Master Thesis

1-Planar RAC Drawings with Bends

Johannes Zink

Date of Submission:	December 1, 2017
Advisors:	Prof. Dr. Alexander Wolff
	Dr. Steven Chaplick
	Fabian Lipp, M. Sc.

Julius-Maximilians-Universität Würzburg
Lehrstuhl für Informatik I
Algorithmen, Komplexität und wissensbasierte Systeme

Abstract

This thesis concerns the relationships among *beyond-planar graphs*, which generalize the planar graphs. In particular, it is about *RAC drawings* of *1-planar graphs* and *NIC-planar graphs* in bounded area and with a bounded number of bends per edge. A *drawing* of a graph is a mapping of each vertex of the graph to a point on the plane and of each edge of the graph to a curve on the plane. In a *RAC drawing*, edges cross only in right angles. A *1-planar graph* can be drawn such that each edge is crossed at most once and for a *NIC-planar graph* we require additionally that two crossings have at most one common vertex.

We contribute three new results, which all apply to the fixed embedding setting. First, we show that every NIC-planar graph admits a RAC drawing with at most one bend per edge on a grid of quadratic size. Second, we show that every 1-planar graph admits a RAC drawing with at most one bend per edge, which was previously known in the variable embedding setting. Third, we show that every 1-planar graph admits a RAC drawing with at most two bends per edge on a grid of polynomial size. Our proofs are constructive and built on prior work regarding straight-line drawings of planar graphs, in particular, the Shift Algorithm. All introduced algorithms run in linear time.

Zusammenfassung

In dieser Masterarbeit befassen wir uns damit, wie sich bestimmte Graphklassen, die die planaren Graphen verallgemeinern und auch als *beyond-planar graphs* bekannt sind, zueinander verhalten. Im Speziellen geht es um *RAC-Zeichnung* von *1-planaren Graphen* und *NIC-planaren Graphen* auf einer asymptotisch begrenzten Zeichenfläche und mit einer begrenzten Anzahl an Knicken pro Kante. Eine *Zeichnung* eines Graphen ist eine Abbildung jedes Knoten des Graphen auf einen Punkt in der Ebene und jeder Kante des Graphen auf eine Kurve in der Ebene. In einer *RAC-Zeichnung* kreuzen sich Kanten nur in rechten Winkeln. Ein *1-planarer Graph* ist ein Graph, der so gezeichnet werden kann, dass jede Kante höchstens einmal gekreuzt wird. Bei einem *NIC-planaren Graphen* fordern wir zusätzlich, dass zwei Kreuzungen höchstens einen gemeinsamen Knoten teilen.

Wir stellen drei neue Resultate vor. Erstens zeigen wir, dass jeder NIC-planare Graph in jeder NIC-planaren Einbettung eine RAC-Zeichnung mit höchstens einem Knick pro Kante auf einem quadratisch großen Gitter besitzt. Zweitens zeigen wir, dass nicht nur jeder 1-planare Graph, wie zuvor bekannt, eine RAC-Zeichnung mit höchstens einem

Knick pro Kante besitzt, sondern auch jeder 1-planar eingebette Graph. Drittens zeigen wir, dass jeder 1-planare Graph in jeder 1-planaren Einbettung eine RAC-Zeichnung mit höchstens zwei Knicken pro Kante auf einem polynomiell großen Gitter besitzt. Unsere Beweise sind konstruktiv und bauen auf bekannten Verfahren zum geradlinigen Zeichnen planarer Graphen auf, insbesondere dem Shift-Algorithmus. Alle hierbei vorgestellten Algorithmen sind Linearzeitalgorithmen.

Contents

1. Introduction

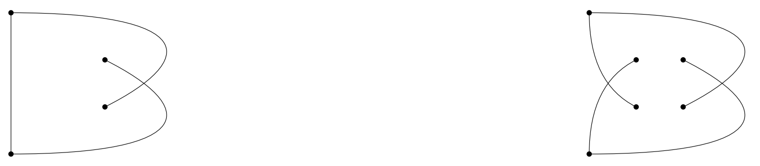
Graphs appear in many domains and applications in science and beyond. Whenever we encounter relations of objects, we may model them as a graph—may it be a social networks, Internet traffic or a molecule in chemistry. Still, such a graph is an abstract definition of relations as vertices and edges. A human viewer usually wants to see the structure of a graph quickly and unambiguously. This can be done with a proper, aesthetically nice drawing of the graph. According to some researches, human viewers prefer drawings with few crossings [Pur00, PCA02, WPCM02], no or few bends per edge [Pur97], and large crossing angles if edges cross [Hua07, HEH14, HHE08]. Furthermore, we aim for a good drawing resolution, i.e., no pair of vertices should be placed too close to each other. We can guarantee some bounded distance by placing every vertex onto a grid point of a regular integer grid of a specified size [HvKKR14]. We focus on these criteria in this thesis. In the case of large crossing angles, we restrict to the largest possible crossing angle, which is 90 degrees. We ignore other criteria like a large angular resolution of the edges at each vertex.

From a theoretical point of view, there are several classes of graphs that admit drawings that fulfill these criteria in the best possible way. The maybe most prominent class of graphs are the planar graphs. This is the set of all graphs that can be drawn on the plane without any crossing. In terms of bend points per edge and size of the drawing, it is known that every planar graph can be drawn on a grid of quadratic size (in the number of vertices) without crossings and with straight lines [dFPP90, Sch90].

There are attempts to generalize the class of planar graphs. In the recent years, the field of *beyond planarity* has experienced increasing interest in graph theory and graph drawing as a generalization of the planar graphs [Lio14, HKKP16]. They include more graphs than the planar graphs and, therefore, reduce the number of general graphs that have no drawing with "nice" aesthetic properties that are defined and researched in the literature. The planar graphs are relatively well studied for many years, but the beyond-planar graphs are currently investigated in more detail.

Previous work There are several different known classes of beyond-planar graphs. Among the most important, there are the *k-planar graphs*. They have a drawing where every edge crosses at most k other edges. Among them, the class of the 1-planar graphs is the probably most studied one. Kobourov et al. [KLM17] give an overview about 1-planarity in their annotated bibliography. In 1965, Ringel [Rin65] introduced the 1-planar graphs when he considered the problem of coloring vertices and faces.

There are subclasses of the 1-planar graphs that require the crossings to be some kind of isolated from each other. If each vertex is incident to at most one crossing edge in a 1-planar drawing, we call this drawing *IC-planar*, where IC stands for *independent*



(a) B-Configuration. (b) W-Configuration.

Fig. 1.1.: Embedded graph configurations that cannot be drawn with straight lines.

crossings. The graphs that admit an IC-planar drawing are the IC-planar graphs. Albertson [Alb08] uses the concept of IC-planar graphs in 2008. Later, they were further investigated by, e.g., Král' and Stacho [KS10], Zhang and Liu [ZL13], and Brandenburg et al. [BDE$^+$16].

If there is at most one common vertex for each pair of crossings in a 1-planar drawing, we call this drawing *NIC-planar*, where NIC stands for *near-independent crossings.* The graphs that admit a NIC-planar drawing are the NIC-planar graphs. The NIC-planar graphs were introduced in 2013 by Zhang [Zha13] and further studied by Czap and Šugerek [CŠ14] and Bachmaier et al. [BBH$^+$17].

The 1-planar graphs generalize the NIC-planar graphs, which generalize the IC-planar graphs, which generalize the planar graphs. Other than for the planar graphs, it is $\mathcal{NP}$-hard to decide whether a given general graph is 1-planar [GB07, KM13], or NIC-planar [BBH$^+$17], or IC-planar [BDE$^+$16]. In this thesis, we also expect a 1-planar embedding of a graph to be given when we are supposed to draw 1-planar graphs in polynomial time. Such an embedded graph is called *1-plane graph.*

In terms of drawing these graphs, a similar result as for the planar graphs exists for IC- and NIC-planar graphs. Any IC-planar graph can be drawn as an IC-planar drawing on a grid of quadratic size [BDE$^+$16]. More general, any NIC-planar graph can be drawn as a NIC-planar drawing on a grid of quadratic size [BBH$^+$17]. Given an embedding, a drawing can be computed in linear time, but the embedding may be changed. This result does not apply to 1-planar graphs. A 1-plane graph admits a straight-line drawing, where the embedding is preserved, if and only if it contains no so called *B-* or *W-configurations* [Tho88]. An illustration of a B- and a W-configurations is given in Fig. 1.1. Straight-line drawings of some of these graphs require exponential area [HELP12]. Using a bend point to overcome B- and W-configurations, we can obviously draw every 1-plane graph with at most one bend per edge. Even if we drop the requirement of preserving the embedding, there are still 1-planar graphs that do not have a 1-planar straight-line drawing [Did13]. On the positive side, every triconnected 1-planar graph admits a 1-planar drawing on a grid of quadratic size with every edge being drawn as a straight line segment except for maybe one edge on the outer face which may have a single bend [ABK13].

Note that these results apply to drawings with any crossing angle. In this thesis, we focus on RAC drawings, where the edges of each pair of crossing edges are drawn orthogonal to each other. RAC stands for *right-angle-crossing.* Graphs that admit a RAC drawing are called RAC graphs. E.g., Didimo et al. [DEL11] and Arikushi et al. [AFK+12] studied this class of graphs. They showed that every graph admits a RAC drawing with at most three bends per edge, but not every graph has a RAC drawing with two or fewer bends per edge. Given an embedding to a graph that should be preserved, three bends per edge may not even be enough. Some embedded graphs need quadratically many bends per edge for any RAC drawing. It is $\mathcal{NP}$-hard to decide whether a graph is straight-line RAC [ABS12].

For the relation of RAC graphs to IC-, NIC-, and 1-planar graphs, the following is known. Every IC-planar graph admits a straight-line IC-planar RAC drawing [BDE+16]. Given an embedding, it can be computed in $O(n^3)$ time where n is the number of vertices, but the embedding may be changed. In terms of area consumption, some of these graphs require exponential area for any straight-line IC-planar RAC drawing. In contrast to this, the straight-line RAC graphs are incomparable with the NIC-planar graphs [BBH+17] and with the 1-planar graphs [EL13]. Moreover, it is $\mathcal{NP}$-hard to decide whether a graph admits a 1-planar RAC drawing even if a rotation system is given [BDL+17].

These results motivate us to examine more closely the cases where bends are allowed. Bekos et al. [BDL+17] showed that every 1-planar graph has a 1-planar RAC drawing with at most one bend per edge. Given a 1-plane graph, there is an algorithm that computes such a drawing in linear time. However, the returned drawing may require exponential area and the embedding may be changed.

Own contribution and structure of this thesis Bekos et al. [BDL+17] ask if all 1-planar embeddings can be realized as 1-bend RAC drawings. We positively answer this question by slightly modifying their algorithm so that the embedding is preserved. This modification is described in Chapter 4.

They also ask if polynomial area is sufficient for all 1-planar graphs for a 1-planar 1-bend RAC drawing.[1] We do not answer this question for 1-planar graphs, but for NIC-planar graphs, and we answer a relaxed version of this question for 1-planar graphs when we allow a second bend for every edge. Note that we refer to drawings where every vertex, every bend point, and every crossing point is placed on a grid point of a grid with the specified size when we speak of drawing area.

First, we positively answer this question for the class of NIC-planar graphs, a subset of the 1-planar graphs. We prove in Chapter 3 that any NIC-planar graph admits a NIC-planar 1-bend RAC drawing on a grid of quadratic size. Given a NIC-plane graph, we give an algorithm that computes such a drawing in linear time without changing the embedding. This result immediately improves the result by Liotta and Montecchiani [LM16], who proved that every IC-plane graph has a 2-bend RAC drawing on a grid of quadratic size, in two respects. On the one hand, we generalize the class of embedded graphs from IC-plane graphs to NIC-plane graphs, and, on the other hand, we reduce the number of bends per edge from two to one. This is the main result of this thesis.

Second, we show that polynomial area is sufficient for 1-planar graphs if we allow a second bend for every edge. We prove in Chapter 5 that any 1-planar graph admits a

[1]In their paper they ask, the other way round, if exponential area is necessary for some family of 1-planar graphs.

1-planar 2-bend RAC drawing on a grid of polynomial size. Given a 1-plane graph, we give an algorithm that computes such a drawing on a grid of size $O(n^3) \times O(n^3)$ in linear time without changing the embedding. Here, n is the number of vertices.

Before we show these results, we establish some basic definitions and present some known facts and techniques in Chapter 2. There, we also give a tabular overview and a visualization as a containment diagram about RAC drawings with bends and their relations to graph classes mentioned before. Moreover, we give an example of a family of NIC-planar graphs with the maximum number of crossings according to a known bound by Zhang [Zha13] and by Czap and Šugerek [CŠ14]. This implies that this bound is tight. We end the main part of this thesis with a conclusion in Chapter 6. In Appendix A, we give an example for a NIC-planar 1-bend RAC drawing returned by a Java implementation of our algorithm from Chapter 3.

2. Preliminaries

In this chapter we introduce some structures, graph classes and algorithms that are used in the following chapters. We start with some basic definitions in Section 2.1 and introduce some specific subgraphs in Section 2.2. Thereafter, we show some known results in Section 2.3. These results concern the edge and the crossing density in IC-, NIC-, and 1-planar graphs and the relationship between beyond-planar graphs and RAC graphs. At the end of this chapter, in Section 2.4, we introduce the linear-time algorithm by Hopcroft and Tarjan [HT73] for making a plane graph biconnected and the linear-time Shift Algorithm [dFPP90, CP95] and its variant by Harel and Sardas [HS98].

2.1. Basic definitions

In this section, basic terms are defined as they appear in other literature. This thesis is about graph drawings where the edge direction does not matter. Therefore, we define graphs with undirected edges.

Definition 2.1 (Graph)**.** A *graph* $G = (V, E)$ is a pair of a finite set of vertices V and edges E, where $E \subseteq \binom{V}{2}$. Here, $\binom{V}{2}$ is the set of all subsets of V of size 2.

We refer to the vertices of a graph G also by $V[G]$. As edges are often seen as a connection of two vertices, we also say for an edge $e = \{u, v\}$ that there is an edge e *between* the vertices u and v. The vertices u and v are called *adjacent* if there is an edge $\{u, v\}$. A vertex v is called *incident* to an edge e and vice versa if $v \in e$. The set of all adjacent vertices of a vertex v is called $\mathrm{Adj}[v]$. Including v itself, we have $\mathrm{Adj}(v) := \mathrm{Adj}[v] \cup \{v\}$. We define the degree of a vertex v as $|\mathrm{Adj}[v]|$.

A graph can be visualized with a drawing.

Definition 2.2 (Drawing of a graph)**.** A mapping Γ is called a *drawing* of the graph $G = (V, E)$ if ...

- for all $v \in V : \Gamma(v) \in \mathbb{R}^2$,
- for all $u, v \in V$ with $u \neq v : \Gamma(u) \neq \Gamma(v)$, and
- for all $\{u, v\} \in E : \Gamma(\{u, v\}) = \Gamma_{\{u,v\}}([0, 1])$, where $\Gamma_{\{u,v\}}([0, 1])$ is a simple open Jordan curve in $\mathbb{R}^2$ with $\Gamma_{\{u,v\}}(0) = \Gamma(u)$ and $\Gamma_{\{u,v\}}(1) = \Gamma(v)$.

For convenience, the simple open Jordan curves that refer to edges are sometimes simply called edges and the points that refer to vertices are sometimes called vertices. This is done in cases where it is clear that not the abstract vertices and edges are meant, but their representation in a drawing. In this thesis, we will consider only *simple* drawings. In a simple drawing, no edge crosses itself (has a loop) and edges incident to the same vertex do not cross.

A *straight-line drawing* is a drawing where the curve of every edge is a line segment. Observe that, in such a drawing, each edge curve is sufficiently defined by only the locations of its two incident vertices.

A *polyline drawing* Γ is a drawing where the curve of every edge is a polygonal chain, i.e., a connected series of line segments, also known as polyline. The endpoints of a line segment of a polyline of an edge $\{u, v\}$ that are not $\Gamma(u)$ or $\Gamma(v)$ are called *bends* or *bend points*. Obviously, the number of bend points of a polyline is one less than the number of line segments of this polyline. E.g., if a polyline consists of four line segments, it has three bend points. To keep a drawing overseeable it is often better to have only a few bend points in total and a few bend points per edge. For us, polyline drawings with a bounded number of bends per edge are of greater interest. For example, we create polyline drawings with at most one bend per edge (*1-bend drawings*) in Chapter 3, and we create polyline drawings with at most two bends per edge (*2-bend drawings*) in Chapter 5. Note that every straight-line drawing is also a polyline drawing with at most 0 bends per edge. Observe that, in a polyline drawing, the curve of the edge $\{u, v\}$ is sufficiently defined by only the locations of u, v, and all bend points, and the order of the bend points.

A *drawing on the grid of size $w \times h$* is a drawing where each vertex has integer coordinates in the range $[0, w] \times [0, h]$. We say that the vertices are placed on grid points. When we speak of a grid drawing with *everything* on the grid, we mean that vertices, bend points, and crossing points have integer coordinates.

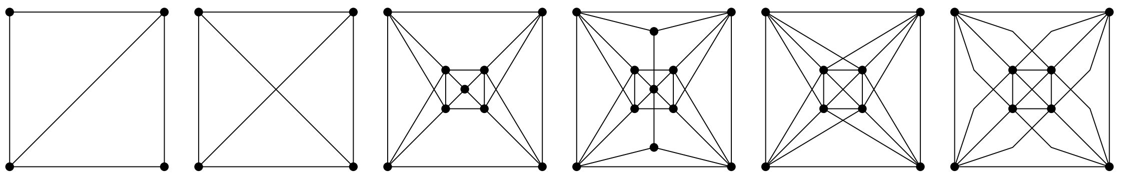
Crossing points emerge when edges cross. We formally define them next.

Definition 2.3 (Edge crossing)**.** Let Γ be a drawing of a Graph $G = (V, E)$. The edges $e_1 = \{v_1, v_2\} \in E$ and $e_2 = \{v_3, v_4\} \in E$ with $e_1 \neq e_2$ are said to *cross* each other (*cross pairwise*) in Γ if there is a so called *crossing point* $c \in \mathbb{R}^2$ with $c \in \Gamma_{e_1}([0, 1])$ and $c \in \Gamma_{e_2}(]0, 1[)$. The vertices v_1, v_2, v_3 and v_4 are called vertices of the crossing.

Note that there can be more than one pairwise edge crossing in the same crossing point. In such a case, three or more edges intersect in the same point. Thus, there can be an edge with only one crossing point that is crossed pairwise twice or more often.

Definition 2.4 (Types of drawings)**.** A drawing is called ...

- ... *crossing-free* if there is no crossing of any pair of edges.
- ... *k-planar* if every edge is crossed pairwise at most k times.
- ... *IC-planar*, where IC stands for *independent crossings*, if every edge is crossed pairwise at most once and for each pair of different edge crossings, there is no common vertex.
- ... *NIC-planar*, where NIC stands for *near-independent crossings*, if every edge is crossed pairwise at most once and for each pair of different edge crossings, there is at most one common vertex.



(a) Straight-line planar drawing.

(b) Straight-line 1-planar RAC drawing.

(c) Straight-line IC-planar drawing.

(d) Straight-line NIC-planar drawing.

(e) Straight-line 1-planar drawing.

(f) 1-bend 1-planar RAC drawing.

Fig. 2.1.: Examples of different types of drawings. Note that the drawing in Fig. 2.1d is not IC-planar and the drawing in Fig. 2.1e is not NIC-planar. The drawings in Fig. 2.1e and Fig. 2.1f are drawings of the same graph. The last drawing equals the example of a 1-planar 1-bend RAC drawing in the annotated bibliography on 1-planarity by Kobourov et al. [KLM17].

- … *RAC*, where RAC stands for *right-angle-crossing*, if it is a polyline drawing and for each crossing point c, all of the following holds:
  - There are at most two edges that cross in c.
  - The crossing point c is no bend point.
  - The line segments of the edges that cross in c intersect in a right angle.
- … *k-bend* if it is a polyline drawing with at most k bends per edge.

Examples of the introduced types of drawings are given in Fig. 2.1. Based on these types of drawings we can define some classes of graphs.

Definition 2.5 (Classes of graphs)**.** A graph is called … .

- … *planar* if it admits a crossing-free drawing.
- … *k-planar* if it admits a k-planar drawing.
- … *IC-planar* if it admits an IC-planar drawing.
- … *NIC-planar* if it admits a NIC-planar drawing.
- … *RAC* if it admits a RAC drawing.

The 0-planar graphs are the planar graphs. For $k \geq 1$, the only considered k-planar graph class here and in most of the literature is the one with $k = 1$, the 1-planar graphs. Observe that the IC-planar graphs are a generalization of the planar graphs, the NIC-planar graphs are a generalization of the IC-planar graphs and the 1-planar graphs are a generalization of the NIC-planar graphs.

A graph can also be in a class combined of these classes and in a combination with *k-bend*. Then it has a drawing that satisfies the requirements of all of the classes. For example, if a graph is 1-planar 1-bend RAC, it has a polyline drawing with at most one bend per edge where every edge is crossed at most once and at every crossing point, the edges cross in a right angle. RAC drawings that use an unbounded number of bends per edge are not considered here—only RAC drawings without bends (*straight line RAC*) and with at most one or two bends per edge. We let RAC_k denote the family of the k-bend RAC graphs, and we let $\mathrm{RAC}_k^{\mathrm{poly}}$ denote the family of the k-bend RAC graphs that admit a k-bend RAC drawing on a grid of polynomial size. Note that the straight-line RAC graphs are RAC_0.

Definition 2.6 (Face of a drawing)**.** Let Γ be a drawing of a graph G. The connected areas of $\mathbb{R}^2$ that are bounded by curves of Γ that refer to edges of G are called *faces* of Γ.

The face of a drawing that is bounded by the outermost curves of the drawing is infinitely large. It is called the *outer face.* The other faces are called *inner faces.*

For each face f in a crossing-free drawing of a planar graph $G = (V, E)$, we can also have a *boundary list.* Assume there is some indexed order of the vertices in V. Going along the boundary of f in counter-clockwise direction, we encounter curves that represent edges either on their right or on their left side. Using the order in which we encounter them, we can define a circular list (a circular sequence). An entry of this circular list consists of an edge $\{v_i, v_j\} \in E$ with $i < j$ and an *edge side.* The edge side is *left* if we encounter $\{v_i, v_j\}$ from the left side seen from v_i and *right* otherwise. We call this circular list the *boundary list* of f. We define the degree of f as the size of its boundary list.

So, for every crossing-free drawing, there is a set of boundary lists and one of them is the boundary list of the outer face. Each edge appears exactly twice in this set of boundary lists. One time in combination with the edge side left and the other time in combination with the edge side right. This can be in the same boundary list or in different ones. Implicitly, a boundary list also defines a list of vertices that are corner points in the face. In each such list, the same vertex can appear many times as a corner point. There are many crossing-free drawings of the same planar graph with the same set of boundary lists and the same boundary list of the outer face. These drawings are similar in dividing the drawing area into faces. We say that they are embedded in the same way.

Definition 2.7 (Planar embedding)**.** Let G be a planar graph. A *planar embedding* $\mathcal{E}(G)$ is an equivalence class of crossing-free drawings of G admitting the same set of boundary lists and having the same boundary list of the outer face.

A boundary list of a (planar) embedding is sometimes also called face. Every crossing-free drawing can be assigned to exactly one planar embedding, but for each planar embedding there can be infinitely many crossing-free drawings respecting this planar embedding. Alternatively, planar embeddings can be defined via a rotation system.

Definition 2.8 (Rotation system)**.** Let $G = (V, E)$ be a graph. A *rotation system* of G is a mapping from each $v \in V$ to a circular list (circular sequence) of the vertices in $\mathrm{Adj}[v]$. A drawing Γ of G *respects* a rotation system R if for each $v \in V$, the order of edges encountered when scanning around v in Γ in counter-clockwise direction is the same as in the circular list of v in R.

Planar embeddings can also be defined as equivalence class of crossing-free drawings respecting the same rotation system and having the same boundary list of the outer face.

Since we work with non-planar graphs in this thesis, we extend the concept of planar embeddings to non-planar graphs next.

Procedure 2.9 (Subdividing an edge)**.** Let $G = (V, E)$ be a graph. *Subdividing an edge* $e = \{u, w\} \in E$ via a dummy vertex $v_{\text{dummy}} \notin V$ means that e is replaced by the two edges $\{u, v_{\text{dummy}}\}$ and $\{v_{\text{dummy}}, w\}$. With this operation, we obtain a modified graph $G' = (V \cup \{v_{\text{dummy}}\}, (E - \{e\}) \cup \{\{u, v_{\text{dummy}}\}, \{v_{\text{dummy}}, w\}\})$. We can subdivide more than one edge via the same dummy vertex.

Definition 2.10 (Planarized graph)**.** Let $G = (V, E)$ be a graph. A planar graph G' is called *planarized graph* of G if G' can be obtained from G by a sequence of operations in which edges are subdivided. In each such operation, two or more edges of the current graph are subdivided via the same dummy vertex.

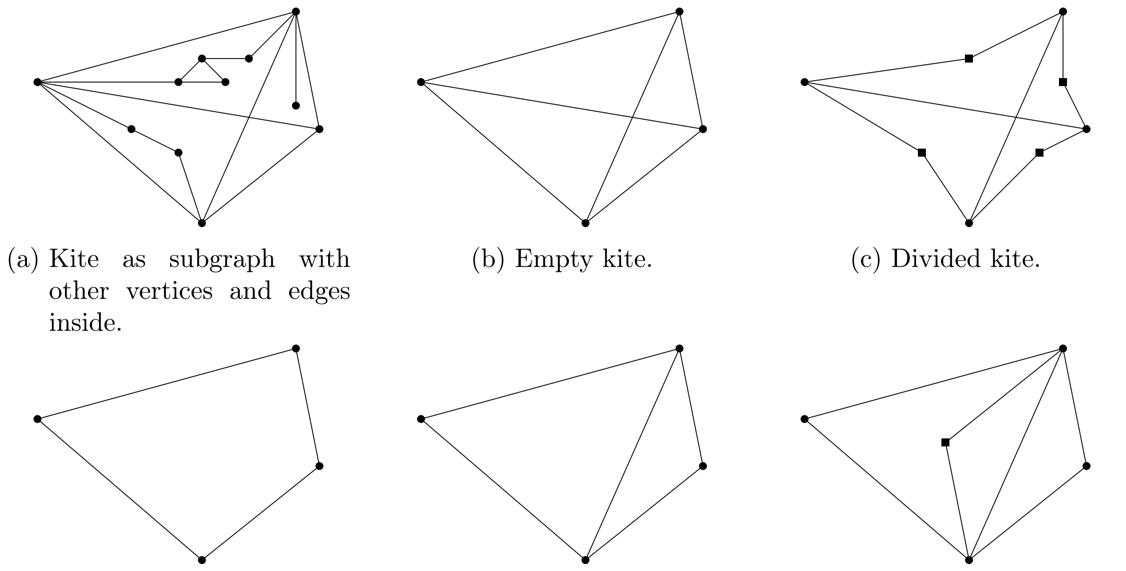
Interpret the subdivided edges as crossed edges and the inserted dummy vertices as crossing points. Given a graph G and a drawing Γ of G, a planarized graph G' of G can easily be obtained by replacing each crossing point in Γ by a dummy vertex (subdivide the crossing edges via a dummy vertex). The resulting graph G' is planar because it has a crossing-free drawing which is Γ with each crossing point being replaced by a dummy vertex. We will call such a dummy vertex *crossing vertex.*

Definition 2.11 (Planarized embedding)**.** Let G be a graph. A *planarized embedding* $\mathcal{E}(G)$ is a planar embedding of a planarized graph of G.

A graph G with a given planar or planarized embedding $\mathcal{E}(G)$ is called an *embedded graph.* We can denote this embedded graph as a pair $(G, \mathcal{E}(G))$ of the graph and its embedding or simply as $\mathcal{E}(G)$ if it is clear that the embedding contains also all informations about the graph. The following definition introduces special types of embedded graphs.

Definition 2.12 (Plane graphs)**.** A *plane graph* $(G, \mathcal{E}(G))$ is a planar graph G with a planar embedding $\mathcal{E}(G)$ of G. A *k-plane graph* $(G', \mathcal{E}'(G'))$ is a k-planar graph G' with a planarized embedding $\mathcal{E}'(G')$ of a k-planar drawing of G'. Analogously, an *IC-plane graph* is an IC-planar graph with a planarized embedding of an IC-planar drawing of this graph, and a *NIC-plane graph* is a NIC-planar graph with a planarized embedding of a NIC-planar drawing of this graph.

We will speak of IC-, NIC-, and 1-planar embeddings when we refer to embeddings of IC-, NIC-, and 1-planar drawings, respectively. A k-plane graph, an IC-plane graph and a NIC-plane graph have the set of crossing points (planarized as crossing vertices) as implicit information. We will speak of a *fixed embedding setting* if an algorithm preserves a given embedding of a given graph, and of a *variable embedding setting* if an algorithm preserves a given graph, but not necessarily a (possibly) given embedding of this graph.


(a) Kite as subgraph with other vertices and edges inside.

(b) Empty kite.

(c) Divided kite.

(d) Empty quadrangle.

(e) Divided quadrangle.

(f) Divided quadrangle with a shift vertex inside.

Fig. 2.2.: Specific embedded (sub)graphs.

We have introduced the concept of dummy vertices in this section. To distinguish them from the other vertices of a graph, we call the other vertices *original vertices*. Analogously, we distinguish *dummy edges* from *original edges*. In the figures and drawings of this thesis, original vertices are drawn as solid circles and dummy vertices as solid squares. Special dummy vertices like crossing vertices and shift vertices, which we introduce and use in Chapter 3, are drawn as solid squares as well. If it is irrelevant or unclear if a vertex is an original vertex or a dummy vertex, we will use solid circles. Bend points are drawn as non-solid square boxes and projection points as non-solid circles (see, e.g., Fig. 5.4b).

2.2. Specific subgraphs

In the context of the drawing algorithms described in the following chapters, we will use some specific embedded graphs and subgraphs that we introduce next.

Definition 2.13 (Kite)**.** A *kite* is a 1-plane graph consisting of the graph K_4 (the complete graph with four vertices) and a planarized embedding of K_4 so that every original vertex is at the outer face and there is a crossing vertex that is not on the outer face. A kite can also be an embedded subgraph of some larger graph.

An example of a kite as an embedded subgraph of a larger graph is given in Fig. 2.2a.

Definition 2.14 (Empty kite)**.** Let G be a graph and $\mathcal{E}(G)$ be a planarized embedding of G. A kite that is an embedded subgraph of $(G, \mathcal{E}(G))$ is called *empty* in $(G, \mathcal{E}(G))$ if the inner faces of the kite are also inner faces of $\mathcal{E}(G)$.

In other words, no other vertices of G are inside the kite and no other edge of G can cross an edge of the kite. An example of an empty kite is given in Fig. 2.2b. In contrast to Fig. 2.2a, there cannot be other edges or vertices of a supergraph inside an inner face of this kite.

Definition 2.15 (Divided kite)**.** Let G be a graph and $\mathcal{E}(G)$ be a planarized embedding of G. A *divided kite* is an embedded subgraph of $(G, \mathcal{E}(G))$ that is a 8-cycle on the outer face and a crossing vertex in its interior. The crossing vertex is adjacent to every second vertex of the 8-cycle.

An example of a divided kite is given in Fig. 2.2c. Analogously, we define the following graph structures.

Definition 2.16 (Empty Quadrangle)**.** Let G be a graph and $\mathcal{E}(G)$ be a planarized embedding of G. A 4-cycle Q that is an embedded subgraph of $(G, \mathcal{E}(G))$ is called *empty quadrangle* in $(G, \mathcal{E}(G))$ if Q has exactly one inner face and this inner face is also a inner face of $\mathcal{E}(G)$.

An example of an empty quadrangle is given in Fig. 2.2d.

Definition 2.17 (Divided Quadrangle)**.** Let G be a graph and $\mathcal{E}(G)$ be a planarized embedding of G. A 4-cycle Q with a chord (edge between non-consecutive vertices of a cycle) that is an embedded subgraph of $(G, \mathcal{E}(G))$ is called *divided quadrangle* in $(G, \mathcal{E}(G))$ if all vertices of Q are on the outer face of Q and Q has exactly two inner faces and these inner faces are also inner faces of $\mathcal{E}(G)$.

A divided quadrangle can also be represented as a pair of triangles. An example of a divided quadrangle is given in Fig. 2.2e. In Chapter 3, we will insert a dummy vertex that we call shift vertex into a divided quadrangle. Such a divided quadrangle will look like the one in Fig. 2.2f. In a drawn graph, we also refer to a drawing of an empty/divided quadrangle as empty/divided quadrangle.

2.3. Known results

In the Introduction (Chapter 1), we gave an overview about previous works and their results. This section shall be an extension to this. We visualize some results mentioned there, and we give some more details from these works. In particular, we revisit the relationships between beyond-planar graph classes and RAC graph classes in Section 2.3.1 and the relationships between the number of edges and crossings in the IC-/NIC-/1-planar graphs on the one side and the number of vertices on the other side in Section 2.3.2.

2.3.1. Relationships between beyond-planar graph classes and RAC graphs

We revisit the results presented in the Introduction and relate them to the results from the next chapters. To better visualize these results, they are collected in Table 2.2. In this table, n is the number of vertices of each graph. If there is no k-bend RAC drawing

Are the graph classes in the rows contained in RAC$_k$?

	RAC$_0$	**RAC$_1$**	**RAC$_2$**	**RAC$_3$**
IC-planar	Yes [BDE$^+$16], required area: exponential in n [BDE$^+$16] sufficient area: exponential in n [BDE$^+$16]	Yes (s. Chapter 3), required area: ? sufficient area: $\Theta(n^2)$ (s. Chapter 3)	Yes, required area: ? sufficient area: $\Theta(n^2)$ [LM16]	Yes, required area: ? sufficient area: $\Theta(n^2)$
NIC-planar	No [BBH$^+$17], open how difficult to decide	Yes (s. Chapter 3), required area: ? sufficient area: $\Theta(n^2)$ (s. Chapter 3)	Yes, required area: ? sufficient area: $\Theta(n^2)$	Yes, required area: ? sufficient area: $\Theta(n^2)$
1-planar	No [EL13], NP-hard to decide [BDL$^+$17]	Yes [BDL$^+$17], required area: ? sufficient area: exponential in n [BDL$^+$17] (see also Chapter 4)	Yes (s. Chapter 5), required area: ? sufficient area: $\Theta(n^6)$ (s. Chapter 5)	Yes, required area: ? sufficient area: $\Theta(n^4)$
general	No [DEL11], NP-hard to decide [ABS12]	No [AFK$^+$12], open how difficult to decide [DEL11]	No [AFK$^+$12], open how difficult to decide [DEL11]	Yes [DEL11], required area: ? sufficient area: $\Theta(n^4)$ (s. Fig. 2.3)

Tab. 2.2.: Overview about some beyond-planar RAC graph classes with 0–3 bends per edge.

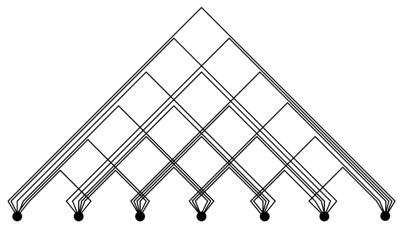


Fig. 2.3.: Example from the paper by Didimo et al. [DEL11] for an arbitrary graph which is drawn with at most three bends per edge as RAC drawing. Using this drawing style, a grid of size $\Theta(n^2) \times \Theta(n^2)$ suffices to have each vertex, bend point, and crossing point on a grid point.

for each graph of a specific class, the cells in this table have gray background color. If there is a k-bend RAC drawing for each graph of the considered class, the cell is colored. The color depends on the area that is required or sufficient for such a drawing. There, "required" means that there is a graph of this class, where any k-bend RAC drawing requires a grid of the specified size to have each vertex, bend point, and crossing point on the grid. On the other hand, "sufficient" means there that for all graphs of this class, there is a k-bend RAC drawing that has every vertex, bend point, and crossing point on the grid of the specified size. Red background color means that exponential area is required, green background color that polynomial area is sufficient, i.e., the considered graph class is a subset of $\mathrm{RAC}_k^{\mathrm{poly}}$, and yellow background color that it is unknown.

Beside this table, these relationships between k-bend RAC graphs with polynomially and arbitrarily sized drawings and beyond-planar graphs is depicted in a containment diagram in Fig. 2.4.

2.3.2. Edge and crossing density

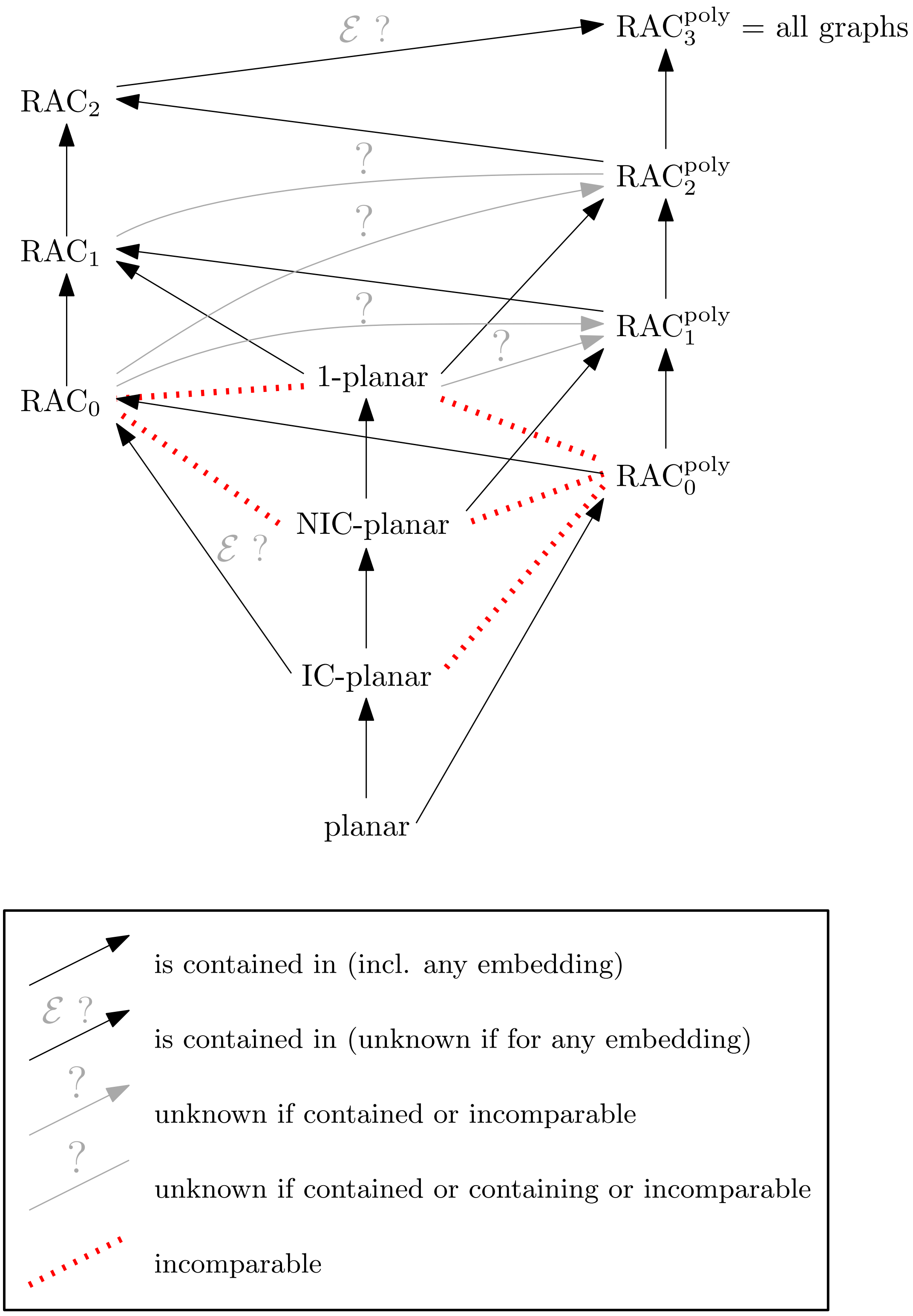
We present known bounds on the number of edges and the number of crossings in IC-, NIC-, and 1-planar graphs. We will use some of the following properties later.

Lemma 2.18 ([Zha13, CŠ14, BBH$^+$17]). Any NIC-planar drawing of any NIC-planar graph with n vertices has at most $18(n-2)/5 = 3.6n - 7.2$ edges. This bound is tight.

Similar results are known for other classes of graphs. Except for the last two, all of the following bounds are tight. IC-planar graphs have at most $3.25n - 6$ edges [ZL13], 1-planar graphs have at most $4n - 8$ edges [PT97], straight-line 1-planar graphs have at most $4n-9$ edges [Did13], straight-line RAC graphs have at most $4n-10$ edges [DEL11], 1-bend RAC graphs have at most $6.5n - 13$ edges [AFK$^+$12], and 2-bend RAC graphs have at most $74.2n$ edges [AFK$^+$12].

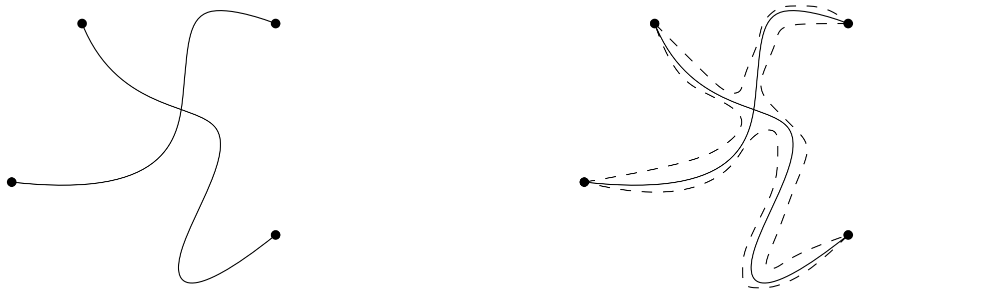
Next, we give some results on the maximum number of crossings in these drawings relative to the number of vertices n. Obviously, IC-planar graphs have up to $n/4$ crossings. For 1-planar drawings, the following bound in the number of crossings has been proven.

Lemma 2.19 ([CH13]). Any 1-planar drawing of any 1-planar graph with n vertices has at most $n - 2$ crossings. This bound is tight.



$\mathcal{E}$?
$\mathrm{RAC}_3^{\mathrm{poly}}$ = all graphs
RAC_2
?
$\mathrm{RAC}_2^{\mathrm{poly}}$
?
RAC_1
?
?
$\mathrm{RAC}_1^{\mathrm{poly}}$
1-planar
RAC_0
$\mathrm{RAC}_0^{\mathrm{poly}}$
NIC-planar
$\mathcal{E}$?
IC-planar
planar
is contained in (incl. any embedding)
$\mathcal{E}$?
is contained in (unknown if for any embedding)
?
unknown if contained or incomparable
?
unknown if contained or containing or incomparable
incomparable

Fig. 2.4.: Relationships between (beyond-)planar graphs and RAC graphs.


(a) Given NIC-planar crossing.

(b) Way for routing the four non-crossing edges of the kite along these two crossing edges.

Fig. 2.5.: An example of a way to route missing edges of a kite alongside the crossing edges without creating new crossings. Used in the proof of Lemma 2.20.

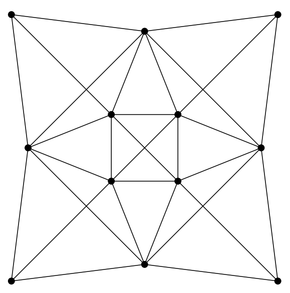
For NIC-planar drawings, an upper bound of $(3n-6)/5$ crossings has been proven by Zhang [Zha13] and by Czap and Šugerek [CŠ14]. They do not explicitly state that this bound is tight. We give another proof to their result, which is similar, but we argue with the maximum number of edges in a NIC-planar graph instead of the face–edge ratio in triangulated graphs. Moreover, we give a series of examples to show that this bound is tight for infinitely many values of n.

Lemma 2.20 ([Zha13, CŠ14])**.** Any NIC-planar drawing of any NIC-planar graph G with n vertices has at most $(3n-6)/5 = 0.6n - 1.2$ crossings. This bound is tight.

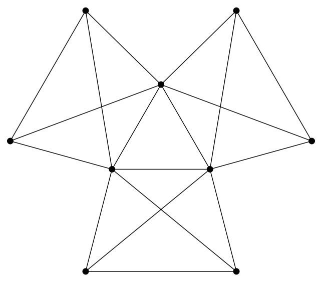
Proof. Let G be an arbitrary NIC-planar graph with n vertices and m edges, and let Γ be an arbitrary NIC-planar drawing of G with k crossings. The drawing Γ is not necessarily maximal NIC-planar. *Maximal NIC-planar* means that we cannot add an edge without violating the NIC-planarity. We can make Γ maximal NIC-planar by adding edges. While there are vertices u and v with no edge $\{u,v\}$ between them, but a way to insert $\{u,v\}$ into the drawing so that the drawing is still NIC-planar, we add the edge $\{u,v\}$ in this way. As soon as we cannot add any more edge without violating the NIC-planarity of the drawing (we always come to this situation since the number of edges is bound in the number of vertices in a NIC-planar graph and we do not add vertices), we have a maximal NIC-planar drawing Γ' of a graph G' with n vertices and m' edges. The drawing Γ' has k' crossings.

Because the graph is maximal NIC-planar, we have a kite around every crossing in Γ'. This kite is not necessarily empty (s. Section 2.2). If one of the four non-crossing edges of a kite has not been in G, we can always insert it without creating a crossing and thus never violating the condition for the NIC-planarity. We can insert it since we can route it along the crossing edges like in Fig. 2.5. Observe that for any pair of kites (A,B) in Γ', the sets of edges of A and B are disjoint as otherwise the graph would not be NIC-planar. Thus, we have at least 6 edges per crossing and can bound the number of crossings k' in the number of edges m':

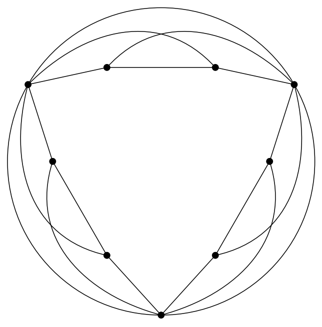
$$k' \leq \frac{m'}{6} \tag{2.1}$$



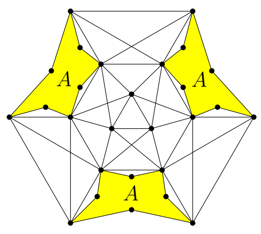
(a) Structure A. Used in the next drawings in the yellow boxes titled with A.



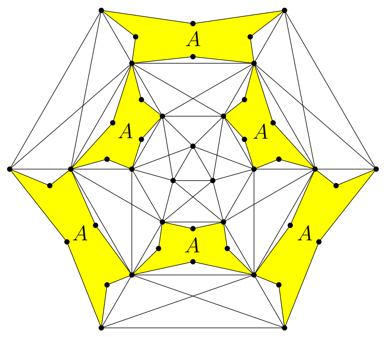
(b) Inner base of the drawing.



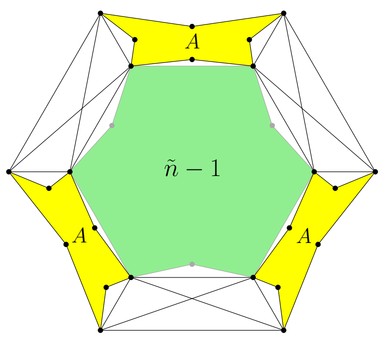
(c) Outer hull of the drawing.



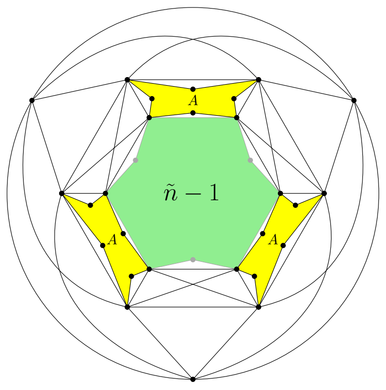
(d) Inner base with the first layer around.



(e) Inner base with the first and the second layer around.



(f) Inner $\tilde{n}-1$ layers hidden in green with the $\tilde{n}$-th layer around.



(g) Complete drawing $\Gamma_{\tilde{n}}$ with the outer hull around the $\tilde{n}$-th layer and the $\tilde{n}$-th layer around $\tilde{n}-1$ layers hidden in green.

Fig. 2.6.: Structures that are used to create a series of drawings $\Gamma_0, \Gamma_1, \Gamma_2, \ldots, \Gamma_{\tilde{n}}$ that have $0.6n - 1.2$ crossings each. Here, n is the number of vertices in a graph.

We know by Lemma 2.18 that in every NIC-planar graph, there are at most $18(|V|-2)/5$ edges where $|V|$ is the number of vertices. Applying this to Inequality 2.1, we get the following bound:

$$k' \leq \frac{18(n-2)}{5 \cdot 6} = \frac{3n-6}{5} \tag{2.2}$$

Since we cannot lose a crossing by adding edges, this bound applies also to the initial drawing Γ of the initial graph G.

$$k \leq k' \leq \frac{3n-6}{5} \tag{2.3}$$

After showing that there is no NIC-planar drawing with more than $0.6n - 1.2$ crossings, it remains to show that this bound is tight. We do this by giving an infinitely large series of examples of NIC-planar drawings $\Gamma_0, \Gamma_1, \Gamma_2, \ldots$ that do have exactly $0.6n - 1.2$ crossings each. The drawings are built up with structures depicted in Figure 2.6. Every drawing $\Gamma_{\tilde{n}}$ with $\tilde{n} \in \mathbb{N}_0$ has three empty kites arranged as in Figure 2.6b as an inner base in its center. Around it, there are $\tilde{n}$ layers of rings that consist of three empty kites and three times a structure which we call Structure A (it is depicted in Figure 2.6a and contains five empty kites). In each such ring, the three empty kites and the three Structures A are rowed in an alternating way as in Figure 2.6d. The inner six and the outer six vertices of each ring are shared with the previous ring (or the inner base) and the next ring (or the outer hull). Around the $\tilde{n}$-th ring, three empty kites arranged as in Figure 2.6c form an outer hull. A layer (ring) is arranged around the previous layer (ring) such that empty kites and Structures A are alternating (so that empty kites do not share edges). See Figure 2.6e. It applies to the inner base and the outer hull in the same way that empty kites do not share edges. See Figure 2.6d and Figure 2.6g. Every drawing $\Gamma_{\tilde{n}}$ is obviously NIC-planar.

In Table 2.4, we analyze the number of vertices, edges, and crossings in each such drawing. Reformulating the entry for the number of vertices in $\Gamma_{\tilde{n}}$, we can see how the drawing index $\tilde{n}$ depends on the number of vertices n.

$$\tilde{n} = \frac{n-12}{30} \tag{2.4}$$

Inserting this into the formula for the number of crossings, we get the number of crossings k in dependency on the number of vertices n:

$$k = 18\tilde{n} + 6 = 18 \cdot \frac{n-12}{30} + 6 = \frac{3n-6}{5} \tag{2.5}$$

□

Observe that $\Gamma_{\tilde{n}}$ is also a NIC-planar drawing with the maximum number of edges:

$$m = 108\tilde{n} + 36 = 108 \cdot \frac{n-12}{30} + 36 = 3.6n - 7.2 \tag{2.6}$$

Furthermore, note that Γ_0 equals examples from other papers [CŠ14, BBH$^+$17, KLM17]. Some of them use it as an example of a NIC-planar graph with the maximum number of edges.

Drawing	**Number of vertices** n	**Number of edges** m	**Number of crossings** k
Structure A	12	30	5
Inner Base	9	18	3
Outer Hull	9	18	3
Ring in each layer	36	108	18
Γ_0	12	36	6
Γ_1	42	144	24
Γ_2	72	252	42
$\Gamma_{\tilde{n}}$	$30\tilde{n} + 12$	$108\tilde{n} + 36$	$18\tilde{n} + 6$

Tab. 2.4.: Number of vertices, edges, and crossings of structures and graphs from Fig. 2.6.

2.4. Known algorithms

In this section, we will introduce two widespread algorithms that we will use in this thesis later. This is, in Section 2.4.1, a framework for the Algorithm by Hopcroft and Tarjan [HT73] to make a graph biconnected. And this is, in Section 2.4.2, the Shift Algorithm by de Fraysseix et al. [dFPP90] and Chrobak and Payne [CP95] and its variant by Harel and Sardas [HS98].

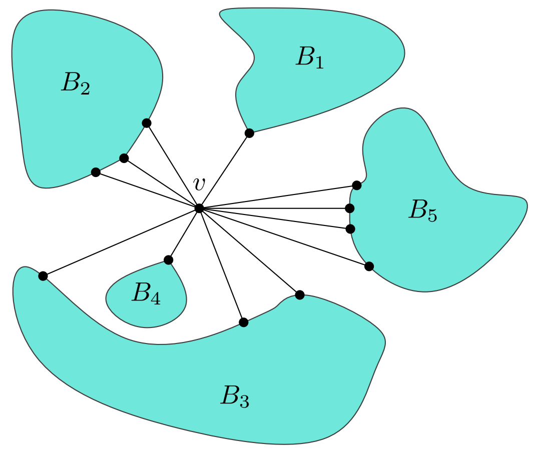
2.4.1. Making a graph biconnected

There are applications like the drawing algorithm by Harel and Sardas [HS98] that require *biconnected* graphs. A *biconnected graph* is a connected graph where we can remove any vertex and the resulting graph is still connected. If a graph is biconnected, it has no *cut vertex*. A *cut vertex* v of a graph G is a vertex whose removal would increase the number of connected components, i.e., $G[V - \{v\}]$ has more connected components than G. As we work with general planar and beyond-planar graphs that are not necessarily biconnected, we describe an algorithm that transforms a plane graph in linear time into a biconnected plane graph and maintains its planarity by only adding dummy edges. The algorithm is also noted down in Algorithm 1.

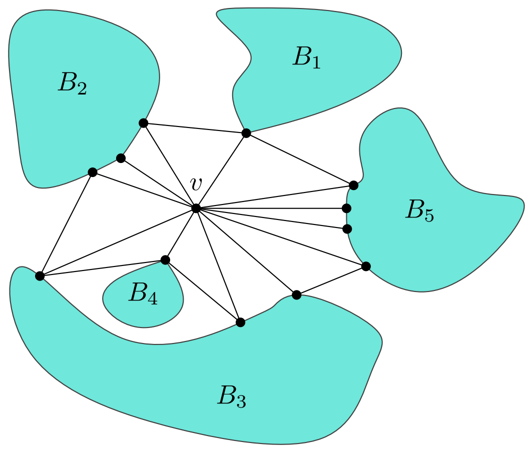
Let $(G, \mathcal{E}(G))$ be a given plane graph. In case G is not connected, we first do a depth-first search to find the connected components. If it is unconnected, we assume that there is no connected component on an inner face of another component. If there is more than one connected component, we take a component and connect it with each other component via a dummy edge. Each such dummy edge connects vertices on the outer face so that it does not create crossings in the embedded graph. Now, we have a plane connected graph that is not necessarily biconnected. We use the linear-time algorithm by Hopcroft and Tarjan [HT73], which is based on a depth-first search, to find all biconnected components and their cut vertices. We obtain biconnectedness by the

Algorithm 1:
makePlaneGraphBiconnected($(G, \mathcal{E}(G))$)

```
Input  : Plane graph $(G, \mathcal{E}(G))$
Output: Biconnected plane graph $(G^*, \mathcal{E}^*(G)^*)$
/* Make the graph connected                                              */
1  $(C_1, \ldots, C_j)$ := dfsFindConnectedComponents($G$) // depth-first search for
       finding connected components
2  Let $c^*$ be a vertex in $C_1$
3  foreach $c \in C_1$ do
4      if $c$ is on the outer face of $\mathcal{E}(G)$ then
5          $c^*$ := c
6  foreach $C \in \{C_2, \ldots, C_j\}$ do
7      foreach $c \in C$ do
8          if $c$ is on the outer face of $\mathcal{E}(G)$ then
9              Insert the dummy edge $\{c, c^*\}$ on the outer face
10             break
/* Make the graph biconnected                                            */
11 $(B, V_{\text{cut}})$ := hopcroftTarjanBiconnectedComponents($G$) // The set of the
       biconnected components of the graph is $B$. Presume that we can
       query via bc($e$) for each edge $e$ in constant time in which
       biconnected component $e$ is. The set of the cut vertices is $V_{\text{cut}}$.
12 foreach $v \in V_{\text{cut}}$ do
13     foreach $u \in \text{Adj}[v]$ in the circular order around $v$ acc. to the embedding do
14         Let $w$ be the predecessor of $u$ in the circular order around $v$
15         if $(\text{bc}(\{v, u\}) \neq \text{bc}(\{v, w\})$ then
16             if $v$ has degree 2 and we inserted $\{u, w\}$ in the previous step then
17                 break // Special case to prevent parallel edges
18             Insert the dummy edge $\{u, w\}$
19             Set $\text{bc}(\{u, w\}) = \{\text{bc}(\{v, u\})\} \cup \{\text{bc}(\{v, w\})\}$ // We set bc($\{u, w\}$)
                   to two values. Original edges are assigned to only one
                   biconnected component. When we compare bc($\{u, w\}$) with
                   other values, we check for both assigned biconnected
                   components if one equals the other value
20 Let $(G^*, \mathcal{E}^*(G^*))$ be the resulting embedded graph
21 return $(G^*, \mathcal{E}^*(G^*))$
```



(a) Cut vertex v of the biconnected components $B_1, \dots, B_5$.



(b) Connecting neighbors of v in their circular order around v if they are in different biconnected components.

Fig. 2.7.: Connecting five biconnected components around a split vertex v.

following operation. For each cut vertex v, consider the neighbor vertices of v in the circular order around v. If two consecutive incident edges between v and these neighbor vertices are in different biconnected components, connect them via a dummy edge. In case v has degree 2 and is a cut vertex of two biconnected components, we insert only one such dummy edge since we may not create parallel edges. We give an example of this step in Fig. 2.7. After applying this to each cut vertex, we return the obtained biconnected plane graph $(G^*, \mathcal{E}^*(G^*))$.

With this algorithm, we follow the suggestions in the remarks of the paper by Harel and Sardas [HS98] for making a graph biconnected. Let n be the number of vertices in G and G^*, and m be the number of edges in G^*. The depth-first search and the algorithm by Hopcroft and Tarjan [HT73] run in linear time. Regarding the insertion of dummy edges in the unconnected case, we insert up to n edges and check for up to n vertices if they are on the outer face. We can do this in linear time. For the insertion of dummy edges in the non-biconnected case, we handle at most n cut vertices with in total at most $2m$ neighbor vertices. Since $m \in O(n)$ holds for planar graphs, we can also do this insertion of dummy edges in linear time. Hence, the algorithm runs in linear time.

2.4.2. The Shift Algorithm for maximal planar and biconnected graphs

In Chapter 3 and Chapter 5, we will employ the algorithm by Harel and Sardas [HS98]. This algorithm is an adaption of the older Shift Algorithm. Due to our intensified use of this algorithm in this thesis, we give a brief overview of it here.

The Shift Algorithm was introduced in 1990 by de Fraysseix et al. [dFPP90]. They give an algorithm that takes a planar graph $G = (V, E)$ as input and returns a crossing-free straight-line drawing Γ of G on a grid of size $(2n - 4) \times (n - 2)$ where $n := |V|$. The algorithm consists of two steps, and a preprocessing, and a postprocessing. As preprocessing, we compute a planar embedding of G in linear time and triangulate each face of it via dummy edges in linear time. Let the resulting graph with a triangulated embedding be G'. The first step is computing a *canonical ordering* Π of the vertices in G and the second step is computing a crossing-free straight-line drawing on a grid of size $(2n-4) \times (n-2)$ based on Π. We will formally define a canonical ordering hereunder. As postprocessing, we remove the dummy edges from the drawing. Their algorithm has a running time of $O(n \log n)$. Chrobak and Payne [CP95] present a modification of their algorithm to achieve linear time.

The classical Shift Algorithm requires triangulated embeddings. Since a plane graph is in general not triangulated, i.e., not maximal planar, we insert dummy edges, which we remove later. A graph is maximal planar if adding an edge would make the graph non-planar. This removal can make a drawing look rather unaesthetic. For this reason, Harel and Sardas [HS98] give a kind of generalization of the Shift Algorithm so that it does not require maximal planar graphs but only biconnected planar graphs. For making a graph biconnected, we usually need fewer dummy edges than for making a graph maximal planar. They name their adaption of the canonical ordering *biconnected canonical ordering*. The drawing algorithm in the second step is almost the same.

Next, we define the canonical ordering and the biconnected canonical ordering and give a little insight into their computation. Moreover, we describe the drawing phase briefly.

Canonical ordering

First of all, we give a formal definition of the canonical ordering.

Definition 2.21 (Canonical ordering)**.** Let G be a maximal plane graph with n vertices, and let u, v, w be the vertices on the boundary of its outer face. A *canonical ordering* Π is a permutation $v_1, \ldots, v_n$ of the vertices in G such that $v_1 = u$, $v_2 = v$, and $v_n = w$, and for every $3 \leq k \leq n$ the following conditions hold

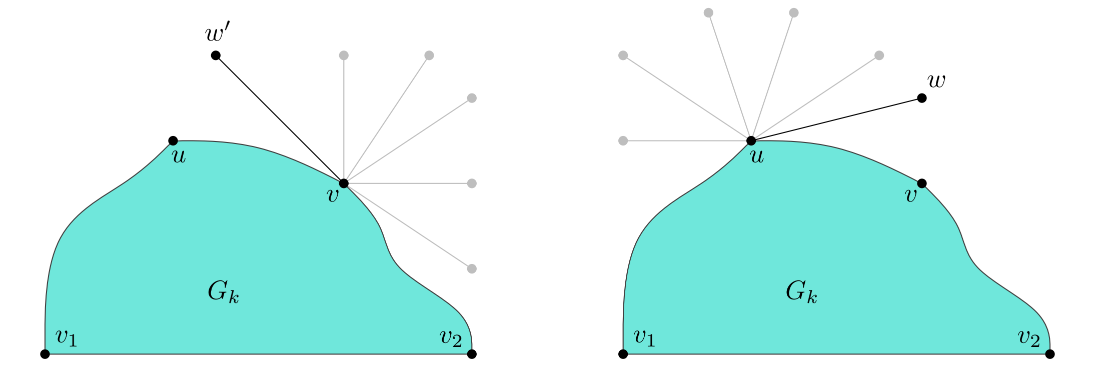
- The subgraph G_{k-1} of G induced by $v_1, \ldots, v_{k-1}$ is biconnected, and the boundary of its outer face is a cycle C_{k-1} containing the edge $\{v_1, v_2\}$.
- The vertex v_k is on the outer face of G_{k-1}, and has at least two neighbors in G_{k-1}. Moreover, all of its neighbors in G_{k-1} are consecutive on the path $C_{k-1} - \{u, v\}$.

We call C_{k-1} the *contour* of G_{k-1}.

De Fraysseix et al. [dFPP90] proof that there is always a canonical ordering for a maximal plane graph. It can be computed in linear time. Their algorithm for computing a canonical ordering works iteratively and top-down. It determines v_n first, then finds a suitable vertex that becomes v_{n-1}, then v_{n-2} and so on until we have a canonical ordering of the whole graph.

Biconnected canonical ordering

Harel and Sardas [HS98] introduce the *biconnected canonical ordering* as a generalization of the canonical ordering. Analog to the canonical ordering, this will be a permutation of



(a) The vertex w' has a left support on C_k. (b) The vertex w has a right support on C_k.

Fig. 2.8.: Schematic example of a legal support of a vertex on the induced subgraph G_k.

the vertices in a given embedded graph, where the first k vertices induce a subgraph G_k. The boundary list C_k of the outer face of G_k is called contour of G_k. We will also refer to the vertex list derived from this boundary list as contour. For their definition, they use the concept of a *left/right support* and a *legal support.* A vertex w on the outer face of G_k has a *right* support on C_k if both of the following conditions are fulfilled.

- There are vertices u and v that are neighbors on the contour of G_k, i.e., C_k and u immediately precedes v in C_k.
- w is adjacent to u.
- v immediately precedes w in the counter-clockwise circular order around u.

Analogously, we define a *left* support. We say a vertex has a *legal* support on C_k if it has a right or a left support on G_k. An example of a vertex with a left and a vertex with a right support is given in Fig. 2.8.

Definition 2.22 (Biconnected canonical ordering)**.** Let G be a biconnected plane graph with n vertices, and let $\{u, v\}$ be an edge that has its right side on the boundary list of the outer face of G. A *biconnected canonical ordering* Π is a permutation $v_1, \dots, v_n$ of the vertices in G such that $v_1 = u$, $v_2 = v$, and for every $2 \leq k \leq n$ the following conditions hold:

- The subgraph G_k of G induced by $v_1, \dots, v_k$ is connected, and C_k contains the right side of the edge $\{v_1, v_2\}$.
- All vertices in $G - G_k$ lie within the outer face of G_k.
- For $k > 2$, the vertex v_k has one or more neighbors in G_{k-1}. If v_k has exactly one neighbor in G_{k-1}, then it has a legal support on C_k.

They also give a linear-time algorithm for computing a biconnected canonical ordering of a biconnected plane graph. Like the algorithm for computing the canonical ordering, it works iteratively and in linear time. Unlike the algorithm for computing the canonical ordering, it works bottom-up. It determines v_1 and v_2 first, then finds a suitable vertex that becomes v_3, then v_4 and so on until we have a canonical ordering of the whole graph. The algorithm maintains three arrays during the computation. These are A, N, and F. Before the k-th step of the algorithm is executed, they save the following information. The array A is indexed by the faces and contains the number of edges of each face that are in G_{k-1}. The array N is indexed by the vertices and contains the number of neighboring vertices of each vertex that are in G_{k-1}. The array F is indexed by the vertices and contains the number of *ready* faces of which each vertex is the only missing vertex outside G_{k-1}. They say a face f is *ready* in G_{k-1} if all but one vertex of f is in G_{k-1}. At the end of each iteration, these arrays are updated according to the choice of v_k. This algorithm is noted down in Algorithm 2. Note that the array A is queried only for updating the array F.

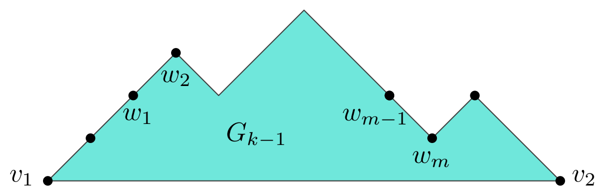
Algorithm 2: Compute a biconnected canonical ordering according to Harel and Sardas [HS98]

```
   Input  : Biconnected plane graph $(G, \mathcal{E}(G))$ with $n$ vertices
   Output: Biconnected canonical ordering $\Pi$ of $(G, \mathcal{E}(G))$
   /* Initialization                                              */
1  Initialize all three arrays $A$, $N$, and $F$ to 0
2  Take as $\{v_1, v_2\}$ any edge of the outer face
3  Set $A(f)$ to 1 for $f$, the left face of $\{v_1, v_2\}$
4  If $f$ is a triangle with vertices $v_1, v_2, v_3$, set $F(v_3)$ to 1, since $f$ is ready
   /* Iterative computation of $v_k$                                 */
5  for $k = 3$ to $n$ do
6      if there is a vertex $v \notin V[G_{k-1}]$ with $N(v) \geq 2$ and $N(v) = F(v) + 1$ then
7          $v_k := v$
8      else
9          Find a $v \notin V[G_{k-1}]$ with a legal support on $C_{k-1}$ and $N(v) = 1$
10         $v_k := v$
11     Update the data structures $A$, $N$, $F$ for $v_k$
12 return $\Pi := (v_1, v_2, \ldots, v_n)$
```

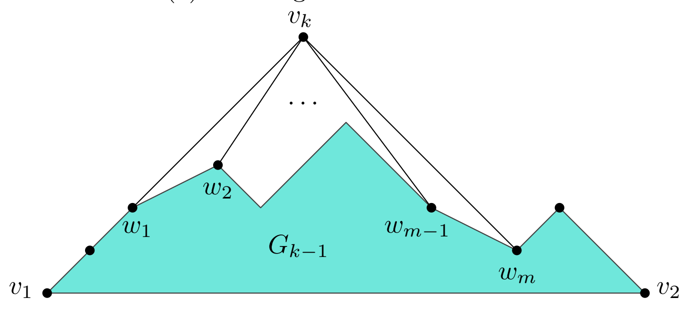
Drawing phase

Chrobak and Payne [CP95] present a linear time algorithm for the drawing phase of the Shift Algorithm. The algorithm takes an embedded maximal planar graph with n vertices and a canonical ordering $(v_1, \ldots, v_n)$ of it as input and returns a crossing-free straight-line drawing on a grid of size $(2n-4) \times (n-2)$. It works iteratively.

Initially, it places v_1, v_2, and v_3 on a triangle. Namely, it places v_1 onto $(0, 0)$, v_2 onto $(2, 0)$, and v_3 onto $(1, 1)$. Thereafter, it adds iteratively each vertex v_k with $k > 3$ to the drawing. It maintains an invariant that is fulfilled at the beginning and at the end



(a) Drawing before iteration k.



(b) Drawing after iteration k.

Fig. 2.9.: Example of an iteration of the drawing phase of the Shift Algorithm. Here, vertex v_k is added to the drawing.

of each iteration. This invariant guarantees that the contour consists of line segments with only slope $+1$ and -1 except for the line segment of the edge $\{v_1, v_2\}$ which is drawn as horizontal line segment on the bottom of the drawing. This makes the drawing look like a mountainous landscape. When we add v_k, it is adjacent to a consecutive sequence $w_1, \dots, w_m$ of vertices on the contour. We shift w_1 and everything beneath it and everything to the left of it to the left by one unit and we shift w_m and everything beneath it and everything to the right of it to the right by one unit relative to $w_2, \dots, w_{m-1}$. Now, we place v_k on the intersection point of the line with slope 1 through w_1 and the line with slope -1 through w_m. We can show that this intersection point is also a grid point. Now, after the k-th step, w_1, v_k, and w_m are consecutive on the contour of the current drawing, while $w_2, \dots, w_{m-1}$ is beneath v_k. We say that the vertices $w_2, \dots, w_{m-1}$ are *covered* by v_k or that they are in the *underset* of v_k. They are not moved relative to v_k later on (they are shifted in the same direction and by the same value if they are shifted). By the maintenance of the invariant, we can show that such a drawing is a crossing free grid drawing. For an illustration of such a step, see Fig. 2.9.

The drawing phase of the Shift Algorithm that uses a biconnected canonical ordering instead of a canonical ordering differs only slightly. In the case where we add a vertex v_k with a legal support, we proceed as if there was the edge between v_k and the vertex via which v_k has a legal support. In this thesis, we let `shiftDrawHS(`$(G, \mathcal{E}(G))$`, `Π`)` denote the drawing phase of the Shift Algorithm for a biconnected plane graph $(G, \mathcal{E}(G))$ and a biconnected canonical ordering Π.

3. Constructing 1-Bend RAC Drawings of NIC-Plane Graphs in Quadratic Area

In this chapter, we will show that every NIC-plane graph with n vertices admits a RAC drawing on a grid of size $O(n) \times O(n)$ with at most one bend per edge. "On a grid" means here that every vertex, every bend point, and every crossing point is drawn on a grid point of a regular integer grid of the specified size. Because every IC-plane graph is also a NIC-plane graph, this holds for IC-plane graphs as well. The proof is constructive. We describe an efficient algorithm that takes a NIC-plane graph as input and returns a 1-bend RAC drawing of this graph on a grid of size at most $(16n - 32) \times (8n - 16)$ as output.

This leads us to the main result of this chapter:

Theorem 3.1. *Every NIC-planar graph $G = (V, E)$ admits a NIC-planar 1-bend RAC drawing with vertices, bend points, and crossing points on a grid of size at most $(16n - 32) \times (8n - 16)$, where $n := |V|$. Given a NIC-planar embedding $\mathcal{E}(G)$ of G, such a drawing can be computed in $O(n)$ time. The returned drawing preserves the given embedding $\mathcal{E}(G)$.*

To prove this theorem, we will draw an augmentation of the given NIC-plane graph. An essential ingredient of our proof is the construction of a specialized canonical ordering of this augmented graph.

We will present an algorithm in Section 3.1 with the previously described properties. We will show the correctness of this algorithm in Section 3.2. In Section 3.3, we will show that this algorithm runs in linear time and everything in the output drawing is on a grid of quadratic size. Putting this together, we will finally proof Theorem 3.1 in Section 3.4. We have also implemented the algorithm in Java. An example of an output drawing is given in Appendix A.

3.1. Algorithm

This section describes how we produce the desired 1-bend RAC drawings. The high-level structure of the presented algorithm is as follows. It is also written down in pseudocode in Algorithm 3.

- Make the graph biconnected and replace each pair of crossing edges by a divided quadrangle. Let the resulting graph be G'. We describe this step in more detail in Section 3.1.1 and have it in pseudocode in Algorithm 4.

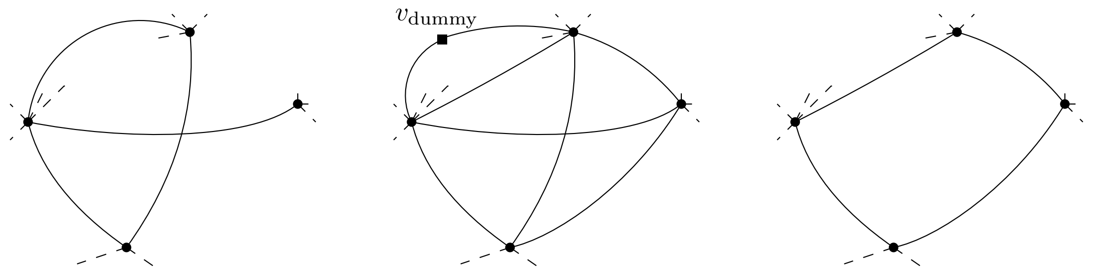
- Generate a biconnected canonical ordering $\hat{\Pi}$ of an augmentation $\hat{G}$ of G'. For this step, see Section 3.1.2 and Algorithm 5.
- Draw $\hat{G}$ according to $\hat{\Pi}$ using the algorithm by Harel and Sardas [HS98]. Let the resulting drawing be $\hat{\Gamma}$.
- Scale up $\hat{\Gamma}$ by a factor of 2 in each dimension and obtain $\hat{\Gamma}_{\times 2}$.
- In $\hat{\Gamma}_{\times 2}$, replace each divided quadrangle by an empty quadrangle and insert the original edges with one bend per edge inside the empty quadrangles so that they cross in a right angle. Let the resulting drawing be Γ^*. In more detail, this step is described in Section 3.1.3 and in Algorithm 6.
- Remove all previously inserted dummy edges and make the remaining dummy vertices bend points. The resulting drawing Γ is the final drawing.

Algorithm 3: Drawing a NIC-plane graph on a grid of a size that is quadratic in the number of vertices

```
  Input  : NIC-plane graph $(G = (V, E), \mathcal{E}(G))$
  Output: 1-bend RAC drawing of $G$ according to $\mathcal{E}(G)$
  /* Modify the graph via the preprocessing.  Afterwards, $Q$ is a list
     of empty quadrangles marking the original crossings.             */
1 $((G', \mathcal{E}'(G')), Q) :=$ preprocessNICPlaneGraph($(G, \mathcal{E}(G))$)
  /* Compute the biconnected canonical ordering.  In this step the
     embedded graph $(G', \mathcal{E}'(G'))$ is slightly modified.  The resulting
     embedded graph is $(\hat{G}, \hat{\mathcal{E}}(\hat{G}))$.  $\hat{\Pi}$ is the biconnected canonical
     ordering of the vertices in this embedded graph.                 */
2 $((\hat{G}, \hat{\mathcal{E}}(\hat{G})), \hat{\Pi}) :=$ modifiedBiconnectedCanonicalOrdering($(G', \mathcal{E}'(G'))$, $Q$)
  /* Draw according to the computed biconnected canonical ordering
     using the drawing step of the Shift Algorithm.  See Section 2.4.2
     */
3 . $\hat{\Gamma} :=$ shiftDrawHS($(\hat{G}, \hat{\mathcal{E}}(\hat{G}))$, $\hat{\Pi}$)
  /* Refining by a factor of 2 means here, scaling up both coordinates
     of each vertex by a factor of 2                                   */
4 $\hat{\Gamma}_{\times 2} := \hat{\Gamma}$ after refining its grid by a factor of 2
  /* Reinsert the crossing edges into the quadrangles saved in $Q$ with
     a bend as RAC crossings.                                          */
5 $\Gamma^* :=$ reinsertCrossingEdges($\hat{\Gamma}_{\times 2}$, $Q$)
6 $\Gamma := \Gamma^*$ after removing all dummy edges and replacing each dummy vertex by a
    bend point
7 return $\Gamma$
```

Algorithm 4: [Subroutine of Algorithm 3]
preprocessNICPlaneGraph($(G, \mathcal{E}(G))$)

```
   Input  : NIC-plane graph $(G, \mathcal{E}(G))$
   Output: (Biconnected plane graph $(G', \mathcal{E}'(G'))$,
            list of empty quadrangles $Q$)
 1 Let $Q$ be an empty list
 2 foreach crossing $c \in \mathcal{E}(G)$ do
 3     if there is no empty kite around $c$ then
 4         Insert dummy edges $D$ so that there is an empty kite around $c$
 5         foreach $d \in D$ do
 6             if there is a parallel edge $e$ to $d$ in $G$ then
 7                 Subdivide the edge $e$ via a dummy vertex
 8     Remove the two edges crossing in $c$ from the graph and the embedding
 9     Add the remaining empty quadrangle to $Q$
10 Let the resulting plane graph be $(G^+, \mathcal{E}^+(G^+))$
11 $(G', \mathcal{E}'(G'))$ := makePlaneGraphBiconnected($(G^+, \mathcal{E}^+(G^+))$) // call Algo. 1
12 return $((G', \mathcal{E}'(G'))$, $Q)$
```



(a) Crossing as it initially appears.
(b) Empty kite with subdivided original edge.
(c) Empty quadrangle.

Fig. 3.1.: Modifying an edge crossing in the preprocessing step (s. Section 3.1.1). At a crossing, we build an empty kite and remove the crossing edges afterwards to obtain an empty quadrangle.

```
Algorithm 5: [Subroutine of Algorithm 3]
modifiedBiconnectedCanonicalOrdering($(G', \mathcal{E}'(G'))$, $Q$)
 Input  : Biconnected plane graph $(G', \mathcal{E}'(G'))$,
          list $Q$ of empty quadrangles in $G'$
 Output: (Biconnected plane graph $(\hat{G}, \hat{\mathcal{E}}(\hat{G}))$,
          biconnected canonical ordering $\hat{\Pi}$ of the vertices in $\hat{G}$)
1  Do the same initialization as in the algorithm by Harel and Sardas [HS98]
2  Define index($v_1$) := 1 and index($v_2$) := 2
3  for $k = 1$ to $n$ do
4      if $k \geq 3$ then
           // $v_1$ and $v_2$ have been determined in the initialization
5          Compute $v_k$ as in the algorithm by Harel and Sardas
6          Define index($v_k$) := $k$
7      foreach empty or divided quadrangle $q = (v_k, a, b, c) \in Q$ containing $v_k$ do
8          if $v_k$ is the first vertex of $q$ encountered yet then
9              Add a dummy edge from $v_k$ to $b$ // $q$ was an empty quadrangle and
                   is a divided quadrangle now
10         if $v_k$ is the last vertex of $q$ encountered and $b$ is not the first vertex
             encountered in $q$ then
11             if $b$ is in the underset of $a$ or $c$ then
                   // We are in Case 2
12                 Insert a subdivided dummy edge $\{a, c\}$ into the face $(v_k, a, c)$
13                 Let the dummy vertex of this subdivision be $v_{\text{shift}}$
14                 Insert $v_{\text{shift}}$ into the biconnected canonical ordering before $v_k$
15             else
                   // We are in Case 3
16                 $v_{\text{lowest}} := \begin{cases} a, \text{ if index}(a) < \text{index}(c) \\ c, \text{ otherwise} \end{cases}$
17                 Subdivide the edge $\{v_{\text{lowest}}, b\}$ by $v_{\text{dummy}}$
18                 Insert $v_{\text{dummy}}$ into the biconnected canonical ordering before $b$
19                 Add a new dummy edge $\{v_{\text{lowest}}, b\}$ as bounding edge for $q$
20         Update the data structures for computing the biconnected canonical
             ordering by including the new edges and faces
21 Let the modified plane graph be $(\hat{G}, \hat{\mathcal{E}}(\hat{G}))$
22 Let the resulting biconnected canonical ordering be $\hat{\Pi}$
23 return $((\hat{G}, \hat{\mathcal{E}}(\hat{G}))$, $\hat{\Pi})$
```

Algorithm 6: [Subroutine of Algorithm 3] reinsertCrossingEdges($\hat{\Gamma}_{\times 2}$, Q)

```
Input  : Planar drawing $\hat{\Gamma}_{\times 2}$ of the graph $\hat{G}$,
         list $Q$ of divided quadrangles in $\hat{\Gamma}_{\times 2}$ where RAC crossings will be
         inserted
Output: 1-bend RAC drawing of $\hat{G}$
1  foreach divided quadrangle $(t_1, t_2) = ((a,b,c),(a,c,d))$ in $Q$ with
     index$(a)$ < index$(v_i)$ for each $i \in \{b,c,d\}$ do
2      Remove the dummy edge $\{a,c\}$
3      Remove a shift vertex $v_{\text{shift}}$ and its incident dummy edges if present
       // We have now an empty quadrangle $(a,b,c,d)$
       // Insert the original crossing edges in three cases
4      $v_{\text{lower}} := \begin{cases} b, \text{ if } y(b) < y(d) \\ d, \text{ otherwise} \end{cases}$
5      if index$(c)$ > index$(v_i)$ for each $i \in \{b,d\}$ then
           // Case 1
6          Insert the edge $\{a,c\}$ with a bend point at $(x(a), y(v_{\text{lower}})+1)$
7          $x_{\text{bend}} := \begin{cases} x(a)-1, \text{ if } v_{\text{lower}} = b \\ x(a)+1, \text{ otherwise} \end{cases}$
8          Insert the edge $\{b,d\}$ with a bend point at $(x_{\text{bend}}, y(v_{\text{lower}}))$
9      else
10         if $v_{\text{lower}}$ is in the underset of $c$ then
               // Case 2
11             if $v_{\text{lower}} = b$ then
12                 $x_{\text{cross}} := 0.5(x(c) - y(c) + x(v_{\text{lower}}) + y(v_{\text{lower}}))$
13                 $y_{\text{cross}} := 0.5(-x(c) + y(c) + x(v_{\text{lower}}) + y(v_{\text{lower}}))$
14                 Insert the edge $\{a,c\}$ with a bend point at $(x_{\text{cross}}-1, y_{\text{cross}}-1)$
15                 Insert the edge $\{b,d\}$ with a bend point at $(x_{\text{cross}}-1, y_{\text{cross}}+1)$
16             else
17                 $x'_{\text{cross}} := 0.5(x(c) + y(c) + x(v_{\text{lower}}) - y(v_{\text{lower}}))$
18                 $y'_{\text{cross}} := 0.5(x(c) + y(c) - x(v_{\text{lower}}) + y(v_{\text{lower}}))$
19                 Insert the edge $\{a,c\}$ with a bend point at $(x'_{\text{cross}}+1, y'_{\text{cross}}-1)$
20                 Insert the edge $\{b,d\}$ with a bend point at $(x'_{\text{cross}}+1, y'_{\text{cross}}+1)$
21         else
               // Case 3
22             Insert the edge $\{a,c\}$ with a bend point at $(x(c), y(v_{\text{lower}})-1)$
23             $x_{\text{bend}} := \begin{cases} x(c) - (y(c) - y(b)), \text{ if } v_{\text{lower}} = b \\ x(c) + (y(c) - y(d)), \text{ otherwise} \end{cases}$
24             Insert the edge $\{b,d\}$ with a bend point at $(x_{\text{bend}}, y(v_{\text{lower}}))$
25 return the modified drawing
```

3.1.1. Preprocessing

In this step, we aim to make the given graph planar and biconnected so that we can apply the algorithm of Harel and Sardas [HS98] to it. We make it planar by replacing each pair of crossing edges by an empty quadrangle. We save each such quadrangle in a list Q so that we can reinsert the crossing edges at the right spot later. This step of our algorithm is formulated in pseudocode in Algorithm 4 and described in detail below.

At each crossing, we first create empty kites and then remove the two crossing edges so that empty quadrangles remain. To accomplish this, we consider each crossing and add dummy edges where necessary so that each crossing induces an empty kite. If adding such a dummy edge leads to parallel edges because this edge is already present in a different position in the planarized embedding, we instead subdivide this original edge using a dummy vertex and add the dummy edge afterwards. In Procedure 2.9, we define how to subdivide an edge. After creating empty kites, we remove the two crossing edges. We save each emerging empty quadrangle in a list Q in order to remember where to reinsert the crossings in the reinsertion step of this algorithm. This reinsertion step is described in Section 3.1.3. An example of applying this preprocessing step to a crossing is given in Fig. 3.1. After this, we have a plane graph $(G^+, \mathcal{E}^+(G^+))$.

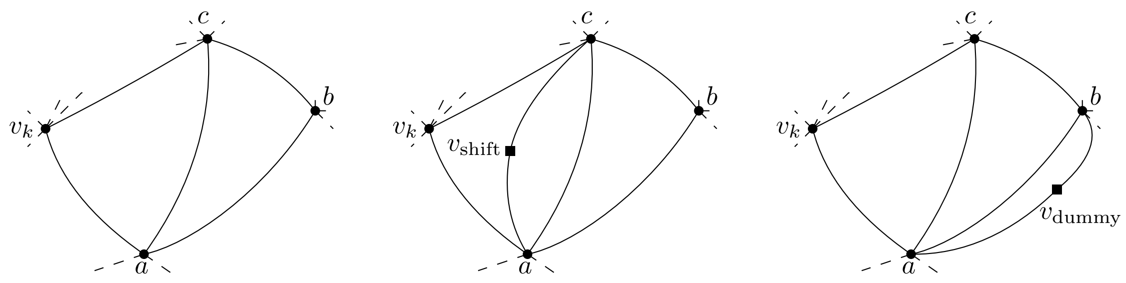
In the last step of the preprocessing, we make the graph biconnected via the algorithm described in Section 2.4.1. Let the resulting biconnected plane graph be $(G' = (V', E'), \mathcal{E}'(G'))$.

3.1.2. Computing the biconnected canonical ordering

The idea of this step is computing a biconnected canonical ordering of the vertices of the plane graph $(G' = (V', E'), \mathcal{E}'(G'))$ while the graph is modified. The modification is here an insertion of edges into the empty quadrangles so that we get divided quadrangles. This step is described in pseudocode in Algorithm 5. The biconnected canonical ordering was first introduced by Harel and Sardas [HS98]. We describe it in Section 2.4.2. We do the same initialization as they do. We also compute the ordering from the lower ordering numbers (vertices with them are drawn earlier) to the higher ordering numbers (drawn later). We use the term *index* for the position of a vertex in the biconnected canonical ordering. The computation of each index itself is not changed. Instead, some additional work is performed after the next vertex is determined.

Let the recently computed vertex index be k and let it have been assigned to v_k. The vertex v_k may be a vertex of one or more empty or divided quadrangles from Q. We consider each quadrangle $q = (v_k, a, b, c)$ containing v_k and we modify the graph in the following three situations.

First, if v_k is the first of the four vertices of q to be encountered, insert a dummy edge from v_k to the opposing vertex b. We say b is the *opposing vertex* of v_k in the quadrangle q if they are different vertices and not adjacent to each other in the empty quadrangle (obtained from) q. Inserting this diagonal into the empty quadrangle makes it a divided quadrangle. The list Q now also contains divided quadrangles. An example of a quadrangle after this insertion is given in Fig. 3.2a.



(a) Divided quadrangle. A quadrangle of Case 1 of the reinsertion step (s. Section 3.1.3) would be like this.

(b) Divided quadrangle where a shift vertex v_{shift} has been inserted below v_k. This is a quadrangle of Case 2 of the reinsertion step (s. Section 3.1.3).

(c) Divided quadrangle where $\{a, b\}$ has been subdivided and another edge $\{a, b\}$ has been inserted. This is a quadrangle of Case 3 of the reinsertion step (s. Section 3.1.3).

Fig. 3.2.: Modifications of quadrangles that emerge while the modified biconnected canonical ordering is computed.

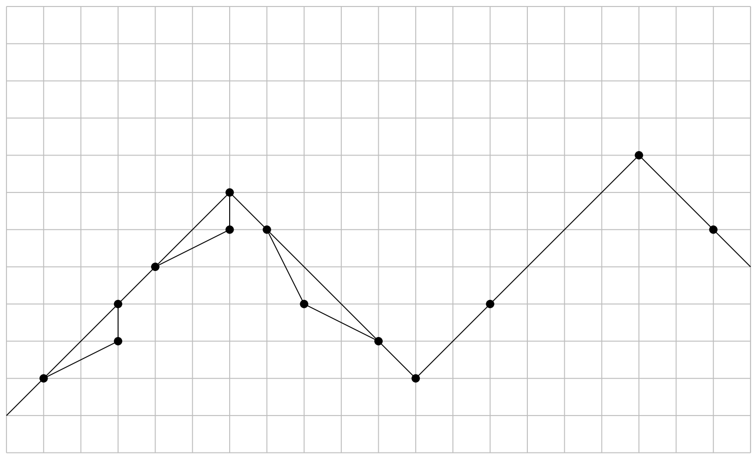
Second, we continue if ...

- v_k has the largest index among the vertices in q,
- b does not have the smallest index among the vertices in q, and
- b is in the underset of a or c.

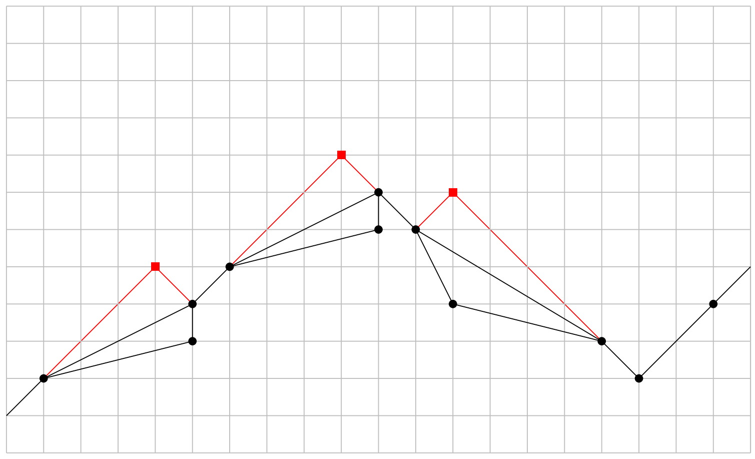
If this is fulfilled, q matches Case 2 of the reinsertion step (s. Section 3.1.3). For the edge reinsertion, we will need an extra shift in the drawing phase that shifts the first vertex of the quadrangle away from the third vertex. To accomplish this, we insert a dummy vertex adjacent to a and c that will assure that we shift c away from a by 2 units when we draw the graph. We will refer to such a dummy vertex as *shift vertex*. Shifting vertices away from each other is its only purpose and we will remove it and its two incident dummy edges before we reinsert the crossing edges. In the embedding, we insert this vertex v_{shift} into the face (v_k, a, c) and in the biconnected canonical ordering, we insert it before v_k. An example of a quadrangle after this insertion is given in Fig. 3.2b. We sill consider q as divided quadrangle, although it contains a shift vertex and two incident dummy edges now. An example of the effect of a shift vertex in the drawing phase is given in Fig. 3.3. The shift vertices and incident dummy edges are colored in red there. Note that v_k can be the last vertex of several quadrangles in Case 2 like in this example.

Third, we perform an additional insertion if q matches Case 3 of the reinsertion step (s. Section 3.1.3) and v_k is drawn on the top of q. Namely, we continue if ...

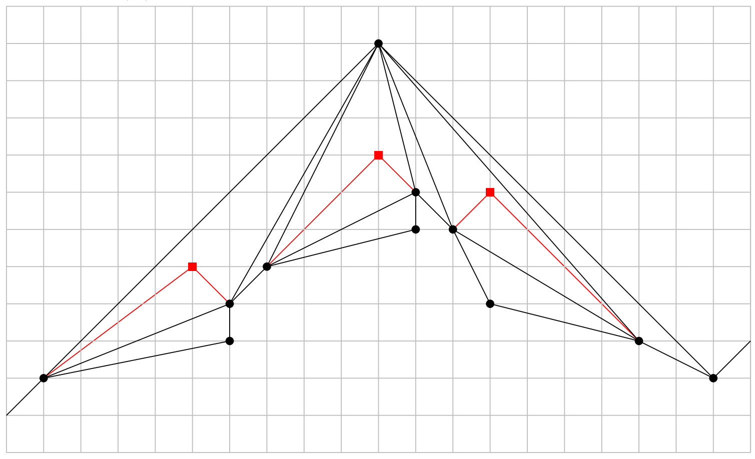
- v_k has the largest index among the vertices in q,
- b does not have the smallest index among the vertices in q, and
- b is not in the underset of a or c.



(a) After three vertices of three divided quadrangles but no shift vertices have been drawn.



(b) After the shift vertices have been drawn.



(c) The last vertex of the three divided quadrangles is added with a regular shift of the Shift Algorithm.

Fig. 3.3.: Example of a vertex v_k that is the last vertex of more than one quadrangle in Case 2. Shift vertices and incident edges are colored in red.

If these three conditions are all true, subdivide the edge between the vertex with the lowest index v_{lowest} in q, which is a or c, and b via a dummy vertex v_{dummy}. In this quadrangle, b has the second lowest index. Insert v_{dummy} into the biconnected canonical ordering before b. After this, add a dummy edge $\{v_{\text{lowest}}, b\}$ to the embedded graph so that it is a bounding edge of this divided quadrangle. It may seem to be pointless to first subdivide this edge and then reinsert it, but we need both of them for the step in which the crossing edges are reinserted, and if $\{v_{\text{lowest}}, b\}$ is an original edge, we will draw it with a bend. An example of a quadrangle after this insertion is given in Fig. 3.2c.

After each insertion of dummy vertices and dummy edges, we update the data structures that are maintained for the computation of the biconnected canonical ordering. After the computation of this ordering has been completed, we have a biconnected canonical ordering $\hat{\Pi}$ of the resulting plane graph $(\hat{G}, \hat{\mathcal{E}}(\hat{G}))$ that is based on the plane graph $(G', \mathcal{E}'(G'))$.

Using the computed biconnected canonical ordering, we draw the graph $\hat{G}$ according to $\hat{\mathcal{E}}(\hat{G})$ with the adaption of the Shift Algorithm by Harel and Sardas [HS98]. We obtain the drawing $\hat{\Gamma}$ and we scale up this drawing by a constant factor of 2 in both dimensions. This is equivalent to refining the underlying grid by a factor of 2 in both dimensions. We call the resulting drawing $\hat{\Gamma}_{\times 2}$.

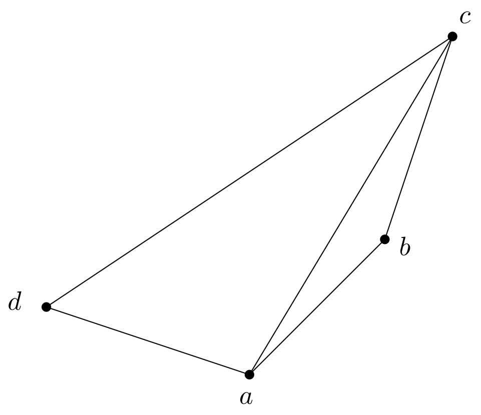
3.1.3. Reinserting the crossing edges

Currently, we have a drawing $\hat{\Gamma}_{\times 2}$ of $\hat{G}$ that is derived from the original graph G. Compared to G, it has some extra edges and vertices but also one edge per crossing missing. This makes the embedded graph plane and not only NIC-plane. In this step, we reinsert every missing edge that crosses another edge of the graph. The procedure described here is formulated in pseudocode in Algorithm 6. Our goal is a 1-bend RAC drawing, i.e., each crossing forms a right angle and each edge consist of at most two straight segments.

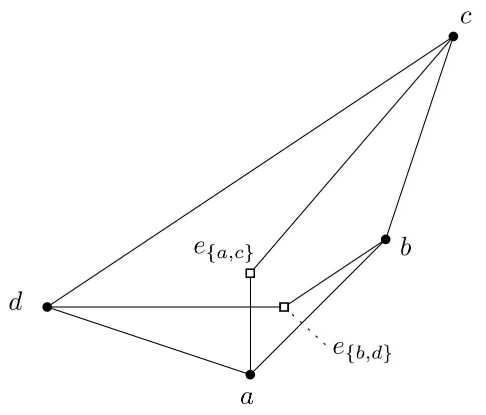
We consider each divided quadrangle q of the list Q—these induce the crossings in the original drawing. Currently, q is a quadrangle with a diagonal in its interior. We remove the diagonal (an edge between two opposing vertices of the divided quadrangle) and a possibly present shift vertex including its two incident dummy edges. Thereafter, we have an empty quadrangle $q = (a, b, c, d)$ where a has the lowest index in $\hat{\Pi}$ and (a, b, c, d) is a cycle in counter-clockwise vertex order in the drawing. Observe that a has a smaller y-coordinate than the other vertices in q because it has a lower index in $\hat{\Pi}$ and is adjacent to b, c, and d. Moreover, we define v_{lower} as b if $y(b) < y(d)$ and otherwise we define v_{lower} as d.

In the following, we use $x(v)$ and $y(v)$ to describe the x- and y-coordinate of v, respectively. We distinguish three cases for each quadrangle that we consider. In this chapter, we will refer to the bend point of the edge $\{a, c\}$ as $e_{\{a,c\}}$ and to the bend point of the edge $\{b, d\}$ as $e_{\{b,d\}}$ in any case.

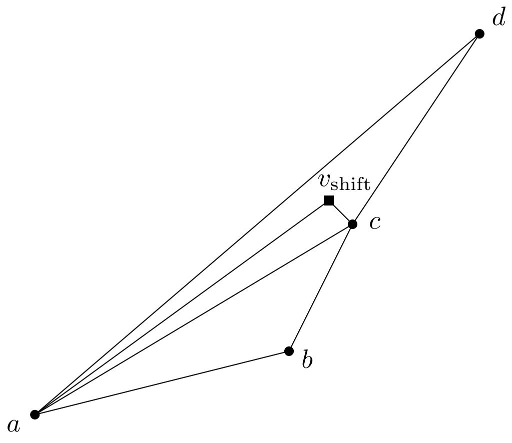
Case 1 In this case, c has the largest index among the four vertices of q in $\hat{\Pi}$. An example of such a quadrangle is given in Fig. 3.4a. We will have a crossing point of the inserted edges at $(x(a), y(v_{\text{lower}}))$. To accomplish this, we insert the edge $\{a, c\}$ with a



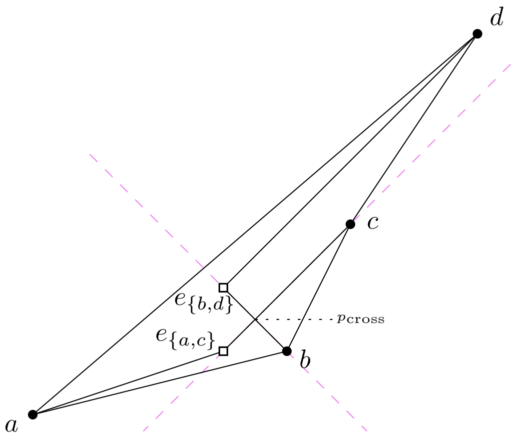
(a) Divided quadrangle in Case 1.



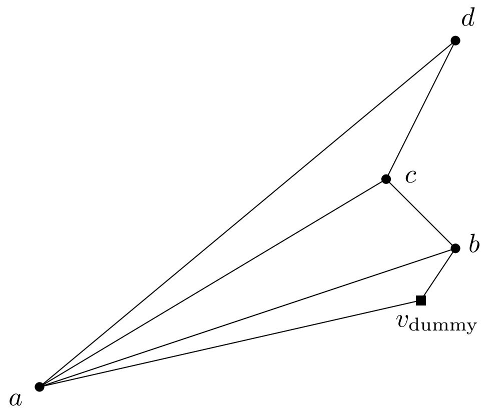
(b) Case 1 after inserting the crossing edges.



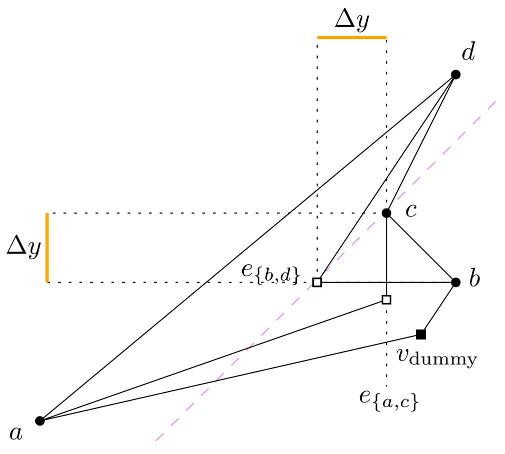
(c) Divided quadrangle in Case 2.
The vertex b is covered by c.



(d) Case 2 after inserting the crossing edges.



(e) Divided quadrangle and the neighboring path $(a, v_{\text{dummy}}, v_{\text{lower}} = b)$ in Case 3.
The vertex b is not covered by c.



(f) Case 3 after inserting the crossing edges.

Fig. 3.4.: Examples for the three cases appearing in Algorithm 6. Lines for orientation with slope 1 or −1 are dashed and colored in violet.

bend point at $(x(a), y(v_{\text{lower}}) + 1)$ and we insert the edge $\{b, d\}$ with a bend point at $e_{\{a,c\}} = (x(a) - 1, y(b))$ if $v_{\text{lower}} = b$ or with a bend point at $(x(a) + 1, y(d))$ if $v_{\text{lower}} = d$. Observe that the subcase $v_{\text{lower}} = b$ is the same as $v_{\text{lower}} = d$ if we mirror the quadrangle q across a vertical axis. An application of this reinsertion operation to Fig. 3.4a is given in Fig. 3.4b.

If c does not have the largest index among the four vertices of q in $\hat{\Pi}$, we distinguish two more cases.

Case 2 In this case, b or d has the largest index among the four vertices in q in $\hat{\Pi}$ and the other one, i.e., v_{lower} is in the underset of c. An example of such a quadrangle is given in Fig. 3.4c. We assume that we are in subcase $v_{\text{lower}} = b$. We define the crossing point $p_{\text{cross}} = (x_{\text{cross}}, y_{\text{cross}})$ as the intersection point of the lines with slope 1 and -1 through c and b, respectively. The coordinates of this crossing point are $x_{\text{cross}} := 0.5(x(c) - y(c) + x(b) + y(b))$ and $y_{\text{cross}} := 0.5(-x(c) + y(c) + x(b) + y(b))$. Despite multiplying each coordinate by a factor of 0.5, both coordinates are integer values since we have refined the grid by a factor of 2. We place the two bend points onto the same lines on the grid points that are next to p_{cross}. In other words, we insert the edge $\{a, c\}$ with a bend point at $(x_{\text{cross}} - 1, y_{\text{cross}} - 1)$ and we insert the edge $\{b, d\}$ with a bend point at $(x_{\text{cross}} - 1, y_{\text{cross}} + 1)$. An application of this operation to Fig. 3.4c is given in Fig. 3.4d.

Again, the subcase $v_{\text{lower}} = d$ is symmetric and can be obtained by mirroring the quadrangle q across a vertical axis. For completeness, we give the coordinates of p_{cross}, $e_{\{a,c\}}$, and $e_{\{b,d\}}$ here as well. The ones of p_{cross} are $x'_{\text{cross}} = 0.5(x(c) + y(c) + x(d) - y(d))$ and $y'_{\text{cross}} = 0.5(x(c) + y(c) - x(d) + y(d))$. The edge $\{a, c\}$ is bent at at $(x_{\text{cross}} + 1, y_{\text{cross}} - 1)$ and the edge $\{b, d\}$ is bent at $(x_{\text{cross}} + 1, y_{\text{cross}} + 1)$.

Case 3 In this case, b or d has the largest index among the four vertices of q in $\hat{\Pi}$ and the other one, i.e., v_{lower} is not in the underset of c. Note, that the edge $\{a, v_{\text{lower}}\}$ is a dummy edge, that we inserted during the computation of $\hat{\Pi}$, and next to this edge in the embedding and between the same two vertices, we have a path of length 2. This path has been the former edge $\{a, v_{\text{lower}}\}$ when we had empty kites around each edge crossing. Hence, it is either a subdivided original edge or two dummy edges and a dummy vertex. An example of such a quadrangle is given in Fig. 3.4e. We will have a crossing point of the inserted edges at $(x(c), y(v_{\text{lower}}))$. We assume that we are in subcase $v_{\text{lower}} = b$.

For the edge reinsertion, we proceed as follows. First, we remove the dummy edge $\{a, b\}$. We bend the edge $\{b, d\}$ on the line with slope 1 through c at $y = y(b)$ because from this point we always "see" d inside q. So, we define $\Delta y = (y(c) - y(b))$ and $x_{\text{bend}} := x(c) - \Delta y$. Second, we insert the edge $\{a, c\}$ with a bend point at $(x(c), y(b) - 1)$ and we insert the edge $\{b, d\}$ with a bend point at $(x_{\text{bend}}, y(b))$. For the symmetric subcase $v_{\text{lower}} = d$, the x-coordinate of $e_{\{b,d\}}$ is $x'_{\text{bend}} = x(c) + (y(c) - y(d))$. An application of this operation to Fig. 3.4e is given in Fig. 3.4f.

3.2. Correctness

In this section, we analyze our algorithm. We make sure that we get a NIC-planar 1-bend RAC drawing in any case. To this end, we establish some Lemmas which form the foundation of our proof of Theorem 3.1. These Lemmas are subdivided into three subsections. In Section 3.2.1, we show that the biconnected canonical ordering which we compute is valid, although we modify the graph during the computation. In Section 3.2.2, we show that the reinserted crossing edges cross each other in a right angle, but cross no other edge. In Section 3.2.3, we show some properties about the finally returned drawing.

3.2.1. Computing the biconnected canonical ordering

The graph is modified while we compute the biconnected canonical ordering. We state and show that this provides a valid biconnected canonical ordering of the modified graph.

Lemma 3.2. The computation of the biconnected canonical ordering in Algorithm 5 yields a valid biconnected canonical ordering $\hat{\Pi}$ of the resulting graph $\hat{G}$.

Proof. The graph is modified in two ways in Algorithm 5:

1. Adding an edge as a diagonal for each quadrangle of Q.
2. Adding a vertex and two incident edges for each quadrangle in Q that fulfills the conditions of Case 2 and Case 3. In Case 2 we add a shift vertex v_{shift}, and in Case 3 we add a dummy vertex v_{dummy}.

Our argument for the first modification is that Algorithm 2, which computes a biconnected canonical ordering, would return the same biconnected canonical ordering as we do if it had the modified graph with all diagonals inserted as input. The reason is that we change by modification 1 only areas of the graph that the algorithm has not reached yet. To show this, we have a closer look on the data structures maintained during the computation. The algorithm of Harel and Sardas [HS98] maintains the three arrays A, N, and F during the computation. For a description of them and the algorithm for computing a biconnected canonical ordering, see Section 2.4.2.

Consider a quadrangle $q \in Q$ into which we insert an edge as diagonal when we encounter the first vertex of q in step k. Regardless of whether this edge is present or not, after step $k-1$ every array would contain the same values. An additional edge as a diagonal does not increase the number of neighboring vertices in $\hat{G}_{k-1}$ for any vertex of q in N because after step $k-1$, no vertex of q is in $\hat{G}_{k-1}$. Analogously, no additional faces will be considered ready since neither the complete quadrangular face (without diagonal) nor the two triangular faces (with diagonal) are ready after step $k-1$. The set of faces differs depending on whether we have this diagonal or not. But the value of the complete quadrangular face (without diagonal) would be 0 in A as well as the value of both triangular faces (with diagonal) since no vertex of q (and therefore also no edge) is in $\hat{G}_{k-1}$. So, we do the same computation as if we initially had the graph with all diagonals being inserted.

For the second modification, we distinguish Case 2 and Case 3. In Case 2, v_{shift} can always be inserted into the biconnected canonical ordering before the top vertex v_{top} of a divided quadrangle $q = (a, b, c, v_{\text{top}}) \in Q$. Assume that v_{shift} is the k-th vertex in $\hat{\Pi}$ and that the conditions of Definition 2.22 are fulfilled for $k-1$. In particular, note that if v_{shift} was not within the outer face of $\hat{G}_{k-1}$, v_{top} would not be there either. Since v_{shift} has two adjacent neighbors in $\hat{G}_{k-1}$, i.e., a and c, the graph $\hat{G}_k$ is connected. The face (a, c, v_{shift}) does not contain a vertex. Hence, all vertices in $\hat{G} - \hat{G}_k$ lie within the outer face of $\hat{G}_k$.

In Case 3, we can argue similarly. Without loss of generality, assume that v_{dummy} was inserted next to a quadrangle $q = (v_{\text{lowest}}, v_{\text{lower}}, c, v_{\text{top}})$ and is adjacent to v_{lowest} and v_{lower}. Among the vertices in q, v_{lowest} has the lowest index in $\hat{\Pi}$ and v_{lower} the second lowest. Since v_{dummy} is inserted into $\hat{\Pi}$ before v_{lower}, the index of v_{dummy} in $\hat{\Pi}$ is in between the indices of v_{lowest} and v_{lower}. Analogously, we assume that v_{dummy} is the k-th vertex in the biconnected canonical ordering and that the conditions of Definition 2.22 are fulfilled for $k-1$. Since v_{dummy} is adjacent to v_{lowest}, which is in $\hat{G}_{k-1}$, $\hat{G}_k$ is connected as well. Clearly, all vertices of $\hat{G} - \hat{G}_k$ lie within the outer face of $\hat{G}_k$. The vertex v_{dummy} has exactly one neighbor in $\hat{G}_{k-1}$. Now, we show that it has a legal support. Observe that v_{lower} does not have a legal support inside q, since its opposing vertex v_{top} is added later. So, either v_{lower} is the last vertex in $\hat{\Pi}$ of other faces (these faces are ready immediately before v_{lower} is added) or it has a legal support outside q. In both cases, the new vertex v_{dummy} can be added with a legal support to one of the edges of the ready faces or the support edge of v_{lower} as v_{dummy} is inserted into $\hat{\Pi}$ immediately before v_{lower}. Thereafter, the vertex v_{lower}, which has index $k+1$ in $\hat{\Pi}$, is adjacent to at least two vertices in $\hat{G}_k$, i.e., v_{lowest} and v_{shift}, and, therefore, does not need a support edge. $\square$

Note that modification 1 is similar to the modification by Brandenburg et al. [BDE+16]. In their algorithm, they keep one of the crossing edges in a quadrangle (they have a divided quadrangle). Then, they compute the canonical ordering top down (starting with the largest index). When they encounter the first vertex of such a quadrangle, they check if the edge from this vertex to the opposing one has been kept in, and if not, they insert it and remove the other one.

3.2.2. Reinserting the crossing edges

In this section, we show that a reinsertion of the crossing edges into an empty quadrangle $q = (a, b, c, d)$ is always possible so that it fulfills our requirements. The cycle (a, b, c, d) is drawn in counter-clockwise order and a has the lowest index in the biconnected canonical ordering $\hat{\Pi}$ among the vertices in q. Furthermore, let v_{lower} denote the vertex with the second lowest y-coordinate among the vertices in q. Note that v_{lower} can be b or d, but not c. This is because c is adjacent to a, b, and d and has a greater index in $\hat{\Pi}$ than either b or d because c has no legal support on a contour containing only a. Moreover, note that in Case 2 and Case 3, v_{lower} is also the vertex with the second lowest index among the vertices in q since c is adjacent to b and d, and c is not drawn secondly by the previous observation and not drawn last by definition. Therefore, its y-coordinate is in between the y-coordinates of b and d. We define v_{top} as the vertex with the largest index in $\hat{\Pi}$ among the vertices in q.

We introduce a notation for lines defined by points. The line with slope m through a point p is denoted by l_p^m and the line through different points p and q is denoted by $l_{p,q}$.

Case 1: v_{top} equals c

Lemma 3.3. Given a divided quadrangle $q = (a, b, c, d)$ in Case 1 of the reinsertion step, the two reinserted edges $\{a, c\}$ and $\{b, d\}$ cross exactly once in a right angle at $(x(a), y(v_{\text{lower}}))$.

Proof. We only describe the case when $v_{\text{lower}} = d$ because the other case $v_{\text{lower}} = b$ is symmetric. An illustration of q before and after the insertion in this case is given in Fig. 3.4a and Fig. 3.4b. The bend points $e_{\{a,c\}}$ and $e_{\{b,d\}}$ of the edges $\{a, c\}$ and $\{b, d\}$, respectively, are placed as follows.

$$e_{\{a,c\}} = (x(a), y(d) + 1) \tag{3.1}$$

$$e_{\{b,d\}} = (x(a) + 1, y(d)) \tag{3.2}$$

We make sure these two edges cross exactly once and in a right angle. Both edges together consist of four line segments. These four line segments are:

$$\overline{ae_{\{a,c\}}} = \{(x, y) \mid x = x(a) \text{ and } y \in [y(a), y(d) + 1]\} \tag{3.3}$$

$$\overline{e_{\{a,c\}}c} = \begin{cases} \{(x, y) \mid y = \frac{y(c)-(y(d)+1)}{x(c)-x(a)}(x - x(a)) + (y(d) + 1) & \\ \qquad \text{and } \; y \in [y(d) + 1, y(c)]\} & \text{, if } x(c) \neq x(a) \\ \{(x, y) \mid x = x(a) \text{ and } y \in [y(d) + 1, y(c)]\} & \text{, otherwise} \end{cases} \tag{3.4}$$

$$\overline{de_{\{b,d\}}} = \{(x, y) \mid y = y(d) \text{ and } x \in [x(d), x(a) + 1]\} \tag{3.5}$$

$$\begin{aligned} \overline{e_{\{b,d\}}b} = \{(x, y) \mid y = \frac{y(b) - y(d)}{x(b) - (x(a) + 1)}(x - (x(a) + 1)) + y(d) \\ \text{and } x \in [x(a) + 1, x(b)]\} \end{aligned} \tag{3.6}$$

The line segments $\overline{ae_{\{a,c\}}}$ and $\overline{e_{\{a,c\}}c}$ and the line segments $\overline{de_{\{b,d\}}}$ and $\overline{e_{\{b,d\}}b}$ intersect in the bend points $e_{\{a,c\}}$ and $e_{\{b,d\}}$, respectively. This is exactly where they should intersect since both pairs form an edge that is bent at this bend point. If both line segments of the same edge lie on the same line, the complete edge will be drawn as straight line segment in the resulting drawing.

The line segments $\overline{ae_{\{a,c\}}}$ and $\overline{de_{\{b,d\}}}$ intersect in $(x(a), y(d))$ in a right angle. Clearly, the line segments $\overline{ae_{\{a,c\}}}$ and $\overline{e_{\{b,d\}}b}$ and the line segments $\overline{de_{\{b,d\}}}$ and $\overline{e_{\{a,c\}}c}$ do not intersect each other since both cover non-overlapping x- and y-ranges, respectively.

The line segments $\overline{e_{\{a,c\}}c}$ and $\overline{e_{\{b,d\}}b}$ do not intersect since we can separate both line segments via the line $l_{e_{\{a,c\}},b}$. The bend point $e_{\{b,d\}}$ is below this line because $x(e_{\{b,d\}})$ is between $x(e_{\{a,c\}})$ and $x(b)$, and $y(e_{\{b,d\}})$ is less than $y(e_{\{a,c\}})$ and at most $y(b)$ ($y(e_{\{b,d\}})$ is at most $y(b)$ since $v_{\text{lower}} = d$). The vertex c is above this line. Since $y(c)$ is greater

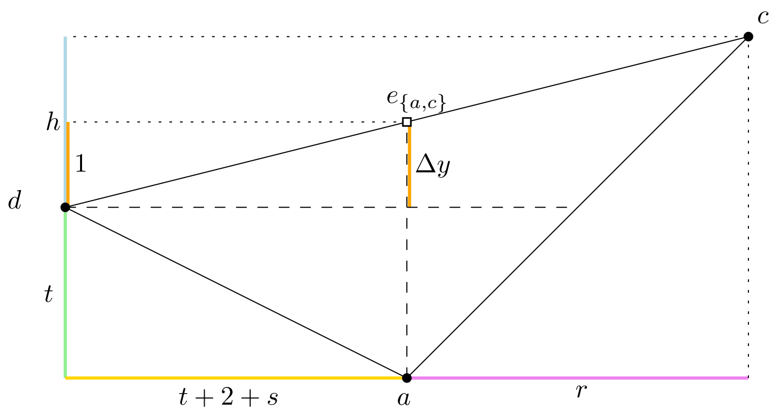


Fig. 3.5.: Situation in Case 1 of the edge reinsertion step. We assume for a contradiction proof that the inserted bend point $e_{\{a,c\}}$ lies on or above the edge $\{c, d\}$ (here: on it). We show that this leads to a contradiction and the bend point $e_{\{a,c\}}$ is below the line $l_{c,d}$.

than $y(e_{\{a,c\}})$ and $y(b)$, it lies above $l_{e_{\{a,c\}},b}$ if $x(c)$ is less than or equal to $x(b)$. If $x(c)$ is greater than $x(b)$, $x(c)$ will cover $x(b)$. Therefore, the slope of $l_{b,c}$ is greater than 1, whereas the slope of $l_{e_{\{a,c\}},b}$ is less than 1. Hence, c is above $l_{e_{\{a,c\}},b}$. Putting these things together, the line segment $\overline{e_{\{a,c\}}c}$ is above the line $l_{e_{\{a,c\}},b}$ except for an endpoint on this line and $\overline{e_{\{b,d\}}b}$ is below this line except for a different endpoint on this line. Thus, these line segments do not intersect. $\square$

Lemma 3.4. Given a divided quadrangle $q = (a, b, c, d)$ in Case 1 of the reinsertion step, the two reinserted edges $\{a, c\}$ and $\{b, d\}$ are inside the quadrangle q, i.e., they do not cross edges of the quadrangle q or other edges of the graph.

Proof. We only describe the case when $v_{\text{lower}} = d$ because the other case $v_{\text{lower}} = b$ is symmetric. The quadrangle $q = (a, b, c, d)$ is drawn as a convex polygon because every internal angle is less than 180 degrees. We show for each of the four internal angles at the vertices of q that they are less than 180 degrees.

- The internal angle at a is less than 180 degrees because a has the lowest index of the quadrangle in $\hat{\Pi}$ and is adjacent to the other three vertices. Namely, $y(b)$, $y(d)$, and $y(c)$ are all greater than $y(a)$ and as such $\angle dab$ is less than 180 degrees.
- The internal angle at c is less than 180 degrees. We can argue analogously to the internal angle at a because c has the largest index among the vertices of q in $\hat{\Pi}$ and the largest y-coordinate.
- The internal angles at b and d are less than 180 degrees because they are angles of the non-degenerate triangles (b, c, a) and (d, a, c), respectively. These two triangles appear as faces in the drawing returned by the Shift Algorithm.

We have shown that this quadrangle is drawn as a convex polygon. Thus, the line segments between any vertex of this polygon and any point inside the polygon lie completely inside the polygon. In our case, this means that if the two bend points lie strictly inside the quadrangle, the four line segments (two 1-bend edges) between vertices of the polygon and the bend points will be completely inside the quadrangle as well. Hence, the two inserted edges do not cross edges of the empty quadrangle or other edges of the drawn graph. In particular, by Lemma 3.3 they would form a right-angle crossing as needed. We will show next that $e_{\{a,c\}}$ and $e_{\{b,d\}}$ are inside q using the property that a point is inside a convex polygon if it is inside every half-plane spanned by its bounding edges. At a half-plane spanned by a bounding edge, inside means the side of the bounding line where the rest of the convex polygon is. In our case, this means above $l_{a,b}$ and $l_{a,d}$, and below $l_{b,c}$ and $l_{c,d}$.

First, consider $e_{\{a,c\}}$. It is placed at $(x(a), y(d) + 1)$. Beside this, we know that $x(d) < x(a) < x(b)$ because, while the graph is drawn, d, a, and b are on the contour when c is added (c is adjacent to all of them in this order and vertices that lie on the contour have strictly increasing x-coordinates). Assume for contradiction that $e_{\{a,c\}}$ lies outside or on the border of the convex quadrangle q. Then, it lies outside or on the border of at least one of the half planes that are spanned by the lines through the edges of the quadrangle. The line $l_{a,b}$ is:

$$l_{a,b} : y = \frac{y(b) - y(a)}{x(b) - x(a)}(x - x(a)) + y(a) \tag{3.7}$$

The point $e_{\{a,c\}}$ has the x-coordinate $x(a)$. At this x-value, $l_{a,b}$ is $y(a)$. We know that $y(a) < y(d)$ and $y(d) + 1$ is the y-coordinate of $e_{\{a,c\}}$. Thus, $e_{\{a,c\}}$ lies above $l_{a,b}$. We can use the same argument analogously for the half plane spanned by the line $l_{a,d}$. It is this line:

$$l_{a,d} : y = \frac{y(d) - y(a)}{x(d) - x(a)}(x - x(a)) + y(a) \tag{3.8}$$

Again, $e_{\{a,c\}}$ lies above the line.

Thus, $e_{\{a,c\}}$ lies outside the half planes spanned by $l_{b,c}$ and $l_{c,d}$. In other words, $e_{\{a,c\}}$ lies above or on $l_{b,c}$ or $l_{c,d}$. If $x(e_{\{a,c\}})$ is less than $x(c)$, $e_{\{a,c\}}$ lies above or on $l_{c,d}$ and otherwise it lies above or on $l_{b,d}$. In detail, we consider only the case that $x(e_{\{a,c\}})$ is less than $x(c)$ because the other one is symmetrically almost the same. This situation is depicted in Fig. 3.5. We know that the difference between $y(e_{\{a,c\}})$ and $y(d)$ is 1 by definition. We let the difference between $y(c)$ and $y(d)$ be h. Because c was drawn later than d and both are adjacent to each other, c has a y-coordinate that is greater than the one of d on the coarser grid. Since the grid has been refined by factor of 2 in each dimension, this means that:

$$h \geq 2 \tag{3.9}$$

This implies that c is to the right side of $e_{\{a,c\}}$ (it has a greater x-coordinate) because the y-coordinate of the line segment from d to c has only been increased by at most 1 unit at $x = x(e_{\{a,c\}})$ compared to its y-value at $x = x(d)$. Thus, $\overline{dc}$ has a slope with an absolute value being less than 1 and, therefore, d cannot be in the underset of c as it would have a slope with an absolute value greater than 1 then. We let the difference between $x(a)$ and $x(c)$ be r and it holds that:

$$r > 0 \tag{3.10}$$

Before c was added to the drawing, a and d were on the outer face and there was a slope of -1 between them. Let t denote their x- and y-difference at this time of the drawing phase. Then, c was added and d was shifted to the left (relative to a) by 1 unit. On the refined grid, this is 2 units. This increases the x-difference between d and a by 2 units and we add another summand s which is the additional x-difference that we got in total when d was shifted to the left further times in the Shift Algorithm. We can bound t by 1 unit of the coarser grid, it is 2 units on the refined grid.

$$t \geq 2 \tag{3.11}$$

And we cannot shift d towards a, so we know:

$$s \geq 0 \tag{3.12}$$

We know, when c was added, the slope of the line through d and c was 1 because both were neighbors on the outer face. This leads to the following equation.

$$1 = \frac{h}{t+2+r} \tag{3.13}$$

Solving this for r provides:

$$r = h - t - 2 \tag{3.14}$$

We let Δy denote the increase of the y-coordinate of the line segment $\overline{dc}$ at $x = x(e_{\{a,c\}})$ compared to its y-value at $x = x(d)$. We know:

$$0 < \Delta y \leq 1 \tag{3.15}$$

In Fig. 3.5 it is $\Delta y = 1$. Furthermore, we know that $\overline{dc}$ has the same slope between d and the point $(x(e_{\{a,c\}}), y(d) + \Delta y)$ and between the point $(x(e_{\{a,c\}}), y(d) + \Delta y)$ and c. It is:

$$\frac{\Delta y}{t+2+s} = \frac{h - \Delta y}{r} \tag{3.16}$$

We multiply with both denominators and get:

$$\Delta y r = ht + 2h + hs - \Delta y t - 2\Delta y - \Delta y s \tag{3.17}$$

Now we use Equation 3.14 to replace r:

$$\Delta y h - \Delta y t - 2\Delta y = ht + 2h + hs - \Delta y t - 2\Delta y - \Delta y s \tag{3.18}$$

We can simplify this to:

$$\Delta y h = ht + 2h + hs - \Delta y s \tag{3.19}$$

And reformulate it:

$$0 = ht + (2 - \Delta y)h + (h - \Delta y)s \tag{3.20}$$

That $ht + (2 - \Delta y)h + (h - \Delta y)s$ becomes 0, contradicts the Equations 3.9, 3.11, 3.12, and 3.15 because they force the first two summands to be greater than 0 and the last one to be greater or equal to 0. Note that $(2 - \Delta y)$ and $(h - \Delta y)$ are greater than 0 since Δy has an upper bound of 1 and h a lower bound of 2. Thus, our assumption was wrong in the case that $x(e_{\{a,c\}})$ is less than $x(c)$ because $e_{\{a,c\}}$ is inside all half-planes spanned by edges of q and, therefore, is inside q. The same holds also in the case that $x(e_{\{a,c\}})$ is not less than $x(c)$. Then, $e_{\{a,c\}}$ could only be outside the half-plane spanned by $l_{b,c}$, but this case is symmetric for the worst-case where $y(b) = y(d)$. For $y(b) > y(d)$, $e_{\{a,c\}}$ is clearly inside since $y(b)$ is greater than $y(e_{\{a,c\}})$ then. Hence, $e_{\{a,c\}}$ is inside q.

Now, consider the bend point $e_{\{b,d\}}$. It is placed at $(x(a)+1, y(d))$. We can use almost the same arguments as for $e_{\{a,c\}}$ if we just switch the x- and y-coordinates. The difference is that $x(a)$ is not necessarily less than $x(c)$ unlike the case for $e_{\{a,c\}}$, where $y(d)$ was less than $y(b)$ since we had $v_{\text{lower}} = d$. So, we are fine with arguing analogously as for the bend point $e_{\{a,c\}}$ if $x(a) \leq x(c)$. If $x(a) > x(c)$, we can argue analogously as for $e_{\{a,c\}}$ for the half planes spanned by $l_{c,d}$, $l_{a,d}$, and $l_{a,b}$. We know that $x(c) < x(a)+1 = x(e_{\{b,d\}}) < x(b)$ and $y(c) > y(b) \geq y(e_{\{b,d\}})$. Thus, $l_{c,b}$ has for $x \in (x(c), x(b))$ y-values that are greater than $y(b)$ and greater than $y(e_{\{b,d\}})$. Therefore, $e_{\{b,d\}}$ is also inside the remaining half plane spanned $l_{b,c}$.

So, both bend points are strictly inside the convex polygon and the line segments between them and a, b, c, and d are inside this polygon as well. Thus, the two edges that are represented by these four line segments do not cross other edges. $\square$

Case 2: v_{top} equals b or d and v_{lower} is in the underset of c

Lemma 3.5. Given a divided quadrangle $q = (a, b, c, d)$ in Case 2 of the reinsertion step, the two reinserted edges $\{a, c\}$ and $\{b, d\}$ cross in a right angle at $p_{\text{cross}} = (x_{\text{cross}}, y_{\text{cross}})$. The crossing point p_{cross} is a point of integer coordinates.

Proof. We only describe the case when $v_{\text{lower}} = b$ because the other case $v_{\text{lower}} = d$ is symmetric. An illustration of q before and after the insertion in this case is given in Fig. 3.4c and Fig. 3.4d.

First of all, we show that the two line segments $\overline{ce_{\{a,c\}}}$ and $\overline{be_{\{b,d\}}}$ cross each other in p_{cross}. The coordinates of p_{cross} are defined as follows:

$$x_{\text{cross}} = 0.5(x(c) - y(c) + x(b) + y(b)) \tag{3.21}$$
$$y_{\text{cross}} = 0.5(-x(c) + y(c) + x(b) + y(b)) \tag{3.22}$$

The two crossing line segments are:

$$\overline{ce_{\{a,c\}}} = \{(x, y) \mid y = \frac{(y_{\text{cross}} - 1) - y(c)}{(x_{\text{cross}} - 1) - x(c)}(x - x(c)) + y(c) \text{ and } x \in [x_{\text{cross}} - 1, x(c)]\} \tag{3.23}$$

$$\overline{be_{\{b,d\}}} = \{(x, y) \mid y = \frac{(y_{\text{cross}} + 1) - y(b)}{(x_{\text{cross}} - 1) - x(b)}(x - x(b)) + y(b) \text{ and } x \in [x_{\text{cross}} - 1, x(b)]\} \tag{3.24}$$

We show that the slopes of these two line segments are 1 and -1. Hence, they cross in a right angle if they cross each other.

The slope of $\overline{ce_{\{a,c\}}}$ is:

$$\begin{aligned}
\frac{(y_{\text{cross}}-1)-y(c)}{(x_{\text{cross}}-1)-x(c)} &= \frac{(0.5(-x(c)+y(c)+x(b)+y(b))-1)-y(c)}{(0.5(x(c)-y(c)+x(b)+y(b))-1)-x(c)} \\
&= \frac{-0.5x(c)-0.5y(c)+0.5x(b)+0.5y(b)-1}{-0.5x(c)-0.5y(c)+0.5x(b)+0.5y(b)-1} \\
&= 1
\end{aligned} \tag{3.25}$$

The slope of $\overline{be_{\{b,d\}}}$ is:

$$\begin{aligned}
\frac{(y_{\text{cross}}+1)-y(b)}{(x_{\text{cross}}-1)-x(b)} &= \frac{(0.5(-x(c)+y(c)+x(b)+y(b))+1)-y(b)}{(0.5(x(c)-y(c)+x(b)+y(b))-1)-x(b)} \\
&= \frac{-0.5x(c)+0.5y(c)+0.5x(b)-0.5y(b)+1}{0.5x(c)-0.5y(c)-0.5x(b)+0.5y(b)-1} \\
&= -1
\end{aligned} \tag{3.26}$$

We insert $x = x_{\text{cross}}$ into the line equation of Equation 3.23 and we use the result from Equation 3.25:

$$\begin{aligned}
y &= \frac{(y_{\text{cross}}-1)-y(c)}{(x_{\text{cross}}-1)-x(c)}(x_{\text{cross}}-x(c))+y(c) \\
&= (x_{\text{cross}}-x(c))+y(c) \\
&= 0.5x(c)-0.5y(c)+0.5x(b)+0.5y(b)-x(c)+y(c) \\
&= -0.5x(c)+0.5y(c)+0.5x(b)+0.5y(b) \\
&= y_{\text{cross}}
\end{aligned} \tag{3.27}$$

And into the line equation of Equation 3.24. Here, we use the result from Equation 3.26:

$$\begin{aligned}
y &= \frac{(y_{\text{cross}}+1)-y(b)}{(x_{\text{cross}}-1)-x(b)}(x_{\text{cross}}-x(b))+y(b) \\
&= -(x_{\text{cross}}-x(b))+y(b) \\
&= -0.5x(c)+0.5y(c)-0.5x(b)-0.5y(b)+x(b)+y(b) \\
&= -0.5x(c)+0.5y(c)+0.5x(b)+0.5y(b) \\
&= y_{\text{cross}}
\end{aligned} \tag{3.28}$$

So, the lines $l_{ce_{\{a,c\}}}$ and $l_{be_{\{b,d\}}}$ intersect in the point p_{cross}. It remains to show that this point is on the interior of the segments $\overline{ce_{\{a,c\}}}$ and $\overline{be_{\{b,d\}}}$. This is true if and only if $x_{\text{cross}}-1 < x_{\text{cross}} < x(c)$ and $x_{\text{cross}}-1 < x_{\text{cross}} < x(b)$. It is trivial that $x_{\text{cross}}-1 < x_{\text{cross}}$, but it remains to show the other two bounds.

To this end, we use the property that b is in the underset of c by definition. Thus, the absolute value of the slope of the line $l_{b,c}$ is greater than 1. In other words, we can bound the reciprocal of the slope of $l_{b,c}$ by -1 and 1, i.e.:

$$-1 < \frac{x(c)-x(b)}{y(c)-y(b)} < 1 \tag{3.29}$$

Since $y(c)$ is greater than $y(b)$, we can derive the following inequalities from Inequality 3.29.

$$x(b) + y(b) < x(c) + y(c) \tag{3.30}$$
$$x(c) < y(c) - y(b) + x(b) \tag{3.31}$$

Using Inequality 3.30 and Inequality 3.31, we can bound x_{cross}:

$$\begin{aligned} x_{\text{cross}} &= 0.5x(c) - 0.5y(c) + 0.5x(b) + 0.5y(b) \\ &< 0.5x(c) - 0.5y(c) + 0.5x(c) + 0.5y(c) \\ &= x(c) & (3.32) \\ x_{\text{cross}} &= 0.5x(c) - 0.5y(c) + 0.5x(b) + 0.5y(b) \\ &< 0.5y(c) - 0.5y(b) + 0.5x(b) - 0.5y(c) + 0.5x(b) + 0.5y(b) \\ &= x(b) & (3.33) \end{aligned}$$

So, we have shown that the line segments $\overline{ce_{\{a,c\}}}$ and $\overline{be_{\{b,d\}}}$ intersect in p_{cross} in a right angle. Thus, the edges $\{a, c\}$ and $\{b, d\}$ cross at this point in a right angle. The point p_{cross} has only integer coordinates because $x(c)$, $y(c)$, $x(b)$, and $y(b)$ are integer coordinates of the coarser grid and, therefore, multiples of 2 on the finer grid. After multiplying them with a factor of 0.5, they are still integer coordinates. □

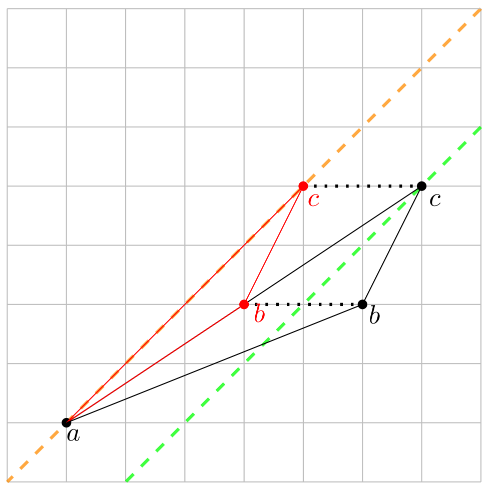
Lemma 3.6. Given a divided quadrangle $q = (a, b, c, d)$ in Case 2 of the reinsertion step, the two reinserted edges $\{a, c\}$ and $\{b, d\}$ are inside the quadrangle q, i.e., they do not cross edges of the quadrangle q or other edges of the graph. They also do not cross more than once.

Proof. We only describe the case when $v_{\text{lower}} = b$ because the other case $v_{\text{lower}} = d$ is symmetric. The bend points $e_{\{a,c\}}$ and $e_{\{b,d\}}$ of the edges $\{a, c\}$ and $\{b, d\}$, respectively, are placed as follows.

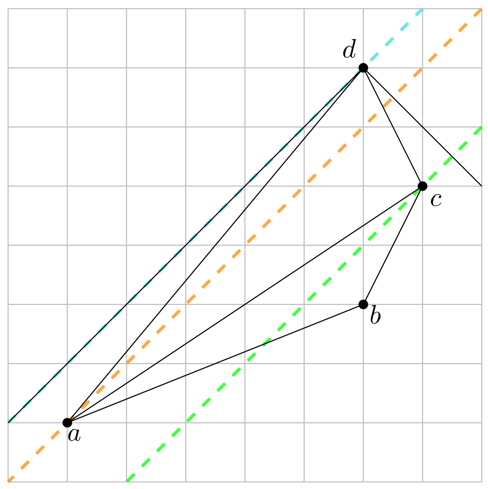
$$e_{\{a,c\}} = (x_{\text{cross}} - 1, y_{\text{cross}} - 1) \tag{3.34}$$
$$e_{\{b,d\}} = (x_{\text{cross}} - 1, y_{\text{cross}} + 1) \tag{3.35}$$

We will use the following relations of the x- and y-coordinates of the vertices a, b, c, and d. The vertex a has the lowest index in the biconnected canonical ordering $\hat{\Pi}$ and is adjacent to the three other vertices. Therefore, it has the lowest y-coordinate of these vertices. The opposite vertex c cannot have a lower index than b and d because it is adjacent to both and would not have a left or a right support before b or d is there. So, $v_{\text{lower}} = b$ occurs before c in $\hat{\Pi}$. Because c is adjacent to b and is added later, $y(c)$ is greater than $y(b)$. Analogously, because $v_{\text{top}} = d$ is adjacent to c and is added later, $y(d)$ is greater than $y(c)$. The crossing point has a y-coordinate between $y(b)$ and $y(c)$. The bend points are 1 unit above and below this crossing point, i.e., half a grid square length on the coarser grid. Thus, they are in the y-range $[y(b), y(c)]$. Putting this together, we get:

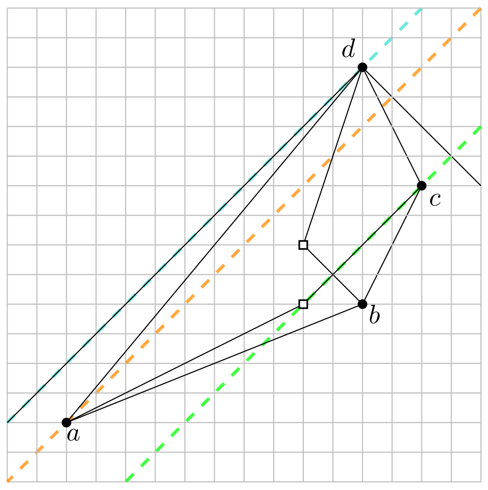
$$y(a) < y(b) \leq y(e_{\{a,c\}}) < y(e_{\{b,d\}}) \leq y(c) < y(d) \tag{3.36}$$



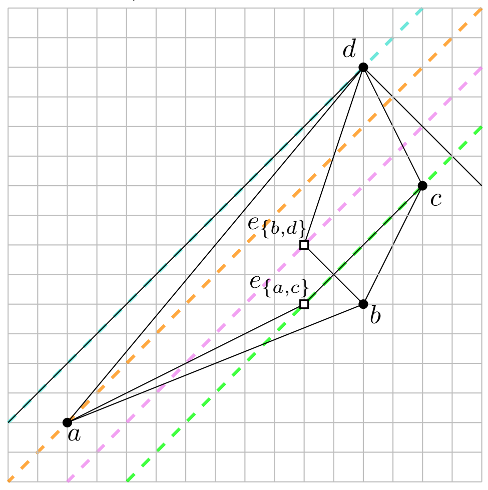
(a) Vertices a, b, and c before d is added. Before d is added, c and b (since b is in the underset of c) are shifted away from a by 2 units via a shift vertex. The situation before the shift is colored in red. There, the slope of $l_{a,c}$ was 1. It is dashed in dark orange. The line l_c^{+1} after the shift is dashed in green.



(b) The divided quadrangle directly after vertex d was added. Consider the left-uppermost and right-uppermost incident edge to d. They have slopes of 1 and -1 because they are part of the contour now. Here, the vertices a and c are covered by d, but in general one or both could also still be on the contour (then they would be on the outer line segments with slope $+1$ or -1).



(c) Situation after the grid has been refined by a factor of 2 in both dimensions and the crossing edges have been reinserted.



(d) Illustration of the minimum distances between the lines with slope 1 through the vertices. The line l_a^{+1} has a minimum Manhattan distance of 2 to the line $l_{e_{\{b,d\}}}^{+1}$ (dashed in violet). The line l_d^{+1} (dashed in turquoise) has a minimum Manhattan distance of 4 to $l_{e_{\{b,d\}}}^{+1}$.

Fig. 3.6.: Example of the edge reinsertion in Case 2.

For the x-coordinates, we know that before c was added, a and b were on the contour on a line with slope 1. Therefore, we have $x(a) < x(b)$. Furthermore, we know by the position of the crossing and the bend points that both bend points have the same x-coordinate. This x-coordinate $x(e_{\{a,c\}}) = x(e_{\{b,d\}})$ is less than $x(b)$ and $x(c)$.

Consider the edge $\{b, d\}$. It is bent at $e_{\{b,d\}}$. We show that its two line segments do not intersect any of the edges of the quadrangle q. First, we show that $\overline{be_{\{b,d\}}}$ intersects none of the four bounding edges of q:

- The line segments $\overline{be_{\{b,d\}}}$ and $\overline{ab}$ do not intersect because both cover non-overlapping y-ranges except for the common endpoint.
- The line segments $\overline{be_{\{b,d\}}}$ and $\overline{bc}$ do not intersect because both have the same endpoint, but different slopes. The line segment $\overline{be_{\{b,d\}}}$ has a slope of -1. If $\overline{bc}$ had the same slope, b would not be in the underset of c.
- The line segments $\overline{be_{\{b,d\}}}$ and $\overline{cd}$ do not intersect because both cover non-overlapping y-ranges except for possibly at $y = y(e_{\{b,d\}})$, but because $x(e_{\{b,d\}})$ is less than $x(c)$, they do not intersect.
- The line segments $\overline{be_{\{b,d\}}}$ and $\overline{ad}$ do not intersect because the line $l^{+1}_{e_{\{b,d\}}}$ separates both line segments from each other. Recall that we shifted a and c away from each other by 2 units by the insertion of a shift vertex before d was added. Hence, l^{+1}_a and l^{+1}_c have a minimum distance of $x = 2$ or one diagonal of a grid square on the coarser grid. This is a Manhattan distance of 2 on the coarser grid and a Manhattan distance of 4 on the refined grid. For an illustration of this situation, see Fig. 3.6a. The distance may be greater if a and c were shifted apart from each other further times. The line l^{+1}_d has a Manhattan distance of at least 3 on the coarser grid and at least 6 on the refined grid to l^{+1}_c because another shift was performed when d was added. See Fig. 3.6b. Later on, the crossing edges are reinserted as in Fig. 3.6c. Now, Consider $l^{+1}_{e_{\{a,c\}}}$ and $l^{+1}_{e_{\{b,d\}}}$. An illustration of them is given in Fig. 3.6d. As shown in the proof of Lemma 3.5, the line segment $\overline{ce_{\{a,c\}}}$ has a slope of 1. Thus, c, $e_{\{a,c\}}$, and p_{cross} lie on the same line with slope 1 (dashed in green in Fig. 3.6). By definition, $e_{\{b,d\}}$ lies one diagonal of a grid square above the crossing point. This means that $l^{+1}_{e_{\{b,d\}}}$ has a Manhattan distance of exactly 2 to l^{+1}_c and is above this line. Therefore, l^{+1}_a and l^{+1}_d are above $l^{+1}_{e_{\{b,d\}}}$ with a Manhattan distance of at least 2 and 4, respectively. So, the line segment $\overline{be_{\{b,d\}}}$, which is on and below $l^{+1}_{e_{\{b,d\}}}$, cannot intersect the line segment $\overline{ad}$, which is above $l^{+1}_{e_{\{b,d\}}}$.

Second, we show that $\overline{e_{\{b,d\}}d}$ intersects none of the four bounding edges of q:

- The line segments $\overline{e_{\{b,d\}}d}$ and $\overline{ab}$ do not intersect because both cover non-overlapping y-ranges.
- The line segments $\overline{e_{\{b,d\}}d}$ and $\overline{bc}$ do not intersect because the line $l^{+1}_{e_{\{b,d\}}}$ separates d and $e_{\{b,d\}}$ from b and c.

- The line segments $\overline{e_{\{b,d\}}d}$ and $\overline{cd}$ do not intersect because both have the same endpoint, but different slopes. Consider l_d^{+1}, $l_{e_{\{b,d\}}}^{+1}$, and l_c^{+1}. Since the $l_{e_{\{b,d\}}}^{+1}$ is in between the other two, $e_{\{b,d\}}$ would be a point of the interior of $\overline{cd}$ if both line segments had the same slope. If $e_{\{b,d\}}$ was a point of the interior of $\overline{cd}$, $y(e_{\{b,d\}})$ would be greater than $y(c)$. However, $y(e_{\{b,d\}}) \leq y(c)$. Thus, $e_{\{b,d\}}$ is no point of the interior of $\overline{cd}$.

- The line segments $\overline{e_{\{b,d\}}d}$ and $\overline{ad}$ do not intersect because both have the same endpoint, but different slopes or they are separated by a line. Consider l_d^{+1}, l_a^{+1}, and $l_{e_{\{b,d\}}}^{+1}$. If l_a^{+1} is above or identical to l_d^{+1}, then l_d^{+1} separates both line segments from each other except for the common endpoint d. Otherwise l_a^{+1} is in between l_d^{+1} and $l_{e_{\{b,d\}}}^{+1}$. Then a would be a point of the interior of $\overline{e_{\{b,d\}}d}$ if both line segments had the same slope. If a was a point of the interior of $\overline{e_{\{b,d\}}d}$, $y(a)$ would be greater than $y(e_{\{b,d\}})$. However, $y(a) < y(e_{\{b,d\}})$. Thus, a is no point of the interior of $\overline{e_{\{b,d\}}d}$.

Now, consider the edge $\{a, c\}$. It is bent at $e_{\{a,c\}}$. It remains to show that none of its two line segments intersects an edge of q. First, we show that $\overline{ae_{\{a,c\}}}$ intersects none of the four bounding edges of q:

- The line segments $\overline{ae_{\{a,c\}}}$ and $\overline{ab}$ do not intersect because both have the same endpoint, but different slopes because $x(e_{\{a,c\}}) < x(b)$ and $y(e_{\{a,c\}}) \geq y(b)$.

- The line segments $\overline{ae_{\{a,c\}}}$ and $\overline{bc}$ do not intersect because both cover non-overlapping x-ranges.

- The line segments $\overline{ae_{\{a,c\}}}$ and $\overline{cd}$ do not intersect because both cover non-overlapping y-ranges.

- The line segments $\overline{ae_{\{a,c\}}}$ and $\overline{ad}$ do not intersect because both have the same endpoint, but different slopes. Observe that $e_{\{a,c\}}$ lies below $l_{e_{\{b,d\}}}^{+1}$ (the dashed violet line in Fig. 3.6d), whereas $\overline{ad}$ lies above. So, if both line segments had the same slope, $\overline{ad}$ would be shorter than $\overline{ae_{\{a,c\}}}$ and, therefore, $y(d) < y(e_{\{a,c\}})$. This contradicts $y(e_{\{a,c\}}) < y(d)$.

Second, we show that $\overline{e_{\{a,c\}}c}$ intersects none of the four bounding edges of q:

- The line segments $\overline{e_{\{a,c\}}c}$ and $\overline{ab}$ do not intersect because both cover non-overlapping y-ranges except for possibly at $y = y(e_{\{a,c\}})$, but because $x(e_{\{a,c\}}) < x(b)$, they do not intersect.

- The line segments $\overline{e_{\{a,c\}}c}$ and $\overline{bc}$ do not intersect because both have the same endpoint, but different slopes. The line segment $\overline{e_{\{a,c\}}c}$ has a slope of 1, but $\overline{bc}$ has a slope with an absolute value greater than 1 because otherwise b would not be in the underset of c.

- The line segments $\overline{e_{\{a,c\}}c}$ and $\overline{cd}$ do not intersect because both cover non-overlapping y-ranges except for the common endpoint.
- The line segments $\overline{e_{\{a,c\}}c}$ and $\overline{ad}$ do not intersect because the line $l^{+1}_{e_{\{b,d\}}}$ separates both line segments from each other.

The line segments of the crossing edges do not intersect each other more often than the one time described before. The line segments $\overline{ae_{\{a,c\}}}$ and $\overline{e_{\{b,d\}}d}$ cover non-overlapping y-ranges. The line segments $\overline{ae_{\{a,c\}}}$ and $\overline{be_{\{b,d\}}}$ cover non-overlapping x-ranges except for at $x = x(e_{\{a,c\}}) = x(e_{\{b,d\}})$, but there their y-coordinates differ. The line segments $\overline{e_{\{a,c\}}c}$ and $\overline{e_{\{b,d\}}d}$ are separated by the line $l^{+1}_{e_{\{b,d\}}}$ as described before.

In Lemma 3.5 and here, we have shown that the edges $\{a, c\}$ and $\{b, d\}$ cross once and in a right angle. They do not cross edges of the quadrangle. Thus, these crossing edges are either completely inside the quadrangle (a, b, c, d) or completely outside. If one of the two bend points is inside the quadrangle, the crossing edges are completely inside. We can easily see that $e_{\{a,c\}}$ is inside the triangle (a, b, c), which is inside (a, b, c, d) since d is not inside (a, b, c). Therefore, the crossing edges $\{a, c\}$ and $\{b, d\}$ are completely inside the quadrangle (a, b, c, d). □

Case 3: v_{top} equals b or d and v_{lower} is not in the underset of c

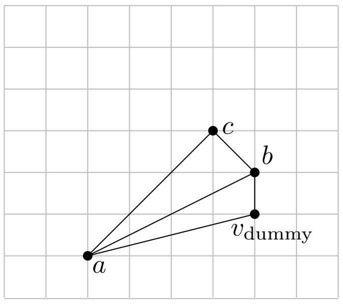
Lemma 3.7. Given a divided quadrangle $q = (a, b, c, d)$ in Case 3 of the reinsertion step, the following conditions hold. The two reinserted edges $\{a, c\}$ and $\{b, d\}$ cross exactly once inside the quadrangle q in a right angle at $(x(c), y(v_{\text{lower}}))$. They do not cross edges of q or other edges of the graph.

Proof. We only describe the case when $v_{\text{lower}} = b$ because the other case $v_{\text{lower}} = d$ is symmetric. In this case, we have the quadrangle $q = (a, b, c, d)$ and, as a neighboring face, a face of degree 3 which is formed by a, b, and a dummy vertex v_{dummy}. An illustration of q before and after the insertion in this case is given in Fig. 3.4e and Fig. 3.4f. The bend points $e_{\{a,c\}}$ and $e_{\{b,d\}}$ of the edges $\{a, c\}$ and $\{b, d\}$, respectively, are placed as follows.

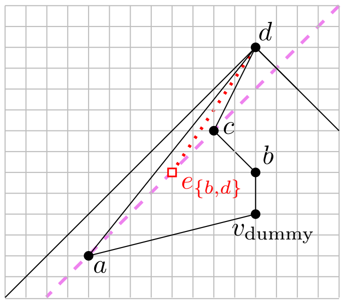
$$e_{\{a,c\}} = (x(c), y(b) - 1) \tag{3.37}$$

$$e_{\{b,d\}} = (x(c) - (y(c) - y(b)), y(b)) \tag{3.38}$$

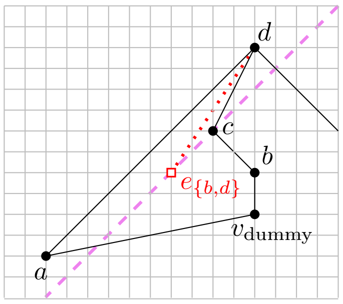
First, we establish some relations of the x- and the y-coordinates of the points appearing in Case 3. The vertex a has the lowest index in the biconnected canonical ordering $\hat{\Pi}$ among the vertices in q and also a lower index than v_{dummy}. It is adjacent to all vertices in q and to v_{dummy}. Therefore, it has the lowest y-coordinate of these vertices. Like in Case 2, the opposite vertex c cannot have a lower index than b and d. So, $v_{\text{lower}} = b$ is the second vertex of q in $\hat{\Pi}$ and occurs directly after v_{dummy}. Again as in Case 2, $y(c)$ is greater than $y(b)$, and $y(d)$ is greater than $y(c)$. By definition, $y(e_{\{b,d\}})$ equals $y(b)$. The other bend point, $e_{\{a,c\}}$ has a y-coordinate of $y(b) - 1$. Thus, $y(e_{\{a,c\}})$ is greater than



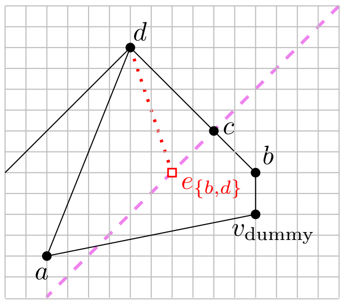
(a) Situation after c was added to the drawing and before d is added to the drawing. This base situation is the same for all of the four following cases.



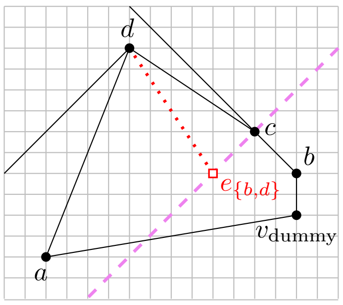
(b) Case that a and c are both covered by d. The bend point $e_{\{b,d\}}$ behaves as if it was a vertex on the contour before and is also covered. Thus, it can also see d.



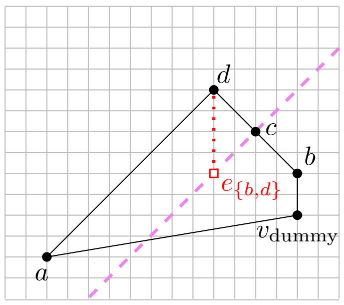
(c) Case that c is covered by d, but a is not covered by d. Again, the bend point $e_{\{b,d\}}$ behaves as if it was a vertex on the contour before and is also covered. Thus, it can also see d. Moving a to the left does not change this.



(d) Case that a is covered by d, but c is not covered by d. The bend point $e_{\{b,d\}}$ behaves as if it was covered and then moved one unit to the right, i.e., further away from the edge $\{a, d\}$. Thus, it can also see d.



(e) Case as in Fig. 3.7d, but with c being shifted another time to the right. With the same argument as before, $e_{\{b,d\}}$ can still see d as it was moved by another unit to the right side. The same applies for more shifts.



(f) Case that a and c are both not covered by d. The bend point $e_{\{b,d\}}$ behaves as if it was covered and then moved one unit to the right, i.e., further away from the edge $\{a, d\}$. Like in Fig. 3.7d and Fig. 3.7e, $e_{\{b,d\}}$ can see d and further shifts of c (and with it $e_{\{b,d\}}$) to the right do not change this.

Fig. 3.7.: Examples for the position of the bend point $e_{\{b,d\}}$ of the edge $\{b, d\}$ relative to the pentagonal face $(a, v_{\text{dummy}}, b, c, d)$ in Case 3 of the reinsertion step with $v_{\text{lower}} = b$. The examples differ in whether a and c are covered by d.

$y(v_{\text{dummy}})$ because v_{dummy} is on a grid point of the coarser grid, i.e., v_{dummy} is at least 2 units below b on the finer grid. So, we have this relation of the y-coordinates:

$$y(a) < y(v_{\text{dummy}}) < y(e_{\{a,c\}}) < y(b) = y(e_{\{b,d\}}) < y(c) < y(d) \tag{3.39}$$

Another observation is that $x(a)$, $x(b)$, and $x(c)$ are pairwise different values because $x(c)$ adjacent to the other both but covers neither of them. Vertex b is not covered by c by definition and a cannot be covered by c since it is adjacent to d which is added after c. We obtain the following relation of x-coordinates:

$$x(a) < x(c) = x(e_{\{a,c\}}) < x(b) \tag{3.40}$$

Now, we analyze the course of the two inserted edges. They consist of two line segments each. These line segments are:

$$\overline{ae_{\{a,c\}}} = \{(x,y) \mid y = \frac{y(b)-1-y(a)}{x(c)-x(a)}(x - x(a)) + y(a) \text{ and } x \in [x(a), x(c)]\} \tag{3.41}$$

$$\overline{e_{\{a,c\}}c} = \{(x,y) \mid x = x(c) \text{ and } y \in [y(b)-1, y(c)]\} \tag{3.42}$$

$$\overline{be_{\{b,d\}}} = \{(x,y) \mid y = y(b) \text{ and } x \in [x(c) - (y(c)-y(b)), x(b)]\} \tag{3.43}$$

$$\overline{e_{\{b,d\}}d} = \begin{cases} \{(x,y) \mid y = \frac{y(d)-y(b)}{x(d)-x(c)+y(c)-y(b)}(x - x(d)) + y(d) \text{ and} \\ \qquad x \in [\min\{x(d), x(c)-(y(c)-y(b))\}, \\ \qquad\quad \max\{x(d), x(c)-(y(c)-y(b))\}]\ \}, \text{ if } x(d) \neq x(c)-(y(c)-y(b)) \\ \{(x,y) \mid x = x(d) \text{ and } y \in [y(b), y(d)]\} \qquad , \text{otherwise} \end{cases} \tag{3.44}$$

The line segments $\overline{e_{\{a,c\}}c}$ and $\overline{be_{\{b,d\}}}$ are vertical and horizontal line segments, respectively. So, they intersect in a right angle if they intersect. Obviously, they intersect in $(x(c), y(b))$ since $(y(c) - y(b))$ is greater than 0 and $x(c)$ is smaller than $x(b)$. Thus, the edges $\{a,c\}$ and $\{b,d\}$ intersect at least once in a right angle. They do not cross another time. The line segments $\overline{ae_{\{a,c\}}}$ and $\overline{e_{\{b,d\}}d}$, and the line segments $\overline{ae_{\{a,c\}}}$ and $\overline{be_{\{b,d\}}}$ cover non-overlapping y-ranges since $y(a) < y(e_{\{a,c\}}) < y(e_{\{b,d\}}) = y(b) < y(d)$. The line segments $\overline{e_{\{a,c\}}c}$ and $\overline{e_{\{b,d\}}d}$ do not intersect either since $\overline{e_{\{a,c\}}c}$ lies on or below the line l_c^{+1} and $\overline{e_{\{b,d\}}d}$ lies on or above this line. This line is:

$$l_c^{+1} : y = x - x(c) + y(c) \tag{3.45}$$

The upper endpoint of $\overline{e_{\{a,c\}}c}$, i.e., c lies obviously on this line. In the following equation, we insert the x-coordinate of the lower endpoint of $\overline{e_{\{b,d\}}d}$, i.e., $e_{\{b,d\}}$ into the line equation of l_c^{+1}.

$$l_c^{+1}(x(e_{\{b,d\}})) = x(c) - (y(c) - y(b)) - x(c) + y(c) = y(b) \tag{3.46}$$

So, this line also runs through $e_{\{b,d\}}$ and, since $e_{\{b,d\}}$ differs from c, these line segments also do not intersect. Therefore, $\{a,c\}$ and $\{b,d\}$ cross exactly once in a right angle.

Now, we show that the edges $\{a,c\}$ and $\{b,d\}$ do not cross any other edge of the graph. We do that by showing that the four line segments of these two edges do not intersect any line segment of an edge of the bounding pentagon $(a, v_{\text{dummy}}, b, c, d)$ and that they are inside this pentagonal face. We start with the two line segments of $\{a, c\}$. First, we show that $\overline{ae_{\{a,c\}}}$ intersects none of the five bounding edges of the pentagonal face $(a, v_{\text{dummy}}, b, c, d)$:

- The line segments $\overline{ae_{\{a,c\}}}$ and $\overline{v_{\text{dummy}}b}$ do not intersect because the line l_b^{+1} separates the endpoints of these two line segments. The bend point $e_{\{a,c\}}$ lies above l_b^{+1} because $e_{\{a,c\}}$ is to the left of b and the slope of the line $l_{e_{\{a,c\}},b}$ is less than 1 since there is a x-distance of at least 2 between them, but only a y-distance of 1. The x-distance of at least 2 follows from the fact that c and b were grid points on the coarser grid and $x(e_{\{a,c\}}) = x(c)$. The vertex a lies above this line because the slope of the line $l_{a,b}$ is less than 1 and a is to the left of b. The slope is less than 1 because, before c was added, the line through a and b had a slope of 1, and then a and b were shifted apart from each other in the x-dimension. The vertex b lies on l_b^{+1}. The dummy vertex v_{dummy} lies below this line because if it was to the left of b, it would be covered by b (otherwise it would conflict with a) and $l_{v_{\text{dummy}},b}$ has a slope greater than 1 then. If v_{dummy} is to the right of b, $l_{v_{\text{dummy}},b}$ has a negative slope and v_{dummy} lies below l_b^{+1} as well.
- The line segments $\overline{ae_{\{a,c\}}}$ and $\overline{av_{\text{dummy}}}$ do not intersect because both have the same endpoint, but different slopes. If they had the same slope, v_{dummy} would be on $\overline{ae_{\{a,c\}}}$ since $y(a) < y(v_{\text{dummy}}) < y(e_{\{a,c\}})$. Because $x(a) < x(e_{\{a,c\}})$, it follows that $x(a) < x(v_{\text{dummy}}) < x(e_{\{a,c\}}) < x(b)$. The line $l_{a,e_{\{a,c\}}}$ has a slope that is less than 1 because, before d was added, the line through a and c (c is vertically above $e_{\{a,c\}}$) already had a slope of 1 and these vertices were possibly moved even further apart from each other later. From the argumentation why $\overline{ae_{\{a,c\}}}$ and $\overline{v_{\text{dummy}}b}$ do not intersect, we know that the slope of $l_{e_{\{a,c\}},b}$ is also less than 1. This means that the slope of $l_{v_{\text{dummy}},b}$ is also less than 1. This contradicts the fact from before that $l_{v_{\text{dummy}},b}$ has a slope greater than 1.
- The line segments $\overline{ae_{\{a,c\}}}$ and $\overline{bc}$ do not intersect because both cover non-overlapping y-ranges.
- The line segments $\overline{ae_{\{a,c\}}}$ and $\overline{cd}$ do not intersect because both cover non-overlapping y-ranges.
- The line segments $\overline{ae_{\{a,c\}}}$ and $\overline{ad}$ do not intersect because both have the same endpoint, but different slopes. If they had the same slope, $e_{\{a,c\}}$ would be on the line $l_{a,d}$. Then, c and the edge $\{c, d\}$ would be above this line because c has the same x-coordinate as $e_{\{a,c\}}$, but a greater y-coordinate and we know that $x(a) < x(c)$. This contradicts our definition of the face bounded by the cycle (a, b, c, d), where the edge $\{c, d\}$ precedes the edge $\{d, a\}$ in counter-clockwise order and not the other way around.

Second, we show that the line segment $\overline{e_{\{a,c\}}c}$ intersects none of the five bounding edges of the pentagonal face $(a, v_{\text{dummy}}, b, c, d)$:

- The line segments $\overline{e_{\{a,c\}}c}$ and $\overline{av_{\text{dummy}}}$ do not intersect because both cover non-overlapping y-ranges.

- The line segments $\overline{e_{\{a,c\}}c}$ and $\overline{v_{\text{dummy}}b}$ do not intersect because the line l_b^{+1} separates the endpoints of these two line segments. Compare with the argumentation that the line segments $\overline{ae_{\{a,c\}}}$ and $\overline{v_{\text{dummy}}b}$ do not intersect. According to this, v_{dummy} and b are below or on l_b^{+1} and e_{ac} is above. Then, c is also above l_b^{+1} since it has the same x-coordinate as $e_{\{a,c\}}$.

- The line segments $\overline{e_{\{a,c\}}c}$ and $\overline{bc}$ do not intersect because both have the same endpoint, but different slopes. The line segment $\overline{e_{\{a,c\}}c}$ is a vertical segment, while the x-coordinates of b and c differ.

- The line segments $\overline{e_{\{a,c\}}c}$ and $\overline{cd}$ do not intersect because both cover non-overlapping y-ranges except for the common endpoint.

- The line segments $\overline{e_{\{a,c\}}c}$ and $\overline{ad}$ do not intersect because c and $e_{\{a,c\}}$ are both below the line $l_{a,d}$. The vertex c is below $l_{a,d}$ because it has a smaller y-coordinate than d and the definition of the vertices in the cycle (a, b, c, d) is in counter-clockwise order in the drawing. Since $e_{\{a,c\}}$ has the same x-coordinate as c but a smaller y-coordinate, it is also below $l_{a,d}$.

Now, we consider the two line segments of the edge $\{b, d\}$. First, we show that the line segment $\overline{be_{\{b,d\}}}$ intersects none of the five bounding edges of the pentagonal face $(a, v_{\text{dummy}}, b, c, d)$:

- The line segments $\overline{be_{\{b,d\}}}$ and $\overline{av_{\text{dummy}}}$ do not intersect because both cover non-overlapping y-ranges.

- The line segments $\overline{be_{\{b,d\}}}$ and $\overline{v_{\text{dummy}}b}$ do not intersect because both have the same endpoint, but different slopes. The line segment $\overline{be_{\{b,d\}}}$ is horizontal, while v_{dummy} and b differ in the y-coordinate.

- The line segments $\overline{be_{\{b,d\}}}$ and $\overline{bc}$ do not intersect because both have the same endpoint, but different slopes. The line segment $\overline{be_{\{b,d\}}}$ is horizontal, while b and c differ in the y-coordinate.

- The line segments $\overline{be_{\{b,d\}}}$ and $\overline{cd}$ do not intersect because both cover non-overlapping y-ranges.

- The line segments $\overline{be_{\{b,d\}}}$ and $\overline{ad}$ do not intersect because the line l_c^{+1} separates the two line segments. First of all, observe that $e_{\{b,d\}}$ lies on the line l_c^{+1} as the y-difference between them is $y(c) - y(b)$, and exactly by this value, $e_{\{b,d\}}$ is shifted to the left of c in x-dimension. Observe that d is above this line because d is adjacent to c (and added to the drawing later), but c is not its left uncovered child. It cannot to be the left uncovered child of d because a has a smaller x-coordinate than c and is adjacent to d as well. Since a and c were on the contour on the same line with slope 1 before d was added, the vertex a is also on l_c^{+1} if a and c are both covered by d (compare with Fig. 3.7b) or above this line otherwise (compare with the other cases in Fig. 3.7). The vertex b is below l_c^{+1} since it has the same y-coordinate as $e_{\{b,d\}}$, but a greater x-coordinate. So, l_c^{+1} separates $\overline{be_{\{b,d\}}}$ from $\overline{ad}$. Both have up to one point on this line but these points differ (the y-coordinate of $e_{\{b,d\}}$ is greater than the one of a).

Second, we show that the line segment $\overline{e_{\{b,d\}}d}$ intersects none of the five bounding edges of the pentagonal face $(a, v_{\text{dummy}}, b, c, d)$:

- The line segments $\overline{e_{\{b,d\}}d}$ and $\overline{av_{\text{dummy}}}$ do not intersect because both cover non-overlapping y-ranges.

- The line segments $\overline{e_{\{b,d\}}d}$ and $\overline{v_{\text{dummy}}b}$ do not intersect because both cover non-overlapping y-ranges except for at $y = y(b) = y(e_{\{b,d\}})$, but there their x-coordinates differ.

- The line segments $\overline{e_{\{b,d\}}d}$ and $\overline{bc}$ do not intersect because the line l_c^{+1} separates both line segments. We have shown before that $\overline{be_{\{b,d\}}}$ and $\overline{ad}$ do not intersect. There, we have argued that d is above l_c^{+1}, b is below l_c^{+1}, and $e_{\{b,d\}}$ is on l_c^{+1}. Since c is also on l_c^{+1} and the points c and $e_{\{b,d\}}$ are different points, l_c^{+1} separates $\overline{e_{\{b,d\}}d}$ from $\overline{bc}$.

- The line segments $\overline{e_{\{b,d\}}d}$ and $\overline{cd}$ do not intersect because both have the same endpoint, but different slopes. If they had the same slope, the three points c, d, and $e_{\{b,d\}}$ would be on the same line. The line $l_{c,e_{\{b,d\}}}$ has a slope of 1 as shown before. So, d would be on this line as well, but we have shown before that it is above this line. Thus, they are not on the same line and these two line segments have different slopes.

- The line segments $\overline{e_{\{b,d\}}d}$ and $\overline{ad}$ do not intersect because both have the same endpoint, but different slopes. Assume that they have the same slope. Then, $e_{\{b,d\}}$ is a point of the line segment $\overline{ad}$ since $y(a) < y(e_{\{b,d\}}) < y(d)$. From the argumentation that $\overline{be_{\{b,d\}}}$ and $\overline{ad}$ do not intersect, we know that $e_{\{b,d\}}$ is on l_c^{+1}, but d is above l_c^{+1}. So, $\overline{ad}$ can also not be on l_c^{+1}. To have $e_{\{b,d\}}$ on $\overline{ad}$, one vertex of a and d is above l_c^{+1} and the other one below. We have also shown before that both are on or above this line, which is a contradiction. So, $\overline{e_{\{b,d\}}d}$ and $\overline{ad}$ do not have the same slope.

We have shown now that neither the edge $\{a, c\}$ nor the edge $\{b, d\}$ crosses an edge of the pentagonal face $(a, v_{\text{dummy}}, b, c, d)$. It remains to show that $\{a, c\}$ and $\{b, d\}$ are both inside this face. Here, it is enough to show that a point of one of these two edges is inside this face since both are connected (they cross each other). Going along the boundary of this face in counter-clockwise direction, we have the interior of the face to our left hand side. There, we reach v_{dummy} immediately before b and b immediately before c. Since we go upwards when we are between these vertices, a line segment that starts at b and runs horizontally to the left is (at least partially) inside this face. Such a line segment is $\overline{be_{\{b,d\}}}$. □

3.2.3. Results

Lemma 3.8. The drawing returned by Algorithm 3 is a 1-bend drawing.

Proof. The drawing $\hat{\Gamma}$, which was drawn by the Shift Algorithm, is a straight-line drawing. All edges that are inserted in the reinsertion step have at most one bend. This makes Γ^* a 1-bend drawing. After this, some dummy vertices are transformed to bend points. We show next that this does not create edges with two or more bend points. There are two types of dummy vertices transformed to bend point. First, these are original edges that were subdivided during the creation of empty kites. They cannot be crossed by other edges because this would violate the NIC-planarity. Therefore, they do not have bends in Γ^* and this one new bend per edge does not make an edge have two or more bends. Second, these are the dummy vertices added during the computation of the biconnected canonical ordering $\hat{\Pi}$. They are added as subdividing vertices to edges of the divided quadrangles. These edges do not cross other edges and cannot be subdivided in the step of creating empty kites because then they would be outside an empty kite or divided quadrangle. So, the dummy vertices that become bend points are never adjacent to each other and the drawing Γ is a polyline drawing with at most one bend per edge. □

Lemma 3.9. The drawing Γ returned by Algorithm 3 is a RAC drawing of the NIC-plane input graph $(G, \mathcal{E}(G))$. The embedding $\mathcal{E}(G)$ is preserved.

Proof. In the preprocessing step, there are vertices and edges added to the embedded graph. The dummy vertices that become bend points at the end do not change the embedding if we interpret them also initially as bend points. Dummy vertices and edges that are removed at the end of the algorithm are added inside of the faces. This does not change the original embedding (we have the original embedding if we ignore these dummy vertices and edges). We change the embedding when the crossing edges are removed. If we remember them and their spot in the given NIC-planar embedding, we can reinsert them later and restore the given original embedded graph. This is done with the data structure Q.

After this, we compute a biconnected canonical ordering $\hat{\Pi}$ of the resulting planar embedded graph. The dummy vertices that may be added while we compute $\hat{\Pi}$, become bend points in the end or they are shift vertices that are removed in the step when we reinsert the crossing edges. Like the dummy vertices from the preprocessing, they do not change the embedding if we interpret them as bend points. Beside these dummy vertices, we insert dummy edges into some faces of the embedded graph. They do not change the embedding if we ignore them as they are removed later. The drawing returned by the Shift Algorithm respects the given embedding and is crossing-free.

In the reinsertion step, we insert the crossing edges at their original positions which we have saved in Q. By Lemma 3.3, Lemma 3.4, Lemma 3.5, Lemma 3.6, and Lemma 3.7, we have shown that in any case the reinserted edges cross no other edge but each other in a right angle within its reserved spot, its quadrangle from Q. □

3.3. Running time and grid size

In this section, we describe the running time of our algorithm and the size of the grid onto which the resulting drawing is placed. In Lemma 3.10, we show that the number of vertices that we add during our augmentation of the given NIC-planar is bounded in the number of vertices of the input graph by a constant factor. In Lemma 3.11, we show the linear running time, and in Lemma 3.12, we show that the graph is drawn on a quadratically sized grid.

Lemma 3.10. Let G be the input graph, G' be the graph after the preprocessing, and $\hat{G}$ be the graph after the computation of the biconnected canonical ordering. Let n, n', and $\hat{n}$ be the number of vertices in G, G', and $\hat{G}$, respectively. It holds that $n' \leq 3.4n - 4.8$ and $\hat{n} \leq 4n - 6$.

Proof. In the first step of the preprocessing, we create empty kites around every crossing. By creating the empty kites for every crossing, there are edges added to the graph (but we do not count edges here) and there are edges subdivided. When we subdivide an edge, we add a new vertex. There are at most four edges per crossing that are subdivided. The number k of crossings in the given NIC-planar embedding is bounded by $0.6n - 1.2$ according to Lemma 2.20. Using this, we can bound the number $n_{\text{subdivide}}$ of vertices that are added in this step to:

$$n_{\text{subdivide}} \leq 4k \leq 2.4n - 4.8 \tag{3.47}$$

In the second step of the preprocessing, we make the graph biconnected. To accomplish this, we only insert edges and the number of vertices is not increased. So, the number n' of vertices of the graph G' is:

$$n' = n + n_{\text{subdivide}} \leq 3.4n - 4.8 \tag{3.48}$$

While computing the biconnected canonical ordering $\hat{\Pi}$, at most one dummy vertex is added per crossing—either as shift vertex in Case 2 or as dummy vertex in Case 3. So the number $n_{\hat{\Pi}}$ of vertices added there is:

$$n_{\hat{\Pi}} \leq k \leq 0.6n - 1.2 \tag{3.49}$$

And in total:

$$\hat{n} = n' + n_{\hat{\Pi}} \leq (3.4n - 4.8) + (0.6n - 1.2) = 4n - 6 \tag{3.50}$$

□

So, we have shown that there are only linearly many new vertices when constructing G' from G and $\hat{G}$ from G'.

Lemma 3.11. The algorithm presented in this chapter, i.e., Algorithm 3 runs in time $O(n)$ where n is the number of vertices in the input graph.

Proof. To determine the required running time of our algorithm, we analyze the time complexity of each of its steps. Let k denote the number of crossings in the given NIC-planar embedding. The steps of the preprocessing are:

- Transform each crossing first into an empty kite and then into an empty quadrangle. Inserting up to four edges and deleting two edges can be done in linear time. Querying if parallel edges, which will be subdivided, exist can be done in linear time in total. If we query an edge, we check the incident edges of the vertex with the lower degree. Only sufficiently few of these vertices can have a high degree since NIC-planar graphs are sparse according to Lemma 2.18. Therefore, we perform this step in $O(k + n)$ time, which is in $O(n)$ according to Lemma 2.20.

- We make the graph biconnected using the linear time algorithm given in Algorithm 1.

So, the preprocessing can be done in $O(n)$ time.

In the original paper, the computation of the biconnected canonical ordering is performed in time linear in the size of the input graph. Here, this would be $O(n')$ time. It remains to determine the extra time needed for the additional work. For each vertex v, we can initialize a list of the empty quadrangles from Q which contain v. Then, we can access the list of empty/divided quadrangles incident to a vertex in constant time. Each quadrangle is accessed exactly four times (once from each of its vertices) and the work performed in such an access can be completed in constant time. Namely, this work consist of adding a dummy edge or subdividing an edge. The data structures can also be updated in constant time since only the values of up to two faces and up to one vertex are adjusted. So, this step can be performed in $O(n' + k)$ time, i.e., $O(n)$ time according to Lemma 3.10 and Lemma 2.20.

Drawing a graph using the Shift Algorithm can be done in time linear in the size of the input graph. Here, this is $O(\hat{n})$ time, which is in $O(n)$ time according to Lemma 3.10.

In the next step, the crossing edges are reinserted. It is easy to see that each divided quadrangle can be processed in constant time. Thus, the edge reinsertion step runs in $O(k)$ time, which is also in $O(n)$ time.

In the last step, the dummy edges are removed and dummy vertices are transformed to bend points. If we save a collection of all dummy objects added throughout the execution of the algorithm, we can remove or transform each of them in constant time. Following the previous argumentation, we have added only $O(n)$ many objects and, therefore, we can do this step in $O(n)$ time.

Since these steps are processed sequentially and not done multiple times, we need $O(n)$ time in total. □

Lemma 3.12. Every vertex, bend point and crossing point of the drawing returned by Algorithm 3 is a grid point of a grid of size at most $(16n - 32) \times (8n - 16)$

Proof. The Shift Algorithm places every vertex of the graph $\hat{G} = (\hat{V}, \hat{E})$ onto a grid point of a grid of size $(2\hat{n} - 4) \times (\hat{n} - 2)$. By the upper bound on $\hat{n}$ from Lemma 3.10, we get the following grid size:

$$\begin{aligned} \text{coarser grid size} &\leq (2(4n-6)-4) \times ((4n-6)-2) \\ &= (8n-16) \times (4n-8) \end{aligned} \tag{3.51}$$

This grid is later refined by a factor of 2 in both dimensions. This bounds the size of the grid as follow:

$$\begin{aligned} \text{total grid size} &\leq (2(8n-16)) \times (2(4n-8)) \\ &= (16n-32) \times (8n-16) \end{aligned} \tag{3.52}$$

We insert bend points onto inner faces only. So, the total size of the drawing and its underlying grid is not increased when we add them. □

3.4. Proof of the main theorem

We plug the results of the previous analysis together in order to prove the main result of this chapter.

Theorem 3.1. *Every NIC-planar graph $G = (V, E)$ admits a NIC-planar 1-bend RAC drawing with vertices, bend points, and crossing points on a grid of size at most $(16n - 32) \times (8n-16)$, where $n := |V|$. Given a NIC-planar embedding $\mathcal{E}(G)$ of G, such a drawing can be computed in $O(n)$ time. The returned drawing preserves the given embedding $\mathcal{E}(G)$.*

Proof. By definition, there is a NIC-planar drawing for every NIC-planar graph $G = (V, E)$. Using the embedding of this drawing, we can assume, without loss of generality, that we have a given NIC-plane graph $(G, \mathcal{E}(G))$. Use this NIC-plane graph as input for Algorithm 3. According to Lemma 3.9 the returned drawing Γ is a RAC drawing of $(G, \mathcal{E}(G))$, thus it preserves the given embedding. Moreover, Γ is a 1-bend drawing according to Lemma 3.8 with vertices, bend points and crossing points on a grid of size $(16n-32) \times (8n-16)$ according to Lemma 3.12. Algorithm 3 runs in time $O(n)$ according to Lemma 3.11. □

The same holds for every IC-planar and IC-plane graph because the IC-planar graphs are a subset of the NIC-planar graphs and an IC-planar embedding is preserved as such by our algorithm.

4. Constructing 1-Bend RAC Drawings of 1-Plane Graphs

This algorithm is a modification of the algorithm presented by Bekos et al. [BDL$^+$17]. In the same paper, they ask if there are 1-planar embeddings that cannot be realized as 1-bend RAC drawings. We negatively answer their question. Indeed, every 1-planar embedding can be realized as 1-bend RAC drawing. We constructively show this by slightly modifying their algorithm to preserve the given embedding of the 1-plane graph. The algorithm still returns a 1-planar 1-bend RAC drawing, but it does not change the embedding.

In this chapter, we first describe the original algorithm by Bekos et al. [BDL$^+$17] in Section 4.1, then our modifications of it in Section 4.2, and last we formulate the improved result obtained by our modification in Section 4.3.

4.1. Original algorithm

We give a short overview about the original algorithm by Bekos et al. [BDL$^+$17] for computing a 1-planar 1-bend RAC drawing of a given 1-plane graph $(G, \mathcal{E}(G))$ in the variable embedding setting, i.e., the returned drawing is possibly not embedded according to $\mathcal{E}(G)$, but according to another 1-planar embedding. The whole algorithm runs in linear time. The resulting drawings may have exponential area.

Augmentation In the augmentation step, they transform $(G, \mathcal{E}(G))$ into a triangulated 1-plane multigraph $(G^+, \mathcal{E}^+(G^+))$. In a multigraph, there can be parallel edges, i.e., more than one edge between the same pair of vertices. The embeddings $\mathcal{E}$ and $\mathcal{E}^+$ may be different. They insert dummy edges around each pair of crossing edges to induce empty kites. Thereby parallel edges can arise. They remove the original edge from each set of parallel edges, and for each face of degree two, i.e., a face bounded by two parallel edges, they remove one of the edges. There can still be parallel dummy edges. The last operation in the augmentation step is star-triangulating every face that has degree greater than 3. If we star-triangulate an embedded graph, we insert a dummy vertex into each face and connect it via a dummy edge with each vertex on the boundary of the face. This may also create parallel edges. The resulting embedded 1-plane multigraph is $(G^+, \mathcal{E}^+(G^+))$.

Hierarchical contraction For each set of parallel edges in $(G^+, \mathcal{E}^+(G^+))$ incident to a separation pair of vertices, they isolate the subgraphs between each neighboring pair

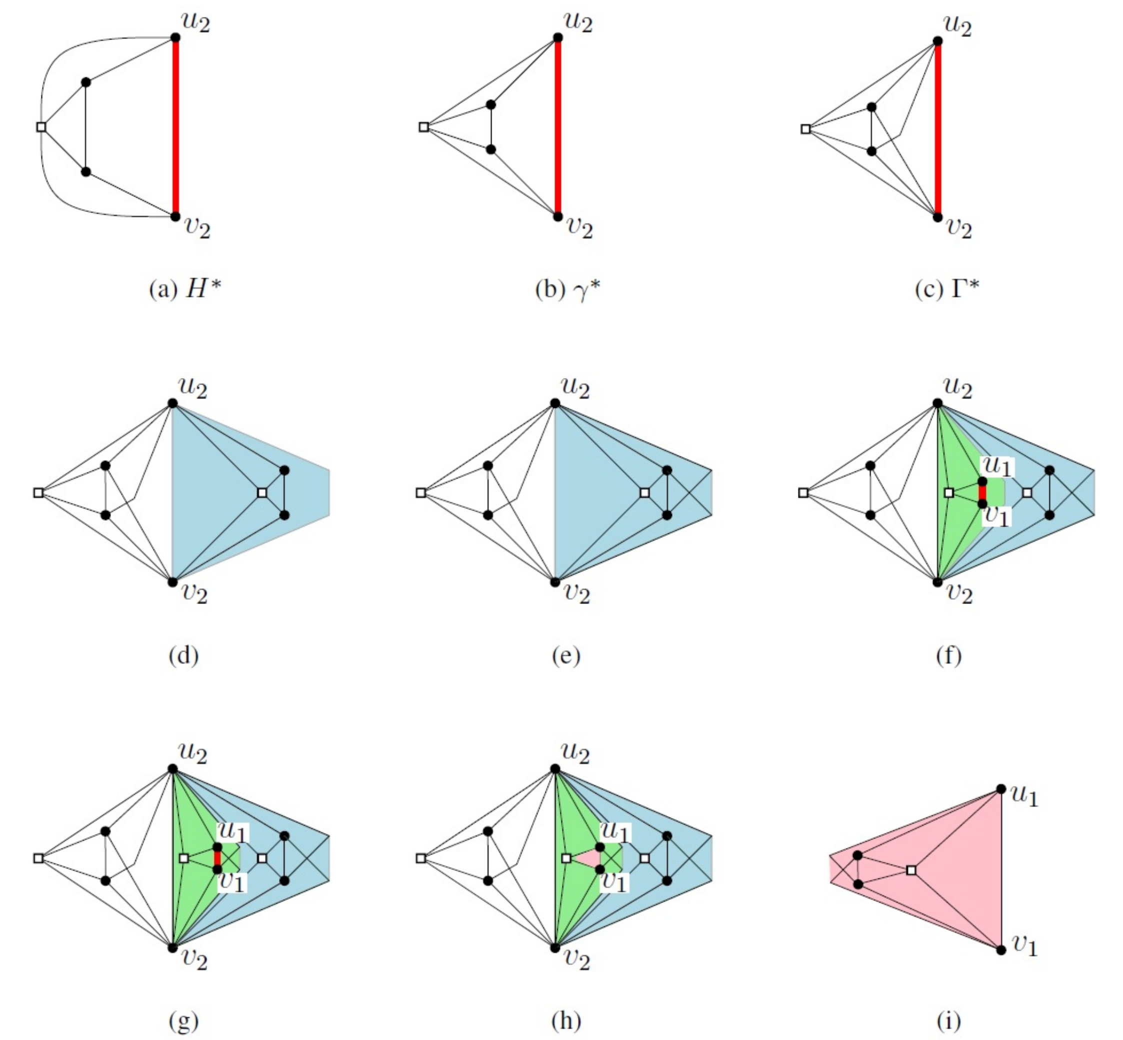


Fig. 4.1.: Figure taken from Bekos et al. [BDL+17]. Example for a recursive drawing of their algorithm. Thick edges are drawn as thick segments in red. Dummy vertices from the star-triangulation are drawn as white square boxes. The top-level graph H^* from (a) is drawn in (b) and the crossing edges are reinserted in (c). In (d), (e), (f), (g), and (h), the subgraphs saved at the thick edges are extracted, drawn, and the crossing edges are reinserted. The drawing of the subgraph from (i) is inserted into the pink face in (h).

of parallel edges and then contract all parallel edges and everything inside to a *thick* edge. For every thick edge, they save the information of the subgraphs. This operation is applied hierarchically to all subgraphs and their subgraphs and so on until there are no parallel edges in the graph or any subgraph. Now, each such graph is simple, 1-planar, 3-connected, and triangulated. The top-level graph after this step is $(G^*, \mathcal{E}^*(G^*))$.

Drawing In this step, the thick edges are recursively extracted, i.e., their subgraphs are drawn and inserted into a face next to the line segments of the thick edges. An example of the drawing step is given as illustration in Fig. 4.1. Before they draw the top-level graph $(G^*, \mathcal{E}^*(G^*))$ or some lower-level graph, they remove the crossing edges and draw the obtained plane 3-connected graph with an algorithm that delivers strictly convex straight-line drawings where the outer face is a prescribed convex polygon. The linear-time algorithm by Chiba et al. [CYN84] fulfills these requirements. They pass as prescribed polygon a trapezoid if the outer face has degree 4 (then there has been a crossing on the outer face which was removed at the beginning of the drawing step) and a triangle otherwise. Next, they manually reinsert the crossing edges. For the inner convex faces, they draw one edge straight-line and the other edge with a bend so that it crosses the first edge in a right angle. For the outer faces, they bend both edges once as in Fig. 4.1i. Since they can prescribe the outer face of each subgraph, they can always pass a shape that fits into the free space next to a thick edge.

After this step, they have a drawing of G^+. They remove every dummy vertex and every dummy edge and a 1-planar 1-bend RAC drawing of G remains.

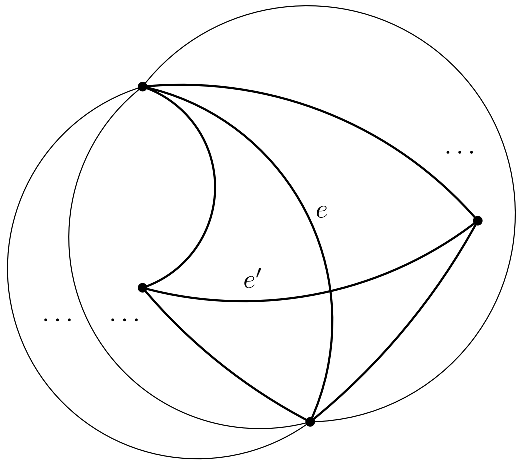
4.2. Our modifications

In this section we give an overview about our modifications to the algorithm described briefly in Section 4.1. We also show the correctness of our changes here.

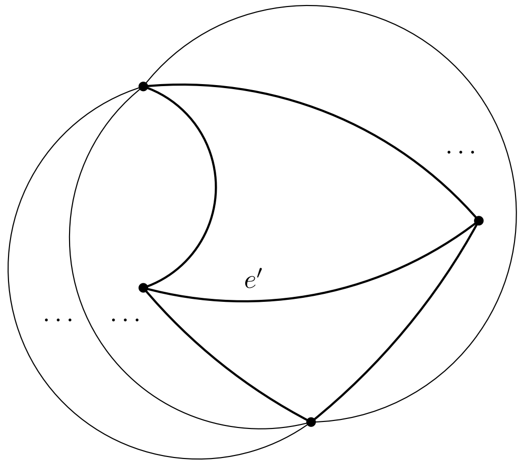
Augmentation As described, they remove from each set of parallel edges the original edge. Our modification is to keep the original edges that are not part of a crossing. But like them, we remove the parallel original edges that cross another edge. When we remove such a crossing edge, let it be e, an empty kite becomes a divided quadrangle (see Fig. 4.2). The edge e' crossed e before the removal. The edge e' cannot have parallel dummy edges because these would have crossed with either e or with parallel dummy edges of e, but, as written in the paper, no inserted dummy edge can cause a new crossing or be part of a crossing and a crossing with e would violate the 1-planarity. We remember where we removed these edges because we will reinsert them later.

Hierarchical contraction We do not modify this step, but we save the order of the subgraphs contracted at each separation pair to a thick edge and mark the original edge.

Drawing This step is almost the same as in the original algorithm, but we make sure that we draw the inner graphs that were contracted at a separation pair (u, v) in the



(a) After the insertion of dummy edges to obtain empty kites. The empty kite containing the crossing edges e and e' is highlighted.



(b) After the removal of the original edge e. By this removal, the empty kite becomes a divided quadrangle containing e' as inner edge.

Fig. 4.2.: Removal of an original edge that has parallel edges and is part of a crossing.

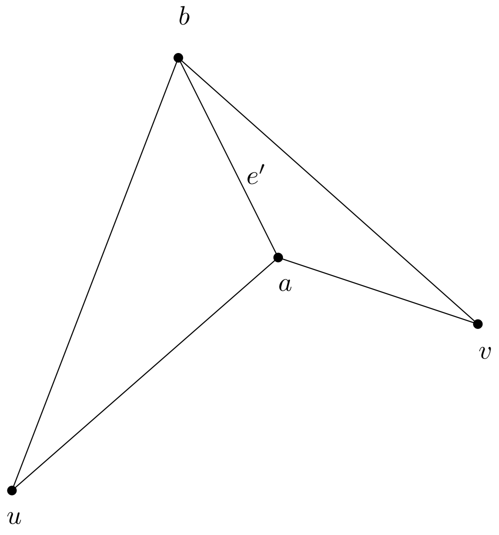
original order. We distinguish two cases.

Case 1 In this case, we have kept the original edge $\{u, v\}$ in the set of parallel edges between u and v. In the original paper, they insert inner graphs stacked on one side (or they do not care on which side) of the straight line between u and v. We insert the original edge $\{u, v\}$ as the straight line segment $\overline{uv}$ and then draw the subgraphs that have been to the left side of this edge in the original embedding, on the left side of this straight line segment in their original internal order. Analogously, we proceed with the subgraphs on the right side of $\{u, v\}$.

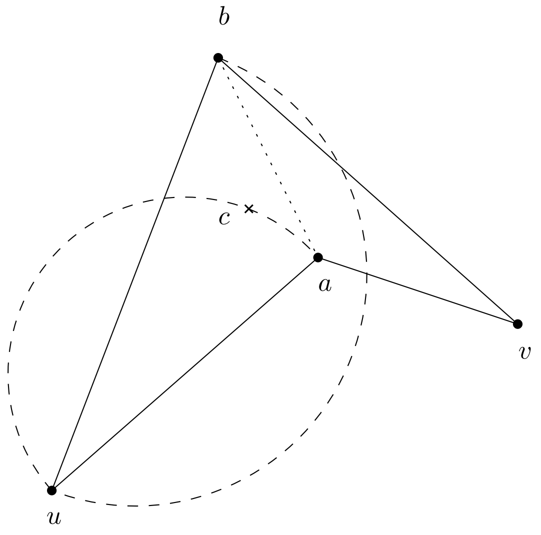
Case 2 In this case, we have removed the original edge $\{u, v\}$ from the set of parallel edges between u and v. Again, we draw all subgraphs of the separation pair $\{u, v\}$ in their original internal order, but now, we do not have the original edge $\{u, v\}$ as the straight-line segment $\overline{uv}$. We are free to draw the inner graphs on one side or on both sides of the straight line as long as we do it in the correct order.

We do not insert an edge along the straight line between u and v. Instead, we reinsert the original edge at its original place in the embedding, i.e., into the divided quadrangle, in a post processing. In this divided quadrangle, one face has the two endpoints of e' (let these be a and b) and u as corner points and the other face has a, b, v as corner points (see Fig. 4.3a). To obtain a 1-bend RAC drawing, we reinsert e in the way described next.

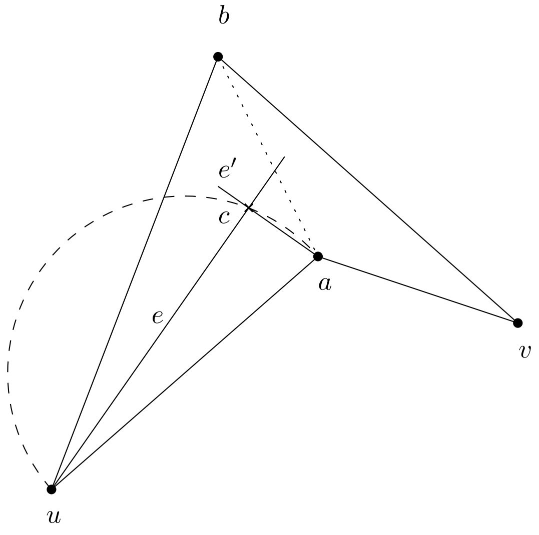
We remove the straight line drawing of e' so that we obtain the empty quadrangle (a, v, b, u). Choose some point c on the Thales' circle around $\{a, u\}$ or $\{b, u\}$ that lies strictly inside the triangle (a, b, u) (see Fig. 4.3b). Such a point exists. Assume for



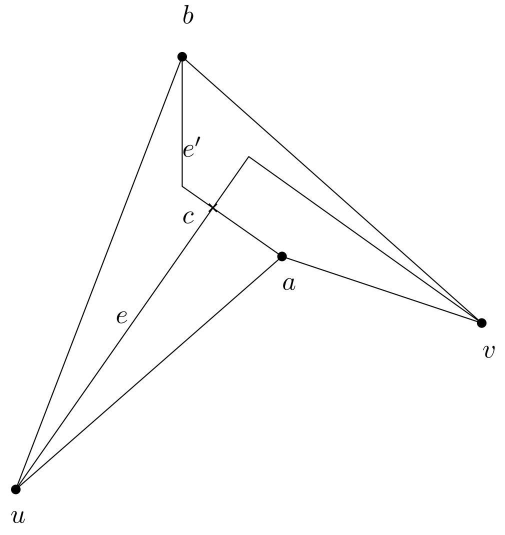
(a) The divided quadrangle as returned by the Shift Algorithm.



(b) Thales' circles onto which we can place the later crossing point c.



(c) Crossing point c and the first parts of the crossing edges $\{a, b\}$ and $\{u, v\}$ up to their bend points.



(d) Reinserted 1-bend edges $\{a, b\}$ and $\{u, v\}$ that cross in a right angle inside the quadrangle (a, v, b, u).

Fig. 4.3.: Reinserting the original edge $e = \{u, v\}$, which crosses the edge $e' = \{a, b\}$ in a right angle.

contradiction that it does not exist. Then, b lies inside the Thales' circle around $\{a, u\}$. Therefore, the angle $\angle abu$ is greater than 90 degrees. Analogously to b, a must lie inside the Thales' circle around $\{b, u\}$. Therefore, the angle $\angle uab$ is also greater than 90 degrees and the triangle (a, b, u) has a sum of internal angles that exceeds 180 degrees. This is a contradiction and, thus, there is a point on one of these two Thales' circles inside the triangle (a, b, u).

The chosen point c will be our crossing point. We can draw the first part of e and e' as straight line segments from the points to which the chosen Thales' circle belongs to to c. Now, we can lengthen the straight line of e' a little so that we are still inside (a, b, u) (see Fig. 4.3c). With e' we can reach a or b because this vertex is a corner point of the triangle we are currently in and we have already passed e. We can see it and reach it with another straight-line segment, using the current endpoint as a bend point. We can also lengthen the straight line segment of e through c until it reaches the other triangle, i.e., (a, v, b), but does not pass or touch the border of the whole face of the empty quadrangle. At such a point, we place a bend point for e and connect it with v. Since the bend point is inside the triangle (a, v, b) we can see v from the bend point (see Fig. 4.3d). This way, we can manually reinsert the crossing edges so that we obtain right-angle crossings.

4.3. Result

Observe that our modifications do not require more than linear time. Therefore, the total running time remains linear. We immediately showed the feasibility of our modifications. Hence, we obtain the following result which improves the theorem by Bekos et al. [BDL+17] in preserving a given embedding.

Theorem 4.1. *Let $(G, \mathcal{E}(G))$ be a 1-plane graph with n vertices. Then G admits a 1-planar RAC drawing Γ with at most one bend per edge that respects the given embedding $\mathcal{E}(G)$. We can compute Γ in $O(n)$ time.*

Like the base algorithm by Bekos et al. [BDL+17], the adapted version also only bends edges that participate in a crossing. Edges that are not crossed are drawn as straight-line segments.

5. Constructing 2-Bend RAC Drawings of 1-Plane Graphs in Polynomial Area

We show that every 1-plane graph $(G, \mathcal{E}(G))$ admits a 1-planar 2-bend RAC drawing with vertices, crossing points and bend points on a grid of size $(8n'^3 - 48n'^2 + 96n' - 64) \times (4n'^3 - 24n'^2 + 48n' - 32)$. Here, n' is the number of vertices of G plus 5 times the number of edge crossings in $\mathcal{E}(G)$. Note that, by Lemma 2.19, the number of crossings is at most $n-2$, where n is the number of vertices in G. The algorithm does not change the embedding. If we ignore the position of the bend points, we will have a 1-planar 2-bend RAC drawing with vertices and crossing points on a grid of size $(2n' - 4) \times (n' - 2)$, i.e., quadratic size.

The main result of this chapter is:

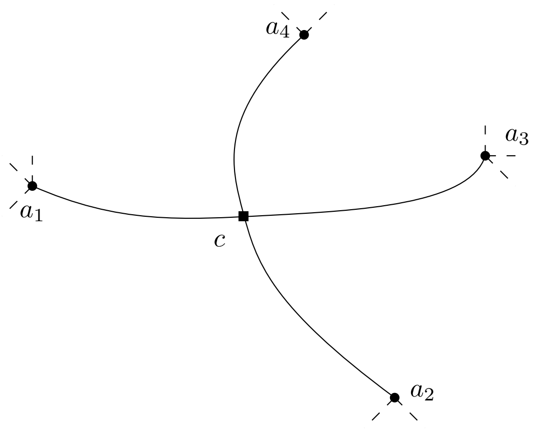
Theorem 5.1. *Every 1-planar graph $G = (V, E)$ with n vertices admits a 1-planar 2-bend RAC drawing with vertices, bend points, and crossing points on a grid of size $O(n^3) \times O(n^3)$, where the bends occur only on edges participating in a crossing. Given a 1-planar embedding $\mathcal{E}(G)$ of G, such a drawing can be computed in $O(n)$ time. The returned drawing preserves the given embedding $\mathcal{E}(G)$ and is placed on a grid of size $(8n'^3 - 48n'^2 + 96n' - 64) \times (4n'^3 - 24n'^2 + 48n' - 32)$, where n' is the number of vertices of G plus 5 times the number of crossings in $\mathcal{E}(G)$. It holds that $n' \leq 6n - 10$.*

Our proof to this theorem is constructive. We give an Algorithm to compute such a drawing in linear time in Section 5.1. In Section 5.2, we show its correctness to finally prove this main theorem in Section 5.3.

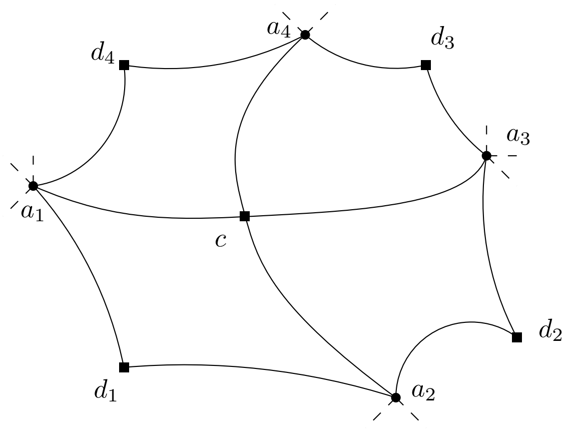
5.1. Algorithm

Our algorithm gets a 1-plane graph $(G, \mathcal{E}(G))$ as input. First, we planarize G by replacing the crossing points by crossing vertices. Second, we enclose each crossing vertex by a divided kite. We use divided kites instead of empty kites so that we do not change the embedding and do not add parallel edges. This step is visualized in Fig. 5.1. Third, we make the graph biconnected using Algorithm 1. After these three steps, we have an embedded planar graph $(G', \mathcal{E}'(G'))$. We draw $(G', \mathcal{E}'(G'))$ by using the algorithm of Harel and Sardas [HS98]. This algorithm returns a crossing-free straight line drawing Γ' with the vertices lying on a grid of size $(2n' - 4) \times (n' - 2)$ and without changing the embedding. Here, n' is the number of vertices of G'.

For each vertex c in G' that corresponds to a crossing point in $(G, \mathcal{E}(G))$, there are four incident edges. They correspond to two edges of G. In the circular order around c, the first and the third incident edge in $(G', \mathcal{E}'(G'))$ correspond to the same edge in $(G, \mathcal{E}(G))$.



(a) Planarized crossing which became a crossing vertex c.



(b) Enclosing the crossing vertex c by a divided kite.

Fig. 5.1.: A crossing point in G is replaced by a crossing vertex c in G' and we insert four 2-paths of two dummy edges and a dummy vertex to induce a divided kite at each crossing. The vertices a_1, a_2, a_3, and a_4 are original vertices, while d_1, d_2, d_3, and d_4 are dummy vertices.

Similarly, the second and fourth edges correspond to the other edge of the crossing. So, Γ' can be seen as a 1-bend drawing of G where every bend point is also a crossing point. To obtain a RAC drawing from this, each of the four line segments of the edges around c is replaced by two line segments. An edge $\{a, c\}$ that is drawn as straight line segment $\overline{ac}$ will then be drawn as 1-bend segment—one segment from c to a point b on a axis-parallel line through c and the other segment from b to a. Observe that we can obtain a RAC crossing at c if we bijectively place the four bend points on the four axis-parallel half-lines (rays) originating in c. For an example of a crossing in Γ' see Fig. 5.3a, and for its replacement in the final drawing see Fig. 5.3f. There, the crossing point c is drawn in the center. The four axis-parallel half-lines originating in c are drawn as dotted lines. Around c, there is the divided kite with vertices as filled black circles and dummy vertices as filled black squares (boxes). To create a 2-bend RAC grid drawing from this, we will determine ...

- a mapping (assignment) of the four incident edges at c to the four half-lines (see Section 5.1.1 and Algorithm 7).
- points with integer coordinates on these four half-lines that become the new bend points (see Section 5.1.2).

This choice will not create new crossing points.

5.1.1. Assignment of edges to axis-parallel half-lines

In the step described in Section 5.1.2, we will redraw the two edges crossing in c so that they have line segments on the two axis-parallel lines through c at c. To this end,

Algorithm 7: Find an assignment of edges to axis-parallel half-lines for each crossing

```
Input  : Graph $G' = (V', E')$,
         drawing $\Gamma'$ of $G'$,
         independent set of degree-4 crossing vertices $C \subseteq V'$
Output: Assignment $a : E' \to \{\text{null}, r_\uparrow, r_\leftarrow, r_\downarrow, r_\rightarrow\}$
1  Set $left(q_\nearrow) = r_\uparrow, left(q_\nwarrow) = r_\leftarrow, left(q_\swarrow) = r_\downarrow, left(q_\searrow) = r_\rightarrow$
2  Set $right(q_\nearrow) = r_\rightarrow, right(q_\nwarrow) = r_\uparrow, right(q_\swarrow) = r_\leftarrow, right(q_\searrow) = r_\downarrow$
3  Initialize a map $a$
4  foreach $e \in E'$ do
5      Set $a(e) := \text{null}$
6  foreach $c \in C$ do
7      if each of the four quadrants around $c$ contains an incident edge of $c$ then
8          findAssignmentInCase1()
9      else
10         Let $q$ be an empty quadrant around $c$
11         if both neighboring quadrants of $q$ contain an incident edge of $c$ then
12             findAssignmentInCase2()
13         else
14             findAssignmentInCase3()
15 return $a$
```

Algorithm 8: [Subroutine of Algo. 7, which can read and write all its variables] findAssignmentInCase1()

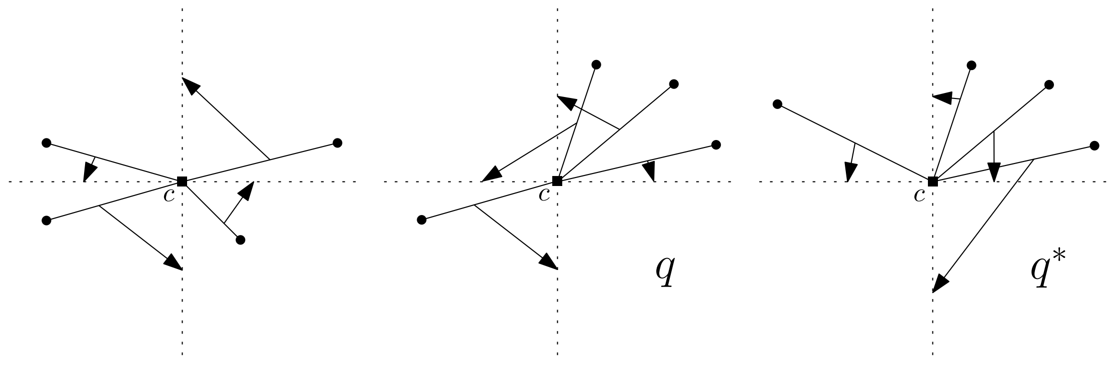
```
1  foreach Quadrant $q$ around $c$ do
       // Note: If edges lie on the axis-parellel half-lines, they lie
          in two quadrants. Thus, there might be more edges in $q$, but
          according to the ‘‘if’’-clause, every edge can be assigned to
          exactly one quadrant. Use this bijection here.
2      Let $e$ be the only edge in $q$
3      Set $a(e) := left(q)$
```

Algorithm 9: [Subroutine of Algo. 7, which can read and write all its variables] findAssignmentInCase2()

```
1 Let $q_l$ and $q_r$ be the neighboring quadrants of $q$ so that $right(q_l) = left(q)$ and
    $left(q_r) = right(q)$
2 Let $e_l$ and $e_r$ be the closest edges to $q$ in $q_l$ and $q_r$, respectively
3 Set $a(e_l) := left(q)$ and $a(e_r) := right(q)$
4 Let $e_3$ and $e_4$ be the remaining two incident edges of $c$ so that $e_l$, $e_3$, $e_4$, $e_r$ is the
    counter-clockwise ordering of these edges around $c$ in $\Gamma'$
5 Let $q_{\text{opposite}}$ be the only quadrant around $c$ that is no neighbor of $q$
6 Set $a(e_3) := right(q_{\text{opposite}})$ and $a(e_4) := left(q_{\text{opposite}})$
```

Algorithm 10: [Subroutine of Algo. 7, which can read and write all its variables] findAssignmentInCase3()

```
1 if $q$ has a neighboring quadrant containing two or more edges then
2     Set $q^* := q$
3 else
4     Let $q^*$ be an empty neighboring quadrant of $q$ that has another neighboring
          quadrant with two or more edges // Exists, because $q$ and its
          neighbors are already three of four quadrants and the four
          incident edges of $c$ are in at least one quadrant
5 Let $e_1$ and $e_2$ be two edges in one neighboring quadrant of $q^*$ that are closest to
    $q^*$ and $e_1$ comes before $e_2$ in the counter-clockwise ordering around $c$
6 Set $a(e_1) := right(q^*)$ and $a(e_2) := left(q^*)$
7 Let $e_3$ and $e_4$ be the remaining two incident edges of $c$ so that $e_3$ comes before $e_4$
    is the counter-clockwise ordering around $c$ in $\Gamma'$
8 Let $q^*_{\text{opposite}}$ be the only quadrant around $c$ that is no neighbor of $q^*$
9 Set $a(e_3) := right(q^*_{\text{opposite}})$ and $a(e_4) := left(q^*_{\text{opposite}})$
```



(a) Case 1. (b) Case 2. (c) Case 3.

Fig. 5.2.: Examples for the three cases of Algorithm 7.

we assign each of the four edges incident to c to one of the four axis-parallel half-lines originating in c.

For the assignment step, we introduce some abbreviations. Consider a crossing vertex c. The four axis-parallel half-lines (rays) originating in c are called $r_\uparrow$, $r_\leftarrow$, $r_\downarrow$, and $r_\rightarrow$, where $r_\uparrow$ is the half-line parallel to the y-axis going against $+\infty$ and $r_\leftarrow$, $r_\downarrow$, and $r_\rightarrow$ follow in the counter-clockwise order around c. These half-lines separate the drawing area into four sectors called *quadrants* in the following. The quadrant between $r_\uparrow$ and $r_\leftarrow$ is $q_\nwarrow$, and the following quadrants in counter-clockwise order around c are $q_\swarrow$, $q_\searrow$ and $q_\nearrow$. The points on the axis-parallel half-lines belong to both incident quadrants.

In Algorithms 7–10, we give a procedure that assigns the four edges to the four half-lines at each crossing vertex c. We distinguish three cases to assign them. If there is one incident edge per quadrant around c, we assign each edge to the half-line that bounds its quadrant from the left (*left* means here in counter-clockwise direction). For an illustration, see Fig. 5.2a. We will refer to this case as *Case 1*.

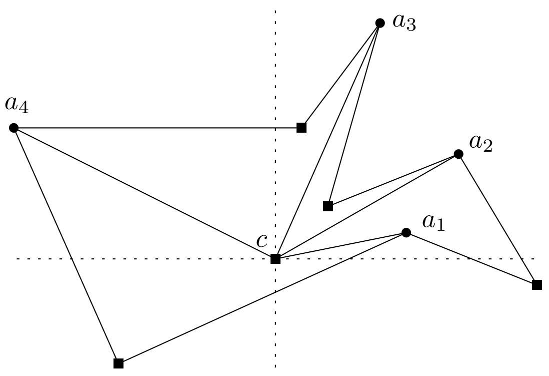
Otherwise, there is a quadrant q that contains none of the four edges that are incident to c. If the two neighboring quadrants of q contain at least one edge incident to c each, take the edge from each of these two quadrants that are closest to q and assign them to the two half-lines bounding q—each of the two edges to the half-line that bounds also the quadrant it is in. Assign the remaining two incident edges to the remaining two half-lines in the same counter-clockwise order as both appear around c. For an illustration, see the example in Fig. 5.2b. We will refer to this case as *Case 2*.

If q has a neighboring quadrant that contains no edge incident to c, we define q^* as q if q has a neighboring quadrant with at least two edges, and q^* as an empty neighboring quadrant of q otherwise. Now, q^* has a neighboring quadrant with two or more edges. Assign these two edges that are closest to q^* to the half-lines bounding q^* in the same counter-clockwise order as both appear around c. Assign the remaining two incident edges to the remaining two half-lines in the same counter-clockwise order as both appear around c. For an illustration, see the example in Fig. 5.2c. We will refer to this case as *Case 3*.

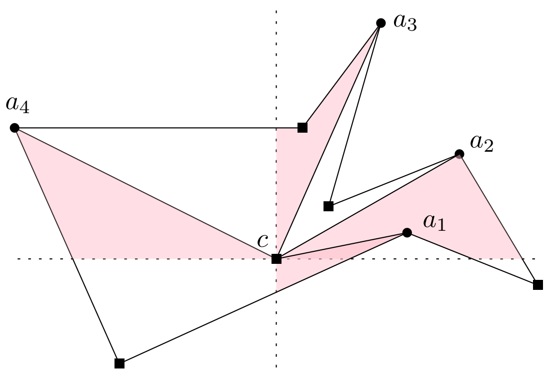
5.1.2. Placement of bend points

Interpreting the crossing vertices in Γ' as crossing points, we have a 1-bend drawing of the input graph G with some additional dummy edges and dummy vertices for the divided kites and an assignment of edges to half-lines. To make this a 2-bend RAC drawing, we redraw the crossing edges inside the divided quadrangles.

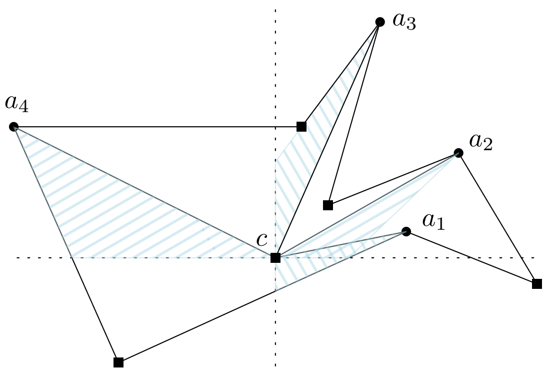
Therefore, consider in Γ' for each crossing vertex c each edge $\{a, c\}$. We will replace this straight line edge by a 1-bend edge by placing a bend point b on the assigned axis-parallel half-line so that the new edge crosses no other redrawn edge or other edge. First, we determine the available region into which we can put this 1-bend edge (two line segments). We consider only the area inside the divided kite and only the area on this side of $\{a, c\}$ that points towards the assigned half-line. This region between $\{a, c\}$ and its assigned half-line inside the divided kite defines a polygon. An example of these polygonal available areas is given in Fig. 5.3b and Fig. 5.4a. Note that these areas might



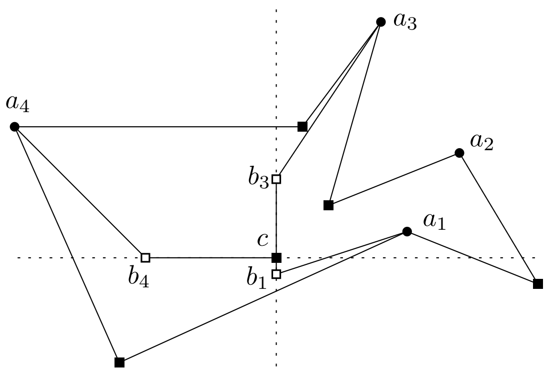
(a) A divided kite as obtained from the algorithm by Harel and Sardas [HS98].



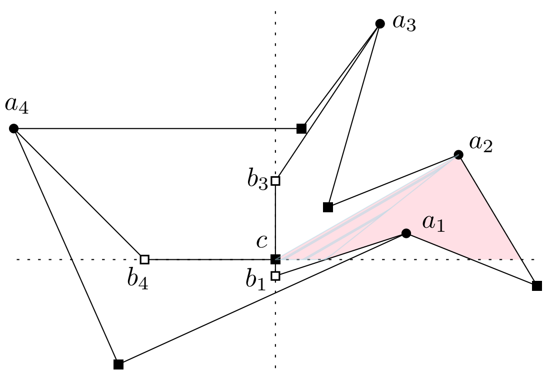
(b) Polygons of available area for each pair of edge and assigned half-line.



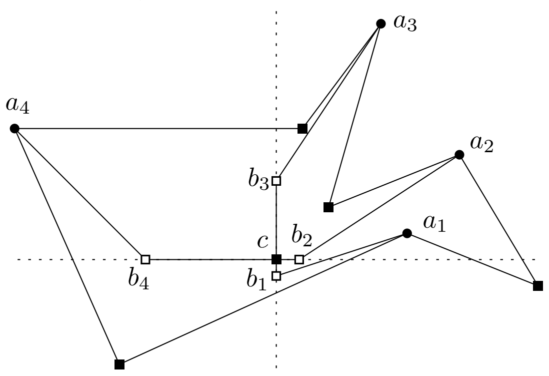
(c) Triangles for valid edge placing.



(d) After the insertion of the three bend points that do not depend on the drawing of other bent edges.

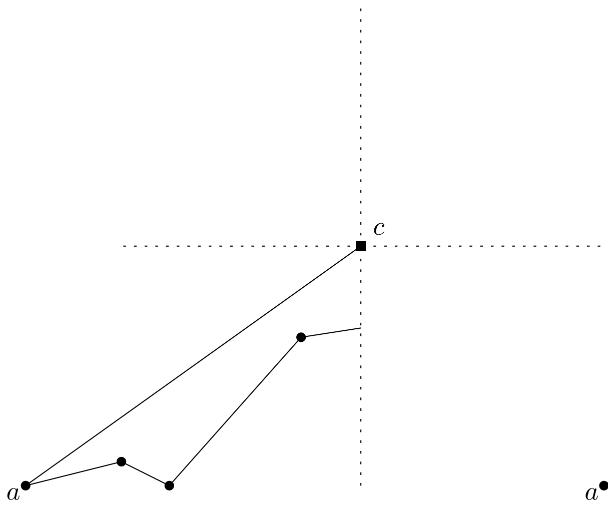


(e) Polygon of available area and triangle for valid edge placing for the edge $\{a_2, c\}$ which depends on the drawing of the edge $\{a_1, c\}$.

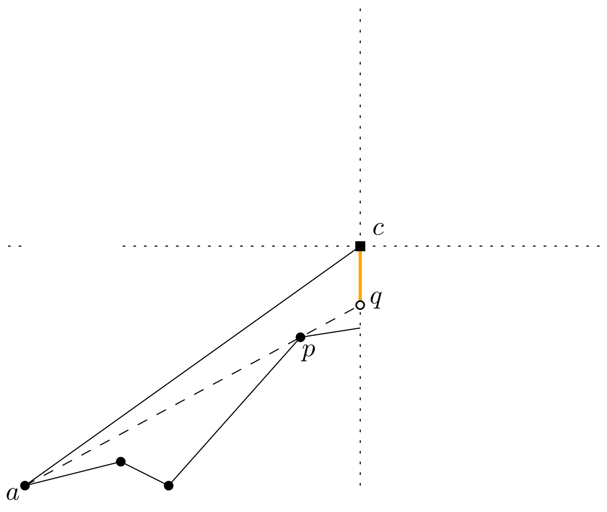


(f) After the insertion of the bend point b_2.

Fig. 5.3.: Example of the postprocessing step for transforming a planarized crossing to a 2-bend RAC crossing.



(a) Polygon of available area.



(b) Triangle for valid edge placing with points p and q.

Fig. 5.4.: Example of a polygon of available area in which we determine the points p and q and with them the triangle for valid edge placing and the line segment $\overline{cq}$.

overlap (as they do once in the example).

Observe that there is only a triangle inside this polygon in which the new line segment $\overline{ab}$ can be placed. These valid triangles for placing edges are depicted in Fig. 5.3c and Fig. 5.4b. Again, they might overlap. Such a triangle covers the angular region around a where we can place this line segment. This angle cannot become arbitrarily small because every determining point lies on a grid point. The triangle is determined by a, c and a corner point p of the polygon of the available area. The point p is the corner point (excluding a and c) for which the angle between $\overline{ac}$ and $\overline{ap}$ inside the polygon is the smallest. Let q be the intersection point of the line through $\overline{ap}$ and the axis-parallel half-line to which $\{a, c\}$ was assigned. One can see q as the projection of p onto the assigned half-line of a seen from a. An example is given in Fig. 5.4a and Fig. 5.4b. Note that we have a degenerated case if a already is on the assigned half-line. Then, the polygon of available area has degree two. It is a line segment with endpoints a and c. In this case let $a = p = q$. Moreover, note that p can be equal to q, because the intersection of the assigned axis-parallel half-line and an edge e of the divided kite is also a corner point of this polygon. This is the only case where p may not be a grid point (We ignore overlapping polygons of available area for the moment). In this case, we name the endpoint of e that is inside the polygon of available area p'_{in} and the endpoint that is outside p'_{out}.

We will place the bend point b onto the line segment $\overline{cq}$. We proceed in two steps. Let Γ' be drawn on a grid of size $\tilde{n} \times \tilde{n}$.

First, we consider only the edges that are assigned to half-lines so that their assignment does not depend on the assignment of other edges to half-lines. In Fig. 5.3c, this would be $\{a_1, c\}$, $\{a_3, c\}$, and $\{a_4, c\}$, but not $\{a_2, c\}$, which depends on the course of $\{a_1, c\}$. We refine the grid by a factor of $\tilde{n}$ in each dimension. Now, the interior of the line segment $\overline{qc}$ contains at least one grid point (we show this later in Lemma 5.4). Pick any of them and place b on it. Remove the line segment $\overline{ac}$ from the drawing and insert the two line segments $\overline{ab}$ and $\overline{bc}$ instead. We will refer to the resulting drawing as Γ''.

Second, we consider the other edges with an assignment to half-lines that depends on the assignment of edges drawn in the step before. Again, we refine the grid by a factor of $\tilde{n}$ in each dimension. Now, we have a grid of size $\tilde{n}^3 \times \tilde{n}^3$. For edges considered now, we compute new polygons of available area and triangles for valid edge placing since we need to take the 1-bend edges into account that were inserted in the previous step. Again, we have a line segment $\overline{qc}$ and its interior contains at least one grid point now (we show this later in Lemma 5.5). Pick any of them and place b on it. Remove the line segment $\overline{ac}$ from the drawing and insert the two line segments $\overline{ab}$ and $\overline{bc}$ instead.

After these two steps, we have replaced each straight-line edge by a 1-bend edge because the depth of dependencies of assigned edges to half-lines is at most one. We will show this by Corollary 5.3.

Finally, remove the dummy edges and dummy vertices that bound the empty kites and interpret the crossing vertices as crossing points. Let the resulting 2-bend RAC drawing be Γ and return Γ.

5.2. Analysis

In this section, we analyze our algorithm presented in this chapter to prove Theorem 5.1. We focus on the two main steps of the algorithm in Section 5.1. We show that we obtain a valid assignment of the edges to half-lines in Section 5.2.1 and that we place the bend points properly so that they are on a refined grid and their placement does not cause new edge crossings in Section 5.2.2.

5.2.1. Assignment of edges to axis-parallel half-lines

For our purposes, we require a found assignment of the four incident edges of a crossing vertex c to the four axis-parallel half-lines originating at c to be *valid.* We call an assignment *valid* if there is a way of redrawing each of the four assigned edges as a 1-bend edge with the second part running along its assigned half-line so that it does not conflict with (touch) another redrawn or original edge and for each of these four half-lines per crossing point there is exactly one assigned edge (the assignment of the four incident edges to the four axis-parallel half-lines is bijective at each crossing point).

We want to find sufficient conditions to have a valid assignment. Assume we have an assignment $a : E' \to \{\text{null}, r_\uparrow, r_\leftarrow, r_\downarrow, r_\rightarrow\}$. Consider an edge $e = \{v, c\}$ incident to a crossing point c. If e lies on one of these four half-lines, it must not be assigned to the opposite half-line o because otherwise it would conflict with c when we bend it somewhere on o.

Excluding this case for every edge, the assigned half line $h := a(e)$ of e is either to the right or to the left of e seen from c to v or h contains e. This way, we can define an angular sector from e to h (or the other way round) at c that is less than 180 degrees. If no other axis-parallel half-line of c and no other incident edge of c lies within this sector, obviously this assigned edge does not cause a conflict. If there is another incident edge e' inside this sector, a conflict can be avoided if $h' := a(e')$ lies outside this sector and h' lies to the right of e' if and only if h lies to the right of e. This is because the redrawn edge e', bent at h', intersects h then. This intersection point can be arbitrarily close to c, but never reach c. Thus, the part of h from c to this intersection point can still be used to place a bend point of the redrawn edge e on. In the following, we say in such a case that the assignment of e *depends* on the assignment of e'. The other way round, the angular sector between e and h contains another axis-parallel half-line h' of c, but no other incident edge. This does not cause a conflict if h' is to the right of its edge e' if and only if h is to the right of e. Then, the assignment of e' depends on the assignment of e. Observe that a assignment that fulfills this has at most a dependency-depth of one, i.e., two assigned edges where the assignment of one of them depends on the other one because otherwise there would be an angular sector that contains in its interior an edge and a half-line. In other words, if the assignment of an edge e depends on the assignment of a second edge e', the assignment of e' cannot depend on the assignment of a third edge e''. Using these sufficient criteria for a valid assignment, we obtain the following Lemma.

Lemma 5.2. Algorithm 7 finds a valid assignment of edges to axis-parallel half-lines for G', Γ', and its set of crossing vertices C in $O(|E'| + |C|)$ time.

Proof. To prove this, we ensure that in all of the three cases in which the algorithm assigns the edges to half-lines for a crossing point c, the assignment is valid. Clearly, there is in every case at every crossing point for every axis-parallel half line of c an incident edge assigned to it.

In Case 1, if each quadrant of c contains an edge incident to c, each incident edge is assigned to its direct left neighbor. This means that no incident edge is assigned to an opposite half-line and no angular sector contains any other axis-parallel half line or any other incident edge of c. Thus, this assignment is valid.

In Case 2, if for an empty quadrant q of c both neighboring quadrants of q contain an incident edge of c, the two closest edges are assigned to the bounding axis-parallel half lines of q. Because they are direct neighbors, there is no other half-line and no other incident edge in these two angular sectors and they are not on opposite sides with respect to c. It remains to show that the assignment of the two remaining incident edges does not cause a conflict. They cannot be assigned to opposite half-lines because then they would have been in q. Assume for contradiction that an angular sector from an edge e to an axis-parallel half-line h contains both: The other considered incident edge e' and its assigned half line h'. This cannot happen because then e would either come before e' in the counter-clockwise ordering around c while h' comes before h or come after e' in the counter-clockwise ordering around c while h' comes after h. In both cases the algorithm would have assigned e to h' and e' to h and both angular sectors would be empty then. Furthermore, assume for contradiction that, without loss of generality, the angular sector from an incident edge e to its assigned half line h contains an incident edge e' that has its assigned half line h' at its left while h is to the right of e. Hence, the edge e comes before e' in the counter-clockwise order around c. But again, then the algorithm would have assigned e to h' and e' to h and the angular sectors would be empty. Thus, this assignment is valid as well.

In Case 3, an empty quadrant q^* of c has a neighboring quadrant q' that contains at least two incident edges of c. The two edges in q' that are closest to q^* are assigned to the bounding half lines of $q*$. The one of these two edges that is closest to q^*, let it be e, is assigned to the more distant bounding half line of q^*. Thus, the angular sector between both contains the axis-parallel half-line to which the other considered incident edge e' is assigned. The sector between e' and this closer axis-parallel half-line contains the edge e. So, the assignment of e' depends on the assignment of e. Because both edges have their assigned half lines on the same side (left or right), this still fulfills the sufficient criteria for a valid assignment. It remains to show that the assignment of the two remaining incident edges does not cause a conflict. As the assignment of these two is the same as in Case 2, the argumentation for the remaining two edges holds here as well. Therefore, in any case the assignment returned by Algorithm 7 is valid.

Initializing the map a with null-values for each edge needs a running time of $O(|E'|)$. The "foreach $c \in C$"–loop is run $|C|$ times. In each run, the if-checks and the assignments can be done in constant time because in any case there are checked and processed at most four quadrants and four incident edges per crossing point. Thus, the algorithm runs in $O(|E'| + |C|)$ time. $\square$

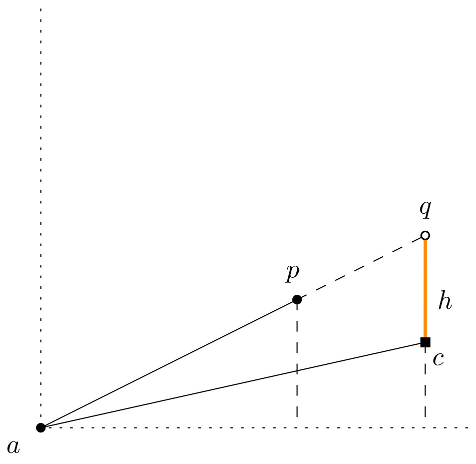
Corollary 5.3. The assignment found by Algorithm 7 has a dependency-depth of at most one. This means that if the assignment of an edge e depends on the assignment of a second edge e', the assignment of e' cannot depend on the assignment of a third edge e''.

Proof. Algorithm 7 fulfills the sufficient criteria for a valid assignment described above. Obviously, no longer sequences of dependencies are possible, because in every angular sector between an edge and its assigned half line, there can only either be another incident edge of the respective crossing point or another axis-parallel half line through this crossing point, but not both. $\square$

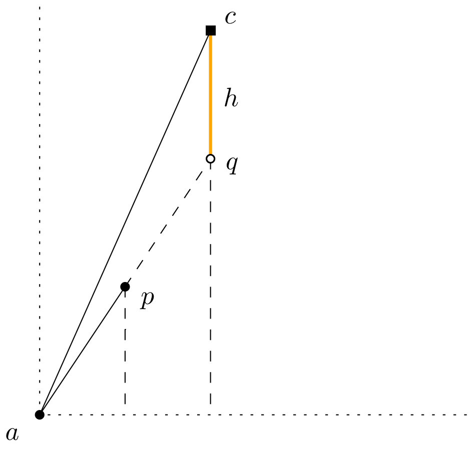
5.2.2. Placement of bend points

In our algorithm, we bend each each edge $\{a, c\}$ that is incident to a crossing vertex c via a point b on the the assigned half-line, or more precisely on the interior of the line segment $\overline{qc}$. For the definition of p and q, see the description of the algorithm in Section 5.1. To place the bend points on a grid point, we need at least one grid point on this line segment. Using the current grid of size $O(\tilde{n}) \times O(\tilde{n})$ onto which the algorithm by Harel and Sardas placed the vertices is not enough. In the following, the x- and y-coordinate of a point p is denoted by $x(p)$ and $y(p)$, respectively.

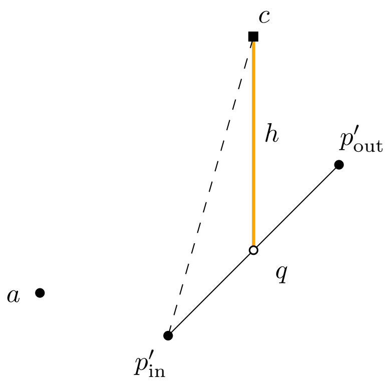
Lemma 5.4. Let Γ' be drawn on a $\tilde{n} \times \tilde{n}$ sized grid. With the previous naming, the interior of each line segment $\overline{qc}$ contains at least one grid point of the refined $\tilde{n}^2 \times \tilde{n}^2$ grid.



(a) Case A1.



(b) Case A4.



(c) Case B.

Fig. 5.5.: Different situation for the analysis of the value of h. Here, h is the length of the line segment $\overline{cq}$.

Proof. Without loss of generality, we can assume that $\overline{qc}$ is parallel to the y-axis (a vertical line segment). If the x-coordinate of a is equal to the x-coordinate of q and c, we have the degenerated case where $q = a$. We do not need to bend the edge $\{a, c\}$ in our algorithm, but, for the completeness of the proof, we can easily see that there are at least $\tilde{n} - 1$ grid points on the refined $\tilde{n}^2 \times \tilde{n}^2$ grid because c and $q = a$ are grid points of the coarser $\tilde{n} \times \tilde{n}$ grid. So, without loss of generality, we can assume that $x(a) < x(q) = x(c)$, because mirroring the drawing across the line through $\overline{qc}$ does not change the structure of the drawing. We can also assume that a has coordinates $(0, 0)$. Without loss of generality, we can assume that $y(c) \geq 0$. If $y(c) = 0$, we can furthermore assume $y(q) > 0$ (both by the argument of mirroring across the x-axis). If $y(c) > 0$ and $y(q) < 0$, we are fine because c and $(x(c), 0)$ are both grid points of the coarser grid. Between them, there is more than one grid point of the finer grid. So, we continue with $y(c) \geq 0$ and $y(q) \geq 0$. For convenience, we will work with coordinates on the coarser $O(\tilde{n}) \times O(\tilde{n})$ grid in the following case distinction. Moreover, observe that c cannot be a vertex on the upper- or lowermost row of the grid or a vertex on the left- or rightmost column of the grid since it is enclosed by the dummy edges of a divided quadrangle. Therefore, we know that the difference in the x- and in the y-coordinate of a and another vertex of the drawing is less than $\tilde{n}$. In particular, we know:

$$x(p) - x(a) = x(p) \leq x(q) < \tilde{n} \tag{5.1}$$

Now, we distinguish two cases.

Case A The point p is a grid point. The points a, p and q are collinear in any case. Clearly, a is the leftmost point of them. For $x(p) \geq y(p)$ and $x(c) \geq y(c)$, the slopes of $\overline{ap}$ and $\overline{ac}$ are values of the Farey sequence of order $\tilde{n} - 1$. The Farey sequence of order $\tilde{n} - 1$ is the sequence of all reduced fractions from 0 to 1 with numerator and denominator being positive integers $\leq \tilde{n} - 1$ [SF07]. The slopes are $\frac{y(p)}{x(p)}$ and $\frac{y(c)}{x(c)}$. One can imagine all these possible slopes going out from a as rays. Without loss of generality, we can assume that the reduced fractions of $\frac{y(p)}{x(p)}$ and $\frac{y(c)}{x(c)}$ are neighbored fractions in the Farey sequence and neighbored rays in the picture of the rays going out from a. We also assume that $\frac{y(p)}{x(p)}$ and $\frac{y(c)}{x(c)}$ are reduced fractions because for a multiple of one of the Farey numbers, the line segment $\overline{qc}$ could only be longer and have more grid points of the finer grid on it but not fewer.

We distinguish more subcases:

Case A1 $y(q) \geq y(c)$, and $\frac{y(p)}{x(p)}$ and $\frac{y(c)}{x(c)}$ are neighbors in the Farey sequence. A sketch is given in Fig. 5.5a.
Then, we have:

$$h = \|\overline{qc}\| = y(q) - y(c) \tag{5.2}$$

$$y(q) = \frac{y(p)}{x(p)} \cdot x(c) \tag{5.3}$$

Putting this together, we get:

$$h = \frac{y(p)}{x(p)} \cdot x(c) - y(c) = x(c) \cdot \left(\frac{y(p)}{x(p)} - \frac{y(c)}{x(c)} \right) \tag{5.4}$$

One property of neighboring numbers in the Farey sequence $\frac{a}{b}$ and $\frac{c}{d}$ with $\frac{a}{b} < \frac{c}{d}$ is:

$$\frac{c}{d} - \frac{a}{b} = \frac{1}{bd} \tag{5.5}$$

Because of $y(q) \geq y(c)$, we know that $\frac{y(p)}{x(p)} > \frac{y(c)}{x(c)}$. Applying this leads to:

$$h = x(c) \cdot \left(\frac{y(p)}{x(p)} - \frac{y(c)}{x(c)} \right) = x(c) \cdot \frac{1}{x(c) \cdot x(p)} = \frac{1}{x(p)} > \frac{1}{\tilde{n}} \tag{5.6}$$

Case A2 $y(q) \leq y(c)$, and $\frac{y(p)}{x(p)}$ and $\frac{y(c)}{x(c)}$ are neighbors in the Farey sequence.
This this almost the same as case A1, only multiplied with -1 because now we have $\frac{y(p)}{x(p)} < \frac{y(c)}{x(c)}$. Precisely, we have:

$$\begin{aligned} h &= y(c) - y(q) \\ &= y(c) - \frac{y(p)}{x(p)} \cdot x(c) \\ &= x(c) \cdot \left(\frac{y(c)}{x(c)} - \frac{y(p)}{x(p)} \right) \\ &= x(c) \cdot \frac{1}{x(c) \cdot x(p)} = \frac{1}{x(p)} > \frac{1}{\tilde{n}} \end{aligned} \tag{5.7}$$

Case A3 $y(q) \geq y(c)$, and $\frac{y(p)}{x(p)}$ and $\frac{y(c)}{x(c)}$ are not numbers of the Farey sequence because their numerator is greater than their denominator. Their reciprocals are neighbors in the Farey sequence. These numbers can be seen as part of an extension of the Farey sequence from 1 to $+\infty$.
This case is also similar to A1. Equations 5.2-5.4 still hold, but we need to be more careful with Equation 5.6 because $\frac{y(p)}{x(p)}$ and $\frac{y(c)}{x(c)}$ are not numbers of the Farey sequence. An implication from Equation 5.5 is the following (Naming as in Equation 5.5):

$$\frac{c}{d} - \frac{a}{b} = \frac{bc - ad}{bd} = \frac{1}{bd} \Rightarrow bc - ad = 1 \tag{5.8}$$

Plugging in the Farey numbers $\frac{x(p)}{y(p)}$ and $\frac{x(c)}{y(c)}$, where we know $\frac{x(p)}{y(p)} < \frac{x(c)}{y(c)}$, we get:

$$y(p) \cdot x(c) - x(p) \cdot y(c) = 1 \tag{5.9}$$

Using this, we can continue with Equation 5.4:

$$\begin{aligned} h &= x(c) \cdot \left(\frac{y(p)}{x(p)} - \frac{y(c)}{x(c)} \right) \\ &= x(c) \cdot \frac{y(p) \cdot x(c) - x(p) \cdot y(c)}{x(c) \cdot x(p)} \\ &= x(c) \cdot \frac{1}{x(c) \cdot x(p)} \\ &= \frac{1}{x(p)} > \frac{1}{\tilde{n}} \end{aligned} \tag{5.10}$$

Case A4 $y(q) \leq y(c)$, and $\frac{y(p)}{x(p)}$ and $\frac{y(c)}{x(c)}$ are not numbers of the Farey sequence because their numerator is greater than their denominator. Their reciprocals are neighbors in the Farey sequence. A sketch is given in Fig. 5.5b.
This case is analog to Case A3 like the analogy of Case A2 to Case A1. Again, we can multiply with -1 or alternatively swap all occurrences of p and c.

Case B The point p is no grid point. This situation may only occur if $p = q$. It is the situation described earlier where we have vertices p'_{in} and p'_{out} as depicted in Fig. 5.5c. Clearly, we have a similar situation as in Case A. Here, p'_{in} is in the position as is a in Case A and p'_{out} is in a similar position as is p in Case A. The points p'_{in} and p'_{out} are vertices of G' and, thus, also grid points of the $\tilde{n} \times \tilde{n}$ grid. The only difference is the order of the points a, p, q and p'_{in}, q, p'_{out} on each common line. Observe that the formulas given in Case A still hold if q is in between p'_{in} and p'_{out} instead of being to the right of both. Therefore, by doing the same analysis as in case A with exchanged a and p, we get the same result.

So, for both cases and each subcase, we get the result that the line segment $\overline{qc}$ has length strictly greater than $\frac{1}{\tilde{n}}$. By refining the $\tilde{n} \times \tilde{n}$ grid by a factor of $\tilde{n}$ in each dimension as we do in our algorithm, we get a $\tilde{n}^2 \times \tilde{n}^2$ grid where each grid point of the coarser grid is also a grid point of the finer grid. Each crossing point c is a grid point of both grids. Going towards the q of each of the four half-lines, we reach the next grid point after a distance of $\frac{1}{\tilde{n}}$ because we are on a axis-parallel line with integer coordinates on the coarser grid. This cannot be q because $\overline{qc}$ is longer. Therefore, the interior of the line segment $\overline{qc}$ contains at least one grid point. $\square$

If we did not have overlapping triangles for valid edge placing, we would have shown now that every 1-plane graph admits a 2-bend RAC drawing with everything on a grid of size $O(n^2) \times O(n^2)$. But since we have overlaps of these triangles, we do a second step in our algorithm by refining the grid again by a factor of $\tilde{n}$ in each dimension and computing a bend point for the remaining edges that depend on the redrawing of other edges.

In the example of Fig. 5.3d, the placement of the bend point b_2 of the edge $\{a_2, c\}$ depends on the placement of the bend point b_1 of the edge $\{a_1, c\}$. The placement

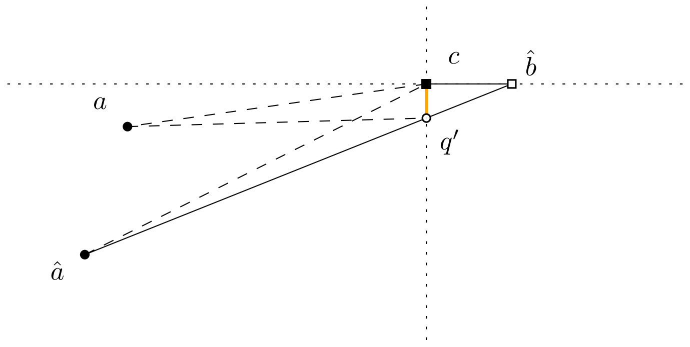


Fig. 5.6.: The bend point of the edge $\{a, c\}$ is placed on a grid point of the interior of the line segment $\overline{cq'}$. The point q' depends on the placement of $\hat{b}$, which is the bend point of the edge $\{\hat{a}, c\}$.

of b_1 does not depend on any other placement of a bend point. A circular sequence of dependencies is not possible since the dependency depth is at most one (see Corollary 5.3).

Consider the edges that are redrawn in this second step. For each such edge, we have computed a polygon of available area and a triangle for valid edge placing in Γ. There, we have determined the points p and q. This polygon and this triangle may change when we take the redrawn edge into account on which this edge placement depends. The intersection point of the assigned half-line and the line through the already placed bend point and its vertex may become a new relevant point (what has been p and q above). If this intersection point becomes such a relevant point for the recomputed triangle (it makes it smaller), let this point be q'. If this intersection point does not affect this triangle, set $q' = q$ with q being the corner point of the triangle computed in first step.

Now, we show that refining the grid again by a factor of $\tilde{n}$ in both dimensions is enough to have also these depending bend points on grid points.

Lemma 5.5. Let Γ'' be drawn on a $\tilde{n}^2 \times \tilde{n}^2$ sized grid. Consider only edges for which the placement of the bend point depends on the redrawing of another edge in Γ''. With the previous naming, the interior of each line segment $\overline{q'c}$ contains at least one grid point of the refined $\tilde{n}^3 \times \tilde{n}^3$ grid.

Proof. All of the following coordinates are relative to the grid of size $\tilde{n}^2 \times \tilde{n}^2$ that has been refined once. Let the placement of the bend point of the edge $\{a, c\}$ depend on the bend point $\hat{b}$ of the edge $\{\hat{a}, c\}$. If the triangle for valid edge placing of $\{a, c\}$ was not shrunk after placing $\hat{b}$, then q' equals q and the analysis of Lemma 5.4 holds here as well. This means that the the interior of $\overline{q'c}$ contains a grid point of the $\tilde{n}^2 \times \tilde{n}^2$ sized grid and, thus, also of the $\tilde{n}^3 \times \tilde{n}^3$ sized grid.

Otherwise, we can assume, without loss of generality, that $x(\hat{a}) = 0$ and $y(\hat{a}) = 0$. Furthermore, we can assume that $x(c) \geq 0$ and $y(c) \geq 0$ because mirroring across some axis-parallel line does not change the structure of the drawing. We assume, without loss of generality, that $\hat{b}$ lies on the half-line originating at c and running to positive infinity in the x-dimension because $\overline{\hat{a}\hat{b}}$ crosses some other axis-parallel half-line (here: the one going to negative infinity in y-dimension) and, again, mirroring does not change the structure of the drawing. This implies $y(c) > y(\hat{a})$. Our current situation is depicted in Fig. 5.6. Now, we analyze how short the line segment $\overline{q'c}$ can become in the worst case.

It will become shorter if ...

- the x-distance $x(\hat{b}) - x(c)$ becomes smaller or
- the y-distance $y(c) - y(\hat{a})$ becomes smaller or
- the x-distance $x(c) - x(\hat{a})$ becomes greater.

So, $\overline{q'c}$ will be shortest if we assume the most extremes of these values. They are:

- x-distance $x(\hat{b}) - x(c) = 1$, and
- y-distance $y(c) - y(\hat{a}) = \tilde{n}$ (it cannot become smaller because both are points of the coarser $\tilde{n} \times \tilde{n}$ grid and $y(c) > y(\hat{a})$), and
- x-distance $x(c) - x(\hat{a}) = (\tilde{n} - 1) \cdot \tilde{n}$ (This is because both are grid points on the coarser $\tilde{n} \times \tilde{n}$ grid. Since $\hat{b}$ is on the right side of both, they cannot both be outermost grid points and, thus, they can only have a distance of $\tilde{n} - 1$ on the initial coarser grid and $(\tilde{n} - 1) \cdot \tilde{n}$ on the refined grid of Γ''.)

Plugging this together, we get as slope m of $\overline{\hat{a}\hat{b}}$:

$$m = \frac{\tilde{n}}{\tilde{n}^2 - \tilde{n} + 1} \tag{5.11}$$

Following this, we determine $y(q')$:

$$y(q') = \left(\tilde{n}^2 - \tilde{n}\right) \cdot m = \left(\tilde{n}^2 - \tilde{n}\right) \cdot \frac{\tilde{n}}{\tilde{n}^2 - \tilde{n} + 1} = \frac{\tilde{n}^3 - \tilde{n}^2}{\tilde{n}^2 - \tilde{n} + 1} \tag{5.12}$$

Now, we can compute the length of the line segment $\overline{q'c}$ this way:

$$\begin{aligned} y(c) - y(q') &= \tilde{n} - \frac{\tilde{n}^3 - \tilde{n}^2}{\tilde{n}^2 - \tilde{n} + 1} = \frac{\tilde{n}^3 - \tilde{n}^2 + \tilde{n} - \tilde{n}^3 + \tilde{n}^2}{\tilde{n}^2 - \tilde{n} + 1} \\ &= \frac{\tilde{n}}{\tilde{n}^2 - \tilde{n} + 1} = \frac{1}{\tilde{n} - 1 + \frac{1}{\tilde{n}}} \\ &> \frac{1}{\tilde{n}} \text{ (for } \tilde{n} > 1) \end{aligned} \tag{5.13}$$

With the same argumentation as in Lemma 5.4, we can see that the interior of $\overline{q', c}$ contains always at least a grid point of the refined $\tilde{n}^3 \times \tilde{n}^3$ grid. □

5.3. Proof of the main theorem

Now, we prove the Theorem from the beginning of this chapter using our previous analysis.

Theorem 5.1. *Every 1-planar graph $G = (V, E)$ with n vertices admits a 1-planar 2-bend RAC drawing with vertices, bend points, and crossing points on a grid of size $O(n^3) \times O(n^3)$, where the bends occur only on edges participating in a crossing. Given a 1-planar embedding $\mathcal{E}(G)$ of G, such a drawing can be computed in $O(n)$ time. The returned drawing preserves the given embedding $\mathcal{E}(G)$ and is placed on a grid of size $(8n'^3 - 48n'^2 + 96n' - 64) \times (4n'^3 - 24n'^2 + 48n' - 32)$, where n' is the number of vertices of G plus 5 times the number of crossings in $\mathcal{E}(G)$. It holds that $n' \leq 6n - 10$.*

Proof. By definition, there is a 1-planar drawing for every 1-planar graph $G = (V, E)$. Using the embedding of this drawing, we can assume, without loss of generality, that we have a given 1-plane graph $(G, \mathcal{E}(G))$ with n vertices and k crossings. According to Lemma 2.19, k is less than or equal to $n - 2$. Use this 1-plane graph as input for the algorithm presented in Section 5.1.

In linear time, we obtain a biconnected planarized graph from $(G, \mathcal{E}(G))$, where each crossing is replaced by a divided kite. This graph has $n' = n + k + 4k \leq 6n - 10$ vertices.

The algorithm by Harel and Sardas [HS98] yields a straight line drawing of the planarized graph in linear time. This drawing is crossing-free and has each original vertex and each crossing vertex on a grid of size $(2n' - 4) \times (n' - 2)$.

In a postprocessing, we replace the straight-line edges incident to the crossing vertices by 1-bend edges. While we do this, we refine the grid by a factor of $(2n' - 4)^2$ in both dimension. By placing the 1-bend edges inside the triangles for valid edge placement, which contain no other edge, we obtain a crossing-free 1-bend drawing with crossing vertices. Lemma 5.2 assures that this is always possible. The refinement of the grid can be done in linear time (recomputing the coordinates of linear many objects). Each divided kite around each crossing vertex has constant complexity. Therefore, we can compute the polygons, triangles and positions for bend points in constant time for linear many crossing vertices. Removing the dummy edges and dummy vertices of the divided kite and the crossing vertices can be done in linear time as well. Thus, the algorithm runs in linear time, i.e., $O(n)$ time.

By Lemma 5.4 and Lemma 5.5, the bend points are grid points of the grid with size $(2n'-4)\cdot(2n'-4)^2 \times (n'-2)\cdot(2n'-4)^2 = (8n'^3-48n'^2+96n'-64) \times (4n'^3-24n'^2+48n'-32)$.

The crossings in the returned drawing are RAC since the crossing edges are on horizontal and vertical lines around the points they cross. As both 1-bend edges in the planarized graph that belong to the same edge in the original graph reach c on the same line, merging these 1-bend edges does not result in 3-bend edges, but in 2-bend edges. Since no other edges are bent, the resulting drawing is a 2-bend drawing. □

By this theorem, we have shown that we can draw any 1-planar graph as 1-planar 2-bend RAC drawing on a grid of polynomial size, a bit more precisely, a grid of size $O(n^3) \times O(n^3)$.

6. Conclusion

In this thesis we have shown the following results.

- The NIC-plane graphs are in $\mathrm{RAC}_1^{\mathrm{poly}}$ (s. Chapter 3). In particular, every NIC-plane graph—and, therefore, also every IC-plane graph— admits a drawing on a grid of quadratic size with at most one bend per edge and only right-angle crossings. Previously, it was known for straight-line RAC drawings of IC- and NIC-planar graphs that some cannot be drawn on a grid of polynomial size [BDE+16, BBH+17].

- The 1-plane graphs are in RAC_1 (s. Chapter 4). Previously, Bekos et al. [BDL+17] showed this in the variable embedding setting, i.e., the 1-planar graphs are in RAC_1. They asked whether it holds in the fixed embedding setting, i.e., for every 1-plane graph as well. We answer their question positively and show this by a small modification of their algorithm.

- The 1-plane graphs are in $\mathrm{RAC}_2^{\mathrm{poly}}$ (s. Chapter 5). In particular, every 1-plane graph admits a drawing on an $O(n^3) \times O(n^3)$ grid with at most two bends per edge and only right-angle crossings. Previously, it was known that every 1-planar graph admits a RAC drawings with at most one bend per edge in exponential area [BDL+17]. Since it is unknown if every 1-planar graph admits a drawing on a grid of polynomial size with one bend per edge and right-angle crossings, we set an upper bound in the number of bends per edge required for drawing a 1-planar graph as polyline RAC drawing in polynomial area.

In the field of polyline RAC drawings of 1-planar graphs and subclasses, there remain some open questions. In particular, this concerns the relationships labeled with a question mark in Fig. 2.4. We ask the following questions.

- In Chapter 5, we have shown that every 1-plane graph admits a 2-bend RAC drawing with vertices, crossing points and bend points on a grid of size $O(n^6)$, where n is the number of vertices in the input graph. Can we improve this grid size to maybe quadratic size?

- Are the 1-planar graphs in $\mathrm{RAC}_1^{\mathrm{poly}}$? This question may be answered in the variable or in the fixed embedding setting.

- In Chapter 3, we give an algorithm for drawing NIC-planar graphs and an example for a drawing generated by this algorithm is given in Appendix A. Can we improve the aesthetics of these drawing by maybe replacing bent edges by (Bézier) curves?

- Are the IC-plane graphs in RAC_0? So far, it is known that this is true for the IC-planar graphs in the variable embedding setting [BDE+16]. We also have concrete ideas how their algorithm can be modified to maintain the given embedding. This will positively answer this question.
- Are RAC_0 and $\mathrm{RAC}_1^{\mathrm{poly}}$ incomparable or is RAC_0 a subset of $\mathrm{RAC}_1^{\mathrm{poly}}$?
- Are RAC_0 and $\mathrm{RAC}_2^{\mathrm{poly}}$ incomparable or is RAC_0 a subset of $\mathrm{RAC}_2^{\mathrm{poly}}$?
- What is the relationship between RAC_1 and $\mathrm{RAC}_2^{\mathrm{poly}}$?

Bibliography

[ABK13] Md. Jawaherul Alam, Franz J. Brandenburg, and Stephen G. Kobourov: Straight-line grid drawings of 3-connected 1-planar graphs. In Stephen K. Wismath and Alexander Wolff (editors): *Graph Drawing - 21st International Symposium, GD 2013*, volume 8242 of *Lecture Notes in Computer Science*, pages 83–94. Springer, 2013. `https://doi.org/10.1007/978-3-319-03841-4_8`.

[ABS12] Evmorfia N. Argyriou, Michael A. Bekos, and Antonios Symvonis: The straight-line RAC drawing problem is NP-hard. *J. Graph Algorithms Appl.*, 16(2):569–597, 2012. `https://doi.org/10.7155/jgaa.00274`.

[AFK$^+$12] Karin Arikushi, Radoslav Fulek, Balázs Keszegh, Filip Moric, and Csaba D. Tóth: Graphs that admit right angle crossing drawings. *Comput. Geom.*, 45(4):169–177, 2012. `https://doi.org/10.1016/j.comgeo.2011.11.008`.

[Alb08] Michael O. Albertson: Chromatic number, independence ratio, and crossing number. *Ars Mathematica Contemporena*, 1(1):1–6, 2008.

[BBH$^+$17] Christian Bachmaier, Franz J. Brandenburg, Kathrin Hanauer, Daniel Neuwirth, and Josef Reislhuber: NIC-planar graphs. arXiv, 2017. `http://arxiv.org/abs/1701.04375`.

[BDE$^+$16] Franz J. Brandenburg, Walter Didimo, William S. Evans, Philipp Kindermann, Giuseppe Liotta, and Fabrizio Montecchiani: Recognizing and drawing IC-planar graphs. *Theor. Comput. Sci.*, 636:1–16, 2016. `https://doi.org/10.1016/j.tcs.2016.04.026`.

[BDL$^+$17] Michael A. Bekos, Walter Didimo, Giuseppe Liotta, Saeed Mehrabi, and Fabrizio Montecchiani: On RAC drawings of 1-planar graphs. *Theor. Comput. Sci.*, 689:48–57, 2017. `https://doi.org/10.1016/j.tcs.2017.05.039`.

[CH13] Július Czap and Dávid Hudák: On drawings and decompositions of 1-planar graphs. *Electr. J. Comb.*, 20(2):P54, 2013. `http://www.combinatorics.org/ojs/index.php/eljc/article/view/v20i2p54`.

[CP95] Marek Chrobak and T. H. Payne: A linear-time algorithm for drawing a planar graph on a grid. *Inf. Process. Lett.*, 54(4):241–246, 1995. `https://doi.org/10.1016/0020-0190(95)00020-D`.

[CŠ14] Július Czap and Peter Šugerek: Three classes of 1-planar graphs. arXiv, 2014. https://arxiv.org/abs/1404.1222.

[CYN84] Norishige Chiba, Tadashi Yamanouchi, and Takao Nishizeki: Linear algorithms for convex drawings of planar graphs. In J. Bondy and U. S. R. Murty (editors): *Progress in graph theory*, page 153–173. Academic Press, Toronto, 1984.

[DEL11] Walter Didimo, Peter Eades, and Giuseppe Liotta: Drawing graphs with right angle crossings. *Theor. Comput. Sci.*, 412(39):5156–5166, 2011. https://doi.org/10.1016/j.tcs.2011.05.025.

[dFPP90] Hubert de Fraysseix, János Pach, and Richard Pollack: How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990. https://doi.org/10.1007/BF02122694.

[Did13] Walter Didimo: Density of straight-line 1-planar graph drawings. *Inf. Process. Lett.*, 113(7):236–240, 2013. https://doi.org/10.1016/j.ipl.2013.01.013.

[EL13] Peter Eades and Giuseppe Liotta: Right angle crossing graphs and 1-planarity. *Discrete Appl. Math.*, 161(7-8):961–969, 2013. https://doi.org/10.1016/j.dam.2012.11.019.

[GB07] Alexander Grigoriev and Hans L. Bodlaender: Algorithms for graphs embeddable with few crossings per edge. *Algorithmica*, 49(1):1–11, 2007. https://doi.org/10.1007/s00453-007-0010-x.

[HEH14] Weidong Huang, Peter Eades, and Seok-Hee Hong: Larger crossing angles make graphs easier to read. *J. Vis. Lang. Comput.*, 25(4):452–465, 2014. https://doi.org/10.1016/j.jvlc.2014.03.001.

[HELP12] Seok-Hee Hong, Peter Eades, Giuseppe Liotta, and Sheung-Hung Poon: Fáry's theorem for 1-planar graphs. In Joachim Gudmundsson, Julián Mestre, and Taso Viglas (editors): *Computing and Combinatorics - 18th Annual International Conference, COCOON 2012*, volume 7434 of *Lecture Notes in Computer Science*, pages 335–346. Springer, 2012. https://doi.org/10.1007/978-3-642-32241-9_29.

[HHE08] Weidong Huang, Seok-Hee Hong, and Peter Eades: Effects of crossing angles. In *IEEE VGTC Pacific Visualization Symposium 2008, PacificVis 2008*, pages 41–46. IEEE Computer Society, 2008. https://doi.org/10.1109/PACIFICVIS.2008.4475457.

[HKKP16] Seok-Hee Hong, Michael Kaufmann, Stephen G. Kobourov, and János Pach: Beyond-planar graphs: Algorithmics and combinatorics (Dagstuhl Seminar 16452). *Dagstuhl Reports*, 6(11):35–62, 2016. https://doi.org/10.4230/DagRep.6.11.35.

[HS98] David Harel and Meir Sardas: An algorithm for straight-line drawing of planar graphs. *Algorithmica*, 20(2):119–135, 1998. https://doi.org/10.1007/PL00009189.

[HT73] John E. Hopcroft and Robert Endre Tarjan: Efficient algorithms for graph manipulation (algorithm 447). *Commun. ACM*, 16(6):372–378, 1973. http://doi.acm.org/10.1145/362248.362272.

[Hua07] Weidong Huang: Using eye tracking to investigate graph layout effects. In Seok-Hee Hong and Kwan-Liu Ma (editors): *APVIS 2007, 6th International Asia-Pacific Symposium on Visualization 2007*, pages 97–100. IEEE Computer Society, 2007. https://doi.org/10.1109/APVIS.2007.329282.

[HvKKR14] Michael Hoffmann, Marc J. van Kreveld, Vincent Kusters, and Günter Rote: Quality ratios of measures for graph drawing styles. In *Proceedings of the 26th Canadian Conference on Computational Geometry, CCCG 2014*. Carleton University, Ottawa, Canada, 2014. http://www.cccg.ca/proceedings/2014/papers/paper05.pdf.

[KLM17] Stephen G. Kobourov, Giuseppe Liotta, and Fabrizio Montecchiani: An annotated bibliography on 1-planarity. *Computer Science Review*, 25:49–67, 2017. https://doi.org/10.1016/j.cosrev.2017.06.002.

[KM13] Vladimir P. Korzhik and Bojan Mohar: Minimal obstructions for 1-immersions and hardness of 1-planarity testing. *Journal of Graph Theory*, 72(1):30–71, 2013. https://doi.org/10.1002/jgt.21630.

[KS10] Daniel Král' and Ladislav Stacho: Coloring plane graphs with independent crossings. *Journal of Graph Theory*, 64(3):184–205, 2010. https://doi.org/10.1002/jgt.20448.

[Lio14] Giuseppe Liotta: Graph drawing beyond planarity: some results and open problems. In Stefano Bistarelli and Andrea Formisano (editors): *Proceedings of the 15th Italian Conference on Theoretical Computer Science 2014*, volume 1231 of *CEUR Workshop Proceedings*, pages 3–8. CEUR-WS.org, 2014. http://ceur-ws.org/Vol-1231/invited2.pdf.

[LM16] Giuseppe Liotta and Fabrizio Montecchiani: L-visibility drawings of IC-planar graphs. *Inf. Process. Lett.*, 116(3):217–222, 2016. https://doi.org/10.1016/j.ipl.2015.11.011.

[PCA02] Helen C. Purchase, David A. Carrington, and Jo-Anne Allder: Empirical evaluation of aesthetics-based graph layout. *Empirical Software Engineering*, 7(3):233–255, 2002.

[PT97] János Pach and Géza Tóth: Graphs drawn with few crossings per edge. *Combinatorica*, 17(3):427–439, 1997. https://doi.org/10.1007/BF01215922.

[Pur97] Helen C. Purchase: Which aesthetic has the greatest effect on human understanding? In Giuseppe Di Battista (editor): *Graph Drawing, 5th International Symposium, GD 1997*, volume 1353 of *Lecture Notes in Computer Science*, pages 248–261. Springer, 1997. `https://doi.org/10.1007/3-540-63938-1_67`.

[Pur00] Helen C. Purchase: Effective information visualisation: a study of graph drawing aesthetics and algorithms. *Interacting with Computers*, 13(2):147–162, 2000. `https://doi.org/10.1016/S0953-5438(00)00032-1`.

[Rin65] Gerhard Ringel: Ein Sechsfarbenproblem auf der Kugel. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 29(1-2):107–117, 1965.

[Sch90] Walter Schnyder: Embedding planar graphs on the grid. In David S. Johnson (editor): *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms 1990*, pages 138–148. SIAM, 1990. `http://dl.acm.org/citation.cfm?id=320176.320191`.

[SF07] Harald Scheid and Andreas Frommer: *Zahlentheorie (4. Aufl.)*. Elsevier Spektrum Akadem. Verl., 2007, ISBN 978-3-8274-1692-6.

[Tho88] Carsten Thomassen: Rectilinear drawings of graphs. *Journal of Graph Theory*, 12(3):335–341, 1988. `https://doi.org/10.1002/jgt.3190120306`.

[WPCM02] Colin Ware, Helen C. Purchase, Linda Colpoys, and Matthew McGill: Cognitive measurements of graph aesthetics. *Information Visualization*, 1(2):103–110, 2002. `https://doi.org/10.1057/palgrave.ivs.9500013`.

[Zha13] Xin Zhang: Drawing complete multipartite graphs on the plane with restrictions on crossings. arXiv, 2013. `http://arxiv.org/abs/1311.1994`.

[ZL13] Xin Zhang and Guizhen Liu: The structure of plane graphs with independent crossings and its applications to coloring problems. *Central European Journal of Mathematics*, 11:308–321, February 2013.

A. Appendix

In the appendix, we give an example of a NIC-plane graph drawn with the algorithm presented in Chapter 3 as NIC-planar 1-bend RAC drawing. It is the output of an implementation of this algorithm in Java. There are three pictures of the same graph turned by 90 degrees. The given embedded graph has four crossings. Fig. A.1 shows the graph after it was drawn by the algorithm by Harel and Sardas [HS98]. The edges of the divided quadrangles in Q (and related edges like the edges incident to a shift vertex) are highlighted. Observe that there are some vertices drawn as squares. These are dummy vertices that occurred by splitting an edge and one is a shift vertex. Fig. A.2 shows the graph after the crossing edges were reinserted. Bend points are drawn as non-solid square boxes. Note that one divided quadrangle fulfilled the conditions for Case 1, one for Case 2, and two for Case 3. Fig. A.3 shows the final graph drawing after the dummy edges that were inserted for obtaining empty kites were removed and the remaining dummy vertices were transformed to bend points. Note that the top left corner point is a bend point that emerged by splitting an original edge when empty kites were built.

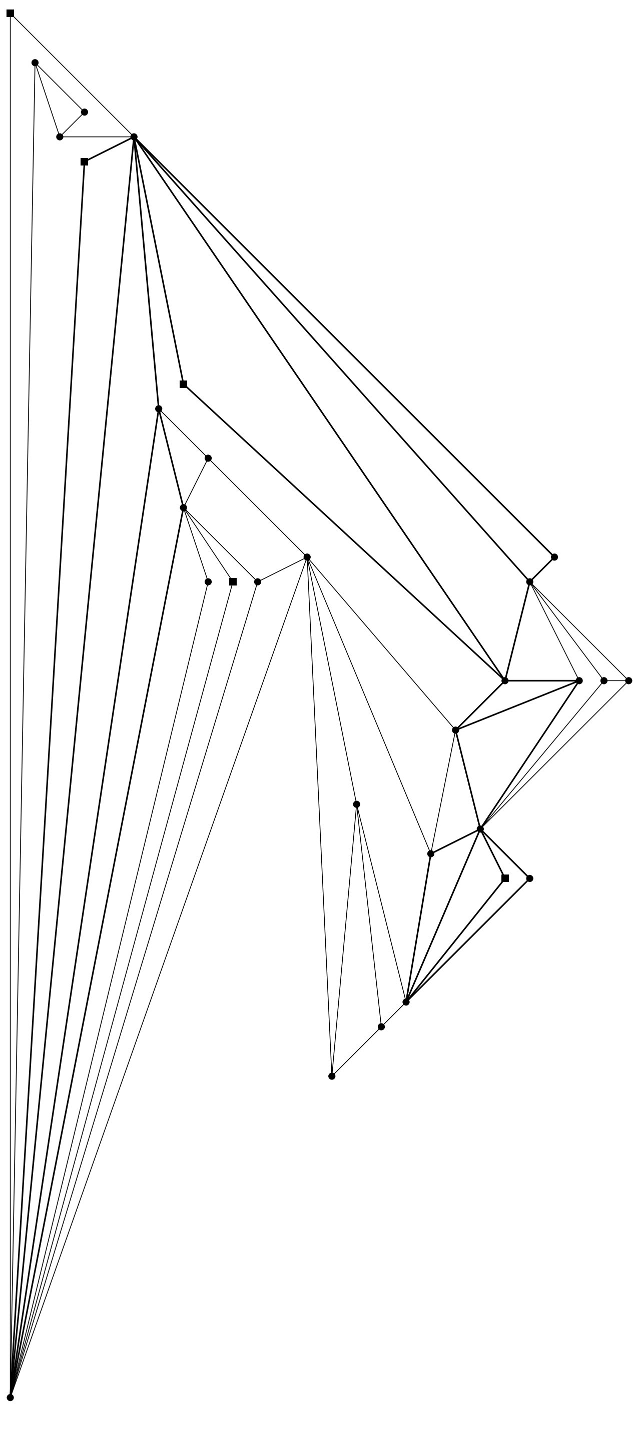


Fig. A.1.: Graph before the reinsertion of crossing edges.

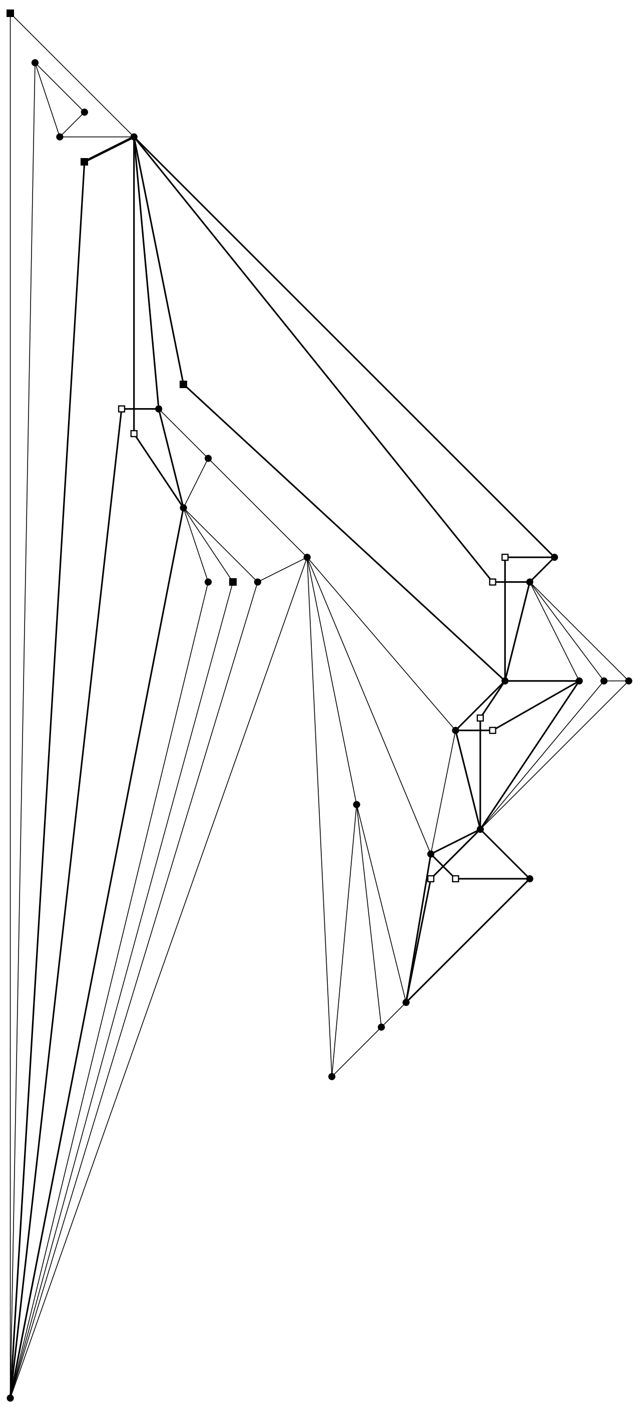


Fig. A.2.: Graph after the reinsertion of crossing edges.

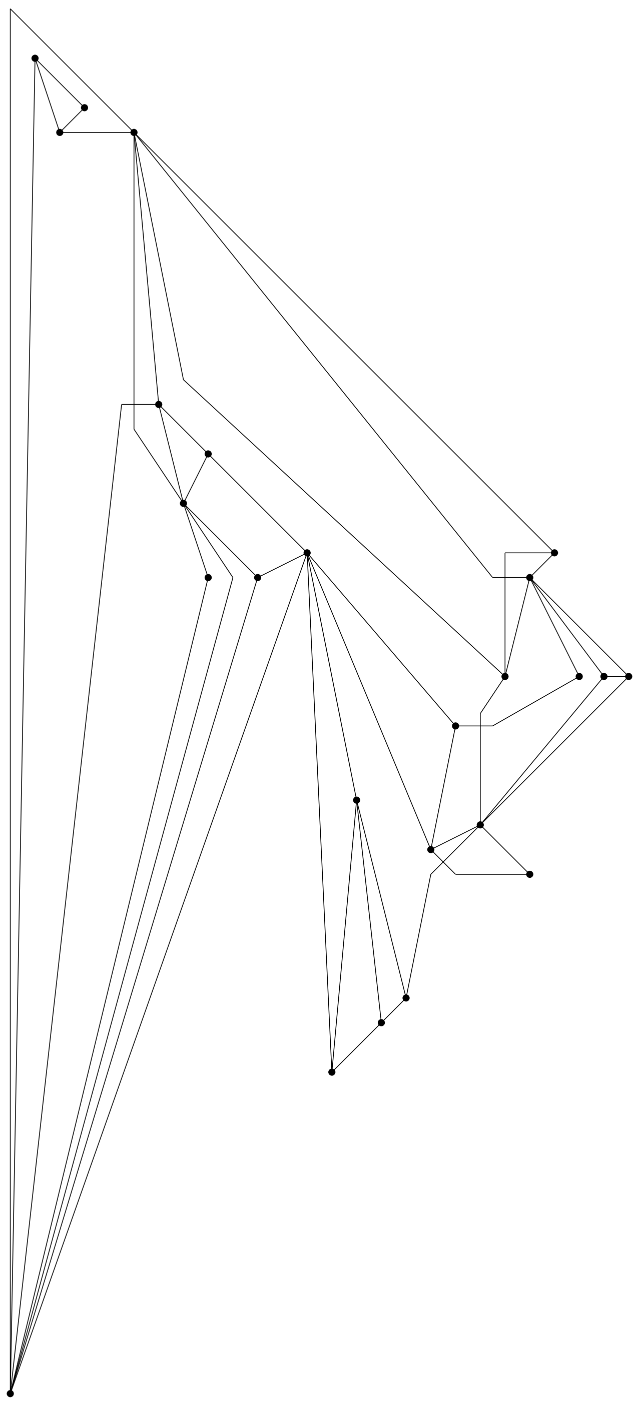


Fig. A.3.: Final NIC-planar 1-bend RAC drawing after the removal of dummy edges and vertices.

Erklärung

Hiermit versichere ich die vorliegende Abschlussarbeit selbstständig verfasst zu haben, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben, und die Arbeit bisher oder gleichzeitig keiner anderen Prüfungsbehörde unter Erlangung eines akademischen Grades vorgelegt zu haben.

Würzburg, den 1. Dezember 2017

. . . .
Johannes Zink