-sequoia-for-first-ever-vc-round/ (accessed March 2015).

[37] H. Chen, C. Qiao, F. Zhou, and C.-K. Cheng. Refined single trunk tree: A rectilinear Steiner tree generator for interconnect prediction. In *Proceedings of the 2002 International Workshop on System-level Interconnect Prediction*, pages 85–89. ACM, 2002.

[38] R. Chernobelskiy, K. I. Cunningham, M. T. Goodrich, S. G. Kobourov, and L. Trott. Force-directed lombardi-style graph drawing. In *Graph Drawing - 19th International Symposium*, volume 7034 of *Lecture Notes in Computer Science*, pages 320–331. Springer, 2011.

[39] M. Chrobak and H. Karloff. A lower bound on the size of universal sets for planar graphs. *SIGACT News*, 20, 1989.

[40] M. Chrobak and S. Nakano. Minimum-width grid drawings of plane graphs. *Computational Geometry*, 11(1):29 – 54, 1998.

[41] C. Collins, G. Penn, and M. S. T. Carpendale. Bubble Sets: Revealing set relations with isocontours over existing visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1009–1016, 2009.

[42] I. F. Cruz and R. Tamassia. Graph drawing tutorial. Website. http://cs.brown.edu/~rt/papers/gd-tutorial/gd-constraints.pdf (accessed February 2015).

[43] W. Cui, H. Zhou, H. Qu, P. C. Wong, and X. Li. Geometry-based edge clustering for graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14:1277–1284, 2008.

[44] H. De Fraysseix and P. O. De Mendez. On topological aspects of orientations. *Discrete Mathematics*, 229(1-3):57–72, 2001.

[45] H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990.

[46] A. M. Dean, W. S. Evans, E. Gethner, J. D. Laison, M. A. Safari, and W. T. Trotter. Bar k-visibility graphs. *Journal of Graph Algorithms and Applications*, 11(1):45–59, 2007.

[47] E. D. Demaine, J. S. B. Mitchell, and J. O'Rourke. The open problems project. Website, 2012. http://cs.smith.edu/~orourke/TOPP/ (accessed February 2015).

[48] A. Desolneux, L. Moisan, and J.-M. Morel. *From Gestalt Theory to Image Analysis: A Probabilistic Approach*. Springer Publishing Company, 1st edition, 2007.

[49] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1st edition, 1999.

[50] G. Di Battista and R. Tamassia. Algorithms for plane representations of acyclic digraphs. *Theoretical Computer Science*, 61(2–3):175 – 198, 1988.

[51] G. Di Battista and R. Tamassia. Incremental planarity testing. In *Foundations of Computer Science*, pages 436–441. IEEE, 1989.

[52] G. Di Battista and R. Tamassia. On-line maintenance of triconnected components with SPQR-trees. *Algorithmica*, 15(4):302–318, 1996.

[53] G. Di Battista, R. Tamassia, and I. G. Tollis. Constrained visibility representations of graphs. *Information Processing Letters*, 41(1):1 – 7, 1992.

[54] E. Di Giacomo, W. Didimo, G. Liotta, and F. Montecchiani. h-quasi planar drawings of bounded treewidth graphs in linear area. In *WG 2012*, Lecture Notes in Computer Science, pages 91–102. Springer, 2012.

[55] E. Di Giacomo, W. Didimo, G. Liotta, and F. Montecchiani. Area requirement of graph drawings with few crossings per edge. *CGTA*, 46(8):909–916, 2013.

[56] M. Dickerson, D. Eppstein, M. T. Goodrich, and J. Y. Meng. Confluent drawings: Visualizing non-planar diagrams in a planar way. *Journal of Graph Algorithms and Applications*, 9(1):31–52, 2005.

[57] W. Didimo. Density of straight-line 1-planar graph drawings. *Information Processing Letters*, 113(7):236–240, 2013.

[58] W. Didimo and G. Liotta. *Thirty Essays on Geometric Graph Theory*, chapter The crossing angle resolution in Graph Drawing. Springer, 2012.

[59] W. Didimo, G. Liotta, F. Montecchiani, and P. Palladino. An advanced network visualization system for financial crime detection. In *Pacific Visualization Symposium*, pages 203–210. IEEE, 2011.

[60] R. Diestel. *Graph theory (Graduate Texts in Mathematics)*. Springer, third edition, August 2005.

[61] I. Dinur, O. Regev, and C. Smyth. The hardness of 3-uniform hypergraph coloring. In *Proceedings of the 35th ACM Symposium on Theory of Computing*, 2003.

[62] S. Dulucq and O. Guibert. Stack words, standard tableaux and baxter permutations. *Discrete Mathematics*, 157(1–3):91 – 106, 1996.

[63] C. A. Duncan, A. Efrat, S. G. Kobourov, and C. Wenk. Drawing with fat edges. In *Graph Drawing*, volume 2265 of *Lecture Notes in Computer Science*, pages 162–177. Springer, 2001.

[64] C. A. Duncan, D. Eppstein, M. T. Goodrich, S. G. Kobourov, and M. Nöllenburg. Lombardi drawings of graphs. *Journal of Graph Algorithms and Applications*, 16(1):85–108, 2012.

[65] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[66] A. Efrat, Y. Hu, S. G. Kobourov, and S. Pupyrev. MapSets: visualizing embedded and clustered graphs. In *Graph Drawing*, pages 452–463. Springer, 2014.

[67] M. Eiglsperger, C. Gutwenger, M. Kaufmann, J. Kupke, M. Jünger, S. Leipert, K. Klein, P. Mutzel, and M. Siebenhaller. Automatic layout of UML class diagrams in orthogonal style. *Information Visualization*, 3(3):189–208, 2004.

[68] D. Eppstein. Separating thickness from geometric thickness. In *Graph Drawing - 10th International Symposium*, volume 2528 of *Lecture Notes in Computer Science*, pages 150–161. Springer, 2002.

[69] D. Eppstein. Planar lombardi drawings for subcubic graphs. In *Graph Drawing - 20th International Symposium*, volume 7704 of *Lecture Notes in Computer Science*, pages 126–137. Springer, 2012.

[70] D. Eppstein, M. T. Goodrich, and J. Y. Meng. Delta-confluent drawings. In *Graph Drawing*, volume 3843 of *Lecture Notes in Computer Science*, pages 165–176. Springer, 2005.

[71] D. Eppstein, M. van Kreveld, E. Mumford, and B. Speckmann. Edges and switches, tunnels and bridges. *CGTA*, 42(8):790–802, 2009.

[72] P. Erdős and G. Szekeres. A combinatorial problem in geometry. *Compositio Mathematica*, 2:463–470, 1935.

[73] T. Eschbach, W. Günther, and B. Becker. Orthogonal hypergraph drawing for improved visibility. *Journal of Graph Algorithms and Applications*, 10(2):141–157, 2006.

[74] S. Even and R. E. Tarjan. Computing an st-numbering. *Theoretical Computer Science*, 2(3):339–344, 1976.

[75] I. Fáry. On straight-line representations of planar graphs. *Acta Sci. Math. Szeged*, 11:229–233, 1948.

[76] S. Felsner. Rectangle and square representations of planar graphs. In *Thirty Essays in Geometric Graph Theory*, volume 29 of *Algorithms and Combinatorics*. Springer, 2012.

[77] S. Felsner, É. Fusy, M. Noy, and D. Orden. Bijections for Baxter families and related objects. *Journal of Combinatorial Theory, Series A*, 18:993–1020, 2011.

[78] S. Felsner, C. Huemer, S. Kappes, and D. Orden. Binary labelings for plane quadrangulations and their relatives. *Discrete Mathematics & Theoretical Computer Science*, 12(3):115–138, 2010.

[79] S. Felsner, M. Kaufmann, and P. Valtr. Bend-optimal orthogonal graph drawing in the general position model. *Computational Geometry*, 47(3):460–468, 2014.

[80] S. Felsner and M. Massow. Thickness of bar 1-visibility graphs. In *Graph Drawing - 14th International Symposium*, volume 4372 of *Lecture Notes in Computer Science*, pages 330–342. Springer, 2006.

[81] J. Flower and J. Howse. Generating Euler diagrams. volume 2317 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2002.

[82] J. Flower, J. Howse, and J. Taylor. Nesting in Euler diagrams: syntax, semantics and construction. *Electronic Notes in Theoretical Computer Science*, 72(3):93 – 102, 2003.

[83] J. Flower, P. Rodgers, and P. Mutton. Layout metrics for Euler diagrams. pages 272–280. IEEE Computer Society, 2003.

[84] FOCUS, Technik-Lexikon. BUS (Binary Unit System). Website. http://www.focus.de/digital/computer/technik-lexikon/bus-binary-unit-system_aid_570553.html (accessed March 2015).

[85] M. Formann and F. Wagner. The VLSI layout in various embedding models. In *Graph-Theoretic Concepts in Computer Science, 16rd International Workshop, WG '90*, pages 130–139, 1990.

[86] U. Fößmeier and M. Kaufmann. Drawing high degree graphs with low bend numbers. In *Graph Drawing 1995*, volume 1027 of *Lecture Notes in Computer Science*, pages 254–266. Springer, 1996.

[87] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, 1991.

[88] R. Fulek and C. D. Tóth. Universal point sets for planar three-trees. In *Algorithms and Data Structures - 13th International Symposium, WADS 2013, London, ON, Canada, August 12-14, 2013. Proceedings*, pages 341–352, 2013.

[89] R. Fulek and C. D. Tóth. Universal point sets for planar three-trees. *Journal of Discrete Algorithms*, 30:101–112, 2015.

[90] E. Fusy. *Combinatoire des cartes planaires et applications algorithmiques*. PhD thesis, LIX Ecole Polytechnique, 2007.

[91] P. Gajer and S. Kobourov. Grip: Graph drawing with intelligent placement. *Journal of Graph Algorithms and Applications*, 6, 2002.

[92] J. L. Ganley. Computing optimal rectilinear Steiner trees: A survey and experimental evaluation. *Discrete Applied Mathematics*, 90(1-3):161–171, 1999.

[93] E. R. Gansner, Y. Hu, S. C. North, and C. E. Scheidegger. Multilevel agglomerative edge bundling for visualizing large graphs. In *Proceedings of the 4th IEEE Pacific Visualization Symposium*, pages 187–194, 2011.

[94] E. R. Gansner and Y. Koren. Improved circular layouts. In *Graph Drawing*, pages 386–398. Springer, 2006.

[95] M. R. Garey, R. L. Graham, and D. S. Johnson. The complexity of computing Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 32(4):835–859, 1977.

[96] M. R. Garey and D. S. Johnson. The rectilinear steiner tree problem is NP-complete. *SIAM Journal on Applied Mathematics*, 32(4):826–834, 1977.

[97] A. Granacher. Auflösen von Kantenkreuzungen in nicht planaren Graphen durch partielles Kantenzeichnen. Diploma thesis, University of Konstanz, 2013.

[98] Graph Drawing. *Rome Graph library* and *North Graph library*. Website. http://www.graphdrawing.org/data/ (accessed January 2015).

[99] P. Gritzmann, J. P. B. Mohar, and R. Pollack. Embedding a planar triangulation with vertices at specified positions. *American Mathematical Monthly*, 98:165–166, 1991.

[100] I. Gurobi Optimization. Gurobi optimizer reference manual. Website. http://www.gurobi.com (accessed November 2014).

[101] C. Gutwenger and P. Mutzel. A linear time implementation of SPQR-trees. In *Graph Drawing 2000*, pages 77–90. Springer, 2001.

[102] F. Harary. *Graph Theory*. Addison-Wesley Series in Mathematics. Addison Wesley, 1969.

[103] J. Håstad. Clique is hard to approximate within $n^{1-\varepsilon}$. *Acta Mathematica*, 182:105–142, 1999.

[104] X. He. On finding the rectangular duals of planar triangular graphs. *SIAM Journal on Computing*, 22:1218–1226, 1993.

[105] M. Hirsch, H. Meijer, and D. Rappaport. Biclique edge cover graphs and confluent drawings. In *Graph Drawing*, volume 4372 of *Lecture Notes in Computer Science*, pages 405–416. Springer, 2006.

[106] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transaction on Visualization and Computer Graphics*, 12(5):741–748, 2006.

[107] D. Holten and J. J. van Wijk. Force-directed edge bundling for graph visualization. *Computer Graphics Forum*, 28(3):983–990, 2009.

[108] S. Hong, D. Merrick, and H. A. D. do Nascimento. The metro map layout problem. In *Graph Drawing - 12th International Symposium*, volume 3383 of *Lecture Notes in Computer Science*, pages 482–491. Springer, 2004.

[109] S.-H. Hong, P. Eades, G. Liotta, and S.-H. Poon. Fáry's theorem for 1-planar graphs. In *Computing and Combinatorics Conference 2012*, volume 7434 of *Lecture Notes in Computer Science*, pages 335–346. Springer, 2012.

[110] P. Hui, M. Schaefer, and D. Štefankovič. Train tracks and confluent drawings. In *Graph Drawing*, volume 3383 of *Lecture Notes in Computer Science*, pages 318–328. Springer, 2004.

[111] F. Hurtado, M. Korman, M. J. van Kreveld, M. Löffler, V. S. Adinolfi, R. I. Silveira, and B. Speckmann. Colored spanning graphs for set visualization. In *Graph Drawing - 21st International Symposium*, volume 8242 of *Lecture Notes in Computer Science*, pages 280–291. Springer, 2013.

[112] F. W. Hwang, D. S. Richards, and P. Winter. The Steiner tree problem. *Annals of Discrete Mathematics*, (53), 1992.

[113] IBM Deutschland GmbH. Logo. Website. http://www.ibm.com/de/de (accessed February 2015).

[114] D. S. Johnson and H. O. Pollak. Hypergraph planarity and the complexity of drawing Venn diagrams. *Journal of Graph Theory*, 11(3):309–325, 1987.

[115] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989.

[116] B. Katz, M. Krug, I. Rutter, and A. Wolff. Manhattan-geodesic embedding of planar graphs. In *Graph Drawing, 17th International Symposium, GD 2009*, pages 207–218, 2009.

[117] M. Kaufmann, M. J. van Kreveld, and B. Speckmann. Subdivision drawings of hypergraphs. In *Graph Drawing - 16th International Symposium*, pages 396–407, 2008.

[118] M. Kaufmann and D. Wagner, editors. *Drawing Graphs, Methods and Models*, volume 2025 of *Lecture Notes in Computer Science*. Springer, 2001.

[119] P. Kindermann and J. Spoerhase. Private communication, March 2012.

[120] B. Klemz, T. Mchedlidze, and M. Nöllenburg. Minimum tree supports for hypergraphs and low-concurrency euler diagrams. In *14th Scandinavian Symposium and Workshops on Algorithm Theory*, pages 265–276, 2014.

[121] D. E. Knuth. Computer-drawn flowcharts. *Commununications of the ACM*, 6(9):555–563, 1963.

[122] D. E. Knuth. *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*. Addison Wesley Longman Publishing Co., Inc., 1997.

[123] S. G. Kobourov. Spring embedders and force directed graph drawing algorithms. *CoRR*, abs/1201.3011, 2012.

[124] S. G. Kobourov. *Handbook of Graph Drawing and Visualization*, chapter Force-Directed Drawing Algorithms. CRC Press, 2013.

[125] K. Koffka. *Principles of Gestalt Psychology*. International library of psychology, philosophy and scientific method. Routledge, 1999.

[126] L. T. Kou, G. Markowsky, and L. Berman. A fast algorithm for Steiner trees. *Acta Informatica*, 15:141–145, 1981.

[127] M. R. Kramer and J. van Leeuwen. The complexity of wire–routing and finding minimum area layouts for arbitrary vlsi circuits. In *Advances in Computing Research*, pages 2–129, 1984.

[128] M. Kurowski. A $1.235n$ lower bound on the number of points needed to draw all n-vertex planar graphs. *Information Processing Letters*, 92(2):95 – 98, 2004.

[129] A. Lauer. Kräftebasierter Layoutalgorithmus für Partial Edge Drawings. Bachelor's thesis, Universität Tübingen, Mai 2014.

[130] B. Lee, C. Plaisant, C. S. Parr, J.-D. Fekete, and N. Henry. Task taxonomy for graph visualization. In *Proceedings of the 2006 AVI Workshop on BEyond Time and Errors: Novel Evaluation Methods for Information Visualization*, BELIV '06, pages 1–5. ACM, 2006.

[131] S. Leibßle. Evaluation von Partial Edge Drawings. Bachelor's thesis, Universität Tübingen, April 2015.

[132] T. Lengauer. VLSI Theory. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A)*, pages 835–868. 1990.

[133] M. Löffler and M. Nöllenburg. Planar lombardi drawings of outerpaths. volume 7704 of *Lecture Notes in Computer Science*, pages 561–562. Springer, 2012.

[134] D. Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11(2):329–343, 1982.

[135] L. Lovász. Coverings and colorings of hypergraphs. *Proceedings of the 4th Southeastern Conference on Combinatorics, Graph Theory and Computing, Utilitas Mathematica*, pages 3–12, 1973.

[136] R. Mazza. *Introduction to Information Visualization*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[137] W. Meulemans, N. H. Riche, B. Speckmann, B. Alper, and T. Dwyer. KelpFusion: A hybrid set visualization technique. *Visualization and Computer Graphics, IEEE Transactions on*, 19(11):1846–1858, 2013.

[138] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *Journal of Visual Languages and Computing*, 6(2):183–210, 1995.

[139] B. Mohar. Open problem garden. Website. http://garden.irmacs.sfu.ca (accessed August 2011).

[140] P. Moore and C. Fitz. Using gestalt theory to teach document design and graphics. *Technical Communication Quarterly*, 2(4):389–410, 1993.

[141] T. More. On the construction of Venn diagrams. *Journal of Symbolic Logic*, 24(4):303–304, 1959.

[142] M. Mucha and P. Sankowski. Maximum matchings in planar graphs via Gaussian elimination. *Algorithmica*, 45(1):3–20, 2006.

[143] W. Mulzer and G. Rote. Minimum-weight triangulation is NP-hard. *Journal of the ACM*, 55(2), 2008.

[144] T. Nishizeki and N. Chiba. *Planar Graphs: Theory and Algorithms, Annals of Discrete Mathematics*. Elsevier Science Publishing Company, Inc. New York, 1988.

[145] T. Nishizeki and M. S. Rahman. *Planar Graph Drawing*. World Scientific, 2004.

[146] J. Pach. Every graph admits an unambiguous bold drawing. In *Graph Drawing 2011*, volume 7034 of *Lecture Notes in Computer Science*, pages 332–342. Springer, 2012.

[147] J. Pach and G. Tóth. Graphs drawn with few crossings per edge. *Combinatorica*, 17(3):427–439, 1997.

[148] A. Papakostas and I. G. Tollis. A pairing technique for area-efficient orthogonal drawings. In *Graph Drawing 1996*, pages 355–370, 1996.

[149] D. A. Patterson and J. L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann Publishers Inc., 3rd edition, 2007.

[150] D. Peng, N. Lu, W. Chen, and Q. Peng. SideKnot: Revealing relation patterns for graph visualization. In *Proceewdings of the 5th IEEE Pacific Visualization Symposium*, pages 65–72, 2012.

[151] J. P. C. Petersen. Die Theorie der regulären Graphen (the theory of regular graphs). *Acta Mathematica*, page 15:193–220, 1891.

[152] A. Pierrot and D. Rossin. 2-stack pushall sortable permutations. *CoRR*, abs/1303.4376, 2013.

[153] A. Pierrot and D. Rossin. 2-stack sorting is polynomial. In *31st International Symposium on Theoretical Aspects of Computer Science*, volume 25 of *LIPIcs*, pages 614–626. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014.

[154] S. Porschen and E. Speckenmeyer. Algorithms for variable-weighted 2-SAT and dual problems. In *Proceedings of the 10th International Conference on Theory Application of Satisfiability Testing*, volume 4501 of *Lecture Notes in Computer Science*, pages 173–186. Springer, 2007.

[155] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer, 1985.

[156] H. C. Purchase. Which aesthetic has the greatest effect on human understanding? In *Graph Drawing - 5th International Symposium*, Lecture Notes in Computer Science, pages 248–261. Springer, 1997.

[157] H. C. Purchase, D. A. Carrington, and J.-A. Allder. Empirical evaluation of aesthetics-based graph layout. *Empirical Software Engineering*, 7(3):233–255, 2002.

[158] H. C. Purchase, R. F. Cohen, and M. James. Validating graph drawing aesthetics. In *Graph Drawing 1995*, Lecture Notes in Computer Science, pages 435–446. Springer, 1996.

[159] H. C. Purchase, J. Hamer, M. Nöllenburg, and S. G. Kobourov. On the usability of lombardi graph drawings. In *Graph Drawing - 20th International Symposium*, volume 7704 of *Lecture Notes in Computer Science*, pages 451–462. Springer, 2012.

[160] N. H. Riche and T. Dwyer. Untangling Euler diagrams. *Visualization and Computer Graphics, IEEE Transactions on*, 16(6):1090–1099, 2010.

[161] H. N. Riley. The von Neumann architecture of computer systems. Website, 1987. https://www.cpp.edu/~hnriley/www/VonN.html (accessed March 2015).

[162] P. J. Rodgers, J. Flower, and G. Stapleton. Introducing 3D Venn and Euler diagrams. In *Proceedings of the 3rd International Workshop on Euler Diagrams*, volume 854 of *CEUR Workshop Proceedings*, pages 92–106. CEUR-WS.org, 2012.

[163] P. Rosenstiehl and R. E. Tarjan. Rectilinear planar layouts and bipolar orientations of planar graphs. *Discrete & Computational Geometry*, 1:343–353, 1986.

[164] A. Rusu, A. J. Fabian, R. Jianu, and A. Rusu. Using the gestalt principle of closure to alleviate the edge crossing problem in graph drawings. 0:488–493, 2011.

[165] P. Schnabel. Computer-Architektur, Elektronik Kompendium. Website. http://www.elektronik-kompendium.de/sites/com/1309261.htm (accessed March 2015).

[166] W. Schnyder. Embedding planar graphs on the grid. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, SODA '90, pages 138–148, 1990.

[167] B. Shneiderman and B. B. Bederson. *The Craft of Information Visualization: Readings and Reflections*. Morgan Kaufmann Publishers Inc., 2003.

[168] P. Simonetto and D. Auber. Visualise undrawable euler diagrams. In *IV*, pages 594–599. IEEE Computer Society, 2008.

[169] P. Simonetto, D. Auber, and D. Archambault. Fully automatic visualisation of overlapping sets. *Computer Graphics Forum*, 28(3):967–974, 2009.

[170] S. Smith, R. Smith, and S. Orgill. Qualtrics (private research software company). Website. http://www.qualtrics.com/ (accessed November 2014).

[171] F. Steinhofer. Der Lombardi Code. Website. http://daremag.de/2012/06/der-lombardi-code/ (accessed February 2015).

[172] J. A. Storer. On minimal-node-cost planar embeddings. *Networks*, 14(2):181–212, 1984.

[173] K. Sugiyama. *Graph drawing and applications for software and knowledge engineers*. Series on software engineering and knowledge engineering. River Edge, N.J. World Scientific, 2002.

[174] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man & Cybernetics*, 11(2):109–125, 1981.

[175] A. Suk and B. Walczak. New bounds on the maximum number of edges in k-quasi-planar graphs. In *Graph Drawing*, Lecture Notes in Computer Science, pages 95–106. Springer, 2013.

[176] M. Sysło and A. Proskurowski. On halin graphs. In *Graph Theory*, volume 1018 of *Lecture Notes in Mathematics*, pages 248–256. Springer, 1983.

[177] R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM Journal on Computing*, 16(3):421–444, 1987.

[178] R. Tamassia and I. Tollis. Planar grid embedding in linear time. *IEEE Transactions on Circuits and Systems*, 36(9):1230–1234, 1989.

[179] R. Tamassia and I. G. Tollis. A unified approach a visibility representation of planar graphs. *Discrete & Computational Geometry*, 1:321–341, 1986.

[180] R. E. Tarjan. A Note on Finding the Bridges of a Graph. *Information Processing Letters*, 2(6):160–161, 1974.

[181] C. Taylor. What cleopatra can tell us about social bpm. Website. http://www.successfulworkplace.org/2012/01/22/what-cleopatra-can-tell-us-about-social-bpm-bpm-socialbpm-ces (accessed February 2015).

[182] C. Thomassen. The jordan-schönflies theorem and the classification of surfaces. *Am. Math. Monthly*, 99(2):116–130, 1992.

[183] C. D. Thompson. *A Complexity Theory for VLSI*. PhD thesis, Carnegie-Mellon University, 1980.

[184] W. T. Tutte. How to draw a graph, proceedings of the london mathematical society, 1963.

[185] M. J. van Kreveld. Bold graph drawings. *Computational Geometry*, 44(9):499–506, 2011.

[186] A. Verroust and M. Viaud. Ensuring the drawability of extended Euler diagrams for up to 8 sets. In *Diagrammatic Representation and Inference - 3rd International Conference, Diagrams 2004*, volume 2980 of *Lecture Notes in Computer Science*, pages 128–141. Springer, 2004.

[187] J. von Neumann. The principles of large-scale computing machines. *IEEE Annals of History of Computing*, 10(4):243–256, 1988.

[188] C. Ware, H. C. Purchase, L. Colpoys, and M. McGill. Cognitive measurements of graph aesthetics. *Information Visualization*, 1(2):103–110, 2002.

[189] M. Wertheimer. Untersuchungen zur Lehre von der Gestalt. II. *Psychological ResearchVol. 4, No. 1*, pages 301–350, 1923.

[190] S. K. Wismath. Characterizing bar line-of-sight graphs. In *Proceedings of the First Annual Symposium on Computational Geometry*, SCG '85, pages 147–152. ACM, 1985.

[191] A. Wolff. *Handbook of Graph Drawing and Visualization*, chapter Graph Drawing and Cartography. CRC Press, 2013.

[192] T. Young. *On the Theory of Light and Colours*, volume 92 of *Bakerian Lecture*. W. Bulmer & Company, 1802.

[193] yWorks GmbH. *yFiles* graph library. Website. http://www.yworks.com (accessed February 2015).

[194] H. Zhou, P. Xu, X. Yuan, and H. Qu. Edge bundling in information visualization. *Tsinghua Science and Technology*, 18(2):145–156, 2013.

[195] X. Zhou and T. Nishizeki. Algorithm for the cost edge-coloring of trees. *Journal of Combinatorial Optimization*, 8(1):97–108, 2004.

Publications of the Author

[196] P. Angelini, T. Bruckdorfer, M. Chiesa, F. Frati, M. Kaufmann, and C. Squarcella. On the area requirements of Euclidean minimum spanning trees. In *Algorithms and Data Structures - 12th International Symposium, WADS 2011*, volume 6844 of *Lecture Notes in Computer Science*, pages 25–36. Springer, 2011.

[197] P. Angelini, T. Bruckdorfer, M. Chiesa, F. Frati, M. Kaufmann, and C. Squarcella. On the area requirements of Euclidean minimum spanning trees. *Computational Geometry*, 47(2):200–213, 2014.

[198] P. Angelini, T. Bruckdorfer, M. Kaufmann, and T. Mchedlidze. A universal point set for 2-outerplanar graphs. In *Graph Drawing and Network Visualization - 23rd International Symposium*, pages 409–422, 2015.

[199] M. A. Bekos, T. Bruckdorfer, M. Kaufmann, and C. N. Raftopoulou. 1-planar graphs have constant book thickness. In *23rd Annual European Symposium on Algorithms*, Lecture Notes in Computer Science. Springer, to appear 2015.

[200] T. Bruckdorfer, S. Cornelsen, C. Gutwenger, M. Kaufmann, F. Montecchiani, M. Nöllenburg, and A. Wolff. Progress on partial edge drawings. In *Graph Drawing 2012*, Lecture Notes in Computer Science, pages 67–78, 2012.

[201] T. Bruckdorfer, S. Cornelsen, C. Gutwenger, M. Kaufmann, F. Montecchiani, M. Nöllenburg, and A. Wolff. Progress on partial edge drawings. *CoRR*, abs/1209.0830, 2012.

[202] T. Bruckdorfer, S. Felsner, and M. Kaufmann. On the characterization of plane bus graphs. In *CIAC 2013*, pages 73–84. Springer, 2013.

[203] T. Bruckdorfer, S. Felsner, and M. Kaufmann. Planar bus graphs, submitted for publication 2015.

[204] T. Bruckdorfer and M. Kaufmann. Mad at edge crossings? Break the edges! In *Proceedings of the 6th International Conference on Fun with*

Algorithms, volume 7288 of *Lecture Notes in Computer Science*, pages 40–50. Springer, 2012.

[205] T. Bruckdorfer, M. Kaufmann, S. G. Kobourov, and S. Pupyrev. On embeddability of buses in point sets. In *Graph Drawing and Network Visualization - 23rd International Symposium*, pages 395–408, 2015.

[206] T. Bruckdorfer, M. Kaufmann, and A. Lauer. A Practical Approach for 1/4-SHPEDs. In *IISA 2015, The 6th International Conference on Information, Intelligence, Systems and Applications*, 2015.

[207] T. Bruckdorfer, M. Kaufmann, and F. Montecchiani. 1-bend orthogonal partial edge drawing. *Journal of Graph Algorithms and Applications*, 18(1):111–131, 2014.