New Approaches on Octilinear Graph Drawing

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

vorgelegt von
Dipl.-Inform. (Bioinf.) Robert Krug
aus Ulm

Tübingen
2015

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

Tag der mündlichen Qualifikation	28.04.2015
Dekan:	Prof. Dr. Wolfgang Rosenstiel
1. Berichterstatter:	Prof. Dr. Michael Kaufmann
2. Berichterstatter:	Prof. Dr. Peter Hauck

Acknowledgements

First and foremost I thank my advisor Prof. Dr. Michael Kaufmann, for giving me the opportunity to be part of his great group and write this thesis, for supporting me this whole time and enabling me to visit several international workshops and conferences.
I am deeply grateful to my colleague Dr. Michael Bekos who not only participated in every research project that made it into this thesis but also gave me great support during the writing of this thesis and was a very pleasant officemate.
My thanks also go to my current and former colleagues Till Bruckdorfer, Philip Effinger, Andreas Gerasch, Niklas Heinsohn, Markus Geyer, Stephan Kottler, Martin Siebenhaller and Christian Zielke who provided a very nice working atmosphere and valuable input in many scientific and non-scientific discussions during our coffee-breaks.
I thank all my coauthors, namely Michael Bekos, Markus Geyer, Martin Gronemann, Michael Kaufmann, Thorsten Ludwig, Stefan Näher and Vincenzo Roselli for the pleasant and productive collaborations.
I am most thankful to my friends Chrissi and Klaus who, together with Julian and Laura, always offered me delicious meals and delightful company which I especially appreciated in stressful times of approaching deadlines.
I am very thankful to the proofreaders of this thesis, Michael Bekos and Christian Zielke for their fast and helpful feedback.
Last but not least I thank my parents for supporting my studies.

Zusammenfassung

Graphenzeichnen ist ein Bereich der Informatik mit langer Tradition. Insbesondere im Bereich des orthogonalen Graphenzeichnens wird seit den 1980er Jahren motiviert durch VLSI-Design (Chip-Design) und Grundrissplanung intensiv geforscht. In dieser Arbeit wird das klassische orthogonale Modell durch neue Elemente, unter anderem aus dem oktilinearen Graphenzeichnen, erweitert.

Die ersten Ergebnisse, die wir in dieser Arbeit vorstellen, befassen sich mit oktilinearem Graphenzeichnen. Dieses Modell ist altbekannt und viele Aspekte wurden schon untersucht. Wir entwickeln eine Methode mit der für planare Graphen mit einem beschränkten maximalen Knotengrad (4 und 5) Zeichnungen mit maximal einem Knick pro Kante erstellt werden können. Außerdem zeigen wir, dass Graphen mit maximalem Knotengrad 6 nicht immer mit einem Knick pro Kante gezeichnet werden können. Damit schließen wir die Lücke zwischen bekannten Ergebnissen, die besagen dass Graphen mit maximalem Knotengrad 3 immer ohne Knicke und alle Graphen bis zu einem maximalen Knotengrad von 8 mit höchstens zwei Knicken pro Kante oktilinear gezeichnet werden können.

Durch Nutzerstudien konnte gezeigt werden, dass die Lesbarkeit von (Graphen) Zeichnungen durch Knicke auf den Kanten und schlecht identifizierbare Kreuzungen besonders beeinträchtigt wird. An diesem Punkt setzt unser neues Modell, das abgeschrägt orthogonale (engl. *slanted orthogonal*, oder kurz: *slog*) Graphenzeichnen an. Im slog Modell ist der kleinste erlaubte Winkel zwischen zwei aufeinander folgenden Kantensegmenten $135°$. Das hat zur Folge, dass slog Zeichnungen keine normalen Knicke mehr haben, sondern sogenannte Halb-Knicke. Um Kreuzungen besser erkennbar zu machen sind im slog Modell Kreuzungen ausschließlich zwischen diagonalen Segmenten erlaubt. Wir zeigen, dass eine knick-minimale slog Zeichnung mindestens doppelt so viele Halb-Knicke benötigt, wie eine knick-minimale orthogonale Zeichnung Knicke hat. Für das slog Modell werden in dieser Arbeit Methoden zur Berechnung von knick-minimalen Zeichnungen vorgestellt. Da diese exponentielle Fläche benötigen können, wird außerdem eine Heuristik entwickelt, die nur quadratische Fläche benötigt, dafür aber mehr Knicke zulässt. Die Ergebnisse einer experimentellen Evaluation des slog Modells werden ebenfalls präsentiert.

Im Anschluss erweitern wir das slog Modell zu einer flexibleren Variante die wir *sloggy* nennen. Das sloggy Modell hat alle Eigenschaften des slog Modells, aber Kreuzungen werden jetzt auch zwischen orthogonalen Segmenten erlaubt. Dafür wird die Anzahl Halb-Knicke beschränkt auf genau zwei Mal die Anzahl Knicke der entsprechenden knick-minimalen orthogonalen Zeichnung. Außerdem wird die Anzahl an Kreuzungen zwischen diagonalen Segmenten maximiert. Wir entwickeln eine Methode zur Berechnung solcher Zeichnungen und zeigen, dass auch hier exponentielle Fläche benötigt werden kann.

Das slog und das sloggy Modell sind auf Graphen mit einem maximalen Knotengrad von 4 beschränkt. Deswegen wenden wir uns als nächstes dem Kandinsky Modell zu, einem bekannten Modell mit dem Graphen mit beliebigem Knotengrad gezeichnet werden können. Wir erweitern das bekannte Modell mit Elementen aus dem slog Modell, den Halb-Knicken, um so zuvor verbotene Konfigurationen zeichnen zu können. Mit unserer Erweiterung wollen wir die Gesamtzahl an Knicken und die Größe der Zeichnungen verkleinern. Wir entwickeln eine LP Formulierung, mit der die optimale Zeichnung berechnet werden kann. Da diese sehr lange Zeit zur Berechnung beanspruchen kann, haben wir zusätzliche eine effiziente Heuristik entwickelt. In einer experimentellen Untersuchung vergleichen wir außerdem das neue Modell mit dem klassischen Kandinsky Modell.

Im letzten Kapitel vereinen wir dann unsere Modifikation des Kandinsky Modells mit dem slog Modell im sogenannten *sloginsky* Modell, um Graphen mit beliebigem Knotengrad mit den Vorteilen des slog Modells zeichnen zu können. Wir entwickeln eine Methode zur Berechnung knick-optimaler sloginsky Zeichnungen, aber wir zeigen auch, dass eine solche Zeichnung nicht für jede Eingabe möglich ist. Auch im sloginsky Modell kann eine Zeichnung exponentielle Fläche beanspruchen, was in der experimentellen Evaluation ebenfalls sichtbar wird.

Contents

1 Introduction

Graph drawing has been an important area in computer science for many years. Even if graphs are at first abstract objects modeling relations between entities, visualizing or drawing graphs has developed into a broad field of research. And, while many people do not realize it, graphs are a part of everyday life. For now, it is sufficient to know that a graph consists of a set of *vertices*, which stand for the entities, and a set of *edges* connecting those vertices, which model the relation between the respective entities.

We start with an example that probably everybody has encountered at some point: metro maps. Almost every larger city has a plan of the public transportation system, be it buses, trams or subways. Usually the information about how to get from one place to the other and which line to use can be retrieved the easiest from a map like the one in Figure 1.1. Of course this information could also be conveyed using a large table containing all the stops and the lines connecting them, but by looking at a map it is much easier to figure out how to get to a certain place. Metro map layout is one field in graph drawing that has received much attention in recent years [72, 96, 109]. The concrete problem here is: given a set of metro stops with their geographical positions and a set of lines connecting them, produce a nice drawing where

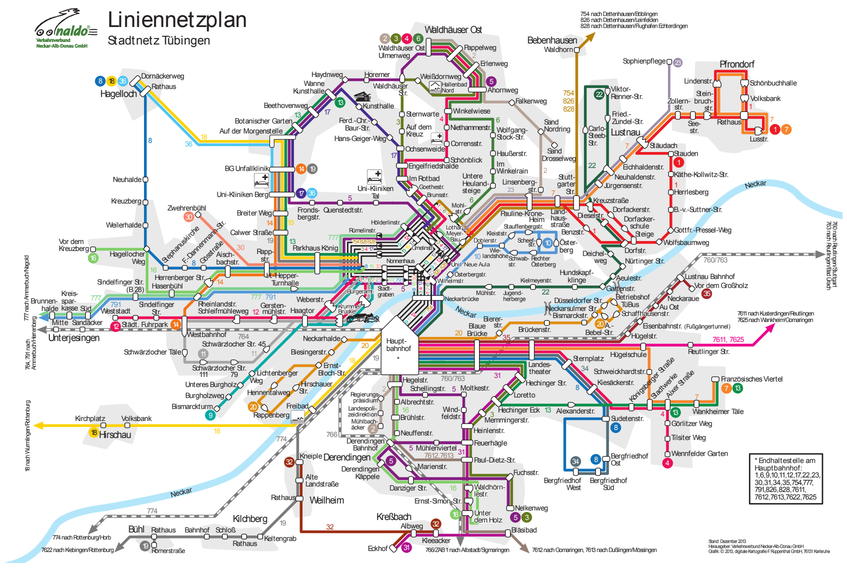


Figure 1.1: Map of the bus net of Tübingen (obtained from `http://www.naldo.de`).

the positions of the stops are as close to the real position as possible and the lines are drawn in a simple-to-follow way that makes it easy to find the connection between two stations in the map. The motivation here is that the actual positions of the stops are not too important as long as the relative positions with respect to the neighboring stops reflect the geographical reality. To make these drawings easy to read and as aesthetically pleasing as possible, it is commonly agreed upon that the number of crossings between lines and the number of bends along lines should be as small as possible. Both criteria have been studied and several results for minimizing the number of crossings [95, 17, 11] and the number of bends [109, 87] are known, as well as hardness results [94, 11]. In fact, the first results we present in this thesis in Section 3, are motivated by minimizing the number of bends in drawings of the octilinear model, which is very often used in metro map layouts.

In general, in the *octilinear graph drawing* model edges are drawn as a combination of horizontal, vertical and diagonal (at 45°) segments (see Figure 1.4d for an example). It has been well studied, not only in the context of metro map layout [72, 96, 109], but also in boundary labeling [16] and wire-routing [90, 91].

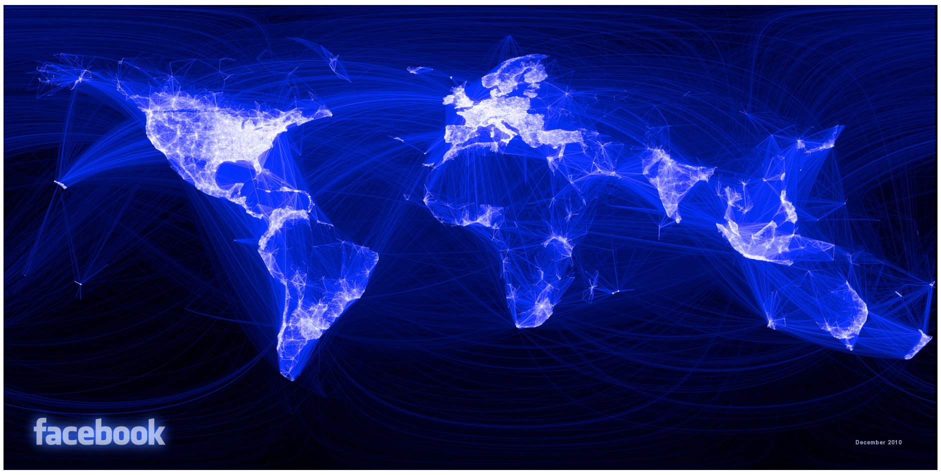


Figure 1.2: A visualization of the relations in facebook (taken from [23]).

The second example where one encounters graphs today is in social networks. In such a network there are participants, the vertices of the graph, and friendships between them, the edges of the graph. The graph drawing challenges in this field are totally different than the ones for metro maps, since social networks usually are very large (millions of vertices and edges). When visualizing social networks rather than optimizing the number of crossings or bends, the standard goal is to visualize the structure of the underlying network. A nice example can be seen in Figure 1.2, a visualization of the well known social network Facebook. Vertices represent places with active Facebook users; the more users, the brighter the dot on the map is drawn. The same holds for the edges, the more connections between two places exist, the brighter the edge is drawn. The result is a drawing in which shapes roughly resembling the continents with active Facebook users appear. This specific visualization was created by Paul Butler, who is a member of Facebooks data infrastructure engineering team, and published on the teams facebook page [23].

Visualizations of social networks are very interesting for researchers in the field of social network analysis [81], who try to find out what the relations between people can tell about a society. Since a lot of information can be obtained from the structure of a social network, much effort has been devoted to obtain meaningful drawings of such graphs [67, 56, 26].

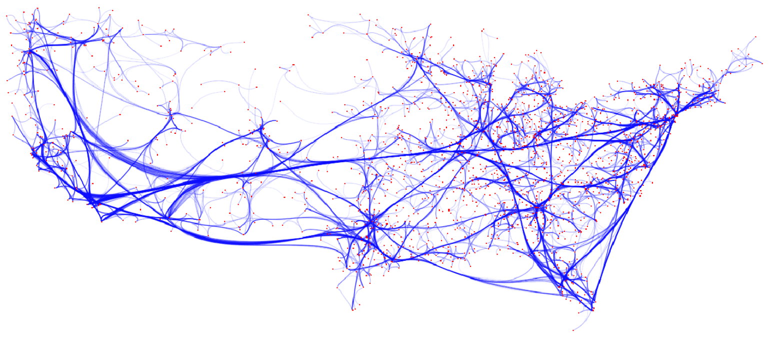


Figure 1.3: An edge-bundling visualization of the network of domestic flights in the US (taken from[82])

Large graphs not only occur in social networks. Many other areas also deal with large amounts of data that can be modeled as a graph. To support researchers and to make the information accessible for other people, it is often helpful to visualize the data. Since simply drawing each edge as a straight line would in most cases result in a very cluttered picture, a method to bundle the edges that connect neighboring vertices is often applied with good results, as can be seen in the example in Figure 1.3. Much research has been done in this area [99, 69, 68, 58, 45], since the size of the graphs that need to be visualized is steadily growing and in order to be able to work with them a meaningful drawing is essential.

A totally different field where graphs and also graph drawing play an important role is VLSI chip design. Here the task is, to design the layout for a chip on which a set of locations has to be connected by wires. Usually this is done in an orthogonal way, meaning the wires (or edges) that connect the locations (or vertices) are layouted using only horizontal and vertical segments. This was one of the original motivations for the orthogonal graph drawing model and primed a lot of research [83, 113, 114]. Many of the layout aesthetics applied to drawings of graphs stem from this area, since the chip design itself motivated the research into their optimization: having smaller area allows for smaller chips, making the production more cost-effective [98]; having less bends on edges makes the routing of the wires easier, which in turn makes the production of the chips easier and cheaper [19, 110]; minimizing the number of crossings is of course also very important since on the actual

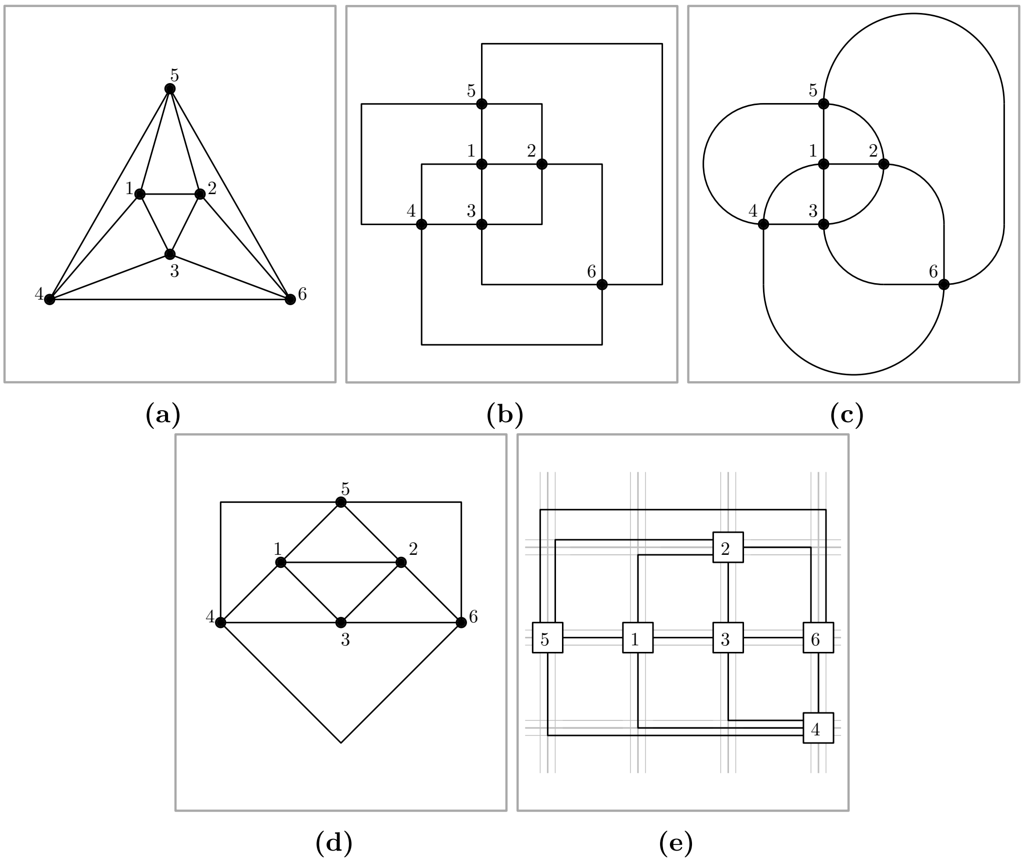


Figure 1.4: Drawings of the octahedron (a) in the straight-line model, (b) in the orthogonal model, (c) in the smooth orthogonal model, (d) in the octilinear model and (e) in the Kandinsky model.

chip wire crossings have to be avoided, but unfortunately this problem turned out to be NP-complete [60].

In the classical *orthogonal graph drawing* model vertices are drawn as points on an integer grid and edges as polygonal lines connecting those points using alternating horizontal and vertical line segments (see Figure 1.4b for an example). This naturally implies a bound of 4 for the maximal number of edges incident to a vertex. This model has been studied for a long time, initially motivated by VLSI layouts and floorplanning applications. Much research has been done to optimize different aspects of orthogonal drawings.

Papakostas and Tollis [97] and Tamassia and Tollis [113] both present linear time algorithms to compute planar orthogonal grid drawings on a grid quadratic in size of the number of vertices with a constant number of bends per edge. In a *rectilinear* drawing of a graph each edge is drawn as either a horizontal or a vertical line. In a *planar* rectilinear drawing no two lines intersect each other except at common endpoints. Garg and Tamassia [61] showed that it is NP-hard to test whether a given graph admits a planar rectilinear drawing and to approximate the minimum number of bends in a planar orthogonal drawing of a graph over all embeddings. Tamassia [110] on the other hand showed how to efficiently compute a bend-minimal orthogonal drawing for a graph with a given embedding, that is, the cyclical order of edges around each vertex is given. It is NP-hard, though, to find the embedding admitting the drawing with the minimal number of bends [61]. Biedl and Kant [19] and Liu et al. [86] both present algorithms that produce planar orthogonal drawings with at most two bends per edge, except for the octahedron (see Figure 1.4a), which requires three bends per edge. Both algorithms require area quadratic in the number of vertices.

A more detailed overview can, for example, be found in [42].

To overcome the restriction to graphs with maximal degree four of the classical orthogonal model, the *Kandinsky model* was developed [54]. It allows for arbitrary vertex degree by drawing vertices as boxes with non-zero side length. This way, multiple edges can be connected to the same side of a vertex. The method makes use of two different grids; a *coarse grid*, which is used to place the (center of) the vertex boxes, and a *fine grid* along which the edges are routed (see Figure 1.4e for an example). This model became very important in many practical applications, since it is able to draw any graph. Several different variants have been proposed and studied [30, 77, 111]. Initially, a min-cost flow formulation was presented [54], but Eiglsperger [41] showed that the formulation was not correct and gave an efficient 2-approximation. Recently, Bläsius et al. [20] showed that the bend-minimization problem in the Kandinsky model is NP-complete, which was an open problem for more than two decades.

It is common to both octilinear and orthogonal graph drawing that the number of different slopes of segments used to draw the edges is restricted (to 2 in orthogonal and to 4 in octilinear drawings). In a more general setting, researchers investigate the *planar slope number of planar graphs*, which are graphs that can be drawn crossing free. The planar slope number of a graph G is the minimum number of different slopes required for edge-segments in a crossing free drawing of G. Keszegh et al. [78] show that, if the maximal number of edges connected to a vertex in a planar graph is d, it can be drawn crossing free with $\left\lceil \frac{d}{2} \right\rceil$ slopes, if two bends per edge are allowed. This implies that, for graphs with maximal vertex degree up to 8, their method can be used to compute octilinear drawings with at most 2 bends per edge. For graphs with maximal vertex degree 3 Kant [76] and Di Giacomo et al. [32] proved that planar octilinear drawings with zero bends always exist if the graphs contains at least 5 vertices. For several restricted classes of graphs the planar slope number is known [73, 79, 84, 89].

Since not all graphs can be drawn in a way that no edges intersect each other, another restriction that came up in recent years aims at the angle formed between two edges that cross [33, 10]. Since the quality of a drawing is heavily influenced by crossings [100], much effort has been invested into improving the drawings of non-planar graphs. One way to do that is to demand that the minimum angle formed between two edges at a common crossing is $90°$, which is the rule in the so called *right-angle crossing* model (for short *RAC*-model). Argyriou et al. [12] show that it is NP-hard to decide whether a straight-line RAC drawing for a given graph exists. Other works characterize classes of graphs admitting RAC drawings [34, 39] or investigate the general crossing resolution in a drawing [36]. Van Kreveld [115] investigates how drawing a planar graph in the RAC model influences aesthetics such as area requirement, the ratio between the longest and the shortest edge and angular resolution. The drawing models for non-planar graphs presented in this thesis all have in common that crossings are drawn with right angles, making them RAC drawings.

The *straight-line* drawing model does neither restrict the number of slopes nor the angles at crossings, instead edges are drawn as a straight line, connecting the respective vertices. An example can be seen in Figure 1.4a. In this model the vertices can be drawn with zero or non-zero size and the edges can use arbitrary slopes. An important result discovered independently by Fáry et al. [49] and other groups [107, 118, 106] is that any planar graph admits a planar straight-line drawing. De Fraysseix et al. [29] showed how to compute a planar straight-line drawing of a planar graph on an integer grid that is linear in the number of vertices in width and height. The method constructs the drawing in an incremental way, based on an ordering of the vertices that we also use for some of our algorithms. We will give more details in Section 2.4.2.

In *polyline drawings* edges are drawn as a combination of different segments which are connected at common points, so called *bends* (see Figure 1.4b for an example). When using bends the number of available slopes for edge-segments can be restricted, for example to two (Figure 1.4b), resulting in orthogonal drawings, or four (Figure 1.4d), resulting in octilinear drawings.

Another method to draw edges is using curves. There are methods using Bezier curves [22, 51] or force-directed approaches [52]. In the so called *smooth orthogonal* drawing model [15, 14, 7], edges are drawn using horizontal, vertical and circular line-segments. Bends are drawn using the circular segments. An example for a graph drawn in the smooth orthogonal model is given in Figure 1.4c.

1.1 Thesis Contribution

After introducing necessary preliminaries in Chapter 2, we present our first results for octilinear drawings of planar graphs with few bends in Chapter 3.

From the research on the planar slope number of a graph it is known that for planar graphs with maximal vertex degree up to 8 (meaning that there are at most eight edges incident to any vertex in the graph) octilinear drawings with at most 2 bends per edge exist [78]. On the other hand, for graphs with maximal vertex degree 3 there is always a planar octilinear drawing without bends [76, 32].

We investigate the gap between these results: 3-planar graphs (i.e. planar graphs with maximal vertex degree 3) can be drawn with zero bends while for all other graphs up to a vertex degree of 8 only methods that produce drawings with 2 bends per edge are known.

To establish a lower bound we will first show that there exist infinitely many 4-planar graphs for which a planar octilinear drawing requires a linear number of bends, but one bend per edge is enough. Then we give algorithms to compute one-bend drawings for 4- and 5-planar graphs. Both algorithms follow the same principle: We first show how to compute planar octilinear drawings for triconnected graphs using the canonical ordering. Then we extend this approach to biconnected graphs using SPQR-trees to decompose the graphs intro triconnected components that are then drawn using the algorithm for triconnected graphs. To further extend the approach to simply connected graphs we decompose them into the biconnected components using the BC-tree and then apply the biconnected algorithm to the subgraphs.

In the case of 4-planar graphs our algorithms produce drawings using cubic area, while for 5-planar graphs there exist inputs which require exponential area. For 6-planar graphs we show that one bend per edge is not enough by giving an infinite family of graphs that require two bends on at least one edge. By this we close the gap between the aforementioned results.

We published our results in [1], a more detailed version is available in [2] and we also submitted our results to the Journal of Graph Algorithms and Applications.

After this we turn our attention to one of the most established graph drawing models, the orthogonal graph drawing, in Chapter 4 and introduce an enhanced version which we call *slanted orthogonal graph drawing* (for short *slog*). We improve orthogonal drawings by paying special attention to layout aesthetics that were identified as very important, namely crossings and the number of bends [100]. The new model makes it easier to identify crossings in a drawing by allowing them only between two diagonal segments and, at the same time, keeps the number of bends minimal. Additionally the appearance of bends is improved by requiring the minimal angle between two consecutive segments of an edge to be 135°, resulting in a shape we call *half-bends*.

A *representation* of a graph is a description of the shape of a drawing of the graph that contains for all vertices v the angles formed between incident edges at v and for all edges e the number and direction of bends along e. It does not contain any coordinates. In an orthogonal representation all angles are multiples of 90°, while a slog representation contains only angles that are multiples of 45°. We prove that the number of half-bends in a bend-optimal slog representation is at least twice the number of bends of the corresponding bend-optimal orthogonal representation and present an algorithm to compute the bend-optimal slog drawing for a graph with a given planar embedding. Our algorithm is based on the famous approach of Tamassia [110], who uses a network flow formulation to compute a bend-optimal orthogonal representation. We modify this flow network such that we can use it for the slog model and show the correctness of our modification.

The improvements of the crossings and the bends are bought with an increased area requirement. To counter this effect, which in the worst case can lead to exponential drawing area, we present a heuristic that is able to compute drawings in quadratic area by allowing more bends. The heuristic takes a bend-optimal orthogonal drawing and transforms it into a drawing in the slog model by stretching parts of the drawing and rotating vertices. Despite an initial scale-up by a constant factor and the subsequent stretching operations, the area requirement of drawings obtained by the heuristic is polynomial. In fact, due to the final step of the heuristic, which compacts the drawing as much as possible, the area requirements of slog drawings obtained by the heuristic and orthogonal drawings seems to be comparable. Drawings obtained by the heuristic contain at most twice the number of half-bends of a bend-optimal slog drawing. However, we found that in practice this number is significantly smaller.

Finally, we experimentally evaluate the slog model and compare bend-optimal and heuristically obtained drawings with classical orthogonal drawings.

We published our results for the slog model first in [5] and later extended them in [4].

In Chapter 5 we extend the slog model to a more flexible variant that we call *sloggy*. In this variant crossings are no longer allowed only on diagonal segments, but also between horizontal and vertical segments. Slog drawings may require significantly more half-bends than twice the number of bends of the corresponding orthogonal drawing, which is mainly due to the constraint that crossings have to be on diagonal segments. We prove that the number of half-bends in sloggy drawings is exactly twice the number of bends of the corresponding bend-optimal orthogonal drawing.

For sloggy drawings we require the number of crossings on diagonal segments to be as large as possible, because we want to improve the visibility of crossings as in the slog model. We develop a method to transform an orthogonal drawing into a representation in the sloggy model by finding certain cycles in the dual of the graph and rotating parts of the graph according to these cycles. We show that there always exists a system of such cycles that allows us to transform an orthogonal drawing into a sloggy representation that has the minimum number of bends and the maximum number of crossings on diagonals. However, we strongly suspect the problem of finding this cycle system to be NP-hard, so we give an ILP formulation for this purpose. Using this ILP formulation we can either optimize the number of crossings on diagonals or try to distribute the half-bends as evenly as possible over the edges, which is a second optimization criterion we identified in our experiments. Of course a weighted combination of both criteria is also possible.

To draw a sloggy representation we show how to modify the approach for the slog model such that we can use it also in the sloggy model. We prove that, in the worst-case sloggy drawings also require exponential area.

We published our results on the sloggy model in [3].

For Chapter 6 we focus on the well known and widely used Kandinsky model, that is able to draw graphs with arbitrary vertex degrees in an orthogonal fashion. We extend this model using elements from the slog model, namely the half-bends, to be able to draw shapes that were previously forbidden in the Kandinsky model. By this we obtain drawings with less bends and smaller area. As already stated, the bend-minimization problem in the Kandinsky model is NP-complete [20]. This motivated us to give an ILP formulation that is able to compute the bend-optimal representation for our extended model. By a simple transformation of the result of this ILP we obtain a representation that can be drawn using any known algorithm to draw Kandinsky representations.

Since we found that there exist graphs for which the linear program requires a very long time to solve, we also developed a heuristic that is able to efficiently modify a classical Kandinsky representation into a representation in the new model. The heuristic identifies faces in the given representation that can be transformed into the new shapes we allow by using half-bends and modifies the representation accordingly. Some of these transformations require twice as many half-bends as the Kandinsky representation required bends, but some also require the same number of half-bends. These are the cases in which we profit the most from the new model. Since they only occur when edges have a specific shape in the input Kandinsky representation, we show how to modify the computation of the bend-optimal Kandinsky representation we use as a starting point for the heuristic, such that it has as many edges with this shape as possible but is still bend-optimal. The drawing for the resulting representation is obtained by the same method we used for the bend-optimal approach.

We experimentally evaluate both algorithms to verify the improvement obtained with our extension.

Finally, in Chapter 7 we extend the slog model to graphs of arbitrary vertex degree, based on the approach of Chapter 6. We call the new model *sloginsky*. It transfers the advantages of the slog model (improved visibility of crossings and smoother shape of the edges) to the high-degree setting. In the sloginsky model we again require crossings to only occur between two diagonal segments and the minimal angle between two consecutive segments of an edge has to be 135°. We give an ILP formulation that can be used to compute a bend-optimal sloginsky representation of a given graph. To obtain a drawing we show how to modify this representation such that the algorithm used to draw a slog representation can be used to obtain a sloginsky drawing.

We also show that in the sloginsky model drawings may require exponential area and that there exist sloginsky representations that can not be drawn. If, however, a drawing exists, we found in our experimental evaluation that the computation can be done quite fast, allowing for practical use.

We present a conclusion in Chapter 8.

2 Preliminaries

In this chapter we introduce the theoretical backgrounds necessary throughout this thesis. We start with the basic definitions in Section 2.1, then we define the graph drawing aesthetics that are relevant for the vast majority of graph drawing algorithms in Section 2.2. In Section 2.3 we revisit basic algorithmic tools required throughout the thesis and in Section 2.4 we introduce standard methods for drawing graphs, that we also use in some of our approaches.

2.1 Terminology

A *graph* $G = (V, E)$ is a tuple consisting of a set of *vertices* V and a set of *edges* $E \subseteq V \times V$ with $|V| = n$ and $|E| = m$. In a *directed graph* the edges have a source and a target vertex, while in an *undirected graph* an edge just connects two vertices. Two vertices that are connected by an edge are said to be *neighbors* of each other. The set $N(v)$ of neighbors of a vertex v contains all vertices connected to v by an edge. If more than one edge between two vertices is allowed, then G is a *multigraph*, otherwise G is called *simple*. If the vertices of G can be separated into two sets U and V, such that all edges connect only vertices from U with vertices from V and vice versa, then G is called *bipartite*. A *path* $P =< v_1, v_2 \ldots, v_k >$ is an ordered set of vertices such that $\forall v_i, v_{i+1}$ with $1 \leq i < k : (v_i, v_{i+1}) \in E$, where k denotes the *length* of the path. A *cycle* is a path of length ≥ 1 starting and ending in the same vertex. A cycle of length 1 is a *self-loop*. In a *connected* graph there exists for each pair of vertices u, v a path connecting u and v. If a graph is k-connected, there exist k disjoint paths connecting each pair of vertices. The *degree* $d(v)$ of a vertex v is the number of edges connected to v. The *degree* $d(G)$ of a graph G is the maximum degree of all vertices in G, that is $d(G) = max_{v \in V} d(v)$.

A *planar drawing* $\Gamma(G)$ of a graph G is a drawing in which no two edges overlap, except at common endpoints. An *embedding* $\mathcal{E}(G)$ of a graph G is a cyclic order of the edges around each vertex, which describes the topology of the graph. A *planar embedding* is an embedding admitting a planar drawing. A *plane* graph is a planar graph with a planar embedding.

A cycle of edges that are all consecutive in the cyclic ordering given by a planar embedding at their common endpoints describes a *face*. The *dual graph* G^* of a plane graph G is a planar graph that contains a vertex for each face of G. For each edge e of G there is an edge in G^* connecting the two vertices that correspond to the two faces adjacent to e.

Given a drawing $\Gamma(G)$ of a graph G, we denote by $p_u = (x_u, y_u)$ the position of vertex $u \in V$ on the plane. Given a pair of points $q, q' \in \mathbb{R}^2$, we denote by $|qq'|$ the Euclidean distance between q and q'. We refer to the line segment defined by q and q' as $\overline{qq'}$.

A *planar graph* is a graph admitting a planar drawing. A *maximal planar graph* is a graph where no edge can be added without violating the planarity. Eulers formula for convex polyhedra [93] states that $n + f - e = 2$, where n is the number of vertices, f is the number of faces and e is the number of edges. It follows from this formula that any planar graph can have at most $3n - 6$ edges. A *k-planar* graph is a planar graph with maximum degree k.

We also use the notion of a *left* or *right* turn, which we formally define in the following.

Definition 1. *Let $e = (u, v)$ be an edge with at least one bend, say b, and let s and s' be two consecutive segments of e with b being the common bend of s and s'. Furthermore, let ϕ be the angle formed by s and s' on their left side when moving along e from u to v. Edge e has a* left turn *on b if $\phi \leq 180°$, otherwise there is a* right turn *on b.*

2.2 Graph Drawing Aesthetics

In Chapter 1 we introduced several different graph drawing models. There exist various different aesthetic criteria that can be used to judge the quality of a drawing of a graph. Purchase et al. [103] described how to validate aesthetics important to graph drawing. In a subsequent study [100] she found that crossings are the most important layout aesthetic, followed by bends and symmetry. Further studies on different algorithms [101] and on metrics able to measure the quality of a drawing of a graph [102] demonstrate the importance of aesthetic aspects to graph drawing. We describe here the most important aspects that any drawing algorithm should take into consideration.

Overlaps of elements of a drawing are considered as very bad drawing style, since they dramatically decrease the readability of a drawing. Most algorithms aim to avoid overlaps, especially if vertices in the model have non-zero size. But since it can not always be guaranteed that there are no overlaps, there is also research on how to post-process a drawing obtained by any algorithm to remove them, for example the work of Dwyer et al. [37].

Crossings of edges are, after the overlaps, the most important layout aesthetic, according to the study of Purchase [100]. While the general problem of minimizing the number of crossings in a drawing for a given graph is NP-complete [60], much research has been devoted to the general problem and restricted versions of it [92, 74, 25], and heuristics have been developed [40, 46, 13]. An experimental evaluation can, for example, be found in [66]

Bends appear only in polyline drawings of graphs. The number of bends of a drawing is very important for the perceived quality of a drawing [100]. This is because the larger the number of bends of a drawing gets, the harder it becomes to visually follow the edges. Minimizing the number of bends of a drawing has been mostly studied in the context of orthogonal [110, 113, 19, 97, 86, 61] or octilinear [76, 78, 32] drawings (refer also to the introduction of the orthogonal and the octilinear graph drawing model in Chapter 1).

Symmetry in graph drawing is, according to the study of Purchase [100], also very important for the perceived quality of a drawing. If parts of a graph are very symmetric, this can identify an important information in the respective context. There are several results for this problem [38, 70, 71], but we will not go into further detail here.

2.3 Algorithmic Tools

We now quickly recall some basic tools used at different points throughout this thesis.

2.3.1 Linear Programming

Linear programming (for short LP) is a method to model linear optimization problems. The core of the linear program is a set of *variables* $x_1, \ldots x_n$ that are allowed to have any real number as value. It is possible to restrict the values of the variables to be between a lower and an upper bound.

Additionally there is a set of m *constraints* of the form $a_{1j}x_1 + a_{2j}x_2 + \ldots + a_{nj}x_n \leq c_j$ for $j = 1, \ldots, m$ with $a \in \mathbb{R}^{m \times n}$ and $c \in \mathbb{R}^m$ that have to be satisfied by a valid assignment to the x_i.

Finally there is an *objective function*, which is a linear combination of a subset of the variables. The goal of the linear program is to find an assignment of values to the variables, such that all values are in the allowed range, all constraints are satisfied and the value of the objective function is maximized (or minimized).

Linear programming is a very powerful tool that can be used to obtain optimal solutions for many problems as a plethora of research from many different fields, not only restricted to computer science, indicates [62, 8, 116, 121, 48, 28, 50].

If the values of the variables have to be integer, the problem becomes NP-complete [59] and is called *integer linear programming* (for short *ILP*). If only some of the variables have to be integer, the problem is called *mixed integer programming.* Despite its NP-hardness, todays solvers are often capable of computing solutions even for large ILPs, which is one of the reasons why they also received a lot of attention [117, 85, 9, 63, 104, 94].

An overview of linear programming techniques can, for example, be found in [105].

2.3.2 Network Flow

A *flow network* consists of a graph $G = (V, E)$, a set of sources $S \subset V$, a set of sinks $T \subset V$ and a *capacity* function $c : (u, v) \in E \to \mathbb{N}$. A *valid flow* f assigns to each edge $e = (u, v)$ a value respecting the following conditions:

- $f(u, v) \leq c(u, v)$; the flow on an edge is not allowed to exceed its capacity
- $f(u, v) = -f(v, u)$; the flow is symmetric
- $\forall v \in V \setminus \{S \cup T\} : \sum_{(u,v) \in E} f(u, v) = \sum_{(v,w) \in E} f(v, w)$; the total amount of flow entering each vertex has to be equal to the total amount of flow leaving it, except for source and sink vertices

The *value* of a flow $val(f)$ is the amount of flow leaving all source vertices: $val(f) = \sum_{s \in S, v \in V, (s,v) \in E} f(s, v)$.

The *maximum flow* (for short *max flow*) for a given flow network is a valid flow with maximum value. It can be computed in time $O(n^2\sqrt{m})$ [6], where n is the number of nodes and m the number of edges of the flow network.

It is possible to extend this model by introducing a second function assigning each edge a *cost* : $(u, v) \in E \to \mathbb{R}$, imposing a cost on each unit of flow traversing an edge. The *min-cost flow* then asks for the maximum flow with minimum cost. There exist several algorithms to solve this problem, where a typical runtime is $O((m \log n)(m + n \log n))$ [6].

Both, the maximum flow and the min-cost flow problem can be extended with additional constraints requiring a minimum flow on certain edges. Finding the optimal flow for such a network can be done without increasing the runtime of the original algorithms [6].

Network flow is an important tool in many areas of computer science and has been used to solve a variety of problems [57, 47, 108, 120, 27, 119].

An overview over algorithms and techniques for network flow problems can, for example, be found in [6].

2.4 Common Methods In Graph Drawing

We will now describe some standard practices employed by many graph drawing algorithms, that will also be important in the methods developed in this thesis.

A *representation* $R(G)$ of a plane graph G is a description of the shape of a drawing of G. It prescribes the angle between each pair of edges that are consecutive in the circular ordering of the edges on a common vertex. Also, the number of bends and their direction along each edge are prescribed in the representation. With this information the shape of a drawing of G is specified; but the actual geometric information is not contained.

An *orthogonal representation* uses only angles which are a multiple of 90°. It consists basically of four values assigned to each edge $e = (u, v)$, two for each direction. The first is $\alpha(u, v) \cdot 90°$ which corresponds to the angle at vertex u between e and its cyclic predecessor on u. The second is $\beta(u, v)$, the number of left turns when traversing e from u to v, each one being a 90° turn. In the classical orthogonal model $1 \leq \alpha(u, v) \leq 4$ and $\beta(u, v) \geq 0$, which means that the maximal degree of any vertex is restricted to four. Also, the sum of all angles around a vertex has to be 360°, so $\sum_{v \in N(u)} \alpha(u, v) = 4$. Also, from basic geometry it has to hold that the sum of angles formed at vertices and bends of a bounded face f equals $180° \cdot (p(f) - 2)$, where $p(f)$ denotes the total number of such angles. This implies that $\sum_{(u,v) \in E(f)} \alpha(u, v) + \beta(u, v) - \beta(v, u) = 2a(f) - 4$, where $a(f)$

denotes the total number of vertex angles in f, and $E(f)$ the directed arcs of f in its counterclockwise traversal. If f is the outer face, the sum is increased by eight.

A drawing of G that has the properties of a given representation is *realizing* this representation. Tamassia [110] shows how to efficiently compute a drawing realizing a given orthogonal representation. However, it is not always possible to compute a drawing for each representation. There exist examples of representations, for example in the octilinear graph drawing model [94], that are not realizable.

2.4.1 TSM Approach

The *topology-shape-metrics* (for short *TSM*) approach was introduced by Tamassia [110, 112] to compute orthogonal drawings for graphs with maximal vertex degree four. Originally it was known as the GIOTTO approach, but later it became the TSM approach [30]. It was used and extended in many other works [18, 35, 54, 80]. Originally the TSM approach consists of three phases:

Planarization: In the planarization phase a planar embedding of the given non-planar graph is computed. If there is a crossing between two edges $e_1 = (u, v)$ and $e_2 = (w, x)$, a dummy vertex c is introduced, connected with u, v, w and x with new edges and e_1 and e_2 are removed.

We refer to the dummy vertices introduced to replace crossings as *crossing-vertices* or *c-vertices*. The vertices of the original graph we call *real-vertices* or *r-vertices*. By *cc-edges* we denote edges connecting two crossing vertices, *rr-edges* connect two real vertices and *rc-edges* connect an r- and a c-vertex.

Orthogonalization: In the orthogonalization phase a representation is computed. Tamassia [110] shows how to compute a bend-optimal orthogonal representation of a given embedding using a flow network. The properties of an orthogonal representation can be directly transferred to a flow network as follows: In general, one unit of flow in the network corresponds to a 90° angle. For each vertex v of the original

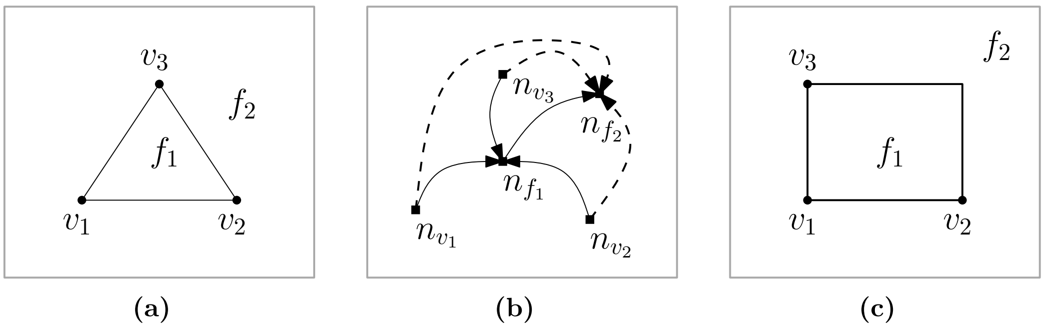


Figure 2.1: (a) Exemplary input to the orthogonalization phase of the TSM approach. (b) Corresponding flow network (only edges with flow in the solution are shown). (c) The drawing corresponding to the min-cost flow.

graph, there is a node n_v in the flow network with a supply of 4 units; these nodes are called *vertex-nodes*. Also, for each face f in the original graph, the network contains a node n_f, with a demand of $2a(f) - 4$ if f is bounded, or $2a(f) + 4$ if f is the outer face; these nodes are called *face-nodes*. Again $a(f)$ denotes the number of vertex angles of f. For each angle α between two edges $e = (u, v)$ and $e' = (u, w)$ at a vertex u, n_u is connected with the face-node representing the face incident to α. The connection has capacity 4, cost zero and minimum flow 1 to make sure of the angular restrictions. If in the solution of the network flow there are i units of flow on such an edge, it means that $\alpha = i \cdot 90°$. Also, for each edge of the input graph, the nodes representing the two adjacent faces are connected with arcs with infinite capacity and cost 1. Each unit of flow over one of those edges symbolizes a bend in the representation. Since with this model each bend is penalized with one unit of cost, a minimum-cost flow solution to the network gives a bend-minimal representation.

We will demonstrate this with an example, refer to Figure 2.1. Figure 2.1a shows the graph for which we want to compute a bend-optimal orthogonal representation, a simple triangle consisting of three vertices and two faces. In Figure 2.1b the corresponding flow network is shown. We only included edges with non-zero flow in a min-cost flow solution.

Solid edges means one unit of flow while dashed edges means three units. Notice that from each vertex-node one unit of flow is passed to face-node n_{f_1}, which indicates a $90°$ angle at each vertex in face f_1, while the remaining three units of supply that each vertex has are sent to n_{f_2}, which is the outer face. Since the demand of n_{f_1} is $2a(f_1)-4=2$ and the demand of n_{f_2} is 10, one unit of flow has to be sent from n_{f_1} to n_{f_2} for a feasible solution, in the example through the edge corresponding to the connection between v_2 and v_3. Figure 2.1c then shows a drawing realizing this solution: The angles on all vertices adjacent to f_1 are $1 \cdot 90°$, the angles adjacent to f_2 are $3 \cdot 90°$ and there is one bend on edge (v_2, v_3).

Compaction: In the compaction phase a drawing realizing the representation from the previous step is computed. Tamassia [110] shows how to compute a drawing realizing the orthogonal representation computed in the previous step by again using network flow. For this it is necessary that the faces of the representation have convex shape. Tamassia achieves this by splitting all faces into rectangles by introducing appropriate vertices and edges. Then he uses two flow networks, one to compute the x- and one to compute the y-coordinates.

2.4.2 Drawing Planar Graphs Step by Step

The canonical ordering is an ordering of the vertices of a triconnected graph. The ordering consists of an ordered set of layers, that contain the vertices of the graph. Many graph drawing algorithms use this order to construct a drawing in an incremental way by adding one layer of the ordering after the other. One example is the well-known algorithm by De Fraysseix et al. [29] that can be used to construct a planar straight line drawing on a grid of quadratic size for a planar graph. They show how to do this by first enriching the graph with additional edges until it becomes maximal planar, then a canonical ordering is computed and one vertex after the other is added to the drawing, according to the canonical order, while maintaining planarity. The original definition of the canonical order of De Fraysseix et al. [29] requires the graph to be triangulated. It was later improved by

Kant [75] such that it only requires the graph to be triconnected. The formal definition of the canonical order (based on Kant [75]) is given in Definition 2.

Definition 2 (Canonical order [75]). *For a given triconnected plane graph $G = (V, E)$ let $\Pi = (P_0, \dots, P_m)$ be a partition of V into paths such that $P_0 = \{v_1, v_2\}$, $P_m = \{v_n\}$ and $v_2 \to v_1 \to v_n$ is a path on the outer face of G. For $k = 0, \dots, m$ let G_k be the subgraph induced by $\cup_{i=0}^{k} P_i$ and assume it inherits its embedding from G. Partition Π is a canonical order of G if for each $k = 1, \dots, m-1$ the following hold: (i) G_k is biconnected, (ii) all neighbors of P_k in G_{k-1} are on the outer face, of G_{k-1} (iii) all vertices of P_k with $k < m$ have at least one neighbor in P_j for some $j > k$. P_k is called a singleton if $|P_k| = 1$ and a chain otherwise.*

When the graph for which a drawing should be computed is biconnected, but the approach to enrich it with additional edges to make it triconnected can not be applied (because, for example, there are restrictions on the maximal degree of a vertex), the SPQR-tree can be used. The standard-practice is to decompose a biconnected graph into its triconnected components using the SPQR-tree, then use an algorithm for triconnected graphs to draw those components, and finally show how to combine the drawings of the subgraphs in a way respecting the constraints of the used drawing model.

An SPQR-tree [31] provides information about the decomposition of a biconnected graph into its triconnected components. It can be computed in linear time and space [65]. Every triconnected component is associated with a node μ in the SPQR-tree $\mathcal{T}$. The triconnected component itself is referred to as the *skeleton* of μ, denoted by $G_\mu^{skel} = (V_\mu^{skel}, E_\mu^{skel})$. We refer to the degree of a vertex $v \in V_\mu^{skel}$ in G_μ^{skel} as $deg_\mu^{skel}(v)$. We say that μ is an *R-node*, if G_μ^{skel} is a simple triconnected graph. A bundle of at least three parallel edges classifies μ as a *P-node*, while a simple cycle of length at least three classifies μ as an *S-node*. By construction R-nodes are the only nodes of the same type that are allowed to be adjacent in $\mathcal{T}$. The leaves of $\mathcal{T}$ are formed by the *Q-nodes*. Their skeleton consists of two parallel edges; one of them corresponds to an edge of G and is referred to as *real edge*. The skeleton edges that are not real are referred to as *virtual edges*. A virtual edge e in G_μ^{skel} corresponds to a tree node μ' that is adjacent to μ in $\mathcal{T}$, more exactly, to another virtual edge e' in $G_{\mu'}^{skel}$. We assume that $\mathcal{T}$ is rooted at a Q-node. Hence, every skeleton (except the one of the root) contains exactly one virtual edge $e = (s, t)$ that has a counterpart in the skeleton of the parent node. We call this edge the *reference edge* of μ denoted by *ref*(μ). Its endpoints, s and t, are named the *poles* of μ denoted by $\mathcal{P}_\mu = \{s, t\}$. Every subtree rooted at a node μ of $\mathcal{T}$ induces a subgraph of G called the *pertinent graph* of μ that we denote by $G_\mu^{pert} = (V_\mu^{pert}, E_\mu^{pert})$. We abbreviate the degree of a node v in G_μ^{pert} with $deg_\mu^{pert}(v)$. The pertinent graph is the subgraph of G for which the subtree describes the decomposition.

Now, assume that G is a simple, biconnected k-planar graph, whose SPQR-tree $\mathcal{T}$ is given. Additionally, we may assume that $\mathcal{T}$ is rooted at a Q-node that is adjacent to an S- or R-node. Notice that at least one such node exists since the graph does not contain any multi-edges, which would be the case if only a P-node existed. Biconnectivity and maximum degree of k yield basic bounds for the graph degree of a node $v \in V$, i.e., $2 \leq deg(v) \leq k$. By construction the pertinent graph of a tree node μ is a (connected) subgraph of G; thus $1 \leq deg_\mu^{pert}(v) \leq deg(v)$. For the degrees in a skeleton graph G_μ^{skel}, we obtain bounds based on the type of the corresponding node. Skeletons of Q-nodes are cycles of length two, whereas S-nodes are by definition simple cycles of length at least three; hence, $deg_\mu^{skel}(v) = 2$. For P- and R-nodes the degree can be bounded by $3 \leq deg_\mu^{skel}(v) \leq k$, since the skeleton of the former is at least a bundle of three parallel virtual edges and the latter's skeleton is triconnected by definition. The upper bound is derived from the relation between skeleton and graph degrees: A virtual edge $e = (s, t)$ hides at least one incident edge of each node (not necessarily an (s, t)-edge). This observation can be proven by induction on the tree. Hence, $2 \leq deg_\mu^{skel}(v) \leq deg(v)$.

Next, we use this observation to derive bounds for the pertinent degree by distinguishing two cases depending on whether v is a pole or not. Recall that G_μ^{pert} is a subgraph of G that is obtained by recursively replacing virtual edges by the skeletons of the corresponding children. In the first case when v is an internal node in G_μ^{pert}, i.e., $v \notin \mathcal{P}_\mu$, v is not incident to the reference edge in G_μ^{skel}. Thus, every edge of G hidden by a virtual edge in G_μ^{skel} is in G_μ^{pert}. Hence, $deg_\mu^{skel}(v) \leq deg_\mu^{pert}(v) \leq k$. In the other case, i.e., $v \in \mathcal{P}_\mu$, at least one edge that is hidden by the reference edge, is not part of G_μ^{pert}, thus, $deg_\mu^{skel}(v) - 1 \leq deg_\mu^{pert}(v) \leq k - 1$. Notice that the lower bounds depend on the skeleton degree which in turn depends on the type of node, unlike the upper bounds that hold for all tree nodes. The next lemma tightens these bounds based on the type of the parent node.

Lemma 2.4.1. *Let μ be a tree node that is not the root in the SPQR-tree $\mathcal{T}$ of a simple, biconnected, k-planar graph G and μ' its parent in $\mathcal{T}$. For $v \in \mathcal{P}_\mu$, it holds that $deg_\mu^{pert}(v) \leq k - 2$, if μ' is a P- or an R-node and $deg_\mu^{pert}(v) \leq k - 1$ otherwise, i.e. μ' is an S- or a Q-node.*

Proof. Since the case where μ' is an S- or a Q-node follows from the fact that G is k-planar and the reference edge hides at least one edge that is not in G_μ^{pert}, we restrict ourselves to the more interesting cases where μ' is either a P- or an R-node. From our previous observations we know that $3 \leq deg_{\mu'}^{skel}(v) \leq k$. Each of the at least three edges in $G_{\mu'}^{skel}$ hides at least one edge of G that is incident to v. However, the total number of edges is at most k due to the degree restriction. Hence, we are left with the problem of k edges of G being hidden by at least three virtual edges, each hiding at least one. As a result the virtual edge that corresponds to μ cannot contribute more than $k - 2$ edges to its pertinent graph G_μ^{pert}. $\square$

Lemma 2.4.2. *In the SPQR-tree $\mathcal{T}$ of a planar biconnected graph $G = (V, E)$ with $\deg(v) \geq 3$ for every $v \in V$, there exists at least one Q-node that is adjacent to a P- or an R-node.*

Proof. Assume to the contrary that all Q-nodes are adjacent to S-nodes only. We pick such a Q-node and root $\mathcal{T}$ at it. Let μ be an S-node (possibly the root itself) with poles $\mathcal{P}_\mu = \{s, t\}$ such that there is no other S-node in the subtree of μ. By definition of an S-node, μ has at least two children. If all of them were Q-nodes then there exists a $v \in V_\mu^{skel}$ with $s \neq v \neq t$ and $\deg(v) = 2$; a contradiction. Hence, there is at least one child μ' that is a P- or an R-node. However, since the leaves of $\mathcal{T}$ are Q-nodes and those are not allowed to be children of P- and R-nodes by our assumption, there exists at least one other S-node in the subtree of μ' and therefore in the subtree of μ which contradicts our choice of μ. $\square$

Should the given graph be simply connected, the SPQR-tree can not be used. Instead the so-called BC-tree (see Definition 3) has to be used to decompose the graph into its biconnected components. The resulting structure will be a tree, that then can be used in a similar procedure as the SPQR-tree for biconnected graphs: the biconnected components are drawn using the SPQR-tree decomposition and the drawings are combined according to the BC-tree and the rules of the applied model.

Definition 3 (BC-tree)**.** *The* BC-tree $\mathcal{B}$ *of a connected graph* G *has a B-node for each biconnected component of* G *and a C-node for each cutvertex of* G*. Each B-node is connected with the C-nodes that are part of its biconnected component.*

3 Planar Octilinear Drawings

3.1 Introduction

Recall from Chapter 1, that in the octilinear graph drawing model edges are drawn as a combination of horizontal, vertical and diagonal (at 45°) segments.

In this chapter we investigate the gap between known results: 3-planar graphs can be drawn with zero bends [76, 32] while for all other planar graphs up to a maximal vertex degree of 8 only methods that produce drawings with 2 bends per edge are known [78].

We will first show that there exist infinitely many 4-planar graphs for which a planar octilinear drawing requires a linear number of bends, but one bend per edge is enough. Then we give algorithms to compute one-bend drawings for 4- and 5-planar graphs. For 6-planar graphs we show that one bend per edge is not enough; by this we close the gap between the aforementioned results.

Central to our algorithms is the principle of constructing a drawing for a given graph in an incremental way, which was introduced in Section 2.4.2.

3.2 Drawing 4-Planar Graphs with One Bend Per Edge

We will first show a family of 4-planar graphs that require a linear number of bends when drawn with at most one bend per edge in Section 3.2.1. Then we will describe how to compute planar octilinear drawings of 4-planar graphs that require at most one bend per edge. To do so we start by giving an algorithm to compute such a drawing for a triconnected 4-planar graph in Section 3.2.2. We then extend to biconnected graphs using SPQR trees in Section 3.2.3. Finally we show how to use BC-trees to construct drawings for simply-connected graphs in Section 3.2.4.

3.2.1 4-Planar Is Not Possible With Zero Bends

Theorem 3.2.1. *There exists an infinite class of 4-planar graphs which do not admit bendless octilinear drawings and if they are drawn with at most one bend per edge, then a linear number of bends is required.*

Proof. In the orthogonal drawing model the drawing of a triangle requires at least one bend, while a triangle can be drawn without bends in an octilinear way, as long as its interior is empty. With a construction like the one depicted in Figure 3.1 we make sure that for each triangular face there is always something in the interior, thus forcing us to use at least one bend per face. Since the example is triconnected, its embedding is fixed up to the choice of the outer face. Our construction is heavily based on the so-called *separating triangle*, i.e., a three-cycle whose removal disconnects the graph. Each vertex of such a triangle has degree four. Any triangle which is drawn bendless has a 45° angle inside. But since the triangles are nested and have incident edges going inside of the triangles, this is impossible. □

3.2.2 Triconnected Graphs

Let $G = (V, E)$ be a triconnected 4-planar graph and $\Pi = \{P_0, \dots, P_m\}$ be a canonical order of G. We momentarily neglect the edge (v_1, v_2) of the

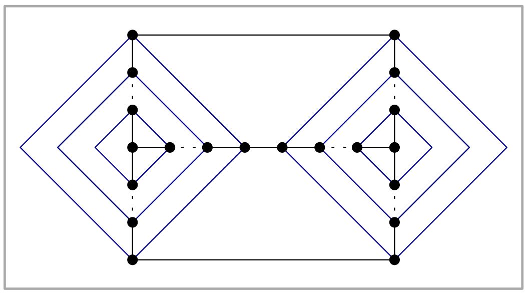


Figure 3.1: Nested separating triangles each requiring one bend.

first partition P_0 of Π and we start by placing the second partition, say a chain $P_1 = \{v_3, \dots, v_{|P_1|+2}\}$, on a horizontal line from left to right. Since by definition of Π, v_3 and $v_{|P_1|+2}$ are adjacent to the two vertices, v_1 and v_2, of the first partition P_0, we place v_1 to the left of v_3 and v_2 to the right of $v_{|P_1|+2}$. So, they form a single chain where all edges are drawn using horizontal line-segments that are attached to the east and west port at their endpoints. The case where P_1 is a singleton is analogous. Having laid out the base of our drawing, we now place in an incremental manner the remaining partitions. Assume that we have already constructed a drawing for G_{k-1} and we now have to place P_k, for some $k = 2, \dots, m-1$.

In case where $P_k = \{v_i, \dots, v_j\}$ is a chain of $j - i + 1$ vertices, we draw them from left to right along a horizontal line one unit above G_{k-1}. Since v_i and v_j are the only vertices that are adjacent to vertices in G_{k-1}, both only to one, we place the chain between those two as in Figure 3.2a. The port used at the endpoints of P_k in G_{k-1} depends on the following rule: Let v'_i (v'_j, resp.) be the neighbor of v_i (v_j, resp.) in G_{k-1}. If the edge (v_i, v'_i) ((v_j, v'_j), resp.) is the last to be attached to vertex v'_i (v'_j, resp.), i.e., there is no vertex v in $P_l \in \Pi$, $l > k$ such that $(v'_i, v) \in E$ ($(v'_j, v) \in E$, resp.), then we use the northern port of v'_i (v'_j, resp.). Otherwise, we choose the north-east port for (v_i, v'_i) or the north-west port for (v_j, v'_j).

In case of a singleton $P_k = \{v_i\}$, we can apply the previous rule if the singleton is of degree three, as the third neighbor of v_i should belong to a partition Π_j for some $j > k$. However, in case where v_i is of degree four we may have to deal with an additional third edge (v_i, v) that connects v_i with G_{k-1}. By the placement so far, we may assume that v lies between the other

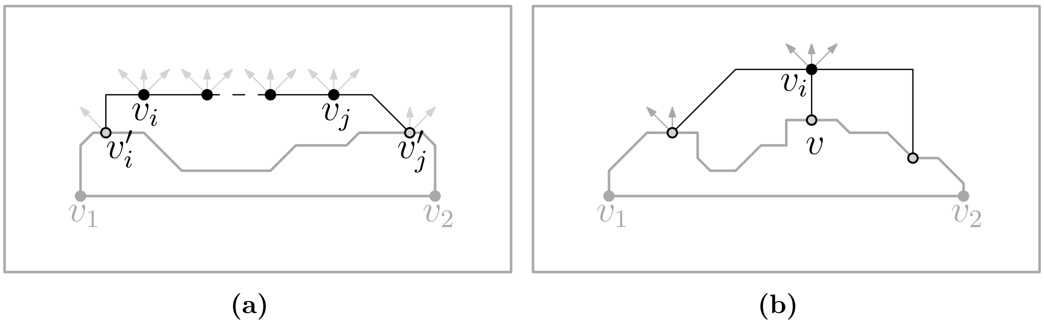


Figure 3.2: (a) Horizontal placement of a chain $P_k = \{v_i, \ldots, v_j\}$. (b) Placement of a singleton $P_k = \{v_i\}$ with degree four.

two endpoints, thus, we place v_i such that $x(v_i) = x(v)$. This enables us to draw (v_i, v) as a vertical line-segment; see Figure 3.2b.

The above procedure is able to handle all chains and singletons except the last partition P_m, because v_n may have 4 edges pointing downwards. One of these edges is (v_n, v_1), by definition of Π. We exclude (v_n, v_1) and draw v_n as an ordinary singleton. Then, we shift v_1 to the left and up as in Figure 3.3. This enables us to draw (v_1, v_n) as a horizontal-vertical segment combination. For (v_1, v_2), we move v_2 one unit to the right and down. We free the west port of v_2 by redrawing its incident edges as in Figure 3.3 and attach (v_1, v_2) to it. Edge (v_1, v_2) will be drawn as a diagonal segment with positive slope connected to v_1 and a horizontal segment connected to v_2, which requires one bend. Let (v_2, v_i) be the other incomplete edge according to Figure 3.3. It will be drawn using a diagonal segment with positive slope connected to v_2 and a horizontal segment connected to v_i, again requiring one bend.

So far, we have specified a valid port assignment and the y-coordinates

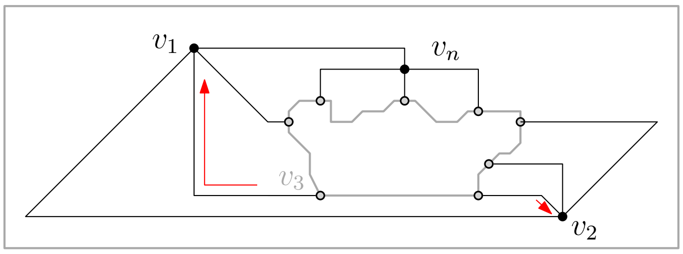


Figure 3.3: Final layout after repositioning v_1 and v_2.

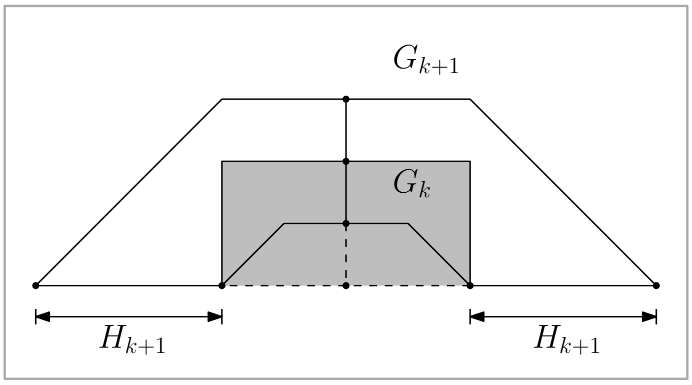
of the vertices. However, we have not fully specified their x-coordinates. Notice that by construction every edge, except the ones drawn as vertical line-segments, contains exactly one horizontal segment. This enables us to stretch the drawing horizontally by employing appropriate cuts. A *cut*, for us, is a y-monotone continuous curve that crosses only horizontal segments and divides the current drawing into a left and a right part. It is not difficult to see that we can shift the right part of the drawing that is defined by the cut further to the right while keeping the left part of the drawing on place and the result remains a valid octilinear drawing.

To compute the x-coordinates, we proceed as follows. We first assign consecutive x-coordinates to the first two partitions and from there on we may have to stretch the drawing in two cases. The first one appears when we introduce a chain, say P_k, as it may not fit into the gap defined by its two adjacent vertices in G_{k-1}. In this case, we horizontally stretch the drawing between its two adjacent vertices in G_{k-1} to ensure that their horizontal distance is at least $|P_k| + 1$. The other case appears when an edge that contains a diagonal segment is to be drawn. Such an edge requires a horizontal distance between its endpoints that is at least the height it bridges. We also have to prevent it from intersecting any horizontal-vertical combinations in the face below it. We can cope with both cases by horizontally stretching the drawing by a factor that is bounded by the current height of the drawing. Since the height of the resulting drawing is bounded by $|\Pi| = O(n)$, it follows that in the worst case its width is $O(n^2)$.

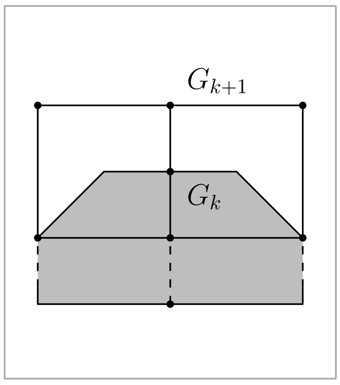
We are now ready to state the main theorem of this subsection.

Theorem 3.2.2. *Given a triconnected 4-planar graph G, we can compute in $O(n)$ time an octilinear drawing of G with at most one bend per edge on an $O(n^2) \times O(n)$ integer grid.*

Proof. In order to keep the time complexity of our algorithm linear, we employ a simple trick. We assume that any two adjacent points of the underlying integer grid are by n units apart in the horizontal direction and by one unit in the vertical direction. This a priori ensures that all edges that contain a diagonal segment will not be involved in crossings and simultaneously does not affect the total area of the drawing, which asymptotically remains cubic.



(a)



(b)

Figure 3.4: (a) An example for which the triconnected algorithm requires quadratic width. (b) The same example drawn with width four.

On the other hand, the advantage of this approach is that we can use the shifting method of Kant [75] to cope with the introduction of chains in the drawing, that needs $O(n)$ time in total by keeping relative coordinates that can be efficiently updated and computing the absolute values only at the last step. □

Note that our algorithm produces drawings that have a linear number of bends in total (in particular, exactly $2|\Pi| = O(n)$ bends). From the construction in Section 3.2.1 it follows that this bound is asymptotically tight.

An example for which our algorithm actually results in quadratic width is depicted in Figure 3.4a. The construction is as follows: the first layer of the canonical ordering is a chain of length $\frac{n}{2}$ and then there are another $\frac{n}{2}$ layers consisting of singletons of degree four. Every second singleton adds twice the height of the current drawing $H_{current}$ to the width, which results in a total width of $O(n^2)$. For comparison a drawing of the same graph requiring a constant width of four units is depicted in Figure 3.4b.

3.2.3 Biconnected Graphs

Following standard practice, we employ a rooted SPQR-tree and assume for a tree node that the pertinent graphs of its children are drawn in a

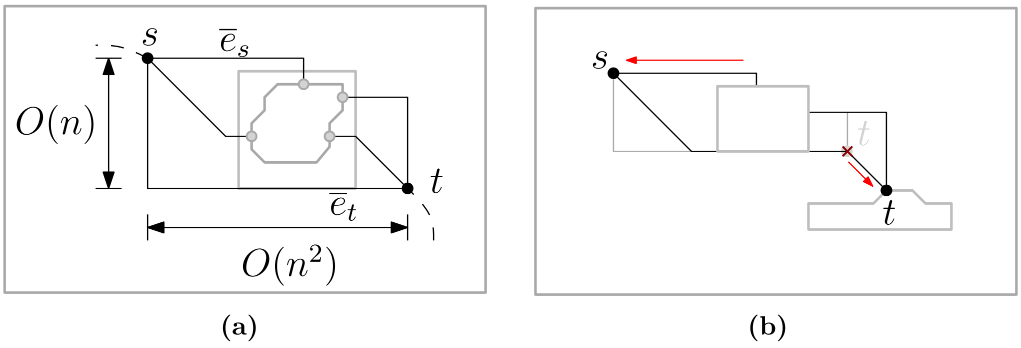


Figure 3.5: (a) Schematic view of the layout requirements. (b) Creating a nose at t.

pre-specified way. Consider a node μ in $\mathcal{T}$ with poles $\mathcal{P}_\mu = \{s,t\}$. In the drawing of G_μ^{pert}, s should be located at the upper-left and t at the lower-right corner of the drawing's bounding box with a port assignment as in Figure 3.5a. In general, we assume that the edges incident to s (t, resp.) use the western (eastern, resp.) port at their other endpoint, except of the northern (southern, resp.) most edge which may use the north (south, resp.) port instead. In that case we refer to s and t as *fixed*; see $\overline{e}_s, \overline{e}_t$ in Figure 3.5a. More specifically, we maintain the following invariants:

IP-1: The width (height) of the drawing of μ is quadratic (linear) in the size of G_μ^{pert}. s is located at the upper-left; t at the lower-right corner of the drawing's bounding box.

IP-2: If $deg_\mu^{pert}(s) \geq 2$, s is fixed; t is fixed if $deg_\mu^{pert}(t) = 3$ and μ's parent is not the root.

IP-3: The edges that are incident at s and t in G_μ^{pert} use the south, south-east and east ports at s and the north, north-west and west port at t, respectively. If s or t is not fixed, incident edges are attached at their other endpoints via the west port (east port, resp.). If s or t is fixed, the northern-most edge at s and the southern-most edge at t may use the north (south, resp.) port at its other endpoint.

Notice that the port assignment, i.e. IP-3, guarantees the ability to stretch the drawing horizontally even in the case where both poles are fixed.

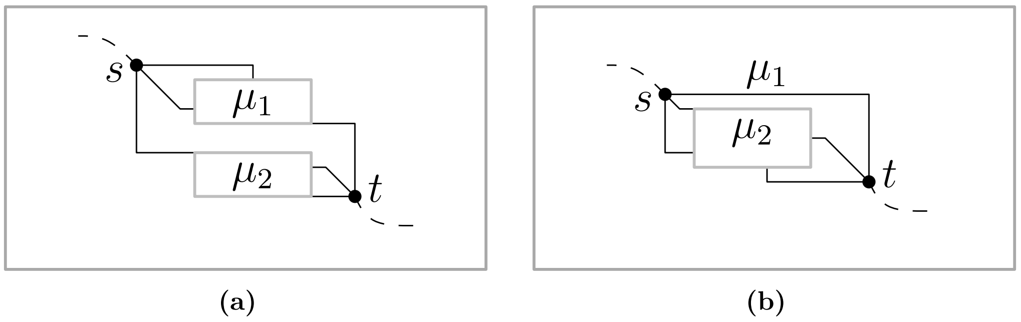


Figure 3.6: (a) First P-node subcase without an (s,t)-edge but s might be fixed in a child μ_1. (b) Second P-node subcase with an (s,t)-edge where t might get fixed in a child μ_2.

Furthermore, IP-2 is *interchangeable* in the following sense: If $deg_{\mu}^{pert}(s) = 2$ and $deg_{\mu}^{pert}(t) = 1$, then s is fixed but t is not. But, if we relabel s and t such that $t' = s$ and $s' = t$, then $deg_{\mu}^{pert}(s') = 1$ and $deg_{\mu}^{pert}(t') = 2$. By IP-2, we can create a drawing where both s' and t' are not fixed and located in the upper-left and lower-right corner of the drawing's bounding box. Afterwards, we mirror the resulting layout vertically and horizontally to obtain one where s and t are in their respective corners and not fixed. Notice that in general the property of being fixed is not symmetric, e.g., when $deg_{\mu}^{pert}(s) = 3$ and $deg_{\mu}^{pert}(t) = 2$ holds, s remains fixed while t becomes fixed as well. For a non-fixed vertex, we introduce an operation that is referred to as forming or creating a *nose*; see Figure 3.5b, where t has been moved downwards at the cost of a bend. As a result, the west port of t is no longer occupied.

P-node case: Let μ be a P-node. By Lemma 2.4.1, for a child μ' of μ, it holds that $deg_{\mu'}^{pert}(s) \leq 2$ and $deg_{\mu'}^{pert}(t) \leq 2$. So, t can form a nose in μ', while s might be fixed in the case where $deg_{\mu'}^{pert}(s) = 2$. Notice that there exists at most one such child due to the degree restriction. We distinguish two cases based on the existence of an (s,t)-edge.

In the first case, assume that there is no (s,t)-edge, i.e., there is no child that is a Q-node. We draw the children of μ from top to bottom such that a possible child in which s is fixed, is drawn topmost (see μ_1 in Figure 3.6a). In the second case, we draw the (s,t)-edge at the top and afterwards the remaining children (see Figure 3.6b). Of course,

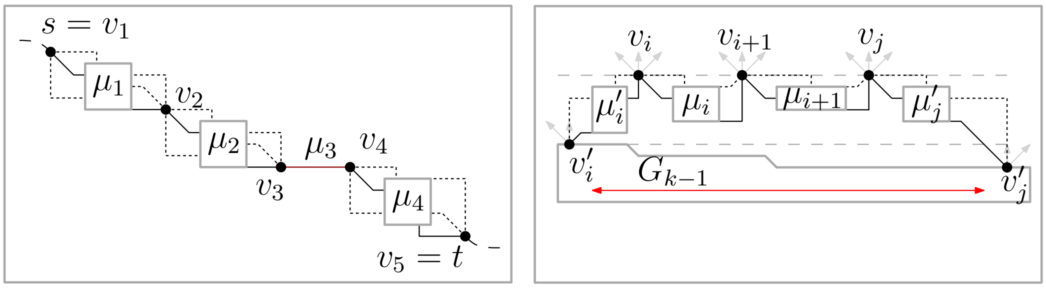


Figure 3.7: (a) S-node with children $\mu_1, \ldots, \mu_4$; μ_3 is a Q-node representing the edge (v_3, v_4). In all figures optional edges are drawn dotted. (b) Example for a chain $v_i, \ldots, v_j$ with virtual edges representing $\mu_i, \ldots, \mu_{j-1}$ in the R-node case.

this works only if s is not fixed in any of the other children. Let μ' be such a potential child where s is fixed, i.e., $deg_{\mu'}^{pert}(s) = 2$, and thus, the only child that remains to be drawn. Here, we use the property of interchangeability to "unfix" s in μ'. As a result s can form a nose, whereas t may now be fixed in μ' when $deg_{\mu'}^{pert}(t) = 2$ holds, as in Figure 3.6b. However, then $deg_{\mu}^{pert}(t) = 3$ follows. Notice that the presence of an (s, t)-edge implies that the parent of μ is not the root of $\mathcal{T}$, since this would induce a pair of parallel edges. Hence, by IP-2 we are allowed to fix t in μ. Port assignment and area requirements comply in both cases with our invariant properties.

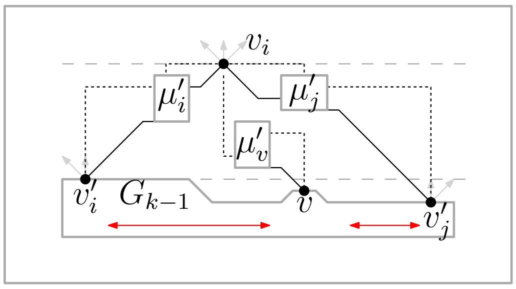
S-node case: We place the drawings of the children, say $\mu_1, \ldots, \mu_\ell$, of an S-node μ in a "diagonal manner" such that their corners touch as in Figure 3.7a. In case of Q-nodes being involved, we draw their edges as horizontal segments (see, e.g., edge (v_3, v_4) in Figure 3.7a that corresponds to Q-node μ_3). Observe that s and t inherit their port assignment and pertinent degree from μ_1 and μ_ℓ, respectively, i.e., $deg_{\mu}^{pert}(s) = deg_{\mu_1}^{pert}(s)$ and $deg_{\mu}^{pert}(t) = deg_{\mu_\ell}^{pert}(t)$. So, we may assume that s is fixed in μ, if s is fixed in μ_1. Similarly, t is fixed in μ, if t is fixed in μ_ℓ. By IP-2, t is not allowed to be fixed in the case where the parent of μ is the root of $\mathcal{T}$. However, Lemma 2.4.2 states that we can choose the root such that t is not fixed in that case, and thus, complies with IP-2. Since we only concatenated the drawings of the children, IP-1 and IP-3 are satisfied.

R-node case: For the case where μ is an R-node with poles $\mathcal{P}_\mu = \{s, t\}$, we follow the basic idea of the triconnected algorithm of the previous section and describe the modifications necessary to handle the drawing of the children of μ. To do so, we assume the worst case where no child of μ is a Q-node. Let μ_{uv} denote the child that is represented by the virtual edge $(u, v) \in E^{skel}_\mu$. Notice that due to Lemma 2.4.1, $deg^{pert}_{\mu_{uv}}(u) \leq 2$ and $deg^{pert}_{\mu_{uv}}(v) \leq 2$ holds. Hence, with IP-2 we may assume that at most one out of u and v is fixed in μ_{uv}. We choose the first partition in the canonical ordering to be $P_0 = \{s, t\}$ and distinguish again between whether the partition to be placed next is a chain or a singleton.

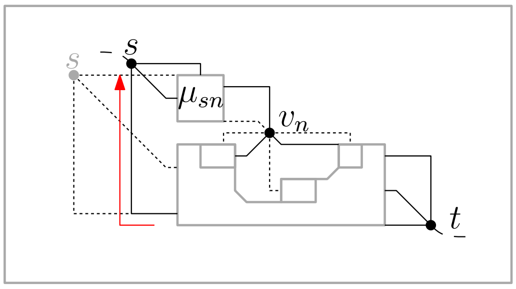
In case of a chain, say $P_k = \{v_i, \dots, v_j\}$ with two neighbors v'_i and v'_j in G_{k-1}, we have to replace two types of edges with the drawings of the corresponding children: the edges $(v_i, v_{i+1}), \dots, (v_{j-1}, v_j)$ representing the children $\mu_i, \dots, \mu_{j-1}$ and (v'_i, v_i) ((v_j, v'_j), resp.) representing μ'_i (μ'_j, resp.). We place the vertices of P_k on a horizontal line high enough above G_{k-1} such that every drawing may fit in-between it and G_{k-1}. Then, we insert the drawings aligned below the horizontal line and choose for $i \leq l < j$, v_l to be the fixed node in μ_l, whereas in μ'_i (μ'_j, resp.), we set v_i (v_j, resp.) to be fixed. Hence, for $i \leq l < j$, v_{l+1} may form a nose in μ_l pointing upwards while v'_i and v'_j form each one downwards as depicted in Figure 3.7b. For the extra height and width, we stretch the drawing horizontally.

For the case where $P_k = \{v_i\}$ and $i \neq n$ is a singleton, we only outline the difference which is a possible third edge (v_i, v) to G_{k-1} representing say μ'_v. While the other two involved children, say μ'_i and μ'_j, are handled as in the chain-case, μ'_v requires extra height now and we may place v_i such that μ'_v fits below μ'_j as in Figure 3.8a. Notice that $deg^{pert}_{\mu'_v}(v_i) = 1$ holds and therefore by IP-2 both v_i and v are not fixed in μ'_v. Hence, forming a nose at v_i and v as in Figure 3.8a is feasible.

It remains to describe the special case where the last singleton $P_k = \{v_n\}$ is placed. Since $s, t \in P_0$, both have not been fixed yet. We



(a)



(b)

Figure 3.8: (a) Singleton v_i with possibly three incident virtual edges representing μ'_i, μ'_v, μ'_j. (b) Placing v_n and moving up s which might be fixed in μ_{sn}.

proceed as in the triconnected algorithm and move $s = v_1$ above v_n as depicted in Figure 3.8b, high enough to accommodate the drawing of the child μ_{sn} represented by the edge (s, v_n). Since we may require v_n to form a nose in μ_{sn} as in Figure 3.8b, we choose s to be fixed in μ_{sn}. However, we are allowed by IP-2 to fix s since t remains unfixed. For the area constraints of IP-1, we argue as follows: Although some diagonal segments may force us to stretch the whole drawing by its height, the height of the drawing has been kept linear in the size of G_μ^{pert}. Since we increase the width by the height a constant number of times per step, the resulting width remains quadratic.

Root case: For the root of $\mathcal{T}$ we distinguish two cases: In the first case, there exists a vertex $v \in V$ with $\deg(v) \leq 3$. Then, we choose as root a Q-node μ that represents one of its three incident edges and orient the poles $\{s, t\}$ such that $t = v$. Hence, for the child μ' of μ follows $deg_{\mu'}^{pert}(t) \leq 2$. In the other case, i.e., for every $v \in V$ we have $\deg(v) = 4$, we choose a Q-node that is not adjacent to an S-node, whose existence is guaranteed by Lemma 2.4.2. In both cases, we may form a nose with t pointing downwards and draw the edge as in the triconnected algorithm.

Theorem 3.2.3. *Given a biconnected 4-planar graph G, we can compute in $O(n)$ time an octilinear drawing of G with at most one bend per edge on an $O(n^2) \times O(n)$ integer grid.*

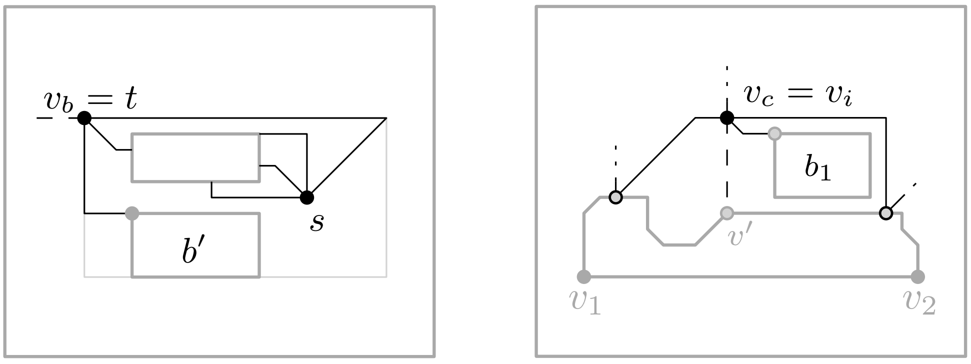
Proof. The SPQR-tree $\mathcal{T}$ can be computed in $O(n)$-time and its size is linear to the size of G [65]. The pertinent degrees of the poles at every node can be pre-computed by a bottom-up traversal of $\mathcal{T}$. Drawing a P-node requires constant time; S- and R-nodes require time linear to the size of the skeleton. However, the sum over all skeleton edges is linear, as every virtual edge corresponds to a tree node. □

3.2.4 Simply-Connected Graphs

After having shown that we can cope with biconnected 4-planar graphs, we turn our attention to the connected case. We start by computing the BC-tree of G and root it at some arbitrary B-node. Every B-node, except the root, contains a designated cut vertex that links it to the parent. A *bridge* for a biconnected component consists only of a single edge. Similar to the biconnected case, we define an invariant for the drawing of a subtree: The cut vertex that links the subtree to the parent is located in the upper left corner of the drawing's bounding box.

Any subgraph, say G_b, induced by a non-bridge biconnected component can be laid out using the biconnected algorithm. However, to construct a drawing that satisfies our invariant we have to take care of two problems. First, the cut vertex, say v_b, that links G_b to the parent, has to be drawn in the upper-left corner of the drawing of the subtree. Second, there may be other cut vertices of G in G_b to which we have to attach their corresponding subtrees.

For the first problem we describe how to root the SPQR-tree $\mathcal{T}_b$ for G_b so that v_b is located in the upper-left corner. There are at least two Q-nodes having v_b as a pole (as G_b is biconnected) and the degree of v_b in G_b is at most 3. In the biconnected case, we distinguished for the root of the tree between whether there exists $v \in V$ with $\deg(v) \leq 3$ or not. Hence, we may choose for the root of $\mathcal{T}_b$ a Q-node having v_b as a pole and orient it such that $v_b = t$, thus, satisfying $\deg(t) \leq 3$. Then, we flip the final drawing of G_b such



(a) **(b)**

Figure 3.9: (a) Rooting the SPQR-tree such that v_b is in the upper-left corner. (b) Attaching a subtree via a bridge to a cut vertex v_c in an R-node. The dashed edge (v_i, v') may only be present if $v_i = v_n$.

that t is in the upper left corner (see Figure 3.9a).

Next, we address the second problem. Let v_c be a cut vertex in G_b that is not the link to the parent. If v_c has degree 3, then it may occur in the pertinent graph of every node. However, in this case we only have to attach a subtree of the BC-tree that is connected via a bridge. This poses no problem, as there are enough free ports available at v_c and we can afford a bend at the bridge. We only consider S- and R- nodes here since the poles of P-nodes occur in the pertinent graphs of the first two. For R-nodes we assume that the south-east port at v_c is free. So, we attach the drawing via the bridge by creating a bend as in Figure 3.9b. In the diagonal drawing of an S-node, the north-east port is free. So, we can proceed similar; see Figure 3.10a.

If v_c has degree 2 in G_b, it only occurs in the pertinent graph of an S-node; see v_3 in Figure 3.10a. However, we may no longer assume that the bridge is available. As a result, we can not afford a bend and have to deal with two incident edges instead of one. We modify the drawing by exploiting the two real edges incident to v_c in the S-nodes layout to free the east and south east port; see v_2 in Figure 3.10a. This enables us to attach the subtrees drawing without modifying it. We finish this section by dealing with the most simple case where there are only bridges attached to a cut vertex. The idea is illustrated in Figure 3.10b and matches our layout specification.

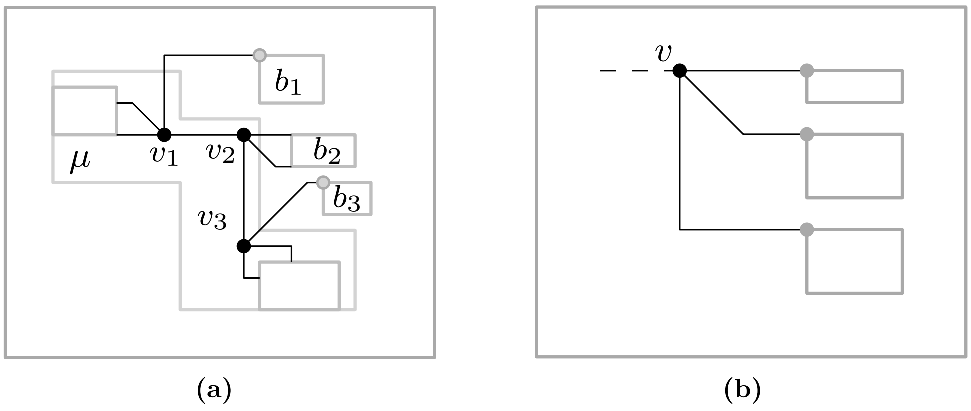


Figure 3.10: (a) All possible situations at an S-node μ. For attaching b_2 to v_2, the layout had to be modified. (b) A cut vertex where all of its children are attached via bridges.

Theorem 3.2.4. *Given a connected 4-planar graph G, we can compute in $O(n)$ time an octilinear drawing of G with at most one bend per edge on an $O(n^2) \times O(n)$ integer grid.*

Proof. Decomposing a connected graph into its biconnected components takes linear time. It remains the area property. Inserting a subtree with n vertices and the given dimensions into the drawing of an R- or S-node clearly increases the width of the drawing by at most $O(n^2)$ and the height by at most $O(n)$. Hence, the total drawing area is cubic, as desired. □

3.3 Drawing 5-Planar Graphs with One Bend Per Edge

Analogously to the 4-planar graphs, we will first give a construction for triconnected graphs, then extend to biconnected graphs using SPQR-trees and finally to simply-connected graphs using the BC-tree. Additionally we will show that the construction by the triconnected algorithm may result in super-polynomial area in the worst case.

3.3.1 Triconnected Graphs

Let $G = (V, E)$ be a triconnected 5-planar graph and $\Pi = \{P_0, \ldots, P_m\}$ be a canonical order of G. We place the first two partitions P_0 and P_1 of Π, similar to the case of 4-planar graphs. Again, we assume that we have already constructed a drawing for G_{k-1} and now we have to place P_k, for some $k = 2, \ldots, m-1$. We further assume that the x- and y-coordinates are computed simultaneously so that the drawing of G_{k-1} is planar and horizontally stretchable in the following sense: If $e \in E(G_{k-1})$ is an edge incident to the outer face of G_{k-1}, then there is always a cut which crosses e and can be utilized to horizontally stretch the drawing of G_{k-1}. This is guaranteed by our construction which makes sure that in each step the edges incident to the outer face have a horizontal segment. In other words, one can define a cut through every edge incident to the outer face of G_{k-1} (*stretchability-invariant*).

If $P_k = \{v_i, \ldots, v_j\}$ is a chain, it is placed exactly as in the case of 4-planar graphs, but with different port assignment. Recall that by v_i' (v_j', resp.) we denote the neighbor of v_i (v_j, resp.) in G_{k-1}. Among the available northern ports of vertex v_i' (v_j', resp.), edge (v_i, v_i') ((v_j, v_j'), resp.) uses the eastern-most unoccupied port of v_i' (western-most unoccupied port of v_j', resp.); see Figure 3.11a. If P_k does not fit into the gap between its two adjacent vertices v_i' and v_j' in G_{k-1}, then we horizontally stretch G_{k-1} between v_i' and v_j' to ensure that the horizontal distance between v_i' and v_j' is at least $|P_k|+1$. This can always be accomplished due to the stretchability invariant, as both v_i' and v_j' are on the outer face of G_{k-1}. Potential crossings introduced by edges of P_k containing diagonal segments can be eliminated by employing similar cuts to the ones presented in the case of 4-planar graphs. So, we may assume that G_k is plane. Also, G_k complies with the stretchability invariant, as one can define a cut that crosses any of the newly inserted edges of P_k and then follows one of the cuts of G_{k-1} that crosses an edge between v_i' and v_j'.

In case of a singleton $P_k = \{v_i\}$ of degree 3 or 4, our approach is very similar to the one for 4-planar graphs. Here, we mostly focus on the case where v_i is of degree five. In this case, we have to deal with two additional edges (called *nested*) that connect v_i with G_{k-1}, say (v_i, v) and (v_i, v'); see

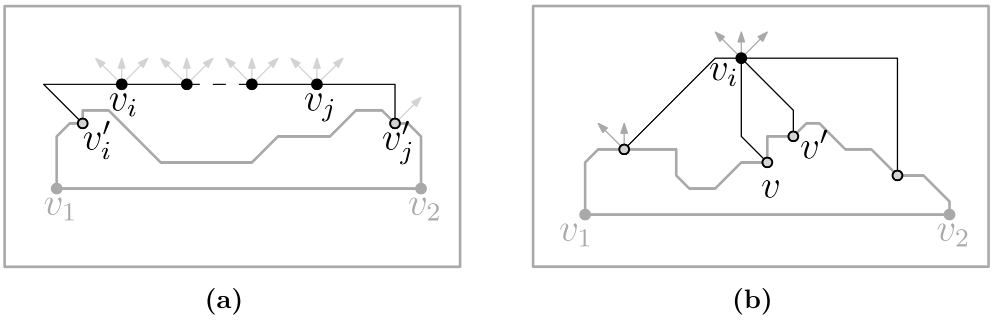


Figure 3.11: (a) Horizontal placement of a chain $P_k = \{v_i, \ldots, v_j\}$. (b) Placement of a singleton $P_k = \{v_i\}$ of degree five.

Figure 3.11b. Such a pair of edges does not always allow vertex v_i to be placed along the next available horizontal grid line; v_i's position is more or less prescribed, as each of v and v' may have only one northern port unoccupied. However, a careful case analysis on the type of ports (i.e., north-west, north or north-east) that are unoccupied at v and v' in conjunction with the fact that G_{k-1} is horizontally stretchable shows that we can always find a feasible placement for v_i (usually far apart from G_{k-1}). Potential crossings due to the remaining edges incident to v_i are eliminated by employing similar cuts to the ones presented in the case of 4-planar graphs. So, we may assume that G_k is planar. Similar to the case of a chain, we prove that G_k complies with the stretchability invariant. In this case special attention should be paid to avoid crossings with the nested edges of v_i, as a nested edge may contain no horizontal segment. Note that the case of the last partition $P_m = \{v_n\}$ is treated in the same way, even if v_n is potentially incident to three nested edges; see Figure 3.12.

To complete the description of our approach it remains to describe how edge (v_1, v_2) is drawn. By construction both v_1 and v_2 are on a horizontal line. So, (v_1, v_2) can be drawn using two diagonal segments that form a bend pointing downwards; see Figure 3.12.

Theorem 3.3.1. *Given a triconnected 5-planar graph G, we can compute in $O(n)$ time an octilinear drawing of G with at most one bend per edge.*

Proof. In contrast to the corresponding proof for 4-planar graphs (see Theorem 3.2.2), in the case of 5-planar graphs the x and y-coordinates are not

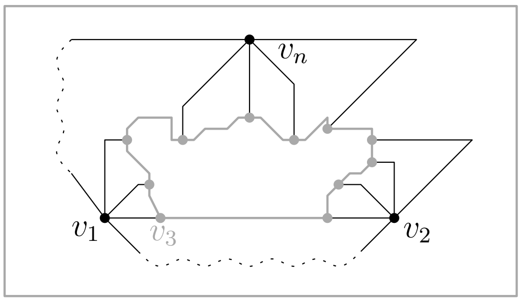


Figure 3.12: Final layout (the shape of the dotted edges can be obtained by extending the stubs until they intersect).

independent. However, since the y-coordinates of the vertices that have been placed already do not change afterwards, we can still use the shifting method of Kant and keep the running time of our algorithm linear. More specifically, in order to determine the y-coordinate of a singleton vertex, we only use the x-distances between its neighbors that have already been drawn. The x-distances can be computed in time proportional to the length of the path connecting them on the outer face of the graph in the tree structure of Kant. Since each such computation will not involve the same set of vertices more than once, the total time needed is linear in total (with respect to the number of the graph's vertices). Note that the aforementioned procedure is not necessary for chains, as they do not impose restriction on the drawing's height. □

Recall that when placing a singleton $P_k = \{v_i\}$ that has four edges to G_{k-1}, the height of G_k is determined by the horizontal distance of its neighbors along the outer face of G_{k-1}, which is bounded by the actual width of the drawing of G_{k-1}. On the other hand, when placing a chain P_k the amount of horizontal stretching required in order to avoid potential crossings is limited by the height of the drawing of G_{k-1}. Unfortunately, this connection implies that for some input triconnected 5-planar graphs our drawing algorithm may result in drawings of super-polynomial area, as the following theorem suggests.

Theorem 3.3.2. *There exist infinitely many triconnected 5-planar graphs for which our drawing algorithm produces drawings of super-polynomial area.*

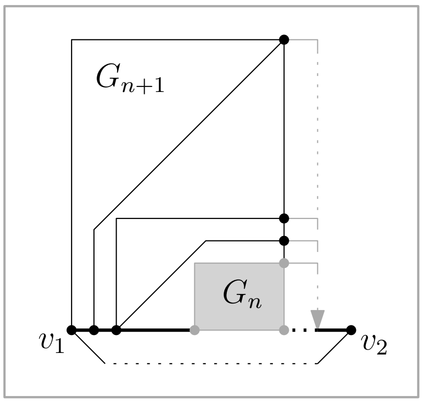


Figure 3.13: A recursive construction of an infinite class of 5-planar graphs requiring super-polynomial drawing area.

Proof. Figure 3.13 illustrates a recursive construction of an infinite class of 5-planar triconnected graphs with this property. The base of the construction is a "long chain" connecting v_1 and v_2 (refer to the bold drawn edges of Figure 3.13). Each next member, say G_{n+1}, of this class is constructed by adding a constant number of vertices (colored black in Figure 3.13) to its immediate predecessor member, say G_n, of this class, as illustrated in Figure 3.13. If W_n and H_n is the width and the height of G_n, respectively, then it is not difficult to show that $W_{n+1} > 2W_n$ and $H_{n+1} > 2H_n$, which implies that the required area is asymptotically exponential. □

3.3.2 Biconnected Graphs

For the 4-planar case we defined several invariants in order to keep the area of the resulting drawings polynomial. Since we drop this requirement now we can define a (simpler) new invariant for the biconnected 5-planar case. When considering a node μ in $\mathcal{T}$ and its poles $\mathcal{P}_\mu = \{s, t\}$, then in the drawing of G_μ^{pert}, s and t are horizontally aligned at the bottom of the drawing's bounding box as in Figure 3.14a. If an (s,t)-edge is present, it can be drawn at the bottom. An (s,t)-edge only occurs in the pertinent graph of a P-node (and Q-node). Again, we use the term *fixed* for a pole-node that is not allowed to form a nose. We maintain the following properties through the

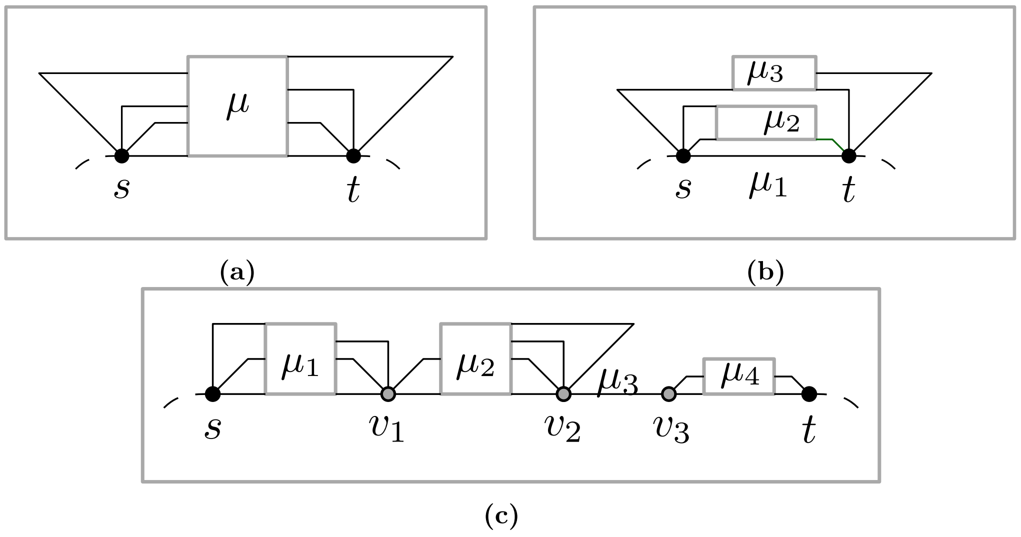


Figure 3.14: (a) Layout specification; s and t are located at the bottom. (b) P-node with an (s,t)-edge from a Q-node μ_1. s and t form a nose in μ_2, μ_3. (c) S-node example with four children $\mu_1, \ldots, \mu_4$.

recursive construction process: In S- and R- nodes, s and t are not fixed. In P- and Q-nodes, only one of them is fixed, say s. But similar to the 4-planar biconnected case, we may swap their roles.

P-node case: Let μ be a P-node. It is not difficult to see that μ has at most 4 children; one of them might be a Q-node, i.e., an (s,t)-edge, which can be drawn at the bottom as a horizontal segment. Since P-nodes are not adjacent to each other in $\mathcal{T}$, the remaining children are S- or R-nodes. By our invariant we may form noses enabling us to stack them as in Figure 3.14b, as s and t are not fixed in them.

S-node case: Let μ be an S-node with children $\mu_1, \ldots, \mu_l$. Instead of the diagonal layout used earlier, we now align the drawings horizontally; see Figure 3.14c. In the S-node case, the poles inherit their pertinent degree from the children and the same holds for the property of being fixed. However, by our new invariant this is forbidden, as it clearly states that s and t are not fixed. It is easy to see that when μ_1 is a P-node, s is fixed by the invariant in μ_1. In this case, we swap the roles of the poles in μ_1 such that s is not fixed. However, the other pole of

μ_1, say v_1, is fixed now. Since the skeleton of an S-node is a cycle of length at least three, $v_1 \neq t$ holds. As a result, both s and t are not fixed in the resulting drawing.

R-node case: To compute a layout of an R-node, we employ the triconnected algorithm (with $s = v_1$ and $t = v_2$). So, let μ be an R-node and μ_e a child of μ that corresponds to the virtual edge $e = (u, v)$ in G_μ^{skel}. Then, $deg_{\mu_e}^{pert}(u) \leq 3$ and $deg_{\mu_e}^{pert}(v) \leq 3$ holds. When inserting the drawing of $G_{\mu_e}^{pert}$, we require at most three consecutive ports at u and v for the additional edges. As the triconnected algorithm assigns ports in a consecutive manner based on the relative position of the endpoints, we modify the port assignment so that an edge may have more than one port assigned. To do so, we assign each edge $e = (u, v)$ in G_μ^{skel} a pair $(deg_{\mu_e}^{pert}(u), deg_{\mu_e}^{pert}(v)) \in \{1, 2, 3\}^2$ that reflects the number of ports required by this edge at its endpoints. Then, we extend the triconnected algorithm such that when a port of u is assigned to an edge $e = (u, v)$, $deg_{\mu_e}^{pert}(u) - 1$ additional consecutive ports in clockwise or counterclockwise order are reserved. The direction depends on the different types of edges that we will discuss next.

The simplest type of edges are the ones among consecutive vertices v_i, v_{i+1} of a chain. Recall that $P_0 = \{v_1, v_2\}$ is a special case and the edge (v_1, v_2) is drawn differently. Also, the edges from P_0 to P_1 are drawn as horizontal segments; see Figure 3.12. For each such edge we reserve the additional ports at v_i in counterclockwise order and at v_{i+1} in clockwise order; see Figure 3.15a. So, we can later plug the drawing of the children into the layout as in Figure 3.15b without forming noses. The second type of edges are the ones that connect $P_k = \{v_i, \ldots, v_j\}$ to v_i' and v_j' in G_{k-1}. No matter if P_k is a singleton or a chain, we proceed by reserving the ports as in the previous case, i.e., at v_i clockwise, (v_j counterclockwise, resp.) and at v_i' counterclockwise (v_j' clockwise); see Figure 3.15c. In case where (v_i, v_i') or (v_j, v_j') is a virtual edge, we choose the poles such that v_i (v_j, resp.) is fixed in $\mu_{(v_i,v_i')}$ ($\mu_{(v_j,v_j')}$, resp.). Thus, we can create a nose with v_i' (v_j', resp.). Having exactly the ports required at both endpoints, we insert the drawing by replacing

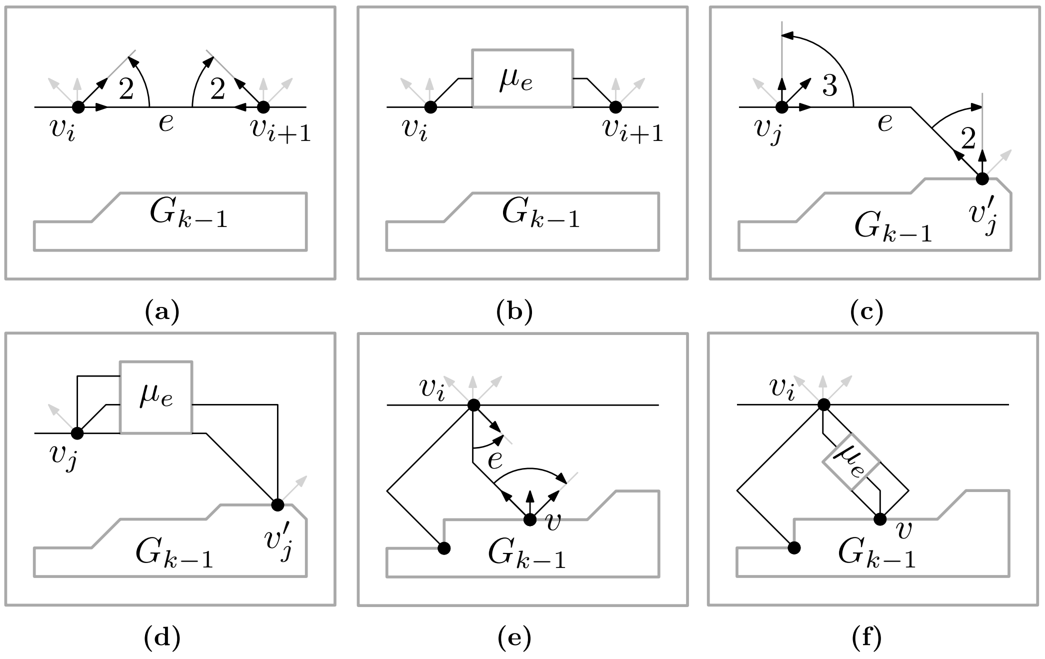


Figure 3.15: (a) Virtual edge $e = (v_i, v_{i+1})$ connecting two consecutive vertices of a chain. At both endpoints the drawing of μ_e requires two ports. (b) Replacing e in (a) with the corresponding drawing of the child μ_e. (c) Example of an edge $e = (v_j, v'_j)$ that requires three ports at v_j and two at v'_j. (d) Inserting the drawing of μ_e into (c) with v_j being fixed and v'_j forming a nose. (e) Reserving ports for the nested edges. A single port for a real edge is reserved and then two ports for the virtual edge e $= (v_i, v)$. (f) Final layout after inserting the drawing of μ_e.

the bend with a nose as in Figure 3.15d. The remaining edges from P_k to G_{k-1} in case of a singleton $P_k = \{v_i\}$ can be handled similarly; see Figure 3.15. Notice that during the replacement of the edges, the fixed vertex is always the upper one. The only exception are the horizontal drawn edges of a chain. There, it does not matter which one is fixed, as none of the poles has to form a nose.

Root case: We root $\mathcal{T}$ at an arbitrarily chosen Q-node representing a real edge (s, t). By our invariant we may construct a drawing with s and t at the bottom of the drawing's bounding box, hence, we draw the

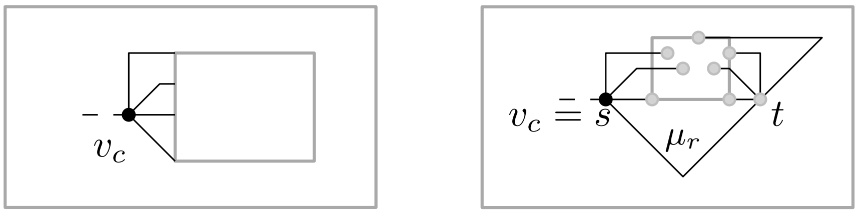


Figure 3.16: (a) Layout scheme for a BC-subtree rooted at v_c. (b) Rooting $\mathcal{T}_b$ at a Q-node μ_r.

edge (s, t) below the bounding box with a ninety degree bend using the south east port at s and south west port at t.

Theorem 3.3.3. *Given a biconnected 5-planar graph G, we can compute in $O(n)$ time an octilinear drawing of G with at most one bend per edge.*

Proof. We have shown that the ability to rotate and scale suffices to extend the result from 4-planar to 5-planar at the expense of the area. Similar to the 4-planar case, computing $\mathcal{T}$ takes linear time. Hence, the overall running time is governed by the triconnected algorithm. □

3.3.3 Simply-Connected Graphs

In the following, we only outline the differences in comparison with the corresponding 4-planar case. As an invariant, the drawing of every subtree should conform to the layout depicted in Figure 3.16a. For a single biconnected component b, let v_c refer to the cut vertex linking it to the parent. As root for the SPQR-tree $\mathcal{T}_b$ of G_b, we again choose a Q-node μ_r whose real edge is incident to v_c; see Figure 3.16b. Hence, the layout generated by the biconnected approach matches this scheme.

It remains to show that we can attach the children. Since we are able to scale and rotate, we keep things simple and look for suitable spots to attach them. Recall that in the drawings of S-nodes and chains in R-nodes all southern ports are free. Hence, we may rotate the drawings of the subtrees and attach the at most three (two for a chain) edges to v_c there (refer to Figure 3.17a for an example of a chain). The only exception are the singletons.

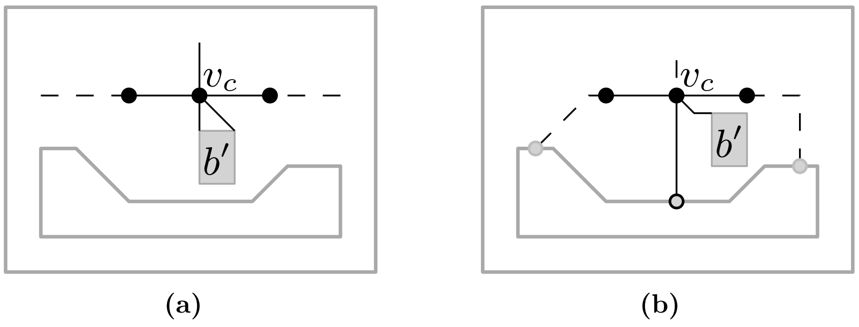


Figure 3.17: (a) Attaching a subtree at a chain and in (b) at a singleton inside an R-node.

Assume that v_i is a singleton that has one nested edge attached. Hence, it has degree four, leaving us with a single bridge to attach the component; Figure 3.17b. However, this does not hold in case $v_i = v_n$. Consider the case where v_n has a nested edge and we have to attach a subtree that requires two ports. As a result v_n has degree 3 in G_b and, thus, all northern ports are free.

Theorem 3.3.4. *Given a connected 5-planar graph G, we can compute in $O(n)$ time an octilinear drawing of G with at most one bend per edge.*

Proof. We described how to attach any subtree to cut vertices inside a biconnected component. Furthermore, the component itself complies with the layout scheme. In addition, this scheme enables us to compose such drawings at a cut vertex using rotations. The running time follows from the fact that the decomposition of a connected graph into its biconnected components takes linear time. □

3.3.4 Improving 5-Planar Drawings

One of the results of the study by Purchase [100] is, that bends have a large impact on the perceived quality of a drawing. We can construct planar octilinear drawings for 5-planar graphs with at most one bend per edge, but it is possible that at some bends angles of 45° occur, as depicted in Figure 3.18a. We call such bends *sharp* bends. To improve the readability of a drawing, which is negatively affected by sharp bends, it is possible to

Figure 3.18: (a) A sharp bend that may occur during the 5-planar construction. (b) Removing the sharp bend by allowing one more bend.

remove them at the cost of one additional bend as depicted in Figure 3.18b. The goal of minimizing the number of bends per edge is to obtain a drawing which is aesthetically pleasing. Since this purpose may be better served by allowing one more bend rather than having a 45° angle, it makes sense to invest one more bend on such edges to avoid them. Note that it is always enough to allow for one more bend to avoid sharp bends.

3.4 6-Planar Graphs Require 2 Bends

In this section, we show that it is not always possible to construct a planar octilinear drawing of a given 6-planar graph with at most one bend per edge. In particular, we present an infinite class of 6-planar graphs, which do not admit planar octilinear drawings with at most one bend per edge.

Theorem 3.4.1. *There exists an infinite class of 6-planar graphs which do not admit planar octilinear drawings with at most one bend per edge.*

Proof. Our proof is heavily based on the following observation: If the outer face $\mathcal{F}(\Gamma(G))$ of a given planar octilinear drawing $\Gamma(G)$ consists of exactly three vertices, say v, v' and v'', that have the so-called *outerdegree-property*, i.e., $deg(v) = deg(v') = 6$ and $5 \leq deg(v'') \leq 6$, then it is not feasible to draw all edges delimiting $\mathcal{F}(\Gamma(G))$ with at most one bend per edge; one of them has to be drawn with (at least) two bends in $\Gamma(G)$. Next, we construct a specific maximal 6-planar graph, in which each face has at most one vertex of degree 5 and at least two vertices of degree 6; see Figure 3.19a. This specific graph does not admit a planar octilinear drawing with at most one

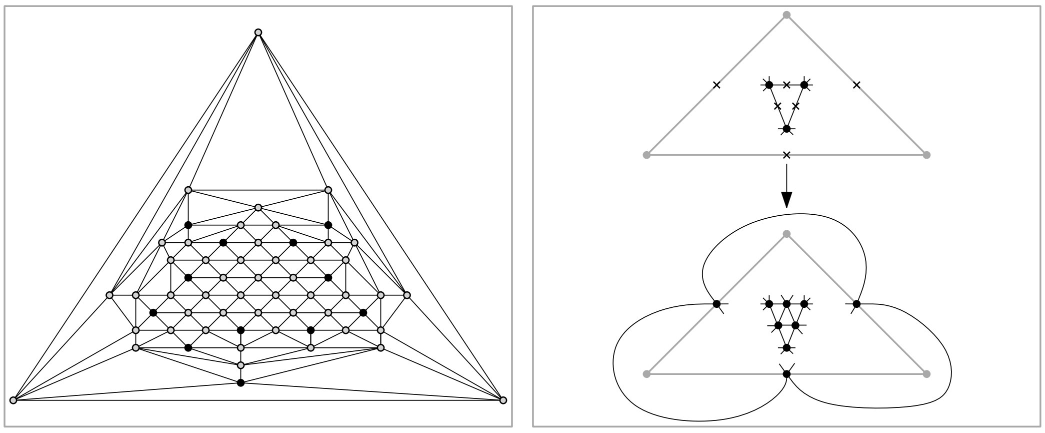

(a) (b)

Figure 3.19: (a) A maximal 6-planar graph in which each face has at most one vertex of degree 5 (black-colored vertices) and at least two vertices of degree 6 (gray-colored vertices). From Euler's formula for maximal planar graphs, it follows that any graph with this property must have at least 12 vertices of degree 5. Hence, this is the smallest graph with this property. (b) Illustration of the recursive construction (crosses mark vertices introduced during the subdivision of edges).

bend, as its outer face is always bounded by three vertices that have the outerdegree-property.

To obtain an infinite class of 6-planar graphs with this property, we give the following recursive construction. We first subdivide each edge by introducing a new vertex (see Figure 3.19b). For each original face there are now three new vertices. By pairwise connecting them as suggested in the figure we obtain four new faces for each original face. Let w be a vertex introduced while splitting edge $e = (u, v)$ and let f and f' be the two faces adjacent to e. Vertex w will be connected with u and v, with the two new vertices introduced in face f and with the two new vertices introduced in face f', giving w a degree of six. Note that this holds for all vertices introduced, including the ones on the outer face, as can be seen in Figure 3.19b.

The newly created graph G' is, by construction, maximal planar. Furthermore, the degree of the vertices of the original graph does not change during this construction, and all newly introduced vertices have degree six. So, for each face of G', the adjacent vertices have the outerdegree-property. □

3.5 Summary

In this chapter we presented algorithms to compute planar octilinear drawings of 4- and 5-planar graphs with at most one bend per edge. Our algorithms require linear running time and compute drawings in cubic area for 4-planar graphs and may require super-polynomial area for 5-planar graphs. The basis of our algorithms is an incremental approach based on the canonical ordering for triconnected graphs. Using the SPQR-tree and the BC-tree we extended the algorithms to biconnected and connected graphs. For 6-planar graphs we gave a construction of an infinite family of graphs that can not be drawn using only one bend per edge.

With our construction we close the gap between known results: 3-planar graphs always admit a planar octilinear drawing without bends and for all planar graphs with vertex degree up to eight only methods were known to construct planar octilinear drawings with two bends per edge.

We published our results in [1] and [2] and submitted a detailed version to the Journal of Graph Algorithms and Applications.

Our work raises several open problems:

- Is it possible to construct planar octilinear drawings of 4-planar (5-planar, resp.) graphs with at most one bend per edge in $o(n^3)$ (polynomial, resp.) area?
- Does any triangle-free 6-planar graph admit a planar octilinear drawing with at most one bend per edge?
- What is the complexity to determine whether a 6-planar graph admits a planar octilinear drawing with at most one bend per edge?
- What is the number of necessary slopes for bendless drawings of 4-planar graphs?
- Is it possible to extend our methods to non-planar graphs?

4 The Slanted Orthogonal Drawing Model

4.1 Introduction

Recall from Chapter 1, that in the classical orthogonal model vertices are drawn as points on an integer grid and edges as lines connecting those points using alternating horizontal and vertical line segments. This naturally implies a bound of 4 for the maximal number of edges connected to a vertex.

Purchase [100] found that the quality of a drawing is heavily affected by crossings, bends and symmetry. Crossings influence the perceived quality of a drawing the most, but bends and symmetry are also very important.

This motivated us to extend the classical orthogonal model to the *slanted orthogonal drawing* model, or *slog* for short. In the new model:

- Vertices are still drawn as points, but edges are drawn using horizontal, vertical and diagonal segments.
- Crossings are only allowed between diagonal segments.
- The minimum angle between two consecutive segments of an edge has to be 135°.

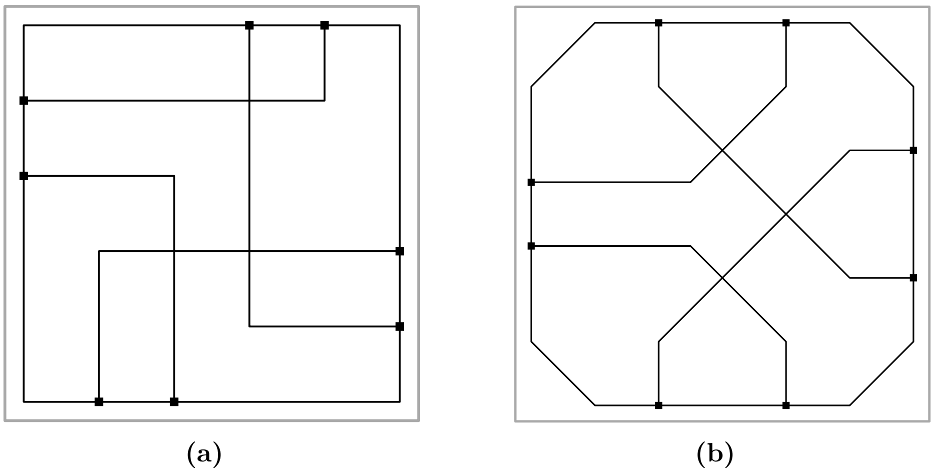


Figure 4.1: (a)-(b) Traditional orthogonal and slanted orthogonal drawings of the same graph, assuming fixed ports.

Figure 4.1a shows a classical orthogonal drawing and Figure 4.1b a drawing of the same graph in the new model. The minimum angle between consecutive segments does not allow classical bends, but results in *half-bends* (see Figures 4.2a and 4.2b). With this modifications we aim to improve the classical orthogonal model with respect to the layout aesthetics identified as important by Purchase [100].

First we will develop a method to compute a bend-optimal slog representation in Section 4.2 and prove theoretical bounds on the number of half-bends (Section 4.2.2) required by slog drawings. Then we will present a heuristic to compute slog drawings (Section 4.3) with almost optimal number of half-bends. To obtain bend-optimal slog drawings we then present a linear programming approach (Sections 4.4 and 4.5). After proving area bounds (Section 4.6) we experimentally evaluate the algorithms (Section 4.7).

For planar slog drawings, observe that the problem of minimizing the number of bends over all embeddings of an input planar graph of maximum degree 4 is NP-hard. This directly follows from [61], since the bends of a planar orthogonal drawing are in one to one correspondence with pairs of half-bends of the corresponding slanted orthogonal drawing. This negative result led us to adopt the TSM approach for our model. So, in the following, we assume that a planar representation of the input graph is given. Then,

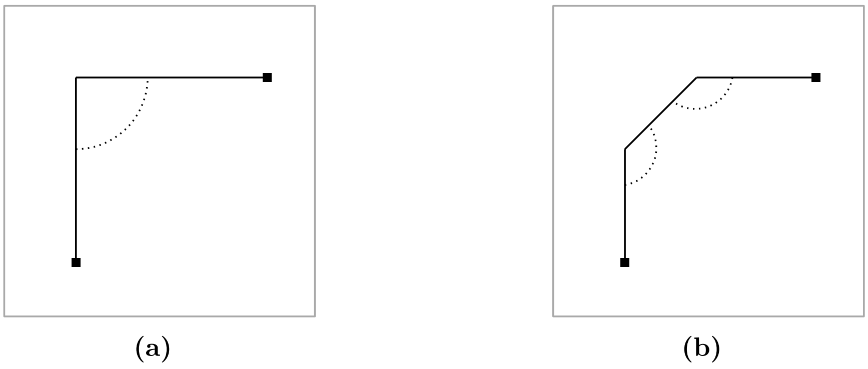


Figure 4.2: (a)-(b) Replacing a 90° bend by two half-bends of 135°.

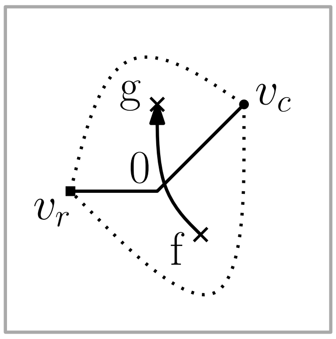
one can easily observe the following requirements: (I) all non-dummy vertices (referred to as *real vertices or r-vertices*) use orthogonal ports and, (II) all c-vertices use diagonal ports. This ensures that the computed drawing will be a valid slog drawing that corresponds to the initial planar representation. Recall that edges connecting real (crossing, resp.) vertices are referred to as *rr-edges* (*cc-edges*, resp.), and edges between r- and c-vertices as *rc-edges*.

4.2 Orthogonalization Using Network Flow

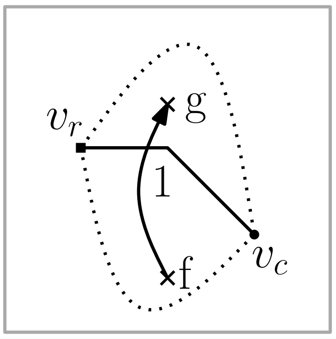
Recall from the introduction in Section 2.4 that an orthogonal representation of a graph G is the description of the shape of a drawing of G, that does not contain coordinates. The classical TSM-approach introduced in Section 2.4.1 uses a flow network to compute a bend-optimal orthogonal representation for a given plane graph with maximum degree four. We will now adopt this approach for our new model.

4.2.1 Modifying the Flow Network

We now present how to modify the algorithm of Tamassia [110], in order to obtain a slog representation of an input plane graph G with minimum number of half-bends. Recall that G contains two types of vertices, namely real and crossing vertices. Real (crossing, resp.) vertices use orthogonal (diagonal, resp.) ports. Observe that a pair of half-bends on an rr-edge of a slog drawing corresponds to a bend of an orthogonal drawing. The same



(a) Zero units of flow, left-turn from v_r to v_c



(b) One unit of flow, right-turn from v_r to v_c

Figure 4.3: Two configurations corresponding to zero or one unit of flow over an rc-edge; f and g are the two adjacent faces.

holds for half-bends on cc-edges. However, an rc-edge must switch from an orthogonal port (incident to the r-vertex) to a diagonal port (incident to the c-vertex). This implies that each rc-edge has at least one half-bend, and, in general, an odd number of half-bends.

Consider an rc-edge (v_r, v_c) that is incident to faces f and g (see Figure 4.3) and assume that the port of the real vertex v_r is fixed. Depending on the (diagonal) port on the crossing vertex v_c we obtain two different representations with the same total number of half-bends. To model this "free-of-cost" choice, we introduce an edge into the flow network connecting f and g with unit capacity and zero cost, i.e., through this edge just one unit of flow can be transmitted and this is for free. Hence, the first half-bend of each rc-edge is free of cost, as desired. For consistency we assume that, if in the solution of the min-cost flow problem there is no flow over (f, g), then there exists a left turn from the real to the crossing vertex on the bend before the crossing; otherwise a right turn, as illustrated in Figures 4.3a and 4.3b.

4.2.2 Properties of Bend-Optimal Slog Representations

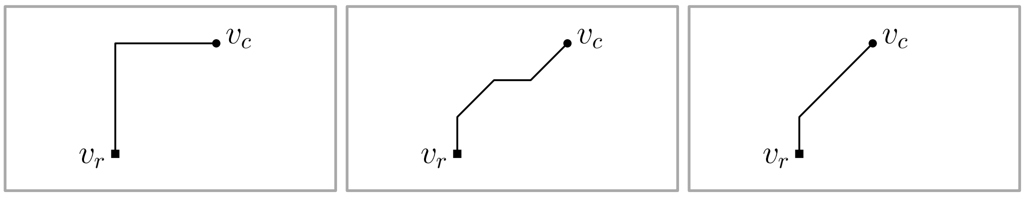
We prove that, for a planarized graph G, the computation of a slog representation with minimum number of half-bends that respects the embedding of G is always feasible. Then, we present a lower bound for the number of half-bends in optimal slog representations. In the following we assume that, together with a planarization, the embedding is also given.

Theorem 4.2.1. *For a planarized graph G of maximum degree 4, we can efficiently compute a slog representation with minimum number of half-bends respecting the embedding of G.*

Proof. The idea is to use a reduction to Tamassia's network flow algorithm. In particular, since the original flow network algorithm computes a (bend-optimal) orthogonal representation for the input plane graph, our algorithm will also compute a slog representation. In the following, we prove that this representation is also bend-optimal.

Assume that we are given an orthogonal representation O. We can uniquely convert O into a slog representation $S(O)$ by turning all crossing vertices counterclockwise by 45°. More precisely, the last segment of every rc-edge before the crossing vertex will become a left half-bend. Furthermore, every orthogonal bend is converted into two half-bends, bending in the same direction as the orthogonal bend (see Figures 4.2a and 4.2b). Note that the left half-bends at the crossings might neutralize with one of the half-bends originating from an orthogonal bend, if the orthogonal bend is turning to the right (see Figure 4.4). In this case, only the first one of the right half-bends remains. Note that this is the only possible saving operation. Therefore, since the number of rc-edges is fixed from the given embedding, a slog representation with minimum number of half-bends should minimize the difference between the number of orthogonal bends of O and the number of first right-bends on rc-edges. However, this is exactly what is done by our min-cost flow network formulation, as the objective is the minimization of the total number of bends in O without the first right-bends on rc-edges. □

This constructive approach can also be reversed such that for each slog representation S, we can construct a unique orthogonal representation $O(S)$. Clearly, $O(S(O)) = O$ and $S(O(S)) = S$. Note that this is true only for bend-minimal representations. If this is not the case, then one has to deal with staircases of subsequent bends; a case that cannot occur in min-cost flow computations. From the construction, we can also derive the following.



(a) Orthogonal input **(b)** Three half-bends **(c)** Reduction by 2 half-bends

Figure 4.4: A case in which two half-bends can be eliminated.

Corollary 4.2.1.1. *Let $S(O)$ be a slog representation and O a corresponding orthogonal representation. Let b_S, rb_S and rc_S be the number of half-bends, the number of first right-bends on rc-edges and the number of rc-edges in $S(O)$. Let also b_O be the number of orthogonal bends in O. Then, $b_S = 2 \cdot (b_O - rb_S) + rc_S$.*

The following theorem gives a lower bound for the number of half-bends in optimal slog representations.

Theorem 4.2.2. *The number of half-bends of a bend-minimal slog representation is at least twice the number of bends of its corresponding bend-minimal orthogonal representation.*

Proof. Let R_s^{opt} be the bend-optimal slog representation and R_o^{opt} be the bend-optimal orthogonal representation. Given a representation R (either slog or orthogonal), we denote by $hb(R)$ the number of half-bends and by $b(R)$ the number of bends of R. Assume for a proof by contradiction that it holds that $hb(R_s^{opt}) < 2b(R_o^{opt})$. We will show how to construct an orthogonal representation R_o' that has less bends than R_o^{opt}, and so obtain a contradiction.

To transform a slog representation into an orthogonal representation the diagonal ports on the c-vertices have to become orthogonal. To this end, we define the notion of *rotating* a vertex to the left or right, which means that the ports assigned to the edges incident to it are rotated cyclically into the respective direction and the number of bends on the edges changes according to Figure 4.5. We consider rc-edges always in the direction from the crossing to the real vertex. Note that if there is exactly one half-bend to the left from

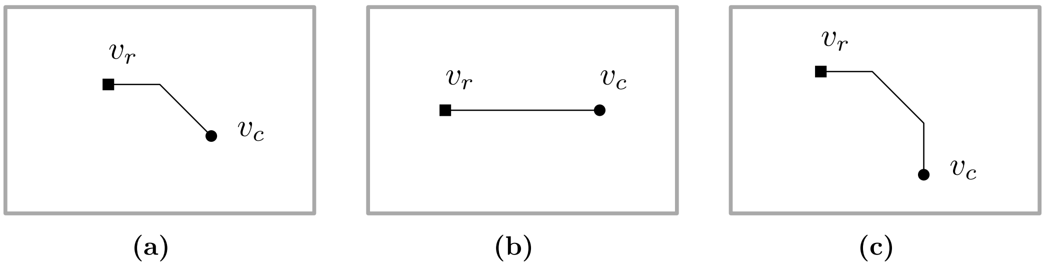


Figure 4.5: Illustration for the proof of Theorem 4.2.2: (a) An rc-edge with a left-turn from v_c to v_r. (b) Rotating c-vertex v_c by 45° to the left saves one half-bend. (c) Rotating c-vertex v_c by 45° to the right increases the number of half-bends.

a crossing-vertex v_c to a real-vertex v_r (see Figure 4.5a) and v_c is rotated by 45° to the left, the resulting drawing has zero bends on this edge (see Figure 4.5b); a rotation by 45° to the right would result in two half-bends (see Figure 4.5c).

By construction, all rr- and cc-edges have an even number of half-bends. In order to obtain R'_o, we replace each pair of half-bends on rr- and cc-edges of R_s^{opt} by an orthogonal bend (and reverse the process of Figures 4.2a and 4.2b). So, for rr- and cc-edges it holds that $hb(R_s^{opt} \setminus \{\text{rc-edges}\}) = 2b(R'_o \setminus \{\text{rc-edges}\})$.

On the other hand, all rc-edges have an odd number of half-bends. Similarly to the case of rr- and cc-edges, we replace each pair of half-bends except the one half-bend closest to the crossing vertex by an orthogonal bend in R'_o. Let $\mathcal{C} = \{C_1, \dots, C_l\}$ be the set of maximal connected components consisting only of c-vertices and the cc-edges between them. By maximality, it follows that $C_i \cap C_j = \emptyset$, for $i \neq j$. Also, observe that if $l = 0$, then we already would have a contradiction. Therefore, $l > 0$ must hold. Now, let $lhb(C_i)$ be the number of left half-bends and $rhb(C_i)$ be the number of right half-bends on the rc-edges connected to C_i that have not been replaced already. If $lhb(C_i) = rhb(C_i)$ for all $i = 1, 2, \dots, l$, then without loss of generality we rotate all vertices in C_i to the left. This implies that all left half-bends disappear and the right half-bends get replaced by orthogonal bends, which contradicts the assumption that R_o^{opt} is bend-optimal.

So, there has to be at least one C_i with $lhb(C_i) \neq rhb(C_i)$. Assume

$lhb(C_i) > rhb(C_i)$ for some $i = 1, 2, \ldots, l$. If we now rotate C_i to the left, then we obtain an orthogonal solution that has even less bends than twice the number of half-bends of the slog solution. Similarly, we can obtain a better orthogonal solution when $lhb(C_i) > rhb(C_i)$ by rotating C_i to the right.

By this construction, it holds that $2b(R'_o) \leq hb(R_s^{opt}) < 2b(R_o^{opt})$, which is a contradiction to the assumption that R_o^{opt} is optimal. $\square$

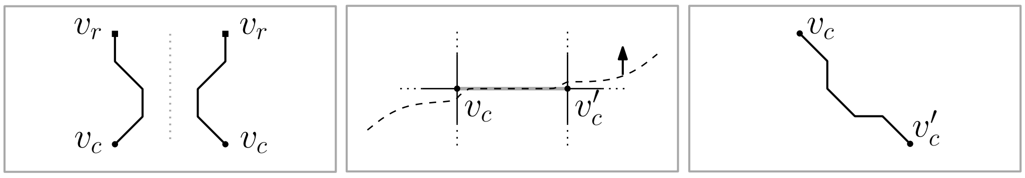
4.3 A Heuristic for Slog Drawings

In this section, we present a heuristic which, given an optimal slog representation, computes an actual drawing, which is close-to-optimal with respect to the total number of bends and requires quadratic area. This is a quite reasonable approach, since insisting on bend-optimal slog drawings may result in exponential area requirements, as we will shortly see in Section 4.6. The basic steps of our approach are outlined in Algorithm 1. In the following, we describe them in detail.

```
Input  : A slog representation $S$ of a given plane graph $G$.
Output: A slog drawing $\Gamma_s(G)$.
S1: Compute an orthogonal drawing $\Gamma(G)$ based on $S$
S2: Replace each orthogonal bend by 2 half-bends {see Figs.4.2a and 4.2b}
S3: Fix ports on rc-edges using the spoon gadget {see Fig.4.6a}
S4: Apply cuts to fix ports on cc-edges {see Figs.4.6b and 4.6c}
S5: Optimize the number of rc half-bends {see Figs.4.7a and 4.7b}
S6: Optimize the number of cc half-bends {see Figs.4.8a, 4.8b and 4.8c}
S7: Heuristically compact the drawing (as post-processing)
```

Algorithm 1: SPOON BASED ALGORITHM

In Step 1 of Algorithm 1, we compute an orthogonal drawing $\Gamma(G)$ based on the input slog representation. If there is flow on an edge e connecting faces f_i and f_j that we added to Tamassia's model, we treat it as if it was flow on the other edge connecting f_i and f_j that was part of the flow network of the original algorithm. With this we get a flow that is still valid and corresponds



(a) The spoon gadget **(b)** Cut through (v_c, v'_c) **(c)** 4 half-bends are needed

Figure 4.6: (a) Illustration of the spoon gadget. (b) The orthogonal input can be transformed into a slog drawing when everything above the dashed cut is moved up. (c) The result contains 4 half-bends.

to an orthogonal representation for which the algorithm of Tamassia [110] can compute a drawing. In the next step, we replace all orthogonal bends with pairs of half-bends. In Step 3 of Algorithm 1, we connect r-vertices with c-vertices by replacing the segment incident to the c-vertex of each rc-edge by a gadget, which we call *spoon* due to its shape (see Figure 4.6a). This gadget allows us to switch between orthogonal and diagonal ports on an edge. Note that the input slog representation specifies the ports on all vertices, thereby defining which configuration is used.

In order to fix the ports of cc-edges (which still use orthogonal ports), we employ appropriate cuts[1] (Step 4 of Algorithm 1). A *cut*, for us, is either (i) an x-monotone continuous curve that crosses only vertical segments and divides the current drawing into a top and a bottom part (*horizontal cut*), or, (ii) a y-monotone continuous curve that crosses only horizontal segments and divides the current drawing into a left and a right part (*vertical cut*). Observe that in order to apply a horizontal (vertical, resp.) cut, we have to ensure that each edge crossed by the cut has at least one vertical (horizontal, resp.) segment. This holds before the introduction of the spoons, as $\Gamma(G)$ is an orthogonal drawing. We claim that this also holds when all spoons are present. This is because a spoon replacing a horizontal (vertical, resp.) segment has two horizontal (vertical, resp.) segments.

To fix a horizontal cc-edge (v_c, v'_c) with v_c being to the left of v'_c in the drawing, we first momentarily remove this edge from the drawing. Then we use a horizontal cut which from left to right passes exclusively through

[1]A *cut* is a standard tool to perform stretchings in orthogonal drawings, see e.g. [53].

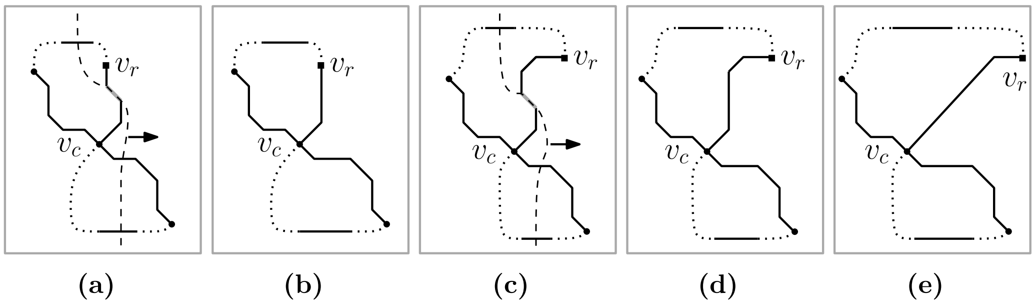
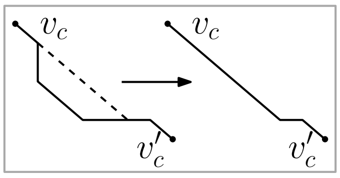


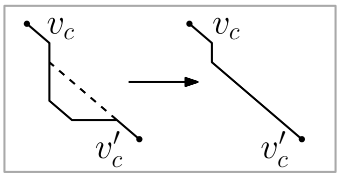
Figure 4.7: Saving bends on rc-edges: (a) A vertical cut through a bend-less rc-edge results in (b) a reduction by two half-bends. (c) Similarly, a vertical cut through a bent rc-edge also results in (d) a reduction by two half-bends. (d) However, the optimal may require four half-bends reduction.

vertical segments. There exist two options for such a cut. The first one starts in the outer face and continues up to the face below (v_c, v_c'), then to the face above and from there again to the outer face. The second one again starts in the outer face and continues up to the face above (v_c, v_c'), then to the face below and from there to the outer face (see Figure 4.6b). Our choice depends on the input slog representation that specifies the ports on each crossing vertex. The result of such a cut is depicted in Figure 4.6c and has a new horizontal and a new vertical segment that replaces edge (v_c, v_c'). The first (second, resp.) one is necessary for potential future vertical (horizontal, resp.) cuts. Similarly, we cope with cc-edges with bends by applying the same technique only to the first and last segments of the edge.

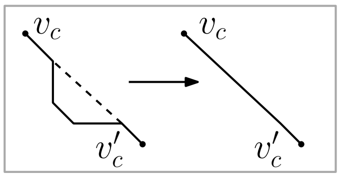
The resulting slog drawing has two additional half-bends for each rc-edge (the spoon gadget adds three half-bends; one is required) and four additional half-bends for each cc-edge (none is required), with respect to the half-bends suggested by the input representation. With similar cuts as the ones described above, we can save two half-bends for each rc-edge, by eliminating the diagonal segment of the spoon gadget (Step 5 of Algorithm 1). Our approach is illustrated in Figures 4.7a and 4.7b. Observe that in this case the cut simply requires the removal of the diagonal segment that is to be eliminated and not the whole edge. The result is optimal for bend-less rc-edges (see Figure 4.7b). However, for rc-edges with bends (see Figure 4.7c),



(a) Reduction by 2 half-bends



(b) Reduction by 2 half-bends



(c) Reduction by 4 half-bends

Figure 4.8: Saving bends on cc-edges by a local operation.

our approach guarantees two half-bends reduction (see Figure 4.7d), while in the optimal case four half-bends could be removed (see Figure 4.7e). Observe that the rectilinear segments of the edge are not affected, in order to be able to apply future cuts.

As already stated, each cc-edge admits four additional half-bends (none is required). It is always possible to remove two of them (Step 6 of Algorithm 1) by applying a local modification as depicted in Figure 4.8. If for example the horizontal part of such an edge is longer than the vertical one, a shortcut like the one in the left part of Figure 4.8a can be applied. Note that this operation does not require any cuts. If the horizontal and the vertical segments of the cc-edge have the same length, then all four half-bends can be saved; see Figure 4.8c.

Once the operations we described above are applied, the drawing will contain zero additional half-bends on rr-edges and bend-less rc-edges and at most two additional half-bends on each cc-edge and each rc-edge with bends, with respect to the input representation. Note that in order to apply our technique we need to scale up the initial drawing by a factor of five at the beginning of our algorithm, to provide enough space for additional half-bends. In subsequent steps, the cuts increase the drawing area. However, since each cut implies a constant factor increment to the drawing area and each edge yields at most one cut, the total drawing area asymptotically remains quadratic. The following theorem summarizes our approach.

Theorem 4.3.1. *Given a slog representation of a planarized graph G of maximum degree 4, we can efficiently compute a slog drawing requiring $O(n^2)$ area with (i) optimal number of half-bends on rr- and bend-less rc-edges and (ii) at most two additional half-bends on cc edges and rc-edges with bends.*

Note that the scaling of the drawing by a factor of five in Step 2 of Algorithm 1 does not asymptotically affect the drawing area; in practice, however, it has negative effects. This motivated us to heuristically further compact the drawing at the cost of some extra bends (as a post-processing; Step 7 of Algorithm 1). First, we enrich all diagonal segments that are of a certain length by a new horizontal and a new vertical segment, so that the remaining diagonal segment is of unit length. To ensure planarity, we apply a rectangular decomposition similar to the one of Tamassia [110] and then we contract along horizontal and vertical cuts. Finally, we remove unnecessary half-bends similarly to Step 6 of Algorithm 1 (see also Figure 4.8).

4.4 Computing a Slog Representation Using ILP

One way to compute a bend-optimal slog representation is the modified flow network presented in Section 4.2.1. We will now present an alternative that is inspired by the LP formulation for the Kandinsky model of Eiglsperger et al. [43].

For the ILP we require integer variables l_e and r_e for all edges $e \in E$, that will hold the number of left (l_e) and right (r_e) half-bends on edge e and have values $\in \{0, \dots, \infty\}$. Furthermore we need variables $a_{(u,v)}$ for each edge that will hold the angle that edge (u, v) forms at vertex u with its cyclic predecessor on u. We count the angle at vertices in 90° steps, which means that $a_{(u,v)} = 1$ stands for a 90° angle at u, $a_{(u,v)} = 2$ stands for a 180° angle and so on. Clearly $a \in \{1, \dots, 4\}$. Finally we need integer variables z_e for all edges $e \in E$, that model the parity of the number of half-bends on an edge. We restrict the values of the z variables to be greater or equal to zero. In Linear Program 1 an overview of the lp is given.

$$
\begin{array}{llll}
\textbf{min} & \sum_{e\in E} l_e + r_e & & \\
\textbf{s.t.} & \sum_{(v,u)\in\mathcal{E}(v)} a_{(v,u)} = 4 & \forall v \in V & (1) \\
& \sum_{(u,v)\in f} (2a_{(u,v)} - r_{(u,v)} + l_{(u,v)}) = & & \\
& \quad \begin{cases} 4k-8 & f \text{ is inner face} \\ 4k+8 & f \text{ is outer face} \end{cases} & \forall f \in F & (2) \\
& r_{(u,v)} = l_{(v,u)} & \forall (u,v) \in E & (3) \\
& r_{(u,v)} + l_{(u,v)} - 2z_{(u,v)} = & & \\
& \quad \begin{cases} 0 & (u,v) \text{ is rr- or cc-edge} \\ 1 & (u,v) \text{ is rc-edge} \end{cases} & \forall (u,v) \in E & (4)
\end{array}
$$

Linear Program 1: The ILP to compute bend-optimal slog representations

With the minimization of the sum of half-bends on all edges in the objective function of Linear Program 1 we make sure that the representation we get from the lp has the optimal number of half-bends. Constraint 1 of Linear Program 1 ensures that the sum of angles around a vertex is 360°. With constraint 2 of Linear Program 1 we force the sum of all angles in a face f to equal $180° \cdot (p(f) - 2)$, where $p(f)$ is the number of angles in f (see also Section 2.4). The coefficients of this constraint stem from the fact that a half-bend adds a 45° angle, but the angles at vertices are measured in 90° steps. For consistency the next constraint 3 forces that the number of half-bends to the left on an edge equals the number of half-bends to the right on the reverse edge. Finally constraint 4 models the parity of the number of half-bends on edges. It ensures that on rr- and cc-edges there is an even number of half-bends and on rc-edges an odd number.

4.5 Realizing the Representation

In this section, we develop a linear program which, given an optimal slog representation S of a plane graph G, computes an actual drawing $\Gamma(G)$, which is optimal with respect to the total number of bends; if one exists. Before we proceed with the description of our linear program, we mention that despite the fact that every experiment we made on random and crafted graphs led to a feasible solution, we could not prove the feasibility of the linear program. However, we strongly believe that there always exists a slog drawing realizing the given representation.

4.5.1 The Core of the Linear Program

Initially, we appropriately augment graph G and obtain a new graph that is a subdivision of G and has at most one half-bend on each edge. More precisely, let (u, v) be an edge of G with more than two half-bends (as defined by the slog representation S). Let $\langle b_1, b_2, \ldots, b_k\rangle$, $k \geq 2$, be the half-bends of edge (u, v) and assume without loss of generality that $b_1, b_2, \ldots, b_k$ appear in this order along the edge (u, v), when traversing (u, v) from vertex u towards vertex v. We first consider the case where vertex u is a real vertex. In this case, we add a new crossing vertex w in G and then we replace the edge (u, v) of G with the edges (u, w) and (w, v). The first half-bend b_1 of the edge (u, v) is assigned to the edge (u, w), while the remaining half-bends $\langle b_2, \ldots, b_k\rangle$ of the edge (u, v) are assigned to the edge (w, v). The case where vertex u is a crossing vertex is treated analogously, with the only exception that in this particular case vertex w would have been a real vertex. If we apply the procedure that we just described on each edge of G with more than two half-bends (as long as there exist such edges), then we will obtain an augmented graph, say G_{aug}, that is clearly a subdivision of G and has at most one half-bend on each edge, as desired. Furthermore, neither the type of each new vertex nor its ports are arbitrarily chosen, as they depend on the types of its incident segments given by the input representation S (either orthogonal or diagonal). This implies a new slog representation, say S_{aug}, for G_{aug}.

Now observe that each face f of G has a corresponding face f' in G_{aug} such that: (i) the vertices of G_{aug} incident to face f' are the same as the ones incident to face f of G, plus the ones from the subdivision; and (ii) the sequence of slopes assigned to the segments bounding f' is the same as the ones of the segments bounding f in G. Hence, a drawing $\Gamma(G_{aug})$ of G_{aug} realizing the slog representation S_{aug} is also a drawing $\Gamma(G)$ of G

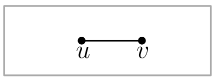
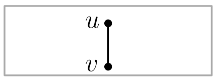
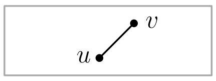
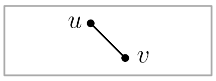
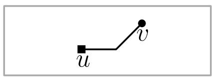
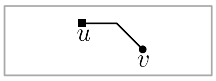
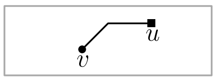
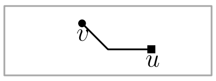
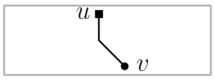
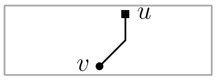
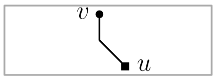
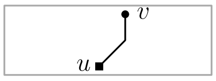
(a) rr-edges		(b) cc-edges	
 $y_u = y_v$ $x_v - x_u \geq 1$	 $x_u = x_v$ $y_v - y_u \geq 1$	 $y_u - y_v = x_v - x_u$ $y_u - y_v \geq 1$	 $y_v - y_u = x_v - x_u$ $y_v - y_u \geq 1$
(c) rc-edges			
 $x_v - x_u \geq y_u - y_v + 1$ $y_u \geq y_v + 1$	 $x_v - x_u \geq y_v - y_u + 1$ $y_v \geq y_u + 1$	 $x_u - x_v \geq y_v - y_u + 1$ $y_v \geq y_u + 1$	 $x_u - x_v \geq y_u - y_v + 1$ $y_u \geq y_v + 1$
 $y_v - y_u \geq x_v - x_u + 1$ $x_v \geq x_u + 1$	 $y_v - y_u \geq x_u - x_v + 1$ $y_v \geq y_u + 1$	 $y_u - y_v \geq x_u - x_v + 1$ $x_u \geq x_v + 1$	 $y_u - y_v \geq x_v - x_u + 1$ $x_v \geq x_u + 1$

Figure 4.9: The list of constraints used by the linear program for (a) rr-edges, (b) cc-edges and (c) rc-edges, assuming that the y-axis points downwards.

realizing the slog representation S, where subdivided edges are routed as their corresponding paths in G_{aug}.

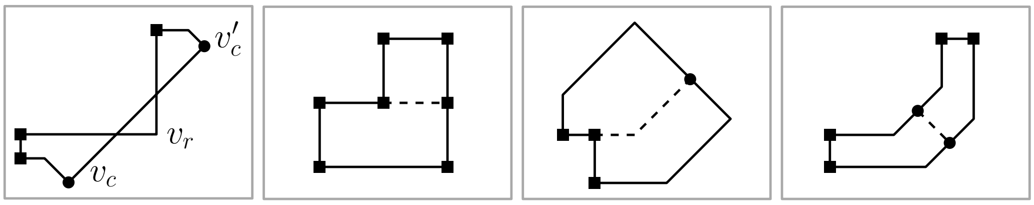
We are now ready to describe our linear program, which computes a drawing $\Gamma(G_{aug})$ of G_{aug} realizing the slog representation S_{aug}. For each vertex u of G_{aug}, we introduce a pair of variables x_u and y_u that corresponds to the coordinates of vertex u on the plane. Then, for each edge (u, v) of G_{aug}, we define a pair of constraints, depending on the type of vertices u and v (i.e., real or crossing vertices). The detailed list of constraints is given in Figure 4.9.

In order to obtain "compact" drawings, we indirectly minimize the area by minimizing the total edge length. In particular, this is our objective function. Note that the slopes of the segments allow us to express the Euclidean length of each edge as a linear function. As an example, the length of the edge depicted in the first cell of Figure 4.9c is defined as $(\sqrt{2}-1)\cdot(y_u-y_v)+x_v-x_u$.

4.5.2 Addressing Planarity Issues

The linear program, as described so far, models the shape of the edges (and subsequently the shape of the faces) and the relative positions between pairs of adjacent vertices. Since there are no constraints among non-adjacent vertices, it is highly possible that the resulting drawing is non-planar. We provide an example in Figure 4.10a, where the relative positions between vertices (*i*) v_r and v_c, and, (*ii*) v_r and v'_c are not defined by the liner program, yielding to a (potential) crossing situation. To cope with this problem, unfortunately, we cannot follow an approach similar to the one that Tamassia suggests in his original algorithm (i.e., he "splits" all non-rectangular faces into rectangular ones), since in our case a face is not necessarily rectilinear.

In order to describe our approach to ensure that each face is drawn planar, we first introduce some necessary terminology. We distinguish two types of corners of a face in a slog representation; *vertex-corners* (or simply vertices) and *bend-corners* (or simply bends). With respect to a face, a corner is either *convex*, if the inner angle is $\leq 135°$, or *non-convex* otherwise. We ignore vertices and bends on corners that form $180°$ angles, since by construction they are always aligned with their neighbors. Hence, there are four possible types of corners in total: convex vertex-corner, convex bend-corner, non-convex vertex-corner and non-convex bend-corner. The *configuration* of a corner describes the shape of the corner by the pair of orientations of its two incident segments in the order they are visited by a counterclockwise traversal of the corresponding face. Possible *orientations* are horizontal (h), vertical (v), diagonal-up (du), and diagonal-down (dd). For example, the configuration of the bend-corner incident to segments s' and s'' of Figure 4.11a is given by *dd-h*. The *type* of a configuration describes the corresponding corner in a more general way by just distinguishing between orthogonal (o) or diagonal (d) orientations. In the example of Figure 4.11a, the configuration of the bend-corner incident to segments s' and s'' is of type *d-o*. We next define the notions of a *split-edge* (Definition 4) and an *almost-convex face* (Definition 5), that are both central in our approach.



(a) A non-planar face **(b)** Split-edge (vertex) **(c)** Split-edge (vertex) with half-bend **(d)** Split-edge (bend)

Figure 4.10: In all figures, real (crossing, resp.) vertices are drawn as squares (disks, resp.); split-edges are drawn dashed.

Definition 4 (split-edge)**.** *For a given face f, a* split-edge *is an edge that:*

- *is bend-less and connects a non-convex vertex-corner v with a new vertex that we introduce by subdividing a side parallel to one of the edges incident to v (see Figure 4.10b).*
- *or, has a half-bend and connects a non-convex vertex-corner v with a new vertex that we introduce by subdividing a diagonal side of f (see Figure 4.10c).*
- *or, is a bend-less edge that connects two new vertices that we introduce by subdividing two parallel edges, when one of them is incident to a non-convex bend-corner (see Figure 4.10d).*

Definition 5 (almost-convex)**.** *A face is* almost-convex *if it does not contain any non-convex vertex-corners and no split-edge exists that separates the face into two non-convex faces.*

First, we make all faces almost-convex (by further augmenting our graph). Later, we will show that the linear program will always compute a planar drawing if all faces are almost-convex.

A non-convex vertex-corner is *eliminated* by introducing a new split-edge (corresponding to new constraints in the linear program) as shown in Figure 4.10b. When there is no parallel side to one of the segments incident to the vertex-corner, we introduce a split-edge with a half-bend, as illustrated in Figure 4.10c. It is important to note that the elimination of a non-convex vertex-corner does not introduce new ones. Hence, all of them can be eliminated sequentially by appropriately adopting one of the two approaches described above.

In order to eliminate a non-convex bend-corner of a face that is not almost-convex, we search for a split-edge (again corresponding to new constraints in the linear program) that yields two non-convex faces. Such a split-edge is illustrated in Figure 4.10d. We will appropriately introduce such split-edges until all faces are almost-convex (without introducing non-convex vertex-corners). To prove that it is always feasible to make all faces almost-convex, we give the following lemma.

Lemma 4.5.1. *Let s' and s'' be two segments of a face f incident to a non-convex bend-corner. Face f contains a segment $s \notin \{s', s''\}$ that is parallel to either s' or s''.*

Proof. For a proof by contradiction, we assume that there is no segment of face f parallel to s' and s''. Without loss of generality, we further assume that s' is a horizontal segment and s'' is a diagonal segment of positive slope; see Figure 4.11a. The cases, where s' is a vertical segment and/or s'' is a diagonal segment of negative slope, are analogous (see Figures 4.11b, 4.11c, and 4.11d). Let $p_{s'}$ and $p_{s''}$ be the end-points of segments s' and s'', respectively, which are not identified with the non-convex bend-corner incident to both s' and s''. Since f is a face, there exists a polygonal chain of segments of f connecting $p_{s'}$ and $p_{s''}$. In our drawing model, such a chain consists of horizontal, vertical and diagonal segments. Now observe that a horizontal or a positively-sloped diagonal segment of the chain connecting $p_{s'}$ and $p_{s''}$ is parallel to s' or s'', respectively, which contradicts our initial assumption that there is no segment of face f parallel to s' and s''. Hence, the polygonal chain connecting $p_{s'}$ and $p_{s''}$ consists of vertical and negatively-sloped diagonal segments, which is a contradiction since $p_{s'}$ and $p_{s''}$ cannot be connected by such a chain, without forming an angle of 45° at a corner of f (a situation that is not allowed by our drawing model). □

From Lemma 4.5.1, it follows that, for a non-convex bend-corner of a face f, there is a split-edge emanating from one of its incident segments towards a parallel segment of face f. If f is not almost-convex (and contains no

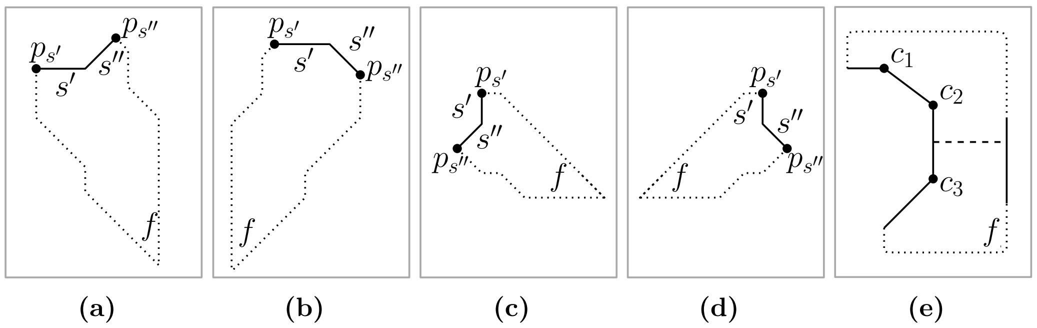


Figure 4.11: (a)-(d) Different configurations used in the proof of Lemma 4.5.1. (e) Configuration used in the proof of Lemma 4.5.2.

convex vertex-corners) and this edge is carefully selected such that it yields exactly two non-convex "subfaces", say f' and f'', of face f, then it is not difficult to see that both f' and f'' have fewer non-convex bend-corners than f. In addition, no convex vertex-corners are introduced. This implies that if one recursively applies this procedure to f' and/or f'' (if either of these is not almost-convex), f will eventually be split into a particular number of "subfaces" that are all almost-convex. In addition, it is not difficult to see that all additional edges, that are required to make all faces almost-convex can be expressed by using the original set of constraints of our linear program. So, it now remains to prove that almost-convex faces are drawn planar. To do so, we give the following lemmas.

Lemma 4.5.2. *An almost-convex face f has at most two consecutive non-convex bend-corners.*

Proof. Assume to the contrary that f has three consecutive non-convex bend-corners, say c_1, c_2 and c_3; see Figure 4.11e. Assume that c_1, c_2 and c_3 appear in this order in the counterclockwise traversal of face f. By Lemma 4.5.1, there exists a segment of f that is parallel to one of the segments incident to c_2. This implies that there exists a split-edge that partitions f into two non-convex faces; one containing c_1 and one containing c_3, which is a contradiction since f is almost-convex. □

Lemma 4.5.3. *An almost-convex face has at most two non-convex bend-corners.*

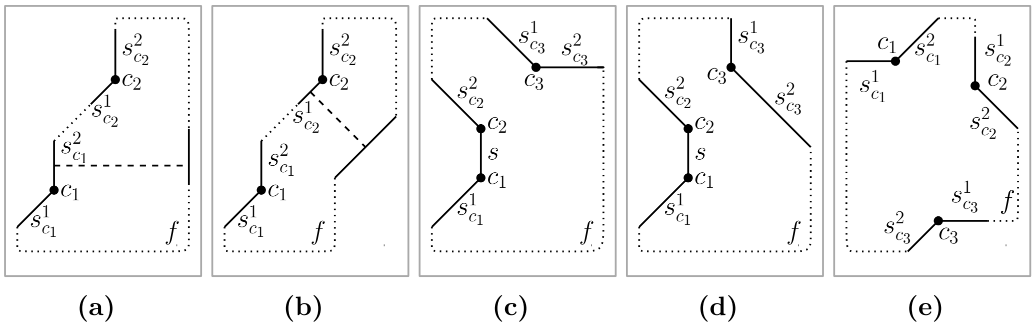


Figure 4.12: Different configurations used in the proof of Lemma 4.5.3.

Proof. In the proof we use the notion of a configuration. More precisely, we assume to the contrary that an almost-convex face f contains at least three non-convex bend-corners c_1, c_2 and c_3 and distinguish four cases. In our case analysis, we denote by $s^1_{c_i}$ and $s^2_{c_i}$ the segments incident to corner c_i and assume that $s^1_{c_i}$ precedes $s^2_{c_i}$ in the clockwise traversal of face f, $i = 1, 2, 3$.

Case 1: *Two of these non-convex bend-corners have the same configuration*; see Figure 4.12a or 4.12b for an illustration. By Lemma 4.5.1, there exists a parallel segment to either $s^1_{c_1}$ or $s^2_{c_1}$, and thereby to either $s^1_{c_2}$ or $s^2_{c_2}$. In both cases, one of the split-edges separates c_1 from c_2, so that the resulting faces are both non-convex. Hence, f is not almost-convex; a contradiction. So, in the following cases we assume that c_1, c_2 and c_3 are of different configurations.

Case 2: *Corners c_1 and c_2 are consecutive corners of f and the first segment of c_3 is parallel to the second segment of c_2*; see Figure 4.12c for an illustration. We denote by s the segment that is incident to both c_1 and c_2 (i.e., $s = s^2_{c_1} = s^1_{c_2}$). From the previous case it follows that c_1 and c_3 are of different configurations. In order to close the face there has to be a segment that is parallel to either s or $s^2_{c_2}$ that is not $s^1_{c_3}$, thereby allowing a split-edge that separates either c_1 from c_2 and c_3, or, c_1 and c_2 from c_3. The resulting faces are both non-convex. Hence, f is not almost-convex; a contradiction.

Case 3: *Corners c_1 and c_2 are consecutive and the second segment of c_3 is parallel to the second segment of c_2*; see Figure 4.12d for an illustration.

Again, we denote by s the segment that is incident to both c_1 and c_2 (i.e., $s = s^2_{c_1} = s^1_{c_2}$) and assume that c_1 and c_3 are of opposite configurations. In this case there is a split-edge between segments $s^2_{c_2}$ and $s^2_{c_3}$ thereby separating c_1 and c_2 from c_3 and resulting in two non-convex faces. Hence, f is not almost-convex; a contradiction.

Case 4: *Corners c_1, c_2 and c_3 are pairwise non-consecutive*; see Figure 4.12e for an illustration. Since there are only two types of diagonals, at least two non-convex corners, say c_1 and c_3, are of the same type. Since they are forced to have opposite configurations (*d-o* or *o-d*) a split-edge between those two parallel diagonals would separate the two respective corners, resulting in two non-convex faces. Hence, f is not almost-convex; a contradiction.

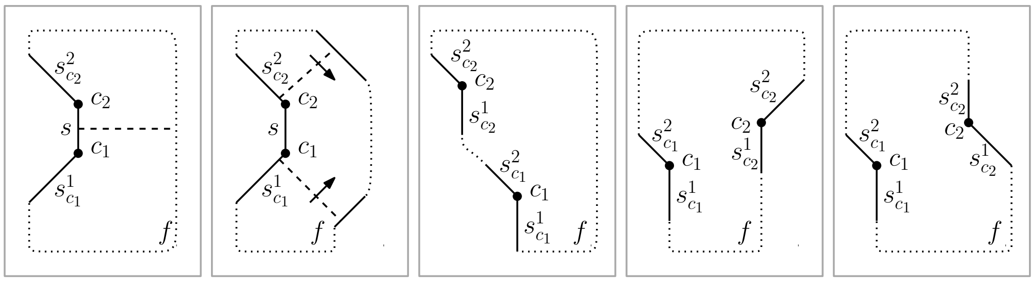
The proof is completed by the observation that one of these four cases will always apply to every almost-convex face with more than two non-convex bend-corners. □

Lemma 4.5.4. *An almost-convex face is always drawn planar.*

Proof. Let f be an almost-convex face. By Lemma 4.5.3, face f has at most two non-convex bend corners.

We claim that in the case where f has exactly one non-convex bend-corner, f is drawn planar. In fact, since completely convex faces are drawn planar by construction, we know that, if there is exactly one non-convex bend-corner c and f is not drawn planar, then one of the two segments s^1_c and s^2_c incident to c must be involved in a crossing. All the other vertex- and bend-corners of f are convex or collinear by construction. Let s be the segment that crosses s^1_c or s^2_c. Since s is only adjacent to convex corners or 180° corners, it is not possible to close f without violating the port assignments as given by the representation. So, our claim holds.

Consider now the more interesting case where face f has exactly two non-convex bend-corners, say c_1 and c_2. We denote by $s^1_{c_i}$ and $s^2_{c_i}$ the segments incident to corner c_i and assume that $s^1_{c_i}$ precedes $s^2_{c_i}$ in the clockwise traversal of face f, $i = 1, 2$. We distinguish the following cases:



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

Figure 4.13: Different configurations used in the proof of Lemma 4.5.4.

Case 1: *Corners c_1 and c_2 are consecutive*; see Figure 4.13a for an illustration. In this case, there is a segment, say s, that is incident to both c_1 and c_2 (i.e., $s = s_{c_1}^2 = s_{c_2}^1$). If there is a segment of f parallel to s, then there exists a split-edge separating f into two non-convex subfaces; one containing c_1 and one containing c_2 (see Figure 4.13a). Hence, f is not almost-convex. It follows that there is no segment of f that is parallel to s. By Lemma 4.5.1, there exist parallel segments to the other two segments that are incident to c_1 and c_2 (see Figure 4.13b). However, since f is almost-convex, a “split-edge” connecting the respective parallel segments would result in at least one convex face. We can move these “split-edges” arbitrary close to c_1 and c_2, so that they separate f into three convex regions. Since convex regions are drawn convex and hence planar by definition, no crossing can occur.

Case 2: *Corners c_1 and c_2 have the same configuration and orientation*; see Figure 4.13c for an illustration. This particular case is identical to Case 1 of Lemma 4.5.3 and therefore cannot occur.

Case 3: *Corners c_1 and c_2 have opposite configuration (meaning that they are made of the same orthogonal and diagonal part but in different orders) and orientation*; see Figure 4.13d for an illustration. Since the number of crossings that occur has to be even (otherwise ports would be violated), and the only way to have two crossings requires that one of the convex regions is drawn non-convex, this situation cannot introduce any crossings.

Case 4: *Corners c_1 and c_2 have the same configuration but opposite orientations*; see Figure 4.13e for an illustration. In this case, it is not difficult to see that there exists a split-edge between the two orthogonal or the two diagonal segments incident to c_1 and c_2, separating them into two non-convex subfaces, so f cannot be almost-convex.

The proof is completed by the observation that one of these four cases will always apply to an almost-convex face with exactly two non-convex bend-corners. □

4.6 Area Bounds

Slog drawings have aesthetic appeal and seem to improve the readability of non-planar graphs, when compared to traditional orthogonal drawings. However, in this section we show that such drawings may require increased drawing area. Note that most of the known orthogonal drawing algorithms require $O(n) \times O(n)$ area. The situation is different if one insists on slog drawings of optimal number of bends. As the following theorem asserts, the area penalty can be exponential.

Theorem 4.6.1. *There exists a graph G whose slanted orthogonal drawing $\Gamma(G)$ of minimum number of bends requires exponential area, assuming that a planarized version $\sigma(G)$ of graph G is given.*

Proof. The planarized version $\sigma(G)$ of G is given in Figure 4.14a and consists of $n+1$ layers $L_0, L_1, \ldots, L_n$. Layer L_0 is the square grid graph on 9 vertices. Each layer L_i, $i = 1, 2, \ldots, n$, is a cycle on 20 vertices with 4 internal chords. Consecutive layers L_{i-1} and L_i, $i = 1, 2, \ldots, n$, are connected by 8 edges which together with the chords of layer L_i define 12 crossings. Hence, G consists of $20n + 9$ vertices and $32n + 12$ edges that define $12n$ crossings.

A slog drawing $\Gamma(G)$ of G with minimum number of bends derived from $\sigma(G)$ ideally introduces (a) no bends on crossing-free edges of $\sigma(G)$, and, (b) two half-bends in total for each rc-edge. Now observe that at each layer there exist four vertices, that have two ports pointing to the next layer (gray-colored in Figure 4.14a). This together with requirements (a) and (b) sug-

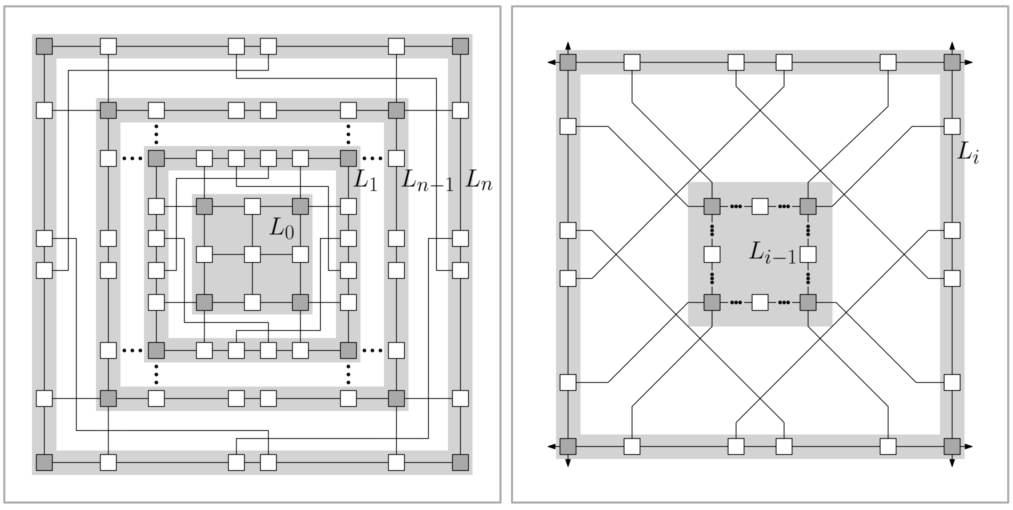


Figure 4.14: (a) A planarized version $\sigma(G)$ of a graph G. (b) Edges involved in crossings in $\sigma(G)$ contribute two half-bends.

gests that the vertices of each layer L_i should reside along the edges of a rectangle, say R_i, such that the vertices of L_i whose ports point to the next layer coincide with the corners of R_i, $i = 0, 1, 2, \ldots, n$ (with the only exception of the "innermost" vertex of L_0; in Figure 4.14b, R_i is identified with cycle L_i). Hence, the routing of the edges that connect consecutive layers should be done as illustrated in Figure 4.14b. Since L_0 is always drawable in a 3×3 box meeting all requirements mentioned above, and, $\sigma(G)$ is highly symmetric, we can assume that each R_i is a square of side length w_i, $i = 0, 1, 2, \ldots, n$. Then, it is not difficult to see that $w_0 = 3$ and $w_{i+1} = 2w_i + 8$, $i = 1, 2, \ldots, n$. This implies that the area of $\Gamma(G)$ is exponential in the number of layers of G and therefore exponential in the number of vertices of G (recall that G has $n + 1$ layers and $20n + 9$ vertices). □

4.7 Experimental Evaluation

In this section, we present an experimental evaluation of the slog model. We compare classic orthogonal drawings obtained with the implementation of the original Tamassia algorithm [110] provided by the yFiles library

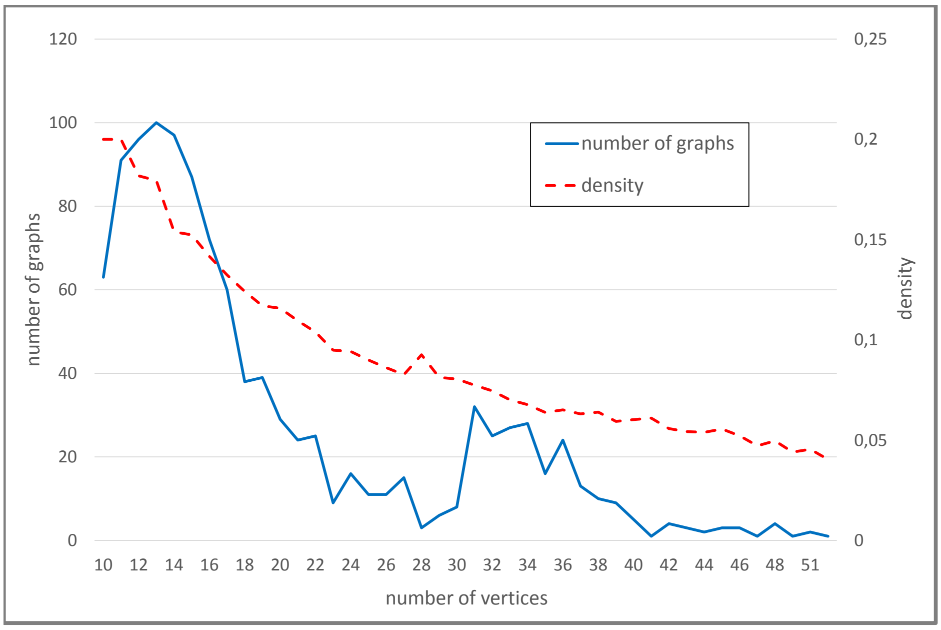


Figure 4.15: Number of instances and density of the test set.

(http://www.yworks.com) with bend-optimal slog drawings and drawings computed by the heuristic presented in Section 4.3. As a test set, we used the Rome graphs (obtained from http://www.graphdrawing.org) which are approximately 11.500 graphs. We filtered them for connected graphs with maximal degree 4, which left 1.122 graphs. Of this 1.122 graphs, 1.039 were planar and 83 non-planar. The average density over all graphs was $0, 14$; recall that the density of a graph is defined as the ratio of the number of its edges to the maximum possible number of edges. The number of vertices ranged from 10 to 56. Figure 4.15 gives a more detailed description of the test set: the number of instances and their average density are plotted against the number of vertices of the test set.

We ran our experiments on a Linux machine with four cores at $2, 5$ GHz and 3 GB of RAM. All implementations were done in Java using the yFiles library. For solving the linear programs, we used the Gurobi solver [64].

To obtain an input for our algorithms, we computed an embedding with the *Combinatorial Embedder* from the yFiles library, which guarantees a planar embedding for planar instances. In all following plots, the curve denoted by *orthogonal* stands for results for the orthogonal drawings, while the curves

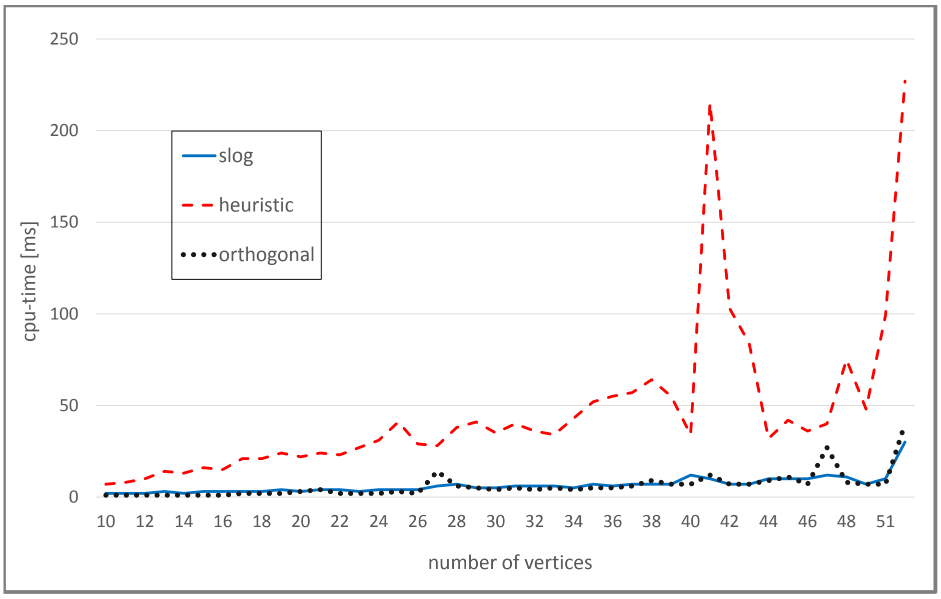


Figure 4.16: Cpu-time against number of vertices.

denoted by *slog* and *heuristic* correspond to the results for bend-optimal and heuristic slog-drawings. To obtain the actual numbers, the results for all graphs with the same number of vertices were averaged.

All results we present in this section were computed in less than 200 ms each, as depicted in the cpu-time chart in Figure 4.16. Apparently, the heuristic requires the most computation time. We observed that this is due to its last step, where the drawing is heuristically compacted. It seems that the computation of the cuts, which are required in order to reduce the area, is relatively time-consuming.

Of course, the graphs of our test set are rather small and not very dense. However, even for hand-crafted dense graphs with more than 400 vertices, the optimal slog drawings could be computed in less than 2 seconds, which suggests that our LP-formulation can be useful for practical applications. Note that these hand-crafted graphs are not included in the experimental evaluation of this section; we simply used them in order to verify that the LP can still be solved within reasonable time even for large and dense input graphs.

In Figure 4.17, the required area is plotted against the number of vertices.

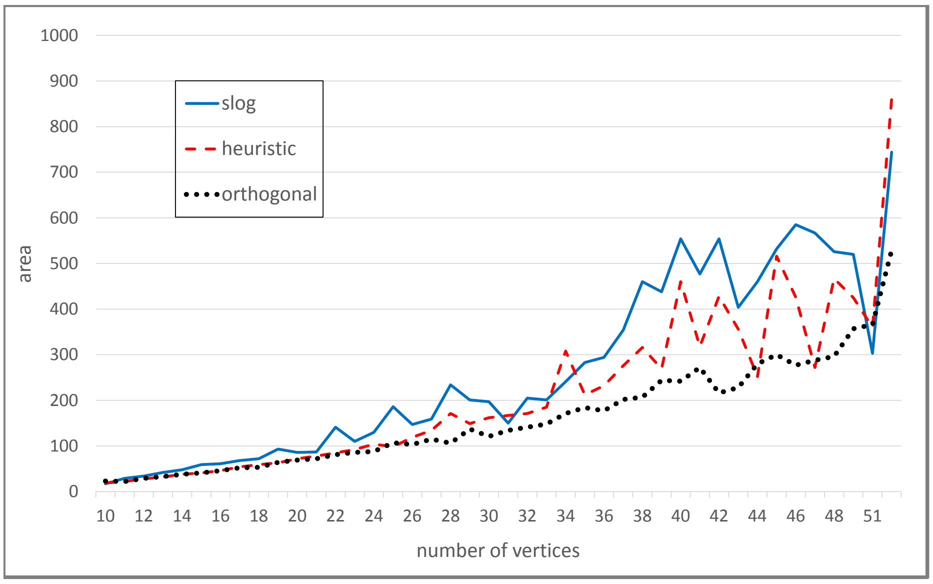


Figure 4.17: Area against number of vertices.

As expected, the area of the slog drawings is larger than the corresponding one of the orthogonal drawings. On the other hand, the bend-optimal slog drawings and the ones computed by the heuristic presented in Section 4.3 are of comparable area, indicating that the minimization of the total edge length as an objective function seems to be very effective. Recall that the orthogonal drawing which is used as an input in the heuristic is scaled up by a factor of five (which yields a factor of 25 in the total area). So, one would expect that the drawings computed by the heuristic would (in practice) require significantly more area than the corresponding orthogonal ones, which apparently is not evident in Figure 4.17. This is due to the last step of the heuristic, where the drawing area is reduced.

As stated in Section 4.2.2, the number of half-bends in the bend-optimal slog drawings is at least twice the number of bends in the bend-optimal orthogonal drawings. So, in Figure 4.18 we plotted twice the number of orthogonal bends against the number of half-bends produced by our algorithms. As expected, the orthogonal drawings require the least amount of bends. We measured that on average the bend-optimal slog drawings required 2.84 times more half-bends than the orthogonal drawings, while the drawings computed by the heuristic required 1.18 times more half-bends than the bend-optimal

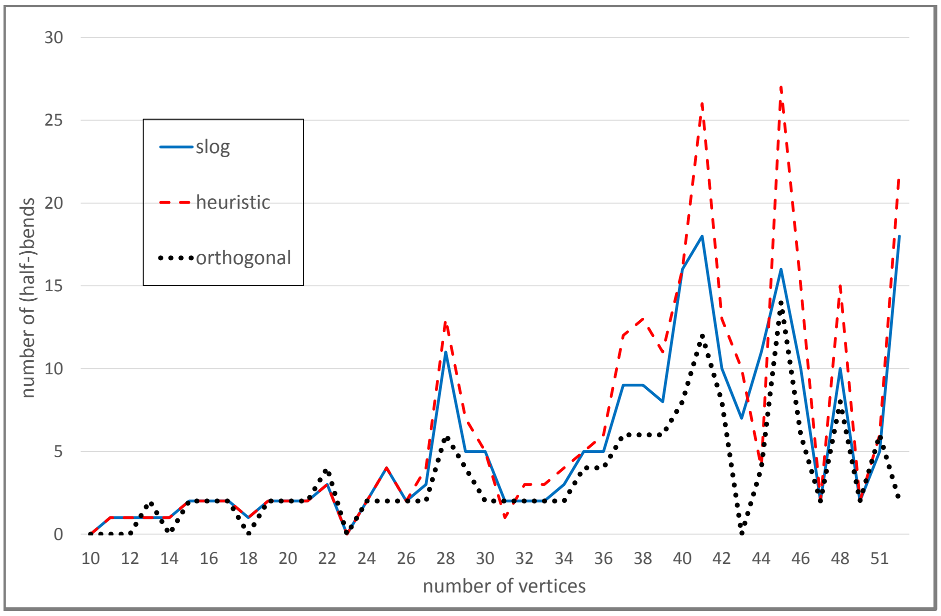


Figure 4.18: Number of bends against number of vertices.

slog drawings. In actual numbers, the bend-optimal slog drawings require (on average) 5 more half-bends, while the drawings computed by the heuristic require 8 more half-bends than the corresponding orthogonal drawings.

Figure 4.19 shows the total edge length in relation to the number of vertices. In our experiments, we found that the plots of the total edge length are comparable to the plots of the area (Figure 4.17). This is exactly as expected, since the larger the area is the larger the total edge length is expected to be. When comparing the ratio of the longest to the shortest edge, again the orthogonal algorithm produced the smallest results, as can be seen in Figure 4.20. This is because the orthogonal drawings were the most compact ones. For the bend-optimal slog drawings, this ratio went up to 37 in our experiments, while the heuristic had a better ratio between the longest and the shortest edge. Note that the high ratios observed in the bend-optimal slog drawings are caused by the long diagonal segments required in the slanted model.

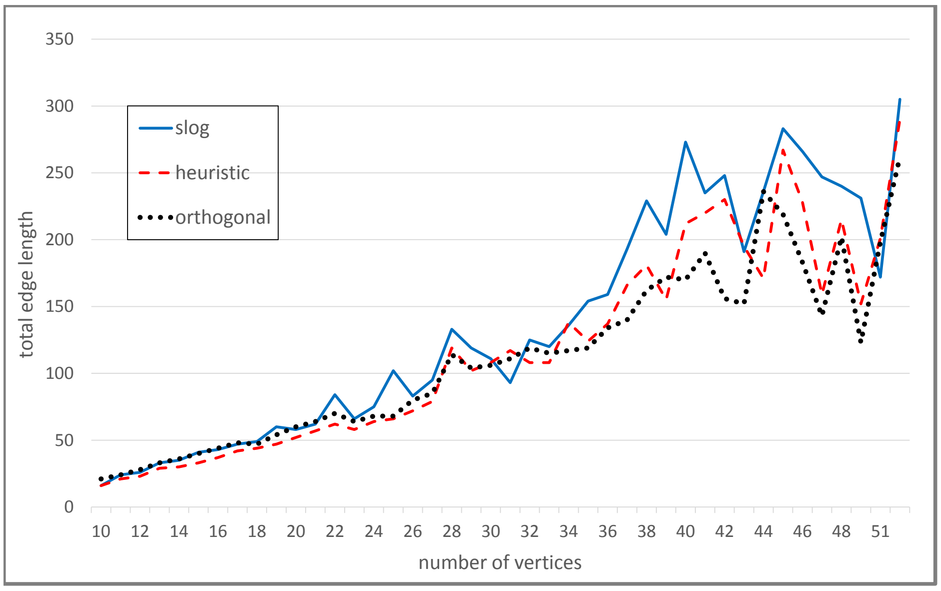


Figure 4.19: Total edge length against number of vertices.

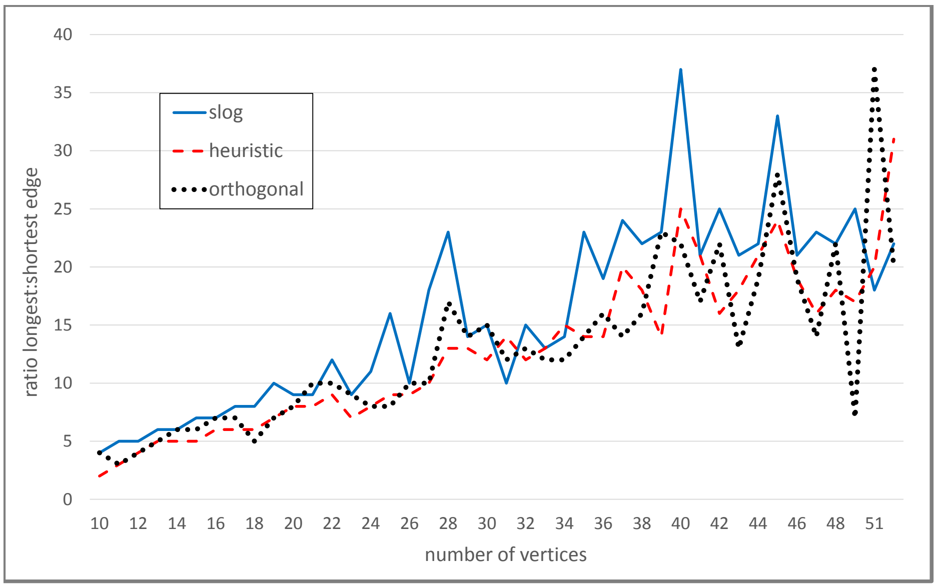


Figure 4.20: Ratio of longest to shortest edge against number of vertices.

4.8 Sample Drawings

In this section we present several sample drawings of graphs in the orthogonal and slog model.

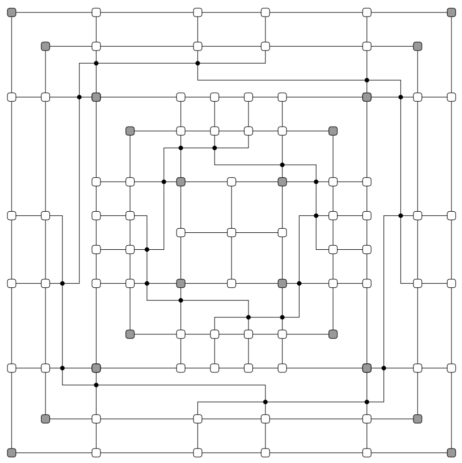


Figure 4.21: An orthogonal drawing of minimum number of bends for the graph of Figure 4.14 establishing the exponential area bound for slog drawings.

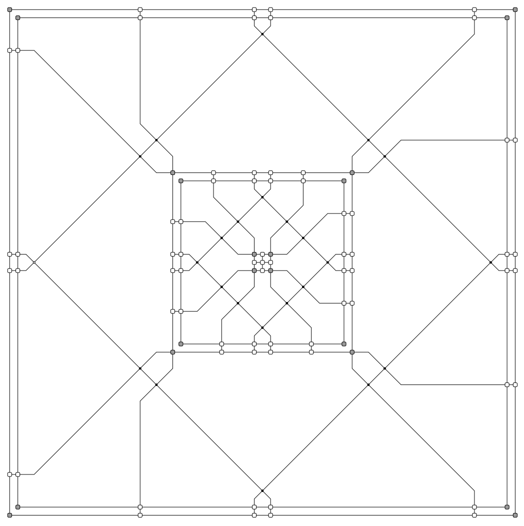


Figure 4.22: The bend-optimal slog drawing corresponding to the one of Figure 4.21.

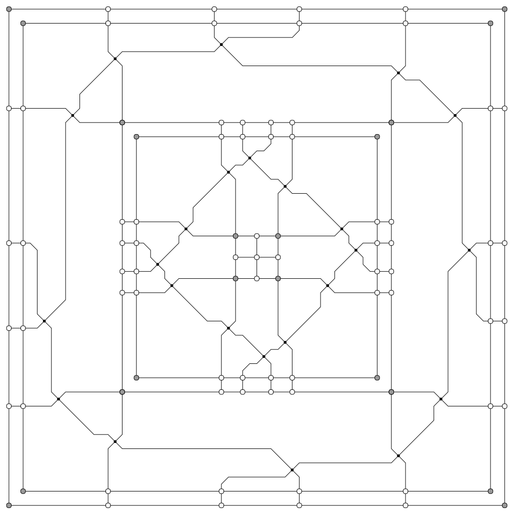


Figure 4.23: The close-to-optimal slog drawing corresponding (to the one of Figure 4.21) produced by our heuristic algorithm of Section 4.3 without Step 7 of Algorithm 1.

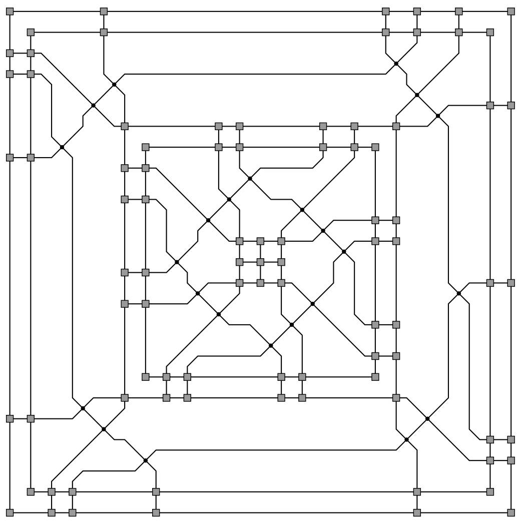


Figure 4.24: The close-to-optimal slog drawing corresponding (to the one of Figure 4.21) produced by our heuristic algorithm of Section 4.3 with Step 7 of Algorithm 1.

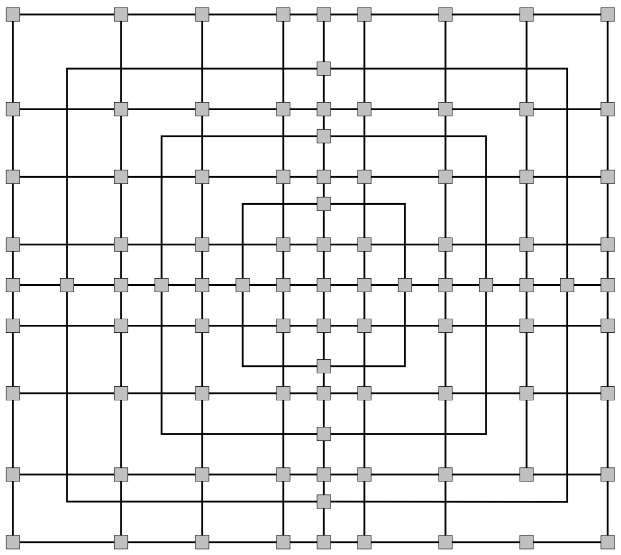


Figure 4.25: A highly symmetric non-planar orthogonal drawing.

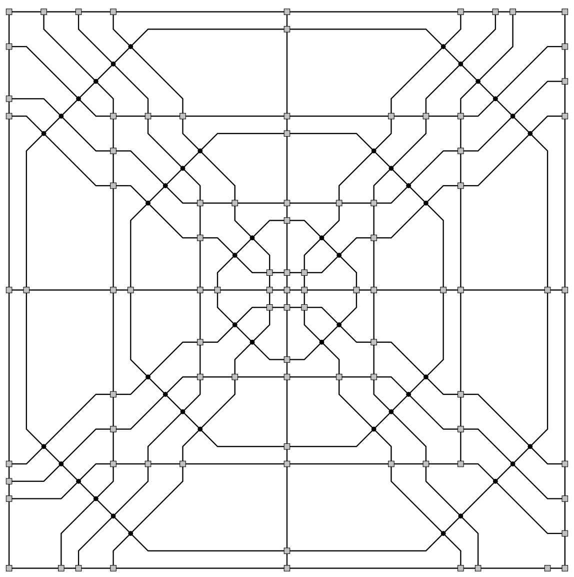


Figure 4.26: The bend-optimal slog drawing corresponding to the one of Figure 4.25.

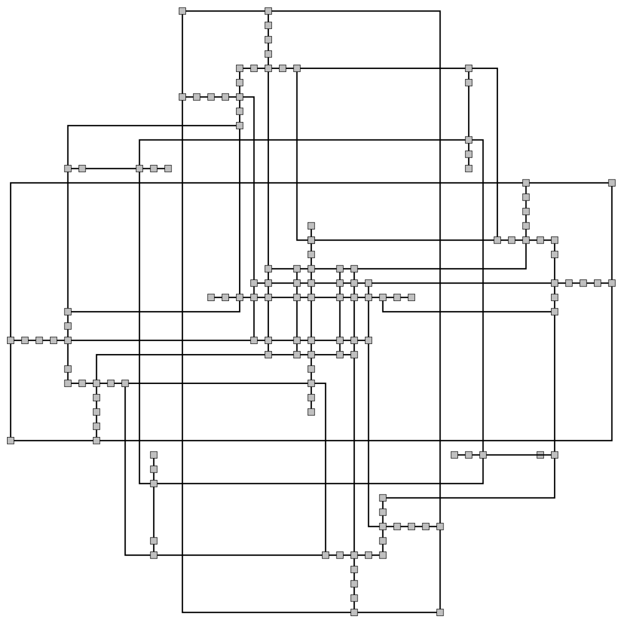


Figure 4.27: A non-planar orthogonal drawing

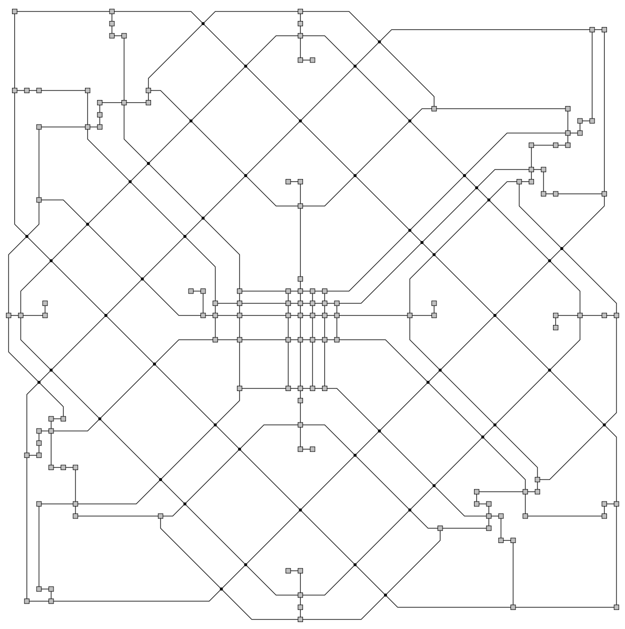


Figure 4.28: The bend-optimal slog drawing corresponding to the one of Figure 4.27.

4.9 Summary

In this chapter we introduced a new drawing model for graphs with maximum vertex degree four that we call slanted orthogonal or slog for short. In the slog model crossings are only allowed between diagonal segments and the minimal angle between consecutive edge-segments has to be 135°, resulting in a shape we called half-bend. We showed that the number of half-bends of a bend-optimal slog representation is at least twice the number of bends of the corresponding bend-optimal orthogonal representation. We gave a network flow formulation to compute a bend-optimal slog representation and showed how to realize it using linear programming. Unfortunately, bend-optimal slog drawings can require exponential area. That is why we also developed a heuristic that computes a slog drawing with at most two times the number of half-bends of the optimal drawing but requires only polynomial area. In an experimental evaluation we found that slog drawings can be computed efficiently and require slightly more bends and area than orthogonal drawings. The advantages of the slog model however are that it becomes much easier to identify crossings and to follow the edges, which in our opinion outweighs the additional area and bends.

Our results have been published in [5] and [4].

The important open question in the slog model is: does every graph with maximal vertex degree four admit a slog drawing? Our experiments lead us to believe that it is true, but so far we could not prove it.

5 The Sloggy Orthogonal Drawing Model

5.1 Introduction

We will now extend the slog model from Chapter 4, that allowed crossings only between two diagonal segments, to a more flexible one we call *sloggy*, that allows crossings to occur also between two orthogonal segments.

While one of the advantages of the slog model was the improved visibility of crossings, this was often paid for with an increased number of half-bends. That is why the new model uses the minimal number of half-bends, but at the same time maximizes the number of crossings on diagonal segments (for an example see Figure 5.1c).

Recall from Chapter 1 that in an orthogonal drawing of a graph every vertex occupies a point on the integer grid and edges are drawn as a combination of horizontal and vertical segments. In the slog model, as introduced in Chapter 4, vertices still are drawn on an integer grid, but edges are drawn as a combination of horizontal vertical and diagonal segments. Additionally, crossings are only allowed between two diagonal segments and the minimum angle formed by any two consecutive segments of an edge at the common point has to be 135°.

For graphs G with maximum degree four we showed in Section 4.2.2, that any planar orthogonal drawing $\Gamma_o(G)$ can be transformed into a planar slog drawing $\Gamma_s(G)$ by appropriately replacing each bend of 90° of $\Gamma_o(G)$ by two "*half-bends*" of 135° in $\Gamma_s(G)$; see Figures 4.2a and 4.2b. Hence, twice the number of orthogonal bends of $\Gamma_o(G)$ equals to the number of half-bends of $\Gamma_s(G)$. However, if G is not a planar graph, then the number of half-bends of a bend-optimal slog drawing $\Gamma_s(G)$ is at least twice the number of orthogonal bends of a bend-optimal orthogonal drawing $\Gamma_o(G)$, as stated in Theorem 4.2.2. In fact, slog drawings often require significantly more half-bends than twice the number of bends of the corresponding orthogonal drawing, as we found in the experimental evaluation in Section 4.7 (refer to the plot in Figure 4.18 and, for an example, see Figure 5.1).

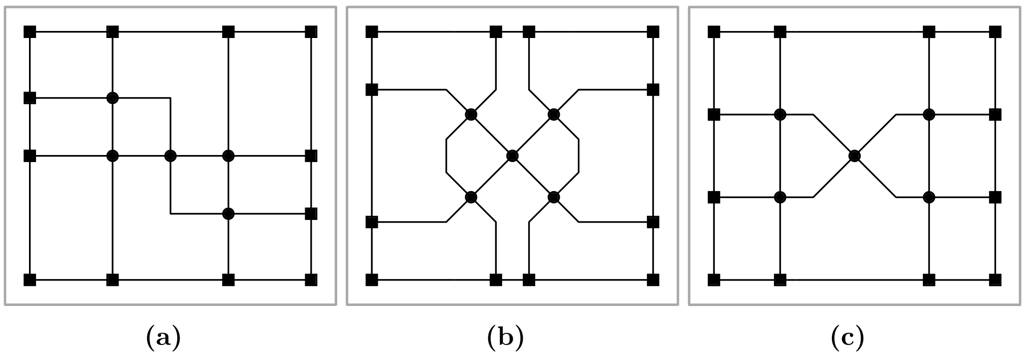


Figure 5.1: Different bend optimal drawings, in which the crossings are illustrated as disks: (a) Orthogonal drawing with two bends. (b) Slanted drawing with twelve half-bends. (c) Sloggy drawing with four half-bends.

In order to reduce the total number of half-bends (that negatively influence the quality of the produced drawings), we propose a mixed approach, according to which a crossing may involve either two rectilinear or two diagonal segments. We balance our preference for crossings that appear at the intersection of two diagonal segments with the need to keep the number of half-bends low, by seeking for drawings that have the following two properties:

Pr-1: The drawings are optimal in terms of the total number of half-bends

Pr-2: The drawings simultaneously maximize the number of crossings on diagonals

We refer to such drawings as *optimal sloggy drawings* or simply as *sloggy drawings*; see Figure 5.1c. We call crossings between two orthogonal segments *orthogonal crossings* and crossings between two diagonal segments *diagonal crossings.*

Figure 5.1 shows the usefulness of the new model and what we might expect from it. More precisely, Figure 5.1a shows a bend-optimal orthogonal drawing containing two bends. A bend-optimal slanted-orthogonal drawing of the same graph is depicted in Figure 5.1b. Observe that the slanted-orthogonal drawing requires 12 half-bends, while the orthogonal one requires only two bends. If the same graph is drawn in the sloggy model as in Figure 5.1c, then it requires only four half-bends (that is, twice the number of orthogonal bends of the bend-optimal orthogonal drawing). However, the number of crossings that appear along two diagonal segments is reduced from five to one.

Following standard practices from the graph drawing literature, in the following we will assume that the input of our problem is a *planar representation* of a graph of maximum degree 4 (or *planarized graph*, for short), which can be computed by the *planarization step* of the *topology-shape-metrics* (*TSM*) approach [110]. By definition of sloggy drawings, it follows that all non-dummy vertices must use rectilinear ports in the produced drawings, while a c-vertex can either use rectilinear or diagonal ports. For consistency, we refer to edges connecting real (crossing) vertices as *rr-edges* (*cc-edges*). Edges between r- and c-vertices are referred to as *rc-edges.*

We will first investigate the properties of sloggy drawings in Section 5.2 before giving an ILP formulation to compute optimal sloggy drawings in Section 5.3.

5.2 Properties of Sloggy Drawings

In this section, we will prove that the number of half-bends of a sloggy drawing $\Gamma_{s\ell}(G)$ of a planarized graph G with maximum degree 4, say $hb(\Gamma_{s\ell}(G))$, is exactly twice the number of bends of a bend-minimal orthogonal drawing $\Gamma_o^{opt}(G)$ of G, say $b(\Gamma_o^{opt}(G))$, i.e., $hb(\Gamma_{s\ell}(G)) = 2b(\Gamma_o^{opt}(G))$. For ease of notation let $\Gamma_j^i(G) = \Gamma_j^i$. To prove the claim, we omit property Pr-2 of sloggy drawings, as it does not influence the total number of half-bends. In particular, we will first prove that there is a sloggy drawing of G that contains exactly $2b(\Gamma_o^{opt})$ half-bends. Then, we will prove that there is no sloggy drawing containing less than $2b(\Gamma_o^{opt})$ half-bends.

5.2.1 The Number of Half-bends of Sloggy Drawings

Lemma 5.2.1. *For a planarized graph G of maximum degree 4, there is a sloggy drawing $\Gamma_{s\ell}(G)$ with $2b(\Gamma_o^{opt})$ half-bends, where $b(\Gamma_o^{opt})$ denotes the number of bends of a bend-minimal orthogonal drawing $\Gamma_o^{opt}(G)$ of G.*

Proof. We employ Tamassia's algorithm [110] to compute a bend-optimal orthogonal drawing Γ_o^{opt} of G with $b(\Gamma_o^{opt})$ bends in total. Then, we can transform Γ_o^{opt} into a sloggy drawing $\Gamma_{s\ell}$ containing exactly $2b(\Gamma_o^{opt})$ half-bends, by adopting the approach of Section 4 that appropriately replaces each bend of Γ_o^{opt} by a pair of half-bends in $\Gamma_{s\ell}$ (see Figures 4.2a and 4.2b). Note that $\Gamma_{s\ell}$ does not maximize the number of crossings on diagonals, as property Pr-2 suggests. In particular, $\Gamma_{s\ell}$ does not contain any crossings on diagonals. However, its existence is enough to show that there is a drawing with the desired amount of half-bends. □

In order to prove that there is no sloggy drawing containing less than $2b(\Gamma_o)$ half-bends, we will neglect the exact geometry of the drawing underneath and focus on its *representation* that simply captures the "shape" of the drawing. Recall that an *orthogonal representation* [110] specifies for each vertex of the graph the angles that are formed between its incident edges, and, for each edge of the graph the number of bends that it contains. A *sloggy representation* additionally specifies for each c-vertex whether its incident edges use rectilinear or diagonal ports. So, let $R_{s\ell}(G)$ ($R_o(G)$) be a bend-minimal sloggy (orthogonal) representation of G with $hb(R_{s\ell}(G))$ half-bends ($b(R_o(G))$ bends). For ease of notation let $R_j^i(G) = R_j^i$. It is known that $b(R_o) = b(\Gamma_o^{opt})$ [110]. On the other hand, it holds that $hb(R_{s\ell}) \leq hb(\Gamma_{s\ell})$. So, it is enough to prove that $hb(R_{s\ell}) \geq 2b(R_o)$.

Lemma 5.2.2. *For a planarized graph G of maximum degree 4 the number of half-bends $hb(R_{s\ell})$ of a bend-optimal sloggy representation $R_{s\ell}(G)$ is not less than twice the number of bends $b(R_o)$ of a bend-minimal orthogonal representation $R_o(G)$ of G, i.e. $hb(R_{s\ell}) \geq 2b(R_o)$.*

Proof. This proof is analogous to the one of Theorem 4.2.2 and also a proof by contradiction. Assume there exists a bend-optimal sloggy representation $R_{s\ell}$ with $hb(R_{s\ell})$ half-bends and a bend-optimal orthogonal representation R_o with $b(R_o)$ bends and $hb(R_{s\ell}) < 2b(R_o)$. We will show how to construct an orthogonal representation R'_o from $R_{s\ell}$ with $b(R'_o) < b(R_o)$, thus constructing the contradiction.

We use the same notion of *rotating* a vertex as in the proof of Theorem 4.2.2, which is illustrated in Figure 4.5. By rotating a vertex by 45° to the left or right we cyclically shift all ports accordingly, which transforms crossing vertices with diagonal ports into real vertices with orthogonal ports. We again consider rc-edges always in the direction from the crossing to the real vertex. Recall that, if there is exactly one half-bend to the left from a crossings vertex v_c to a real vertex v_r and v_c is rotated by 45° to the left, the result will be zero bends on this rc-edge (see Figure 4.5b).

Note that, in the slog model there always is an odd number of half-bends on rc-edges. In the sloggy model however, there is an odd number of half-bends on rc-edges that connect diagonal crossings with real vertices, while the rc-edges connecting orthogonal crossings with real vertices have an even number of half-bends. By construction, rr- and cc-edges always have an even number of half-bends. Let E_{even} be the set of all edges with an even number of half-bends, and $E_{odd} = E \setminus E_{even}$.

We can now describe how to construct R'_o:
We replace all pairs of half-bends on edges in E_{even} by one bend (as in Figures 4.2a and 4.2b). This means that the total number of half-bends on all edges in E_{even} in $R_{s\ell}$ is exactly twice the number of bends on the respective edges in R'_o.

For the edges in E_{odd} we replace each pair of half-bends except the half-bend closest to the diagonal crossing vertex by a bend. Now let $\mathcal{C} = \{C_1, \ldots, C_l\}$ be the set of maximal connected components consisting only of crossing vertices using diagonal ports and the cc-edges between them. By maximality $C_i \cap C_j = \emptyset$ for $i \neq j$. Also, $l > 0$, since otherwise R'_o would already be an orthogonal representation with $b(R'_o) < b(R_o)$, a contradiction.

Let $lhb(C_i)$ be the number of half-bends to the left and $rhb(C_i)$ the number of half-bends to the right on the edges connected to the crossing-vertices in C_i. Note that, by construction, each of the edges connected to the C_i has exactly one half-bend left, all other half-bends that this edges may have had have already been replaced by orthogonal bends.

If for all $i = 1, \ldots, l$ $rhb(C_i) = lhb(C_i)$, we rotate all vertices in the components without loss of generality to the left, and obtain an orthogonal representation R'_o by removing all left half-bends and replacing the right half-bends by a bend. For the so obtained R'_o it holds that $hb(R_{s\ell}) = 2b(R'_o)$ which is a contradiction to the assumption of minimality of R_o.

This means there has to be at least one j with $rhb(C_j) \neq lhb(C_k)$. We first assume that $lhb(C_j) > rhb(C_j)$. By rotating C_j to the left we get an orthogonal representation with even less bends than twice the number of half-bends in $R_{s\ell}$. The same holds if $lhb(C_j) < rhb(C_j)$ and we rotate C_j to the right.

With this method we can construct an orthogonal representation R'_o, such that $2b(R'_o) \leq hb(R_{s\ell}) < 2b(R_o)$, which is a contradiction to the assumption that R_o is optimal.

□

We are now ready to state the main theorem of this section.

Theorem 5.2.3. *For a planarized graph G of maximum degree 4, the number of half-bends $hb(\Gamma_{s\ell}(G))$ of a sloggy drawing $\Gamma_{s\ell}(G)$ is exactly twice the number of bends $b(\Gamma_o(G))$ of a bend-minimal orthogonal drawing $\Gamma_o(G)$ of G, i.e., $hb(\Gamma_{s\ell}) = 2b(\Gamma_o)$.*

Proof. Directly follows from Lemma 5.2.1 and Lemma 5.2.2. □

5.2.2 The Area Requirements of Sloggy Drawings

We now analyze the area requirements of sloggy drawings. For slanted orthogonal drawings it is known that there exist infinitely many graphs whose slanted orthogonal drawing of minimum number of bends require exponential area, assuming that a planarized version of the resulting drawing is given (Theorem 4.6.1). Unfortunately sloggy drawings have the same property:

Theorem 5.2.4. *There exists a graph G whose sloggy drawing $\Gamma_s(G)$ of minimum number of half-bends requires exponential area, assuming that a planarized version $\sigma(G)$ of G is given.*

Proof. Analogous to the slanted-orthogonal case the sloggy drawing of the graphs constructed according to the rules from Section 4.6 require exponential area, since in the sloggy drawing all crossings can be drawn diagonal which results in a slanted-orthogonal drawing. □

5.3 Bend-optimal Sloggy Drawings

In this section we will present two ways to compute bend-optimal sloggy representations. The first method employs cycle systems which transform orthogonal into sloggy representations (Section 5.3.1). Since it is not clear how to efficiently compute the cycle system which gives the bend-optimal sloggy representation, we then present an ILP formulation which can give the optimal solution (Section 5.3.2). To realize the representations we use the algorithm to draw slog representations from Section 4.5.

5.3.1 Cycle Systems

Consider a directed cycle C in the dual G^* of G which contains in its interior only c-vertices of the primal graph G; see the dotted cycle in Figure 5.2a. Initially, say that C is simple (i.e., an edge of G is crossed at most once by the edges of C). If in the sloggy representation $R_s(G)$ we rotate all c-vertices in the interior of C by 45° in the direction implied by the orientation of C, then we obtain a new sloggy representation in which some edges, say $L_1(C) \subseteq E$, of G that are crossed by C save one half-bend, while some others, say $L_2(C) \subseteq E$, admit an extra half-bend; see Figure 5.2b. We say that C is a *zero cycle*, if $|L_1(C)| = |L_2(C)|$; cycle C is called *positive* (*negative*, resp.), if $|L_1(C)| < |L_2(C)|$ ($|L_1(C)| > |L_2(C)|$, resp.). A set of simple cycles $C_1, \ldots, C_k$ is called *zero simple cycle-system*,

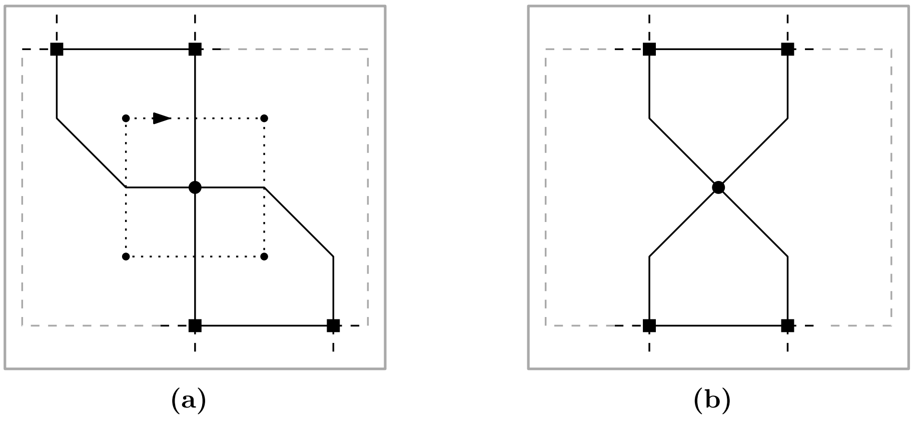


Figure 5.2: (a) A simple (zero) cycle in the dual graph. (b) The result of its application

if $\sum_{i=1}^{k} |L_1(C_i)| = |\sum_{i=1}^{k} |L_2(C_i)|$. Similarly, we define the positive and negative simple cycle-systems. Non-simple cycles in the aforementioned notions are treated as systems of simple cycles, as they are always decomposable into a particular number of simple cycles. There can also be r-vertices in the interior of some cycles, as long as their rotation in the rotation scheme implied by the cycle-system is a multiple of 90°.

Obviously, there can be no negative cycles or cycle-systems in bend-minimal sloggy representations. In a 4-regular plane graph one can convert any orthogonal or sloggy representation to another orthogonal or sloggy representation respectively, just by employing appropriate cycles or cycle-systems to rotate the vertices. The sufficiency of this operation is guaranteed by the facts that there are no unoccupied ports at the vertices and that the embedding is fixed.

If the planarized graph contains vertices of degree two or three, it might be necessary to change some of the angles that are formed by the edges incident to degree two or three vertices. This cannot be realized by the cycle-systems that we have developed so far. However, Figure 5.3 illustrates a modification of our cycle-system, which supports cycles that also pass through vertices and are able to change the angles as well.

By employing appropriate zero cycles or zero cycle-systems consisting of cycles of one of the two types that we presented in this section, one can trans-

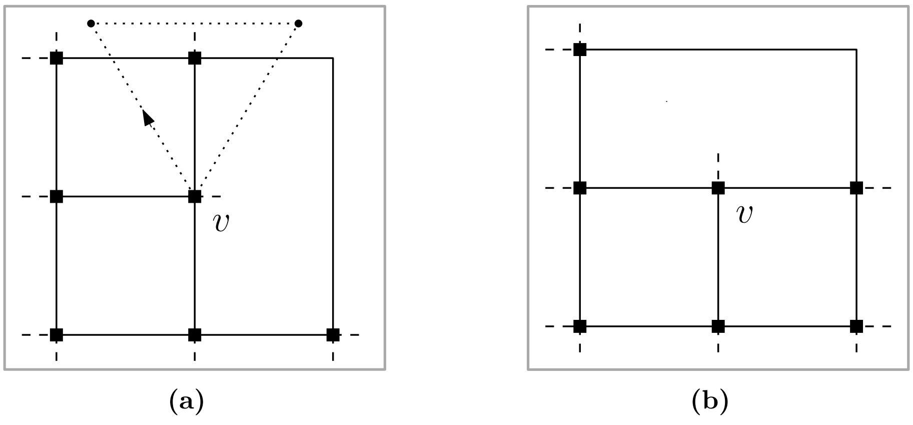


Figure 5.3: (a) A (zero) cycle in the "dual" graph. (b) The result of its application on vertex v is a new port assignment at v.

form a bend-optimal sloggy representation as the one implied by the bend-optimal drawing of Lemma 5.2.1 (which contains no crossings on diagonals) into a bend-optimal one containing the maximum number of crossings along diagonals (as property Pr-2 of sloggy drawings requires). Unfortunately, it is not clear and we do not know yet how to efficiently compute the required cycle-system that realizes this transformation. We strongly believe that this problem is an NP-hard problem, but we have not managed to prove it. For this reason, we give an ILP formulation for this problem in the next section.

5.3.2 An ILP to compute Sloggy Drawings

Since in sloggy drawings the number of half-bends has to be equal to two times the number of orthogonal bends in a bend-optimal orthogonal drawing, we compute in a first step a bend-optimal orthogonal representation $R_o(G)$ of the (planarized) input graph with the algorithm of Tamassia ([110]). From $R_o(G)$ we compute an initial port-assignment for all vertices. We use an ILP to compute which crossings will be drawn diagonal and which crossings stay orthogonal. An overview of the complete linear program is given in Linear Program 2, we will explain the single constraints in detail.

We start with some notation: Let $G = (V, E)$ be the planarized graph we want to draw in the sloggy model. We denote the set of all crossing vertices by $\mathcal{C}$. By $o(e = (u, v))$ we denote the number of bends to the right on edge e

from vertex u to vertex v in $R_o(G)$. Bends to the left are denoted by negative values.

In the ILP we introduce an integer variable h for all edges $e \in E$ that will hold the number of right half-bends on the respective edges (again negative values mean left half-bends). Additionally we need integer variables h^+ for all $e \in E$ that will count the number of half-bends on edge e. Constraints 5 and 6 then give lower bounds to the h^+ variables. The upper bound is implicitly contained in the objective function, when the number of half-bends has to be equal to two times the number of bends in $R_o(G)$ and the number of diagonal crossings is maximized.

$$\begin{array}{lll r}
\textbf{max} & w_d \cdot (\sum_{v\in\mathcal{C}} b_i(v) |\ i \text{ is odd}) - w_f \cdot (\sum_{e\in E} f(e)) & & \\
\textbf{s.t.} & h^+(e) \geq h(e) & \forall e \in E & (5) \\
& h^+(e) \geq -h(e) & \forall e \in E & (6) \\
& \sum_{i\in\{-7\ldots7\}} b_i(v) = 1 & \forall v \in \mathcal{C} & (7) \\
& \frac{h^+(e)-1}{2} \leq f(e) \leq \frac{h^+(e)}{2} & \forall e \in E & (8) \\
& \sum_{e\in E} h^+(e) = 2 \cdot \sum_{e\in E} o(e) & & (9) \\
& h(e) = 2 \cdot o(e) & & \\
& -b_1(v_1) - 2 \cdot b_2(v_1) - \ldots - 7 \cdot b_7(v_1) & & \\
& +b_{-1}(v_1) + \ldots + 7 \cdot b_{-7}(v_1) & & \\
& +b_1(v_2) + \ldots + 7 \cdot b_7(v_2) & & \\
& -b_{-1}(v_2) - \ldots - 7 \cdot b_{-7}(v_2) & & \\
& -8 \cdot c(v_1) + 8 \cdot c(v_2) & \forall e \in E & (10)
\end{array}$$

Linear Program 2: ILP to compute bend-optimal sloggy representations

Furthermore we need binary variables $b_i, i \in -7, \ldots, 7$ for each vertex $v \in \mathcal{C}$ that determine if the respective (crossing) vertex is rotated counter-clockwise (negative indices) or clockwise (positive indices). The assignment $b_1(v) = 1$ then means that the ports on v are rotated clockwise by 45°, $b_2(v) = 1$ means they are rotated by $2 \cdot 45°$ and so on. Since each vertex has to be rotated by exactly one value (with $b_0 = 1$ signaling no rotation), we require the sum of the b_i to be equal to one (Constraint 7).

We also need integer variables $c(v)$ for all vertices $v \in \mathcal{C}$ that hold the number of 360° rotations (positive values meaning clockwise and negative values counterclockwise) that have to be applied to a vertex. To count the number of full bends (2 half-bends $\equiv$ 1 full bend) on an edge we introduce variables $f(e)$ for all edges, for which we assign the correct values with Constraint 8.

With Constraint 9 we make sure that the sloggy solution has the correct number of half-bends.

Since we start from a bend-optimal orthogonal representation the angles between adjacent edges on a vertex are fixed. If a vertex has degree less than 4 we may require this angles to change in order to compute an optimal solution. This is modeled in the following way. For all edges e that are incident to a vertex v with degree less than 4 we introduce variables $p(v, e)$ that are integer variables for degree 2 and binary variables for degree 3 vertices. If $p(v, e)$ has a value greater than zero in the solution it means, that the port of e is rotated clockwise by $p(v, e) \cdot 90°$ around v. Degree 1 vertices require no such variables since there is only one edge.

The set of constraints required to model the shifting of the ports around a degree two or three vertex is given in Constraint Set 1. Constraint 1 is added for each degree three vertex v, with e_1, e_2 and e_3 being its three incident edges. For each pair of edges e and e' consecutive in the clockwise order around v the Constraint 2 of the set is added.
For each degree two vertex v with incident edges e_1 and e_2 we add the Constraint 3. For each pair of consecutive edges e and e' on v we add one of the Constraints 4-6, depending on the angle α formed between e and e' on v.

$$p(v, e_1) + p(v, e_2) + p(v, e_3) \leq 2 \quad \textbf{1}$$
$$p(v, e) \geq p(v, e') \quad \textbf{2}$$
$$p(v, e_1) + p(v, e_2) \leq 1 \quad \textbf{3}$$
$$\alpha = 90° : p(v, e) \leq p(v, e') \quad \textbf{4}$$
$$\alpha = 180° : p(v, e), p(v, e') \leq 1 \quad \textbf{5}$$
$$\alpha = 270° : p(v, e') \leq p(v, e) \quad \textbf{6}$$

Constraint Set 1: Constraints for degree two and three vertices.

The key constraints of the ILP are the ones assigning the correct value to the h variables. We have to distinguish between edges connecting a real vertex and a crossing vertex and edges connecting two crossing vertices. We describe in Constraint 10 exemplary the constraints for an edge $e = (v_1, v_2)$ connecting two crossing vertices since it is the most complicated one. For the rc-edges some of the rotations are forbidden (the real vertex must only be rotated in 90° steps), and the constraint has to be modified accordingly.

With the constraints according to Constraint 10 the variables h get assigned the right value by “counting” the number of half-bends as follows: If neither v_1 nor v_2 are rotated $(d_0(v_1) = d_o(v_2) = 1)$ then the number of half-bends on e is twice the number of bends in $R_o(G)$. If for example v_1 is rotated clockwise $(b_1(v_1) = 1)$, this adds a left half-bend to e and therefore we subtract $b_1(v_1)$ from $h(e)$. Since left half-bends cancel out right half-bends and vice versa, we can determine the number of half-bends on e by summing up the b_i- and c-variables with appropriate coefficients.

Should a vertex v of e be of degree two or three, the equation for $h(e)$ has to be extended by $\pm 2 \cdot p(v, e)$ to account for the half-bends caused by shifting e around v.

We also want to allow the ILP to rotate the ports on real vertices. Here there are only three possible rotation values in each direction and each step results in a difference of two half-bends. The constraints and variables are chosen analogously to the crossing vertex case.

In the objective function for our ILP we use the strength of linear programming that enables us to optimize with respect to several objectives at the same time. In our experiments we realized that it not only makes sense to maximize the number of diagonal crossings but also to strive for a drawing where the half-bends are distributed over the edges of the graph as evenly as possible. This can be achieved by minimizing the number of full bends (a full bend consists of two consecutive half-bends on an edge) of the whole drawing. To be able to weigh this two optimization criteria against each other we introduce appropriate weights w_d for the number of diagonal crossings and w_f for the number of full bends and include them in the objective function as in Linear Program 2.

By adjusting the weights for diagonal crossings and full bends it is possible to determine which part of the optimization is more important. By allowing more bends per edge with a lower weight on the full bends more diagonal crossings become possible, which makes it easier to distinguish them from real vertices. Penalizing many bends per edge by increasing w_f on the other hand may lead to less diagonal crossings but edges of simpler shape, making it easier to follow them.

The solution of this ILP determines which crossings can be drawn diagonally and which crossings have to stay orthogonal. By construction the number of half-bends that the solution will use is two times the number of bends in the bend-optimal orthogonal drawing. To compute the actual drawing from the representation we get based on the result from the ILP we use the algorithm to compute slanted-orthogonal drawings from Section 4.5 and treat orthogonal crossings like normal vertices.

5.4 Sample Drawings

We implemented the approach from the previous section and used the gurobi solver [64] to solve the ILP for the sloggy representation. We found that even for large input graphs (> 600 vertices, > 1000 edges) we were able to compute sloggy drawings in less than 10 seconds on a standard desktop machine, which suggests that our algorithm is usable for practical applications.

In the following, we give some examples for drawings produced by our algorithm. Figure 5.4a shows a bend optimal orthogonal drawing of the input graph. It requires 16 bends. The sloggy drawings of the same graph in Figures 5.4b, 5.5a and 5.5b all have 32 half-bends.

In Figure 5.4b we show the result of maximizing the number of diagonal crossings by assigning a weight of 0 to the full bends. The result has 16 diagonal crossings but many edges that require two half-bends. In Figure 5.5a the number of full bends is minimized by assigning a weight of 0 to the number of diagonal crossings, which results in a drawing where no edge has more than one half-bend, but the total number of diagonal crossings is reduced to 11.

The final Figure 5.5b shows the result if both weights are set to 1. There is only one edge left that has more than one half-bend while the number of diagonal crossings is 13.

The figures from this section show that our algorithm not only produces aesthetically pleasing drawings, but is quite flexible with respect to the two optimization criteria presented here, namely the number of diagonal crossings and the number of the full bends. By adjusting the weights in the objective function accordingly we can compute drawings adapted to the task at hand.

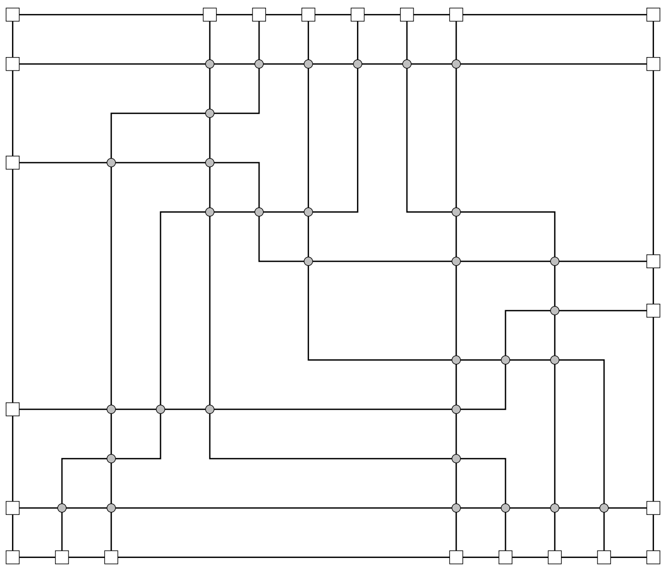
5.5 Summary

In this chapter we introduced a more flexible variant of the slog model from the previous chapter, which we call sloggy. The sloggy model has the same properties as the slog model, but additionally crossings are also allowed between horizontal and vertical segments. This model is motivated by the observation that insisting on crossings only on diagonals often leads to a large number of half-bends. We showed that the number of half-bends of a bend-optimal sloggy representation is exactly twice the number of bends of the corresponding bend-optimal orthogonal representation. We gave an ILP formulation to compute a bend-optimal sloggy representation and showed how to realize it using the method for slog representations. Analogous to the slog model, sloggy drawings also may require exponential area.

We published our results on the sloggy model in [3].

There are several open problems for the sloggy model:

- Is there an efficient algorithm to compute an optimal sloggy representation?
- If no such algorithm exists, what is the complexity to determine the optimal sloggy representation?
- Given an (optimal) sloggy representation, how can a drawing realizing this representation efficiently be computed? (This question is also still open for slanted orthogonal drawings.)
- Can the exponential area requirement be avoided if one does not insist on the maximum number of diagonal crossings?
- Investigate the effect of different combinations of weight factors in the objective function of the ILP.


(a)

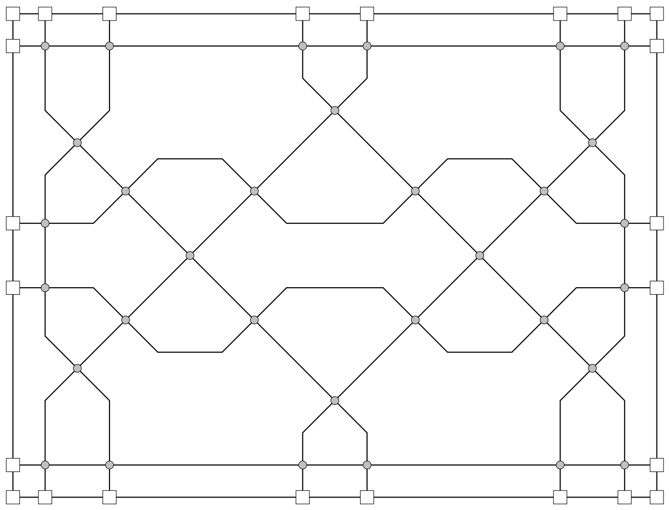
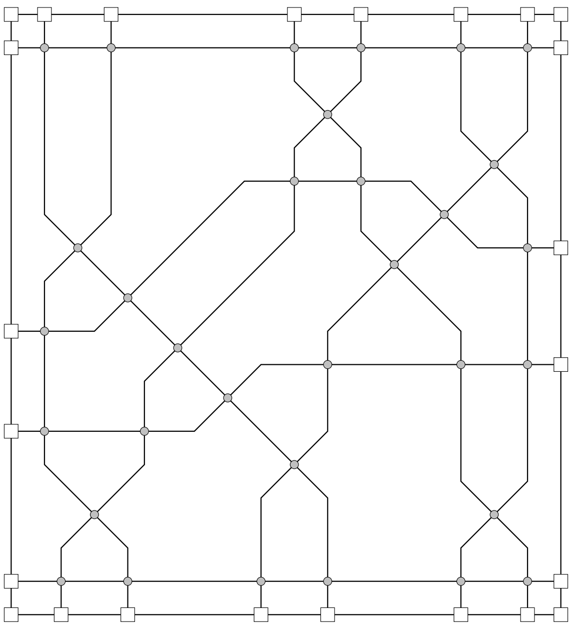
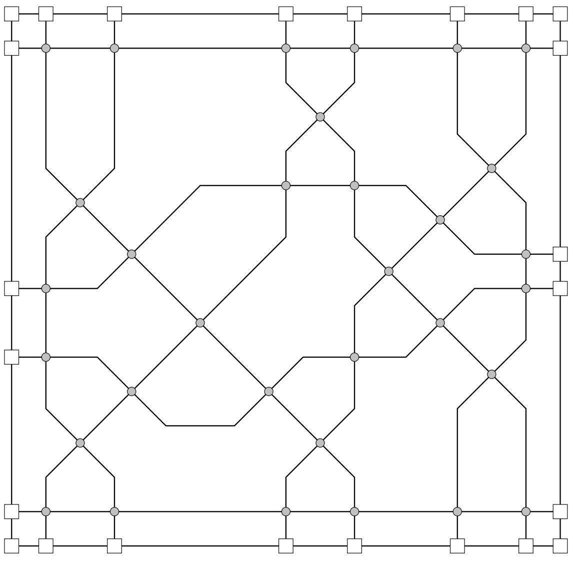

(b)

Figure 5.4: (a) Bend optimal orthogonal drawing of a graph. (b) Sloggy drawing of the same graph with maximum number of diagonal crossings.



(a)



(b)

Figure 5.5: Sloggy drawings of the graph from Figure 5.4a: (a) with minimum number of orthogonal bends. (b) with equal weights for diagonal crossings and full bends.

6 Extending the Kandinsky Model

6.1 Introduction

In this chapter we will present an extension of the well known Kandinsky model [54]. Recall from the introduction (Chapter 1) that the Kandinsky model employs two grids; a coarse one on which the vertices are placed, and a fine one to route the edges. Vertices are drawn as boxes centered on grid points on the coarse grid and edges are drawn as a combination of horizontal and vertical line segments on the fine grid. The distinctive feature of the Kandinsky model is that an arbitrary number of edges can be connected to (any side of) a vertex.

A central aspect of the Kandinsky model is, that so called *empty faces* are avoided, since they can either not be drawn without overlaps (see Figure 6.1a) or make it very hard to distinguish between edges (see Figure 6.1b). Because of their shape they are also called *empty L* and *empty T*. In order to avoid this unwanted shapes usually the so called *bend-or-end* property is used, which only allows one unbent edge on each side of a vertex. To achieve this, Fößmeier et al. [54], for example, required that for each zero degree angle

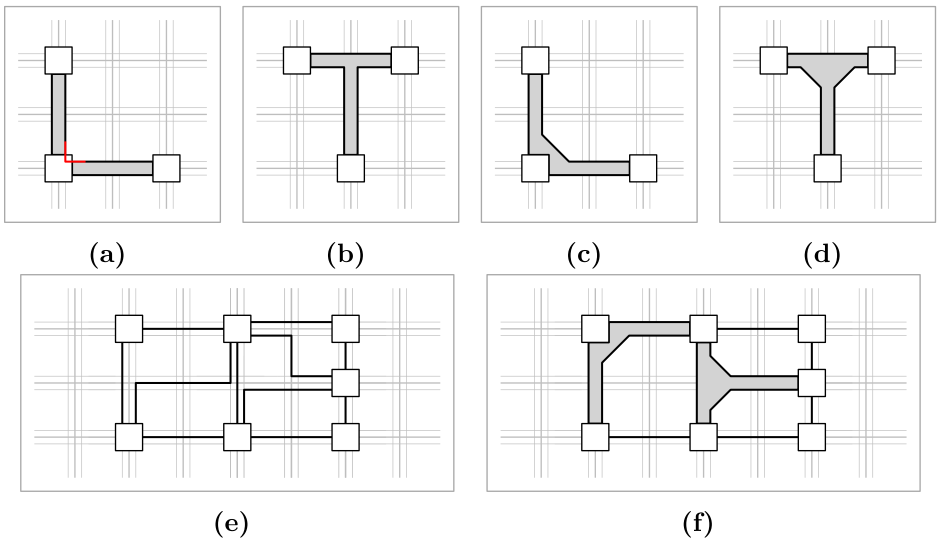


Figure 6.1: Illustration of empty faces: (a) empty L and (b) empty T; Illustration of almost-empty faces: (c) almost-empty L and (d) almost-empty T; Illustration of a (e) Kandinsky drawing and a (f) podevsaef drawing of the same graph.

between two edges connected to the same vertex, there has to be a bend, the so called *Kandinsky bend*.

We now want to use diagonal segments, as introduced in the slog model in Section 4, to be able to draw empty faces in a way that avoids the problems occurring in the orthogonal model. By employing diagonal segments, we are able to draw the previously forbidden shapes in a way that makes the faces *almost empty*, as can be seen in Figures 6.1c and 6.1d. To be consistent in the way we draw bends, we also replace all 90° bends by two half-bends, as in the slog model. We refer to this drawings as *Kandinsky drawings with almost empty faces* or, for short, *podevsaef drawings*, which stands for **P**lanar **O**rthogonal **D**rawings with **E**qual **V**ertex **S**ize and **A**lmost **E**mpty **F**aces. The term podevsaef stems from another term that also refers to classical Kandinsky drawings: *podevsnef drawings*, which stands for **P**lanar **O**rthogonal **D**rawings with **E**qual **V**ertex **S**ize and **N**o **E**mpty **F**aces. Figure 6.1f shows an example drawing in the new model and the advantages we get by allowing almost-empty faces: since two half-bends are always in correspondence with one 90° bend, the total number of bends is reduced and the readability of the drawing increased, as can be seen when comparing with classical Kandinsky drawings as in Figure 6.1e. For a larger example refer to Figure 6.13.

If we ask for the bend-optimal podevsaef drawing, a recent result by Bläsius et al. [20] immediately implies that the problem is NP-complete. This is why we give an ILP formulation based on the one of Eiglsperger et al. [43], that is able to compute the optimal solution in Section 6.2. As expected and verified in our experiments, computing the optimal solution can be time consuming. In fact, with increasing number of triangular faces (that allow for empty L- or T-shapes) the ILP approach becomes unusable for practical applications. This motivated us to also develop an efficient heuristic, which we present in Section 6.3. The heuristic computes podevsaef drawings by modifying classical Kandinsky drawings, but it results in drawings that are not bend-optimal. We experimentally evaluate the approaches and compare them with classical Kandinsky drawings in Section 6.4.

6.2 Optimal Drawings with ILP

Following the TSM-approach (see Section 2.4.1), we first compute a representation, and, in a second step, a drawing realizing it. We will first recall the ILP formulation for the bend minimization in the Kandinsky model as introduced by Eiglsperger et al. [43] in Section 6.2.1. Then we will show how to extend this formulation to the new podevsaef model in Section 6.2.2. Finally we show in Section 6.2.3 how to use an easy transformation on the representation to be able to use known compaction algorithms for the Kandinsky model to compute podevsaef drawings.

6.2.1 Bend-Optimal Kandinsky Representations

For each edge $e = (u, v)$, variable $a_{(u,v)} \cdot 90°$ corresponds to the angle formed by edge e and its cyclic predecessor at vertex u. Clearly, $a_{(u,v)} \in \{0, 1, 2, 3, 4\}$. Since the sum of the angles around a vertex equals to 360°, it follows that for each vertex $u \in V$, $\sum_{(u,v),v\in N(u)} a_{(u,v)} = 4$ must hold, where $N(u)$ denotes the neighbors of u.

$$
\begin{array}{llll}
\textbf{min} & \sum_{(u,v)\in E}(l_{(u,v)} + r_{(u,v)}) & & \\
\textbf{s.t.} & 0 \le a_{(u,v)} \le 4 & \forall (u,v) \in E & (11)\\
& \sum_{(u,v)\in N(u)} a_{(u,v)} = 4 & \forall u \in V & (12)\\
& \sum_{(u,v)\in f}(a_{(u,v)} + l_{(u,v)} - r_{(u,v)}) & & \\
& \quad = \begin{cases} 2a(f)-4; & f \text{ bounded} \\ 2a(f)+4; & f \text{ unbounded} \end{cases} & \forall f \in F & (13)\\
& lb^u_{(u,v)} + rb^u_{(u,v)} \le 1 & \forall (u,v) \in E & (14)\\
& lb^u_{(u,v)} = rb^u_{(v,u)} & \forall (u,v) \in E & (15)\\
& lb_{(u,v)} = rb_{(v,u)} & \forall (u,v) \in E & (16)\\
& lb^v_{(u,v)} = rb^v_{(v,u)} & \forall (u,v) \in E & (17)\\
& a_{(v,u)} + lb^v_{(v,w)} + rb^v_{(v,u)} \ge 1 & \forall (v,w),(v,u) & \\
& & \text{subsequent in } N(v) & (18)
\end{array}
$$

Linear Program 3: The ILP from Eiglsperger et al. [43] to compute bend-optimal Kandinsky representations.

In order to count the number of left turns (or simply *left-bends*) along each edge $e = (u,v)$, three variables, $lb^u_{(u,v)}$, $lb^v_{(u,v)}$ and $lb_{(u,v)}$, are employed, which correspond to the left Kandinsky-bend (i.e., the special bend resulting from the bend-or-end property) at vertex u, the left Kandinsky-bend at vertex v and the remaining left-bends of edge (u,v). For the right-bends, variables $rb^u_{(u,v)}$, $rb^v_{(u,v)}$ and $rb_{(u,v)}$ are defined similarly. Clearly, for reasons of symmetry $lb^u_{(u,v)} = rb^u_{(v,u)}$, $lb_{(u,v)} = rb_{(v,u)}$ and $lb^v_{(u,v)} = rb^v_{(v,u)}$ must hold. Note that variables $lb^u_{(u,v)}$, $lb^v_{(u,v)}$, $rb^u_{(u,v)}$ and $rb^v_{(u,v)}$ are binary, while variables $lb_{(u,v)}$ and $rb_{(u,v)}$ are non-negative integers.

Since only one Kandinsky-bend is allowed at each end of each edge, $lb^u_{(u,v)}+rb^u_{(u,v)} \le 1$ must hold for each edge $(u,v) \in E$. For ease of notation, we denote by $l_{(u,v)} = lb^u_{(u,v)}+lb_{(u,v)}+lb^v_{(u,v)}$ and $r_{(u,v)} = rb^u_{(u,v)}+rb_{(u,v)}+rb^v_{(u,v)}$ the total number of left and right bends per edge, respectively. Since the sum of the angles formed at the vertices and at the bends of a bounded face f equals to $180 \cdot (p(f) - 2)$, where $p(f)$ denotes the number of such angles, it follows that $\sum_{(u,v)\in f}(a_{(u,v)}+l_{(u,v)}-r_{(u,v)}) = 2a(f)-4$, where $a(f)$ denotes the number of vertex angles in f. If f is unbounded, the sum is increased by 8. Empty faces are forbidden by requiring $a_{(v,u)} + lb^v_{(v,w)} + rb^v_{(v,u)} \geq 1$, for all pairs of consecutive edges (v,w) and (v,u) around all vertices $v \in V$. Of course, the objective function of the corresponding ILP-formulation must minimize the sum of all (i.e., either left or right) bends, that is $\min \sum_{(u,v)\in E}(l_{(u,v)} + r_{(u,v)})$. The complete linear program is given in Linear Program 3.

6.2.2 Bend-Optimal podevsaef Representations

To enable the ILP-formulation of the previous section to use the almost-empty T and almost-empty L-shapes (see Figure 6.1a and 6.1b, respectively), we first assume that a bend of a classical Kandinsky drawing always corresponds to a pair of half-bends in our model (therefore, the notions of left and right-bends are well-defined) and then we replace Constraint 18 of Linear Program 3 with new ones (refer to Constraint Sets 2 and 3). More precisely, for each triangular face f, we introduce two binary variables, say T_f and L_f, that are set to one if and only if f is drawn using the almost-empty T or the almost-empty L-shape, respectively. We also employ a large constant M which we use to "activate" or "deactivate" constraints; a common trick used in formulating ILPs [24]. We denote the set of edges of a face f by E_f.

If variable T_f of face f is set to one (i.e., f is drawn using the almost-empty T-shape), then Constraint 1 of Constraint Set 2 ensures that the angles at all vertices of f are zero. Constraints 2 and 3 force f to have in total two right-bends on all edges (so, the third edge of face f must be bend-less). Constraint 4 ensures that no edge has any left-bend and Constraint 5 forces all edges to have at most one right-bend in total. On the other hand, if T_f is set to zero, then these constraints are all deactivated using the constant M, so that they impose no restriction on the representation.

The constraints for the L-shape are similar. If variable L_f is set to one, then Constraints 6 and 7 of Constraint Set 3 ensure that there is a 90° angle at exactly one vertex in face f and a 0° angle at all other vertices. Constraints 8 and 9 force f to have in total exactly one right-bend and Constraint 10 makes sure that there are no left-bends. Again, setting L_f to zero trivially fulfills all these constraints and they pose no restriction on the representation. Note that the constraints for T- and L-shapes exclude each other. So, there

$\sum_{e\in E_f} a_e \leq 0 + (1 - T_f)\cdot M$	**1**
$\sum_{e\in E_f} r_e \geq 2 - (1 - T_f)\cdot M$	**2**
$\sum_{e\in E_f} r_e \leq 2 + (1 - T_f)\cdot M$	**3**
$\sum_{e\in E_f} l_e \leq 0 + (1 - T_f)\cdot M$	**4**
$\forall e \in E_f : r_e \leq 1 + (1 - T_f)\cdot M$	**5**

Constraint Set 2: T-shaped face f

$\sum_{e\in E_f} a_e \leq 1 + (1 - L_f)\cdot M$	**6**
$\sum_{e\in E_f} a_e \geq 1 - (1 - L_f)\cdot M$	**7**
$\sum_{e\in E_f} r_e \leq 1 + (1 - L_f)\cdot M$	**8**
$\sum_{e\in E_f} r_e \geq 1 - (1 - L_f)\cdot M$	**9**
$\sum_{e\in E_f} l_e \leq 0 + (1 - L_f)\cdot M$	**10**

Constraint Set 3: L-shaped face f

is no reason to add an extra constraint for this purpose.

Since we intend to allow either the almost-empty T or the almost-empty L-shape, it follows that it suffices to replace Constraint 18 of Linear Program 3 with the following constraint for each triangular face f:

$$a_{(u,v)} + lb^v_{(v,w)} + rb^v_{(v,u)} + T_f + L_f \geq 1,\ \forall (v,w),(v,u) \text{ subsequent in } N(v)$$

Using this modified constraint it is also possible to restrict the almost empty faces to only the L-shape or the T-shape by just adding the respective variable. So, if for example only L-shapes are to be used, the constraint for all triangular faces f would be:

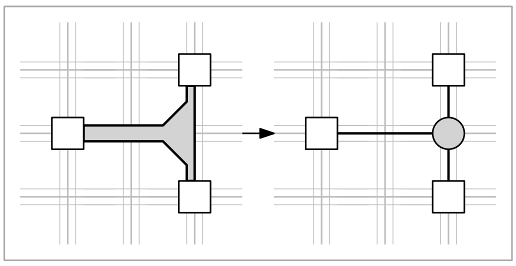
$$a_{(u,v)} + lb^v_{(v,w)} + rb^v_{(v,u)} + L_f \geq 1,\ \forall (v,w),(v,u) \text{ subsequent in } N(v)$$

Note that, even if the constraints for the T_f variables are still part of the linear program, in the solution there will be no almost-empty T-shapes, since without adding the T_f variable to the Constraint 18 the configuration of the almost empty T-shape does not fulfill the restrictions posed by the constraint.

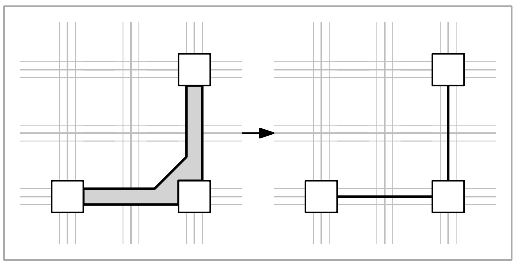
In order to prove that the solution of the modified ILP corresponds to a valid podevsaef-representation, we first observe that by setting all T_f and L_f variables to zero all new constraints no longer affect the equation system and all old constraints stay exactly as in the original formulation of Eiglsperger et al. [43]. So, the original proof of correctness holds. On the other hand, it is not difficult to see that if a face is to be drawn either as an almost-empty T or as an almost-empty L-shape, then Constraint Sets 2 and 3 respectively ensure that all angles and edge-bends are appropriately computed.

6.2.3 Realizing the Representation

As already stated, in the compaction phase (where the computed representation has to be transformed into an actual drawing) we employ a simple transformation that allows us to use any known algorithm for the compaction phase of the original Kandinsky model; for an example refer to [54, 55, 44, 21]. The transformation is illustrated in Figure 6.2. More precisely, for an almost-empty T-shaped face a new auxiliary vertex is required (refer to the gray colored vertex in Figure 6.2a) and the angles follow directly from the T-shape. For an almost-empty L-shaped face, we simply ignore the bent edge involved (see Figure 6.2b).



(a)



(b)

Figure 6.2: (a) Transformation for T-shapes. (b) Transformation for L-shapes.

Once all almost-empty T-shaped and L-shaped faces are transformed according to the rules of Figure 6.2, we proceed to draw the new representation using one of the known compaction algorithms for the original Kandinsky model. In the resulting drawing, the applied transformations can be easily reversed by introducing the missing edges of the L-shaped faces and replacing the auxiliary vertices of the T-shaped faces with the original edges. Note that if at least one of the sides of an T or L-shaped face is of unit length, then the whole drawing must be scaled-up by a factor of two to accommodate the necessary diagonal segments on the underlying integer-grid. However, this scale-up does asymptotically not influence the required drawing area.

6.3 Heuristic

In this section, we present a heuristic which, given an orthogonal representation of minimum number of bends, computes a podevsaef drawing with equal or even fewer bends (assuming that a bend of a traditional orthogonal drawing is always in correspondence with a pair of half-bends of a podevsaef drawing). This heuristic is motivated by the observation that insisting on bend-optimal podevsaef drawings may require a lot of time to solve the corresponding linear program for computing the required bend-optimal podevsaef representation (as we will shortly see in Section 6.4).

The main idea of our approach is to start from a traditional bend-optimal orthogonal representation and heuristically modify the shape of as many triangular faces as possible, to become T-shaped or L-shaped. To achieve this, we have identified several shapes that allow an easy transformation into the new almost-empty shapes; see Figure 6.3. As in the previous section, these transformations do not require a drawing, but they are directly applicable on a given (not necessarily bend-optimal) orthogonal representation.

Once all triangular faces have been transformed according to these rules, one can either proceed with the compaction phase to obtain the final drawing (as described in the previous section) or heuristically try to further improve the number of bends by adding another orthogonalization step, hoping that the transformed graph allows a drawing with even less bends. For the second orthogonalization step the transformations of Figure 6.2 need to be applied to the graph first, and it has to be made sure that the affected edges are not bent. This can be achieved by inflicting appropriate penalties on bends on this edges. Since we start from a valid representation, the maximal number of half-bends required is known, so this can be used as cost for bends on fixed edges, thereby guaranteeing that they will not get bends in the second

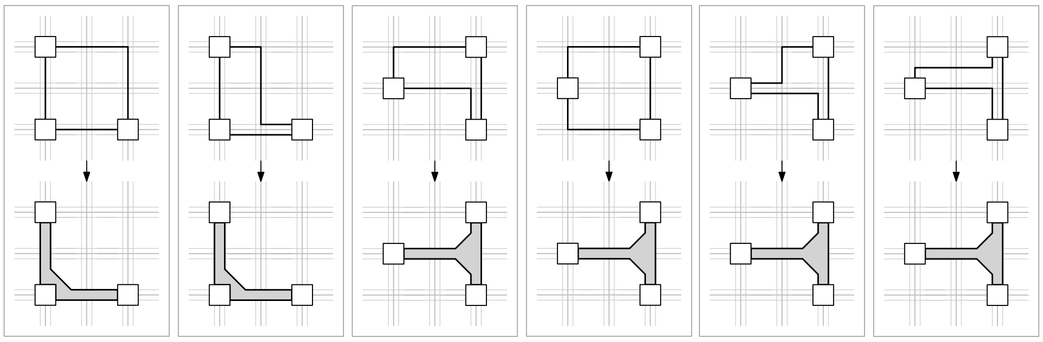


Figure 6.3: (a),(b) Shapes that can be transformed into an L. (c)-(f) Shapes that can be transformed into a T.

orthogonalization step. Additionally, the angles formed at the edges involved in almost-empty L-shapes or T-shapes also have to be fixed by imposing appropriate restrictions in the orthogonalization phase. Theoretically this step could be repeated as long as the number of bends is improved. However, in our experiments we found that more than one repetition did not improve the number of half-bends any further. Our result is summarized in the following theorem.

Theorem 6.3.1. *Given a Kandinsky drawing* $\Gamma(G)$ *with* b *bends, the heuristic computes a podevsaef drawing* $\Gamma'(G)$ *with* b' *half-bends with* $b' \leq 2b$.

Figure 6.13 shows an example of a graph drawn in the classical Kandinsky model and a bend-optimal podevsaef drawing as well as a drawing produced by the heuristic of this section of the same graph.

6.3.1 Optimizing the Input for the Heuristic

As already mentioned, one bend of a Kandinsky drawing is in correspondence with two half-bends of a podevsaef drawing. We denote by $b(e)$ the number of orthogonal bends on an edge e in the input Kandinsky representation, and by $hb(e)$ the number of half-bends of e in the podevsaef representation computed by the heuristic. By applying the transformations according to Figures 6.3a, 6.3c, or 6.3d, it holds for the transformed edges that $2b(e) = hb(e)$. However, if one of the transformations according to Figures 6.3b, 6.3e or 6.3f is applied, for some edges $b(e) = hb(e)$ holds. This is true for edges with exactly one left bend and one right bend in the Kandinsky representation. We say that such an edge has an *S-shape*.

In the original Kandinsky the S-shape is rarely used; in fact it only occurs in the presence of Kandinsky bends. Since the overall number of half-bends in a podevsaef representation decreases as the number of S-shaped edges in the input Kandinsky representation increases, we now show a way to modify ILP 3 such that it can be used to compute a bend-optimal Kandinsky representation with maximum number of S-shaped edges.

We define for each edge e five new variables. Variables l_e^0 and l_e^1 are set to one if and only if e has zero ($l_e^0 = 1$) or exactly one ($l_e^1 = 1$) left bend. Variables r_e^0 and r_e^1 are defined for right bends accordingly. Additionally, variable s_e is set to one if and only if in the representation e has an S-shape, which is the case when e has exactly one left bend and one right bend. All new variables are binary variables. Recall that we abbreviate the sum of the left bends as $lb_e + lb_e^u + lb_e^v = l_e$ and the sum of the right bends as $rb_e + rb_e^u + rb_e^v = r_e$.

$l_e^0 \geq 1 - l_e$	**1**
$(1 - l_e^0) \cdot M \geq l_e$	**2**
$r_e^0 \geq 1 - r_e$	**3**
$(1 - r_e^0) \cdot M \geq r_e$	**4**
$l_e^1 \leq l_e$	**5**
$(1 - l_e^1) \cdot M \geq l_e - 1$	**6**
$l_e - 1 \geq 1 - l_e^1 - 2l_e^0$	**7**
$r_e^1 \leq r_e$	**8**
$(1 - r_e^1) \cdot M \geq r_e - 1$	**9**
$r_e - 1 \geq 1 - r_e^1 - 2r_e^0$	**10**
$l_e^1 + r_e^1 \leq 1 + s_e$	**11**
$s_e \leq l_e^1$	**12**
$s_e \leq r_e^1$	**13**

Constraint Set 4: Constraints for the recognition of S-shapes.

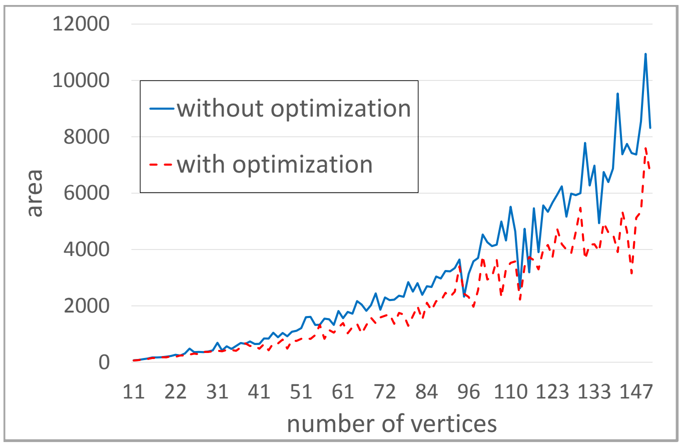


Figure 6.4: Area against number of vertices for the two variants of the heuristic.

The constraints in Constraint Set 4 then model the correct assignment of values to the newly introduced variables. Constraints 1 and 3 ensure that l_e^0 and r_e^0 are assigned value 1 when there are no bends on e. With Constraints 2 and 4 we make sure that, if there are bends on e, l_e^0 and r_e^0 are set to zero. If e has no bends then Constraints 5 and 8 make sure that l_e^1 and r_e^1 are set to zero, and if there is more than one bend on e this is taken care of with Constraints 6 and 9. To force variables l_e^1 and r_e^1 to be one in the case that there is exactly one bend on e, we add Constraints 7 and 10. As before, we use a sufficiently large constant M to "deactivate" constraints when they are not needed.

Now that the binary variables for zero bends and exactly one bend have the right value, we can model the assignment for the binary variables that indicate whether an edge has a S-shape in the representation. With Constraint 11 we make sure that s_e is set to one if e has exactly one left bend and one right bend. Constraints 12 and 13 make sure that s_e has value zero in any other case.

In order to maximize the number of edges that are drawn with an S-shape we modify the objective function as follows:

$$\min w_b \sum_{e\in E}(r_e + l_e) - w_s \sum_{e\in E} s_e$$

We weight the two parts of the objective function to make sure that the solution stays bend-optimal by setting the weight of a single bend to $w_b = |E|$

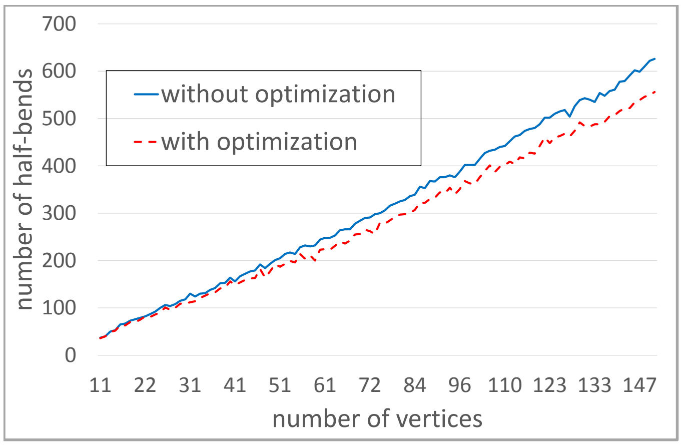


Figure 6.5: Half-bends against number of vertices for the two variants of the heuristic.

and the weight of an S-shape to $w_s = 1$.

To validate that this changes actually improve the results of the heuristic, we implemented both variants and evaluated the number of half-bends, the area requirement and the running time for a test-set consisting of 940 randomly created triangulations. All computations were done on a standard Linux machine with 4 cores at 2.5 GHz and 3 GB RAM. In the following plots the curve denoted by "*with optimization*" depicts the results for the heuristic where the number of S-shapes has been maximized, while the curve denoted by "*without optimization*" depicts the results for the heuristic when applied to the representation obtained by the original ILP of Eiglsperger et al. [43]. All implementations were done in Java using the yFiles library (`http://www.yworks.com`) and the gurobi solver [64], to solve the ILPs.

Figure 6.4 shows that by maximizing the number of S-shaped edges the area requirements of the resulting drawings can be reduced. This effect is caused by the second improvement we could observe: The number of half-bends in the drawings decreases by approximately 10%, if the number of S-shaped edges is maximized, as Figure 6.5 suggests. This confirms the usefulness of this extension. Of course this positive effects can most clearly be observed in graphs with many triangular faces, since only there we can save half-bends by using the almost-empty shapes of the podevsaef model.

However, this improvements are payed for with an increased running time,

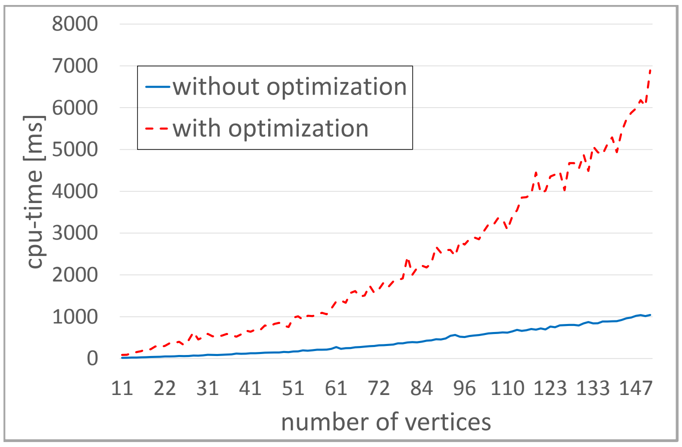


Figure 6.6: Cpu-time against number of vertices for the two variants of the heuristic.

as can be seen in the curves in Figure 6.6. But since the overall running time does not exceed 8 seconds even for the largest instances, we feel that the reduction in the number of bends and the area justifies the increased computation time.

6.4 Experimental Evaluation

In this section, we present an experimental evaluation of the podevsaef drawing model. We compared bend-optimal Kandinsky drawings obtained by implementing the original ILP of Eiglsperger [41] with bend-optimal podevsaef drawings and drawings computed by the heuristic presented in Section 6.3 with the modification that maximizes the S-shaped edges according to Section 6.3.1. All implementations were done in Java using the yFiles library (`http://www.yworks.com`) and the gurobi solver [64], to solve the different ILPs. The experiment was performed on a standard Linux machine with 4 cores at 2.5 GHz and 3 GB RAM.

As a test set, we used the *Rome graphs* (a collection of 11531 graphs with average density[1] of 0.07 obtained from `http://www.graphdrawing.org`), the

[1]Recall that the *density* of a graph is defined as the ratio of the number of its edges to the maximum possible number of edges in a simple graph with the same number of vertices.

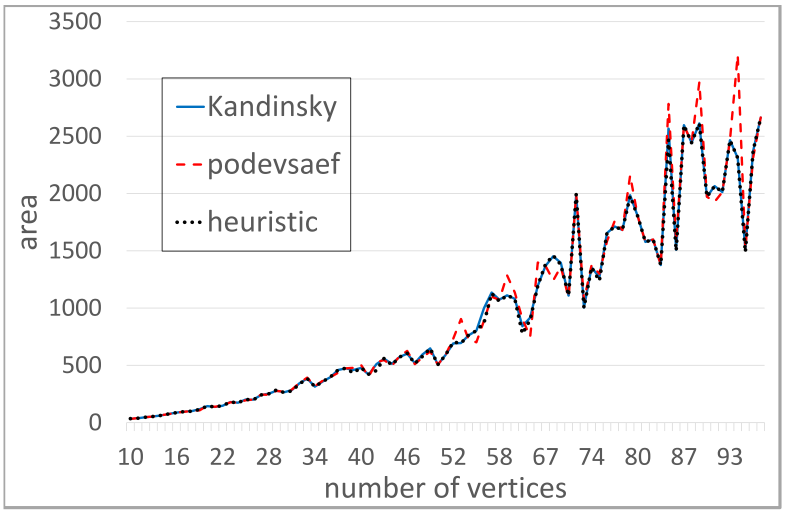


Figure 6.7: Area against number of vertices for the test set of the planar North graphs.

North graphs (a collection of 1275 graphs with average density of 0.13 also obtained from `http://www.graphdrawing.org`) and 940 randomly created (planar) triangulations with average density 0.11. Of course, we filtered both the Rome and the North graphs for planar graphs, which left 3279 Rome graphs and 854 North graphs.

From our experiment, we quickly realized that the time required to solve the ILP to compute a bend-optimal podevsaef representation increases rapidly with the number of triangular faces of the graph. So, we set a time-limit of 300 seconds in our experiment. If the solver was not able to find any (close-to-optimal) solution within this time-limit, then the instance counted as failed and was excluded from the experiment (in total we found two such faulty instances, both stemmed from the randomly created triangulations test set).

To obtain an input for our algorithms, we applied the combinatorial embedder of the yFiles graph library, which guarantees that if the input graph is planar, then the computed combinatorial embedding will be planar as well. In all following plots, the curve denoted by *Kandinsky* stands for results for orthogonal drawings, while the curves denoted by *podevsaef* and *heuristic* correspond to the results for bend-optimal and heuristically-computed podevsaef drawings, respectively. The values for a specific number of vertices were

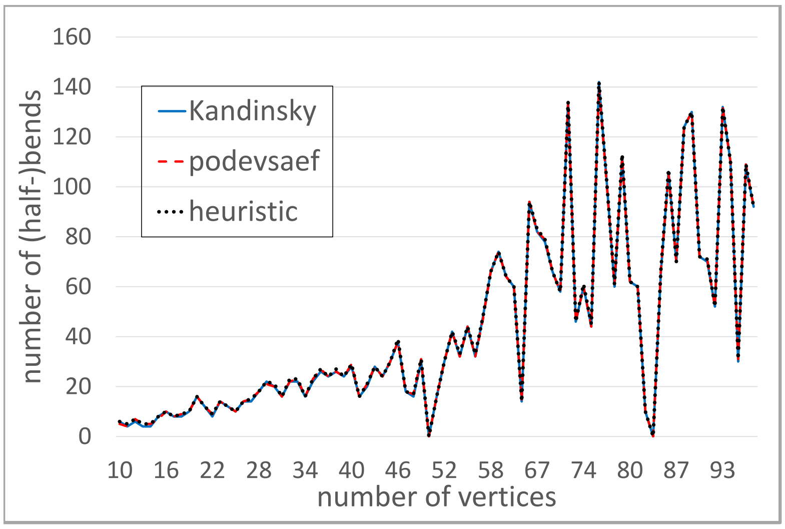


Figure 6.8: Number of (half-)bends against number of vertices for the test set of the planar North graphs.

obtained by averaging over all instances with the same number of vertices.

In Figure 6.7, the required area is plotted against the number of vertices (for the test set of the planar North graphs). It seems that the podevsaef drawings (both the bend-optimal ones and the ones created by the heuristic of Section 6.3) require comparable and sometimes even less area than the classical Kandinsky drawings for this test set. The results for the other test sets are similar; see Figures 6.9 and 6.11.

In Figure 6.8, the required number of bends is plotted against the number of vertices (again for the test set of the planar North graphs). Since a bend of a traditional orthogonal drawing is always in correspondence with a pair of half-bends of a podevsaef drawing, in Figure 6.8 we plotted two times the number of orthogonal bends against the number of half-bends produced by our algorithms. Clearly, all algorithms need very similar number of bends for all instances. The same holds for the test set of planar Rome graphs; see Figure 6.10. As expected, however, for the test set of the randomly created triangulations, the profit is bigger; see Figure 6.12. The reason is that these graphs contain only triangular faces, which explains why the podevsaef drawings require less half-bends than twice the number of bends of the classical Kandinsky drawings. In addition, it is worth mentioning that the computed

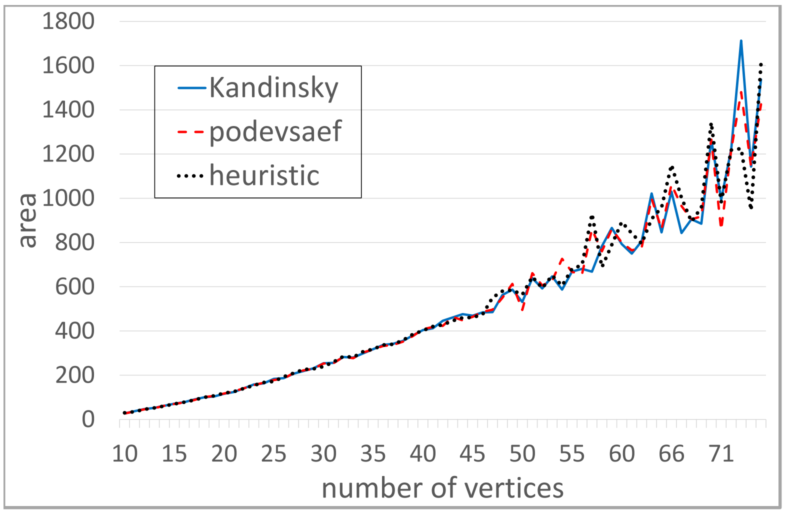


Figure 6.9: Area requirements against number of vertices for the test set of the planar Rome graphs.

number of half-bends for the test set of randomly created triangulations in most cases was not the optimal one, as the time-limit was reached.

On the negative side, the time required by the ILP to compute bend-optimal podevsaef representations increases rapidly with the number of triangular faces of the graph. This behavior was not observed in the test sets of planar Rome and planar North graphs, which are more or less sparse and with very few triangular faces. However, in the test set of the triangulations, we observed that only for graphs with at most 20 vertices we could compute an optimal drawing within the time-limit of 300 seconds. On the positive side, the solver was almost always able to compute at least a close-to-optimal solution. On the other hand, both the ILP of Eiglsperger [41] and the heuristic of Section 6.3 seem to have comparable running times.

6.5 Summary

In this chapter we introduced the podevsaef model, which can be used to draw planar graphs with arbitrary vertex-degree. By using half-bends we are able to draw shapes of faces that were forbidden in the classical Kandinsky model. By allowing these shapes we want to obtain drawings with less bends

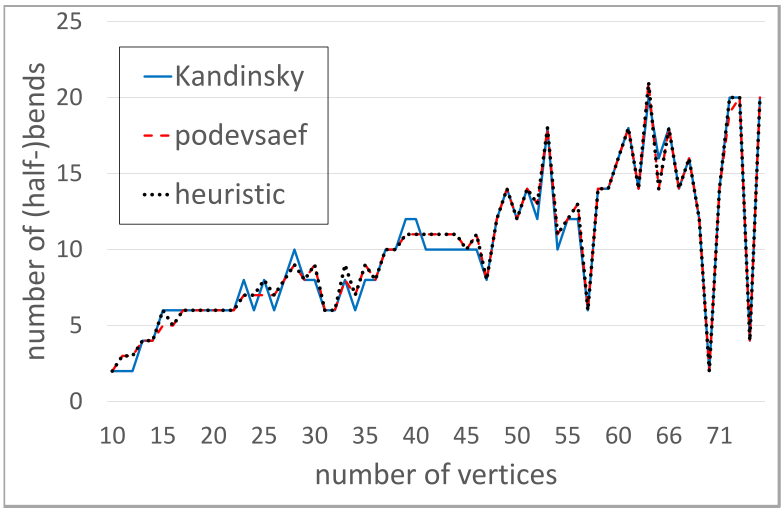


Figure 6.10: Number of (half-)bends against number of vertices for the test set of the planar Rome graphs.

and smaller area. Since the problem of minimizing the total number of bends in this model turned out to be NP-complete [20], we modeled it as an ILP. In our experimental evaluation we found that the time required to solve these ILPs increases rapidly with the number of triangular faces of the graph. In fact, for triangulations with more than 20 vertices it was not possible to compute the optimal solution within a time limit of 300 seconds. To compensate we also developed an efficient heuristic that can be used to transform a given Kandinsky representation into a representation in the podevsaef model. The experimental evaluation showed little improvement for the test-sets of the Rome and the North graphs, but for the triangulations both, the ILP and the heuristic, where able to compute drawings with smaller area and less bends than in the classical Kandinsky model.

There are two possible directions for further research:

1. Develop a different approach that will allow for faster computation of optimal (in terms of the total number of bends) podevsaef drawings, especially when the input graph is triangulated.

2. A more sophisticated heuristic or a constant-factor approximation algorithm for computing close-to-optimal podevsaef drawings would also be of interest.

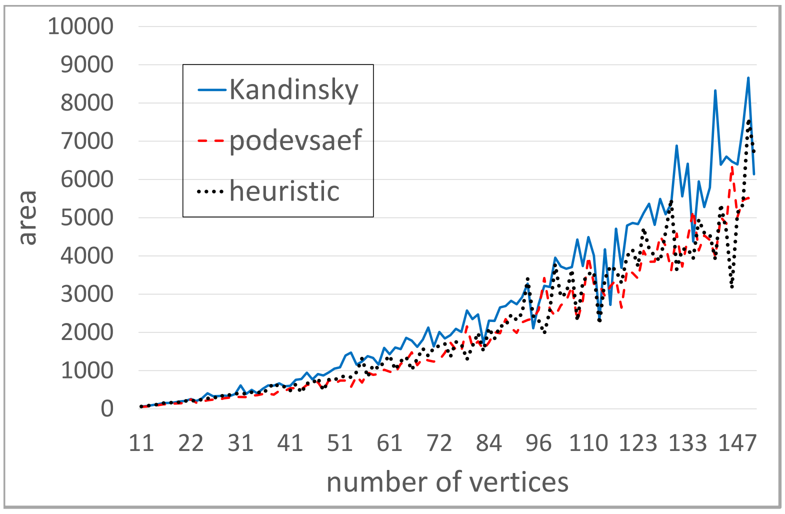


Figure 6.11: Area requirements against number of vertices for the test set of the randomly created triangulations.

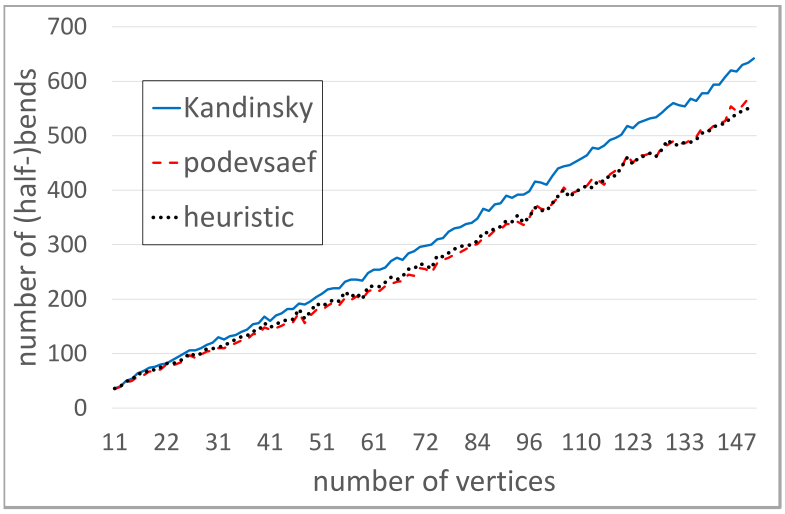


Figure 6.12: Number of (half-)bends against number of vertices for the test set of the randomly created triangulations.

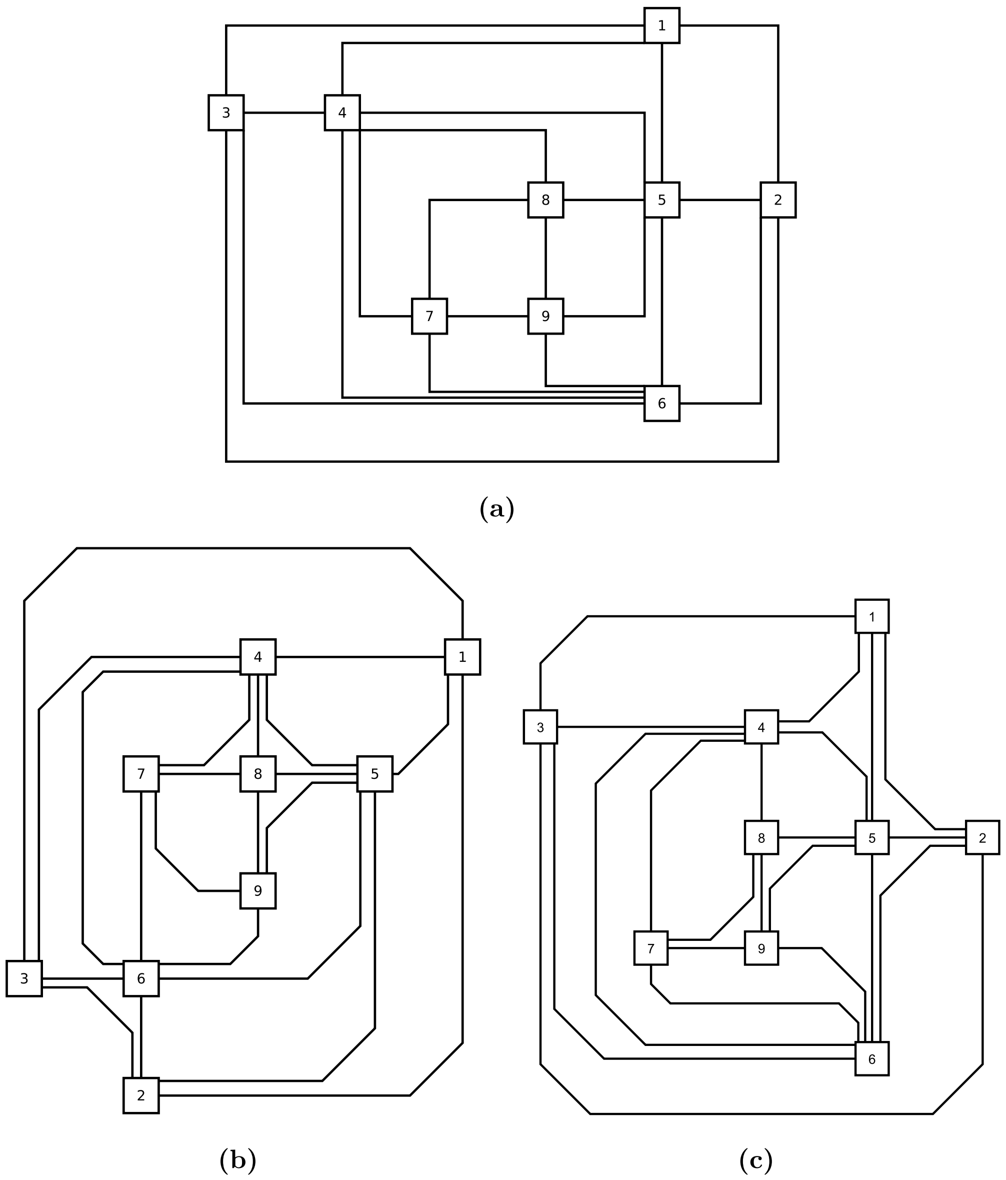


Figure 6.13: A sample planar graph drawn (a) in the classical Kandinsky model requiring 15 bends, (b) in the podevsaef drawing model with 30 half-bends and (c) with the podevsaef heuristic using 32 half-bends.

7 The Sloginsky Model

7.1 Introduction

In this chapter we extend the podevsaef model from the previous chapter to non-planar graphs. Recall that the Kandinsky model and the podevsaef model both allow arbitrary vertex degrees. In both models two grids are used, a coarse one to place the vertices and a fine one to route the edges. Vertices are drawn as boxes with non-zero side length and edges are drawn as a combination of horizontal and vertical line segments (Kandinsky model) and, additionally, diagonal segments in the podevsaef model.

An important aspect of the slog model is that crossings are only allowed between two diagonal segments, which increases the readability of the drawings since crossings become much easier to identify. In this section we will combine the slog model and the podevsaef model into the *sloginsky* model. In the new model vertices will be drawn as boxes with size greater zero, edges will be drawn using horizontal, vertical and diagonal segments and crossings will only be allowed between two diagonal segments. Additionally, the minimum angle between two consecutive segments of an edge has to be

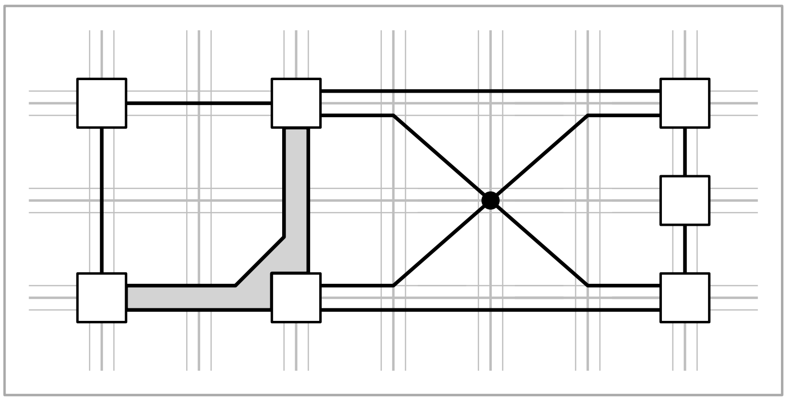


Figure 7.1: Illustration of a sloginsky drawing.

135°, resulting in half-bends, as in the previous sections. We also require that vertices are only incident to vertical or horizontal segments. Note that, for a planar graph a drawing in the sloginsky model is at the same time a podevsaef drawing. An example sloginsky drawing can be seen in Figure 7.1. As in the podevsaef model there is an almost empty face, marked with the gray shade, and, as the sloginsky model requires, the crossing is between two diagonal segments. For a larger example refer to Figure 7.8b.

As for the slog and the sloggy model, we again will adopt the TSM approach (see Section 2.4.1) to compute sloginsky drawings. Since we are mainly interested in non-planar graphs, the first phase of the TSM approach, the planarization, is required. However, we assume it has already been applied, a planar embedding has been calculated and crossings have been replaced by dummy vertices. We use the same notation as before: we denote the dummy vertices by crossing or c-vertices and the normal vertices as real or r-vertices. Also, edges between two real vertices are called rr-edges, between two crossing vertices we call them cc-edges and between a real and a crossing vertex we call them rc-edges.

We will first show how to modify the ILP formulation for the podevsaef model to be able to compute bend-optimal sloginsky representations in Section 7.2. Then we will give a method to modify the representation, such that the linear program used to draw slog representations from Section 4.5 can be used to compute sloginsky drawings in Section 7.3, if such a drawing exists. We experimentally evaluate the Sloginsky model in Section 7.4.

7.2 Bend-Optimal Sloginsky Representations

$$
\begin{array}{lllr}
\textbf{min} & \sum_{(u,v)\in E}(l_{(u,v)} + r_{(u,v)}) & & \\
\textbf{s.t.} & 0 \le a_{(u,v)} \le 4 & (u,v) \text{ is rr-edge} & (19) \\
 & 1 \le a_{(u,v)} \le 4 & (u,v) \text{ is rc- or cc-edge} & \\
 & & u \text{ is a c-vertex} & (20) \\
 & \sum_{(u,v)\in N(u)} a_{(u,v)} = 4 & \forall u \in V & (21) \\
 & \sum_{(u,v)\in f}(a_{(u,v)} + l_{(u,v)} - r_{(u,v)}) & & \\
 & \quad = \begin{cases} 4a(f) - 8; & f \text{ bounded} \\ 4a(f) + 8; & f \text{ unbounded} \end{cases} & \forall f \in F & (22) \\
 & lb^u_{(u,v)} + rb^u_{(u,v)} \le 1 & \forall (u,v) \in E & (23) \\
 & lb^u_{(u,v)} = rb^u_{(v,u)} & \forall (u,v) \in E & (24) \\
 & lb_{(u,v)} = rb_{(v,u)} & \forall (u,v) \in E & (25) \\
 & lb^v_{(u,v)} = rb^v_{(v,u)} & \forall (u,v) \in E & (26) \\
 & a_{(v,u)} + lb^v_{(v,w)} + rb^v_{(v,u)} \ge 1 & \forall (v,w),(v,u) & \\
 & & \text{subsequent in } N(v) & (27) \\
 & l_{(u,v)} + r_{(u,v)} - 2z_{(u,v)} = 0 & (u,v) \text{ is rr- or cc-edge} & (28) \\
 & l_{(u,v)} + r_{(u,v)} - 2z_{(u,v)} = 1 & (u,v) \text{ is rc-edge} & (29)
\end{array}
$$

Linear Program 4: The ILP to compute bend-optimal sloginsky representations.

The ILP-formulation to compute bend-optimal sloginsky representations is similar to the one for computing bend-optimal podevsaef representations of Section 6.2. Linear Program 4 gives the complete ILP to compute bend-optimal sloginsky representations. The main difference is that the variables that count the number of left and right bends are evaluated in terms of half-bends, that is, they correspond to multiples of 45°; see for example Constraint 22 of Linear Program 4. Furthermore, there exists an additional integer variable z_e for each edge e to model the parity of the number of half-bends on edge e. More precisely, this variable in conjunction with Constraints 28 and 29 of Linear Program 4 ensures that the number of half-bends of an rr-edge or an cc-edge is always even (Constraint 28), while the number of half-bends on an rc-edge is always odd (Constraint 29). This is enough to guarantee that r-vertices and c-vertices use orthogonal and diagonal ports, respectively, which in turn guarantees that the ILP-formulation correctly computes a bend-optimal sloginsky representation.

7.2.1 Not Realizable Representations

Unfortunately, however, there exist sloginsky representations that are not realizable in the sloginsky drawing model. One example of such a representation is given in Figure 7.2[1] (with the prescribed port, bend and angle assignment, it is not possible to draw the dashed edge in a legal way). One way to heuristically cope with this problem is to equip Linear Program 4 with extra variables and constraints to reduce the S-shaped edges, which are, in part, responsible for the infeasibility of the representation. We will describe in Section 7.2.2 how this can be done. However, this does not completely solve the problem. Note that a similar problem arises in the octilinear graph drawing model; Nöllenburg [94] presents a similar counterexample for an octilinear representation that can not be realized.

In order to determine whether the computed realization is feasible (and in the case of an affirmative answer to compute the corresponding bend-optimal drawing), we employ another LP, that was also used in the compaction phase in the slog model (see Section 4.5). Of course, we need to modify it appropriately in order to cope with vertices with more than one edge attached to the same side (and subsequently, with vertices of degree greater than four). We will describe the necessary modifications in Section 7.3.

7.2.2 Reduction of S-shapes

To reduce the number of S-shapes in the representation computed by the ILP, we use the same set of variables and constraints as in the podevsaef model to determine whether an edge has an S-shape (see Section 6.3.1). The only difference to the S-shaped edges in the podevsaef model is that in the sloginsky model an edge that has an S-shape has a left and a right half-bend, while in the podevsaef model there was a left and a right bend. Nevertheless

[1]We thank Thomas Bläsius and Ignaz Rutter who, in a personal correspondence, proposed the counterexample

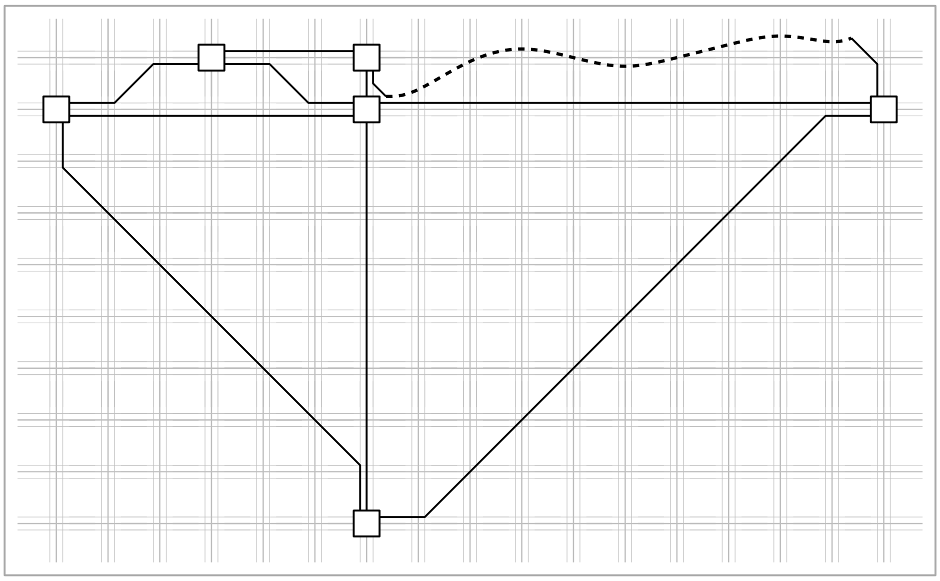


Figure 7.2: A bend-optimal representation, which is not realizable in the sloginsky drawing model.

the variables and constraints are exactly the same for the sloginsky model, so we will not repeat them here.

In order to minimize the number of edges that are drawn with an S-shape we use the following objective function:

$$\min w_b \sum_{e \in E} (r_e + l_e) + w_s \sum_{e \in E} s_e$$

We weigh the two parts of the objective function to make sure that the solution stays bend-optimal by setting the weight of a single half-bend to $w_b = |E|$ and the weight of an S-shape to $w_s = 1$.

The reduction of the S-shapes is only a heuristic step that does not guarantee that each representation can be realized. In fact we found that there are examples that can not be realized in any case, with the S-shape reduction and without it. However, several instances that could not be solved admit a drawing when the number of S-shapes is reduced. Additionally, we found that reducing the number of S-shapes in the representation often leads to drawings requiring less area. We present more detailed results in the experimental evaluation in Section 7.4.

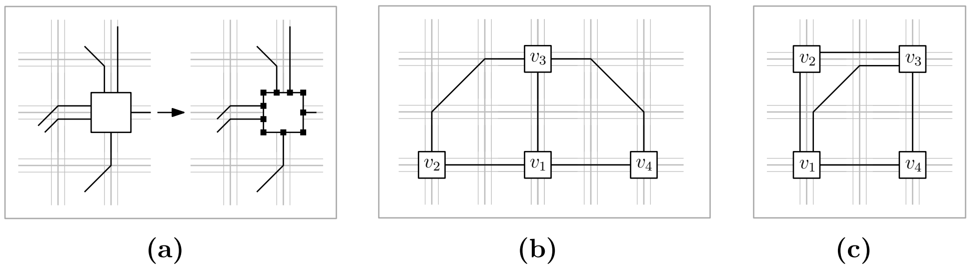


Figure 7.3: (a) Vertices with more than one edge attached to a side are split; (b) a slog drawing requiring 4 half-bends, and, (c) a sloginsky drawing of the same graph requiring 2 half-bends.

7.3 Realizing a Sloginsky Representation

We will now describe how to modify the representation obtained with the linear program from the previous section, such that the method used to realize slog representations from Section 4.5 can be used.

Each vertex that has strictly more than one edge attached to the same side (according to the computed sloginsky representation) “is split”, that is, it is removed and replaced by several new vertices as follows: four new vertices make up its corners and for each edge that is connected to it an additional vertex is added; vertices that correspond to consecutive edges of the same side of the original split-vertex are connected by an edge as in Figure 7.3a. If there is no edge connected to a side of the vertex to be split, then the two respective corners are connected directly. Additional constraints are added to the LP to make sure that the sum of the lengths of the edges of each side of the split-vertex structure equals the size of the normal vertices that have not been split.

With this modifications we are able to use the drawing LP from Section 4.5 to compute a drawing for a given sloginsky representation, if one exists. We will shortly see that sloginsky drawings may require exponential area (Section 7.3.1). On the positive side, however, a bend-optimal sloginsky drawing of a given graph requires, in general, less bends than the corresponding slog drawing (keeping the embedding unchanged); an example is given in Figure 7.3b and 7.3c.

7.3.1 Area Requirements of Sloginsky Drawings

Sloginsky drawings also can require exponential area:

Theorem 7.3.1. *There exists a graph G whose sloginsky drawing $\Gamma_s(G)$ of minimum number of half-bends requires exponential area, assuming that a planarized version $\sigma(G)$ of G is given.*

Proof. Analogous to the slanted-orthogonal case, the sloginsky drawing of the graphs constructed according to the rules from Section 4.6 require exponential area. The sloginsky drawings of these graphs have the exact same properties as the slog drawings, resulting in the exponential area requirement. For more details refer to the proof of Theorem 4.6.1. □

7.4 Experimental Evaluation

In this section, we present an experimental evaluation of the sloginsky drawing model similar to the one of Section 6.4. In particular, the experiment's setup is the same with two modifications. First, the test set we used consisted of the non-planar Rome and non-planar North graphs (as the sloginsky model is mostly appropriate for non-planar graphs). Second, we employed the Smart Organic Layouter from the yFiles graph library, which is basically a spring embedder algorithm, and as input to our algorithms we used a planarized version of its output.

In all following plots, the curves denoted by *Kandinsky* show results for orthogonal drawings (obtained by implementing the ILP of Eiglsperger [41]), while the curves denoted by *sloginsky* correspond to the results for bend-optimal sloginsky drawings.

As expected, the use of area seems to be more demanding in the sloginsky drawing model; see Figures 7.4 and 7.6. On the positive side, however, the total number of half-bends that are required in a sloginsky drawing are comparable and sometimes even less than two times the number of bends of a classical Kandinsky drawing for both test sets; see Figures 7.5 and 7.7. Finally, the running times of both algorithms tested seem to be comparable and we were able to compute all drawings within at most three seconds (without any negative behavior, as the one observed in Section 6.4).

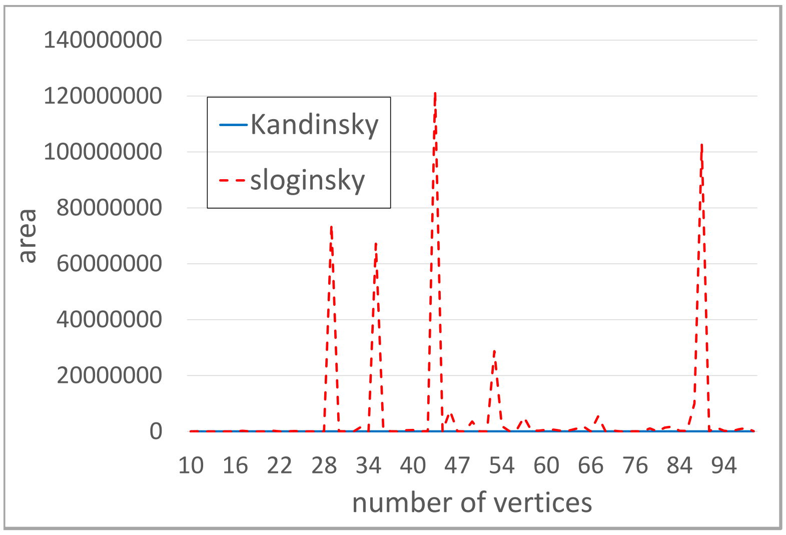


Figure 7.4: Area requirements against number of vertices for the test set of the non-planar North graphs.

Figure 7.8 shows an example drawing of a graph in the Kandinsky and the sloginsky model.

7.5 Summary

In this chapter we introduced the sloginsky model. It is appropriate for (non-planar) graphs with arbitrary vertex degrees. In the sloginsky model crossings are only allowed between diagonal segments, edges are drawn as a combination of horizontal, vertical and diagonal segments and the minimum angle between any two consecutive edge-segments has to be 135°. Together with the arbitrary vertex degree the sloginsky model unites the advantages of both, the Kandinsky and the slog model. We gave an ILP formulation to compute a bend-optimal sloginsky representation and showed how to modify the representation in order to be able to use the compaction step from the slog model to obtain an actual drawing. On the negative side we showed that there exist sloginsky representations that can not be realized and that drawings in the sloginsky model may require exponential area. In our experimental evaluation we found that sloginsky drawings in general require more area and more bends than drawings in the Kandinsky model.

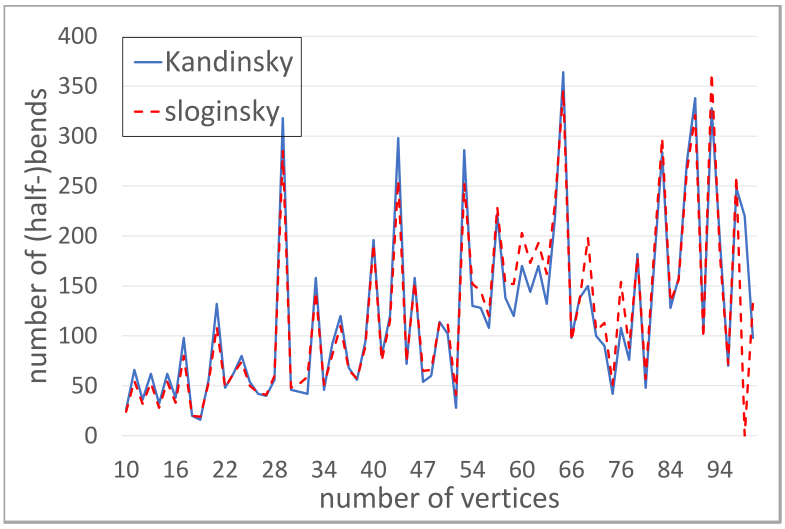


Figure 7.5: Number of (half-)bends against number of vertices for the test set of the non-planar North graphs.

There are several open problems connected to the sloginsky model:

- Is it possible to compute bend-optimal sloginsky representations that are always realizable by a corresponding drawing?
- Bend-optimal sloginsky drawings may require exponential area. Therefore, efficient heuristics that will result in sloginsky drawings with few additional bends in polynomial area are of interest.

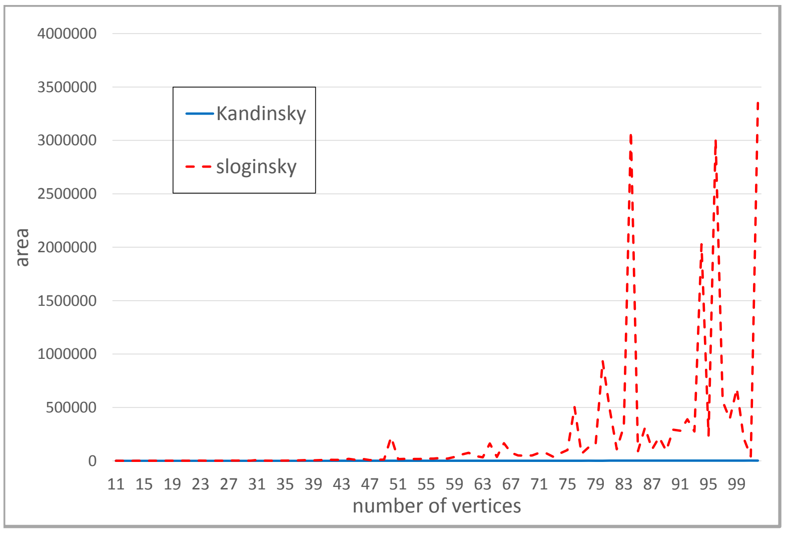


Figure 7.6: Area requirements against number of vertices for the test set of the non-planar Rome graphs.

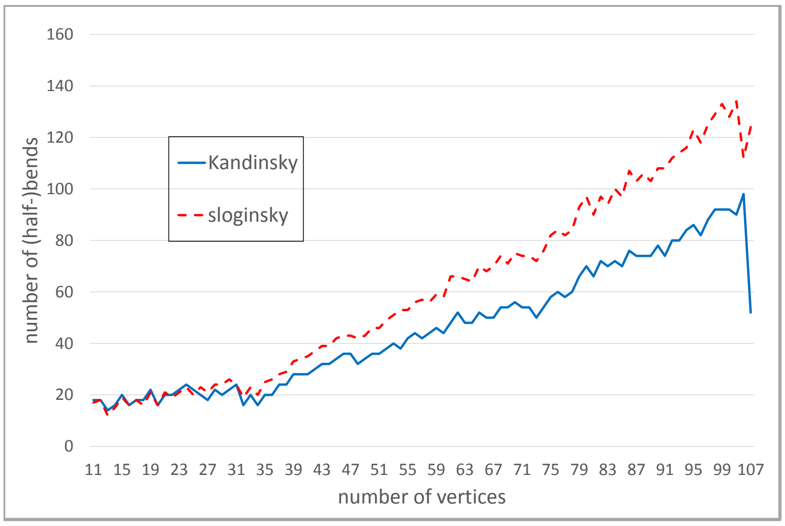


Figure 7.7: Number of (half-)bends against number of vertices for the test set of the non-planar Rome graphs.

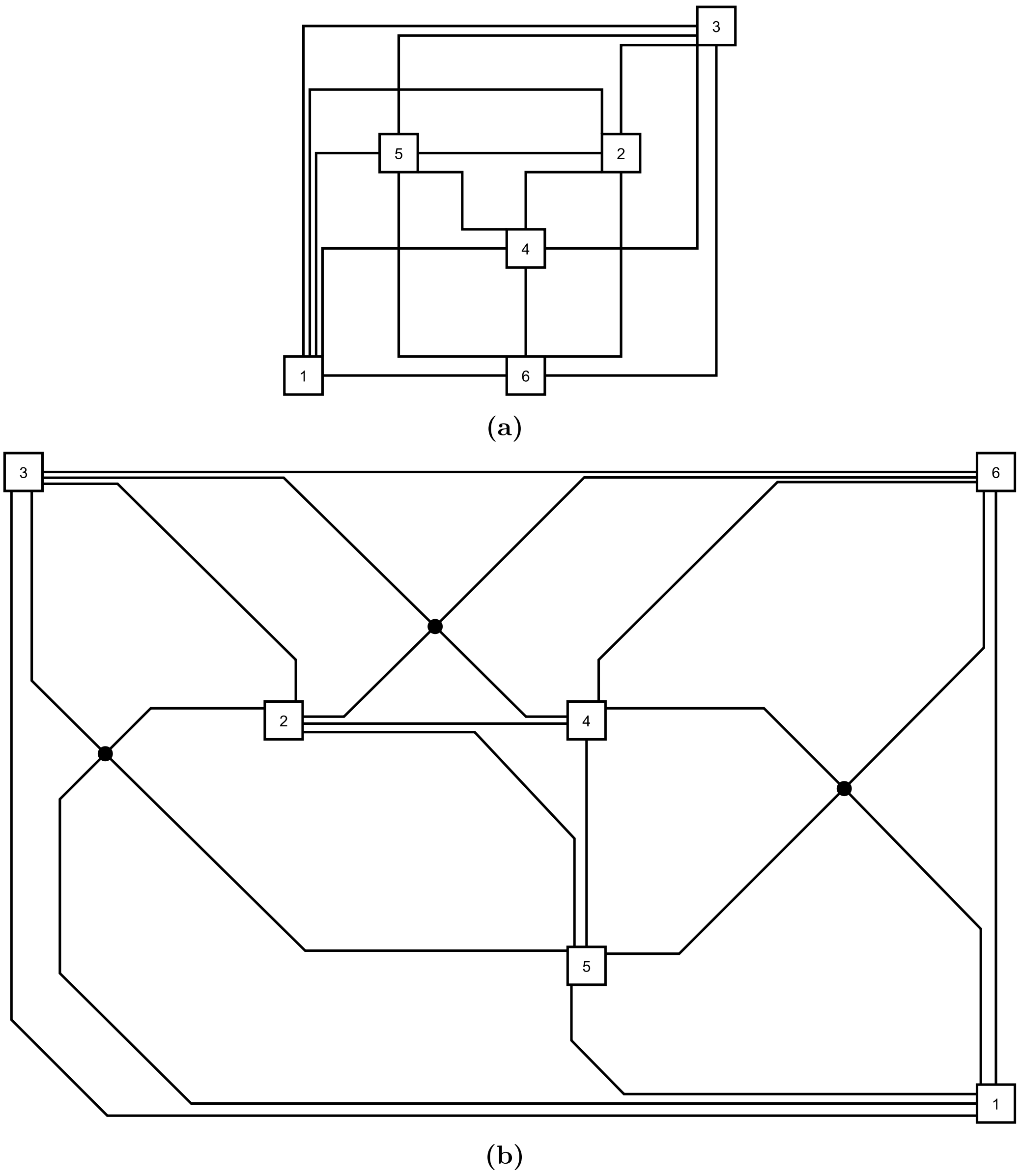


Figure 7.8: Non-planar graph K_6 drawn (a) in the classical Kandinsky drawing model, and, (b) in the sloginsky drawing model.

8 Conclusion

In this thesis we presented results for several graph drawing models, some of which were well known and studied before (octilinear graph drawing and the Kandinsky model) and others we introduced (slog, sloggy and sloginsky). Our approaches aim to optimize layout aesthetics known to be important for the perceived quality of a drawing of a graph, such as crossings and the number of bends. We achieve this by making sure that in the drawings we compute, crossings are realized in a way that makes it easy to distinguish them with respect to vertices. We further improve the readability of our drawings by using half-bends rather than traditional 90° bends, thereby smoothening the shape of the edges, and by keeping the total number of bends minimal.

In this chapter we present an overview of the results of this thesis and give several possibilities for further research based on our findings.

8.1 Results

We first studied octilinear drawings of planar graphs with one bend per edge. Previously it was known that 3-planar graphs always admit planar octilinear drawings with zero bends. It was also known that, for planar graphs with maximal vertex degree up to eight it is always possible to construct a planar octilinear drawing when two bends per edge are allowed. Our results close this gap:

- We showed that 4-planar graphs require one bend per edge to be drawn in the octilinear model and we gave algorithms to compute such drawings in cubic area in linear time.
- For 5-planar graphs we gave linear time algorithms to compute octilinear drawings with one bend per edge that may require super-polynomial area.
- We proved that for octilinear drawings of 6-planar graphs two bends are necessary for at least one edge.

We introduced an enhanced version of the classical orthogonal graph drawing model, the slanted orthogonal graph drawing, for short: slog. The minimum angle between two consecutive segments of an edge in the new model has to be 135°, which leads to a bend shape we called half-bend. This improves the readability of the drawing, since the smoother shape of edges makes it easier to follow them. Additionally, we also allow crossings in slog drawings only between two diagonal segments. By this it becomes easier to identify crossings and distinguish them from vertices. We obtained the following results for slog drawings:

- The number of half-bends in a bend-optimal slog drawing is greater than or equal to twice the number of bends in the corresponding bend-optimal orthogonal drawing.
- We developed a method, based on the well-known approach from Tamassia [110], that can be used to compute a bend-optimal slog representation using network flow.
- To realize the representation we gave a linear programming formulation that computes a drawing with minimum total edge length.
- Bend-optimal slog drawings can require exponential area.

- To counter the exponential area requirement of slog drawings we developed a heuristic that requires quadratic area but uses for some edges up to two times the number of half-bends required in the bend-optimal slog drawing.
- Using the Rome graphs we experimentally evaluated the bend optimal slog algorithm and the slog heuristic and compared them to orthogonal drawings. We found that even though we use linear programming to compute the actual drawing, the cpu time required to construct the drawing is very small, allowing to compute a bend-optimal slog drawing even for large graphs with more than 400 vertices in less than 2 seconds.

As a more flexible model we introduced the sloggy model, which basically follows the same rules as the slog model, but crossings are now also allowed between horizontal and vertical segments. This was motivated by the observation that the number of half-bends in slog drawings often is significantly more than twice the number of bends of a corresponding bend-optimal orthogonal drawing. For sloggy drawings we found that:

- The number of half-bends of a bend-optimal sloggy drawing is exactly twice the number of bends of a corresponding bend-optimal orthogonal drawing.
- To compute a bend-optimal sloggy representation we gave a formulation as linear program. With this linear program we maximize either the number of crossings on diagonals or minimize the number of pairs of half-bends on edges or a weighted combination of both.
- With minor modifications of the linear program used to realize slog representations, we can also realize sloggy representations.
- Sloggy drawings can also require exponential area; the construction for a family of graphs with this property is the same as for the slog model.

For graphs of arbitrary vertex degree we extended the classical Kandinsky model with half-bends, to be able to realize empty faces, which were previously forbidden. The new model is called podevsaef, which stands for

Planar **O**rthogonal **D**rawings with **E**qual **V**ertex **S**ize and **A**lmost **E**mpty **F**aces. We presented the following results:

- Since the NP-hardness of the bend-minimization problem for the Kandinsky model was recently shown, and it could be immediately transferred to the podevsaef model, we developed an ILP formulation to compute bend-optimal representations using our extended model. To realize them we showed a set of transformations that allowed us to use existing compaction algorithms for the Kandinsky model.

- The time required to solve the integer linear programs for the bend-optimal podevsaef representations can become quite large when many triangular faces are present. That is why we also developed a heuristic that can efficiently transform a classical Kandinsky drawing into a podevsaef drawing with few additional half-bends.

- An experimental evaluation of the podevsaef model showed that the new model is able to compete with classical Kandinsky drawings in terms of bends and area for the sparse graphs of the Rome library. For graphs that contain more triangular faces the podevsaef model often leads to drawings requiring less bends and smaller area.

To transfer the advantages of the slog model to graphs of arbitrary vertex degree, we developed the sloginsky model. In sloginsky drawings crossings are only allowed on diagonal segments, the minimum angle between consecutive segments of an edge is again 135° and arbitrary many edges can be connected to one side of a vertex. For the sloginsky model we obtained the following results:

- Inspired by the ILP for the podevsaef model, we developed an integer linear programming formulation to compute bend-optimal sloginsky representations.

- On the negative side, there exist sloginsky representations that can not be realized. Additionally, sloginsky drawings may require exponential area.

- To realize a sloginsky representation, if possible, we show how to modify the representation so that the linear program used for slog drawings can be used.

8.2 Future Work

There are several possibilities for future research on the different topics studied in this thesis, some of which we present here as open problems:

- Is it possible to construct planar octilinear drawings of 4-planar (5-planar, respectively) graphs with at most one bend per edge in $o(n^3)$ (polynomial, respectively) area?
- Does any triangle-free 6-planar graph admit a planar octilinear drawing with at most one bend per edge?
- What is the complexity to determine whether a 6-planar graph admits a planar octilinear drawing with at most one bend per edge?
- What is the number of necessary slopes for bendless drawings of 4-planar graphs?
- Is it possible to extend the algorithms we presented to compute octilinear drawings for 4- and 5-planar graphs to non-planar graphs using, for example, the Mondschein sequence [88]?
- Is there a (bend-optimal) slog drawing for every graph with maximal vertex degree four?
- Given a (bend-optimal) slog representation, is there a polynomial time algorithm that can compute a drawing realizing this representation?
- Is there a polynomial time algorithm that computes an optimal sloggy representation?
- If no such algorithm exists, what is the complexity to determine the optimal sloggy representation?

- Can the linear program used to compute bend-optimal sloggy representations for graphs with maximal vertex degree four be extended to graphs with arbitrary vertex degree?
- Given a (bend-optimal) sloggy representation, is there a polynomial time algorithm that can compute a drawing realizing this representation?
- Can the exponential area requirement for sloggy drawings be avoided if one does not insist on the maximum number of diagonal crossings?
- Investigate the effect of different combinations of weight factors in the objective function of the ILP for bend-optimal sloggy representations.
- Develop a method to compute bend-optimal podevsaef drawings more efficiently than with our ILP formulation (especially for graphs with many triangular faces).
- Develop a more sophisticated heuristic or a constant-factor approximation algorithm for computing close-to-optimal podevsaef drawings.
- Is it possible to compute bend-optimal sloginsky representations that are always realizable by a corresponding drawing?
- Bend-optimal sloginsky drawings may require exponential area. Therefore, efficient heuristics that will result in sloginsky drawings with few additional bends in polynomial area are of interest.

Publications of the Author

[1] M. A. Bekos, M. Gronemann, M. Kaufmann, and R. Krug. Planar Octilinear Drawings with One Bend Per Edge. In C. A. Duncan and A. Symvonis, editors, *Graph Drawing*, volume 8871 of *Lecture Notes in Computer Science*, pages 331–342. Springer, 2014.

[2] M. A. Bekos, M. Gronemann, M. Kaufmann, and R. Krug. Planar Octilinear Drawings with One Bend Per Edge. *arXiv preprint arXiv:1408.5920*, 2014.

[3] M. A. Bekos, M. Kaufmann, and R. Krug. Sloggy Drawings of Graphs. In *IISA 2014, The 5th International Conference on Information, Intelligence, Systems and Applications*, pages 82–87. IEEE, July 2014.

[4] M. A. Bekos, M. Kaufmann, R. Krug, T. Ludwig, S. Näher, and V. Roselli. Slanted Orthogonal Drawings: Model, Algorithms and Evaluations. *Journal of Graph Algorithms and Applications*, 18(3):459–489, 2014.

[5] M. A. Bekos, M. Kaufmann, R. Krug, S. Näher, and V. Roselli. Slanted Orthogonal Drawings. In S. Wismath and A. Wolff, editors, *Graph Drawing*, volume 8242 of *Lecture Notes in Computer Science*, pages 424–435. Springer, 2013.

Bibliography

[6] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice hall, 1993.

[7] M. J. Alam, M. A. Bekos, M. Kaufmann, P. Kindermann, S. G. Kobourov, and A. Wolff. Smooth Orthogonal Drawings of Planar Graphs. In A. Pardo and A. Viola, editors, *LATIN 2014: Theoretical Informatics - 11th Latin American Symposium*, volume 8392, pages 144–155. Springer, 2014.

[8] E. Anderheggen and H. Knöpfel. Finite Element Limit Analysis Using Linear Programming. *International Journal of Solids and Structures*, 8(12):1413–1431, 1972.

[9] Y. P. Aneja. An Integer Linear Programming Approach to the Steiner Problem in Graphs. *Networks*, 10(2):167–178, 1980.

[10] P. Angelini, L. Cittadini, G. Di Battista, W. Didimo, F. Frati, M. Kaufmann, and A. Symvonis. On the Perspectives Opened by Right Angle Crossing Drawings. In D. Eppstein and E. Gansner, editors, *Graph Drawing*, volume 5849 of *Lecture Notes in Computer Science*, pages 21–32. Springer, 2010.

[11] E. N. Argyriou, M. A. Bekos, M. Kaufmann, and A. Symvonis. On Metro-Line Crossing Minimization. *J. Graph Algorithms Appl.*, 14(1):75–96, 2010.

[12] E. N. Argyriou, M. A. Bekos, and A. Symvonis. The Straight-Line RAC Drawing Problem is NP-Hard. In I. Cerna, T. Gyimothy, J. Hromkovic, K. Jefferey, R. Kralovic, M. Vukolic, and S. Wolf, editors, *SOFSEM 2011: Theory and Practice of Computer Science*, volume 6543 of *Lecture Notes in Computer Science*, pages 74–85. Springer, 2011.

[13] M. Baur and U. Brandes. Crossing Reduction in Circular Layouts. In J. Hromkovic, M. Nagl, and B. Westfechtel, editors, *Graph-Theoretic*

Concepts in Computer Science, volume 3353 of *Lecture Notes in Computer Science*, pages 332–343. Springer Berlin Heidelberg, 2005.

[14] M. A. Bekos, M. Gronemann, S. Pupyrev, and C. N. Raftopoulou. Perfect Smooth Orthogonal Drawings. In N. G. Bourbakis, G. A. Tsihrintzis, and M. Virvou, editors, *IISA 2014, The 5th International Conference on Information, Intelligence, Systems and Applications*, pages 76–81. IEEE, 2014.

[15] M. A. Bekos, M. Kaufmann, S. G. Kobourov, and A. Symvonis. Smooth Orthogonal Layouts. In W. Didimo and M. Patrignani, editors, *Graph Drawing*, volume 7704 of *Lecture Notes in Computer Science*, pages 150–161. Springer, 2013.

[16] M. A. Bekos, M. Kaufmann, M. Nöllenburg, and A. Symvonis. Boundary Labeling with Octilinear Leaders. *Algorithmica*, 57(3):436–461, 2010.

[17] M. A. Bekos, M. Kaufmann, K. Potika, and A. Symvonis. Line Crossing Minimization on Metro Maps. In S.-H. Hong, T. Nishizeki, and W. Quan, editors, *Graph Drawing*, volume 4875 of *Lecture Notes in Computer Science*, pages 231–242. Springer, 2008.

[18] P. Bertolazzi, G. Di Battista, and W. Didimo. Computing Orthogonal Drawings with the Minimum Number of Bends. *Computers, IEEE Transactions on*, 49(8):826–840, 2000.

[19] T. C. Biedl and G. Kant. A Better Heuristic for Orthogonal Graph Drawings. In J. van Leeuwen, editor, *Proc. of 2nd European Symposium on Algorithms (ESA 1994)*, volume 855 of *Lecture Notes in Computer Science*, pages 24–35, 1994.

[20] T. Bläsius, G. Brückner, and I. Rutter. Complexity of Higher-Degree Orthogonal Graph Embedding in the Kandinsky Model. In A. S. Schulz and D. Wagner, editors, *Symposium on Algorithms*, volume 8737 of *Lecture Notes in Computer Science*, pages 161–172, 2014.

[21] U. Brandes, M. Eiglsperger, M. Kaufmann, and D. Wagner. Sketch-Driven Orthogonal Graph Drawing. In M. Goodrich and S. Kobourov, editors, *Graph Drawing*, volume 2528 of *Lecture Notes in Computer Science*, pages 1–11. Springer Berlin Heidelberg, 2002.

[22] U. Brandes and D. Wagner. Using Graph Layout to Visualize Train Interconnection Data. In S. Whitesides, editor, *Graph Drawing*, volume 1547 of *Lecture Notes in Computer Science*, pages 44–56. Springer, 1998.

[23] P. Butler. Visualizing Friendships. https://www.facebook.com/Engineering/notes, 2010.

[24] D.-S. Chen, R. G. Batson, and Y. Dang. *Applied Integer Programming: Modeling and Solution.* John Wiley & Sons, 2011.

[25] M. Chimani, P. Mutzel, and I. Bomze. A New Approach to Exact Crossing Minimization. In D. Halperin and K. Mehlhorn, editors, *Algorithms-ESA 2008*, volume 5193 of *Lecture Notes in Computer Science*, pages 284–296. Springer, 2008.

[26] C. D. Correa and K.-L. Ma. Visualizing Social Networks. In C. C. Aggarwal, editor, *Social Network Data Analytics*, pages 307–326. Springer, 2011.

[27] T. J. Cova and J. P. Johnson. A Network Flow Model for Lane-Based Evacuation Routing. *Transportation Research Part A: Policy and Practice*, 37(7):579–604, 2003.

[28] G. B. Dantzig. *Linear Programming and Extensions.* Princeton university press, 1998.

[29] H. De Fraysseix, J. Pach, and R. Pollack. How to Draw a Planar Graph on a Grid. *Combinatorica*, 10(1):41–51, 1990.

[30] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice Hall, 1999.

[31] G. Di Battista and R. Tamassia. On-Line Graph Algorithms with SPQR-Trees. In M. Paterson, editor, *Automata, Languages and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 598–611, 1990.

[32] E. Di Giacomo, G. Liotta, and F. Montecchiani. The Planar Slope Number of Subcubic Graphs. In A. Pardo and A. Viola, editors, *LATIN 2014: Theoretical Informatics*, volume 8392 of *Lecture Notes in Computer Science*, pages 132–143. Springer, 2014.

[33] W. Didimo, P. Eades, and G. Liotta. Drawing Graphs with Right Angle Crossings. In F. Dehne, M. Gavrilova, J.-R. Sack, and C. Toth, editors, *Algorithms and Data Structures*, volume 5664 of *Lecture Notes in Computer Science*, pages 206–217. Springer, 2009.

[34] W. Didimo, P. Eades, and G. Liotta. A Characterization of Complete Bipartite RAC Graphs. *Information Processing Letters*, 110(16):687–691, 2010.

[35] W. Didimo and G. Liotta. Computing Orthogonal Drawings in a Variable Embedding Setting. In K.-Y. Chwa and O. Ibarra, editors, *Algorithms and Computation*, volume 1533 of *Lecture Notes in Computer Science*, pages 80–89. Springer, 1998.

[36] W. Didimo and G. Liotta. The Crossing-Angle Resolution in Graph Drawing. In J. Pach, editor, *Thirty Essays on Geometric Graph Theory*, pages 167–184. Springer, 2013.

[37] T. Dwyer, K. Marriott, and P. J. Stuckey. Fast Node Overlap Removal. In P. Healy and N. Nikolov, editors, *Graph Drawing*, volume 3843 of *Lecture Notes in Computer Science*, pages 153–164. Springer, 2006.

[38] P. Eades and X. Lin. Spring Algorithms and Symmetry. *Theoretical Computer Science*, 240(2):379–405, 2000.

[39] P. Eades and G. Liotta. Right Angle Crossing Graphs and 1-Planarity. In M. van Kreveld and B. Speckmann, editors, *Graph Drawing*, volume

7034 of *Lecture Notes in Computer Science*, pages 148–153. Springer, 2012.

[40] P. Eades and N. Wormald. Edge Crossings in Drawings of Bipartite Graphs. *Algorithmica*, 11(4):379–403, 1994.

[41] M. Eiglsperger. *Automatic Layout of UML Class Diagrams: A Topology-Shape-Metrics Approach.* PhD thesis, Universität Tübingen, 2003.

[42] M. Eiglsperger, S. P. Fekete, and G. W. Klau. Orthogonal Graph Drawing. In M. Kaufmann and D. Wagner, editors, *Drawing graphs*, volume 2025 of *Lecture Notes in Computer Science*, pages 121–171. Springer, 2001.

[43] M. Eiglsperger, U. Fößmeier, and M. Kaufmann. Orthogonal Graph Drawing with Constraints. In D. B. Shmoys, editor, *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 3–11. ACM/SIAM, 2000.

[44] M. Eiglsperger and M. Kaufmann. Fast Compaction for Orthogonal Drawings with Vertices of Prescribed Size. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Graph Drawing*, volume 2265 of *Lecture Notes in Computer Science*, pages 124–138. Springer Berlin Heidelberg, 2002.

[45] O. Ersoy, C. Hurter, F. V. Paulovich, G. Cantareiro, and A. Telea. Skeleton-Based Edge Bundling for Graph Visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2364–2373, 2011.

[46] G. Even, S. Guha, and B. Schieber. Improved Approximations of Crossings in Graph Drawings and VLSI Layout Areas. *SIAM Journal on Computing*, 32(1):231–252, 2002.

[47] S. Even and R. E. Tarjan. Network Flow and Testing Graph Connectivity. *SIAM journal on computing*, 4(4):507–518, 1975.

[48] A. Farag, S. Al-Baiyat, and T. Cheng. Economic Load Dispatch Multiobjective Optimization Procedures Using Linear Programming Techniques. *Power Systems, IEEE Transactions on*, 10(2):731–738, 1995.

[49] I. Fáry. On Straight Lines Representation of Plane Graphs. *Acta. Sci. Math. Szeged*, 11:229–233, 1948.

[50] J. Feldman, M. J. Wainwright, and D. R. Karger. Using Linear Programming to Decode Binary Linear Codes. *Information Theory, IEEE Transactions on*, 51(3):954–972, 2005.

[51] M. Fink, H. Haverkort, M. Nöllenburg, M. Roberts, J. Schuhmann, and A. Wolff. Drawing Metro Maps Using Bézier Curves. In W. Didimo and M. Patrignani, editors, *Graph Drawing*, volume 7704 of *Lecture Notes in Computer Science*, pages 463–474. Springer, 2013.

[52] B. Finkel and R. Tamassia. Curvilinear Graph Drawing Using the Force-Directed Method. In J. Pach, editor, *Graph Drawing*, volume 3383 of *Lecture Notes in Computer Science*, pages 448–453. Springer, 2005.

[53] U. Fößmeier, C. Heß, and M. Kaufmann. On Improving Orthogonal Drawings: The 4M-Algorithm. In S. Whitesides, editor, *Graph Drawing*, volume 1547 of *Lecture Notes in Computer Science*, pages 125–137, 1998.

[54] U. Fößmeier and M. Kaufmann. Drawing High Degree Graphs with Low Bend Numbers. In F. J. Brandenburg, editor, *Graph Drawing*, volume 1027 of *Lecture Notes in Computer Science*, pages 254–266. Springer, 1996.

[55] U. Fößmeier and M. Kaufmann. Algorithms and Area Bounds for Nonplanar Orthogonal Drawings. In G. Di Battista, editor, *Graph Drawing*, volume 1353 of *Lecture Notes in Computer Science*, pages 134–145. Springer Berlin Heidelberg, 1997.

[56] L. C. Freeman. Visualizing Social Networks. *Journal of Social Structure*, 1(1):4, 2000.

[57] D. R. Fulkerson. A Network Flow Computation for Project Cost Curves. *Management science*, 7(2):167–178, 1961.

[58] E. R. Gansner, Y. Hu, S. North, and C. Scheidegger. Multilevel Agglomerative Edge Bundling for Visualizing Large Graphs. In *Pacific Visualization Symposium (PacificVis), 2011 IEEE*, pages 187–194. IEEE, 2011.

[59] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* Freeman & Co., San Francisco, 1979.

[60] M. R. Garey and D. S. Johnson. Crossing Number is NP-complete. *SIAM Journal on Algebraic Discrete Methods*, 4(3):312–316, 1983.

[61] A. Garg and R. Tamassia. On the Computational Complexity of Upward and Rectilinear Planarity Testing. *SIAM Journal on Computing*, 31(2):601–625, 2001.

[62] L. L. Garver. Transmission Network Estimation Using Linear Programming. *Power Apparatus and Systems, IEEE Transactions on*, PAS-89(7):1688–1697, 1970.

[63] F. Glover. Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers & Operations Research*, 13(5):533–549, 1986.

[64] I. Gurobi Optimization. Gurobi optimizer reference manual, 2014.

[65] C. Gutwenger and P. Mutzel. A Linear Time Implementation of SPQR-Trees. In J. Marks, editor, *Graph Drawing*, volume 1984 of *Lecture Notes in Computer Science*, pages 77–90. Springer, 2001.

[66] C. Gutwenger and P. Mutzel. An Experimental Study of Crossing Minimization Heuristics. In G. Liotta, editor, *Graph Drawing*, volume 2912 of *Lecture Notes in Computer Science*, pages 13–24. Springer, 2004.

[67] J. Heer and D. Boyd. Vizster: Visualizing Online Social Networks. In *Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on*, pages 32–39. IEEE, 2005.

[68] D. Holten. Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):741–748, 2006.

[69] D. Holten and J. J. Van Wijk. Force-Directed Edge Bundling for Graph Visualization. In *Computer Graphics Forum*, volume 28, pages 983–990. Wiley Online Library, 2009.

[70] S.-H. Hong. Drawing Graphs Symmetrically in Three Dimensions. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Graph Drawing*, volume 2265 of *Lecture Notes in Computer Science*, pages 189–204. Springer, 2002.

[71] S.-H. Hong, B. McKay, and P. Eades. Symmetric Drawings of Triconnected Planar Graphs. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 356–365. Society for Industrial and Applied Mathematics, 2002.

[72] S.-H. Hong, D. Merrick, and H. A. D. do Nascimento. Automatic Visualisation of Metro Maps. *Journal of Visual Languages and Computing*, 17(3):203–224, 2006.

[73] V. Jelínek, E. Jelínková, J. Kratochvíl, B. Lidický, M. Tesar, and T. Vyskocil. The Planar Slope Number of Planar Partial 3-Trees of Bounded Degree. *Graphs and Combin.*, 29(4):981–1005, 2013.

[74] M. Jünger, E. K. Lee, P. Mutzel, and T. Odenthal. A Polyhedral Approach to the Multi-Layer Crossing Minimization Problem. In G. Di-Battista, editor, *Graph Drawing*, volume 1353 of *Lecture Notes in Computer Science*, pages 13–24. Springer, 1997.

[75] G. Kant. Drawing Planar Graphs Using the lmc-Ordering (Extended Abstract). In *FOCS*, pages 101–110, 1992.

[76] G. Kant. Hexagonal Grid Drawings. In E. Mayr, editor, *Proc. of 18th Workshop on Graph-Theoretic Concepts in Computer Science (WG 1992)*, volume 657 of *Lecture Notes in Computer Science*, pages 263–276, 1992.

[77] M. Kaufmann and D. Wagner. *Drawing Graphs: Methods and Models*, volume 2025. Springer Berlin Heidelberg, 2001.

[78] B. Keszegh, J. Pach, and D. Pálvölgyi. Drawing Planar Graphs of Bounded Degree with Few Slopes. *SIAM J. Discrete Math.*, 27(2):1171–1183, 2013.

[79] B. Keszegh, J. Pach, D. Pálvölgyi, and G. Tóth. Drawing Cubic Graphs with At Most Five Slopes. *Computational Geometry*, 40(2):138–147, 2008.

[80] G. W. Klau and P. Mutzel. Quasi-Orthogonal Drawing of Planar Graphs. *Informatik-Spektrum*, 20(4):199–207, 1997.

[81] D. Knoke and S. Yang. *Social Network Analysis*, volume 154. Sage, 2008.

[82] A. Lambert, R. Bourqui, and D. Auber. Winding Roads: Routing Edges Into Bundles. In *Computer Graphics Forum*, volume 29, pages 853–862. Wiley Online Library, 2010.

[83] C. E. Leiserson. Area-Efficient Graph Layouts. In *Proc. of 21st Annual Symposium on Foundations of Computer Science (FOCS 1980)*, volume 1547, pages 270–281, 1980.

[84] W. Lenhart, G. Liotta, D. Mondal, and R. I. Nishat. Planar and Plane Slope Number of Partial 2-Trees. In S. Wismath and A. Wolff, editors, *Graph Drawing*, volume 8242 of *Lecture Notes in Computer Science*, pages 412–423, 2013.

[85] J. D. Little. The Synchronization of Traffic Signals by Mixed-Integer Linear Programming. *Operations Research*, 14(4):568–594, 1966.

[86] Y. Liu, A. Morgana, and B. Simeone. A Linear Algorithm for 2-Bend Embeddings of Planar Graphs in the Two-Dimensional Grid. *Discrete Applied Mathematics*, 81(1-3):69–91, 1998.

[87] D. Merrick and J. Gudmundsson. Path Simplification for Metro Map Layout. In M. Kaufmann and D. Wagner, editors, *Graph Drawing*, volume 4372 of *Lecture Notes in Computer Science*, pages 258–269. Springer, 2007.

[88] L. F. Mondshein. *Combinatorial Ordering and the Geometric Embedding of Graphs*. PhD thesis, MIT, 1971.

[89] P. Mukkamala and D. Pálvölgyi. Drawing Cubic Graphs with the Four Basic Slopes. In M. van Kreveld and B. Speckmann, editors, *Graph Drawing*, volume 7034 of *Lecture Notes in Computer Science*, pages 254–265, 2012.

[90] M. Müller-Hannemann and A. Schulze. Approximation of Octilinear Steiner Trees Constrained By Hard and Soft Obstacles. In L. Arge and R. Freivalds, editors, *Algorithm Theory–SWAT 2006*, volume 4059 of *Lecture Notes in Computer Science*, pages 242–254. Springer, 2006.

[91] M. Müller-Hannemann and A. Schulze. Hardness and Approximation of Octilinear Steiner Trees. *International Journal of Computational Geometry & Applications*, 17(03):231–260, 2007.

[92] P. Mutzel and T. Ziegler. The Constrained Crossing Minimization Problem. In J. Kratochvil, editor, *Graph Drawing*, volume 1731 of *Lecture Notes in Computer Science*, pages 175–185. Springer, 1999.

[93] T. Nishizeki and M. S. Rahman. Planar Graph Drawing. *AMC*, 10:12, 2004.

[94] M. Nöllenburg. Automated Drawings of Metro Maps. Technical Report 2005-25, Fakultät für Informatik, Universität Karlsruhe, 2005.

[95] M. Nöllenburg. An Improved Algorithm for the Metro-Line Crossing Minimization Problem. In D. Eppstein and E. Gansner, editors, *Graph*

Drawing, volume 5849 of *Lecture Notes in Computer Science*, pages 381–392. Springer, 2010.

[96] M. Nöllenburg and A. Wolff. Drawing and Labeling High-Quality Metro Maps by Mixed-Integer Programming. *IEEE Trans. on Vis. and Comp. Graphics*, 17(5):626–641, 2011.

[97] A. Papakostas and I. G. Tollis. A Pairing Technique for Area-Efficient Orthogonal Drawings. In S. North, editor, *Graph Drawing*, volume 1190 of *Lecture Notes in Computer Science*, pages 355–370. Springer, 1997.

[98] A. Papakostas and I. G. Tollis. Algorithms for Area-Efficient Orthogonal Drawings. *Computational Geometry*, 9(1):83–110, 1998.

[99] S. Pupyrev, L. Nachmanson, and M. Kaufmann. Improving Layered Graph Layouts with Edge Bundling. In U. Brandes and S. Cornelsen, editors, *Graph Drawing*, volume 6502 of *Lecture Notes in Computer Science*, pages 329–340. Springer, 2011.

[100] H. Purchase. Which Aesthetic Has The Greatest Effect On Human Understanding? In G. DiBattista, editor, *Graph Drawing*, volume 1353 of *Lecture Notes in Computer Science*, pages 248–261. Springer, 1997.

[101] H. C. Purchase. Effective Information Visualisation: A Study of Graph Drawing Aesthetics and Algorithms. *Interacting with Computers*, 13(2):147–162, 2000.

[102] H. C. Purchase. Metrics For Graph Drawing Aesthetics. *Journal of Visual Languages & Computing*, 13(5):501–516, 2002.

[103] H. C. Purchase, R. F. Cohen, and M. James. Validating Graph Drawing Aesthetics. In F. Brandenburg, editor, *Graph Drawing*, volume 1027 of *Lecture Notes in Computer Science*, pages 435–446. Springer, 1996.

[104] A. Richards, T. Schouwenaars, J. P. How, and E. Feron. Spacecraft Trajectory Planning With Avoidance Constraints Using Mixed-Integer

Linear Programming. *Journal of Guidance, Control, and Dynamics*, 25(4):755–764, 2002.

[105] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, 1998.

[106] S. K. Stein. Convex Maps. *Proceedings of the American Mathematical Society*, 2(3):464–466, 1951.

[107] E. Steinitz and H. Rademacher. *Vorlesungen über die Theorie der Polyeder unter Einschluss der Elemente der Topologie.* Springer-Verlag, Berlin-New York, 1976. Reprint der 1934 Auflage, Grundlehren der Mathematischen Wissenschaften, No. 41.

[108] H. S. Stone. Multiprocessor Scheduling With the Aid of Network Flow Algorithms. *Software Engineering, IEEE Transactions on*, (1):85–93, 1977.

[109] J. M. Stott, P. Rodgers, J. C. Martinez-Ovando, and S. G. Walker. Automatic Metro Map Layout Using Multicriteria Optimization. *IEEE Trans. on Vis. and Comp. Graphics*, 17(1):101–114, 2011.

[110] R. Tamassia. On Embedding a Graph in the Grid with the Minimum Number of Bends. *SIAM Journal of Computing*, 16(3):421–444, 1987.

[111] R. Tamassia. *Handbook of Graph Drawing and Visualization*, volume 10. Chapman & Hall/CRC, 2013.

[112] R. Tamassia, G. Di Battista, and C. Batini. Automatic Graph Drawing and Readability of Diagrams. *Systems, Man and Cybernetics, IEEE Transactions on*, 18(1):61–79, 1988.

[113] R. Tamassia and I. G. Tollis. Planar Grid Embedding In Linear Time. *IEEE Transactions on Circuits and Systems*, 36(9):1230–1234, 1989.

[114] L. G. Valiant. Universality Considerations in VLSI Circuits. *IEEE Transaction on Computers*, 30(2):135–140, 1981.

[115] M. Van Kreveld. The Quality Ratio of RAC Drawings and Planar Drawings of Planar Graphs. In U. Brandes and S. Cornelsen, editors, *Graph Drawing*, volume 6502 of *Lecture Notes in Computer Science*, pages 371–376. Springer, 2011.

[116] R. Villasana, L. Garver, and S. Salon. Transmission Network Planning Using Linear Programming. *Power Apparatus and Systems, IEEE Transactions on*, (2):349–356, 1985.

[117] H. M. Wagner. An Integer Linear-Programming Model for Machine Scheduling. *Naval Research Logistics Quarterly*, 6(2):131–140, 1959.

[118] K. Wagner. Bemerkungen zum Vierfarbenproblem. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 46:26–32, 1936.

[119] H. Xiang, X. Tang, and M. D. Wong. Min-Cost Flow-Based Algorithm for Simultaneous Pin Assignment and Routing. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 22(7):870–878, 2003.

[120] J. Xu and J. P. Kelly. A Network Flow-Based Tabu Search Heuristic for the Vehicle Routing Problem. *Transportation Science*, 30(4):379–393, 1996.

[121] R. Yang and C. Chuang. Optimal Topology Design Using Linear Programming. *Computers & Structures*, 52(2):265–275, 1994.