Computing Crossing Numbers

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik
von

Markus Chimani

Dortmund
2008

Tag der mündlichen Prüfung:
24.11.2008

Dekan:
Prof. Dr. Peter Buchholz

Gutachter:
Prof. Dr. Petra Mutzel, Technische Universität Dortmund
Prof. Dr. Martin Skutella, Technische Universität Berlin

Abstract

The graph theoretic problem of crossing numbers has been around for over 60 years, but still very little is known about this simple, yet intricate non-planarity measure. The question is easy to state: Given a graph, draw it in the plane with the minimum number of edge crossings. A lot of research has been devoted to giving an answer to this question, not only by graph theoreticians, but also by computer scientists. The crossing number is central to areas like chip design and automatic graph drawing. While there are algorithms to solve the problem heuristically, we know that it is in general NP-complete. Furthermore, we do not know if the problem is efficiently approximable, except for some special cases.

In this thesis, we tackle the problem using Mathematical Programming. We show how to formulate the crossing number problem as systems of linear inequalities, and discuss how to solve these formulations for reasonably sized graphs to provable optimality in acceptable time—despite its theoretical complexity class. We present non-standard branch-and-cut-and-price techniques to achieve this goal, and introduce an efficient preprocessing algorithm, also valid for other traditional non-planarity measures. We discuss extensions of these ideas to related crossing number variants arising in practice, and show a practical application of a formerly purely theoretic crossing number derivative.

The thesis also contains an extensive experimental study of the formulations and algorithms presented herein, and an outlook on its applicability for graph theoretic questions regarding the crossing numbers of special graph classes.

Zusammenfassung

Das Kreuzungszahlproblem wird von Graphentheoretikern seit über 60 Jahren betrachtet, jedoch ist noch immer sehr wenig über dieses einfache und zugleich hochkomplizierte Maß der Nichtplanarität bekannt. Die Aufgabenstellung ist simpel: Gegeben ein Graph, zeichnen Sie ihn mit der kleinstmöglichen Anzahl an Kantenkreuzungen. Nicht nur Graphentheoretiker sondern auch Informatiker beschäftigten sich ausgiebig mit dieser Aufgabe, denn es handelt sich dabei um ein zentrales Konzept im Chipdesign und im automatischen Graphenzeichnen. Zwar existieren Algorithmen um das Problem heuristisch zu lösen, jedoch wissen wir, dass es im Allgemeinen NP-vollständig ist. Darüberhinaus ist auch unbekannt, ob sich das Problem, außer in Spezialfällen, effizient approximieren lässt.

In dieser Dissertation, versuchen wir das Problem mit Hilfe der Mathematischen Programmierung zu lösen. Wir zeigen, wie man das Kreuzungszahlproblem als verschiedene Systeme von linearen Ungleichungen formulieren kann und diskutieren wie man diese Formulierungen für nicht allzu große Graphen beweisbar optimal und in akzeptabler Zeit lösen kann—unabhängig von seiner formalen Komplexitätsklasse. Wir stellen dazu benötigte maßgeschneiderte Branch-and-Cut-and-Price Techniken vor, und präsentieren einen effizienten Algorithmus zur Vorverarbeitung; dieser ist auch für andere traditionelle Maße der Nichtplanarität geeignet. Wir diskutieren Erweiterungen unserer Ideen für verwandte Kreuzungszahlkonzepte die in der Praxis auftreten, und zeigen eine praktische Anwendung eines vormals rein theoretisch behandelten Kreuzungszahl-Derivats auf.

Diese Arbeit enthält auch eine ausführliche experimentelle Studie der präsentierten Formulierungen und Algorithmen, sowie einen Ausblick über deren mögliche Nutzung für graphentheoretische Fragen bezüglich der Kreuzungszahlen von speziellen Graphenklassen.

Acknowledgements

I want to thank my advisor Prof. Petra Mutzel for giving me the opportunity to do research in a wonderful scientific environment; for introducing me to the topic of crossing minimization, which became one of my dearest research passions; for offering the possibilities and freedom to pursue multiple research areas, which may have been a distraction from the topic of this very thesis, but much more was it a fruitful and worthwhile experience in the long run; for funding conference trips and introducing me to established researchers; and most of all, for knowing when to offer freedom and when to help with comments and ideas.

I want to thank the Chair XI for Algorithm Engineering at TU Dortmund for relaxing coffee breaks that made long hours worthwhile and refreshing; I want to thank especially our secretary Gundel Jankord for taking a lot of organizational stuff from our shoulders. I also want to thank in particular Petra's research group I had the pleasure to be a part of, for all the interesting scientific discussions and collaborations. Thank you, Carsten Gutwenger, Maria Kandyba, Karsten Klein, Wolfgang Paul, and Hoi-Ming Wong. I also want to thank the co-authors of my papers for sharing their experience and knowledge, in particular Michael Schulz for sharing my interest in weird crossing number concepts and for having a great time. And thank you for proof-reading parts of this thesis, Karsten, Wolfgang, and Michael.

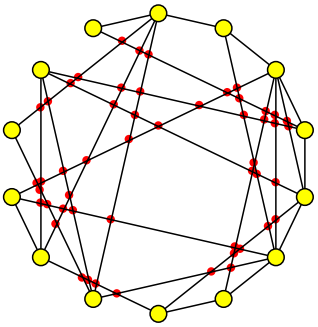
I want to thank Prof. Ernst-Erich Doberkat, Petra, Prof. Martin Skutella, and Henrik Björklund, for their readily willingness to serve in my defense commission, commenting on my thesis, and making it surprisingly easy to find a date when everybody was able to attend. In particular, I want to thank Martin, who served as my second examiner, for carefully working through my thesis and for his gracious comments regarding it; and for taking it upon himself to attend my defense even though this required traveling long hours between Berlin and Dortmund for this only reason.

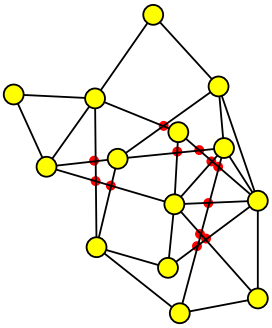
I want to thank my family, in particular my parents, for supporting me in all concievable ways from when I was young, up to now. And last but not least, I want to thank Maria, for putting up with me during the last weeks when I was notoriously stressed. I love you.

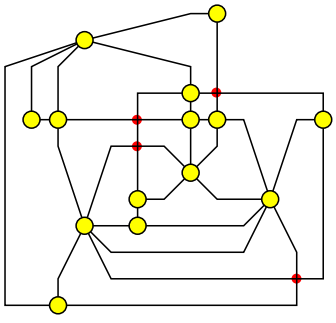
Markus Chimani
Dortmund, 2008

Motivational Example

Three drawings of the same graph with 51, 12, and 4 crossings, respectively. Most aesthetic criteria like few edge bends, uniform edge lengths, or a small drawing area would favor the first two drawings, while the last drawing is preferable with respect to the number of edge crossings.







Notation

For quick reference, we list common notations, abbreviations, and naming schemata used throughout this thesis.

Common Variables	
s, t, u, v	nodes
e, f, g, h	edges/arcs
G, H	graphs
U, V, W	set of nodes
A, E, F	set of edges/arcs
$w(\cdot)$	(integral) edge weights
Γ	embedding of a graph
$\mathcal{D}$	drawing of a graph
φ	face in an embedding/drawing of a graph
ψ	hyperedge/hyperarc
Ψ	set of hyperedges/hyperarcs
$\mathcal{H} = (V, \Psi)$	hypergraph
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	simultaneous graph
$\mathcal{S}, \mathcal{T}$	SPR-tree of a graph
ν, μ, η	nodes of an SPR-tree
$\mathcal{CP}, \mathcal{R}$	set of edge pairs
x, y, z, χ	variable vectors in an ILP
$\bar{x}, \bar{y}, \bar{z}, \bar{\chi}$	assignments to variable vectors in an ILP
$\tilde{x}, \tilde{y}, \tilde{z}, \tilde{\chi}$	0/1-assignments to variable vectors in an ILP

Notation, Operators, Generators, etc.	
(u, v)	directed arc from u to v
$\{u, v\}$	undirected edge between u and v
$(u \to v)$	path from u to v
$V(G), E(G), A(G)$	the nodes, edges, and arcs of the graph G, respectively
$\overline{e}$	the undirected edge $\{u, v\}$ for an arc $e = (u, v)$
$\overrightarrow{e}$, $\overrightarrow{e}^G$	for $e = \{u, v\}$, the directed arc (u, v) or (v, u) in G, if unique
$G[W], G[F]$	subgraph of G induced by the node set W, or by the edge/arc set F, respectively
$G+v, G-v, G+e, G-e$	The graph G with/without the node v or edge/arc e, respectively
$S^{\{k\}}$	all unordered k-tuples of pairwise disjoint elements of S
$S^{\langle k \rangle}$	all ordered k-tuples of pairwise disjoint elements of S
$\{e; f_1, f_2, \ldots\}$	$\{\{e, f_1\}, \{e, f_2\}, \ldots\}$
$G_\nu^{\mathcal{T}}$	the skeleton graph corresponding to the node ν in the SPR-tree $\mathcal{T}$
$G^{[\ell]}$	the graph obtained from G by replacing each edge by a chain of length ℓ
$G[\bar{z}]$, $G[\bar{x}, \bar{y}]$	partial planarization of G realizing the crossings specified by $\bar{z}$ or $(\bar{x}, \bar{y})$
$H \preceq G$, $H \preceq_W G$	H is a minor of G (only merging nodes of W)
$\text{mincut}_{st}(G)$	size of a minimium st-cut in G
$X(e)$	$\sum_{f:\{e,f\}\in\mathcal{CP}} x_{e,f}$
$X(v)$	$\sum_{\{e,f\}\in\mathcal{CP}:v\in e} x_{e,f}$

Considered Non-planarity Measures	
$\mathrm{cr}(G)$	**(traditional) crossing number.** minimum number of edge crossings required to obtain a planar drawing.
$\mathrm{scr}(G)$	**simple crossing number.** at most one crossing per edge.
$\mathrm{pcr}(G)$	**pairwise crossing number.** we count only the pairs of crossing edges, independent on how often these pairs cross each other.
$\mathrm{ocr}(G)$	**odd crossing number.** we count only pairs of edges which cross each other an odd number of times.
$\mathrm{mcr}(G)$	**minor(-monotone) crossing number.** the smallest $\mathrm{cr}(H)$ among all graphs H with $G \preceq H$.
$\mathrm{mcr}_W(G)$	**W-restricted minor(-monotone) crossing number.** the smallest $\mathrm{cr}(H)$ among all graphs H with $G \preceq_W H$.
$\mathrm{phcr}(G)$	**point-based hypergraph crossing number.** hyperedges are represented by stars.
$\mathrm{thcr}(G)$	**tree-based hypergraph crossing number.** hyperedges are represented by trees.
$\mathrm{ccr}(C)$	**circuit crossing number.** crossing number of an electrical circuit C.
$\mathrm{simcr}(G)$	**simultaneous crossing number.** crossing number of a simultaneous graph.
$\mathrm{bcr}(G)$	**bimodal crossing number.** crossing number when the in- and out-going arcs of each node have to be drawn consecutively.
$\mathrm{skew}(G)$	**skewness.** minimum number of edges to remove, in order to obtain a planar subgraph.
$\mathrm{thick}(G)$	**thickness.** minimum number of planar subgraphs that, when merged, give G.
$\mathrm{coarse}(G)$	**coarseness.** maximum number of edge-disjoint non-planar subgraphs.
$\mathrm{genus}(G)$	**genus.** minimum genus g such that the graph can be embedded on an oriented surface of genus g (number of "handles on a sphere") without crossings.

Contents

Part I

Introduction

Chapter 1

Introduction

crossing: *(Graph Theory)*
The point in the plane where the images of two edges of a drawn graph intersect.

Assume we are given a *relationship diagram*: It shows various people; people that know each other are connected by lines. We want to draw such a diagram nicely on a piece of paper, cf. Figure 1.1. In our drawing, we would like to have as few line crossings as possible, since such crossings make the diagram hard to read. How many crossings are inevitable in our drawing?

In terms of graph theory, the diagram is called *graph*, the people in the diagram are *nodes*, the lines between people are *edges*, and we want to draw this graph in the plane. The crossing number of a graph is a very intuitive notion for measuring the non-planarity of a graph. Informally, the crossing number problem is quite easy to formulate: *Given a graph, draw it in the plane with a minimum number of edge crossings.*

The origin of the problem was described by Turán in his "Note of Welcome" in the first issue of the Journal of Graph Theory [139]: While working in a labor camp during World War II, he noted that crossings of the rails

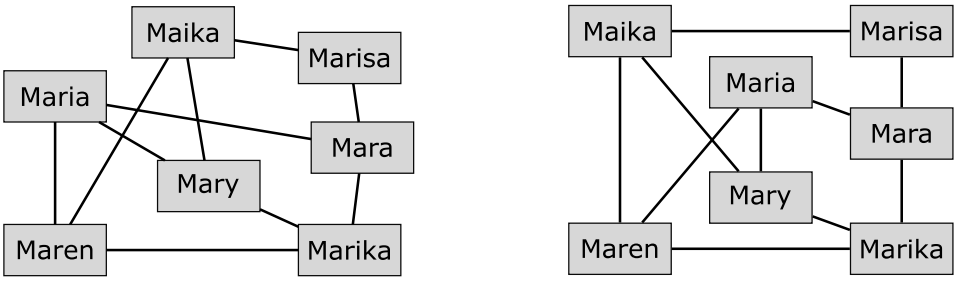


Figure 1.1: Two different drawings of a relationship diagram. While the left one has 3 edge crossings, the one on the right realizes the minimum crossing number of 1.

between kilns and storage yards caused the trucks to jump the rails. Minimizing these crossings corresponds to the crossing minimization problem for a complete bipartite graph $K_{n,m}$.

Since then, the problem has received lively attention, see [137] for an overview and [140] for an extensive bibliography: Graph theoreticians enjoy the problem for its intriguing clearness and analytical hardness, and the problem lies at the heart of areas like VLSI design and automatic graph drawing. Purchase [126] even showed in a study that it is one of the most important criteria for the comprehensibility of drawn graphs. Nonetheless, although a lot of research has been conducted, most fundamental questions remain unanswered. For example, the exact crossing number of complete bipartite graphs, for which Turán originally asked, is still unknown for general graph sizes.

From the algorithmic point of view, the problem is known to be NP-complete, which means it is extremely unlikely that there exists a polynomial algorithm to find the crossing number of a given graph. The problem is even hard in the sense that concepts like its approximability are still unknown and until know, the practically best way of solving the problem was by heuristics.

1.1 Overview on this thesis

In this thesis we present how to solve the crossing number problem to *provable optimality*, using integer linear programming techniques. Formally, the thesis is structured into four parts: Part I is a general introduction to the thesis' topic and summarizes the necessary preliminaries and notations used throughout this thesis; the final Part IV contains the author's CV, references, and an index.

The main content is contained in Part II. In Chapter 4, we discuss novel strong preprocessing routines also valid for other non-planarity measures than only the crossing number. We will then introduce two different ILP approaches to solve the problem for real-world graphs in Chapter 5, and discuss some of their polyhedral properties. The following Chapter 6 describes how we can solve these models to provable optimality. We will also discuss alternative variants of the crossing number and solution strategies for them, see Chapter 7.

Part III reports on experiments and gives an outlook on ongoing research. Nearly all described approaches have been implemented, and we are therefore able to see their effectiveness in practice; this is summarized in Chapter 8 and 9 for the traditional crossing number and for variants of it, respectively. In Chapter 10, we give an extended outlook regarding the crossing number computation for special graph classes that are interesting from the graph theoretical point of view. We conclude with a summary and final outlook in Chapter 11.

The content of this thesis is largely based on the following original papers:

- Non-Planar Core Reduction [29] (Section 4.2, 4.3);
- Subdivision-based Crossing Minimization ILP [23, 30] (Section 5.2, 6.1–6.3);
- Ordering-based Crossing Minimization ILP [37] (Section 5.3, 6.1, 6.4);
- Minor and Hypergraph Crossing Number [28] (Section 7.2); and
- Simultaneous Crossing Number [35] (Section 7.3).

Furthermore, this thesis contains yet unpublished results regarding:

- the simple definition of SPR-trees and its compatibility with SPQR-trees (Section 4.2.1, 4.2.2);
- the non-planar core reduction in the context of the graph genus (Section 4.3.4);
- the class of triangle inequalities for OECM (Section 5.3.1);
- the facet-defining properties of Kuratowski constraints (Section 5.4);
- the application of hypergraph crossing minimization for electrical circuit design (Section 7.2.7); and
- the consideration of special graph classes (Chapter 10).

To give a more focused thesis, we only briefly discuss the efficient extraction method of multiple Kuratowski subdivisions [39], and refer to the full paper for details. Furthermore, we will not discuss the author's related crossing number research regarding the minimum cut properties of planarizations [31], upward crossing minimization [33], the polynomial solution for vertex insertion [32], nor its applicability for approximating the crossing number of apex graphs [34].

Chapter 2

Preliminaries: Graph Theory

crossing: *(Handicraft)*
A painting technique whereby freshly applied paint is rebrushed at right angles to the direction of application and then rebrushed at right angles again to provide even distribution of paint over the surface.

As this thesis is focused on the crossing number of graphs, this section outlines the necessary prerequisits and serves to define the exact notation we will use throughout this work. The basic definitions are based on Diestel [47] and Harary [84].

2.1 Graphs

We use the following generator function for notational simplicity. Let S be any set. Then

$$S^{\{k\}} := \{S' \subseteq S \mid |S'| = k\}$$

is the set of all unordered k-tuples with pairwise disjoint elements of S, and

$$S^{\langle k \rangle} := \{(a_1, \ldots, a_k) \mid \forall 1 \leq i < j \leq k : a_i, a_j \in S \wedge a_i \neq a_j\}$$

is the set of all ordered k-tuples with pairwise disjoint elements of S.

A *graph* G is a 2-tuple of a set of *vertices*—or *nodes*—V and a collection of structures connecting these vertices. We can define a variety of graph types based on the exact definition of these connectors.

Simple undirected graph, $G = (V, E)$**.** The *edges* $E \subseteq V^{\{2\}}$ are unordered 2-tuples of vertices. Let $e = \{u, v\} \in E$; we say e is *incident* to u and v, and two edges or nodes are *adjacent* if they share a common node

or edge, respectively. All nodes $N(v)$ adjacent to some node $v \in V$ are called *neighbors* of v. We define the *degree* $\deg(v)$ as the number of edges incident to v.

Simple directed graph, $G = (V, A)$. The *arcs* $A \subseteq V^{\langle 2 \rangle}$ are ordered 2-tuples of vertices. Let $e = (u, v) \in E$, we say e is directed from its *source node* $\mathrm{src}(e) := u$ to its *target node* $\mathrm{trgt}(e) := v$. Two arcs which have both nodes in common are *reversals* of each other. We define the *in-degree* $\deg^-(v)$ and the *out-degree* $\deg^+(v)$ as the number of arcs having v as their target or source node, respectively.

Multi-graph with self-loops. Sometimes it can be interesting to consider graphs where E or A are not simple sets but *multi-sets*, i.e., we allow edges or arcs to appear multiple times. Also, we may allow *self-loops*, i.e., edges or arcs where the two incident nodes are identical. In the following we will almost exclusively deal with *simple* graphs, i.e., graphs without multi-edges and self-loops. In the case of crossing numbers we can efficiently deal with non-simple graphs by a straight-forward transformation, described later in this section.

Hypergraph $\mathcal{H} = (V, \Psi)$. A generalization of traditional undirected graphs are undirected hypergraphs, where traditional edges are replaced by *hyperedges*: we have $\Psi \subseteq \bigcup_{i=2}^{|V|} V^{\{i\}}$, i.e., a hyperedge is a subset of V of any size larger than 1. Analogously, we can define directed hypergraphs by using *hyperarcs*: we have $\Psi \subseteq \{(\psi_{\mathrm{src}}, \psi_{\mathrm{trgt}}) \mid \emptyset \subset \psi_{\mathrm{src}}, \psi_{\mathrm{trgt}} \subset V \wedge \psi_{\mathrm{src}} \cap \psi_{\mathrm{trgt}} = \emptyset\}$, i.e., each hyperarc consists of a 2-tuple of non-empty disjoint vertex subsets specifying its source and target nodes.

If we are given a graph G without explicitly specifying its vertices and connectors, we can use $V(G)$, $E(G)$, and $A(G)$ to obtain its set of vertices, edges or arcs, respectively.

Orientation. We say a directed graph $H = (V, A)$ is an *orientation* of an undirected graph $G = (V, E)$, if for each edge $\{u, v\} \in E$, A contains exactly one of its directed versions, i.e., $|\{(u, v), (v, u)\} \cap A| = 1$. We call G the *shadow* of H. We therefore define the operations $\overline{(u, v)} := \{u, v\}$ to obtain an undirected edge from an arc, and the inverse $\overrightarrow{\{u, v\}}^H$ to obtain the unique directed arc in H which is incident to u and v; we may omit the graph specification if it is clear from the context.

Path and Cycle. A *path* $(u \to v)$ of length k in a simple graph is a connected sequence $p = \langle e_1, e_2, \ldots, e_k \rangle$ of disjoint sequentially connected edges or arcs in that graph: In the directed case, we have $\mathrm{src}(e_1) = u$, $\mathrm{trgt}(e_k) = v$, and $\mathrm{trgt}(e_i) = \mathrm{src}(e_{i+1})$ for all $1 \leq i < k$. For an undirected graph G, we can find an orientation H such that $\langle \overrightarrow{e_1}^H, \overrightarrow{e_2}^H, \ldots, \overrightarrow{e_k}^H \rangle$ is a path in H. If $u = v$,

and therefore $k \geq 3$ in undirected graphs, we call the path a *cycle*. If all the nodes on a path are pairwise disjoint, we call it a *simple path*. If all nodes of a cycle are pairwise disjoint, except for the identical start and end node, we call it a *simple cycle*. A simple path where all *inner nodes*, i.e., the nodes incident to two edges or arcs of the path, have degree 2 is called a *chain*.

Let $G = (V, E)$ be an undirected graph in the following. Although the definitions given below are for undirected graphs, they can be applied analogously to directed graphs.

Subgraph. A graph $G' = (V', E')$ is a *subgraph* of G, denoted by $G' \subseteq G$, if $V' \subseteq V$ and $E' \subseteq E$. The fact that the edges of E' only connect vertices of V' follows from G' being a valid graph. We define an *edge-induced* subgraph $G[E'] := (V[E'], E')$ with $V[E'] := \bigcup_{e \in E'} e$, i.e., the subgraph that contains exactly the edges E' and all incident vertices. Inversely, we can define a *node-induced* subgraph $G[V'] := (V', E[V'])$ with $E[V'] := \{e \in E \mid e \subseteq V'\}$, i.e., the subgraph that contains exactly the nodes V' and all edges of G that connect nodes of V'.

An alternative specification of subgraphs is via exclusion of certain nodes or edges. We write $G - v$ and $G - e$ to denote the subgraphs obtained by removing the node v or the edge e, respectively. Inversely, we define $G + v$ and $G + e$ as the graphs obtained by introducing v or e, respectively.

A third alternative to specify a subgraph is by defining it as an antipode to some other subgraph. If $G' \subseteq G$, we say $\bar{G}' := G[E \setminus E(G')]$ is the *complement* of G' with respect to G, i.e., $\bar{G}'$ contains exactly the edges in G but not in G'.

Connectedness and Trees. The graph G is *connected* if there exists a path between every pair of nodes. If it is not connected, it consists of several *connected components*, i.e., the maximal subgraphs that are connected. A connected graph without a cycle is called *tree*, a non-connected graph without a cycle is called *forest*, as its connected components form trees.

Generally, G is k-connected ($k < |V|$) if the removal of less than k nodes cannot disconnect the graph. I.e., $G[V \setminus W]$ is connected for all node sets $W \subset V$ with $|W| < k$. A 2-connected graph is also known as a *biconnected* graph. If a graph is connected but not biconnected, it consists of several *biconnected components* (*blocks*), i.e., the maximal subgraphs that are biconnected.

Blocks are attached to each other through *cut vertices*, i.e., vertices that are part of two or more blocks. The removal of a cut vertex would make the remaining graph disconnected. Given a connected graph G, we can define a special Block-Cut-structure $\mathcal{B}$ of G as the graph that contains a node of type B for each block of G, and a node of type C for each cut vertex in G. There is an edge between a B-node and a C-node if and only if the respective block contains the respective cut vertex. We call this structure *BC-tree*, as it is easy to see that the structure forms a tree: Assume there would be a cycle,

then the deletion of a single vertex could not disconnect any two cut vertices or blocks in that cycle, and hence all these elements would be part of a single larger biconnected component.

Analogous to cut vertices, we can define *separation pairs* in biconnected components B as the set of two vertices whose removal makes B disconnected. Graphs without a separation pair are by definition 3-connected, also known as *triconnected*.

Subdivision and Minor. We say that H is a *minor* of G, denoted by $H \preceq G$, if H can be obtained from G by a series of vertex deletions (including all its incident edges) and *edge contractions*: An edge $e = \{u, v\}$ is contracted by merging its two incident nodes into one that becomes adjacent to all former neighbors of u and v. We obtain a *subdivision* G' of G by replacing all edges of G by chains of length at least 1. Let G'' be any graph with $G' \subseteq G''$. G is said to be a *topological minor* of G''.

Complete Graphs. Special graphs of further interest are the *complete graphs* $K_n := (V, V^{\{2\}})$ with $|V| = n$, i.e., simple graphs with n vertices that contain all possible edges. A graph is *bipartite* if we can partition its node set into two disjoint sets V_1 and V_2 such that each edge connects nodes of different partition sets. We define *complete bipartite graphs* $K_{n,m} = (V_1 \cup V_2, \{\{u, v\} \mid u \in V_1, v \in V_2\})$ with $|V_1| = n$ and $|V_2| = m$, i.e., bipartite graphs with n and m nodes per partition set, respectively, that contain all possible edges.

2.2 Planarity

A *drawing* $\mathcal{D}$ of a graph G on the plane is a one-to-one mapping of each vertex to a point in $\mathbb{R}^2$, and each edge to a simple open curve between its two endpoints. A curve is not allowed to contain other vertices than its two endpoints. A *crossing* is a common point of two curves, other than their endpoints. We only consider drawings where there are no common points of more than two curves, other than their endpoints.

Embedding. Assume the given graph $G = (V, E)$ can be drawn without any crossings. We say the graph is *planar* and each drawing of G which requires no crossings is a *planar drawing*. We can define equivalence classes of such planar drawings called *combinatorial embeddings*, also known as *planar rotation systems*. A combinatorial embedding Γ of G gives, for each vertex $v \in V$, the cyclic orderings of the edges $E(v) := \{e \in E | v \in e\}$ around v. Any drawing realizing Γ on a sphere defines *faces*, i.e., regions bounded by edges. Formally, a face is represented as the cyclic sequence of its bounding edges, whereby an edge is listed twice if the region touches both sides of the edge. The special property is that there is a one-to-one mapping between

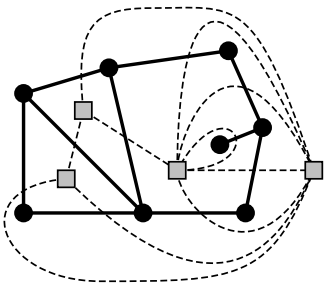


Figure 2.1: An embedded graph G (circular vertices and solid edges) and its dual G^D (square nodes and dashed edges).

a combinatorial embedding Γ and the set of faces arising from any drawing realizing Γ.

Realizing a combinatorial embedding in the plane can be seen as choosing any face on the sphere—subsequently called *outer face*—, cutting this face out of the sphere, and flattening the former sphere which is now topologically equivalent to a disc. Then, each edge-bounded region is called an *inner face*, and the outer face is the only unbounded region in the drawing. A combinatorial embedding together with the selection of some outer face is called *planar embedding*.

The definition of embeddings gives rise to the *dual graph* G^D of some graph G with respect to Γ, cf. Figure 2.1. The vertices of G^D are the faces defined by Γ. For each primal edge e we have a dual edge connecting the two (not necessarily disjoint) faces that border e. In general, G^D will be a multi-graph with self-loops. Let Γ^D be the embedding of G^D implied by Γ. Then the dual of G^D with respect to Γ^D is again G.

Note that combinatorial embeddings—planar rotation systems—can only be defined for planar graphs and always allow a planar drawing realizing them. In contrast to this, we can define arbitrary *rotation systems*, i.e., cyclic orderings of the edges around the vertices, that do not correspond to planar drawings or not even to planar graphs.

Euler's theorem. A central observation for planar simple graphs, going back to Euler, is that they cannot have arbitrarily many edges. In fact, we have $|V|-|E|+|\Phi| = 2$ for any planar embedding, where Φ denotes the set of faces. For planar embeddings of simple planar graphs with at least 3 edges, we can observe that each face is bordered by at least 3 edges, and each face lies on the border of at most 2 faces. It follows that $|E| \leq 3|V|-6$ for $|V| \geq 3$. Generally, we can say that in planar graphs $|E| = \mathcal{O}(|V|)$ and $|\Phi| = \mathcal{O}(|V|)$.

Kuratowski's theorem. A natural question is whether a given graph G is planar. In 1930, Kuratowski [104] already categorized the class of planar graphs as the graphs which do not contain a K_5 or $K_{3,3}$ subdivision as a

subgraph. In other words, a graph G is non-planar if and only if K_5 or $K_{3,3}$ is a topological minor of G. Wagner [142] later strengthened this theorem by showing that a graph G is non-planar if and only if K_5 or $K_{3,3}$ is a minor of G. In the following, K_5 and $K_{3,3}$ subdivisions will be summarized under the term *Kuratowski subdivisions*. We call the chains of such a subdivision *Kuratowski paths*, and the nodes which remain after contracting these chains into single edges *Kuratowski nodes*.

Planarity testing. Having the above classification, we now ask how we can efficiently test membership to the class of planar graphs. The first polynomial algorithm capable of testing planarity was due to Auslander and Parter [5] in 1961, and later corrected by Goldstein [69], which required $\mathcal{O}(|V|^3)$ time. In 1964, Demoucron et al. [43] gave a relatively simple $\mathcal{O}(|V|^2)$ algorithm and Lempel et al. [107] presented another $\mathcal{O}(|V|^2)$ algorithm three years later, based on vertex addition. The latter became particularly important as Booth and Luecker [16] in 1976 where able to reduce its runtime complexity to linear time using PQ-trees and a linear-time *st*-numbering algorithm by Even and Tarjan [55]. Before that, Hopcroft and Tarjan [89] in 1974 were the first to present a linear time planarity testing algorithm, based on path addition.

Both these linear time algorithms are highly complex, and it is non-trivial to obtain a planar embedding from them when planarity is detected: Linear algorithms for this subsequent problem were introduced ten to twenty years later by Chiba et al. [27] for the vertex addition algorithm and by Mehlhorn and Mutzel [111] for the path addition algorithm.

Since these first algorithms, much effort was put into the development of simpler algorithms with optimal runtime performance. The two currently easiest and fastest [18] algorithms are both based on a DFS-traversal of the graph and are due to de Fraysseix et al. [60, 61, 62] and Boyer and Myrvold [19, 20]. Both algorithms allow to easily obtain either a planar embedding if the graph is indeed planar, or they return a Kuratowski subdivision as a witness of the graph's non-planarity. We will sketch the latter of these two algorithms in Section 6.1.2 when considering an extension to it to efficiently extract multiple such subdivisions at once.

2.3 Crossing Number

After deciding that a graph is non-planar, one might ask *How non-planar is it?* The most common parameter to measure non-planarity is the *crossing number* $\mathrm{cr}(G)$: the smallest number of crossings in any drawing of G. We therefore obtain the decision problem *Given some graph G and some integer number k. Does there exist a drawing with at most k crossings?* And we have the corresponding optimization problem *Given some graph G. What is the smallest number of necessary crossings?*

These and related problems have been widely studied in the literature, see [137] for an overview and [140] for an extensive bibliography. In the following, we will restrict ourselves to simple undirected graphs. At the end of this section, we will discuss why this is a sensible restriction.

Graph Theory. The decision variant of the crossing number problem was shown to be NP-complete by Garey and Johnson [64] and remains NP-complete for cubic graphs [85], i.e., for graphs where each vertex has exactly degree 3. Not only is it a difficult problem from the perspective of complexity theory, but also from the graph theoretical point of view: Even for seemingly simple graph classes, calculating—or at least bounding—the crossing number tends to be difficult. In particular, even for the classes of complete and complete bipartite graphs—which have been the origin of crossing number research as described by Turán [139] over 60 years ago—we still do not know a definitive formula which gives their crossing numbers. For both problems there are only two long-standing conjectures on their crossing number by Zarankiewicz [147]—the proof in his paper was shown to be false in [79], which reduced his theorem to a conjecture—and by Guy [80] for the complete bipartite and the complete graphs, respectively.

The theoretic research on crossing numbers has divided into several directions, trying to find some general idea to solve the original problem or to improve the understanding of the fundamental difficulty of the concept. Hence there are many papers on bounds and certain properties of crossing-minimal drawings (e.g., [48, 125, 145]), on the crossing numbers of particular graph classes (see also Chapter 10) and their Cartesian products (e.g., [13, 106, 128, 148]), or on the crossing numbers with regard to oriented or non-oriented surfaces of higher genus (e.g., [17, 68, 82, 87, 121, 135]). In particular there is also a lot of research on "other" crossing numbers, i.e., where the concepts of crossings, allowed drawings or counting schemes are modified. Some of these different crossing numbers come from a theoretical background and are often targeted to understand the traditional crossing number better [15, 119]; others originate from practical needs in graph drawing applications [21, 25].

Algorithmics. The research on algorithms to solve the crossing number problem has long been only devoted to heuristics. The currently best known heuristic is the so-called *planarization method*, introduced in [6]: We start with a planar subgraph G' of G and reinsert the temporarily removed edges one after another[1]. Thereby we try to insert the edge e with the smallest number of edge crossings into the planar graph G'. The emerging crossings are then replaced by *dummy vertices*, and hence the new graph G'' is again

[1]The problem of inserting multiple edges simultaneously is NP-complete; experiments with ILP-based algorithms suggest that the problem is also hard in practice. See [115, 149] for details.

planar, and we can insert the next edge into G''. We call the graph G^* resulting from the series of these insertions a *planarization* of G. Using any planar drawing of G^*, we can obtain a drawing of G by substituting the dummy vertices by edge crossings. This drawing then has exactly as many crossings as there were dummy vertices in G^*. Generally, we define:

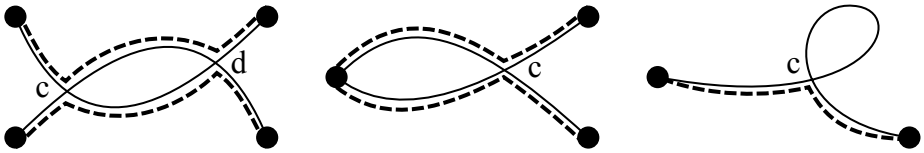
Definition 2.1 (Planarization)**.** Given a non-planar graph $G = (V, E)$ and a planar graph $P = (V \cup V_P, E_P)$ with $\deg(v) = 4$ for all $v \in V_P$. We say P is a *planarization* of G if there exists a drawing $\mathcal{D}$ of G in the plane with the following property: When replacing the crossings in $\mathcal{D}$ by dummy nodes of degree 4, we obtain a planar drawing of P.

The complexity of computing a planarization of a planar graph G plus some edge e with the minimum number of dummy nodes is still an open problem. Therefore, the planarization method considers the related *edge insertion* problem instead, i.e., the problem of inserting e into G such that a combinatorial embedding of the resulting planarization P induces a planar rotation system for G. In other words, we force all crossings to appear on the newly inserted edge e.

If this induced planar rotation system is fixed, we can easily compute P by finding a shortest path in the dual graph between faces incident to e's respective end nodes. If the induced combinatorial embedding is not fixed, we can still solve the problem in linear time, using a more involved algorithm by Gutwenger et al. [77], based on SPQR-trees. Such trees represent the triconnected structures in a graph; we will describe them in more detail in Section 4.2.

Despite many efforts, the problem's hardness w.r.t. approximability is still unknown. For the general crossing number problem, the best known polynomial algorithm approximates not directly the crossing number but $|V|+\mathrm{cr}(G)$ within a factor of $\log^3 |V|$ [54], and is restricted to graphs with bounded degree. The only constant factor approximations for $\mathrm{cr}(G)$ known are for projective [68], almost-planar, and apex graphs, all under the restriction of bounded degrees. Interestingly, both latter algorithms are two-fold in the following sense: On one hand, there is the polynomial algorithm to solve the edge insertion problem to optimality [77]; on the other hand it can be shown that an optimal solution for this problem constitutes an approximation for almost-planar graphs (i.e., graphs G with an edge e such that $G-e$ is planar) with bounded degree [86, 26]. Analogously, there is a recent polynomial algorithm to solve the *vertex insertion* problem to optimality [32] (i.e., inserting a star—a vertex plus incident edges—into an optimally chosen embedding of a planar graph); it can be shown that an optimal solution for this problem constitutes an approximation for apex graphs (i.e., graphs G with a vertex v such that $G - v$ is planar) with bounded degree [34].

Another advancement in algorithmic theory concerning the crossing number was spear-headed by Grohe [72], who showed in 2001 that the problem



(a) Edges do not cross multiple times.

(b) Adjacent edges do not cross.

(c) Edges do not cross themselves.

Figure 2.2: Properties of a good drawing.

is *fixed parameter tractable* (FPT) in quadratic time, i.e., one can test in quadratic time whether a graph can be drawn with at most k crossings, if k is fixed. Clearly, the algorithm's running time is exponentially dependent on k. Also due to the high constants involved, this algorithm is of pure theoretical interest. Recently, Kawarabayashi and Reed [95] showed that the problem is even FPT in linear time; still the algorithm remains infeasible from the practical point of view.

In this thesis we will especially focus on *practical exact* algorithms to compute the crossing number of a graph, see Section 5 and 6. For this class of algorithms, there have not been any other algorithms before, besides from trivial enumeration schemes which become inapplicable even for very small graphs.

Good Drawings and Non-simple Graphs. In the paragraphs above, we restricted ourselves to simple undirected graphs. Clearly, we can also compute the crossing number of directed graphs by considering their shadows, as long as there are no special properties demanded for the drawing of the arcs. The latter is the case for *upward drawings* of directed graphs, where all edges (or at least as many edges as possible) have to be drawn in an upward fashion, i.e., while tracing the curve from the source to the target node of a directed arc, the vertical y-coordinate must monotonously increase. In this thesis, we do not deal with such drawings.

We say a drawing of some graph G is a *good drawing*, if it satisfies the three properties below, cf. Figure 2.2. It is important to recognize that any crossing minimal drawing is in fact a good drawing.

Edges do not cross multiple times. Assume there is some crossing minimal drawing $\mathcal{D}$ in which the edges $e = \{u_e, v_e\}$ and $f = \{u_f, v_f\}$ cross at least twice. Let c and d be any two crossing points between e and f. Assume w.l.o.g. that u_e and u_f are closer to c than to d when tracing along the respective edge curves. For e, we have the curve segments e_1 between u_e and c, e_2 between c and d, and e_3 between d and v_e. Analogously, we have the segments f_1, f_2, f_3 for f.

We can consider a modified drawing that results from rerouting e along

$\langle e_1, f_2, e_3\rangle$ and f along $\langle f_1, e_2, f_3\rangle$. Thereby, we displace e and f by an arbitrarily small epsilon such that they do not coincide at c and d. We get rid of these two crossings, while all other crossings in the drawing remain as they were. Hence $\mathcal{D}$ was not crossing minimal.

Adjacent edges do not cross. Assume there would exist some crossing minimal drawing $\mathcal{D}$ in which the adjacent edges e and f cross. Let $v = e \cap f$ be the vertex incident to both edges. As we know that edges do not cross multiple times, let c be the unique point where e and f cross. Hence we have the curve segments e_1 and f_1 of e and f, respectively, between v and c. And we have the segments e_2 and f_2 for the respective remaining curves.

Analogously to above, we can reroute e along $\langle f_1, e_2\rangle$, and f along $\langle e_1, f_2\rangle$. By a small enough displacement around the touching point c, and since the remainder of the drawing remains identical, we obtain a drawing with one crossing less. This contradicts the crossing minimality of $\mathcal{D}$.

Edges do not cross themselves. Assume there is an edge $\{u, v\}$ which crosses itself at c, and therefore forms a loop. We can simply redraw the edge using the segments between u and c and between c and v, getting rid of the loop-segment and the crossing at c.

Considering self-loops, it is clear that they do not influence the crossing number: We can draw a graph without its self-loops first, and then add the loops in arbitrarily small regions incident to the corresponding vertices.

If the given graph has multi-edges, we can observe that there will always be a crossing minimal drawing where all the separate edges are routed along the same curve, arbitrarily close to each other: Assume a drawing where two edges with the same end nodes are routed differently. We can reroute the edge involved in more crossings such that it runs parallel to the other edge, choosing the rerouted edge arbitrarily if the number of crossings is equal. This transformation does not increase the number of crossings. Hence, when given a multi-graph $\bar{G} = (V, \bar{E})$, we can instead consider a simple graph $G = (V, E)$ which has a single edge for each pair of nodes connected by one or more edges in $\bar{E}$. Furthermore, we define integer edge weights $w : E \to \mathbb{N}$ which give the number of multi-edges corresponding to an edge of E. We then have the *weighted crossing number*: The cost of a crossing between the edges e and f is $w(e) \cdot w(f)$, since such a drawing will result in that many crossings when transforming G back into $\bar{G}$. The weighted crossing number then is not the minimum number of crossings, but the minimum sum of crossing costs. Note that the above observations on adjacent edges and multiple crossings still hold for the weighted crossing number.

Interestingly, most algorithms for the traditional crossing number can also be applied to the weighted problem variant. The need for the integer

edge weights is further strengthened by the fact that certain preprocessing strategies, in particular the non-planar core reduction described in Section 4.2, work more efficiently when allowing such weights. The algorithms presented herein, in particular also the exact approaches, all allow weighted edges.

Chapter 3

Preliminaries: Integer Linear Programs

crossing: *(Religious Architecture)* That part of a cruciform church in which the transepts intersect the nave.

This section only gives a brief overview on the key concepts of linear and integer linear programming and corresponding solving strategies. We thereby restrict ourselves to the minimum required in the course of this thesis. The definitions are based on the books by Nemhauser and Wolsey [117], Schrijver [134], and Wolsey [144], all of which give a much more in-depth theoretical background on this topic.

The crossing number problem and related problems studied in this thesis belong to the class of *combinatorial optimization problems*. Linear programs and integer linear programs proved to be useful tools to solve many kinds of optimization problems. Especially the latter turned out to be particularly applicable for combinatorial optimization problems: many such problems can be reformulated as (integer) linear programs and then solved as such.

A *linear program* (LP) is a system of linear inequalities and a linear objective function. Canonically we write

$$\min\{cx \ : \ Ax \leq b, x \geq \mathbf{0}\} \tag{LP}$$

where A is a $m \times n$ matrix, c an n-dimensional row vector, b an m-dimensional column vector, and x an n-dimensional column vector of variables[1]. We say, c is the *cost vector*, A the *constraint matrix* and b the right-hand side. The

[1]In most textbooks, the canonical form for LPs are maximization problems, but the transformation into a minimization problem is easily possible by multiplying c by -1. For consistency, we will use the latter problem type in this overview, since this thesis exclusively deals with minimization problems.

inequality $A_i x \leq b_i$, where A_i is the ith row of A is called a *constraint*. All points $\bar{x} \geq \mathbf{0}$ which satisfy $A\bar{x} \leq b$ are *feasible solutions* to the LP, and the objective is to find a feasible solution x^* which minimizes cx^*. The probably most important fact for LPs is that the feasible solutions—given there are any—form a convex polyhedron in the n-dimensional space. In the following, we will only consider LPs with *bounded* polyhedrons, i.e., polytopes.

We have an *integer linear program* (ILP) when we add the restriction that all variables have to take integer values:

$$\min\{cx \ : \ Ax \leq b, x \geq \mathbf{0} \text{ integer}\}. \tag{ILP}$$

In the following we will only consider ILPs arising from combinatorial optimization, i.e., we have a finite set of feasible solutions.

When we require only some but not all variables to be integer, we obtain a *mixed integer linear program* (MIP). When we require all variables to be not only integer but either 0 or 1, we have a 0/1-integer linear program (0/1-ILP), also known as *binary integer linear program* (BIP):

$$\min\{cx \ : \ Ax \leq b, x \in \{0,1\}^n\}. \tag{0/1-ILP}$$

The ILPs considered in this thesis will all belong to the latter class.

Solving (polynomially sized) LPs is a very well studied topic and can be done in polynomial time, using quite intricate methods such as the ellipsoid method or interior point methods. Usually, most LP-solvers use the Simplex algorithm (or variants thereof); although it has an exponential running time in the worst-case, it is in practice not only the simplest but in particular the fastest algorithm to solve linear programs. On the other hand, solving general ILPs, MIPs, and 0/1-ILPs is NP-hard. Furthermore, in many applications we deal with exponentially sized constraint matrices. Therefore, several solution strategies have been devised, which allow to find optimal or provably near-optimal solutions for many such problems and real-world problem instances.

Cutting Planes. Let S be the (finite) set of all integer feasible solutions to some ILP. A usual approach is to consider not only S, but a convex polyhedron in n-dimensional space containing all points S. It can be shown that solving an ILP over S is equivalent to solving an LP with the same objective function over the polyhedron given by the convex hull of S. Unfortunately, it is in general not possible to give a polynomially sized description of this convex hull, hence we use polyhedra which are larger and contain the convex hull.

The most commonly considered polyhedron corresponds to the *LP relaxation*, i.e., the LP resulting from ignoring the integrality constraints. In case of a 0/1-ILP, we obtain the relaxation

$$\min\{cx \ : \ Ax \leq b, \mathbf{0} \leq x \leq \mathbf{1}\}. \tag{LP relaxation of a 0/1-ILP}$$

Clearly, each feasible solution of the original ILP is also feasible in its LP relaxation. On the other hand, there are no additional integer feasible solutions. By solving such an LP relaxation, we obtain a *fractional solution*. The value of the objective function for this solution is a *dual bound* (a lower bound in case of minimization, an upper bound in case of maximization) to the optimal ILP solution. Hence, if the fractional solution turns out to be integral for all variables, we found an optimal solution of the original ILP. If this is not the case, one can try to identify *valid cuts*, also known as *cutting planes*, i.e., constraints which do not influence the feasibility of any integer point but "cut off" the current fractional solution from the feasible polyhedron. A lot of research, starting in the 1950s by Dantzig et al. [41] and Gomory [70], has been conducted on identifying integer-valid cuts. The concept of cuts can be generalized by allowing the considered polyhedron to contain integer points which are infeasible for the original ILP. This is particularly interesting for ILPs where the constraint matrix contains too many (e.g., exponentially many) rows, in order to solve the LP relaxation efficiently. Thereby we choose only a subset of all constraints—the so-called *active constraints*—for the initially considered ILP model, and solve the corresponding LP relaxation. The obtained solution then has to be checked whether it violates any original constraints not yet in the current model. The process of identifying such violated constraints is called *separation routine*, and in many applications there exist polynomial algorithms to check an exponentially sized set of inactive constraints. When one or more such violated constraints or integer-valid cuts have been identified, they are added to the current model, and the problem is resolved iteratively. The process stops when no more cuts can be found, giving a hopefully strong dual bound for the original ILP.

Column Generation. Analogously to the concept of active and inactive constraints, we can consider *active* and *inactive variables*. This approach is particularly interesting when the constraint matrix contains too many (e.g., exponentially many) columns to solve efficiently, i.e., the ILP contains too many variables, and most of these variables will be 0 in the optimal solution.

We start with a subset of variables, and consider all inactive variables—the ones not in the current model—to be 0. After solving the corresponding LP relaxation, one has to figure out if the objective function can be improved by activating currently inactive variables. The concept of adding variables like that is called *column generation*. Dantzig and Wolfe [42] presented a general technique we call *algebraic pricing*[2] that can be used to decide which variables have a potential to be beneficial if included. The key idea is that this decision is based purely on the *reduced costs* of the inactive variable;

[2]In the literature, the terms "pricing" and "column generation" are often used interchangeably. To clarify our distinction between the Dantzig-Wolfe machinery and other strategies, we will use the term "algebraic pricing" for the former. See also Section 6.1.4 and the sections thereafter.

in particular, we only have to look at the sign of these costs, cf. Section 6.2. The pricing problem hence is to identify variables with negative reduced costs, among the set of possibly exponentially many inactive variables.

In this thesis we will describe also another variant of column generation which we call *combinatorial column generation*; thereby we do not use reduced costs as the activation criterion, but some criterion based on graph theoretical observations.

Branch-and-Bound. In general, both cutting planes and column generation will not yield optimal integer solutions. To solve ILPs to provable optimality, the most common approach is to use branch-and-bound techniques. In its pure form, the root node of the branch-and-bound tree corresponds to solving the LP relaxation of the given ILP. We then select some variable—e.g., one that has a fractional part near 0.5 in the solution—and generate two subproblems: in one, the variable is fixed to 0, in the other it is fixed to 1. We can recursively solve the corresponding LP relaxation for each subproblem, etc. The process stops when an all-integer solution is found, and there are no unsolved subproblems with a dual bound lower than that solution.

Usually, this approach is combined with some *primal heuristic*, i.e., a traditional heuristic which gives primal bounds (i.e., upper bounds in minimization problems). Often, these are *LP-based heuristics*, i.e., they take the fractional solution of the current LP relaxation into account to guide the heuristic. Note that whenever some LP relaxation in a branch-and-bound node other than the root node gives an integer solution, this can also be considered a heuristic solution.

The benefit of such (global) primal bounds is that we can prune all subproblems (and their subtrees in the branch-and-bound tree) which have a lower bound greater or equal to this primal bound.

Branch-and-Cut(-and-Price). In 1991, Padberg and Rinaldi [122] combined the cutting-plane approach with the branch-and-bound strategy, resulting in a *branch-and-cut* algorithm. In such algorithms, we solve each branch-and-bound node not using the pure LP relaxation of the full ILP, but use cutting planes to obtain stronger bounds and deal with the probably otherwise prohibitive number of constraints. Branch-and-cut algorithms turned out to be very effective in practice, and lead to the most effective optimal algorithms in many applications, cf. e.g. [4, 36, 108], sometimes even beating non-optimal meta-heuristics in terms of running time.

The conceptual idea of branch-and-cut can be reused to give *branch-and-price* algorithms—by combining branch-and-bound with column generation techniques at the nodes—and the full combination of all three techniques in *branch-and-cut-and-price* algorithms.

Part II

Theory & Algorithmics

Chapter 4

Preprocessing

crossing: *(Genetics)*
The act of mixing different species or varieties of animals or plants and thus to produce hybrids.

Preprocessing means to shrink or decompose a given problem into one or more smaller subproblems which are then easier to solve. There are several preprocessing techniques available for the crossing number problem and related problems. The transformation from a multi-graph with self-loops into a simple graph with edge weights can also be seen as a preprocessing strategy, as it shrinks the instance by temporarily removing the self-loops and merging parallel edges such that only a single drawing-path for each bundle of edges has to be computed. In the following, we will always consider a simple undirected graph $G = (V, E)$, possibly with integer edge weights w.

4.1 Traditional Preprocessing

Let $C_1, C_2, \ldots$ be the connected components of G. It is clear that we can compute $\mathrm{cr}(G) = \sum_i \mathrm{cr}(C_i)$ by computing the crossing numbers of these components individually. Hence, in the following we will restrict ourselves to G being connected.

Let $B_1, B_2, \ldots$ be the blocks of G. Again, we can compute $\mathrm{cr}(G) = \sum_i \mathrm{cr}(B_i)$ by considering the blocks individually. Since the blocks are edge-disjoint, we clearly have $\sum_i \mathrm{cr}(B_i) \leq \mathrm{cr}(G)$; it remains to show that we can realize a drawing $\mathcal{D}$ of G by pasting optimal drawings of the blocks together without introducing additional crossings. Let $\mathcal{D}_1, \mathcal{D}_2, \ldots$ be optimal drawings of the blocks on spheres. Hence, by selecting any appropriate face, we can easily generate a drawing in the plane where any prespecified node lies on the outer face. Hence our pasting algorithms works by iteratively building a complete drawing $\mathcal{D}$: We start with an arbitrary block and fix its planar

drawing by choosing an arbitrary outer face. We then iteratively take a not yet drawn block B_i which has a node v in common with the current drawing $\mathcal{D}$. We choose an arbitrary face φ in $\mathcal{D}$ incident to v and select a planar drawing $\mathcal{D}'_i$ of $\mathcal{D}_i$ where v lies on the outer face. We can then paste $\mathcal{D}'_i$ into φ of $\mathcal{D}$, identifying the two images of v, without introducing any crossings.

Hence, in the following we can restrict ourselves to G being biconnected.

4.2 Non-planar Core Reduction

The following preprocessing scheme is based on the analysis of the triconnected components of a graph. These components form a tree of linear size which we will discuss in the following.

4.2.1 SPR-trees

The SPQR-tree was first defined by Di Battista and Tamassia [46], based on prior work of Bienstock and Monma [12]. It makes use of the linear-time algorithm to decompose a biconnected graph into its triconnected components by Hopcroft and Tarjan [88], which has later been corrected by Gutwenger and Mutzel [75]. SPQR-trees have been used in various applications mostly in the context of planarity and triconnectivity. One of their main properties is that they can be seen as a representation and enumeration tool for all exponentially many planar embeddings of a graph, although these trees have only linear size.

We will give an alternative but compatible definition to the one usually used, cf. [46, 143]; this will simplify our data structure, without sacrificing any of its important properties. We consider the SPQR-trees unrooted and undirected; the original definition roots and directs the tree choosing any leaf arbitrarily, which is not necessary in our context. The second modification deals with the tree's node types, which are S,P,Q, and R. While S-, P- and R-nodes hold structural information of the graph, Q-nodes are only redundant representations of the graph's edges. We will omit the latter nodes, hence technically defining an *SPR-tree*. Furthermore, our definition does not require the definitions of separation pairs, split pairs, and maximal split pairs. For the sake of completeness, the section after this one will prove the soundness of our alternative definition. See Figure 4.1 for an example of an SPR-tree, according to the following definition:

Definition 4.1 (SPR-tree)**.** Let $G = (V, E)$ be a biconnected graph with at least three nodes. The SPR-tree $\mathcal{T}$ of G is the smallest tree satisfying the following properties:

1. Each node ν in $\mathcal{T}$ corresponds to a *skeleton* $G_\nu^{\mathcal{T}} = (V_\nu^{\mathcal{T}}, E_\nu^{\mathcal{T}})$ which is a simplification of G, as certain subgraphs are replaced by single *virtual*

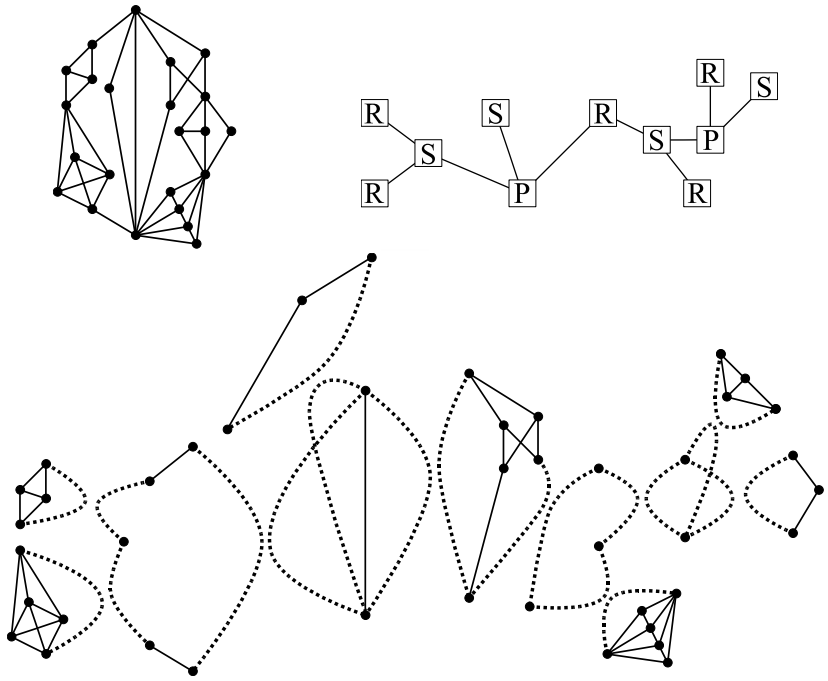


Figure 4.1: A graph (top left), its SPR-tree (top right), and its decomposition showing the skeletons (bottom). Virtual edges are represented by dotted lines.

edges. (We may omit the superscript in our notation, if the considered SPR-tree is clear from the context.)

2. The tree has three different node types with specific skeleton structure:

   S: The skeleton is a simple cycle—it represents a *serial* component.

   P: The skeleton consists of two vertices and multiple edges between them—it represents a *parallel* component.

   R: The skeleton is a simple triconnected graph. (Since planar triconnected graphs have only a single combinatorial embedding and its mirror, this structure was originally considered a *rigid* structure.)

3. For the edge $\{\nu, \mu\}$ in $\mathcal{T}$ we have: G_ν contains a virtual edge e_μ which represents the subgraph described by G_μ, and vice versa.

4. We can obtain the original graph G by iteratively merging the skeletons of adjacent tree nodes: For the edge $\{\nu, \mu\}$ in $\mathcal{T}$, let e_μ (e_ν) be the virtual edge in ν (μ) representing the subgraph described by G_μ (G_ν, respectively). Clearly, both edges e_μ and e_ν connect the same nodes, say u and v. We obtain a merged graph $(V_\nu \cup V_\mu, E_\nu \cup E_\mu \setminus \{e_\mu, e_\nu\})$ by glueing the graph together at u and v and removing e_μ and e_ν.

The non-virtual edges in a skeleton are referred to as *original edges*.

Observation 4.2. *Each original edge appears only once over all skeleton graphs. Each virtual edge appears in exactly two skeleton graphs.*

Observation 4.3. *The SPR-tree does not contain any adjacent S-nodes. Nor does it contain adjacent P-nodes.*

Proof. Assume there are two adjacent S-nodes ν and μ. When merging them along their corresponding virtual edges, we obtain a graph which has the structure required for an S-node. Hence we can introduce a single S-node replacing ν and μ, obtaining a smaller SPR-tree, which is a contradiction to the trees minimality. The analogous holds for P-nodes. □

Lemma 4.4. *The SPR-tree $\mathcal{T}$ of a biconnected graph G always exists.*

Proof. Let G be the smallest graph which has to corresponding SPR-tree, i.e., its structure cannot be modeled by gluing S-, P- and R-skeletons together. G cannot be triconnected—the SPR-tree would be a single R-node. Nor can it have any triconnected component—attached to the rest of the graph by some nodes u and v—as we could model the component via an R-node, and replace it in G by an edge $\{u, v\}$ which would give a smaller graph without an SPR-tree. Since G is biconnected and has at least three nodes, we know there is a cycle in G. Let $\langle c_1, B_1, c_2, B_2, \ldots, c_k, B_k, c_1\rangle$ $(k \geq 3)$ be such a cycle with c_i $(1 \leq i \leq k)$ being cut vertices, and B_i the blocks between them. The blocks may be degenerated to single edges. Then we can model this cycle via an S-node. The skeleton of this node is a cycle of k edges $e_1, \ldots, e_k$: Edge e_i $(1 \leq i \leq k)$ is the original edge of B_i if B_i is degenerated, or a virtual edge representing B_i otherwise. If G cannot be represented by an SPR-tree, then one of the blocks $B_1, \ldots, B_k$ cannot be modeled by such a tree, which is a contradiction to the minimality of G. □

Lemma 4.5. *The SPR-tree $\mathcal{T}$ of a biconnected graph G is unique.*

Proof. Assume there would be two different SPR-trees $\mathcal{T}_1$ and $\mathcal{T}_2$. Since both are minimal, they have the same size. Assume there is some R-node with skeleton $G' = (V', E')$ unique to $\mathcal{T}_1$, and assume w.l.o.g. that $\mathcal{T}_2$ does not contain an R-node with a skeleton which has G' as a subgraph. Since G' is triconnected and simple, we cannot find a separation pair. Hence we cannot substitute any substructure of G' via a single arc and still satisfy Property 4 of the definition. Since the skeleton G' is unique to $\mathcal{T}_1$, but $\mathcal{T}_2$ satisfies Property 4, $\mathcal{T}_2$ has to contain an R-node whose skeleton G'' "overlaps" G', i.e., $G'' \cap G' \neq \emptyset$ and $G'' \setminus G' \neq \emptyset$. Hence there is a connected component C in G'' but not in G' which is attached to G' via at least 3 nodes. Since it is impossible for $\mathcal{T}_1$ to glue connected components attached to more than two nodes to some other component (G'), $\mathcal{T}_1$ would not be a valid SPR-tree. Hence we know that all R-nodes are both in $\mathcal{T}_1$ and $\mathcal{T}_2$.

Assume there is some P-node with skeleton G' unique to $\mathcal{T}_1$, and assume w.l.o.g. that $\mathcal{T}_2$ does not contain a P-node with a skeleton which has G' as

a subgraph. Let G' have edges $e_1, \ldots, e_k$ between the nodes u and v. Since $\mathcal{T}_1$ contains no adjacent P-nodes, we know the skeletons represented by any virtual edge e_i ($1 \leq i \leq k$) are either cycles or triconnected components. But then, the substructure described by G' can only be modeled in $\mathcal{T}_2$ by multiple adjacent P-nodes—which does not happen in valid SPR-trees.

Hence, since $\mathcal{T}_1$ and $\mathcal{T}_2$ agree on the R- and P-nodes, they also agree on the S-nodes, as they are non-adjacent and of equal number and therefore uniquely defined. □

We omit the proof of the linearity lemma, as it follows from the consistency between our definition and the traditional SPQR-tree definition for which it has been shown:

Lemma 4.6. *The SPR-tree $\mathcal{T}$ of G has linear size in G and can be computed in linear time.*

4.2.2 Supplemental: Compatibility with SPQR-trees

In this section, we prove that our definition of SPR-trees is compatible to the traditional definition of SPQR-trees—readers only interested in the reduction strategy may safely skip this section. We will cite the definition given in [143], describe the transformation between SPQR- and SPR-trees, and finally show their consistency.

The following definitions from [143] are given verbatim (denoted by the vertical line on the left-hand side border), only adjusted to match the notation used in this thesis and under omission of text explaining complementing figures or non-crucial definitions and properties. The numbering scheme of the definitions and observations is consistent with this thesis rather than with the original source.

> A pair $\{u, v\}$ of vertices in a biconnected graph G is a *split pair*, if $\{u, v\}$ is a separation pair or $\{u, v\}$ is an edge in G. A *split component* of a split pair $\{u, v\}$ is either an edge that connects u and v together with the vertices u and v, or a maximal connected subgraph G' of G such that removing the vertices u and v does not disconnect G'. [...]
>
> In the definition of SPQR-trees, we will need the notion of the *split graph* of a split pair with respect to some edge e in a graph G. The split graph of a split pair $\{u, v\}$ is the union of all the split components of G that do not contain the edge e. Informally, the split graph of the split pair $\{u, v\}$ is the part of the original graph that can only be reached from e via u or v. [...]
>
> We say that a split pair $\{u, v\}$ is *dominated* by another split pair $\{x, y\}$ w.r.t. an edge e if the split graph of $\{u, v\}$ w.r.t. e is a proper subgraph of the split graph of $\{x, y\}$ w.r.t. e. So if a split pair dominates another split

pair w.r.t. an edge, the dominated split pair can only be reached from the edge by passing a vertex of the dominating split pair. [...]

Using the dominance relation on split pairs, we can define a *maximal split pair* w.r.t. a certain edge e in a graph as a split pair that is not dominated by any other split pair in the graph w.r.t. e. There may be several maximal split pairs in a graph w.r.t. a certain edge. [...]

The nodes [of the SPQR-tree] have four different types (S, P, Q, and R) that are explained in the definition below. Each node is associated with a special graph which is called the *skeleton* of that node. We can think of each skeleton as a simplified version of the original graph where some subgraphs have been replaced by single edges.

The tree is constructed by recursively decomposing the original graph into triconnected components. This decomposition starts with an arbitrary edge of the graph which is called *reference edge* of the decomposition. The node in the SPQR-tree corresponding to this edge will be the root of the tree.

Definition 4.7 (Proto-SPQR-tree)**.** Let $G = (V, E)$ be a biconnected [multi]graph and $e = \{s, t\} \in E$ one of its edges. Get G' be the graph after deletion of the edge e ($G' = (V, E \setminus \{e\})$). Then the *Proto-SPQR-tree* $\mathcal{T}''$ with reference edge e is defined as follows:

- **Trivial case:** G' is a single edge. In this case, $\mathcal{T}'$ consists of just one Q-node. The skeleton of this node is G itself (so it consists of two vertices connected by two edges).

- **Serial case:** G' is not biconnected and is not a single edge. So let v_1 to v_{k-1} (with $k > 1$) be the cut vertices of G'. Since G is biconnected, we know that the blocks of G' must form a chain, so there are k blocks B_1 to B_k such that v_i is only contained in the two blocks B_i and B_{i+1} for $1 \leq i \leq k-1$. We assume that s is contained in B_1 and t in B_k. [...]

  The root of $\mathcal{T}''$ is an S-node μ where the skeleton S is a simple cycle containing the vertices $s, v_1, \ldots, v_{k-1}, t$ in that order. The edge $\{s, t\}$ in S is the *virtual edge* of S. If we define v_0 as s and v_k as t, then the edge $e_1 = \{v_{i-1}, v_i\}$ represents the block B_i in G for $1 \leq i \leq k$. [...]

  The children of μ are defined by graphs G_i that are constructed from B_i by adding edge e_i for $1 \leq i \leq k$. Edge e_i is the reference edge of G_i. The k children of μ are the roots of the Proto-SPQR-trees for the subgraphs G_i.

- **Parallel case:** The vertices s and t are a split pair of G' with the split components $C_1, \ldots, C_k$ where k is at least two. [...] The root of $\mathcal{T}''$ is a P-node μ whose skeleton S consists of two vertices s and t

and the edges $e_1, \ldots, e_{k+1}$. Edge e_i for $1 \leq i \leq k$ represents subgraph C_i while edge e_{k+1} is the virtual edge of the skeleton. [...]

The children of μ are defined by the graphs $G_1, \ldots, G_k$ where G_i is constructed from C_i by adding edge e_i. The children of μ are the roots of the Proto-SPQR-trees for the G_i where the reference edge is e_i.

- **Rigid case:** Neither of the previous cases is applicable. So the pair $\{s, t\}$ is not a split pair of G' with at least two split components and G' is biconnected. Then the root of $\mathcal{T}''$ is an R-node μ. Let $\{s_1, t_1\}, \ldots, \{s_k, t_k\}$ be the maximal split pairs of G with respect to $\{s, t\}$. For each of these maximal split pairs $\{s_i, t_i\}$, we define the graph U_i as the split graph of $\{s_i, t_i\}$ with respect to $\{s, t\}$. [...]

  The vertices in the skeleton S of μ are the vertices s and t together with all vertices s_i and t_i for $1 \leq i \leq k$. Apart from the virtual edge $\{s, t\}$, S contains k edges where edge e_i connects the vertices s_i and t_i. Edge e_i represents subgraph U_i for $1 \leq i \leq k$. [...]

  The children of μ are defined by the graphs $G_1, \ldots, G_k$ where G_i is constructed from U_i by adding edge e_i. The children of μ are the roots of the Proto-SPQR-trees of the G_i with reference edge e_i.

Definition 4.8 (SPQR-tree)**.** The SPQR-tree $\mathcal{T}'$ of a biconnected graph G consists of a Q-node representing the reference edge e whose child is the root of the Proto-SPQR-tree $\mathcal{T}''$ for G with reference edge e.

We can observe the following properties of SPQR-trees, as given in [143].

Observation 4.9. *All leaves of an SPQR-tree [...] are Q-nodes.*

Observation 4.10. *The skeletons of R-nodes are triconnected graphs.*

Observation 4.11. *Let G be a biconnected graph with at least two edges and let $\mathcal{T}''$ be the Proto-SPQR-tree of G w.r.t. reference edge e. Then there is exactly one Q-node q_i in $\mathcal{T}''$ for each edge $e_i \in E \setminus \{e\}$ where e_i is contained in the skeleton S_i of q_i.*

Observation 4.12. *Let G be a biconnected graph and e_1 and e_2 two of its edges. Let $\mathcal{T}_1'$ be the SPQR-tree of G with reference edge e_1 and $\mathcal{T}_2'$ the SPQR-tree of G with reference edge e_2. Modifying $\mathcal{T}_1'$ by making the Q-node that represents e_2 the root of the tree results in $\mathcal{T}_2'$.*

The latter observation shows that the SPQR-tree is unique w.r.t. choosing some Q-node as its root. Furthermore, the recursion of the above SPQR-tree construction ensures:

Observation 4.13. *The SPQR-tree as defined above has no adjacent S-nodes and no adjacent P-nodes.*

We define a transformation called *Q-addition*, in order to obtain a traditional SPQR-tree from the SPR-tree $\mathcal{T}$: In each G_ν, we replace each original edge e by a virtual edge e_μ and insert a Q-node μ together with the tree edge $\{\nu, \mu\}$. The skeleton G_μ thereby consists of the original edge e and a virtual edge with the same start and end vertex as e which represents the rest of the graph. We can root the tree arbitrarily by choosing any Q-node. We denote the resulting tree by $Q_e^+(\mathcal{T})$, whereby e is the original edge in the root's skeleton.

It is clear from the definitions and the Q-addition that both SPQR-tree types are at least similar, as the structural properties of the skeletons coincide. Furthermore, the SPQR-tree according to Definition 4.8 also satisfies Property 4 of the SPR-tree definition 4.1, i.e., we can obtain the original graph by merging the skeletons according to the tree structure. We will show that the SPQR-tree definitions are equivalent.

Inversely to the Q-addition we can define the following *Q-substraction* transformation: Start with the SPQR-tree $\mathcal{T}'$ according to Definition 4.8. Consider $\mathcal{T}'$ undirected and prune all leaves, i.e., all Q-nodes (in the directed tree, the Q-nodes are either the root or leaves). Finally, mark the virtual edges which do not have a child anymore as *original edges* instead. We denote the resulting graph by $Q^-(\mathcal{T}')$.

Based on Observation 4.12 we can see:

Observation 4.14. *Let $\mathcal{T}'$ be any SPQR-tree according to Definition 4.8. The tree $Q^-(\mathcal{T}')$ is unique and independent on the reference edge chosen for $\mathcal{T}'$.*

Observation 4.15. *Let $\mathcal{T}$ be the SPR-tree according to Definition 4.1, and let $\mathcal{T}'$ be an SPQR-tree according to Definition 4.8 w.r.t. an edge e'. Let e be any edge of G. Applying the Q-substraction and the Q-addition sequentially (or vice versa) gives the original tree:*

$$Q_{e'}^+(Q^-(\mathcal{T}')) = \mathcal{T}', \quad Q^-(Q_e^+(\mathcal{T})) = \mathcal{T}.$$

Observation 4.16. *Let $\mathcal{T}'$ be an SPQR-tree according to Definition 4.8. The tree $Q^-(\mathcal{T}')$ satisfies the properties 1.–4. stated in the SPR-tree definition.*

Lemma 4.17. *Let $\mathcal{T}$ be the SPR-tree according to Definition 4.1, and let $\mathcal{T}'$ be an SPQR-tree according to Definition 4.8. The number of nodes in $Q^-(\mathcal{T}')$ is equal to the number of nodes in $\mathcal{T}$.*

Proof. Since $\mathcal{T}$ is minimal by definition, we have to show that $Q^-(\mathcal{T}')$ does not require more nodes. Since the recursive definition of $\mathcal{T}'$ stops when finding a triconnected graph, we know that it will not decompose them further. Hence the number of R-nodes in $\mathcal{T}'$ cannot be larger than in $\mathcal{T}$.

Analogous to the proof of Lemma 4.5, assume there is a P-node μ unique to $\mathcal{T}$. It has the skeleton G' and the edges $e_1, \dots e_k$ between u and v, none of which represents another parallel component. Consider the recursive construction of $\mathcal{T}'$, and consider the step where for the first time some edge e_i ($1 \leq i \leq k$) is selected as a reference edge. Note that this a situation has to arise as $\{u, v\}$ is a maximal split pair in G' for any edge e_i (substituting the other edges by their respective skeletons). Assume w.l.o.g. $i = 1$. Then $\{u, v\}$ would be a split pair and we would thus construct a P-node with a skeleton having the reference edge e_1 and the parallel edge $e_2, \dots, e_k$. This node is therefore equivalent to μ, which contradicts the uniqueness of μ to $\mathcal{T}$.

Hence, $\mathcal{T}$ and $\mathcal{T}'$ agree on the R- and P-nodes. Therefore they also agree on the S-nodes, since they are non-adjacent in $\mathcal{T}$ and $\mathcal{T}'$. □

Putting everything together we can conclude:

Theorem 4.18. *Let $\mathcal{T}$ be the SPR-tree according to Definition 4.1, and let $\mathcal{T}'$ be any SPQR-tree according to Definition 4.8 w.r.t. edge e. We have*

$$Q^-(\mathcal{T}') = \mathcal{T}, \quad Q_e^+(\mathcal{T}) = \mathcal{T}'.$$

Hence, the definitions 4.1 and 4.8 are compatible (with respect to undirectedness and the omission of the Q-nodes).

Proof. Due to Observation 4.16 and Lemma 4.17, we know that $Q^-(\mathcal{T}')$ is a valid SPR-tree. Since Lemma 4.5 guarantees that the SPR-tree is unique, we have $Q^-(\mathcal{T}') = \mathcal{T}$. The second equation then follows from Lemma 4.15. □

4.2.3 Defining the Non-planar Core

Definition 4.19 (Planar 2-Component)**.** Let $s, t \in V$ be two distinct vertices, and $E_H \subset E$ a proper subset of edges. We call the edge-induced subgraph $H = G[E_H]$ a *planar 2-component* of G with *contact points* s and t, if $H + \{s, t\}$ (i.e., H plus the edge connecting the contact points) is planar and $V(H) \cap V' = \{s, t\}$. Thereby $V' := V(G[E \setminus E_H])$ denotes the vertex set of the graph induced by the edges not contained in H. For brevity, we also call H a *planar st-component*. A single edge $e = \{s, t\}$ is a *trivial* planar *st*-component.

Intuitively, a planar *st*-component H is a subgraph that connects to the rest of the graph only via its contact points. The 2-connectivity of G implies that $H + \{s, t\}$ is also 2-connected.

Definition 4.20 (Maximal Planar 2-Component)**.** Let H be a non-trivial planar 2-component of G. We call H a *maximal planar 2-component* of G, if and only if there is no planar 2-component H^* of G with $H \subset H^*$.

An important property of maximal planar 2-components is that they do not overlap:

Lemma 4.21. *All maximal planar 2-components of G are pairwise vertex and edge disjoint, except for their contact points.*

Proof. Consider two distinct maximal planar 2-components H_1 and H_2 with contact points s_1, t_1 and s_2, t_2, respectively. Let

$$\bar{V} := (V(H_1) \cap V(H_2)) \setminus \{s_1, t_1, s_2, t_2\}$$

be the common vertices between the components, disregarding the contact points, and assume that $\bar{V} \neq \emptyset$.

Since the components H_1 and H_2 are maximal, $E_1 := E(H_1) \setminus E(H_2)$ and $E_2 := E(H_2) \setminus E(H_1)$ are non-empty sets, i.e., H_2 contains edges not in H_1 and vice versa. Since H_2 is a planar 2-component, only its contact points are incident to the rest of the graph. Hence E_1 cannot contain edges incident to $\bar{V}$. Analogously, neither can E_2 contain any edges incident to $\bar{V}$.

Therefore, the edges in H_1 and H_2 incident to s_1, t_1 and s_2, t_2, respectively, are in $E(H_1) \cap E(H_2)$. Thus H_1 contains the contact points of H_2 and vice versa. Since they are both planar 2-components, they are only connected to the rest of the graph via two contact points, hence the pairs of their respective contact points have to be identical. But then, the union $H_1 \cup H_2$ would also be a planar 2-component, which contradicts the maximality of H_1 and H_2.

When H_1 and H_2 are vertex disjoint, except for their contact points, they are also edge disjoint, except for edges $\{s_1, t_1\}$, if $s_1 = s_2$ and $t_1 = t_2$. $H_1 \cup H_2$ would be a planar 2-component, which again contradicts the maximality of H_1 and H_2. □

The main idea of the core reduction is to replace each maximal planar st-component by a single edge $\{s, t\}$ of appropriate weight. We will analyze such a maximal planar st-component H further, to obtain its suitable weight:

Definition 4.22 (Cut)**.** A *cut* in H is a bipartition $(W, \bar{W})$ of the vertices of G. The *capacity* $c(W, \bar{W})$ of the cut is the cardinality of the set $E(W, \bar{W})$ of all the edges connecting vertices in W with vertices in $\bar{W}$; if we consider edge weights, the capacity of the cut is the sum of the edge weights of $E(W, \bar{W})$. We call $(W, \bar{W})$ an st-cut if s and t are in different sets of the cut. A *minimum st-cut* is an st-cut of minimum capacity. We denote the capacity of a minimum st-cut in H with $\text{mincut}_{s,t}(H)$.

For a planar embedding Γ of $H + \{s, t\}$, we define the *traversing costs* of Γ with respect to $\{s, t\}$ to be the shortest path in the dual graph of Γ that connects the two faces adjacent to $\{s, t\}$ without using the dual edge of $\{s, t\}$. We call the corresponding list of primal edges a *traversing path* for s and t. Gutwenger et al. [77] showed that the traversing costs are independent of the choice of the embedding Γ of H. Hence, we define the *traversing costs* of H with respect to $\{s, t\}$ to be the traversing costs of an arbitrary embedding Γ with respect to $\{s, t\}$. It is easy to see that a traversing path defines an st-cut.

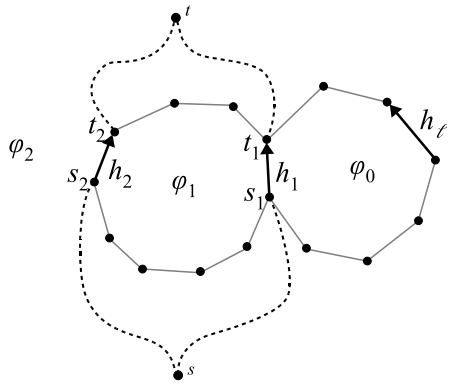


Figure 4.2: Proof of Lemma 4.24: selecting the edges $h_1, h_2, \ldots, h_\ell$.

Lemma 4.23. *Let H be a planar st-component, and let Γ be an embedding of $H + \{s,t\}$. If $e_1, \ldots, e_k$ is a traversing path of Γ with respect to $\{s,t\}$, then there exists an st-cut $(W, \bar{W})$ in H with $E(W, \bar{W}) = \{e_1, \ldots, e_k\}$.*

Proof. By the definition of a traversing path, we can draw a Jordan curve in a drawing realizing Γ that crosses exactly the edges $e_1, \ldots, e_k$ and divides the plane into two regions: one region R_s containing s and one region R_t containing t. Let W be the set of vertices in R_s and $\bar{W}$ be the set of vertices in R_t. Then, every edge in $E' := \{e_1, \ldots, e_k\}$ connects a vertex in W with a vertex in $\bar{W}$, and there is no edge in $E \setminus E'$ that connects a vertex in W with a vertex in $\bar{W}$. Hence, $E' = E(W, \bar{W})$ and $(W, \bar{W})$ is an st-cut. □

The following lemma shows that this st-cut is even a minimum st-cut.

Lemma 4.24. *Let H be a planar st-component. Then, the traversing costs of H with respect to $\{s,t\}$ are equal to* $\text{mincut}_{s,t}(H)$.

Proof. Let λ be the capacity of a minimum st-cut in H, and κ the traversing costs of H with respect to $\{s,t\}$. By Lemma 4.23, we have $\lambda \leq \kappa$. We have to show that $\kappa \leq \lambda$. Let Γ be an arbitrary embedding of $H' := H + \{s,t\}$ and let Γ^* be the corresponding dual graph. Let $(W, \bar{W})$ be a minimum cut with $s \in W$ and $t \in \bar{W}$. Since H is connected and the cut $(W, \bar{W})$ is minimal, removing the edges $E(W, \bar{W})$ splits H into two connected graphs $H_s := H[W]$ and $H_t := H[\bar{W}]$. We can write the edges in $E(W, \bar{W})$ as $e_1 = \{s_1, t_1\}, \ldots, e_\lambda = \{s_\lambda, t_\lambda\}$ such that $s_i \in W$ and $t_i \in \bar{W}$ for $1 \leq i \leq \lambda$. Moreover, there is a path from s to s_i in H_s and from t_i to t in H_t for every $1 \leq i \leq \lambda$.

We show that the dual edges of (possibly a subset of) $e_1, \ldots, e_\lambda$, and $\{s,t\}$ form a cycle $\varphi_0, \langle h_1^*, \varphi_1, h_2^*, \ldots, h_\ell^*, \varphi_\ell \rangle$ in Γ^*. Here, h_i^* denotes the dual of the edge $h_i = e_j$ (for some j), and φ_i denotes a face (i.e. a vertex in Γ^*). Obviously, $\{s,t\}$ is one of the edges h_i. This implies that removing the edges

$h_1, \ldots, h_\ell$ splits H into two parts which must be H_s and H_t. It follows that $\kappa \leq \ell - 1 \leq \lambda$ and the lemma holds.

We start our construction with $h_1 := \{s_1, t_1\}$. Let φ_0 be the face right of h_1 and φ_1 the face left of h_1; compare Figure 4.2. Since H' is 2-connected, h_1 is not a bridge[1] and hence $\varphi_0 \neq \varphi_1$. Since φ_1 is also a cycle in H' and the cut separates s_1 and t_1, there must be another edge $h_2 := \{s_2, t_2\}$ in $E(W, \bar{W})$ which is on φ_1. Let φ_2 be the face right of h_2. We distinguish two cases. If φ_2 is one of the faces φ_0, φ_1, then we have found a cycle in Γ^* and we are done. Otherwise, there must be an edge $h_3 \in E(W, \bar{W})$ with $h_3 \neq h_2$ and h_3 is on φ_2, since φ_2 is a cycle. We can continue this construction until we end up with an edge h_ℓ such that the left face of h_ℓ is one of the faces $\varphi_0, \ldots, \varphi_{\ell-1}$. The construction terminates, since $E(W, \bar{W})$ is an st-cut. □

Based on the above definitions and lemmata we define the non-planar core as follows:

Definition 4.25 (Non-planar Core)**.** The *non-planar core* $\text{core}(G) = (\mathcal{C}, w)$ of G is a graph $\mathcal{C}$ with integer edge weights w such that $\mathcal{C}$ is a copy of G in which each maximal planar st-component H of G is substituted with a *merge edge* $e_H = \{s, t\}$ with weight $w(e_H) = \text{mincut}_{s,t}(H)$, and each non-merge edge e has weight $w(e) = 1$.

By Lemma 4.21, we get:

Corollary 4.26. *The non-planar core of a graph G is well-defined and unique.*

4.2.4 Computing the Non-planar Core

In the following, we will compute the non-planar core using SPR-trees; hence the structure of the core's SPR-tree is of main concern:

Observation 4.27. *Let $\mathcal{S}$ be the SPR-tree of the non-planar core $(\mathcal{C}, w)$ of G. Each leaf of $\mathcal{S}$ is an R-node with non-planar skeleton.*

Proof. Since G, and therefore $(\mathcal{C}, w)$, is non-planar, $\mathcal{S}$ must contain a node with non-planar skeleton. Suppose $\nu \in \mathcal{S}$ is a leaf whose skeleton is planar. Since $\mathcal{S}$ contains at least one further node, ν has exactly one adjacent node μ in $\mathcal{S}$. But then the expansion graph of the virtual edge of ν in G_μ is planar and constitutes a planar 2-component. This is a contradiction to the definition of the non-planar core. It follows that every leaf of $\mathcal{S}$ is a node with non-planar skeleton. This must be an R-node, since only R-node skeletons can be non-planar. □

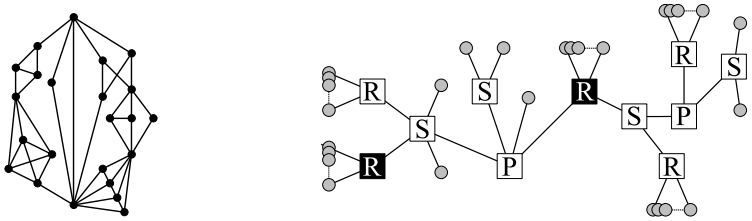
We give the pseudo code for computing the non-planar core $(\mathcal{C}, w)$ of a non-planar graph G in Algorithm 1; Figure 4.3 gives an example of the performed

[1] A *bridge* is an edge whose removal would increase the number of connected components in a graph.

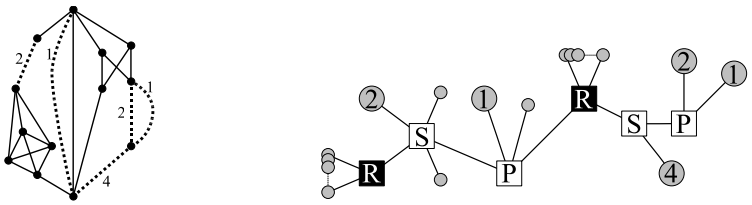
Algorithm 1 Computation of the non-planar core.

Require: 2-connected, non-planar graph $G = (V, E)$
Ensure: non-planar core $(\mathcal{C}, w)$ of G

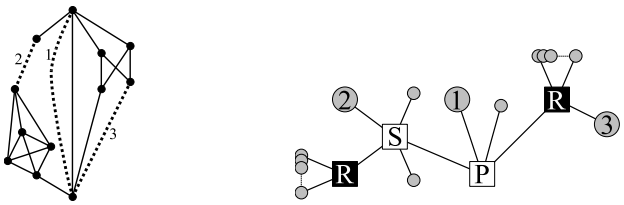
Let $\mathcal{T}$ be the SPR-tree of G
$\mathcal{C} := G$
$\mathcal{S} := \mathcal{T}$
for all uninspected leaves ν of $\mathcal{S}$ **do**
    if G_ν is planar **then**
        Let μ be the adjacent node of ν in $\mathcal{S}$.
        Let e_ν be the virtual edge in $G_\mu^{\mathcal{S}}$ representing ν.
        Remove ν from $\mathcal{S}$: Substitute the subgraph represented by $G_\nu^{\mathcal{S}}$ by the single merge edge e_ν in $\mathcal{C}$.
    end if
end for
for all $\nu \in \mathcal{S}$ **do**
    if ν is a P-node with multiple non-virtual edges F in $G_\nu^{\mathcal{S}}$ **then**
        In $\mathcal{C}$, merge the edges F into a single merge edge.
    else if ν is S-node **then**
        for all $M :=$ edges of a maximal path of non-virtual edges in $G_\nu^{\mathcal{S}}$ **do**
            Let s and t be the end nodes of the path of F.
            In $\mathcal{C}$, merge the edges F into a single merge edge $\{s, t\}$.
        end for
    end if
end for
for all edges $e \in \mathcal{C}$ **do**
    if e is a merge edge **then**
        Let $\mathcal{T}_e \subset \mathcal{T}$ be the subtree which is represented by the merge edge e, i.e., the pruned subtree that was merged into e.
        Let G_e be the graph corresponding to $\mathcal{T}_e$.
        $w(e) :=$ traversing costs of G_e with respect to e
    else
        $w(e) := 1$
    end if
end for



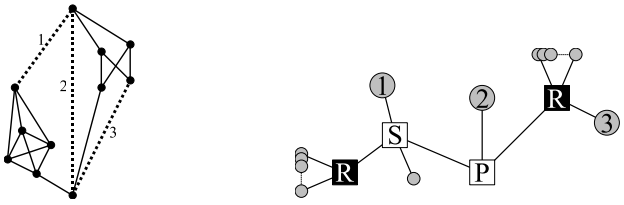
(a) Original graph G and its SPR-tree $\mathcal{T}$.



(b) Graph and SPR-tree after pruning original leaves.



(c) Graph and SPR-tree after pruning all leaves.



(d) The non-planar-core $(\mathcal{C}, w)$ and its SPR-tree $\mathcal{S}$ obtained after merging at S- and P-nodes.

Figure 4.3: Example of non-planar core reduction. In the SPR-trees, the small circles represent original edges; merge edges are denoted by larger circles including their weight; non-planar R-nodes are marked black. For the ease of understanding, the weights are updated during the reduction, instead of afterwards, as it is done by the algorithm.

reduction steps. The algorithm's basic idea is as follows: We start with the original graph G and its SPR-tree $\mathcal{T}$ (line 1–3), and transform both stepwise into the resulting core and its SPR-tree. Thereby we introduce an additional edge type—the *merge edge*—in addition to the original and virtual edges. Such a merge edge is non-virtual in the context of the core's SPR-tree $\mathcal{S}$ and behaves like an original edge. It arises from merging subgraphs into a single edge. Hence, a merge edge is virtual in the context of the original SPR-tree $\mathcal{T}$ of G.

In the first loop (line 4–10), we prune the leaves of $\mathcal{S}$ as long as they represent planar 2-components (i.e., S- and P-nodes, and planar R-nodes). For each such leaf ν, the corresponding planar st-component is replaced by a merge edge $e_\nu = \{s, t\}$ in $\mathcal{C}$. Thereby, the SPR-tree shrinks by one node (ν). Each pruning might introduce a new leaf which formerly was an inner node of $\mathcal{S}$. By holding the initial and the newly introduced leaves in a queue, from which we iteratively extract the next uninspected leaf, this loop can be performed in linear time.

The second loop (line 11–20) looks for remaining planar 2-components in H: these may consist of multiple non-virtual edges within a common P-node of $\mathcal{S}$ (line 12–13), or they form a chain of non-virtual edges within an S-node of $\mathcal{S}$ (line 14–18).

Finally (line 21–29), for each merge edge e in the core graph we can compute the traversing costs of the fully expanded component which was (probably recursively) merged into e.

To prove the correctness of algorithm, we first need the following simple observations:

Observation 4.28. *Let H be a planar 2-component of G. Then, H remains a planar 2-component*

1. *after replacing a planar st-component contained in H by an edge $\{s, t\}$; and*
2. *after replacing an edge $\{s, t\}$ in H by a planar st-component.*

Observation 4.29. *After the first loop of Algorithm 1 (i.e., after line 10), all leaves of $\mathcal{S}$ are non-planar R-nodes.*

This is evident, since the loop is designed to prune all leaves as long as they are planar. In the following lemmata we can now show that we only reduce the core by non-trivial planar 2-components, and that in the end, the remaining core does not contain any non-trivial planar 2-component anymore.

Lemma 4.30. *Algorithm 1 substitutes only non-trivial planar 2-components with merge edges.*

Proof. This follows directly from the algorithm and its description above. In the initial SPR-tree, each original edge is a trivial planar 2-component. We

show that it is an invariant of the algorithm that non-virtual edges represent planar 2-components.

All modifications performed by the algorithm replace some substructure with a merge edge in the core; furthermore, it is clear that the modifications are such that $\mathcal{C}$ and $\mathcal{S}$ are always consistent with each other. We modify $\mathcal{C}$ and $\mathcal{S}$ on three distinct places in the pseudo code:

Loop 1 – Pruning of $\mathcal{S}$. We prune the node ν; since it is a leaf in $\mathcal{S}$, we know that all but one edge, say e, of its skeleton are non-virtual edges. Clearly, μ is the node representing e. Let $e_\nu = \{s, t\}$ be the virtual edge of ν in $G_\mu^{\mathcal{S}}$. If $G_\nu^{\mathcal{S}}$ is planar, we could clearly substitute it for e_ν in $G_\mu^{\mathcal{S}}$, where it would be a planar st-component. Since μ represents the whole graph, ν represents a non-trivial planar 2-component in G. Hence it can be substituted with a merge edge, representing the planar 2-component.

Loop 2 – Merging at P-nodes. Clearly, a multi-set of edges, all incident to two distinct vertices s and t, form a planar st-component. The algorithm inspects all P-nodes to find such sets. It merges them into a single new merge edge, which represents a non-trivial planar st-component.

Loop 2 – Merging at S-nodes. Clearly, a chain of edges connecting two vertices s and t, form a planar st-component. The algorithm inspects all S-nodes to find such chains. It merges all consecutive non-virtual edges into a single new merge edge, which represents a non-trivial planar st-component. □

Lemma 4.31. *Algorithm 1 substitutes all maximal planar 2-components by merge edges.*

Proof. Assume that there exists a non-trivial planar 2-component H in $\mathcal{C}$. Let s and t be the contact points of H. Let $H' := H + \{s, t\}$ be the component augmented by the edge $\{s, t\}$. We distinguish between three cases:

Case 1: *H' forms the complete $G_\nu^{\mathcal{S}}$ of some node $\nu \in \mathcal{S}$.* Since all leaves of $\mathcal{S}$ are non-planar, ν has to be an inner node, and is therefore adjacent to at least two other nodes in the SPR-tree. Furthermore, there are at least two disjoint paths in $\mathcal{S}$ leading from ν to two distinct non-planar leaves μ^* and η^*. Let μ and η be the SPR-tree nodes adjacent to ν on these paths, respectively, and let e_μ and e_η be the virtual edges in $G_\nu^{\mathcal{S}}$ representing μ and η, respectively. Vice versa, let f_μ and f_η be the virtual edges in $G_\mu^{\mathcal{S}}$ and $G_\eta^{\mathcal{S}}$ corresponding to ν, respectively. Hence in $G_\mu^{\mathcal{S}}$, the component represented by f_μ is non-planar due to the non-planarity of η^*; analogously in $G_\eta^{\mathcal{S}}$, the component represented by f_η is non-planar due to the non-planarity of μ^*. In all other nodes adjacent to ν, ν is non-planar due to μ^* and η^*. Hence H' cannot resemble a planar st-component in G.

Case 2: *H' is a subgraph of $G_\nu^{\mathcal{S}}$ of some node ν in $\mathcal{S}$.* Let H be chosen in such a way that it is the smallest among all non-trivial planar 2-components in $\mathcal{C}$. Since H' is connected to the rest of the graph only via the contact points s and t, we know that H' cannot be 3-connected; it would resemble a complete R-node (Case (1)). Hence, and due to our minimality assumption, H can only be either a chain of (two) edges, connecting s and t, or a set of (two) edges, each incident to s and t. Since the algorithm merges all such constructs in the second loop, H cannot exist.

Case 3: *H' is contained in multiple skeleton graphs of $\mathcal{S}$.* Due to the maximality property of the SPR-tree nodes, there has to exist a non-trivial subcomponent H_0 of H, for which $H_0 + \{s, t\}$ is completely contained within a single tree node ν. Its nonexistence has been shown above. □

Theorem 4.32. *Let $G = (V, E)$ be a 2-connected graph. The non-planar core of G and the corresponding edge weights can be computed in $\mathcal{O}(|V|+|E|)$ time.*

Proof. The correctness of Algorithm 1 is given by Lemma 4.30, Lemma 4.31, and the fact that the edge weights are computed correctly. We achieve linear running time, since constructing an SPR-tree, testing planarity, and computing traversing costs [77] takes only linear time. □

4.2.5 Non-planar Core and Crossing Number

We show that the crossing number of G is exactly the weighted crossing number of the non-planar core $(\mathcal{C}, w)$ of G. Therefore we start with the following lemma. It allows us to restrict the crossings in which the edges of a planar 2-component may be involved so that we can still obtain a crossing minimal drawing of G. A similar result has been reported by Širáň in [141]. However, as pointed out in [31], the proof given by Širáň is not correct.

Lemma 4.33. *Let $H = (V_H, E_H)$ be a planar st-component of $G = (V, E)$. There exists a crossing minimal drawing $\mathcal{D}^*$ of G such that the induced drawing $\mathcal{D}_H^*$ of H has the following properties:*

1. *$\mathcal{D}_H^*$ contains no crossings;*
2. *s and t lie in a common face φ_{st} of $\mathcal{D}_H^*$;*
3. *all vertices in $V \setminus V_H$ are drawn in the region of $\mathcal{D}^*$ defined by φ_{st}; and*
4. *there exists a set $F \subseteq E_H$ with $|F| = \mathrm{mincut}_{s,t}(H)$ such that any edge $e \in E \setminus E_H$ may only cross through all edges of F, or through none of E_H.*

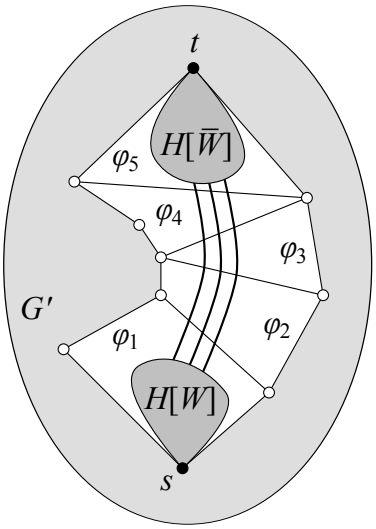


Figure 4.4: Proof of Lemma 4.33: final drawing $\mathcal{D}^*$ of G where $p = \langle \varphi_1, \varphi_2, \ldots, \varphi_5 \rangle$ is the shortest path in $\Gamma^*_{P'}$.

Proof. Let $G' := G[E \setminus E_H]$ be the graph that results from cutting H out of G. Let $\mathcal{D}$ be an arbitrary, crossing minimal drawing of G, and let $\mathcal{D}_H$ and $\mathcal{D}'$ be the induced drawing of H and G', respectively. We denote with P' the planarized representation of G' induced by $\mathcal{D}'$, i.e., the planar graph obtained from $\mathcal{D}'$ by replacing edge crossings with dummy vertices. Let $\Gamma_{P'}$ be the corresponding embedding of P and $\Gamma^*_{P'}$ the dual graph of $\Gamma_{P'}$.

Let $\langle \varphi_1, \ldots, \varphi_{k+1} \rangle$ be the faces (dual nodes) of the shortest path p in $\Gamma^*_{P'}$ that connects an adjacent face of s with an adjacent face of t. There are $\lambda := \text{mincut}_{s,t}(H)$ edge disjoint paths from s to t in H. Each of these λ paths crosses at least k edges of G' in the drawing $\mathcal{D}$. Hence, there are at least $\lambda \cdot k$ crossings between edges in H and edges in G'. We denote with E_p the set of primal edges of the edges of p. Let $\mathcal{D}^*_H$ be a planar drawing of H in which s and t lie in the same face φ_{st}, and let F be the edges in a traversing path in $\mathcal{D}^*_H$ with respect to s and t. By Lemma 4.24, there is a minimum st-cut $(W, \bar{W})$ with $E(W, \bar{W}) = F$, and thus $|F| = \lambda$. We can combine $\mathcal{D}'$ and $\mathcal{D}^*_H$ by placing the drawing of $H[W]$ in face φ_1 and the drawing of $H[\bar{W}]$ in φ_{k+1}, such that all the edges in E_p cross all the edges in F; see Figure 4.4. It is easy to verify that the conditions *1.–4.* hold for the resulting drawing $\mathcal{D}^*$. □

The following theorem shows that it is sufficient to compute the crossing number of the non-planar core.

Theorem 4.34. *Let G be a 2-connected graph, and let $(\mathcal{C}, w)$ be its non-planar core. Then,*

$$\text{cr}(G) = \text{cr}(\mathcal{C}, w).$$

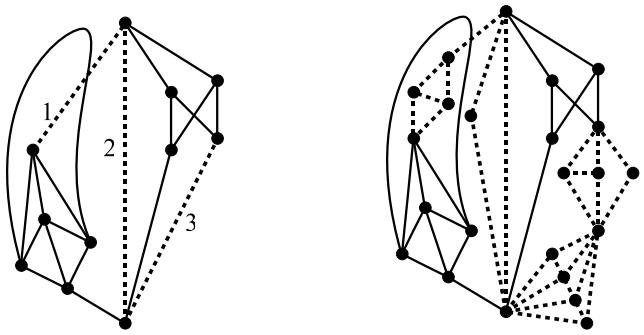


Figure 4.5: Application of the core reduction to the crossing number.

Proof. "$\leq$" Let $\mathcal{D}_\mathcal{C}$ be a drawing of $\mathcal{C}$ with minimum crossing weight. For each merge edge $e = \{s, t\} \in \mathcal{C}$, we replace e by a planar drawing $\mathcal{D}_e$ of the corresponding planar st-component so that all edges that cross e in $\mathcal{D}_\mathcal{C}$ cross the edges in a traversing path in $\mathcal{D}_e$ with respect to $\{s, t\}$. Since $w(e)$ is equal to the traversing costs of $\mathcal{D}_e$ with respect to $\{s, t\}$ by definition, replacing all merge edges in this way leads to a drawing of G with $\mathrm{cr}(\mathcal{C}, w)$ crossings, and hence $\mathrm{cr}(G) \leq \mathrm{cr}(\mathcal{C}, w)$.

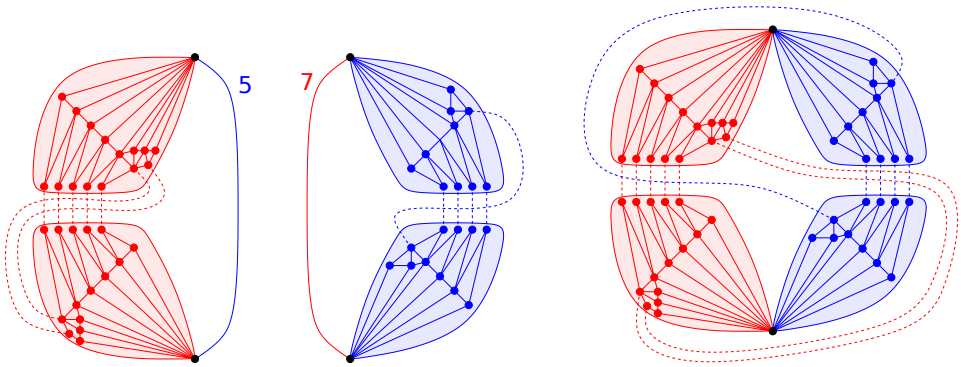
"$\geq$" On the other hand, let $\mathcal{D}$ be a crossing minimal drawing of G. For each merge edge $e = \{s, t\} \in \mathcal{C}$, we modify $\mathcal{D}$ in the following way. Let H be the planar st-component corresponding to e, and let G' be the rest of the graph. By Lemma 4.33, we obtain another crossing minimal drawing of G if we replace the drawing of H with a planar drawing $\mathcal{D}_H$ of H such that all edges of G' that cross edges in H will cross the edges in $E(W, \bar{W})$, where $(W, \bar{W})$ is a minimum st-cut in H. If we replace $\mathcal{D}_H$ with an edge $e = \{s, t\}$ with weight $w(e) := |E(W, \bar{W})| = \mathrm{mincut}_{s,t}(H)$, we obtain a drawing with the same crossing weight.

By replacing all merge edges in that way, we obtain a drawing of $\mathcal{C}$ whose crossing weight is the crossing number of G. It follows that $\mathrm{cr}(G) \geq \mathrm{cr}(\mathcal{C}, w)$, and hence the theorem holds. □

Figure 4.5 continues the example introduced in Figure 4.3; it shows a crossing minimal drawing of the core and the induced crossing minimal drawing of the original graph.

Further Reduction. It is a straight-forward idea to try to restrain the computation of the crossing number to the skeletons of non-planar R-nodes individually. To do this, it would be necessary to be able to merge two components with the following properties:

1. Both components have exactly two vertices, say s and t, in common.



(a) Crossing minimal (component-wise): 10 and 4.

(b) Crossing minimal (as a whole): 13.

Figure 4.6: Incorrectness of calculating crossing number for non-planar skeletons separately.

2. Each component is—if augmented with a virtual edge $\{s, t\}$—non-planar and 3-connected.

3. The crossing number of the merged component is the sum of the crossing numbers of the components.

Observation 4.35. *Let $\mathcal{R}$ be the set of non-planar R-nodes of the SPR-tree of some core graph $(\mathcal{C}, w)$. For $\nu \in \mathcal{R}$, let G_ν be the skeleton of ν and c_ν the corresponding weight function, where each virtual edge has the weight of the minimum st-cut through the represented component. In general,*

$$\mathrm{cr}(G) \neq \sum_{\nu \in \mathcal{R}} \mathrm{cr}(G_\nu, w_\nu).$$

Proof. Figure 4.6(a) shows an example and thus this decomposition approach fails. We see two components and their crossing minimal embedding, with regards to the minimum st-cut of their counterpart that defines the weight of the virtual edge. The two components have unique minimum st-cuts, denoted by dashed lines.

The minimum st-cut of the left component is 7, and the minimum st-cut of the right one is 5. The minimum crossing numbers of the left and right components are 10 and 4, respectively, summing up to 14. The minimum crossing number of the merged result is only $2 \cdot 4 + 5 = 13$ (Figure 4.6(b)). The reason is that we have edges that partially cross through the counterpart component. □

4.3 Digression: Non-Planar Core for Other Non-planarity Measures

The non-planar core reduction cannot only be applied with respect to the traditional crossing number. In the following section, we will show that the core or similar concepts also constitute a valid preprocessing for other non-planarity measures, which are not directly related to the crossing number concept.

4.3.1 Skewness

A well-known NP-complete problem in the context of non-planar graphs is the *Maximum Planar Subgraph* (MPS) problem. Thereby we are given a non-planar graph and ask for a planar subgraph with the largest edge cardinality possible. For weighted graphs, we analogously ask for a planar subgraph with largest total weight.

This optimization problem can be turned into a decision problem by asking if there is a planar subgraph with k edges, for some parameter k. Anyhow, we can observe that we can ask the question also the other way around: Given a non-planar graph G, can we obtain a planar graph by deleting no more than k' edges? The smallest such k' for which we can obtain such a planar subgraph is called the *skewness* of G, and is denoted by $\text{skew}(G)$. Clearly, by solving the MPS problem, we obtain the skewness of the graph, and vice versa.

Probably the first mention of skewness is by Kotzig [100], where he proved formulas to directly compute the skewness of special graph classes like complete and complete bipartite graphs. Since then, there has been research on the skewness/MPS problem, partly for special graph classes, reaching from heuristics, over approximation algorithms, to exact ILP approaches [92, 93, 113].

Interestingly, the non-planar core $(\mathcal{C}, w)$ has the property of preserving the skewness of the original graph, i.e.,

Theorem 4.36. *Let G be a 2-connected graph, and let $(\mathcal{C}, w)$ be its non-planar core. Then,*

$$\text{skew}(G) = \text{skew}(\mathcal{C}, w).$$

In order to prove this theorem we will first show the following lemma that establishes our main argument.

Lemma 4.37. *Let $G = (V, E)$ be a non-planar, 2-connected graph, $H = (V_H, E_H)$ a planar st-component of G, and $P = (V, E_P)$ a maximum planar subgraph of G. Then, either $H \subseteq P$, or $|E_H \backslash E_P| = \text{mincut}_{s,t}(H)$.*

Proof. We distinguish two cases.

Case 1: *There is a path from s to t in P which consists only of edges of H.* Consider an embedding Γ of P. If we cut out H from Γ, then s and t lie on a common face of the resulting embedding Γ'. On the other hand, we can construct an embedding Γ_H of H in which s and t lie on the external face. Inserting Γ_H into Γ' yields a planar embedding of $P \cup H$. Since P is a maximum planar subgraph of G and $H \subseteq G$, it follows that $H \subseteq P$.

Case 2: *There is no such path from s to t.* It follows that $H' = (V_H, E_H \cap E_P)$, the subcomponent of H contained in P, has at least two connected components, one containing s, and the other one containing t. Hence, it contains at most $|E_H| - \text{mincut}_{s,t}(H)$ edges, which implies $\text{mincut}_{s,t}(H) \leq |E_H| - |E_H \cap E_P| = |E_H \backslash E_P|$.

On the other hand, we can construct an embedding of H with s and t on the external face, and remove the $\text{mincut}_{s,t}(H)$ edges in a traversing path of H with respect to $\{s,t\}$. This yields an embedding Γ with two connected components H_s and H_t with $s \in H_s$ and $t \in H_t$. Let $G' = G[E \setminus E_H]$ be the rest of the graph. Since H_s has only s in common with G' and H_t has only t in common with G', we can insert Γ into any embedding of $G' \cap P$ while preserving planarity. This implies that $|E_H \backslash E_P| \leq \text{mincut}_{s,t}(H)$ and the lemma holds. □

Using this lemma, we can show that the non-planar core reduction is invariant with respect to skewness.

Proof of Theorem 4.36. Let $G = (V, E)$ and $\mathcal{C} = (V_\mathcal{C}, E_\mathcal{C})$.

"$\leq$" Let $P_\mathcal{C} = (V_\mathcal{C}, E_P)$ be a maximum weight planar subgraph of $\mathcal{C}$, and let D be a drawing of $P_\mathcal{C}$. We have $\text{skew}(\mathcal{C}, w) = w(E_\mathcal{C}) - w(E_P)$. We show that we can construct a planar subgraph $P' = (V, E')$ of G with $|E| - |E'| = \text{skew}(\mathcal{C}, w)$.

We consider a maximal planar st-component H of G. Let $e = \{s,t\}$ be the corresponding merge edge in $\mathcal{C}$, and let $\mathcal{D}_H$ be a planar drawing of H in which both s and t lie in the external face. If e is in $P_\mathcal{C}$, we can replace e with the drawing $\mathcal{D}_H$ and the resulting drawing remains planar. If e is not in $P_\mathcal{C}$, we remove the edges of a traversing path of H with respect to $\{s,t\}$ from $\mathcal{D}_H$. This yields a drawing $\mathcal{D}'_H$ with two connected components, one containing s, and the other one containing t. Obviously, we can add the drawing $\mathcal{D}'_H$ to $\mathcal{D}$ while preserving planarity, and we removed exactly $w(e) = \text{mincut}_{s,t}(H)$ edges from G.

We perform this operation for each merge edge and finally end up with a drawing of a planar subgraph $P' = (V, E')$ of G with $|E| - |E'| = \text{skew}(\mathcal{C}, w)$.

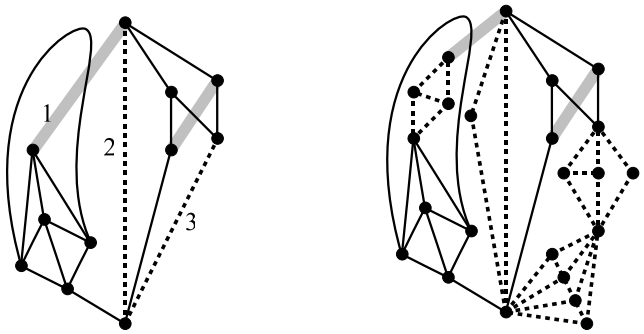


Figure 4.7: Application of the core reduction to the skewness.

"≥" Let $P = (V, E_P)$ be a maximum planar subgraph of G. We have $\text{skew}(G) = |E| - |E_P|$. We show that we can construct a planar subgraph $P_{\mathcal{C}} = (V_{\mathcal{C}}, E')$ of $\mathcal{C}$ with $w(E_{\mathcal{C}}) - w(E') = \text{skew}(G)$.

Consider a maximal planar st-component H of G. By Lemma 4.37, we know that either H is completely contained in P, or exactly $\text{mincut}_{s,t}(H)$ many edges of H are not in E_P. In the first case, we know that an st-path is in P, and hence replacing H by the corresponding edge $\{s, t\}$ preserves planarity. In the second case, the corresponding merge edge $e = \{s, t\}$ with weight $w(e) = \text{mincut}_{s,t}(H)$ will not be in $P_{\mathcal{C}}$.

Constructing $P_{\mathcal{C}}$ by performing this operation for each maximal planar 2-component obviously yields a planar subgraph $(V_{\mathcal{C}}, E')$ of $\mathcal{C}$ with $w(E_{\mathcal{C}} \setminus E') = \text{skew}(G)$. □

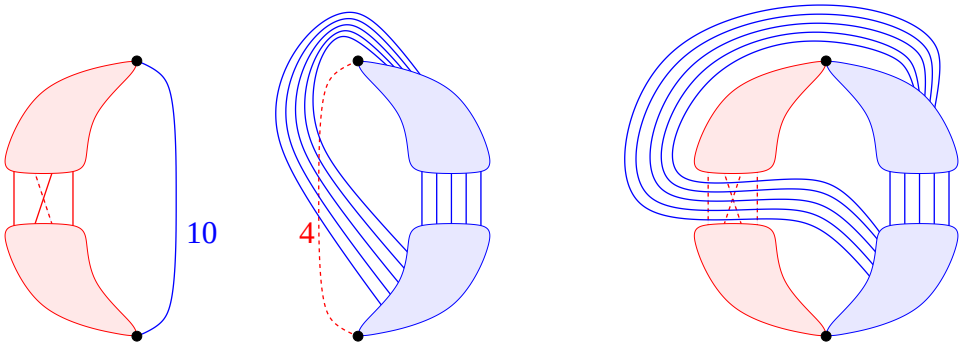
Figure 4.7 illustrates the reduction for our example graph by showing an optimal solution for the core on the left, and the induced optimal solution for the original graph on the right. Removing the two thick gray edges—each with weight 1—leads to a maximum planar subgraph.

Further Reduction. Analogously to Observation 4.35 regarding the crossing number, we can show that we cannot easily restrict ourselves to the non-planar triconnected components of a graph to compute its skewness:

Observation 4.38. *Let $\mathcal{R}$ be the set of non-planar R-nodes of the SPR-tree of some core graph $(\mathcal{C}, w)$. For $\nu \in \mathcal{R}$, let G_ν be the skeleton of ν and w_ν the corresponding weight function, where each virtual edge has the weight of the minimum st-cut through the represented component. In general,*

$$\text{skew}(G) \neq \sum_{\nu \in \mathcal{R}} \text{skew}(G_\nu, w_\nu).$$

Proof. Figure 4.8(a) shows two components including the virtual edges with the weights of their counterpart's minimum st-cut. The shaded regions denote



(a) Skewness (component-wise): 1 and 4.

(b) Skewness (as a whole): 4.

Figure 4.8: Incorrectness of calculating skewness for non-planar skeletons separately.

dense, crossing-free, 3-connected subgraphs, similar to the ones in Figure 4.6. The edges to be removed in order to get a planar subgraph are the drawn as dashed lines.

The skewness of the left component is 1 (the choice between the two possibilities is arbitrary). The skewness of the right component corresponds to removing its virtual edge, and therefore has the value of 4. We can see that we have one edge that has to be removed for both components, and is therefore counted twice. The merged drawing has a skewness of only 4, although the sum of the separate skewnesses would have suggested $1+4=5$. We cannot even find any set of edges which does not include the virtual edge, has the size 5, and can be removed in order to get a planar subgraph. □

4.3.2 Thickness

Definition 4.39 (Thickness)**.** The *thickness* $\text{thick}(G)$ of a graph G is the minimum number of planar subgraphs $G_1, G_2, \ldots$ such that their union gives the original graph, i.e., $\bigcup_{i=1}^{\text{thick}(G)} G_i = G$.

This definition was first given by Harary [83], asking for the thickness of K_9. The problem of identifying the thickness of a general graph was shown to be NP-complete, but there are formulas to directly compute the thickness of complete and complete bipartite graphs [3, 9, 10]. See [114] for a survey on this non-planarity measure.

For the thickness, there is no need to consider edge weights in the core graph. However, it is not correct to simply compute the thickness of the non-planar core as defined before:

Observation 4.40. *In general,* $\text{thick}(G) \neq \text{thick}(\text{core}(G))$.

Proof. Consider the graph obtained from K_9 by replacing each edge with a chain of two edges, denoted by $K_9^{[2]}$. Then, K_9 is the non-planar core of $K_9^{[2]}$. While $\text{thick}(K_9) = 3$, $K_9^{[2]}$ has only thickness 2: The first subgraph contains one edge of each chain, and the second subgraph contains the other edge of each chain. □

Therefore, we slightly modify $\mathcal{C}$. We call a planar st-component H *splittable* if it does not contain an edge $\{s,t\}$. We denote by $\text{core}^+(G)$ the graph obtained from $\mathcal{C}$ by splitting every merge edge corresponding to a splittable 2-component into a *2-chain*, i.e., a chain of two edges.

Theorem 4.41. *Let G be a 2-connected graph, and let $\mathcal{C}' = \text{core}^+(G)$. Then,*

$$\text{thick}(G) = \text{thick}(\mathcal{C}')$$

Proof. **"≤"** Let $\text{thick}(\mathcal{C}') = k$, and let $G_1, \ldots, G_k$ be k pairwise edge disjoint planar graphs with $G_1 \cup \ldots \cup G_k = \mathcal{C}'$. We consider a merge edge $e = \{s,t\}$ of the non-splitted non-planar core of G, and let $H = (V_H, E_H)$ be the corresponding planar st-component. If H is not splittable, then e is contained in $\mathcal{C}'$, and thus there is a subgraph, say G_i, containing e. We replace e in G_i by H. It is easy to see that the resulting graph remains planar.

Otherwise, H is splittable and e was split into two edges, say $e_1 = \{s,d\}$ and $e_2 = \{d,t\}$, in $\mathcal{C}'$. Let G_i be the graph containing e_1, and let G_j be the graph containing e_2. If $i = j$, then we replace e_1 and e_2 in G_i by H. Otherwise, we split H into two edge disjoint graphs H_1 and H_2 in the following way: Let E' be the set of edges incident with s. Then, H_1 is the graph induced by E', and H_2 is the graph induced by $E_H \setminus E'$. Since $\{s,t\}$ is not an edge in H, neither $t \in H_1$ nor $s \in H_2$. We replace e_1 by H_1 in G_i, and e_2 by H_2 in G_j. Each of these modifications preserves planarity.

Applying these modifications to all merge edges of $\mathcal{C}$ constructs k planar subgraphs of G whose union is G, and thus $\text{thick}(G) \leq k$.

"≥" Let $\text{thick}(G) = k$, and let $G_1, \ldots, G_k$ be k planar graphs with $G_1 \cup \ldots \cup G_k = G$. We consider a maximal planar st-component H. We distinguish two cases:

(i) If there is a graph G_i such that $G_i \cap H$ contains a path between s and t, then we remove all edges and vertices $\neq s,t$ of H from all graphs $G_1, \ldots, G_k$, and we add the edge $e = \{s,t\}$ to G_i. If H does not contain an edge $\{s,t\}$, then we also split e. Obviously, G_i is still planar after these modifications.

(ii) Otherwise, we know that $k \geq 2$, and therefore there are two graphs G_i and G_j with $i \neq j$. First, we remove all edges and vertices $\neq s, t$ of H from all graphs $G_1, \ldots, G_k$. Then, we add the edges $e_s = \{s, d\}$ to G_i and $e_t = \{d, t\}$ to G_j, where d is a new dummy vertex. If any of the end vertices of e_1 (resp. e_2) is not yet contained in G_i (resp. G_j), we also add this vertex. Since d has degree 1 in both G_i and G_j, the resulting graphs are planar.

Applying these modifications to all maximal planar 2-components constructs k planar graphs whose union is $\mathcal{C}'$, and thus $\text{thick}(G) \geq \text{thick}(\mathcal{C}')$. □

See Figure 4.7 again, for an example regarding the thickness. The two thick gray edges that have to be removed to retrieve a planar subgraph form the second subgraph, which leads to $\text{thick}(\mathcal{C}') = \text{thick}(G) = 2$.

Further Reduction. Having this modified core graph at hand, we can even show a much stronger reduction in the context of graph thickness. The central argument is:

Lemma 4.42. *Let $G = (V, E)$ be a 2-connected, non-planar graph, and let $H = (V_H, E_H)$ be a splittable maximal planar st-component of G. Then,*

$$\text{thick}(G) = \max(2, \text{thick}\left(G[E \setminus E_H]\right)).$$

Proof. Let $G' = G[E \setminus E_H]$. If G' is planar, we have $\text{thick}(G') = 1$ and the thickness of G is 2. Otherwise, let $k = \text{thick}(G') \geq 2$ and let $G_1, \ldots, G_k$ be planar subgraphs of G' such that $G_1 \cup \ldots \cup G_k = G'$. We may assume w.l.o.g. that s and t are both contained in G_1 and G_2. We have to show that there are k subgraphs of G whose union yields G.

Since $\{s, t\} \notin H$, we can split H into two edge induced subgraphs $H_1 = H[E']$ and $H_2 = H[E_H \setminus E']$ of H, such that E' consists of the edges incident to s in H. Then, we add H_1 to G_1 and H_2 to G_2. Since H_1 is only connected with s in G_1 and H_2 is only connected with t in G_2, both G_1 and G_2 remain planar. Thus, we have constructed k planar subgraphs of G whose union is G. □

Since we know that the thickness of a non-planar graph G is at least 2, we can simply remove splittable planar 2-components from the core of G. The resulting graph may contain new planar 2-components, or may even decompose into several blocks. We can treat the resulting blocks separately, and apply the reduction method again until no further reduction is possible.

Based on this observation, we define a new graph reduction as follows. We denote with $\beta(G)$ the set of non-planar blocks of G, and with $\text{core}^-(G)$ the non-planar core of G (without edge weights) in which every merge edge

corresponding to a splittable planar 2-component is removed. For a set of non-planar, 2-connected graphs $\mathcal{B}$, we define

$$\Upsilon(\mathcal{B}) := \bigcup_{B \in \mathcal{B}} \beta(\text{core}^{-}(B)).$$

$\Upsilon^{(n)}(\cdot)$ denotes the n-fold application of the function Υ. We obtain the following theorem.

Theorem 4.43. *Let G be a non-planar, 2-connected graph, and let $n \in \mathbb{N}$ be an arbitrary integer. Then,*

$$\text{thick}(G) = \text{thick}\left(\Upsilon^{(n)}\left(\{G\}\right)\right),$$

where $\text{thick}(\mathcal{B}) := \max_{B \in \mathcal{B}} \text{thick}(B)$ *if* $\mathcal{B} \neq \emptyset$*, and* $\text{thick}(\mathcal{B}) := 2$ *otherwise.*

In the example in Figure 4.3, we can remove the merge edges with weight 1 and 3, and obtain a planar graph. Hence, we directly have $\Upsilon^{(1)}(\{G\}) = \emptyset$ and $\text{thick}(G) = 2$.

4.3.3 Coarseness

Definition 4.44 (Coarseness)**.** The *coarseness* $\text{coarse}(G)$ of a graph is the maximum number of edge-disjoint non-planar subgraphs of G.

According to Harary [84], the origin of coarseness was a slip of tongue by Erdös when explaining the concept of graph thickness. Nevertheless, Beineke, Guy, and Chartrand started to study the problem and developed formulas and bounds for complete and complete bipartite graphs [7, 8, 78, 81]. From the point of algorithmics, not much is known about computing the coarseness of a graph. Its complexity status is unknown, and there is no published heuristic for the problem. An obvious approach for the latter is to iteratively extract a Kuratowski subdivision; each such extraction can be done in linear time [20, 60], cf. Section 6.1.2.

The definition of the coarseness of a graph can be reformulated in a rather obvious way:

Observation 4.45. *The coarseness of a graph G is the maximum number of edge-disjoint Kuratowski subdivisions in G.*

Thus, we can focus on edge-disjoint Kuratowski subdivisions. The following lemma shows that a planar 2-component can only be (part of) a Kuratowski path, but cannot include a Kuratowski node as an inner node. Hence it is sufficient to represent this path by an edge within the core graph.

Lemma 4.46. *Let H be a planar st-component of G, and let K be a Kuratowski subdivision in G. Then, $V(H) \setminus \{s, t\}$ contains no Kuratowski node of K.*

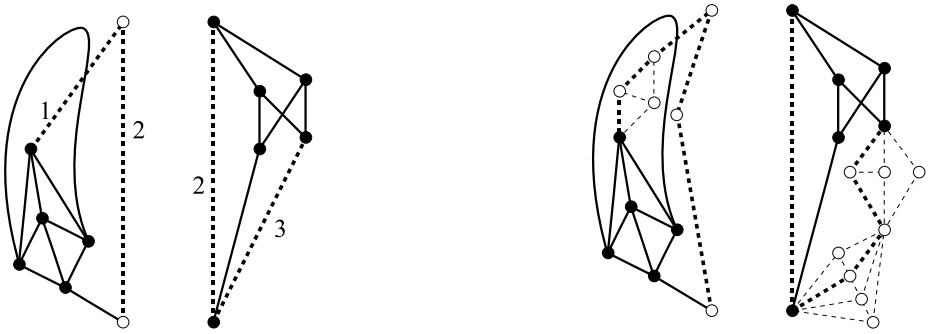


Figure 4.9: Application of the core reduction to the coarseness.

Proof. Let $H = (V_H, E_H)$ and $K = (V_K, E_K)$. Suppose H and K share a vertex $\neq s, t$. Then, H also contains some edges of K, i.e., $E_H \cap E_K \neq \emptyset$. Since H is planar and K contains no cut vertex, both s and t must be contained in K and K contains a vertex not contained in H. Hence, $\{s, t\}$ is a separation pair in K. Since K is a Kuratowski subdivision, the only possibility is that $\{s, t\}$ separates a subpath p (from s to t) from the rest of K. But then, only p can be in H, since $H + \{s, t\}$ is planar. It follows that H cannot contain a Kuratowski node of K. □

In the context of integer-weighted graphs, we have to extend the definition of coarseness as the maximum number of subgraphs (and equivalently Kuratowski subdivisions) of G such that each edge appears in at most $w(e)$ subgraphs. Using Observation 4.45 and Lemma 4.46, we can show that the non-planar core is also invariant with respect to coarseness.

Theorem 4.47. *Let G be a 2-connected graph, and let $(\mathcal{C}, w)$ be its non-planar core. Then,*

$$\text{coarse}(G) = \text{coarse}(\mathcal{C}, w).$$

Proof. **"≤"** Let $\text{coarse}(\mathcal{C}, w) = k$, and let $K_1, \ldots, K_k$ be k Kuratowski subdivisions in $\mathcal{C}$ such that each edge e' of $\mathcal{C}$ appears in at most $w(e')$ subdivisions.

Consider a merge edge $e = \{s, t\}$ in $\mathcal{C}$ and let H be the corresponding maximal planar st-component of G. By definition of the weighted coarseness, e appears in at most $w(e)$ of the k Kuratowski subdivisions. On the other hand, there are $w(e)$ edge disjoint st-paths $p_1, \ldots, p_{w(e)}$ in H, since $w(e) = \text{mincut}_{s,t}(H)$. Hence, we can replace each occurrence of e in $K_1, \ldots, K_k$ by an st-path p_i such that the resulting graphs remain edge-disjoint.

Applying these modifications to every merge edge of $\mathcal{C}$ results in k edge-disjoint Kuratowski subdivisions of G, and hence $\text{coarse}(G) \leq$

coarse$(\mathcal{C}, w)$.

"$\geq$" Let coarse$(G) = k$, and let $K_1, \ldots, K_k$ be k edge-disjoint Kuratowski subdivisions in G. Consider a planar st-component H in G. From Lemma 4.46, we know for each K_i that $E(H) \cap E(K_i)$ is either empty or an st-path in H. Since there are $w(e) = \text{mincut}_{s,t}(H)$ edge-disjoint st-paths in H, we need at most $w(e)$ copies of $\{s, t\}$ in order to replace all these st-paths by an edge $\{s, t\}$.

Applying these modifications to every maximal planar st-component results in k Kuratowski subdivisions in $\mathcal{C}$ such that each edge e' appears in at most $w(e')$ subdivisions, and hence coarse$(G) \geq$ coarse$(\mathcal{C}, w)$. □

See Figure 4.9 for the coarseness of our example graph. We find two edge-disjoint non-planar graphs, since the vertical virtual edge has weight 2, and can therefore be included into two distinct subgraphs. These subgraphs directly induce two subgraphs in the original graph, and coarse$(\mathcal{C}, w) =$ coarse$(G) = 2$.

Further Reduction. Also for the coarseness problem, we cannot simply restrict ourselves to the non-planar R-nodes:

Observation 4.48. *Let $\mathcal{R}$ be the set of non-planar R-nodes of the SPR-tree of some core graph $(\mathcal{C}, w)$. For $\nu \in \mathcal{R}$, let G_ν be the skeleton of ν and w_ν the corresponding weight function, where each virtual edge has the weight of the minimum st-cut through the represented component. In general,*

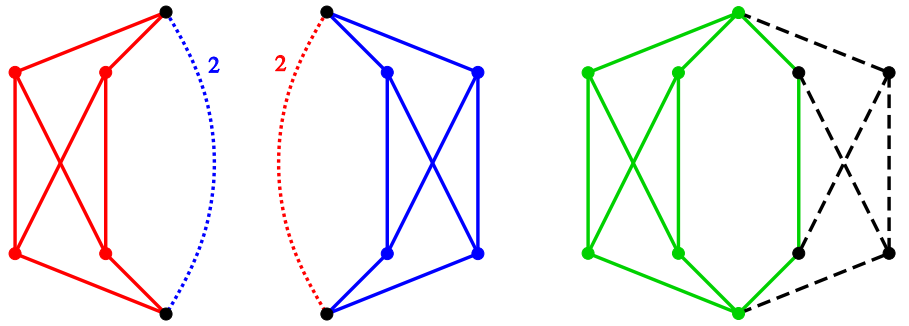
$$\text{coarse}(G) \neq \sum_{\nu \in \mathcal{R}} \text{coarse}(G_\nu, w_\nu).$$

Proof. For the coarseness problem, the counter-example is even simpler, based on two interweaved $K_{3,3}$s; see Figure 4.10. No matter whether we allow the use of the virtual edges within each component or not—leading to two times coarseness 1 or 0, respectively—we will never come up with the correct overall coarseness of 1. □

4.3.4 Genus

To define the *genus* of a graph, we first have to discuss some topological definitions of surfaces that we will recapitulate here briefly. In-depth discussions of this topic can be found in any book on topological graph theory; an introduction to the topic can be found, e.g., in [47].

Definition 4.49 (Surface)**.** A *surface* S is a compact connected 2-manifold without boundary. Intuitively, that is a topological space in which every point has a neighborhood homeomorphic to the Euclidean plane $\mathbb{R}^2$, and each point on S can be reached from any other point on S by moving along S.



(a) Coarseness (component-wise): 1 and 1. (b) Coarseness (as a whole): 1.

Figure 4.10: Incorrectness of calculating coarseness for non-planar skeletons separately.

All surfaces can be categorized regarding their homeomorphisms. Disregarding cross-caps, which introduce *unorientable* surfaces, we will only consider *orientable* surfaces in the following.

Proposition 4.50 (Genus of a surface)**.** *All oriented surfaces (up to homeomorphism) can be generated by adding a number of* handles *to a sphere. The number of handles is called* genus *of such a surface. The surface obtained by adding g handles is denoted by S^g.*

Adding a *handle* to a surface S is done as follows: First, we cut two circular discs out of S. Then we identify the borders of these circles with each other. Using the argument of homeomorphy, we can describe adding a handle as adding a "tube" connecting the two circular boundaries. As a third analogy, we can also interpret adding a handles as drilling a hole through the surface's body. Figure 4.11[2] shows example surfaces with genus 0–3.

Definition 4.51 (Genus of a graph)**.** The *genus* of a graph G—denoted by $\text{genus}(G)$—is the smallest genus g such that G can be drawn on S^g without any crossings.

Identifying the genus of a graph is NP-complete, as shown by Thomassen [138]. Even though the problem looks somehow dissimiliar to the other non-planarity measures, the non-planar core also preserves the genus of the original graph. Furthermore, since the solution of the genus problem will add handles to the surface, and not delete edges or allow crossings, we do not need the edge weights for computing the core's genus: Any handle always offers enough space to place any number of (planarly embedded) edges onto it.

[2]Image sources: first row `www.cims.nyu.edu/~steiner`, chocolate `www.magazinusa.com`, doughnut `www.dadcando.com`, baby cup `www.rotho-babydesign.com`, pretzel `pdc.wikipedia.org`.

Figure 4.11: Surfaces of different genus.

Theorem 4.52. *Let G be a 2-connected graph, and let $\mathcal{C}$ be its non-planar core. Then,*

$$\text{genus}(G) = \text{genus}(\mathcal{C}).$$

Proof. "$\leq$" Let $g = \text{genus}(\mathcal{C})$ and let $\mathcal{D}_\mathcal{C}$ be a crossing-free drawing of $\mathcal{C}$ on S^g. For each merge edge $e = \{s, t\} \in \mathcal{C}$, let $\mathcal{D}_e$ be a planar drawing of the corresponding planar st-component where s and t are on the outer face.

We replace each e by the corresponding drawing $\mathcal{D}_e$, within a small enough neighborhood of e. This can be done, as each point on the line representing e in $\mathcal{D}_\mathcal{C}$ has a neighborhood homeomorphic to the Euclidean plane. Furthermore, there exists a small enough neighborhood where the introduced drawing $\mathcal{D}_e$ does not interfere (i.e., cross) with any other parts of the drawing $\mathcal{D}_\mathcal{C}$. By these replacements, we obtain a crossing-free drawing of G on S^g.

"$\geq$" Let $g = \text{genus}(G)$ and let $\mathcal{D}$ be a crossing-free drawing of G on S^g. Clearly, each maximal planar 2-component H is drawn without any crossings. Considering the induced drawing $\mathcal{D}_H$ of the component, this drawing might not be embedded in the plane (S^0) but on some surface S^h with $0 < h \leq g$. Yet, let s and t be the connection points of H, and find any simple st-path p in H. We then replace H in $\mathcal{D}$ by the edge $e = \{s, t\}$ routed along p.

Clearly, e is the merge edge in $\mathcal{C}$ corresponding to H. Furthermore, e can now be embedded in S^0—the overall genus of the required surface will hence never increase. By these replacements, we obtain a crossing-free drawing of $\mathcal{C}$ on S^g. □

Chapter 5

0/1-ILPs for the Crossing Number Problem

crossing: *(Entertainment)*
A 2005 Canadian independent feature film by director Roger Evan Larry, starring Sebastian Spence, Crystal Buble, Bif Naked and Fred Ewanuick.

In this chapter, we present two competing practical 0/1-ILP formulations and discuss their similarities and differences. The order in this thesis corresponds to the chronological order of their development. While the newer formulation is stronger in practice (cf. Chapter 8), the older has certain advantages with regard to its adaptability to other crossing number schemes (cf. Chapter 7). Before we will discuss the two formulations, we will start with a theoretical 0/1-ILP: Though not practical by itself, it defines the crossing number polytope and forms the basis for both subsequent formulations. In the remainder of this chapter, we are always given a graph $G = (V, E)$ with edge weights w, for which we seek the crossing number.

For a description of how to solve the presented ILPs within a branch-and-bound framework, and in particular for the necessary separation routines and column generation schemes, see Chapter 6.

5.1 General Concepts

Let

$$\mathcal{CP} := \left\{ \{e, f\} \in E^{\{2\}} \ : e \cap f = \emptyset \right\}$$

be the set of all edge pairs which may cross in any optimal drawing of G; we know that adjacent edges will never cross. The central idea of all current

crossing minimization 0/1-ILPs is to introduce variables

$$x_{\{e,f\}} \in \{0,1\} \quad \forall \{e,f\} \in \mathcal{CP} \tag{5.1}$$

which are 1 if the edges e and f cross, and 0 otherwise. The objective function can then be written as:

$$\min \sum_{\{e,f\} \in \mathcal{CP}} w(e) \cdot w(f) \cdot x_{\{e,f\}},$$

i.e., we want to minimize the weight of the crossings, i.e., the crossing number if $w(e) = 1$ for all $e \in E$.

It remains to model the requirement that the realization of these variables induces a feasible—i.e., planar—planarization of the given graph. The problem is that if an edge is involved in multiple crossings, the realization of the assignment $\bar{x}$ for the variable vector x is ambiguous regarding the order of the crossings on that edge: Some realizations may introduce feasible planarizations, others may not. Formally, we can state the problem:

Definition 5.1 ((Strong) Realizability)**.** Given a graph $G = (V, E)$ and a set of edge pairs $\mathcal{R} \subseteq E^{\{2\}}$, does there exist a drawing of G with the crossings $\mathcal{R}$?

In our context, we are given a graph G and a 0/1 vector $\bar{x}$ as described above. Does $\bar{x}$ allow to a feasible planarization? This problem is called *realization problem* and was shown to be NP-complete by Kratochvíl [101]. Hence, even if we are given some presumably optimal solution $\bar{x}$, we can in general not even decide in polynomial time whether $\bar{x}$ is at all feasible, unless $P = NP$. If, however, every edge is involved in at most one crossing—or if the order of the crossings on each edge is specified—the realization problem is solvable in linear time: We replace each crossing by a dummy node and test for planarity of the resulting graph.

Let $\mathcal{F}$ be the set of feasible (binary) solution vectors for x, i.e., each $\bar{x} \in \mathcal{F}$ allows a feasible planarization. Let $\bar{\mathcal{F}} := \{0,1\}^{|\mathcal{CP}|} \setminus \mathcal{F}$ be the set of infeasible binary solution vectors. For any binary vector $\bar{x}$, let $k(\bar{x})$ ($k \in \{0,1\}$) be the set of edge pairs $\{e,f\}$ for which $\bar{x}_{\{e,f\}} = k$. We can theoretically write the exponential number of *realizability constraints*

$$\sum_{\{e,f\} \in 1(\bar{x})} x_{\{e,f\}} - \sum_{\{e,f\} \in 0(\bar{x})} x_{\{e,f\}} \leq \bar{x}\bar{x}^T \qquad \forall \bar{x} \in \bar{\mathcal{F}}. \tag{5.2}$$

Since $\bar{x}\bar{x}^T$ gives the number of ones in $\bar{x}$, this constraint requires that at least one x-variable is flipped either from 1 to 0 or vice versa, relative to the forbidden vector $\bar{x}$. Since the realization problem is NP-complete, we can (unless $P = NP$) in general not construct all these constraints without testing non-trivial $\bar{x}$ vectors in exponential time to decide whether they are in

$\bar{\mathcal{F}}$. This renders this *realizability-based exact crossing minimization* (RECM) formulation infeasible for practical use in general.

It is an open problem, whether there exists a practically relevant formulation using only the variables described above, while directly including the realizability subproblem.

Complete graphs. For complete graphs, the realization problem was recently shown to be solvable in polynomial time [105]. This would allow a branch-and-cut algorithm heuristically separating realizability constraints, based on rounding the solution and performing the realizability check, cf. Section 6.1. Due to the very specific nature of the single realizability constraints—they only forbid one specific infeasible solution vector, instead of a whole class of them—we will only obtain a very weak and slow algorithm that is infeasible for practical use.

The proof in [105] only gives a highly complex testing procedure, but not a description of forbidden minors or crossing configurations. The latter could potentially be used to construct stronger linear constraints for RECM, thus allowing a practically relevant algorithm to solve the crossing number problem for complete graphs.

Kuratowski constraints. Despite the above setbacks, we can describe a special variant of the realizability constraints that will form the basis of the planarity-establishing constraints in the subsequent formulations: A so-called *Kuratowski constraint*

$$\sum_{\{e,f\}\in\mathcal{CP}(K)} x_{\{e,f\}} \geq 1 \tag{5.3}$$

is conceptually bound to a specific Kuratowski subdivision K and requires at least one crossing between the edges of K. Thereby, $\mathcal{CP}(K) \subseteq E(K)^{\{2\}} \cap \mathcal{CP}$ are the edge pairs of the subdivision, where the two edges of a pair belong to non-adjacent Kuratowski paths.

We will revisit this formulation, the above Kuratowski constraints, and the polytope induced by RECM in Section 5.4. Therein we will discuss the central properties of these constraints, i.p., when they define facets in the crossing number polytope. The following two 0/1-ILPs are based on the ideas of RECM; their main difference to RECM and between each other is how they circumvent the problem of ambiguously ordered multiple crossings on edges.

5.2 Subdivision-based 0/1-ILP

As described above, the realization problem becomes easy when each edge is involved in at most one crossing. The smallest crossing number achievable with this restriction is known as the *simple crossing number* $\operatorname{scr}(G)$. Clearly

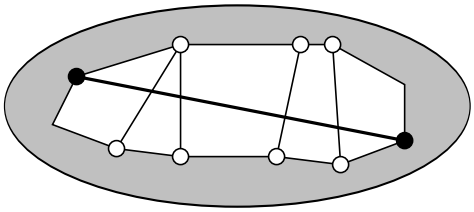


Figure 5.1: The optimal drawing of a graph might require all crossings to be placed on a single edge. The gray region denotes a dense, triconnected subgraph.

$\text{scr}(G) \geq \text{cr}(G)$, and there exist graphs where inequality holds, cf. Figure 5.1. It is even NP-complete to decide whether a given graph allows such a restricted drawing at all [71]. Hence, it would be invalid to restrict the solution space in such a way, when computing $\text{cr}(G)$.

Let ℓ be some heuristic solution to the crossing number optimization problem, i.e., $\ell \geq \text{cr}(G)$. Clearly, the optimal solution will not require more than ℓ crossings on a single edge; on the other hand, we cannot easily restrict the maximum number of crossings per edge any further, as there are graphs where all the crossings lie on a single edge in any optimal solution, cf. again Figure 5.1.

The central idea of the following ILP is to consider a transformed graph $G^{[\ell]} = (V', E')$ which we obtain from G by subdividing each edge into a chain of ℓ *edge segments*. For linguistic simplicity, *edges* will in the following denote the edges of the original graph, and the term *segments* will denote the edges in the subdivided graph $G^{[\ell]}$. Considering any optimal crossing number solution for G, we can easily distribute the crossings over the different segments such that we obtain:

Observation 5.2. *If ℓ is any upper bound on the crossing number of G, the simple crossing number of $G^{[\ell]}$ is identical to the traditional crossing number of G, i.e.,*

$$\ell \geq \text{cr}(G) \quad \Rightarrow \quad \text{cr}(G) = \text{scr}\left(G^{[\ell]}\right).$$

We will not directly solve the crossing number problem on G, but the simple crossing number problem on the subdivided graph $G^{[\ell]}$. Hence we call our ILP *subdivision-based exact crossing minimization* (SECM).

Let $E'(e)$ be the set of segments corresponding to an edge $e \in E$, and vice versa, $E(e')$ the original edge in G corresponding to the segment $e' \in E'$. We define $\mathcal{CP}'$ as the set of all pairs of segments which may cross in an optimal drawing of $G^{[\ell]}$. Clearly, segments of the same original edge will never cross, neither will segments of adjacent original edges, cf. Section 2.3:

$$\mathcal{CP}' := \left\{ \{e, f\} \in E'^{\{2\}} \ : \ E(e) \cap E(f) = \emptyset \right\}.$$

This set allows us to define our crossing variables that are 1 if the segments

cross in the considered solution, and 0 otherwise:

$$z_{\{e,f\}} \in \{0,1\} \qquad \forall \{e,f\} \in \mathcal{CP}' \tag{5.4}$$

The objective function is then simply:

$$\min \sum_{\{e,f\} \in \mathcal{CP}'} w(E(e)) \cdot w(E(f)) \cdot z_{\{e,f\}} \tag{5.5}$$

And since we consider the simple crossing number on $G^{[\ell]}$ we only allow at most one crossing on each segment; we call these the *simplicity constraints*:

$$\sum_{f:\{e,f\} \in \mathcal{CP}'} x_{\{e,f\}} \leq 1 \qquad \forall e \in E' \tag{5.6}$$

Let a *simple crossing set* $\mathcal{R}' \subseteq \mathcal{CP}'$ be a set of unordered segment pairs which define simple crossings: A drawing realizing $\mathcal{R}'$ has a crossing between the edges e and f, for each $\{e,f\} \in \mathcal{R}'$. Furthermore, $\mathcal{R}'$ guarantees that it contains no pair $\{e,g\}$ with $g \neq f$, i.e., each edge is involved in at most one crossing. Starting from G, we can *realize* all crossings of $\mathcal{R}'$ by modeling the crossings via dummy nodes: For each $\{e,f\} \in \mathcal{R}'$ we create a new node d and substitute $e = \{u_e, v_e\}$ and $f = \{u_f, v_f\}$ by four edge segments $e_1 = \{u_e, d\}, e_2 = \{v_e, d\}$ and $f_1 = \{u_f, d\}, f_2 = \{v_f, d\}$ connecting the end points of e and f with d. We define the $\hat{}$-function as $\hat{e}_1 = \hat{e}_2 = e$ and $\hat{f}_1 = \hat{f}_2 = f$; for all unmodified edge segments e we have $\hat{e} = e$. We call the graph $G[\mathcal{R}']$ resulting from all these substitutions a *partial planarization* of G.

Clearly, all integer solutions $\bar{z}$ of the partial ILP above induce a simple crossing set $\mathcal{R}'[\bar{z}]$, which contains a segment pair if and only if the corresponding z-variable is set to 1. We write $G[\bar{z}]$ as a shorthand for $G[\mathcal{R}'[\bar{z}]]$, the partial planarization resulting from realizing the simple crossing set induced by $\bar{z}$. We have:

Lemma 5.3. *Let $\bar{z}$ be a solution of the 0/1-ILP subject to* (5.4) *and* (5.6)*. The vector $\bar{z}$ corresponds to a solution of the simple crossing number problem for $G^{[\ell]}$—and therefore of the crossing number problem for G—if and only if $G[\bar{z}]$ is planar.*

We say, $\bar{z}$ is *realizable* if $G[\bar{z}]$ is planar. From Kuratowski's theorem (Section 2.2) we know that $G[\bar{z}]$ will be planar if and only if it contains no subdivision of a K_5 or a $K_{3,3}$. For some fixed simple crossing set $\mathcal{R}'$, let K be the edges of a Kuratowski subdivision in $G[\mathcal{R}']$. Then, $\mathcal{CP}'(K)$ denotes the set of all edges pairs $\{\hat{e}_1, \hat{e}_2\} \subset E'$ with

1. $\{\hat{e}_1, \hat{e}_2\} \in \mathcal{CP}'$;
2. e_1, e_2 belong to different Kuratowski paths p_1, p_2 in K; and

3. the edges corresponding to p_1 and p_2 in the underlying K_5 or $K_{3,3}$ are non-adjacent (i.e., they may cross in order to planarize K).

We can hence formulate the Kuratowski constraints:

$$\forall \text{ simple crossing sets } \mathcal{R}', \forall \text{ Kuratowski subdivisions } K \text{ in } G[\mathcal{R}'] : \sum_{\substack{\{e,f\}\in \\ \mathcal{CP}'(K)\setminus\mathcal{R}'}} z_{\{e,f\}} \geq 1 - \sum_{\substack{\{e,f\}\in \\ \mathcal{CP}'(K)\cap\mathcal{R}'}} \left(1 - z_{\{e,f\}}\right) \tag{5.7}$$

Lemma 5.4. *Let $\bar{z}$ be a feasible solution of the ILP subject to* (5.4) *and* (5.6)*. The partial planarization $G[\bar{z}]$ is planar if and only if it satisfies all Kuratowski constraints* (5.7)*.*

Proof. $\Longleftarrow$: Assume there is a solution $\bar{z}$ which induces a planar $G[\bar{z}]$ but violates some constraint (5.7). Let $\mathcal{R}'$ be the simple crossing set and K the Kuratowski subdivision of this violated constraint, respectively. Since the z-variables are binary, the constraint can only by violated if the left-hand side is 0 and the right-hand side is 1. Hence the sum on the right-hand side has to be zero, and all z-variables in that sum are therefore 1. We have:

$$\bar{z}_{\{e,f\}} = \begin{cases} 0 & \text{if } \{e,f\} \in \mathcal{CP}'(K) \setminus \mathcal{R}' \\ 1 & \text{if } \{e,f\} \in \mathcal{CP}'(K) \cap \mathcal{R}' \end{cases}$$

Recall that $\mathcal{R}'[\bar{z}]$ is the simple crossing set induced by $\bar{z}$. But then K is also a Kuratowski subdivision in $G[\mathcal{R}'[\bar{z}]] = G[\bar{z}]$ and there are no crossings on the edges of K which would planarize the non-planar structure. Hence, $G[\bar{z}]$ is non-planar, which is a contradiction.

$\Longrightarrow$: Assume there is a solution $\bar{z}$ which satisfies all constraints but does not induce a planar $G[\bar{z}]$. Then, $G[\bar{z}]$ contains some Kuratowski subdivision K. We therefore consider the Kuratowski constraint for the simple crossing set $\mathcal{R}'[\bar{z}]$ and this K:

$$\sum_{\substack{\{e,f\}\in \\ \mathcal{CP}'(K)\setminus\mathcal{R}'[\bar{z}]}} z_{\{e,f\}} \geq 1 - \sum_{\substack{\{e,f\}\in \\ \mathcal{CP}'(K)\cap\mathcal{R}'[\bar{z}]}} (1 - z_{\{e,f\}})$$

The variables on the left-hand side are all 0, and the variables on the right-hand side are all 1. Hence the constraint is violated, which is a contradiction to $\bar{x}$ being feasible for all Kuratowski constraints. □

This leads to the main theorem on the validity of our ILP:

Theorem 5.5. *An optimal solution to the* SECM *0/1-ILP*

$$\min \left\{ \sum_{\{e,f\}\in\mathcal{CP}'} w(E(e)) \cdot w(E(f)) \cdot z_{\{e,f\}}, \text{ subject to } (5.4), (5.6), \text{ and } (5.7) \right\}$$

induces an optimal solution to the crossing number problem for G.

Remark 5.6. (Reducing the number of segments) When we are given some upper bound ℓ on the number of crossings, we usually will know how to obtain a planarization with that many crossings. Hence the ILP will mainly be used to answer the following question: Is the heuristic solution optimal, i.e. $\mathrm{cr}(G) = \ell$, or do we have $\mathrm{cr}(G) < \ell$? In the latter case, what does the solution look like?

Hence we can reduce ourselves to solve the problem not on $G^{[\ell]}$ but only on $G^{[\ell-1]}$. If there is a solution with at most $\ell - 1$ crossings, this graph will allow a corresponding solution for the simple crossing number as well. Otherwise, using $\ell - 1$ as an upper bound of the objective function, the ILP will turn out to be infeasible, which is a proof that ℓ is optimal.

5.3 Ordering-based 0/1-ILP

The second ILP solves the realization problem of multiple crossings per edge in a different way. Instead of transforming the graph, we explicitly allow that an edge is involved in multiple crossings, and introduce linear-ordering subproblems on each edge. Hence we call the approach *ordering-based exact crossing minimization* (OECM).

To be able to define orderings on an edge, we need the edges to have a direction. Therefore, we consider any arbitrary but fixed *orientation* of G—for notational simplicity we will continue to denote this now-orientated graph by G.

We can directly reuse the x-variables and the set $\mathcal{CP}$ of edges that may cross in any optimal drawing of G, as originally introduced for RECM. Using the shorthand $\{e; f_1, f_2, \ldots\} := \{\{e, f_1\}, \{e, f_2\}, \ldots\}$, we can define two variable sets:

$$x_{\{e,f\}} \in \{0,1\} \qquad \forall \{e,f\} \in \mathcal{CP} \tag{5.8}$$

$$y_{e,f,g} \in \{0,1\} \qquad \forall (e,f,g) \in E^{\langle 3 \rangle}, \{e;f,g\} \subseteq \mathcal{CP} \tag{5.9}$$

A variable $x_{\{e,f\}}$ specifies whether or not the edges e and f cross. A variable $y_{e,f,g}$ is 1 if and only if both edges f and g cross e, and the crossing (e, f) is nearer to e's source node than the crossing (e, g). We say e is the *base* of the variable. The objective function of our ILP is identical to RECM:

$$\min \sum_{\{e,f\} \in \mathcal{CP}} w(e) \cdot w(f) \cdot x_{\{e,f\}} \tag{5.10}$$

Linear-Ordering Constraints. We define a set of *linear-order* (LO) constraints which ensure a consistent linear ordering over all edges:

$$x_{\{e,f\}} \geq y_{e,f,g}, \quad x_{\{e,g\}} \geq y_{e,f,g} \qquad \forall (e,f,g) \in E^{\langle 3 \rangle}, \ \{e;f,g\} \subseteq \mathcal{CP} \tag{5.11}$$

$$1 + y_{e,f,g} + y_{e,g,f} \geq x_{\{e,f\}} + x_{\{e,g\}} \qquad \forall (e,f,g) \in E^{\langle 3 \rangle}, \ \{e;f,g\} \subseteq \mathcal{CP} \tag{5.12}$$

$$y_{e,f,g} + y_{e,g,f} \leq 1 \qquad \forall (e,f,g) \in E^{\langle 3 \rangle}, \ \{e;f,g\} \subseteq \mathcal{CP} \tag{5.13}$$

$$y_{e,f,g} + y_{e,g,h} + y_{e,h,f} \leq 2 \qquad \forall (e,f,g,h) \in E^{\langle 4 \rangle}, \ \{e;f,g,h\} \subseteq \mathcal{CP} \tag{5.14}$$

We introduce *crossing-existence* constraints (5.11) which connect the x- and y-variables by ensuring that the x-vector specifies a crossing if the y-variables do. Vice versa, the *order-existence* constraints (5.12) ensure that if x specifies two crossings on the same edge, the y-vector has to specify their order. The *mirror-order* constraints (5.13) guarantee that two crossings are uniquely ordered if they exist. Analogously, the *cyclic-order* constraints (5.14) ensure that the orderings are acyclic. A solution $(\bar{x}, \bar{y})$ which satisfies the LO-constraints is called *LO-feasible*. Since no two edges will ever cross more than once in any optimal solution, we have:

Proposition 5.7. *Let $\bar{x}$ be the assignment for the vector x, describing any optimal solution to the crossing number problem of any graph G. There exists an assignment $\bar{y}$ for the vector y such that $(\bar{x}, \bar{y})$ is LO-feasible.*

Checking feasibility. Let $(\bar{x}, \bar{y})$ be any integer LO-feasible solution. We replace each crossing in G by a dummy vertex. Since we know the intended order of these dummy vertices on each edge from the information in $(\bar{x}, \bar{y})$, the resulting graph is a *partial planarization* of G, which we denote by $G[\bar{x}, \bar{y}]$. We can check whether $(\bar{x}, \bar{y})$ describes a feasible solution to the crossing number problem by testing $G[\bar{x}, \bar{y}]$ for planarity; if it does, we say $(\bar{x}, \bar{y})$ is *realizable*. Therefore we have, similar to Lemma 5.3:

Lemma 5.8. *Let $(\bar{x}, \bar{y})$ be a feasible solution of the 0/1-ILP subject to* (5.8), (5.9) *and the LO-constraints* (5.11)–(5.14). *The vector $(\bar{x}, \bar{y})$ corresponds to a solution of the crossing number problem for G if and only if $G[\bar{x}, \bar{y}]$ is planar.*

Kuratowski Constraints. The final class of constraints required to fully describe the feasible points of our ILP are the *Kuratowski constraints*. They guarantee that a computed integer LO-feasible solution $(\bar{x}, \bar{y})$ corresponds to a feasible planarization, i.e., $G[\bar{x}, \bar{y}]$ is planar.

For any Kuratowski subdivision K, we require at least one crossing between the edges of K. Such a subdivision might not be a subgraph of the

original graph G, but might occur only in a partial planarization $G[\bar{x}, \bar{y}]$ for some integer LO-feasible solution $(\bar{x}, \bar{y})$.

For SECM we simply use the crossings in such a planarization to "turn off" Kuratowski-constraints which are only valid if these crossings are selected, cf. Section 5.2. The drawback is that we unavoidably have a multitude of very similar constraints, where, e.g., an involved segment f_1 is replaced by another segment f_2, but $\hat{f}_1 = \hat{f}_2$, i.e., f_1 and f_2 were created by the initial subdividing of the graph and correspond to the same original edge.

We cannot reuse such a simple approach straight-forwardly for OECM. But now the additional effort is compensated for by constraints which correspond to a whole class of similar Kuratowski constraints in SECM. Let $(\bar{x}, \bar{y})$ be an integer LO-feasible solution, and let K be a Kuratowski subdivision in $G[\bar{x}, \bar{y}]$. We define $\mathcal{Z}_K[\bar{x}, \bar{y}]$ as the set of crossings induced by $(\bar{x}, \bar{y})$ whose dummy nodes form integral parts of K: Any $\{e, f\} \in \mathcal{Z}_K[\bar{x}, \bar{y}]$ either induces a Kuratowski node or there exist a segment e' of e, a segment f' of f, and a Kuratowski path which contains $\langle e', f' \rangle$ as a sub-path. We can then define the *crossing shadow* $(\mathcal{X}_K[\bar{x}, \bar{y}], \mathcal{Y}_K[\bar{x}, \bar{y}])$ as a pair of sets as follows:

$\mathcal{Y}_K[\bar{x}, \bar{y}] := \{(e, f, g) \in E^{\langle 3 \rangle} \mid \{e, f\}, \{e, g\} \in \mathcal{Z}_K[\bar{x}, \bar{y}] \wedge \bar{y}_{e,f,g} = 1 \wedge \nexists \{e, h\} \in \mathcal{Z}_K : \bar{y}_{e,f,h} = \bar{y}_{e,h,g} = 1\}$, i.e., the triple (e, f, g) is in $\mathcal{Y}_K[\bar{x}, \bar{y}]$, if no other edge crosses e between f and g. Thus $\mathcal{Y}_K[\bar{x}, \bar{y}]$ contains a minimal description of all crossings and their orderings in K, except for crossings of two edges that are both not involved in multiple crossings; these are collected in the following set:

$\mathcal{X}_K[\bar{x}, \bar{y}] := \{\{e, f\} \in \mathcal{Z}_K[\bar{x}, \bar{y}] \mid \forall g \in E : \{(e, f, g), (e, g, f), (f, e, g), (f, g, e)\} \cap \mathcal{Y}_K[\bar{x}, \bar{y}] = \emptyset\}$, i.e., all *singular crossings* in K not contained in $\mathcal{Y}_K[\bar{x}, \bar{y}]$.

Proposition 5.9. *For each integer LO-feasible solution $(\bar{x}, \bar{y})$ and each Kuratowski subdivision K in $G[\bar{x}, \bar{y}]$ we have: The partial planarization of G only realizing the crossings (and their order) as defined by the crossing shadow, contains K as a Kuratowski subdivision.*

Using the crossing shadow, we can now define Kuratowski constraints as

$$\sum_{\{e,f\} \in \mathcal{CP}(K)} x_{\{e,f\}} \geq 1 - \sum_{a \in \mathcal{X}_K[\bar{x}, \bar{y}]} (1 - x_a) - \sum_{b \in \mathcal{Y}_K[\bar{x}, \bar{y}]} (1 - y_b) \tag{5.15}$$

for all LO-feasible integer vectors $(\bar{x}, \bar{y})$ and all Kuratowski subdivisions K in $G[\bar{x}, \bar{y}]$. Here and in the sequel, $\mathcal{CP}(K)$ denotes the set of all edges pairs $\{e_1, e_2\} \subset E$ with

1. $\{e_1, e_2\} \in \mathcal{CP}$;
2. e_1, e_2 (or segments of them, due to the introduction of dummy nodes) belong to different Kuratowski paths p_1, p_2 in K; and

3. the edges corresponding to p_1 and p_2 in the underlying K_5 or $K_{3,3}$ are non-adjacent (i.e., they may cross in order to planarize K).

Our constraints require at least one crossing on every Kuratowski subdivision if it exists; this existence is detected via the crossing shadow.

Lemma 5.10. *Each optimal solution to the crossing number problem of any graph G corresponds to a feasible integer solution vector.*

Proof. Clearly, any solution to the crossing number problem can be described by an integer LO-feasible solution $(\bar{x}, \bar{y})$ by construction, see Proposition 5.7. It remains to show that this vector does not violate any constraint (5.15). Assume there is some $(\bar{x}, \bar{y})$ and K which induces a violated Kuratowski constraint. Then

$$\sum_{\{e,f\} \in \mathcal{CP}(K)} x_{\{e,f\}} < 1 - \sum_{a \in \mathcal{X}_K(\bar{x},\bar{y})} (1 - x_a) - \sum_{b \in \mathcal{Y}_K[\bar{x},\bar{y}]} (1 - y_b).$$

Since we only consider integer solutions, the left-hand side is 0 while the right-hand side is 1. We thus have:

$$\forall \{e, f\} \in \mathcal{CP}(K) : x_{\{e,f\}} = 0 \text{ , and} \tag{5.16}$$

$$\forall a \in \mathcal{X}_K[\bar{x}, \bar{y}] : x_a = 1 \ \wedge \ \forall b \in \mathcal{Y}_K(\bar{x}, \bar{y}) : y_b = 1 \,.$$

But then, due to Proposition 5.9, the crossing shadow of $(\bar{x}, \bar{y})$ w.r.t. K specifies exactly the crossings to induce a graph that contains K as a Kuratowski subdivision. Due to (5.16) we know that there are no further crossings on K which could lead to a planarization of this non-planar subgraph. This is a contradiction to the feasibility of the original solution. □

Lemma 5.11. *Every feasible solution to the above ILP corresponds to a feasible solution of the crossing number problem.*

Proof. We can interpret any integer LO-feasible solution $(\bar{x}, \bar{y})$ as a (partial) planarization $G' := G[\bar{x}, \bar{y}]$. Assume that the solution vector satisfies all Kuratowski constraints, but G' is non-planar. Then there exists a Kuratowski subdivision in G'. Let K be such a Kuratowski subdivision with the smallest number of contained dummy nodes, and among them the one with the fewest edges. We construct a crossing shadow $(\mathcal{X}_K[\bar{x}, \bar{y}], \mathcal{Y}_K[\bar{x}, \bar{y}])$ which describes the precise crossing configuration necessary to identify K. Since K is a non-planar (minimal) Kuratowski subdivision, we know that there are no crossings on any pair of $\mathcal{CP}(K)$. But then, constraint (5.15) is violated for K and $(\mathcal{X}_K[\bar{x}, \bar{y}], \mathcal{Y}_K(\bar{x}, \bar{y}))$, as the left-hand side sums up to 0 and the right-hand side is 1. □

We therefore obtain:

Theorem 5.12. *Every optimal solution of the 0/1-ILP*

$$\min \left\{ \sum_{\substack{\{e,f\} \\ \in \mathcal{CP}}} w(e)w(f)x_{\{e,f\}}, \textit{ subj. to } (5.8),(5.9),(5.11)\text{–}(5.14), \textit{ and all } (5.15) \right\}$$

yields an optimal solution of the crossing number problem.

5.3.1 Triangle Constraints

We present an additional class of constraints for the above OECM formulation. We therefore consider (undirected) 3-cycles in the graph and can define *triangle-constraints* forbidding certain crossing structures on such cycles, cf. Figure 5.2: Consider a 3-cycle of the edges e, f, g; say f is the edge incident to e's target node. Furthermore, consider two adjacent edges a and b, with a common vertex p, both of which cross over e; say a crosses nearer to e's source node. Then $\{a, f\}$ and $\{b, g\}$ may only be crossings if either a also crosses g, or b also crosses f. We can write this formally as our *simple triangle constraints*:

$$y_{e,a,b} + x_{\{f,a\}} + x_{\{g,b\}} \leq 2 + x_{\{f,b\}} + x_{\{g,a\}} \tag{5.17}$$

$\forall e = \{u, v\}, f = \{v, w\}, g = \{u, w\}, a = \{p, q\}, b = \{p, q'\} \in E$ such that $|\{u, v, w, p, q, q'\}| = 6$ and $e = (u, v)$ in the chosen orientation of G. I.e., the triangle and a, b are disjoint and we specifically require that f specifies the edge incident to e's target node, considering the fixed orientation used in the OECM formulation.

We can extend this notion further, by requiring a and b not to be adjacent, but only connected via a simple path P. We obtain the *extended triangle constraints*:

$$y_{e,a,b} + x_{\{f,a\}} + x_{\{g,b\}} \leq 2 + x_{\{f,b\}} + x_{\{g,a\}} + x_{\{a,b\}} + \sum_{e' \in \{e,f,g\}} \sum_{f' \in P} x_{\{e',f'\}} \tag{5.18}$$

$\forall e = \{u, v\}, f = \{v, w\}, g = \{u, w\}, a = \{p, q\}, b = \{p', q'\} \in E, \forall P = (p \to p')$ such that $|\{u, v, w, p, p', q, q'\}| = 7$, $e = (u, v)$ in the chosen orientation of G, and $P \cap \{e, f, g, a, b\} = \emptyset$. The constraint describing the crossings between a, b and the triangle is only restrictive as long as the path P does not cross the triangle.

Theorem 5.13. *The triangle-constraints* (5.17) *and* (5.18) *are valid.*

Proof. Assume there is an optimal integer solution $(\bar{x}, \bar{y})$ of the ILP without the triangle-constraints, but it violates at least one of the triangle-constraints: Let e, f, g, a, b be the appropriate edges of such a violated constraint. We know that

$$y_{e,a,b} = x_{\{f,a\}} = x_{\{g,b\}} = 1 \text{ and } x_{\{f,b\}} = x_{\{g,a\}} = x_{\{a,b\}} = 0.$$

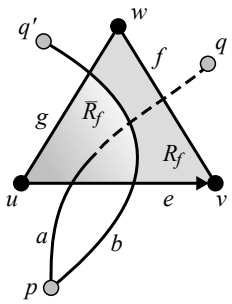


Figure 5.2: A triangle constraint forbids illegal crossing configurations for edges crossing over a triangle.

Furthermore, $x_{\{e',f'\}} = 0$ for all $\{e', f'\} \in \{e, f, g\} \times P$, if the constraint is an extended triangle constraint. Consider a drawing of the solution $(\bar{x}, \bar{y})$ on a sphere and let R be the region defined by the 3-cycle $\{e, f, g\}$ that does not include p. If the constraint is extended, we also know that p' does not lie in R, since the path $P = (p \to p')$ does not cross the 3-cycle.

Since any pair of edges crosses at most once, we have: Any edge starting outside of R which crosses over one of these three edges, say e, either (i) does not cross over f nor g and ends inside of R, (ii) crosses exactly one of $\{f, g\}$, and ends outside of R, or (iii) crosses f and g and ends inside of R. For the edge b, we have case (ii), and hence b partitions the region R into two subregions R_f and $\bar{R}_f$, the former being the region which borders the edge f. We know that a crosses e and f. Assume w.l.o.g. that a is oriented from p to q. The crossing with f may occur before or after the crossing with e:

$y_{a,e,f} = 1$**:** The edge a crosses e first, and f later. Since $y_{e,a,b} = 1$, we know that a has a crossing with e before e is crossed by b. Hence a starts from outside of R, crosses over e and enters the region $\bar{R}_f$. It cannot cross e twice in an optimal drawing, and we know that a does not cross g nor b. Hence a cannot cross f which is a contradiction to $x_{\{f,a\}} = 1$.

$y_{a,f,e} = 1$**:** The edge a crosses f first, and e later. But, starting from outside of R, after crossing f, a enters the region R_f. It cannot cross f twice in an optimal drawing, and we know that a does not cross g nor b. Hence, in order to leave R_f, a has to cross over e after e was crossed by b, which is a contradiction to $y_{e,a,b} = 1$. □

Observation 5.14. *There are only polynomially many simple triangle constraints. Hence they can be separated in polynomial time.*

5.4 Facets in the Crossing Number Polytope

In this section, we will study some polyhedral properties of the *crossing number polytope*. Both above formulations were centered around the same con-

cept, denoted by RECM, of having variables to describe pairs of crossing edges, cf. Section 5.1.

Definition 5.15 (Crossing Number Polytope)**.** Let $\mathcal{F}$ be the assignments of the x-variables, which correspond to feasible solutions to the crossing number problem. The convex hull of the points in $\mathcal{F}$ then forms the *crossing number polytope*

$$\mathcal{P}_{\text{cr}} := \text{conv}\{x \in \mathcal{F}\}.$$

As stated before, it is an open problem whether there exists a practically relevant formulation using only these variables, while directly including the realizability subproblem. We can look at our above formulations from the polyhedral point of view:

SECM**:** The formulation expands the given graph in order to not consider the crossing number polytope directly; instead it considers the higher-dimensional *simple crossing number polytope* $\mathcal{P}_{\text{scr}}$, and uses the projection

$$\text{proj}_{z\to x}: \sum_{\substack{\{e',f'\}\in\mathcal{CP}',\\ e'\in E'(e),\\ f'\in E'(f)}} z_{\{e',f'\}} = x_{\{e,f\}} \qquad \forall\{e,f\}\in\mathcal{CP} \tag{5.19}$$

to obtain a solution within $\mathcal{P}_{\text{cr}}$.

OECM**:** This formulation on the other hand, cuts $\mathcal{P}_{\text{cr}}$ with $|E|$ linear-ordering polytopes, in order to give a description of the feasible solutions.

In the following, we will show the strength of Kuratowski constraints in the context of complete and complete bipartite graphs. Recall that, theoretically, our ILP allows us to expand any given graph $G = (V, E)$ into a complete graph by adding the edges $\bar{E} = V^{\{2\}} \setminus E$ and setting $w(e) = 0$ for all $e \in \bar{E}$. We could then solve the problem on a weighted complete graph $K_{|V|}$.

We use the abbreviations $\mathcal{CP}_n$ or $\mathcal{CP}_{n,m}$ for $\mathcal{CP}$ if considering K_n or $K_{n,m}$, respectively.

Definition 5.16 (Complete (Bipartite) Crossing Number Polytope)**.** Let $\mathcal{F}_n$ be the set of all feasible solutions for K_n in terms of x variables, i.e., for each $\bar{x} \in \mathcal{F}_n$ there exists a drawing of K_n which has exactly the crossings described by $\bar{x}$. Analogously, let $\mathcal{F}_{n,m}$ be the feasible solutions with respect to drawings of $K_{n,m}$. As special variants of $\mathcal{P}_{\text{cr}}$ we can then define the *complete crossing number polytope*

$$\mathcal{K}_n := \text{conv}\{x \in \mathcal{F}_n\},$$

and the *complete bipartite crossing number polytope*

$$\mathcal{K}_{n,m} := \text{conv}\{x \in \mathcal{F}_{n,m}\}.$$

We write V_n and E_n for the nodes and edges of the complete graph K_n, respectively. Analogously, we write V_n and V'_m for the node partitions of the graph $K_{n,m}$, with $|V_n| = n$ and $|V'_m| = m$, and $E_{n,m}$ for its edges.

A central constraint of all above formulations are the *Kuratowski constraints*. We will consider these constraints in the realm of the pure x-variable space. Furthermore, we can expand these constraints to variants where we consider more complex non-planar graphs than only K_5 and $K_{3,3}$ subgraphs.

Definition 5.17 (K_5-constraint)**.** Fix some $n > 5$ and consider the complete graph $K_n = \left(V_n, E_n = V_n^{\{2\}}\right)$. For each $W \in V_n^{\{5\}}$ we define a K_5-*constraint* C_5^W:

$$\sum_{\substack{\{e,f\} \in \mathcal{CP}_n, \\ e,f \subset W}} x_{\{e,f\}} \geq 1.$$

Definition 5.18 (K_m-constraint)**.** Fix some $n > 5$ and consider the complete graph $K_n = \left(V_n, E_n = V_n^{\{2\}}\right)$. Assume we know $\mathrm{cr}(K_m)$, for some $5 \leq m < n$, then for each $W \in V_n^{\{m\}}$ we define a K_m-*constraint* C_m^W as a generalization of the K_5 constraints:

$$\sum_{\substack{\{e,f\} \in \mathcal{CP}_n, \\ e,f \subset W}} x_{\{e,f\}} \geq \mathrm{cr}(K_m).$$

It is obvious that a K_m subgraph, $m \geq 3$, can never be found in complete bipartite graphs. On the other hand, $K_{3,3}$ subgraphs occur in both complete, and complete bipartite graphs. This is of particular interest since in most non-planar real-world graphs, $K_{3,3}$ subdivisions are much easier to find (from the practical point of view) than K_5 subdivisions: Non-planarity witnesses of planarity tests are much more likely to be subdivisions of $K_{3,3}$ rather than of K_5 [133]. Even in complete graphs K_n, there are only $\binom{n}{5}$ K_5 subgraphs, while we can enumerate $10\binom{n}{6}$ $K_{3,3}$ subgraphs.

Definition 5.19 ($K_{3,3}$-constraint (in K_n))**.** Consider the complete graph $K_n = (V_n, E_n = V_n^{\{2\}})$ for some fixed n. For each $W \in V_n^{\{3\}}$ and $W' \in (V_n \setminus W)^{\{3\}}$ we define a $K_{3,3}$-*constraint* $C_{3,3}^{W,W'}$:

$$\sum_{\substack{\{e,f\} \in \mathcal{CP}_n \\ |e \cap W| = |e \cap W'| = |f \cap W| = |f \cap W'| = 1}} x_{\{e,f\}} \geq 1.$$

Definition 5.20 ($K_{3,3}$-constraint (in $K_{n,m}$))**.** Consider the complete graph $K_{n,m} = (V_n \dot\cup V'_m, E_{n,m} = \{\{u,v\} : u \in V_n, v \in V'_m\})$ for some fixed n and m. For each $W \in V_n^{\{3\}}$ and $W' \in V_m'^{\{3\}}$ we define a $K_{3,3}$-*constraint* $C_{3,3}^{W,W'}$:

$$\sum_{\substack{\{e,f\} \in \mathcal{CP}_{n,m} \\ e,f \subset (W \cup W')}} x_{\{e,f\}} \geq 1.$$

We will show the strength of the above constraints, in particular that they are *facet defining* in the following settings:

- K_5-constraints define facets for K_n (Section 5.4.2)
- K_m-constraints define facets for K_n (Section 5.4.3)
- $K_{3,3}$-constraints define facets for $K_{n,m}$ (Section 5.4.4)
- $K_{3,3}$-constraints define facets for K_n (Section 5.4.5)

Although the second item also induces the first item, we prove the special case K_5 separately beforehand, as it showcases the strategy used in all the subsequent proofs in the most simple way. This overall structure is described in detail in the section hereafter. The proofs themselves will then heavily reference the argumentation outlined as our proving strategy; they basically only fill in the gaps left by the overall strategy.

5.4.1 Proving strategy

The general proof structure is as follows:

Fixing. We fix the considered input graph G and any arbitrary Kuratowski constraint C of the considered types. Such a constraint can be written as $c^T x \geq \hat{c}$. We define $X_c = \{x \in \mathcal{F} \mid c^T x = \hat{c}\}$ as the set of feasible solutions satisfying C with equality. Note that c is a binary vector, i.e., each entry in c is either 0 or 1.

Let there be a valid inequality A with $a^T x \geq \hat{a}$ for which $X_c \subseteq X_a = \{x \in \mathcal{F}_n \mid a^T x = \hat{a}\}$ holds.

(Aim.) In order to show that C defines a facet, the remaining parts of the respective sections focus on showing that for any such inequality A we have $a_{\{e,f\}} = \lambda c_{\{e,f\}}$ and $\hat{a} = \lambda \hat{c}$ for some $\lambda > 0$. We can show this by proving that for all pairs $\{e, f\}$ with $c_{\{e,f\}} = 0$, we have $a_{\{e,f\}} = 0$; furthermore all non-zero coefficients of a have to be identical.

Partitioning. We start with partitioning the edges of the underlying complete (bipartite) graph into sets according to their incidence with W (and W'). These edge sets are denoted by $S_\star$, $\star \in \mathcal{I}$, over some suitable index set $\mathcal{I}$.

This partitioning also induces a natural partitioning of the crossing pairs in $\mathcal{CP}$, according to the memberships of their edges: A pair $\{e, f\} \in \mathcal{CP}$ belongs to the partition $P_{\star,\bullet}$ if $e \in S_\star$ and $f \in S_\bullet$, or vice versa. To avoid ambiguities, we assume that the index set $\mathcal{I}$ is *ordered* and require that $\star \leq \bullet$, using this ordering.

We will always observe: There exists exactly one index $\dagger \in \mathcal{I}$ such that all variables corresponding to $P_{\dagger,\dagger}$ have a non-zero coefficient in C. For all other partitions all induced coefficients in C are zero.

Lemma: Permutation Classes. The next step of the proof is to establish a lemma describing *permutation classes*: We prove—by the argument of permuting the index-labels of the vertices in the given complete (bipartite) graph—that all edge pairs $\{e, f\}$ belonging to the same partition $P_{\star,\bullet}$ form an equivalence class with respect to their coefficients $a_{\{e,f\}}$, i.e., they have to have a common coefficient in A, denoted by $\alpha_{\star,\bullet}$ in the following.

Theorem: Facet. The final step in each proof is performed as follows: Step by step we fix a partition $P_{\star,\bullet} \neq P_{\dagger,\dagger}$. We consider two (similar) feasible solutions that satisfy C with equality, and investigate the difference of their crossings. The solutions are chosen in such a way that they only differ in the number of crossings of type $P_{\star,\bullet}$, thus inducing $\alpha_{\star,\bullet} = 0$ in order for both solutions to also satisfy A with equality—as required by the initial declaration of A.

Clearly, in later steps the considered drawings may also differ in the number of other types of crossings, if prior steps already showed that the coefficients for these crossing types are zero.

After performing this step for each permutation class $P_{\star,\bullet} \neq P_{\dagger,\dagger}$, we know that $\alpha_{\star,\bullet} = 0$ unless $\star = \bullet = \dagger$. Furthermore we know that all crossing pairs in $P_{\dagger,\dagger}$ have a common coefficient $\alpha_{\dagger,\dagger} \neq 0$. Hence the inequality A can only be a positive multiple of C. Since A is not stronger than C, each constraint of the considered type defines a facet in the crossing number polytope of the given graph.

5.4.2 K_5-constraints in K_n

Please refer to Section 5.4.1 for the overall structure of the proof.

Fixing 5.21. Fix any $n > 5$ and $W \in V_n^{\{5\}}$. This induces:

$$\begin{aligned} G &= K_n \\ C &= C_5^W \quad (K_5\text{-constraint}) \\ c_{\{e,f\}} &= \begin{cases} 1 & \text{if } e, f \subset W, \\ 0 & \text{else.} \end{cases} \qquad \forall \{e, f\} \in \mathcal{CP}_n \\ \hat{c} &= 1 \\ \mathcal{F} &= \mathcal{F}_n \end{aligned}$$

Partitioning 5.22. We can partition all edges $e \in E_n$ using the ordered index set $\mathcal{I} = \langle 0, 1, 2 \rangle$, based on which nodes they connect:

$$e \in S_i \quad \Longleftrightarrow \quad |e \cap W| = i.$$

Considering the induced partitions of $\mathcal{CP}_n$, we have $\dagger = 2$, i.e., $c_{\{e,f\}} = 0$ for all $\{e, f\} \notin P_{2,2}$, and 1 otherwise.

Lemma 5.23 (Permutation classes)**.** *For all $i, j \in \mathcal{I}$, $i \leq j$, we have:*

$$\{e, f\}, \{g, h\} \in P_{i,j} \quad \Longrightarrow \quad a_{\{e,f\}} = a_{\{g,h\}}$$

Proof. W.l.o.g. assume that $W = \{v_1, \ldots, v_5\}$. Consider a feasible solution as shown in Figure 5.3(a). The edges S_2, over which C_5^W is defined, are drawn with bold lines. Clearly, this solution satisfies C_5^W with equality. Furthermore, C_5^W has exactly one variable for each possible optimal K_5 solution and only one of its variables can be 1 in any solutions of X_c.

We can relabel the nodes W arbitrarily in Figure 5.3(a), without violating the feasibility of the solution, nor the equality of C_5^W. We do also not violate these properties by any permutation of the node labels of $V \setminus W$.

Take any two edges e and f of the same partition S_i. Note that there is a permutation of the node labels of V and a permutation of the node labels of $V \setminus W$ in Figure 5.3(a), such that these two edges switch their roles. Considering all possible label permutations on W and on $V \setminus W$, A cannot distinguish between two edges if they belong to the same partition S_i for some i, because it has to satisfy all solutions arising from all possible label permutations on W and on $V \setminus W$ with equality. Since A can only differentiate between different partitions S_i, it can also only differentiate between different partitions $P_{i,j}$ of the crossing pairs. □

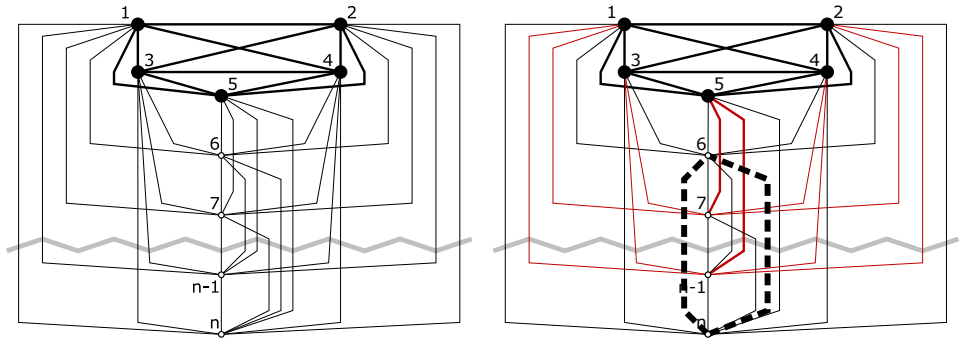
Theorem 5.24. *Fix any $n > 5$ and $W \in V_n^{\{5\}}$. C_5^W is a facet in $\mathcal{K}_n$.*

Proof. Again, we assume w.l.o.g. that $W = \{v_1, \ldots, v_5\}$ and consider a first feasible solution as shown in Figure 5.3(a). We call the drawing of the path $v_5, v_6, \ldots, v_n$ the *spine* of the drawing.

$\alpha_{0,1} = 0$: cf. Figure 5.3(b). We can redraw the edge $\{v_6, v_n\} \in S_0$ (bold, dashed) on the left side of the spine instead of the right side, and still obtain a solution in X_c. Thereby we only change crossings of type $P_{0,1}$. Since we remove more crossings then we introduce, and all coefficients of these crossings are identical, we have: $\alpha_{0,1} = 0$.

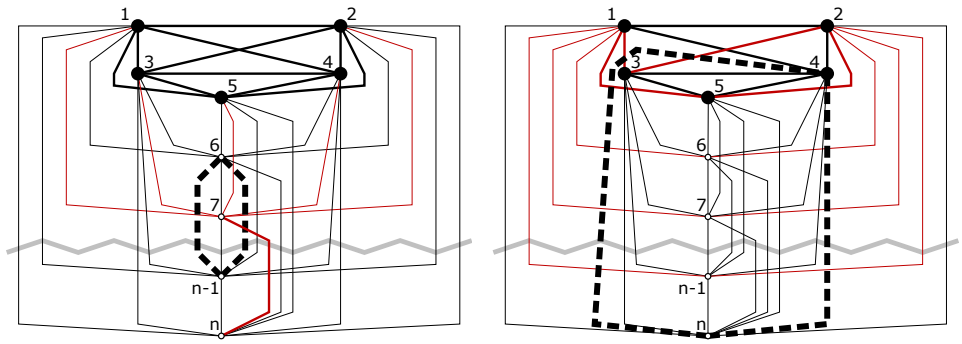
$\alpha_{0,0} = 0$: cf. Figure 5.3(c). We can redraw the edge $\{v_6, v_{n-1}\} \in S_0$ on the left side of the spine instead of the right side, and still obtain a solution in X_c. We remove more $P_{0,1}$ crossings than we introduce, but we already know that $\alpha_{0,1} = 0$. Hence it remains to observe that the redrawing removes a $P_{0,0}$ crossings and we therefore have: $\alpha_{0,0} = 0$.

$\alpha_{1,2} = 0$: cf. Figure 5.3(d). We can redraw the edge $\{v_4, v_n\} \in S_1$ on the left side of the spine instead of the right side, and let it cross through multiple edges of S_2. We still obtain a solution in X_c. We introduce more $P_{1,2}$ crossings than we remove and hence we have: $\alpha_{1,2} = 0$.



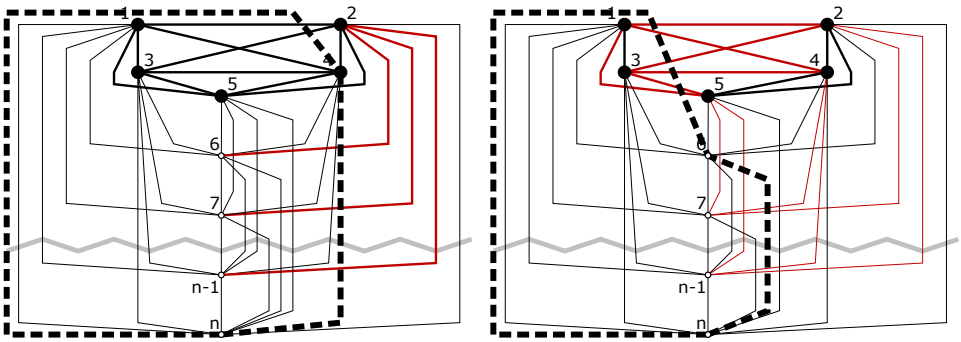
(a) Drawing scheme of K_n

(b) $a_{\{e,f\}} = 0,\ \forall\{e,f\} \in P_{0,1}$



(c) $a_{\{e,f\}} = 0,\ \forall\{e,f\} \in P_{0,0}$

(d) $a_{\{e,f\}} = 0,\ \forall\{e,f\} \in P_{1,2}$



(e) $a_{\{e,f\}} = 0,\ \forall\{e,f\} \in P_{1,1}$

(f) $a_{\{e,f\}} = 0,\ \forall\{e,f\} \in P_{0,2}$

Figure 5.3: K_5-constraints are facet-defining in K_n.

$\underline{\alpha_{1,1} = 0}$: cf. Figure 5.3(e). We can redraw the edge $\{v_4, v_n\} \in S_1$ on the left side of the drawing instead of the right side of the spine. Thereby it crosses multiple edges of S_2 but no S_1 edges. We still obtain a solution in X_c. We already know that $\alpha_{1,2} = 0$ and can concentrate on the $P_{1,1}$ crossings. We remove such crossings without introducing any, and conclude: $\alpha_{1,1} = 0$.

$\underline{\alpha_{0,2} = 0}$: cf. Figure 5.3(f). We can redraw the edge $\{v_6, v_n\} \in S_0$ on the left side of the drawing instead of the right side of the spine. Thereby it crosses multiple edges of S_2 but no S_1 edges. We still obtain a solution in X_c. We already know that $\alpha_{0,1} = 0$ and can concentrate on the $P_{0,2}$ crossings. We introduce such crossings without removing any, and conclude: $\alpha_{0,2} = 0$. □

5.4.3 K_m-constraints in K_n

Please refer to Section 5.4.1 for the overall structure of the proof.

Fixing 5.25. Fix any $m \geq 5$, $n > m$, and $W \in V_n^{\{m\}}$. This induces:

$$\begin{aligned} G &= K_n \\ C &= C_m^W \quad (K_m\text{-constraint}) \\ c_{\{e,f\}} &= \begin{cases} 1 & \text{if } e, f \subset W, \\ 0 & \text{else.} \end{cases} \qquad \forall \{e, f\} \in \mathcal{CP}_n \\ \hat{c} &= \operatorname{cr}(K_m) \\ \mathcal{F} &= \mathcal{F}_n \end{aligned}$$

Partitioning 5.26. We can partition all edges $e \in E_n$ using the ordered index set $\mathcal{I} = \langle 0, 1, 2 \rangle$, based on which nodes they connect:

$$e \in S_i \quad \Longleftrightarrow \quad |e \cap W| = i.$$

Considering the induced partitions of $\mathcal{CP}_n$, we have $\dagger = 2$, i.e., $c_{\{e,f\}} = 0$ for all $\{e, f\} \notin P_{2,2}$, and 1 otherwise.

Lemma 5.27 (Permutation classes)**.** *For all $i, j \in \mathcal{I}$, $i \leq j$, we have:*

$$\{e, f\}, \{g, h\} \in P_{i,j} \quad \Longrightarrow \quad a_{\{e,f\}} = a_{\{g,h\}}$$

Proof. Consider a feasible solution as shown in Figure 5.4(a). The nodes W lie in the cyclic shaded region on the left-hand side of the drawing. We assume that this region contains an optimal drawing of the K_m induced by W. Since we do not know the exact drawing for arbitrary m, we only visualize a couple of S_2 edges; our proof will not need the knowledge of the exact edge placements. We note that for any drawing of the K_m, we can choose an outer face, where at least one nodes lies on the outside of the K_m-drawing: In our

drawing we assume that the left-most node lies on the outside, denoted by its placement on the circle's border; for all other nodes we do neither assume that they lie on the outside, not that they do not.

All nodes $V \setminus W$ are lined up on the left border of rectangular shaded region on the right-hand side of the drawing: The region itself contains all S_0 edges. Finally, the S_1 edges (connecting W with $V \setminus W$) are partially drawn using thick gray edges, denoting that all the considered edges are drawn in parallel without crossings, until they split up into normal thin black lines. All crossings between S_1 edges are in the shaded region at the center of the drawing.

Clearly, this solution satisfies C_m^W with equality.

We can label the nodes W arbitrarily in Figure 5.4(a), without violating the feasibility of the solution, nor the equality of C_m^W. We do also not violate these properties by any permutation of the labels of the nodes of $V \setminus W$.

Take any two edges e and f of the same partition S_i. Note that there is a permutation of the node labels of V and a permutation of the node labels of $V \setminus W$ in Figure 5.4(a), such that these two edges switch their roles. Considering all possible label permutations on W and on $V \setminus W$, A cannot distinguish between two edges if they belong to the same partition S_i for some i, because it has to satisfy all solutions arising from all possible label permutations on W and on $V \setminus W$ with equality. Since A can only differentiate between different partitions S_i, it can also only differentiate between different partitions $P_{i,j}$ of the crossing pairs. □

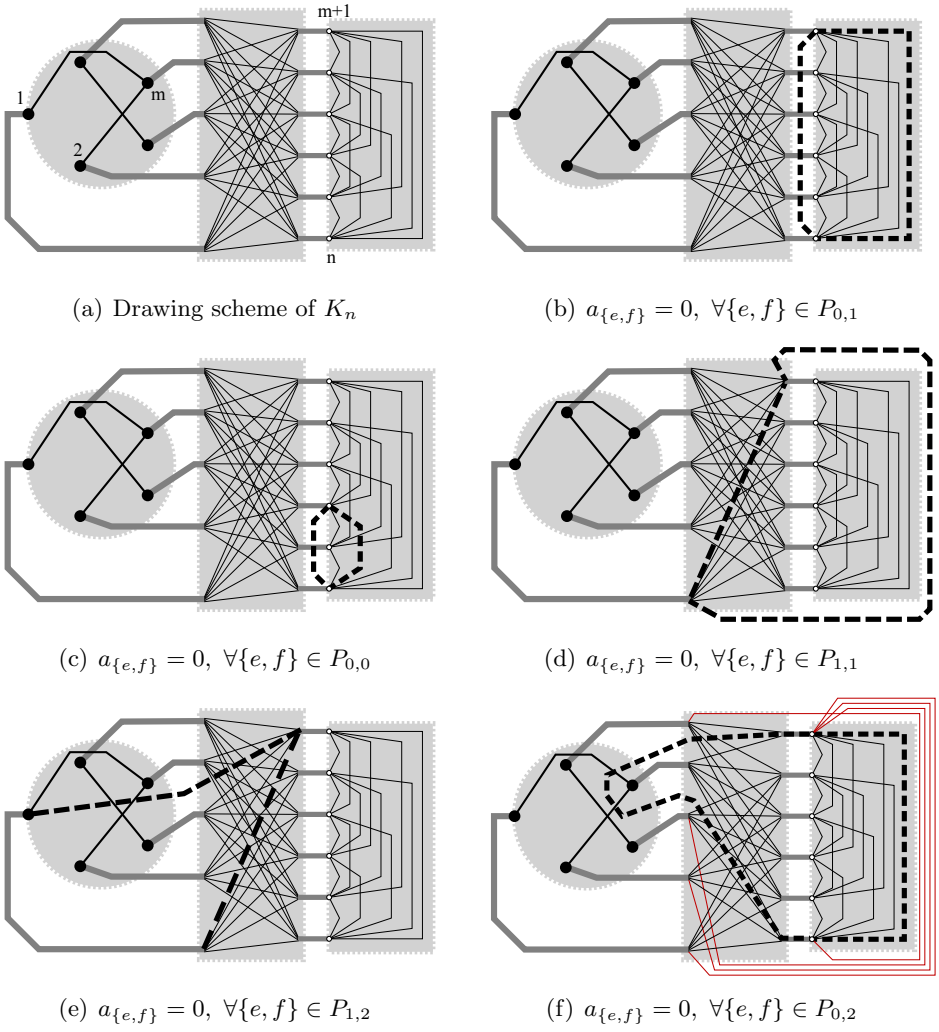
Theorem 5.28. *Fix any $m \geq 5$, $n > m$, and $W \in V_n^{\{m\}}$. C_m^W is a facet in $\mathcal{K}_n$.*

Proof. Again, we consider a first feasible solution as shown in Figure 5.4(a). Assume w.l.o.g. that $W = \{v_1, \ldots, v_m\}$, and that v_1 is a node on the outside of the drawing induced by W.

$\underline{\alpha_{0,1} = 0}$: cf. Figure 5.4(b). We can reroute the S_0 edge $\{v_{m+1}, v_n\}$ which originally has no crossings, such that it crosses over S_1 edges, but no other edges. Hence we have: $\alpha_{0,1} = 0$.

$\underline{\alpha_{0,0} = 0}$: cf. Figure 5.4(c). We can reroute the S_0 edge $\{v_{n-2}, v_n\}$ such that it crosses S_1 edges instead of S_0 edges. We already know for the former, that their coefficients are zero. The introduction of the latter crossings therefore induces: $\alpha_{0,0} = 0$.

$\underline{\alpha_{1,1} = 0}$: cf. Figure 5.4(d). We can reroute the S_1 edge $\{v_1, v_{m+1}\}$ such that it does not cross any S_1 edge anymore, without introducing any new crossings. Therefore we obtain: $\alpha_{1,1} = 0$.



(a) Drawing scheme of K_n

(b) $a_{\{e,f\}} = 0,\ \forall\{e,f\} \in P_{0,1}$

(c) $a_{\{e,f\}} = 0,\ \forall\{e,f\} \in P_{0,0}$

(d) $a_{\{e,f\}} = 0,\ \forall\{e,f\} \in P_{1,1}$

(e) $a_{\{e,f\}} = 0,\ \forall\{e,f\} \in P_{1,2}$

(f) $a_{\{e,f\}} = 0,\ \forall\{e,f\} \in P_{0,2}$

Figure 5.4: K_m-constraints are facet-defining in K_n.

$\underline{\alpha_{1,2} = 0}$: cf. Figure 5.4(e). We can reroute the S_1 edge $\{v_1, v_{m+1}\}$ such that it crosses through the graph induced by W, i.e., through S_2 edges. Thereby, the number of crossings with S_1 edges also changes, but we already know that these coefficients are zero. We therefore have: $\alpha_{1,2} = 0$.

$\underline{\alpha_{0,2} = 0}$: cf. Figure 5.4(f). Finally, we can reroute the crossing free S_0 edge $\{v_{m+1}, v_n\}$ such that it crosses through some S_1 and S_2 edges. Therefore we have to reroute some other S_1 edges beforehand, to avoid crossings of adjacent edges. These rerouted edges may cross each other, but we already know that $\alpha_{1,1} = 0$. Furthermore we know that $\alpha_{0,1} = 0$ and we can therefore deduce: $\alpha_{0,2} = 0$. □

5.4.4 $K_{3,3}$-constraints in $K_{n,m}$

Please refer to Section 5.4.1 for the overall structure of the proof.

Fixing 5.29. Fix any $n, m \geq 3$ with $n + m > 6$, $W \in V_n^{\{3\}}$, and $W' \in V_m'^{\{3\}}$. This induces:

$$\begin{aligned} G &= K_{n,m} \\ C &= C_{3,3}^{W,W'} \quad (K_{3,3}\text{-constraint}) \\ c_{\{e,f\}} &= \begin{cases} 1 & \text{if } e, f \subset (W \cup W'), \\ 0 & \text{else.} \end{cases} \qquad \forall \{e, f\} \in \mathcal{CP}_{n,m} \\ \hat{c} &= 1 \\ \mathcal{F} &= \mathcal{F}_{n,m} \end{aligned}$$

Partitioning 5.30. We can partition all edges $e \in E_{n,m}$ using the ordered index set $\mathcal{I} = \langle 0, 1, 1', 2 \rangle$, based on which nodes they connect:

$$\begin{aligned} e \in S_0 &\iff |e \cap (W \cup W')| = 0, \\ e \in S_1 &\iff |e \cap W| = 1, |e \cap W'| = 0, \\ e \in S_{1'} &\iff |e \cap W| = 0, |e \cap W'| = 1, \\ e \in S_2 &\iff |e \cap (W \cup W')| = 2. \end{aligned}$$

Considering the induced partitions of $\mathcal{CP}_{n,m}$, we have $\dagger = 2$, i.e., $c_{\{e,f\}} = 0$ for all $\{e, f\} \notin P_{2,2}$, and 1 otherwise.

Lemma 5.31 (Permutation classes)**.** *For all $i, j \in \mathcal{I}$, $i \leq j$, we have:*

$$\{e, f\}, \{g, h\} \in P_{i,j} \quad \Longrightarrow \quad a_{\{e,f\}} = a_{\{g,h\}}$$

Proof. W.l.o.g. assume that $W = \{v_1, \ldots, v_3\}$ and $W' = \{v_1', \ldots, v_3'\}$. Consider a feasible solution as shown in Figure 5.5(a). The edges of S_2, over which $C_{3,3}^{W,W'}$ is defined, are drawn with bold lines. Clearly, this solution satisfies

$C_{3,3}^{W,W'}$ with equality. Furthermore, $C_{3,3}^{W,W'}$ has exactly one variable for each possible optimal $K_{3,3}$ solution. We can relabel the nodes W arbitrarily in this drawing, without violating the feasibility of the solution nor the equality of $C_{3,3}^{W,W'}$. The same holds for all permutations of the nodes W'. We do also not violate these properties by any permutation of the labels of the nodes of $V_n \setminus W$ and of the nodes $V'_m \setminus W'$.

These permutations allow us to generate a relabeled drawing such that any edge $e \in S_i$ can play the role of any other edge in the same edge partition set, with all such drawings satisfying $C_{3,3}^{W,W'}$ with equality. Using the analogous arguments, in conjunction with the allowed permutations described above, we obtain that the edges of the same partition sets, and therefore the crossing pairs of the same partition sets are indistinguishable from another. □

Theorem 5.32. *Fix any $n, m \geq 3$ with $n + m > 6$, $W \in V_n^{\{3\}}$, and $W' \in V'_m{}^{\{3\}}$. $C_{3,3}^{W,W'}$ is a facet in $\mathcal{K}_{n,m}$.*

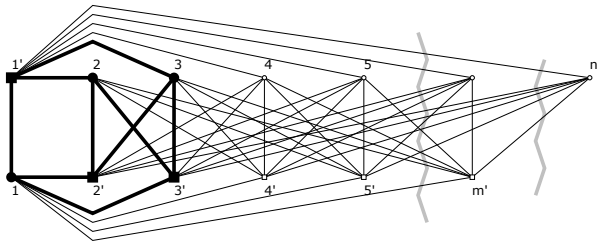
Proof. Again, we assume w.l.o.g. that $W = \{v_1, \ldots, v_3\}$ and $W' = \{v'_1, \ldots, v'_3\}$ and consider a feasible solution as shown in Figure 5.5(a).

$\underline{\alpha_{1,1'} = 0:}$ cf. Figure 5.5(b). Consider two drawings of the edge $\{v'_2, v_n\} \in S_{1'}$ (bold, dashed) corresponding to solutions in X_c: We may route if over the top of the drawing resulting in a single crossing with the edge $\{v_1, v'_1\} \in S_2$. Alternatively, we may route it below the drawing requiring again one crossing with an S_2 edge ($\{v_1, v'_3\}$) but additionally crossings with the S_1 edges $\{v_1, v'_4\}, \ldots, \{v_1, v'_m\}$. Both drawings correspond to elements of X_c and both require exactly one $P_{1,2}$ crossing (with identical coefficients). Hence we have: $\alpha_{1,1'} = 0$.

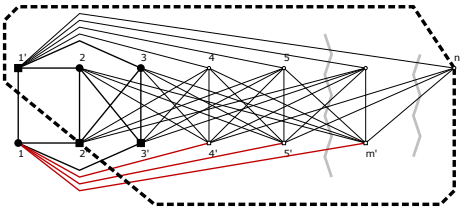
$\underline{\alpha_{1',2} = 0, \alpha_{1,2} = 0:}$ cf. Figure 5.5(c). Consider two drawings of the edge $\{v'_3, v_n\} \in S_{1'}$ corresponding to solutions in X_c: We may route it over the top of the drawing resulting in two crossings with the edges $\{v_1, v'_1\}, \{v_1, v'_2\} \in S_2$. Alternatively, we may route it below the drawing crossing through the S_1 edges $\{v_1, v'_4\}, \ldots, \{v_1, v'_m\}$. For the latter we already know that $\alpha_{1,1'} = 0$, hence we have: $\alpha_{1',2} = 0$.

Consider the analogous feasible drawings where we exchange the node labels of W and W'. Both drawings correspond to solutions in X_c and we obtain $\alpha_{1,2} = 0$ by symmetry.

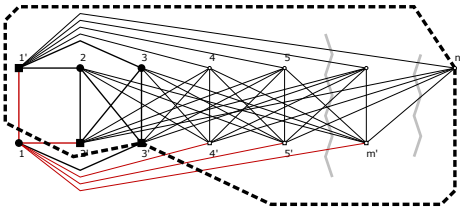
$\underline{\alpha_{1',1'} = 0, \alpha_{1,1} = 0:}$ cf. Figure 5.5(d). Consider two drawings of the edge $\{v'_2, v_n\} \in S_{1'}$ corresponding to solutions in X_c: We may route it over the top of the drawing resulting in a single crossing with the edge $\{v_1, v'_1\} \in S_2$. Alternatively, we may route it between the nodes v_2 and v_3 and between the top two edges, resulting in two crossings with S_2 edges ($\{v'_1, v_3\}, \{v_2, v'_3\}$), crossings with the S_1 edges $\{v_2, v'_3\}$, ...,



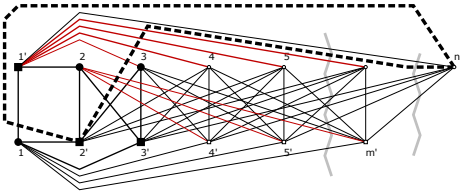
(a) Drawing scheme of $K_{n,m}$



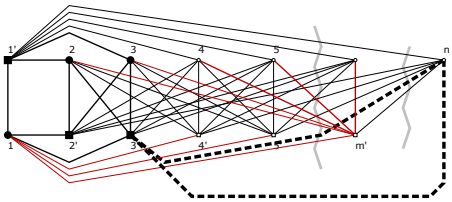
(b) $a_{\{e,f\}} = 0,\ \forall\{e,f\} \in P_{1,1'}$



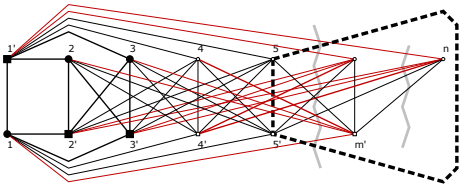
(c) $a_{\{e,f\}} = 0,\ \forall\{e,f\} \in P_{1,2} \cup P_{1',2}$



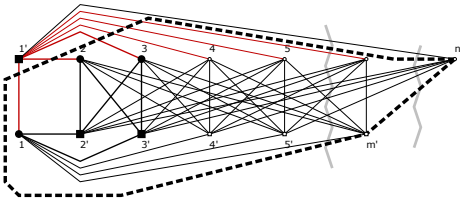
(d) $a_{\{e,f\}} = 0,\ \forall\{e,f\} \in P_{1,1} \cup P_{1',1'}$



(e) $a_{\{e,f\}} = 0,\ \forall\{e,f\} \in P_{0,1} \cup P_{0,1'}$



(f) $a_{\{e,f\}} = 0,\ \forall\{e,f\} \in P_{0,0}$



(g) $a_{\{e,f\}} = 0,\ \forall\{e,f\} \in P_{0,2}$

Figure 5.5: $K_{3,3}$-constraints are facet-defining in $K_{n,m}$.

$\{v_2, v'_m\}$, and crossings with the $S_{1'}$ edges $\{v'_1, v_4\}, \ldots, \{v'_1, v_{n-1}\}$. We already know that the coefficients for the crossings with S_2 and S_1 edges are zero, hence we have: $\alpha_{1',1'} = 0$. Again, we obtain $\alpha_{1,1} = 0$ by symmetry.

$\underline{\alpha_{0,1'} = 0, \alpha_{0,1} = 0:}$ cf. Figure 5.5(e). Consider two drawings of the edge $\{v'_3, v_n\} \in S_{1'}$ corresponding to solutions in X_c: We may route it below rest of the drawing resulting in crossings with the $S_{1'}$ edges $\{v_1, v'_4\}, \ldots, \{v_1, v'_m\}$. Alternatively, we may route it between the nodes v_{m-1} and v_m resulting in: crossings with the $S_{1'}$ edges $\{v_1, v'_4\}, \ldots, \{v_1, v'_{m-1}\}$, two crossings with S_1 the edges $\{v_i, v'_m\}$ for $i = 2, 3$, and crossings with the S_0 edges $\{v_4, v'_m\}, \ldots, \{v_{n-1}, v'_m\}$. We already know that the coefficients for the crossings with $S_{1'}$ and S_1 edges are zero, hence we have: $\alpha_{0,1'} = 0$. Again, we obtain $\alpha_{0,1} = 0$ by symmetry.

$\underline{\alpha_{0,0} = 0:}$ cf. Figure 5.5(f). Consider two drawings of the edge $\{v_5, v'_5\} \in S_0$ corresponding to solutions in X_c: We may route it via a straight line, requiring multiple crossings with S_1, $S_{1'}$, and S_0 edges. Alternatively, we may route it over the right-hand side of the drawing, requiring only some crossings with S_1 and $S_{1'}$ edges. Since we already know that $\alpha_{0,1} = \alpha_{0,1'} = 0$, we have: $\alpha_{0,0} = 0$.

$\underline{\alpha_{0,2} = 0:}$ cf. Figure 5.5(g). Consider two drawings of the edge $\{v'_m, v_n\} \in S_0$ corresponding to solutions in X_c: We may route it via a straight line requiring no crossings at all. Alternatively, we may route it below the drawing to the left-hand side, and back through S_2 and $S_{1'}$ edges. Since we have $\alpha_{0,1} = \alpha_{0,1'} = 0$ for crossings with the latter type of edges, we also have: $\alpha_{0,2} = 0$. □

5.4.5 $K_{3,3}$-constraints in K_n

Please refer to Section 5.4.1 for the overall structure of the proof.

Fixing 5.33. Fix any $n > 6$ and $(W \dot\cup W') \in V_n^{\{6\}}$ with $|W| = 3$. This induces:

$$
\begin{aligned}
G &= K_n \\
C &= C_{3,3}^{W,W'} \quad (K_{3,3}\text{-constraint}) \\
c_{\{e,f\}} &= \begin{cases} 1 & \text{if } |\tilde{e} \cap \tilde{W}| = 1 \\ & \forall (\tilde{e}, \tilde{W}) \in \{e, f\} \times \{W, W'\}, \qquad \forall \{e, f\} \in \mathcal{CP}_n \\ 0 & \text{else.} \end{cases} \\
\hat{c} &= 1 \\
\mathcal{F} &= \mathcal{F}_n
\end{aligned}
$$

Partitioning 5.34. We can partition all edges $e \in E_n$ using the ordered index set $\mathcal{I} = \langle 0, 1, 1', 2, 2', 2'' \rangle$, based on which nodes they connect:

$$\begin{aligned} e \in S_0 &\iff |e \cap (W \cup W')| = 0, \\ e \in S_1 &\iff |e \cap W| = 1, |e \cap W'| = 0, \\ e \in S_{1'} &\iff |e \cap W| = 0, |e \cap W'| = 1, \\ e \in S_2 &\iff |e \cap W| = 2, \\ e \in S_{2'} &\iff |e \cap W| = |e \cup W'| = 1, \\ e \in S_{2''} &\iff |e \cap W'| = 2. \end{aligned}$$

Considering the induced partitions of $\mathcal{CP}_{n,m}$, we have $\dagger = 2'$, i.e., $c_{\{e,f\}} = 0$ for all $\{e, f\} \notin P_{2',2'}$, and 1 otherwise. Note that the set $P_{2,2}$ is empty, since S_2 forms a simple cycle of length 3, and hence all pairs of these edges are adjacent. The same holds for $P_{2'',2''}$.

Lemma 5.35 (Permutation classes)**.** *For all $i, j \in \mathcal{I}$, $i \leq j$, we have:*

$$\{e, f\}, \{g, h\} \in P_{i,j} \quad \Longrightarrow \quad a_{\{e,f\}} = a_{\{g,h\}}$$

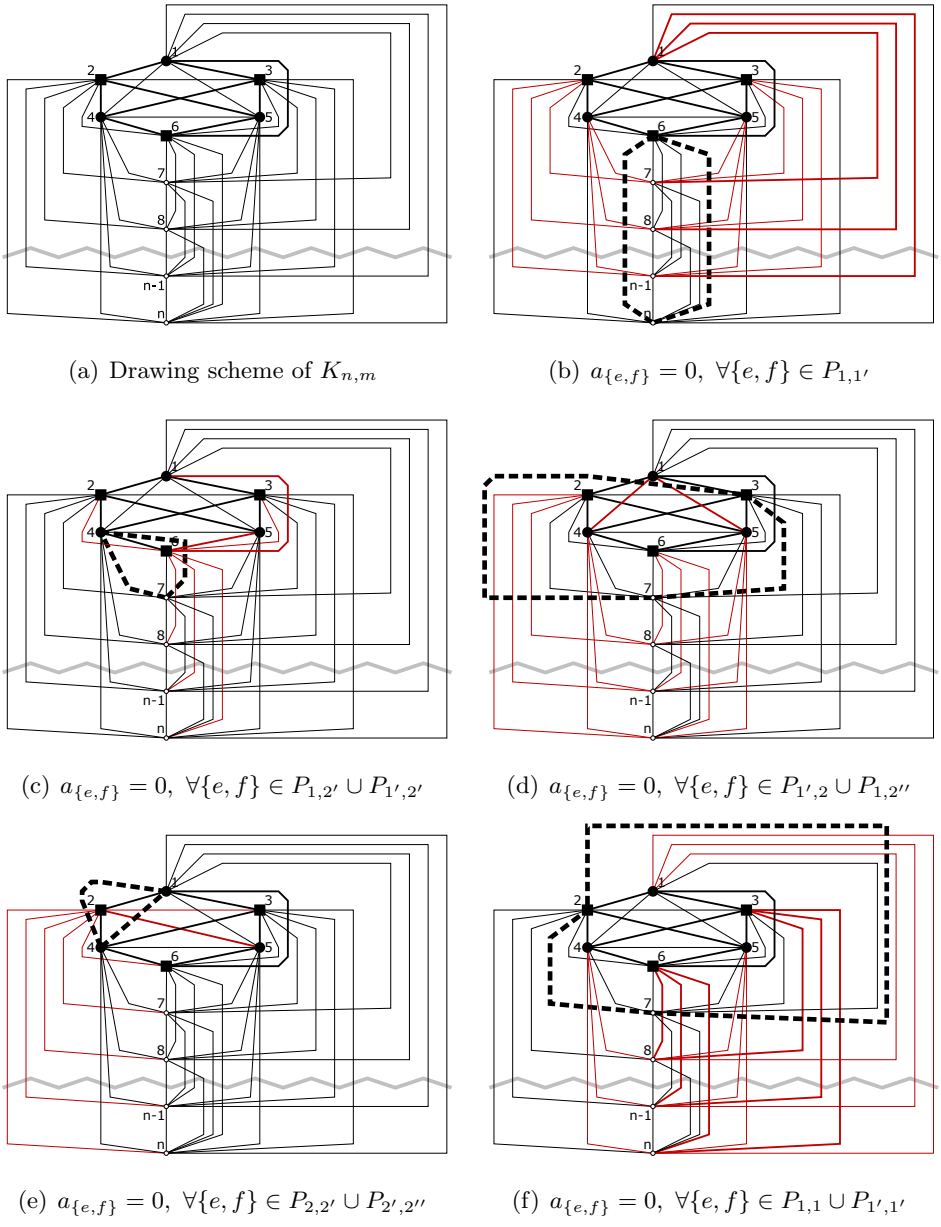
Proof. W.l.o.g. assume that $W = \{v_1, v_4, v_5\}$ and $W' = \{v_2, v_3, v_6\}$. Consider a feasible solution as shown in Figure 5.6(a). The edges of $S_{2'}$, over which $C_{3,3}^{W,W'}$ is defined, are drawn with bold lines.

Clearly, this solution satisfies $C_{3,3}^{W,W'}$ with equality. Furthermore, we already know that $C_{3,3}^{W,W'}$ has exactly one variable for each possible optimal $K_{3,3}$ solution. We can relabel the nodes W arbitrarily in this drawing, without violating the feasibility of the solution, nor the equality of $C_{3,3}^{W,W'}$. The same holds for the nodes W'. We do also not violate these properties by any permutation of the labels of the nodes of $V_n \setminus (W \cup W')$. Using these permutations in conjunction with the arguments of the previous permutation lemmata, we obtain the above lemma. □

Theorem 5.36. *Fix any $n > 6$ and $(W \dot\cup W') \in V_n^{\{6\}}$ with $|W| = 3$. $C_{3,3}^{W,W'}$ is a facet in $\mathcal{K}_n$.*

Proof. Again, we assume w.l.o.g. that $W = \{v_1, v_4, v_5\}$ and $W' = \{v_2, v_3, v_6\}$ and consider a feasible solution as shown in Figure 5.6(a) as our starting point. Again, we call the drawing of the path $v_6, \ldots, v_n$ the *spine* of the drawing.

$\underline{\alpha_{1,1'} = 0}$: cf. Figure 5.6(b). We can redraw the edge $\{v_6, v_n\} \in S_{1'}$ (bold, dashed) on the left side of the spine instead of the right side, and still obtain a solution in X_c. The crossings with the edges $S_{1'}$ edges $\{v_3, v_i\}$ and the S_1 edges $\{v_5, v_i\}$ $(7 \leq i \leq n-1)$ are replaced by the same number of crossings with the $S_{1'}$ edges $\{v_2, v_i\}$ and the S_1 edges $\{v_4, v_i\}$. But the original drawing also requires crossings with the S_1 edges $\{v_1, v_i\}$, which are not necessary in the redrawn drawing. Hence we have: $\alpha_{1,1'} = 0$.



(a) Drawing scheme of $K_{n,m}$

(b) $a_{\{e,f\}} = 0,\ \forall\{e,f\} \in P_{1,1'}$

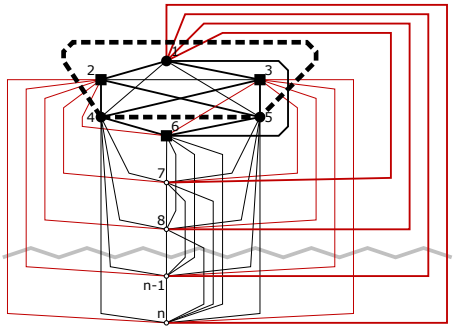
(c) $a_{\{e,f\}} = 0,\ \forall\{e,f\} \in P_{1,2'} \cup P_{1',2'}$

(d) $a_{\{e,f\}} = 0,\ \forall\{e,f\} \in P_{1',2} \cup P_{1,2''}$

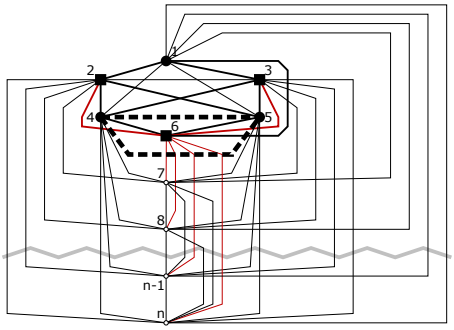
(e) $a_{\{e,f\}} = 0,\ \forall\{e,f\} \in P_{2,2'} \cup P_{2',2''}$

(f) $a_{\{e,f\}} = 0,\ \forall\{e,f\} \in P_{1,1} \cup P_{1',1'}$

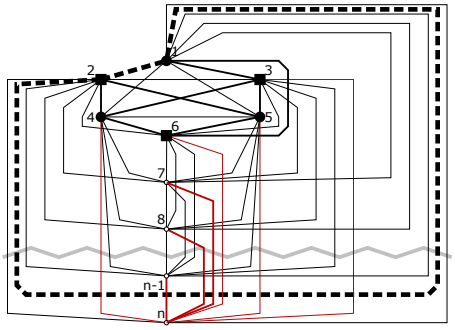
Figure 5.6: $K_{3,3}$-constraints are facet-defining in K_n (part 1).



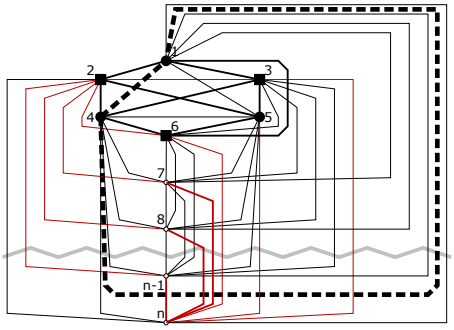
(g) $a_{\{e,f\}} = 0,\ \forall\{e,f\} \in P_{1,2} \cup P_{1',2''}$



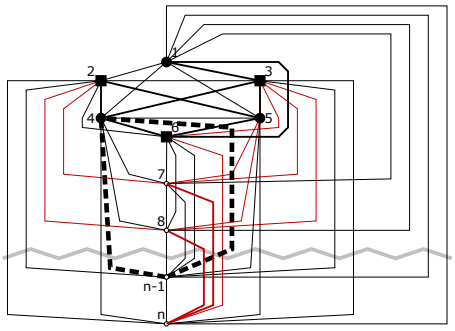
(h) $a_{\{e,f\}} = 0,\ \forall\{e,f\} \in P_{2,2''}$



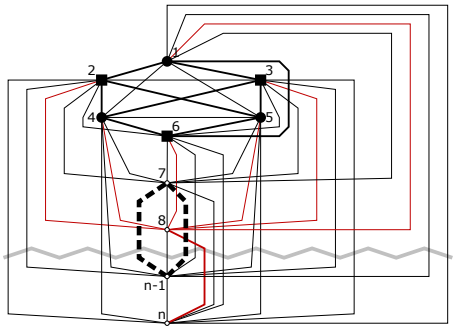
(i) $a_{\{e,f\}} = 0,\ \forall\{e,f\} \in P_{0,2'}$



(j) $a_{\{e,f\}} = 0,\ \forall\{e,f\} \in P_{0,2} \cup P_{0,2''}$



(k) $a_{\{e,f\}} = 0,\ \forall\{e,f\} \in P_{0,1} \cup P_{0,1'}$



(l) $a_{\{e,f\}} = 0,\ \forall\{e,f\} \in P_{0,0}$

Figure 5.6: $K_{3,3}$-constraints are facet-defining in K_n (part 2).

$\alpha_{1,2'} = 0, \alpha_{1',2'} = 0:$ cf. Figure 5.6(c). We can redraw the edge $\{v_4, v_7\} \in S_1$ over v_6 and on the right side of the spine. We still obtain a solution in X_c. The new drawing also requires exactly one crossing with a $S_{2''}$ edge, but additionally it requires multiple crossings with $S_{1'}$ edges adjacent to v_6 and crossings with two $S_{2'}$ edges. For the former new crossings we already know $\alpha_{1,1'} = 0$ and can conclude: $\alpha_{1,2'} = 0$.

Consider the analogous feasible drawings where we exchange the node labels of W and W'. Both drawings correspond to solutions in X_c and we obtain $\alpha_{1',2'} = 0$ by symmetry.

$\alpha_{1',2} = 0, \alpha_{1,2''} = 0:$ cf. Figure 5.6(d). We can redraw the edge $\{v_3, v_7\} \in S_{1'}$ such that it is routed over the top left corner of the drawing, and still obtain a solution in X_c. In both drawings we have the same number of $P_{1,1'}$, $P_{1',1'}$, and $P_{1',2'}$ crossings, but the new drawing requires additional crossings with two S_2 edges. Hence we conclude: $\alpha_{1',2} = 0$. Again, we obtain $\alpha_{1,2''} = 0$ by symmetry.

$\alpha_{2,2'} = 0, \alpha_{2',2''} = 0:$ cf. Figure 5.6(e). We can redraw the edge $\{v_1, v_4\} \in S_2$ such that it is routed over the top left corner of the drawing, and still obtain a solution in X_c. In both drawings there is exactly one crossing with an $S_{2''}$ edge. While the new drawing requires multiple crossings with S'_1 edges, the original drawing crosses an $S_{2'}$ edge. For the former crossings we already know that $\alpha_{1',2} = 0$ and we conclude: $\alpha_{2,2'} = 0$. Again, we obtain $\alpha_{2',2''} = 0$ by symmetry.

$\alpha_{1',1'} = 0, \alpha_{1,1} = 0:$ cf. Figure 5.6(f). We can redraw the edge $\{v_2, v_7\} \in S_{1'}$ such that it is routed over the top right corner of the drawing, and still obtain a solution in X_c. We already know for all crossings $P_{1,1'}$, that $\alpha_{1,1'} = 0$. But the new drawing requires additional crossings with the $S_{1'}$ edges $\{v_3, v_i\}$ and $\{v_6, v_i\}$ for $8 \leq i \leq n$. Hence we have: $\alpha_{1',1'} = 0$. Again, we obtain $\alpha_{1,1} = 0$ by symmetry.

$\alpha_{1,2} = 0, \alpha_{1',2''} = 0:$ cf. Figure 5.6(g). Consider an alternative drawing of Figure 5.6(a) where $\{v_3, v_6\}$ is drawn as a straight line. In this drawing we can redraw the edge $\{v_4, v_5\} \in S_2$ such that it is routed over the top of the drawing. Both drawings correspond to solutions in X_c. In both drawings we have exactly one crossing with an $S_{2''}$ edge. The latter drawing requires additionally (a) a crossing with an $S_{2'}$ edge, (b) crossings with $S_{1'}$ edges, and (c) crossings with S_1 edges. For the crossings of (a) and (b) we already know that their coefficients are 0, hence we conclude: $\alpha_{1,2} = 0$. Again, we obtain $\alpha_{1',2''} = 0$ by symmetry.

$\alpha_{2,2''} = 0:$ cf. Figure 5.6(h). We can redraw the edge $\{v_4, v_5\} \in S_2$ such that it is routed below v_6, and still obtain a solution in X_c. While the original drawing requires no crossings on this edge, the new drawing

crosses multiple $S_{1'}$ edges, a single $S_{2'}$ edge, and two $S_{2''}$ edges. For the former crossings we already know that $\alpha_{1',2} = \alpha_{2,2'} = 0$ and can conclude: $\alpha_{2,2''} = 0$.

$\alpha_{0,2'} = 0$: cf. Figure 5.6(i). We can redraw the edge $\{v_1, v_2\} \in S_{2'}$ such that it is routed just above v_n, and still obtain a solution in X_c. While the original drawing requires no crossings on this edge, the new drawing crosses two S_1 and two $S_{1'}$ edges. Furthermore, it crosses multiple S_0 edges. As we already know that $\alpha_{1,2'} = 0$, we can conclude: $\alpha_{0,2'} = 0$.

$\alpha_{0,2} = 0, \alpha_{0,2''} = 0$: cf. Figure 5.6(j). We can redraw the edge $\{v_1, v_4\} \in S_2$ such that it is routed just above v_n, and still obtain a solution in X_c. For all involved crossings in both drawings we already know that their coefficients are 0, except for the crossings with S_0 edges in the redrawn drawing. Hence we conclude: $\alpha_{0,2} = 0$. Again, we obtain $\alpha_{0,2''} = 0$ by symmetry.

$\alpha_{0,1} = 0, \alpha_{0,1'} = 0$: cf. Figure 5.6(k). We can redraw the edge $\{v_4, v_{n-1}\} \in S_1$ above v_6 and on the right side of the spine, and still obtain a solution in X_c. For all involved crossings in both drawings we already know that their coefficients are 0, except for the crossings with S_0 edges in the redrawn drawing. Hence we conclude: $\alpha_{0,1} = 0$. Again, we obtain $\alpha_{0,1'} = 0$ by symmetry.

$\alpha_{0,0} = 0$: cf. Figure 5.6(l). We can redraw the edge $\{v_7, v_{n-1}\} \in S_0$ on the left side of the spine instead of the right side, and still obtain a solution in X_c. For all involved crossings in both drawings we already know that their coefficients are 0, except for the crossings with S_0 edges in the redrawn drawing. Hence we conclude: $\alpha_{0,0} = 0$. □

This proof concludes the section, in which we showed that the considered Kuratowski constraints indeed define facets in the complete and complete bipartite crossing number polytopes.

5.5 Comparison between the ILPs

It is hard to directly compare the theoretical strengths of our formulations SECM and OECM. It is not possible to project fractional OECM solutions into the SECM variable space. Even projecting SECM solutions into OECM is non-trivial. We can straight-forwardly project both variable spaces into RECM's x-variable space as described in the beginning of the previous section. But there we lose all knowledge of the respective edge orderings over the crossings, which is the main difference between our formulations.

To show that one formulation is stronger than the other, we would have to find a feasible fractional solution of one and show that this solution is

infeasible for the other formulation. Yet, we did not succeed with finding such examples, since the exponential number of Kuratowski subdivisions makes a thorough analysis very complicated. In particular, it is hard to manually consider all the complex constraints where there are variables on the right-hand side, i.e., $\mathcal{CP}'(K) \cap \mathcal{R}' \neq \emptyset$ for SECM or $(\mathcal{X}_K[\bar{x}, \bar{y}], \mathcal{Y}_K[\bar{x}, \bar{y}]) \neq (\emptyset, \emptyset)$ for OECM. An in-depth theoretical comparison of the approaches remains as an open problem.

Hence we divert the main comparison between the two formulations to the experimental section, were we will see that OECM turns out to be clearly stronger than SECM in practice. This is mainly due to the fact, that OECM is in general more compact than SECM: It requires only $\mathcal{O}(|E|^3)$ variables while SECM requires $\mathcal{O}(|E|^2 \cdot \ell^2) = \mathcal{O}(|E|^4)$ since the crossing number, and therefore also the upper bound ℓ, can be quadratic in the number of edges. This is amplified by the fact that OECM's variable structure allows less symmetric solutions and a greedier column generation, cf. Section 8.3. Yet, SECM has benefits regarding its adaptability for alternative crossing number concepts, cf. Chapter 7.

Chapter 6

Solving the 0/1-ILPs

crossing: *(Sports)*
The movement of a horse which begins from one of the positions out wider on the track and moves down to the inside fence, is referred to as a crossing to the fence.

In this section, we will discuss how the exponentially sized SECM and OECM formulations presented in the previous section can be solved in practice. Therefore, we employ a branch-and-cut-and-price scheme, as introduced in Section 3. We start by outlining the general structure of the algorithms and their common—or similar—sub-steps like the separation routines and branching strategies. Thereby we will focus on a pure branch-and-cut framework, i.e., we will ignore the fact that certain variables may be dynamically generated. In the sections thereafter, we will discuss the specializations for SECM and OECM, in particular we will describe the different column generation schemes for the ILPs. Again, we are given a graph $G = (V, E)$ with integer edge weights w in the following.

6.1 Common Framework

Algorithm 2 gives an overview on the general branch-and-cut(-and-price) framework used for our algorithms. Since the column generation routines are highly ILP specific we only mark the code lines where the corresponding routines have to be inserted. For the descriptions in this section, we can safely assume that no column generation is performed and all necessary variables are active.

Our algorithms starts (line 1–3) with obtaining initial upper bounds by using PRIMALHEURISTIC. In our implementation, this is the efficient planarization heuristic described in [76], which starts with a planar subgraph

Algorithm 2 Branch-and-cut algorithm for exact crossing minimization. By χ we denote the variables of the 0/1-ILPs which may be either z or (x, y).

Require: 2-connected, non-planar graph $G = (V, E)$ with integer edge weights w.
Ensure: crossing number $\mathrm{cr}(G)$ and a planarization G^* of G realizing $\mathrm{cr}(G)$.

var U: global upper bound
var $\bar{\chi}_U$: feasible integer variable assignment realizing U
$U, \bar{\chi}_U \leftarrow$ PrimalHeuristic(G)
var *Open*: list of open subproblems (initially empty)
Open.push(**new** Subproblem($\mathcal{J}$)) ▷ first problem with initial constraints $\mathcal{J}$
while *Open* not empty **do**
    $S \leftarrow$ extract subproblem from *Open*
    loop
        var L: local lower bound
        var $\bar{\chi}$: fractional values for the variables χ
        $L, \bar{\chi} \leftarrow$ solve LP relaxation of S
        if $\lceil L \rceil \geq U$ **then break** ▷ subproblem can be pruned
        constraints $\leftarrow$ CyclicOrderSeparation($\bar{v}$) ▷ Oecm specific
        if *constraints* not empty **then**
            add *constraints* to S
            continue ▷ restart loop
        end if
        (⇝ insert Combinatorial Column Generation code)
        $\tilde{\chi} \leftarrow$ Round($\bar{\chi}$)
        $P \leftarrow$ (partial) planarization induced by $\tilde{\chi}$.
        $H, \bar{\chi}_H \leftarrow$ PrimalHeuristic(P) ▷ local upper bound
        if $H < U$ **then** $U, \bar{\chi}_U \leftarrow H, \bar{\chi}_H$ ▷ new global upper bound found
        if $\lceil L \rceil \geq H$ **then break** ▷ no more room for improvement
        constraints $\leftarrow$ KuratowskiSeparation($\bar{\chi}$, $\tilde{\chi}$, P)
        if *constraints* not empty **then**
            add *constraints* to S
            continue ▷ restart loop
        end if
        (⇝ insert Algebraic Pricing code)
        $B \leftarrow$ select variable or constraint to branch on
        $b_0, b_1 \leftarrow$ the two fixings or constraints induced by branching on B
        Open.push(**new** Subproblem(S, b_1))
        Open.push(**new** Subproblem(S, b_2))
        break ▷ leave loop
    end loop
end while
$\mathrm{cr}(G) \leftarrow U$
$G^* \leftarrow$ planarization induced by $\bar{\chi}_U$

and inserts the edges one after another, see Part III of this thesis for details on the chosen parameters in our various experiments.

We then initialize the root of our branch-and-bound tree and store it as the only element in the list *Open* (line 4–5): This first subproblem is basically the full problem, without any restrictions other than the constraints in the ILP. If no column generation is used, we can assume that the initial constraint set $\mathcal{J}$ includes all variable bounds, and all simplicity constraints (5.6) for SECM or all linear ordering constraints (5.11)–(5.13) for OECM, respectively. Note that we do not add the $\mathcal{O}(|E|^4)$ cyclic-order constraints (5.14); we will add them via a separation routine later.

Furthermore, we start with a subset of Kuratowski constraints: We identify a certain number of Kuratowski subdivisions in G using the separation routine (see below), and add the corresponding inequalities to $\mathcal{J}$.

The main while loop (line 6–32) handles the various subproblems that may occur in the course of the computation. In particular, in line 7 we select one of the open, i.e. not yet considered, subproblems in the list *Open*; usually we use a *best search* strategy, i.e., we select the subproblem with the most promising dual bound.

The loop between line 8 and 31 processes a specific subproblem S. We iteratively solve the LP relaxation to obtain a dual (lower) bound L and check whether S might give us a new solution that is better than the primal (upper) bound U we already know (line 9–12). Since the edge weights are integral, we know that the optimal solution will be integral as well and we can therefore consider L rounded up.

If the subproblem is still promising, we proceed. In the case of OECM, we want to ensure that the fractional solution is LO-feasible, even though we did not add all cyclic-order constraints (5.14). Therefore (block at line 13), we simply check all inactive such constraints and add them if they are violated by the current solution. If we did identify violated constraints we resolve the LP relaxation.

Otherwise, we continue with the inspection of the current solution: We round our fractional solution (if it is not already integer), and realize the crossings specified by this assignment to obtain a partial planarization P (line 15–16; see Section 6.1.1 below for a detailed description of the applied rounding schemes). P contains dummy nodes modeling the crossings in this rounded solution. We can then perform our PRIMALHEURISTIC on P (line 17). The union of the crossings modeled by dummy nodes in P and the crossings introduced by the heuristic then constitutes a full planarization of G, and therefore a feasible solution. Note that P may be planar, in which case the primal heuristic will return the unmodified graph P as its solution. If the new solution is better than our current global upper bound U, we update the latter (line 18–19). To avoid spending too much time on computing heuristics, in practice we only perform the lines 17–19 if the obtained rounded solution is different from the one obtained in the prior iteration.

If the subproblem is not pruned yet, we start our separation routine KuratowskiSeparation (line 20) which tries to identify violated inactive Kuratowski constraints, see Section 6.1.2. If the routine finds such constraints (line 21–24) we add them to our current subproblem and reiterate the process, i.e. solve the LP relaxation, check for bound clashes, etc.

If no constraints were identified, we have to resort to branching (line 26–29). We will only consider 2-branches, i.e., we generate exactly two new subproblems for each branch. We first select a variable or constraint on which to branch and compute the two constraints arising from this branch. We then generate two new subproblems which are initialized with the constraints in S and the additional branching constraint. We add the subproblems to our list *Open* of open subproblems and stop considering the current subproblem S. See Section 6.1.3 for details on the branching strategies.

Finally (line 33–34), when all open subproblems have been considered, our upper bound U will constitute the optimal solution of our ILP, i.e., the crossing number of G. We can return this number, as well as the integer feasible variable assignment realizing $\mathrm{cr}(G)$ and the thereby induced planarization G^*.

We will now discuss rounding, Kuratowski separation, and branching in more detail, as they were only sketched above.

6.1.1 Rounding the Fractional Solution

Rounding a fractional solution $\bar{z}$ of Secm is rather straight-forward. Generally, we can establish various different rounding schemes, but preliminary experiments showed that their influence is rather weak [49]. Hence we use a simple *threshold rounding*, i.e., we fix a threshold $0.5 < \tau < 1$ and have:

$$\tilde{z}_{\{e',f'\}} = \begin{cases} 1 & \text{if } \bar{z}_{\{e',f'\}} \geq \tau, \\ 0 & \text{else.} \end{cases} \qquad \forall \{e', f'\} \in \mathcal{CP}'.$$

The rounding of $(\bar{x}, \bar{y})$ for Oecm is somewhat more complex, since we require that the rounded solution $(\tilde{x}, \tilde{y})$ is LO-feasible. We proceed in two steps; first, we round the x-variables analogously to before:

$$\tilde{x}_{\{e,f\}} = \begin{cases} 1 & \text{if } \bar{x}_{\{e,f\}} \geq \tau, \\ 0 & \text{else.} \end{cases} \qquad \forall \{e, f\} \in \mathcal{CP}.$$

Based on $\tilde{x}$, we can then restrict the set of $\tilde{y}$-variables that may be 1. For each edge e, let F_e be the set of edges which cross e, according to $\tilde{x}$. We can set $\tilde{y}_{e,f,g} = 0$ for all variables with $\{f, g\} \not\subseteq F_e$. If $|F_e| \geq 2$, we define a complete bidirected weighted graph, using F_e as its vertex set. We choose the weight of an arc (f, g) as $\bar{y}_{e,f,g}$. Then we solve the linear ordering problem on this graph, using a straight-forward greedy heuristic [44]. Using the resulting order, we can decide the values for $\tilde{y}_{e,f,g}$, for all $\{f, g\} \subseteq \mathcal{D}_e$.

In our experiments, we usually generate two rounded solutions for each fractional solution: one for $\tau = 0.7$ and one for $\tau = 1 - \tilde{\varepsilon}$ (for some small $\tilde{\varepsilon} >$

0). The subsequent steps of computing a heuristic solution and performing the separation (see below) is then performed twice, once for each rounded solution.

See Remark 6.1 in the next section for a rationale why, e.g., a pure randomized rounding of the variables would be unsuitable.

6.1.2 Kuratowski Separation

In the following, we will concentrate on identifying violated Kuratowski constraints either in SECM or OECM. It is an open problem whether there exists an exact polynomial-time separation routine for Kuratowski constraints in either of the two formulations. Even for the supposedly simpler Kuratowski constraints in the context of maximum planar subgraphs, no such routine is known[1] [92, 93].

Hence, we try to identify violated constraints heuristically: Any violated constraint we find is valid, but we may not find any violated constraint even though there is one in the full ILP. Thereby, our algorithm is still valid, gives the optimal solution, etc., but we may start to branch before it would theoretically be necessary and hence our algorithm may have to consider more subproblems than necessary with an exact separation routine.

Our heuristic works as follows: We consider the planarization P constructed from the rounded solution—if we have multiple rounded solutions, we perform the separation multiple times, once for each rounded solution. We then try to identify Kuratowski subdivisions in P. For each identified subdivision we construct the corresponding Kuratowski constraint (5.7) or (5.15) for SECM or OECM, respectively. Using the fractional solution of the LP relaxation, we can easily check if the constraint is currently violated. If it is, we have identified a constraint $C := p^T\chi \geq b$ (with variables χ and coefficients p) with violation $\phi_C = b - p^T\bar{\chi}$.

We iterate this process to obtain multiple violated constraints. In the end, we may not add all identified constraints but only a certain number of the most violated constraints and/or only ones where ϕ_C is above a certain threshold. See Part III for the experiment-specific parameters.

Remark 6.1. Recall the rounding scheme described in Section 6.1.1. We can now easily see why certain other rounding schemes are unsatisfactory in our context. Consider the pure randomized rounding, where a variable is rounded to 1 with the probability given by its fractional solution. A constraint based on a Kuratowski which contains more than one dummy crossing of P which has a fractional solution of 0.5 can never be violated, and we would perform many useless extractions. Even worse, on average, every 5th variable with

[1] To identify a maximum planar subgraph via a 0/1-ILP, we have a variable $x_e \in \{0, 1\}$ for each edge e, maximize $\sum_e x_e$, and for each Kuratowski subdivision K in the graph we have $\sum_{e \in K} x_e \leq |K| - 1$, i.e., at least one edge has to be removed. See [113] for details.

value 0.2 would be rounded to 1. A Kuratowski constraint containing only one such variable will already be highly unlikely to be violated.

It remains to discuss how to identify a Kuratowski subdivision in P. The simplest variant—only requiring any planarity testing algorithm—removes edges from P as long as P remains non-planar. If P becomes planar the edge is added again, and the next one is deleted, etc; cf. Algorithm 3. In the end, a Kuratowski subdivision will remain. Clearly, this simple algorithm denoted by SimpleExtract requires $\mathcal{O}(|V(P)|^2)$ time, since planarity testing for graphs with $|E(P)| > 3 \cdot |V(P)| - 6$ can be performed in constant time as these graphs can never be planar, and in linear time otherwise.

Algorithm 3 SimpleExtract: Extraction of a single Kuratowski subdivision in quadratic time.

Require: non-planar graph P
Ensure: a Kuratowski subdivision K contained in P
1: $K \leftarrow P$
2: **for each** $e \in E(P)$ **do** ▷ in random order
3: $K \leftarrow K - e$
4: **if** K is planar **then**
5: $K \leftarrow K + e$
6: **end if**
7: **end for**

Anyhow, as noted in Section 2.2, state-of-the-art planarity testing algorithms like the ones by de Fraysseix et al. [61] or Boyer and Myrvold [20] extract such a subdivision directly in linear time $\mathcal{O}(|V(P)|+|E(P)|)$ within a single run of the planarity testing algorithm. Note that for the extraction, we have to consider the case that the number of edges is larger than the number of nodes.

In our context, we are interested in obtaining several Kuratowski subdivisions in P. Hence would we have to run the planarity algorithm several times, perturbing its outcome, e.g., by randomizing the initial DFS-traversal. Note that after k runs, we are not guaranteed to have obtained k distinct subdivisions.

Efficient Extraction of Multiple Kuratowski Subdivisions. Using the testing and extraction algorithm by Boyer and Myrvold as a starting point, we develop an algorithm with optimal linear running time which can extract multiple distinct Kuratowski subdivisions in one run [39]. This algorithm is quite involved and requires a thorough understanding of the original planarity testing algorithm. Although this algorithms constitutes a work by this thesis' author together with Petra Mutzel and Jens M. Schmidt, it would be beyond the focus of this thesis. We will only very briefly sketch its main ideas herein and refer the interested reader to the original publications [39, 38, 133] (in increasing order of extensiveness) for more details.

The original Boyer-Myrvold algorithm starts with a DFS-tree D of P, which is trivially planar. It then iterates over the tree's nodes in decreasing order of their DFS-index. For each node v it tries to embed all backedges that start below v (i.e., at a node with larger index) and end at v. Thereby *embedding a backedge* e means to include it in D. Since e will be part of a cycle, its addition may merge several blocks in D into a new, larger block. By carefully choosing the order in which the backedges ending at v are embedded, we can establish the property that the embedding of a block is never changed.

If at some point the embedding of a backedge fails, this is because its addition would block the path for backedges which should be added later on. Hence the algorithm stops; by careful investigation of this *stopping configuration*, we can identify a witness for the graph's non-planarity, i.e., a Kuratowski subdivision. Clearly, the extraction of a subdivisions may take linear time in the number of nodes and edges of P without influencing the algorithm's overall runtime analysis.

The challenges with developing an algorithm to efficiently extract multiple subdivisions is therefore three-fold. Surprisingly, all these can be solved, at the only cost of a more involved and harder to prove algorithm:

- *After identifying a stopping configuration: Can we extract multiple distinct Kuratowski subdivisions from it?*
  Yes, there may be multiple matching minor types (i.e., patterns how the witness is contained in the graph), and each minor type may allow multiple realization to be found in P.

- *To obtain an output-sensitive algorithm, the runtime for the extraction of a subdivision has to be bounded by its own size, instead of the size of the given graph.*
  While there are examples where the time for extracting a single subdivision cannot be bounded by its size, it can be shown that we can amortize the costs over all extracted subdivisions at the same crossing configuration.

- *After extracting subdivisions at some stopping configuration: Can we proceed with the original algorithm to arrive at further stopping configurations, extract additional subdivisions, etc., until arriving at the original DFS-tree's root node?*
  After arriving at some stopping configuration Ω, the algorithm cannot proceed as it was, since further edge embeddings would not be possible. Therefore, after the extractions at Ω, we remove the backedge whose attempted embedding established Ω. The central problem is that such a modification invalidates all the algorithm's inner data structures; recomputing them would require linear time. It can be shown that by extending the data structures by a constant factor, we can continue the algorithm and re-validate the data structures on the fly, without

increasing the asymptotic runtime performance.

We end up with an algorithm MultiExtract which is optimal in terms of its asymptotic running time: It is linear in the input size and its required output.

Theorem 6.2 ([39]). *The overall running time of* MultiExtract *is*

$$\mathcal{O}\left(|V(P)| + |E(P)| + \sum_{K \in \mathcal{K}} |E(K)|\right),$$

where $\mathcal{K}$ is the set of all extracted Kuratowski subdivisions.

The identified paths of a Kuratowski subdivision will in general go through multiple blocks in D. By enumerating multiple paths per blocks, we can further increase the number of identified subdivisions. These are similar but distinct to the subdivisions identified before. Though, this algorithm denoted by MultiExtractBundle comes at the cost of an additional log-factor:

Theorem 6.3 ([39]). *The overall running time of* MultiExtractBundle *is*

$$\mathcal{O}\left(|V(P)| + |E(P)| + \log(|V(P)|) \cdot \sum_{K \in \mathcal{K}} |E(K)|\right).$$

6.1.3 Branching Strategies

Generally, we use two different branching strategies:

Branching on K_5-constraints. We can use Kleitman's parity argument for complete graphs with an odd number of vertices [96, 97]: If a K_{2n+3}, $n \in \mathbb{N}$, has an even or odd crossing number, every possible drawing of K_{2n+3} will also have an even or odd number of crossings, respectively. Since we know that $\mathrm{cr}(K_5) = 1$, we have for every K_5-subdivision that if it is drawn with more than one crossing, it will require at least 3 crossings.

This jump in the crossing number can be used for branching. We check for any K_5-constraint of the type $p^T\chi \geq 1$, with p and χ being the vectors of the coefficients and variables, respectively. We can then generate two subproblems, one with $p^T\chi = 1$ and one with $p^T\chi \geq 3$. Note that, theoretically, we can continue to branch on the latter constraint, generating $p^T\chi = 3$ and $p^T\chi \geq 5$, etc.

Branching on Variables. If our current model does not include any K_5-constraint on which we could branch, we resort to common *variable branching*. We select a variable χ_i, which has a current fractional value $\bar{\chi}_i$ close to 0.5 and a high coefficient in the objective function. We can then generate two subproblems with the constraints $\chi_i = 0$ and $\chi_i = 1$, respectively.

6.1.4 Column Generation

By introducing all variables of the ILP in the first subproblem, as assumed above, we obtain a pure branch-and-cut algorithm for SECM and OECM. Yet, the high number of variables—recall that SECM requires $\mathcal{O}(|E|^4)$ and OECM requires still $\mathcal{O}(|E|^3)$ variables—becomes a bottleneck when solving even medium sized problems. See Chapter 8 for an experimental demonstration. Furthermore, we know that in the final solution, only few of the variables will be non-zero. E.g., we know for SECM that the optimal solution will set at most $\mathcal{O}(|E|^2)$ variables to 1, for any given graph.

Based on these arguments, we will discuss in the following how to turn the above branch-and-cut algorithm into a branch-and-cut-and-price algorithm, i.e., we will generate variables during the computation. As briefly discussed in Chapter 3, we will use two different approaches for this task:

Algebraic Pricing: The most common method for column generation is based on the Dantzig-Wolfe decomposition theorem [42]. The key idea is to compute the *reduced costs* of the inactive variables, and generate a variable with (in the case of minimization problems) a highly negative reduced cost. On first sight, this is particular attractive in our case, where there is only a polynomial number of variables to check.

Combinatorial Column Generation: We will see that we can heavily improve on the above concept with a tailored approach where we identify potentially beneficial new columns based on combinatorial arguments, instead of algebraic computations.

The central idea thereby is to recognize the main reason why we could not simply use the RECM formulation, only having one x-variable for each edge pair: The substitution with z-variables for SECM, and the introduction of y-variables for OECM are only needed to decide the order of crossings on edges. But for most real-world graphs, cf. Chapter 8, most edges are involved in at most 1 crossing; and is it very seldom that a single edge is involved in many crossings.

Hence, our combinatorial column generation schemes will allow *unclear* solutions, i.e., solutions where the crossings on some edges are not properly ordered. Such a solution cannot be used to obtain a partial planarization, nor can we deduce Kuratowski subdivisions from it. Hence, before we try to interpret the solution in any such way, we will add exactly the variables to resolve the unordered situations.

The order of the different schemata given in this thesis reflects the historical development: For SECM, we first developed an approach based on the traditional algebraic pricing method. We later improved on this by a combinatorial column generation strategy and investigated the advantages of the latter experimentally. When developing OECM, we concentrated on the combinatorial column generation scheme right from the beginning.

Bounding. Algorithm 2 already shows the conceptual places where the additional steps for the column generation schemata will be performed. Yet note that by introducing algebraic pricing, we loose certain bounding properties known from branch-and-cut algorithms: Mainly, the solution of the LP relaxation is not a proper lower bound for the problem, since the addition of additional variables might allow smaller objective values. Therefore the bounding schemes have to be adopted accordingly.

We will see in the subsequent sections, that the combinatorial column generation schemes deal with this problem by guaranteeing that the LP solution can still be rounded to a valid lower bound as in the given code.

6.2 Algebraic Pricing for SECM

When performing algebraic pricing, we have to identify at least one inactive variable whose activation could decrease the objective function. This can be tested by computing the *reduced cost* of each inactive variable: Let p be the vector of coefficients the inactive variable z_i would have (when added) in the currently active constraints, and let d be the vector of the dual variables of the current solution. The reduced cost $r(z_i)$ of z_i is simply defined as the scalar product of these vectors. If $r(z_i) \geq 0$ (in a minimization problem), we know that the addition of z_i cannot lead to better solutions. On the other hand, if $r(z_i)$ is negative it could be necessary to add z_i to the current model to guarantee overall optimality.

This strategy may give false positives, hence we only add a certain number of variables with the smallest reduced costs first. Their addition may then lead to the rejection of other variables in the next iteration, which had a negative reduced cost before.

Due to the structure of our specific problem, we actually do not need to test all inactive variables. It suffices to test variables which "extend" the currently introduced edge segments. More technically, this means that we start with a quadratic number of variables: For all pairs $\{e, f\} \in \mathcal{CP}$, we label one of their expanded segments with e_0 and f_0, and add the variable $z_{\{e_0,f_0\}}$ representing a crossing between these segments. In the pricing step, we have only to consider variables that are *convex extensions* of the active variables. We can add variables for a segment left or right to the segment for which a variable already exists, leading to positive and negative indices. In the first iteration, we will have to test the variables $z_{\{e_{\pm 1},f_0\}}$ and $z_{\{e_0,f_{\pm 1}\}}$. Figure 6.1 shows the situation at some later iteration step: The indices of segments of e are plotted on the horizontal axis, the indices of the segments of f on the vertical. The gray rectangles mark active variables; the circles denote potential new variables. The crosses mark variables which, although adjacent to the current set of realized variables, are not tested: They would extend the variable set in a way that is not orthogonally convex. Note that

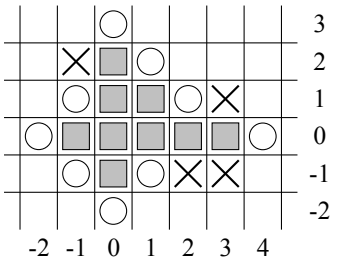


Figure 6.1: Algebraic Pricing for SECM. We allow only orthogonally convex extensions of the active variable set, cf. text.

the orthogonally convex extensions are sufficient, and crossed out variables can be realized by first extending via the variables "in between".

Recall that the number of segments per edge is restricted to $\ell - 1$ (cf. Remark 5.6). Clearly, we will not test any variables which would induce a span of more that $\ell - 1$ between the smallest and the largest segment index per edge. Also note that we do not add all simplicity constraints (5.6) to the initial subproblem. We always add such a constraint together with the first variable which has a non-zero coefficient in it.

6.3 Combinatorial Column Generation for SECM

Of any edge e, we will denote its segments $E'(e)$ by $\langle e_1, e_2, \ldots, e_{\ell-1} \rangle$, considering their natural order (recall Remark 5.6 why we require at most one segment less than ℓ). We say e_1 is the *primary* segment of e, while the others are its *secondary* segments.

We start with a quadratic number of active variables, representing crossings between the primary segments of the edges, i.e., we initially activate all $z_{\{e_1,f_1\}}$ for all $\{e, f\} \in \mathcal{CP}$. Disregarding the fact that subdividing the graph is necessary for efficient realizability testing (and subsequently primal heuristics and separation), we want to be able to describe a feasible solution with this restricted variable set, similar to the x-variables of RECM. Therefore, we will not introduce the simplicity constraints (5.6) for the primary segments. We call these unused constraints *imaginary simplicity constraints*.

When activating a variable later, which uses some secondary segment e_i $(1 < i < \ell)$ for the first time, we will introduce the simplicity constraint for e_i to the current model.

When solving the initial subproblem, we may find a fractional solution $\bar{z}$ that violates an imaginary simplicity constraints C_e on the segment e_1. Recall that we will use $\bar{z}$ only to generate a rounded solution $\tilde{z}$ which will then in turn be used to generate a planarization P. The only problem arises when there are multiple rounded variables which are all 1 and share the same primary segment: their order would be unclear, we cannot interpret the

solution as a partial planarization in a meaningful way, and hence our cutting plane approach could not be used. Note that this argument is based on the rounded values $\tilde{z}$, not on the fractional solution $\bar{z}$ itself. In fact, we do not have to care if an imaginary constraint is violated due to the sum of some variables, if at most one of these variables is 1 in the rounded solution: We can still stablely round this fractional solution into an unambiguous integer solution.

Our column generation scheme simply checks each imaginary simplicity constraint C_e. Let F_e be the set of segments f' with $z_{\{e_1,f'\}} \geq \tau$; we use the notation $F_e = \left\{f^{(1)}_{\gamma(1)}, f^{(2)}_{\gamma(2)}, \ldots, f^{(k)}_{\gamma(k)}\right\}$, where $f^{(i)}_{\gamma(i)}$, $1 \leq i \leq k$, is the $\gamma(i)$-th segment of the edge $f^{(i)}$. If $k \geq 2$, we have to generate new variables. For all $1 \leq i \leq k$, let $\kappa(i)$ be the largest index for which the variable $x_{\{e_{\kappa(i)},{f^{(i)}}_{\gamma(i)}\}}$ is active. We will add the variables $x_{\{e_{\kappa(i)+1},{f^{(i)}}_{\gamma(i)}\}}$ for all i.

By the introduction of these variables, we allow that instead of having all these segments cross over e_1, at least one of them can be routed via the newly activated secondary segment.

It is essential to add an incentive for the ILP to prefer solutions with few crossings over the primary edges, i.e., it should be beneficial to cross over the secondary edges, for which we have simplicity constraints. By subtracting a small ε from the objective function coefficients of variables containing a secondary segment, we make them slightly more attractive than primary segments (if they are activated). Clearly, ε has to be chosen small enough to not influence the final integer-rounded outcome of the algorithm.

If at some point $\kappa(i) + 1 = \ell$, we will not add an according variable. If there are no other variables to be introduced, we establish a *partial* simplicity constraint for e_1 only restricting the crossings with segments f' for which $z_{\{e_{\ell-1},f'\}}$ is already activated. We can then resolve our LP relaxation.

Using the described generation strategy, the Kuratowski constraints could be left unchanged without making the ILP invalid. However, if there is any active constraint based on a simple crossing set $\mathcal{R}'$ and the Kuratowski subdivision K, such that $\left\{e_1, f^{(i)}_{\gamma(i)}\right\} \in \mathcal{CP}'(K) \cap \mathcal{R}'$ for any $f^{(i)}_{\gamma(i)} \in F_e$, the strength of this constraint degrades, as potential crossings can be shifted from e_1 to $e_{\kappa(i)+1}$ now. In order to reduce this effect, we duplicate all such Kuratowski constraints replacing e_1 by $e_{\kappa(i)+1}$.

See Figure 6.2 for an illustration of the presented column generation scheme. Consider a rounded integer solution, based on some LP relaxation, as shown on the left: There are two segments $f^{(1)}_1$ and $f^{(2)}_2$ crossing the primary segment of the horizontal edge e. Since this violates the imaginary simplicity constraint on e_1, we add the variables $z_{\{e_2,{f^{(1)}}_1\}}$ and $z_{\{e_2,{f^{(2)}}_2\}}$, such that a solution as given on the right can be computed. While the first solution has an objective value of $1 + 1 = 2$, the new solution is $1 + 1 - \epsilon = 2 - \epsilon$, and will therefore be preferred by the LP solver.

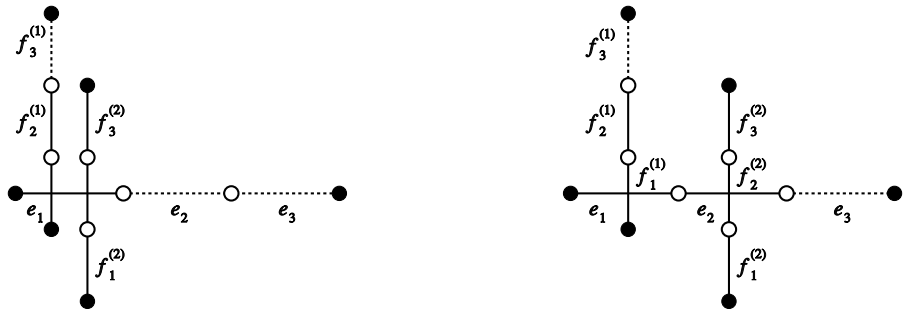


Figure 6.2: Combinatorial Column Generation for SECM. Left: imaginary simplicity constraint is violated; Right: by adding additional variables and reducing the cost on the additional segment, the situation is resolved.

Whenever the column generation terminates without activating additional variables—and therefore not forcing us to re-solve the LP relaxation—we are guaranteed to be able to interpret the solution for our cutting scheme. Clearly, whenever no imaginary simplicity constraint is violated by the LP relaxation, the currently inactive variables are not necessary for the optimal solution. Hence the optimal objective value on the restricted variable set is at least as small as on the fully expanded ILP, which suffices to prove the optimality of the induced solutions.

6.4 Combinatorial Column Generation for OECM

The combinatorial column generation scheme for OECM builds on the conceptual ideas of the scheme for SECM, i.e., we will also start with allowing solutions with ambiguous ordering, and add ordering variables as necessary. Since the variables of OECM intrinsically describe the crossings and their order in distinct variable sets x and y, respectively, the column generation is somewhat more direct than for SECM.

Our initial subproblem will contain all x-variables. Clearly, we do not require y-variables if there is at most one crossing on each edge—then all y variables are zero. Hence, conceptually, having some solution $\bar{x}$, we only require the y-variables with a base edge e, if there are multiple edges crossing over e.

Recall that only the x-variables enter the objective function: The values of the y-variables do not influence the solution value as they are only introduced to solve the ordering problems on the edges. In particular, we add y-variables only to restrict certain orderings, and their addition will never allow solutions not also valid before. Hence, the value of the LP relaxation can never decrease due to our column generation, which is in stark contrast to typical pricing schemes. This property allows us to use the same strong bounding arguments

as known from traditional branch-and-cut strategies.

Since the separation routine only uses integer rounded interpretations of the current fractional solution, we only require the knowledge of the crossing order if $\sum_{f:\{e,f\}\in\mathcal{CP}} \tilde{x}_{\{e,f\}} \geq 2$ for some edge e. Let F_e be the set of edges f with $\tilde{x}_{\{e,f\}} = 1$. The order of performing the variable generation prior to the separation routine is again crucial: we first obtain a fractional solution and check if the solution can be uniquely interpreted as a partial planarization, i.e., if all the variables $y_{e,f,g}$, with $\{f,g\} \subseteq F_e$, are active. If not, we add all inactive such variables, together with their corresponding LO-constraints, and resolve our LP model.

Hence, the variable generation takes place before we interpret a fractional solution as a partial planarization for the separation routine, and before the bounding heuristic. Therefore, for these steps we guarantee that all necessary y-variables are in the model, and the solution is LO-feasible.

Chapter 7

Crossing Number Variants

crossing: *(Martial Arts)*
When one crosses blades with the opponent.

7.1 Simple Crossing Number

We already introduced the *simple crossing number* $\mathrm{scr}(G)$ of a graph G in the description of SECM (Section 5.2). It is defined analogously to the traditional crossing number, under the restriction that every edge is involved in at most one crossing.

SECM solves the crossing number of some given graph G by subdividing all edges into long enough chains, obtaining the graph $G^{[\ell]}$, and then computing $\mathrm{scr}(G[\ell])$.

To solve the simple crossing number problem, we can simply apply SECM to the original graph, instead of $G^{[\ell]}$, obtaining $O(|E|^2)$ variables.

We could also use OECM by fixing all y-variables to 0. Under these conditions, the crossing-existence, mirror-order, and cyclic-order constraints (5.11)–(5.14) are redundant and trivially satisfied. The stacked-order constraints (5.12) reduce to

$$x_{\{e,f\}} + x_{\{e,g\}} \leq 1 \qquad \forall \{e; f, g\} \subseteq \mathcal{CP}, f \neq g \tag{7.1}$$

which are weaker versions of the simplicity constraints (5.6): While any solution feasible for (5.6) also satisfies (7.1), the assignment $\bar{x}_{\{e,f\}} = \bar{x}_{\{e,g\}} = \bar{x}_{\{e,h\}} = 0.5$, for $\{e; f, g, h\} \subseteq \mathcal{CP}$ and pairwise disjoint f, g, h, violates (5.6) but not (7.1). Hence, as the Kuratowski constraints become equivalent, SECM is preferable to solve the simple crossing number problem.

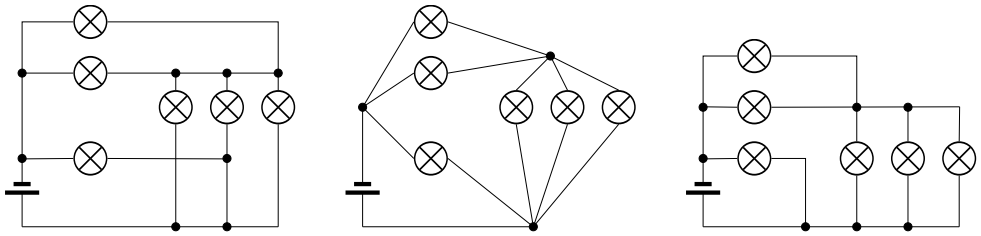


Figure 7.1: The wiring scheme G'' (left) cannot be drawn without any crossing. By computing a minor G (middle) and considering a realizing graph G' (right), we obtain an equivalent but planar wiring scheme.

7.2 Minor and Hypergraph Crossing Number

In this section we will consider both the *minor crossing number*, and the crossing number of hypergraphs, as defined below. We will investigate the relationship between these two numbers and introduce the generalized *W-restricted minor crossing number*, which can be used to model both aforementioned crossing numbers.

Besides from their theoretical appeal, these problems also occur, e.g., for crossing minimal layouts of electrical wiring schemes [15], cf. Figure 7.1 and Section 7.2.7. Usually, the exact topology of such a wiring scheme G'' is not interesting for the connected subgraphs which have the same electric potential. Hence we can "merge" these nodes into one node, which is exactly the operation to obtain a minor G, compute the minor crossing number $\mathrm{mcr}(G)$ and expand the graph accordingly to obtain G'. In this example, we can observe the connection to hypergraphs: By seeing the resistors on the wires as nodes, we can interpret the wires on the same potential as *hyperedges*, i.e., edges with multiple incident nodes.

7.2.1 Definitions

Minor Crossing Number. Recall from Section 2.1 that a graph H is a minor of the graph G, if H can be obtained from G by a series of *minor operations*. Such an operation is to either (i) delete a node and its incident edges, or (ii) contract an edge $\{v_1, v_2\}$, thereby unifying the two incident nodes into a new node v which is incident to all former neighbors of v_1 and v_2. The latter operation is called *edge contraction*.

Symmetrically, we can define the *inverse minor operations*; H is a minor of G, if we can obtain G from H by a series of the following operations: We either (i) introduce a new node, probably incident to some nodes in the graph, or (ii) we replace some node v by an edge $\{v_1, v_2\}$, and for each neighbor $u \in N(v)$ of v we introduce an edge $\{v_1, u\}$, $\{v_2, u\}$, or both. We call the latter operation *node expansion*.

Definition 7.1 (Minor Crossing Number)**.** The *minor-monotone crossing number* $\mathrm{mcr}(G)$, or *minor crossing number* for short, is the smallest crossing number of any graph G' which has G as its minor, i.e.,

$$\mathrm{mcr}(G) = \min_{G \preceq G'} \mathrm{cr}(G').$$

Let $G \preceq G'$ and $\mathrm{cr}(G') = \mathrm{mcr}(G)$. We say G' is a *realizing graph* of $\mathrm{mcr}(G)$.

Clearly, we always have $\mathrm{mcr}(G) \leq \mathrm{cr}(G)$. The eponymous property of this number is that, unlike the traditional crossing number, it is monotonously decreasing with respect to the minor relation; if $H \prec G$, we have $\mathrm{mcr}(H) \leq \mathrm{mcr}(G)$. The minor crossing number has yet only been studied in the context of theoretical lower and upper bounds [14, 15], but was never before tackled algorithmically.

Observation 7.2. *Consider a cubic graph G, i.e., a graph where each node has degree 3. We have* $\mathrm{mcr}(G) = \mathrm{cr}(G)$.

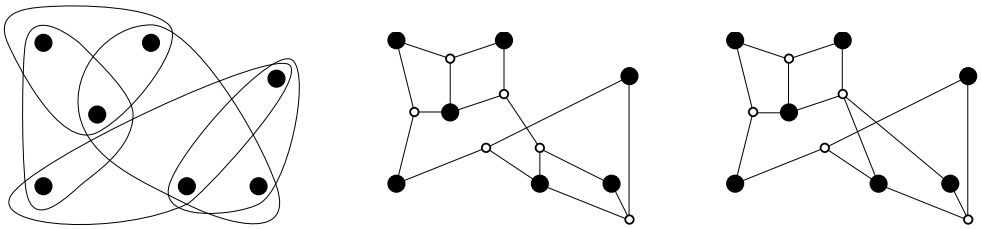
Proof. Since $\mathrm{mcr}(G) \not> \mathrm{cr}(G)$, assume $\mathrm{mcr}(G) < \mathrm{cr}(G)$. Let G' be a realizing graph of $\mathrm{mcr}(G)$. Since $G \preceq G'$, there has to exist a series of inverse minor operations to transform G into G'. Clearly, the addition of nodes and edges will never result in a crossing number smaller than the original $\mathrm{cr}(G)$. The node expansion do also not decrease the crossing number: Expand a node v with $N(v) = \{u_1, u_2, u_3\}$ into an edge $\{v_1, v_2\}$; we can either

1. attach all neighbors of v to only one of the new nodes (say v_1); or
2. attach two neighbors (say u_1, u_2) to one node (say v_1) and the other one to the other; or
3. choose one of the above and add additional edges connecting $N(v)$ with v_1 or v_2.

The third possibility will never allow a crossing number less than the former two options. In the first case, the crossing number does not change, since we effectively only add a single edge incident to a new node (v_2) which has degree 1. The second case resembles subdividing an edge (in our case the edge $\{v, u_3\}$) into a chain of two edges; this does not change the overall crossing number. Analogously, expanding any of the newly introduced nodes v with $\deg(v) < 3$ can neither reduce the crossing number.

This is a contradiction to $\mathrm{cr}(G') < \mathrm{cr}(G)$ and hence equality holds. □

In [85], Hliněný showed that the crossing number problem remains NP-complete when considering cubic graphs; hence the minor crossing number problem is NP-complete as well.



(a) Subset (b) Edge, tree-based (c) Edge, point-based

Figure 7.2: A hypergraph, drawn using different drawing styles.

Hypergraph Crossing Number. Recall from Section 2.1 that a hypergraph $\mathcal{H} = (V, \Psi)$ differs from an ordinary graph in that instead of edges—which have exactly two incident nodes—we consider *hyperedges*: A hyperedge $\psi \in \Psi$ is a proper subset of V (i.e., $\psi \subset V$) with $|\psi| \geq 2$. See, e.g., [91] for details. In the following, we will restrict ourselves to undirected hypergraphs; Section 7.2.7 will briefly discuss directed hyperarcs as well.

There are two major variants on how to draw hypergraphs [109], cf. Figure 7.2: the *subset-standard* and the *edge-standard*. The first variant becomes very confusing with more hyperedges, and it is ambiguous how to define a consistent notion of crossings. Hence, most applications, cf. [52, 98, 130], focus on the edge-standard which allows two sub-variants: In the *tree-based* drawing style, each hyperedge ψ is drawn as a tree-like structure of lines, whose leaves are the incident nodes of ψ. If we restrict the tree-like structure of every hyperedge to be a star, we obtain the *point-based* drawing style.

Formally, each hyperedge $\psi \in \Psi$ has a set of associated *hypernodes* N_ψ, which form the branching points of the line tree. Each node $v \in \psi$ is connected to exactly one $n \in N_\psi$, and all hypernodes N_ψ are tree-wise connected. By this *tree-based transformation* we obtain a traditional graph L. We denote the set of all graphs L obtainable by such transformations by $\mathcal{L}(H)$, and can naturally define:

Definition 7.3 (Tree-based Hypergraph Crossing Number)**.** Let $\mathcal{H}$ be a hypergraph. We define the *tree-based hypergraph crossing number* as

$$\operatorname{thcr}(\mathcal{H}) := \min_{L \in \mathcal{L}(\mathcal{H})} \operatorname{cr}(L).$$

The tree-based hypergraph crossing number has the elegant property that it is equivalent to the traditional crossing number if all hyperedges have cardinality 2. Because of this property, computing $\operatorname{thcr}(H)$ is NP-hard.

We further define the *point-based transformation* $\Lambda(H)$ as the special tree-based transformation where each hyperedge has exactly one associated hypernode, i.e., $\Lambda(\mathcal{H}) := (V \cup \Psi, E')$ with $E' := \{\{v, \psi\} \mid v \in \psi, \psi \in \Psi\}$. Clearly, this leads to the point-based drawing style and the definition of the *point-based hypergraph crossing number* $\operatorname{phcr}(\mathcal{H}) := \operatorname{cr}(\Lambda(\mathcal{H}))$.

Observation 7.4. *For any $L \in \mathcal{L}(\mathcal{H})$ we have $\Lambda(\mathcal{H}) \preceq L$, i.e., the point-based transformation of $\mathcal{H}$ is the minor of any tree-based transformation of $\mathcal{H}$.*

Point-based hypergraph planarity of $\mathcal{H}$ can be defined as $\text{phcr}(\mathcal{H}) = 0$ straightforwardly and is equivalent to Zykov planarity [91]. It can be efficiently tested by transforming $\mathcal{H}$ into $\Lambda(\mathcal{H})$ in linear time and applying any traditional linear-time planarity testing algorithm to $\Lambda(\mathcal{H})$. Analogously, *tree-based hypergraph planarity* can be defined as $\text{thcr}(\mathcal{H}) = 0$. Since $L \in \mathcal{L}(\mathcal{H})$ is planar if and only if $\Lambda(\mathcal{H})$ is planar, all three planarity definitions are equivalent.

Obviously, the point-based hypergraph crossing minimization of $\mathcal{H}$ is equivalent to the traditional crossing minimization on the graph $\Lambda(\mathcal{H})$. Hence we will focus on computing $\text{thcr}(\mathcal{H})$.

7.2.2 Relationship and Observations

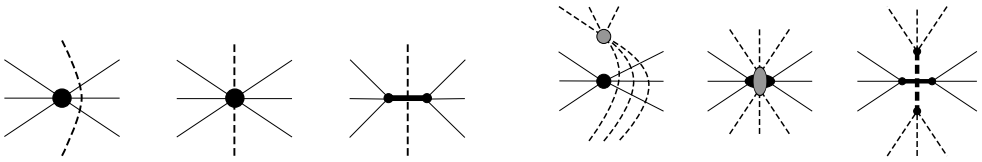
Restricted Minor Crossing Number. Let $W \subseteq V$. We can define the minor relation $G' \succeq_W G$, i.e., G is a W-minor of G' if we can obtain G' from G by only expanding nodes of W. This leads to the more general *W-restricted Minor Crossing Number* $\text{mcr}_W(G)$, i.e., the smallest crossing number of any graph $G' \succeq_W G$. Clearly $\text{mcr}_W(G) = \text{mcr}(G)$ if $W = V$. Since nodes with degree less than 4 are irrelevant for the differences between the traditional and the minor crossing number, we have:

Theorem 7.5. *Let $\mathcal{H} = (V, \Psi)$ be a hypergraph and $\hat{\Psi} := \{\psi \in \Psi \mid |\psi| \geq 4\}$. The tree-based hypergraph crossing number of $\mathcal{H}$ is equivalent to the $\hat{\Psi}$-restricted minor crossing number of $\Lambda(\mathcal{H})$, i.e., $\text{thcr}(\mathcal{H}) = \text{mcr}_{\hat{\Psi}}(\Lambda(\mathcal{H}))$.*

Hence, computing the tree-based hypergraph crossing number of $\mathcal{H}$ is equivalent to finding a realizing graph $\Lambda' \succeq_{\hat{\Psi}} \Lambda(\mathcal{H})$ with smallest crossing number, i.e., we may obtain Λ' by only expanding hypernodes of degree at least 4. In the following, we will always consider an undirected graph $G = (V, E)$ with $W \subseteq V$, and we are interested in $\text{mcr}_W(G)$.

General Observations. For the following algorithms, there are two points of view which are helpful when discussing the problem of minor crossing numbers: We can replace each node $v \in W$ with $\deg(v) \geq 4$ by an *expansion tree* T_v, which is incident to all nodes—or their respective expansion trees—originally incident to v. The nodes of T_v are called the *split nodes* of v. The W-restricted minor crossing number problem can then be reformulated as finding a *tree expansion* G', i.e., a graph obtained by such transformations, with smallest crossing number.

Another possibility to view the problem is that in the traditional crossing number problem, edges are allowed to cross. For the minor crossing number, edges are also allowed to cross through vertices, and moreover vertices may



(a) Edge crosses node (b) Node crosses node

Figure 7.3: New types of crossings. Both (a) and (b) give three visualizations: On the left, we see a situation for the traditional crossing number. We require less crossings for the minor-monotone case, by allowing novel crossing types (middle), which lead to a realizing graph structure depicted on the right.

even "cross" other vertices, cf. Figure 7.3. Such crossings can be seen as crossings between an expansion tree and a traditional edge, or between two expansion trees, respectively.

7.2.3 Edge and Node Insertion Results

In the following, we will always consider the W-restricted variant of the minor crossing number. Since W may be the full set V, the algorithmic results also hold for the pure minor-monotonous case.

We will first summarize our results on insertion problems w.r.t. the minor crossing number, and prove them in the section thereafter. We will then show how to use these results to obtain a practical heuristic to tackle the minor crossing number problem. In Section 7.2.6 we will outline a concept of an 0/1-ILP to compute this number exactly.

We consider two embeddings Γ of G and Γ' of G' ($G \preceq G'$) *equivalent*, if we can obtain Γ by performing the necessary minor operations stepwise on G' and Γ' in the natural way: Let us merge the connected nodes v_1 and v_2 with their respective cyclic orders $\pi_{v_1} = \langle \{v_1, v_2\}, e_1, \ldots, e_{\deg(v_1)-1} \rangle$ and $\pi_{v_2} = \langle \{v_2, v_1\}, f_1, \ldots, f_{\deg(v_2)-1} \rangle$ of their incident edges. The new node v will have the cyclic order $\pi_v = \langle e_1, \ldots, e_{\deg(v_1)-1}, f_1, \ldots, f_{\deg(v_2)-1} \rangle$.

Similar to the corresponding problems for the traditional crossing number, we can state the following related problems:

Definition 7.6 (Minor Edge Insertion)**.** Let $G = (V, E)$ be a planar undirected graph, and let $e = \{s, t\} \in V^{\{2\}} \setminus E$ be an edge not yet in G. The *W-restricted Minor Edge Insertion Problem with Variable Embedding (MEIV)* is to find the W-restricted minor crossing number of the graph $G + e$, under the restriction that the realizing drawing induces a planar drawing of G.

Given a specific planar embedding Γ of G, the *W-restricted Minor Edge Insertion Problem with Fixed Embedding (MEIF)* is to find the W-restricted minor crossing number of the graph $G + e$, under the restriction that the realizing drawing induces an embedding of G equivalent to Γ.

For both problems, the corresponding problems concerning the traditional crossing number can be solved in linear time. We will show:

Theorem 7.7. *MEIF and MEIV can be solved to optimality in linear time.*

In Section 7.2.5, we show how to use this result to obtain a heuristic following the planarization method.

Definition 7.8 (Minor Node Insertion)**.** Let $G = (V, E)$ be a planar undirected graph, let $v \notin V$ be a node not yet in G, and let E' be edges connecting v with nodes of V. Let $W^- := W$ and $W^+ := W \cup \{v\}$. The *W^--restricted (W^+-restricted) Minor Node Insertion Problem with Variable Embedding* is to find the W^--restricted (W^+-restricted) minor crossing number of the graph $G' = (V \cup \{v\}, E \cup E')$, under the restriction that the realizing drawing induces a planar drawing of G. We abbreviate these problems MNIV$^-$ and MNIV$^+$, respectively.

Assume we are given a specific planar embedding Γ of G. Then the *W^--restricted (W^+-restricted) Minor Node Insertion Problem with Fixed Embedding* is to find the W^--restricted (W^+-restricted) minor crossing number of the graph G', under the restriction that the realizing drawing induces an embedding of G equivalent to Γ. We abbreviate these problems MNIF$^-$ and MNIF$^+$, respectively.

The corresponding problem for the traditional crossing number and a fixed embedding can be solved in $\mathcal{O}(|V| \cdot |E'|)$ time. An analogous algorithm, together with the ideas of Theorem 7.7, can be used to show:

Theorem 7.9. *MNIF$^-$ is solvable in $\mathcal{O}(|V| \cdot |E'|)$ time.*

The problem for the traditional crossing number where all embeddings are considered—and therefore a special case of MNIV$^-$—has been open until recently [32], when it was shown to be polynomially solvable. In contrast to this result, we show that the problem is hard when the inserted node is allowed to be expanded:

Theorem 7.10. *MNIF$^+$ and MNIV$^+$ are NP-hard. This also holds for the case $W = V$, i.e., non-restricted minor-monotonicity.*

Corollary 7.11. *Let $\mathcal{H} = (V, \Psi)$ be a hypergraph, and $\psi \notin \Psi$ a hyperedge not yet in $\mathcal{H}$. Under the restriction that $\mathcal{H}$ has to be drawn planar and independent on whether a specific embedding of $\mathcal{H}$ is given or not, we have: Computing* $\mathrm{thcr}(\mathcal{H} + \psi)$ *is NP-hard.*

The theorem is based on the observation that we can turn any planar Steiner tree problem instance (NP-complete, [63]) into a corresponding MNIF$^+$ problem, see below. The corollary does not follow from Theorem 7.10 itself, but from its proof. Furthermore, since computing $\mathrm{thcr}(\mathcal{H})$ is a special case of a W-restricted minor crossing number we have in particular:

Observation 7.12. *The heuristic and exact algorithms presented below can be used to solve the tree-based hypergraph crossing number problem heuristically and to optimality, respectively.*

7.2.4 Edge and Node Insertion Details

For the edge insertion, our task is to find a tree expansion G' of G along with an insertion path connecting s and t, i.e., an ordered list of edges that are crossed when inserting e. Observe that it is never necessary to expand s or t.

Algorithm for Theorem 7.7 (Fixed Embedding). We show that MEIF is solvable in linear time.

Let Γ be an embedding of G. We define a directed graph $D_{\Gamma,s,t} = (N, A)$ as follows. N contains a node n_φ for each face φ in Γ and a node n_v for each node $v \in W \cup \{s, t\}$. Each arc $a \in A$ has an associated cost $c_a \in \{0, 1\}$; we have the following arcs:

- For each pair φ, φ' of adjacent faces, we have two arcs $(n_\varphi, n_{\varphi'})$ and $(n_{\varphi'}, n_\varphi)$ with cost 1.
- For each node $v \in W \setminus \{s, t\}$ and face φ incident to v we have an arc (n_v, n_φ) with cost 1 and an arc (n_φ, n_v) with cost 0.
- Finally, we have arcs (n_s, n_φ) for each face φ incident to s and (n_φ, n_t) for each face φ incident to t; all these arcs have cost 0.

Then, the solution to MEIF is the length of a shortest path p in $D_{\Gamma,s,t}$ from n_s to n_t; each arc $(n_\varphi, n_{\varphi'})$ in p corresponds to crossing an edge separating φ and φ'; each sub-path $(n_\varphi, n_v), (n_v, n_{\varphi'})$ corresponds to splitting node v and crossing the edge resulting from the split.

The number of nodes in N is bounded by $|V| + |\Phi|$ (where Φ is the set of faces in Γ), and the number of arcs in A by $4 \cdot |E|$, since we have at most four arcs per edge. Hence, we can apply breadth-first-search for finding a shortest path in $D_{\Gamma,s,t}$ which takes time $\mathcal{O}(|V|)$ as in planar graphs $|E|$ and $|\Phi|$ are of the same order as $|V|$. We remark that BFS can easily be extended to graphs with 0/1-arc costs. Thus, we can solve MEIF in linear time. □

Algorithm for Theorem 7.7 (Variable Embedding). We show that MEIV is solvable in linear time.

In order to solve MEIV, we adapt the algorithm by Gutwenger et al. [77] which solves the problem for the traditional crossing number, i.e., $W = \emptyset$ and no tree expansions are possible. They showed that it is sufficient to consider the shortest path $B_0, v_1, B_1, \ldots, v_k, B_k$ in the BC-tree of G and independently compute optimal edge insertion paths in the blocks B_i from v_i to v_{i+1} ($0 \leq i \leq k$, $v_0 = s$, and $v_{k+1} = t$). This is also true when we are allowed to split

the nodes W: Concatenating the respective paths in the blocks results in a valid insertion path, and alternately crossing edges from different blocks or splitting (and crossing through) a cut vertex v_i would result in unnecessary crossings.

Thus, we can restrict ourselves to a biconnected graph G. Let $\mathcal{T}$ be the SPR-tree of G, cf. Section 4.2.1. We consider the shortest path $p = \mu_1, \dots, \mu_h$ in $\mathcal{T}$ from a node μ_1 whose skeleton contains s to a node μ_h whose skeleton contains t. Let G_i be the skeleton of μ_i ($1 \leq i \leq h$). The *representative* $\text{rep}(v)$ of a node $v \in G$ in a skeleton G_i is either v itself if $v \in G_i$, or the virtual edge $e \in G_i$ whose expansion graph contains v. If $W = \emptyset$, the optimal algorithm only considers the R-nodes—triconnected components with therefore unique embeddings—on p and independently computes optimal edge insertion paths in fixed embeddings of the respective skeletons from $\text{rep}(s)$ to $\text{rep}(t)$. If the representative is an edge, we assume that a virtual node is placed on this edge and serves as start or endpoint of the insertion path.

This approach is invalid if $W \neq \emptyset$: An optimal insertion path in a skeleton G_i might cross through an endpoint a of the edge representing t in G_i, and continuing this path from a in G_{i+1} might save a crossing. We circumvent this problem by processing p in order from μ_1 to μ_h: For each R-node μ_i where $\text{rep}(t)$ is an edge $e_t = \{a, b\}$, we compute three insertion paths p_a, p_b, and p_e in a fixed embedding of G_i, which are optimal insertion paths to a, b, and e_t, respectively. Observe that for the respective lengths ℓ_a, ℓ_b, and ℓ_e of these paths we have $\ell_e \leq \ell_a, \ell_b \leq \ell_e + 1$. If $a \in W$, $\ell_a = \ell_e$, and a is contained in the skeleton of the next processed R-node μ_j, then a is a possible start node for an insertion path in G_j; the analogous is true for b; $\text{rep}(s)$ is always a possible start node. We compute the optimal insertion paths in the R-node skeletons by slightly modifying the search network introduced for MEIF. We introduce a super start node s^* and connect it to the possible start nodes. Then we compute shortest paths from s^* to $\text{rep}(t)$ and, if this is an edge e_t, to the endpoints of e_t.

After processing all nodes on p, we reconstruct the optimal insertion path backwards from t to s. The insertion path in G_h ends in t; we determine which insertion path in the preceding R-node skeleton to chose by checking which start node is used, until we reach s. This algorithm can be implemented in linear time, thus showing that MEIV can be solved in linear time as well. □

Algorithm for Theorem 7.9. We show that MNIF$^-$ is solvable in $\mathcal{O}(|V| \cdot |E'|)$ time.

Let U be the nodes of V incident to edges of E'. We can solve the node insertion problem with fixed topology for the traditional crossing number by considering the dual graph G^D of G with respect to Γ. Each node in G^D is labeled with a number which is initially 0. We then start a BFS for each $u \in U$, augmenting G^D with edges between u and its incident faces. The labels in G^D are incremented by their BFS-depth minus 1, for each different

u. Finally, each node of G^D holds the sum of the shortest distances between itself and the nodes U. We then simply pick a node of G^D with smallest number, and insert the new node v into the corresponding face in Γ.

Using the ideas from solving MEIF, we can use the same algorithm but allow edges to cross through nodes. Since all inserted edges are incident to v, they will not cross each other in any optimal node insertion. Therefore, no conflicting edge-node crossings can occur, other than ones based on paths with equal length. Such conflicts can easily be resolved by choosing any of the conflicting paths for both inserted edges. The correctness and running time of the algorithm follows directly. □

Proof of Theorem 7.10. We show that MNIF$^+$ and MNIV$^+$ are NP-hard.

We can restrict ourselves to MNIF$^+$. Since a planar 3-connected graph has only a unique planar embedding and its mirror, we naturally have NP-hardness for MNIV$^+$ if MNIF$^+$ is NP-hard. We only briefly sketch the idea of the proof.

We reduce to the planar Steiner tree problem, which is known to be NP-complete [63]. Thereby we are given a planar graph D with integer edge-weights and a subset of its nodes are marked as *terminals*. We ask for a weight-minimum tree T connecting all terminals (and possibly some other nodes).

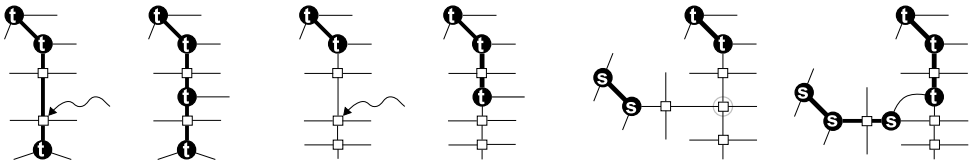
Choose any embedding of D and augment it greedily by additional edges to obtain a planar triangulated graph D' (i.e., each face is a triangle); the new edges get high enough costs such that they will not be chosen for an optimal solution T. Note that these costs, as well as the given edge costs, are polynomial in the graph size.

Let G be the dual of D'. For each terminal t in D', let φ_t be the corresponding face in G. We introduce a star of degree 3 into φ_t for each t, i.e., a new node u_t with incident edges to the three nodes bordering φ_t, calling the resulting graph G'.

Recall that the embeddings of D' and G' are unique. Finding a cost minimal Steiner tree in D' (and therefore in D) becomes equivalent to finding a tree-wise expansion for a node v that we want to insert into G, connecting it to the vertices u_t, for each terminal t in D; thereby we do not need any minor operations on the nodes originally in G. I.e., by solving the weighted $\emptyset^+$-restricted minor node insertion problem on the fixed embedding of G, we solve the Steiner tree problem.

Since all integer crossing weights in G' can be bounded to be a polynomial in the graph size, we can replace each edge of weight w by a parallel structure of this thickness, as described in Section 2.3. Hence we have NP-hardness for MNIF$^+$.

We can show that this also holds when $W = V$, i.e., all vertices, not only the new node v, are allowed to be split. Therefore we have to replace higher degree nodes by grids of maximum degree 3, similar to the brick-wall style



(a) Dummy on target (b) Dummy near target (c) Dummy near source and target

Figure 7.4: Modification of insertion paths ending at dummy nodes. Bold solid edges are part of expansion trees, dummy-nodes are denoted by squares.

grids used in the proof of [85], and attach the edges to this substructure. We already know from Observation 7.2 that for graphs with maximum degree 3, splitting nodes does not influence the crossing number. □

7.2.5 Heuristic Crossing Minimization

As outlined in Section 2.3, the *planarization method* is a well-known and successful heuristic for traditional crossing minimization; see [76, 73] for experimental studies. First, a planar subgraph is computed. Then the remaining edges are inserted one after another by computing edge insertion paths and inserting the edges accordingly, i.e., edge crossings are replaced by dummy vertices of degree 4.

In order to apply the algorithms from the previous section in a planarization approach for computing $\mathrm{mcr}_W(G)$, we need to generalize them, since the insertion of edges splits nodes and thereby expands them to trees. Furthermore, edges of G and edges resulting from node splits get subdivided by dummy vertices during the course of the planarization. We call the resulting paths *edge paths* and *tree paths*, respectively. Hence, we are not simply given two nodes s and t but two node sets S and T, and we have to find an insertion path connecting a node of S with a node of T. Thereby, S (T) is the set of all split nodes of s (t) and all dummy nodes on edge or tree paths starting at a split node of s (t). The dummy nodes in these sets have the property that a simple extension of a tree expansion is sufficient to connect an insertion path to a correct split node; see Figure 7.4 for a visual description.

Before we discuss the details, we give an overview of the planarization approach for $\mathrm{mcr}_W(G)$.

(1) Compute a planar subgraph $G' = (V, E')$ of G.

(2) For each edge $e = \{s, t\} \in E \setminus E'$:

   (a) Compute S and T.

   (b) Find an insertion path p from S to T in G'.

   (c) Insert e into G' according to p by splitting nodes if required and introducing new dummy nodes for crossings.

It remains to show how to generalize the edge insertion algorithms. In the fixed embedding scenario, we simply introduce a super start node s^* connected to all nodes in S, and a super end node t^* connected from all nodes in T in the search network. The following proposition shows the key property for generalizing the variable embedding case.

Proposition 7.13. *1. The blocks of G' containing a node in S (T) and the cut vertices of G' contained in S (T) form a subtree of the BC-tree of G'.*

2. Let $\mathcal{T}$ be the SPR-tree of a block of G'. The nodes of $\mathcal{T}$ whose skeletons contain a node in S (T) form a subtree of $\mathcal{T}$.

This allows us to compute the shortest paths in the BC- and SPR-trees in a similar way as described above. The only difference is that we consider blocks and skeletons containing any node in S (or T). The computation of insertion paths in R-node skeletons is generalized as for the fixed case if several nodes of S or T are contained.

In [76], two improvement techniques for the planarization approach are described which are both also applicable in our case. The *permutation* strategy calls step (2) several times and processes the edges in $E \setminus E'$ in random order. The *post-processing* strategy successively removes an edge path and tries to find a better insertion path. This can also be done for tree paths which in fact is a key optimization of our approach, since it allows to introduce crossings between two tree expansions as well. Finally, we remark that we also contract tree paths during the algorithm if they no longer contain a dummy node and thus become redundant.

7.2.6 Exact Crossing Minimization

We briefly sketch how to construct a 0/1-ILP to compute $\mathrm{mcr}_W(G)$, and therefore also $\mathrm{thcr}(\mathcal{H})$.

We can use both SECM and OECM formulations by considering tree expansions of G, instead of G itself. We replace each node $v \in W$, with $\deg(v) \geq 4$, by a vertex set V'_v with $|V'_v| = 2\deg(v) - 2$. Each edge originally incident to v is incident to a unique vertex of this set. By augmenting V'_v with edges, we can model any tree-wise connection of the vertices $\{w \in V'_v \mid \deg(w) = 1\}$. Hence, considering the edges $E'_v = V'^{\{2\}}_v$ of the complete graph on the set V'_v, we introduce 0/1-variables ξ_e for all $e \in E'_v$. If such a variable is 1, the corresponding edge is used for the tree-wise connection of V'_v.

We now require two main types of constraints. We can generalize the Kuratowski constraints straight-forwardly: let $M(K)$ be the set of edges with ξ variables which are contained in some Kuratowski subdivision K. We can subtract the term $\sum_{e \in M(K)}(1-\xi_e)$ on the right-hand side of the corresponding Kuratowski constraint, i.e., we only require a crossing on K if all its edges are selected.

Additionally, we have to assure the tree-wise connection for each V'_v. Therefore we require to select exactly $|V'_v| - 1$ edges of E'_v for each set V'_v, and assure connectivity via traditional cut constraints:

$$\forall v \in W, \deg(v) \geq 4, \forall \emptyset \neq S \subset V'_v : \sum_{u_1 \in S, u_2 \in V'_v \setminus S} \xi_{\{u_1,u_2\}} \geq 1.$$

All these constraints can be added dynamically within a branch-and-cut framework using known separation routines. Nonetheless, the resulting ILP seems to be too large even for relatively small graphs. It is therefore mainly of theoretical interest.

7.2.7 Application: Electrical Wiring Schemes

We will now briefly divert from the mainly theoretical background of crossing numbers and look at some application for hypergraph crossing numbers.

We consider the real-world problem of drawing electrical circuit designs. On a chip, the most important criteria for realizing such circuits is a small required area, leading to compact but confusing edge routings. But on the other hand, as stated in [51], a readable drawing with few edge crossings is beneficial for debugging, teaching, presentation, and documentation purposes. This is further strengthened by the fact that gate-level descriptions may by automatically synthesized from other, e.g., register-transfer level, descriptions. Furthermore, according to [11], the crossing number of a graph seems to be a good estimate for the required area on a chip.

As outlined in the beginning of Section 7.2, the properties of electrical wirings allow us to consider such circuits as hypergraphs. The problem of finding a crossing minimal drawing for such circuits was, e.g., tackled in [51, 52], the latter of which also gives a short overview on previous approaches. The currently best known approach [52] is based on Sugiyama's three-stage algorithm [136] of layering the graph, performing crossing minimizations between adjacent layers, and finally assigning coordinates.

Circuit Design. Gate-level networks are composed of the following components; see Figure 7.5 for an example:

Logical gates: Such a gate performs a specified logical operation, like *NOT*, *AND*, *NOR*, etc. Its corresponding electrical component takes one or more input signals on its *in-ports* and outputs a single signal on its *out-port*.

Input gates: Such a gate will retrieve an input signal from outside of the network. Therefore, it is only connected to in-ports of logical gates. Conceptually, it hence has no in-ports and a single out-port.

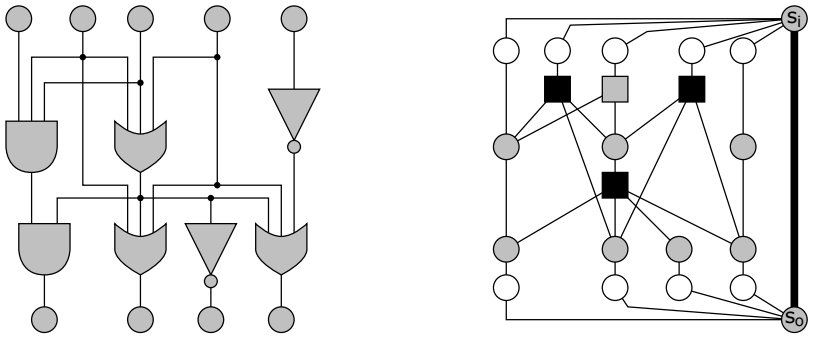


Figure 7.5: An electrical circuit (left) and its transformation into a graph for the restricted minor crossing number problem (right). Circles denote original nodes, squares denote the hypernodes from the point-based expansion. We allow the minor relations on the black nodes, the white nodes (input and output gates) are merged into the nodes s_i and s_o, respectively.

Output gates: Such a gate will output a resulting signal to the outside of the network. Therefore, it is only connected to out-ports. Conceptually, it itself has no out-ports and a single in-port.

Wires: Each wire will connect a single out-port to one or more in-ports. We will never have a wire directly connecting two out-ports: If the ports do not have the equivalent signal, this would result in a circuit failure.

For higher abstraction levels (register-transfer level, etc.), we may also consider *operational components*, which represent logical networks that perform more complex computations, like a 4-bit ADDER. Such components may have multiple out-ports, and the order of the in- and out-ports may be crucial.

Drawing requirements. A drawing of such a circuit has to follow certain established norms. The most common of which is that the input and output gates are drawn on opposing borders of the drawing, say inputs on the top, outputs on the bottom, and all other gates and wires in between. In terms of graph drawing we can deduce the following drawing requirements:

DR1. Input and output gates have to lie on the outer face of the drawing.

DR2. Consider the cyclic order of the input and output gates on the outer face. All input gates have to occur consecutively. This induces the same property for the output gates.

Consider a planarization and an embedding of it. If the embedding satisfies the above two properties, we can easily find a drawing where the input and output gates are on opposing borders, and where all other wires and gates are in-between.

Furthermore, for each operational component the order of its ports has to be retained by the drawing. If no order is given, we always require that the component's in-ports are consecutive in the embedding (and hence so are its out-ports).

In the aforementioned currently best known crossing minimization algorithm for electrical wiring schemes, the drawings furthermore have the property that they are drawn *upward*, i.e., the y-coordinate increases monotonically for each edge, when traversing it from its source node to one of its target nodes. This is an intrinsic property of Sugiyama-based algorithms, and it is unspecified whether this property is a requirement or a mere side-effect. In our context we will not take upwardness into account.

From Circuit Designs to Hypergraphs. Clearly, the gates in a circuit C correspond to nodes V in a hypergraph $\mathcal{H} = (V, \Psi)$. We partition V into the sets I, O, and L, corresponding to the input, output, and logical gates, respectively. Each wire connects an out-port to several in-ports; for each such wire we have a hyperedge connecting the nodes of the wire's ports, cf. Figure 7.5.

We start with the point-based transformation $\Lambda(\mathcal{H}) = (V \cup \Psi, E')$ of this hypergraph and want to solve the $\hat{\Psi}$-restricted minor crossing number problem on $\Lambda(\mathcal{H})$. However, this translation itself would not yet guarantee the drawing requirements discussed above. Therefore we further modify $\Lambda(\mathcal{H})$ into the graph Λ^+ as follows: we unify all input nodes into a node s_i, all output nodes into a node s_o, and we introduce a new edge $\{s_i, s_o\}$. We then have to solve the problem of finding $\mathrm{mcr}_{\hat{\Psi}}^{s_i,s_o}(\Lambda^+)$, i.e., the $\hat{\Psi}$-restricted minor crossing number of Λ^+ under the restriction that the edge $\{s_i, s_o\}$ has no crossings. Assuming we (exactly or heuristically) solve this crossing minimization problem, we obtain a planarization of Λ^+, where hypernodes might be expanded into subtrees.

We can deduce a drawing for the circuit C by reinterpreting the hyperedges as wires; it remains to reintroduce the input and output nodes. Since the edge $\{s_i, s_o\}$ has no crossings, we know that s_i and s_o lie in a common face, which we choose as our outer face. We can then choose a (conceptually arbitrarily small) crossing free region around s_i and place the input nodes on their corresponding edges next to s_i; we do the analogous for the output nodes. After removing $\{s_i, s_o\}$, we finally obtain a drawing of the circuit C where all inputs and outputs lie on the outside (DR1), and the input nodes occur consecutively (DR2). Hence we obtain a valid drawing of C and can deduce:

Proposition 7.14. *Given a circuit C, let $\mathrm{ccr}(C)$ be the crossing number of the circuit C under the drawing requirements DR1 and DR2. Let Λ^+ be its transformed graph as described above. We have:* $\mathrm{ccr}(C) = \mathrm{mcr}_{\hat{\mathcal{F}}}^{s_i,s_o}(\Lambda^+)$.

It remains to solve the restricted minor crossing number problem, as described below.

More complex operational components with multiple in- and out-ports can be modeled by subgraphs as follows: Each port is represented by a vertex. If the order of the vertices is given, we connect the vertices accordingly via a cycle. If the order is specified and if it is furthermore important whether this order occurs clockwise or counterclockwise, we can reuse the machinery of *embedding constraints* and their planarization, as described in [74]. If, on the other hand the order is not specified at all, we connect all in-port nodes to a new node, all out-ports to a second new node, and then connect both new nodes. In any case, we have to forbid any crossings through these new subgraphs.

Crossing Minimization of Circuit Designs. We have to augmented our above algorithms regarding $\mathrm{mcr}_W(G)$ in order to restrict the use of the edge $\{s_i, s_o\}$. The simplest possibility to enforce the restriction is to assign a crossing weight $k = \min(|I|, |O|) + 1$ to the restricted edge, or to replace the edge by a parallel substructure of that thickness. An optimal solution will then never cross through $\{s_i, s_o\}$ or its replacement, since it is cheaper to cross over all other edges incident to s_i or s_o, close to these nodes.

Although this strategy suffices, we prefer a method which does not require edge costs or the enlargement of the graph: The key concept of the insertion algorithm is to find a shortest path in the dual of the graph (or within subgraphs). To forbid the crossing of an edge then means to remove its dual from the routing network. It is obvious, that this strategy still always allows us to find an insertion path. We have:

Proposition 7.15. *The edge insertion problem corresponding to* $\mathrm{mcr}^{s_i,s_o}_{\hat{\mathcal{F}}}$ *can be solved in linear time, both over a fixed and over all possible embeddings.*

7.3 Simultaneous Crossing Number

In simultaneous graph drawing, we are given a series of graphs, rather than a single one, and consider visualization and embedding problems over this set of graphs. Research on simultaneous embeddings of graphs started in 2003 by Brass et al. [21], and resulted in a number of publications, e.g. [21, 50, 58, 59, 65, 66, 67], that deal with different kinds of simultaneous embeddings, e.g., simultaneous embeddings without restrictions on the edge drawings, simultaneous embeddings with fixed edges, simultaneous geometric embeddings. The main interest was the examination whether given pairs of graphs allow a simultaneous embedding or not. Recently, also related complexity questions have been studied [53, 65]. See, e.g., [35] for an introduction to the practical applications of simultaneous graph drawing, as this would be beyond the scope of this thesis.

We consider *simultaneous drawings with fixed edges*, i.e., given a series of graphs, we consider layouts for each graph such that all vertices and edges belonging to more than one graph are drawn identically in all drawings in which they appear. In this section, we deal with the problem of *crossing minimization for simultaneous graphs*; the objective is to minimize the total number of edge crossings over all these drawings. We also consider the *weighted* problem, where a small number of crossings may be more important for certain graphs of the series than for others.

While simultaneous graph drawing currently is a very active topic in graph drawing, the concept of crossing minimization for this problem class has, to our knowledge, not been studied before. Hence, the concise definition of the simultaneous crossing number, as given in the following subsection, also constitutes a contribution of this section.

We will discuss some of its main differences to the traditional crossing number in Section 7.3.2, wherein we show upper and lower bounds for the simultaneous crossing number, and prove its NP-completeness. The sections thereafter will discuss preprocessing and show how to extend both heuristic and exact crossing minimization algorithms to solve the problem. The latter also constitutes the first algorithm for testing general simultaneous planarity. In Part III of this thesis, we will give a brief experimental study and offer some insights on the quantitative difference between the traditional crossing number and the simultaneous one.

7.3.1 Definitions

Let $\{G_i = (V_i, E_i) \mid i = 1, \ldots, k\}$ be a set of graphs called *basic graphs*. We define their *simultaneous graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ by $\mathcal{V} := \bigcup_{i=1,\ldots,k} V_i$ and $\mathcal{E} := \bigcup_{i=1,\ldots,k} E_i$. Instead of creating layouts for each graph G_i independently, we draw $\mathcal{G}$ in the plane. We obtain a drawing for any G_i by deleting all images of the vertices and edges not belonging to G_i. Thus, a drawing $\mathcal{D}$ of $\mathcal{G}$ leads to a drawing $\mathcal{D}_i$ for each G_i such that all vertices and edges belonging to more than one graph are drawn identically in all corresponding drawings.

Consider a drawing $\mathcal{D}$ of $\mathcal{G}$ with crossings. We can distinguish between two different kinds of crossings: Assume there is a crossing between the edges e and f. We say it is a *phantom crossing*, if there is no basic graph G_i which contains both e and f. It is a *proper crossing* otherwise. Hence, phantom crossings do not correspond to a crossing in a drawing of any basic graph, as none of these graphs contains both edges. Thus, a drawing of a simultaneous graph $\mathcal{G}$ yields a set of planar drawings for each basic graph, if it contains only phantom crossings, if any at all.

Definition 7.16 (Simultaneous Planarity)**.** A set of graphs G_i, $i = 1, \ldots, k$, as well as their simultaneous graph $\mathcal{G}$, are called *simultaneously planar* if $\mathcal{G}$ can be drawn with only phantom crossings, if any at all.

Obviously, a planar simultaneous graph is always simultaneously planar. This definition of simultaneous planarity is equivalent to the definition of simultaneous embedding with fixed edges [50, 58, 65, 67]. Thus, we can reformulate a result of Gassner et al. as follows:

Theorem 7.17 (Gassner et al. [65])**.** *It is NP-complete to decide whether three or more basic graphs are simultaneously planar.*

The theorem states that testing simultaneous planarity is NP-complete for three or more basic graphs. The corresponding problem for two graphs is open, without strong conjectures for either a polynomial time algorithm nor for NP-completeness. The problem to decide whether two (or more) graphs have a simultaneously planar straight-line drawing is known to be NP-hard [53].

We use the new planarity definition instead of the usual embedding definition to emphasize our ambition to minimize crossings and produce simultaneous drawings.

Given a simultaneous graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we define *crossing costs* $\text{cc}(e, f) := |\{i \in \{1, \ldots, k\} \mid e, f \in E_i\}|$ for each edge pair $\{e, f\} \in \mathcal{E}^{\{2\}}$, i.e., $\text{cc}(e, f)$ is the number of basic graphs which contain both e and f. Hence, $\text{cc}(e, f)$ reflects the number of basic graphs whose induced drawings will have a crossing if e and f cross in a drawing of the simultaneous graph $\mathcal{G}$. Given a drawing $\mathcal{D}$ of $\mathcal{G}$, we define $\mathcal{R}_\mathcal{D}$ as a multi-set of edge pairs which lists all crossings in $\mathcal{D}$. Unlike for the traditional crossing number, optimal simultaneous drawings may require multiple crossings of the same pair of edges $\{e, f\}$ (see Section 7.3.2); $\mathcal{R}_\mathcal{D}$ contains exactly as many instances of $\{e, f\}$ as there are crossings induced by this pair.

We can define the simultaneous crossing number of a simultaneous graph $\mathcal{G}$ with respect to a drawing $\mathcal{D}$ of $\mathcal{G}$ as $\text{simcr}(\mathcal{G}, \mathcal{D}) := \sum_{\{e,f\} \in \mathcal{R}_\mathcal{D}} \text{cc}(e, f)$. Thus, $\text{simcr}(\mathcal{G}, \mathcal{D})$ is the total number of (proper) crossings in the drawings of all basic graphs which are induced by $\mathcal{D}$.

Definition 7.18 (Simultaneous Crossing Number)**.** We define the *simultaneous crossing number* $\text{simcr}(\mathcal{G})$ of a simultaneous graph $\mathcal{G}$ as the minimum number $\text{simcr}(\mathcal{G}, \mathcal{D})$ over all drawings $\mathcal{D}$ of $\mathcal{G}$.

A phantom crossing between two edges e and f has no effect on the simultaneous crossing number, as $\text{cc}(e, f) = 0$. Furthermore, we easily see the relationship to the previously defined simultaneous planarity.

Proposition 7.19. *A simultaneous graph is simultaneously planar if and only if its simultaneous crossing number is zero.*

Formally, we can state the problem:

Definition 7.20 (Simultaneous Crossing Minimization (SimCM))**.** Given a simultaneous graph $\mathcal{G}$, identify $\text{simcr}(\mathcal{G})$.

The crossing minimization for the simultaneous crossing number does not include the minimization of phantom crossings as they have cost zero. However, we still think that these crossings should be avoided if possible. Hence we also consider:

Definition 7.21 (Phantom Crossing Aware Simultaneous Crossing Minimization (SimCM$^+$))**.** Given a simultaneous graph $\mathcal{G}$, identify simcr($\mathcal{G}$) and the smallest number of phantom crossings possible among all drawings realizing simcr($\mathcal{G}$).

If not mentioned otherwise, we refer to a drawing realizing SimCM$^+$ as an *optimal drawing*. Clearly, a solution to this problem also solves SimCM. Considering SimCM$^+$ instead of SimCM becomes crucial for the implementation and analysis of our algorithms.

Sometimes a small number of crossings is more important for certain basic graphs than for others; e.g., we might want to have few crossings in the first and the last graph of the series. This can be achieved by considering *graph weights* $gw : \{1, \dots, k\} \to \mathbb{N} \setminus \{0\}$ that define a positive weight for each graph G_i. We can then define *weighted crossing costs* $\mathrm{cc}_{gw}(e, f) := \sum_{i \in \mathcal{I}_{e,f}} gw(i)$ where $\mathcal{I}_{e,f} = \{i \in \{1, \dots, k\} \mid e, f \in E_i\}$ is the index set of the basic graphs that contain both e and f. This leads to the *Graph-Weighted Simultaneous Crossing Number* simcr($\mathcal{G}, gw$), and therefore to the *Graph-Weighted Simultaneous Crossing Minimization* (gwSimCM) and the *Phantom Crossing Aware Graph-Weighted Simultaneous Crossing Minimization* (gwSimCM$^+$) problems.

While we will only discuss SimCM and SimCM$^+$ in the following, all algorithms can be used for gwSimCM and gwSimCM$^+$ as well.

7.3.2 Bounds and Complexity

Gassner et al. [65] also discuss the relationship between simultaneous planarity and the weak realizability problem. Furthermore, the problems are closely related to the abstract topological graph problem introduced by Kratochvíl [102]. Both the abstract topological graph problem and the simultaneous crossing minimization problem are quite different from traditional crossing minimization as we demonstrate in Figure 7.6: We know that adjacent edges do not cross in crossing minimal drawings. Furthermore, non-adjacent edges cross each other at most once. This, however, does not hold for simultaneous crossing minimization. The following propositions improve on similar propositions in the original paper [35] and answer an open problem stated therein:

Proposition 7.22. *There are simultaneous graphs $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $k \geq 2$ whose optimal drawings require that pairs of edges, even adjacent ones, cross multiple times.*

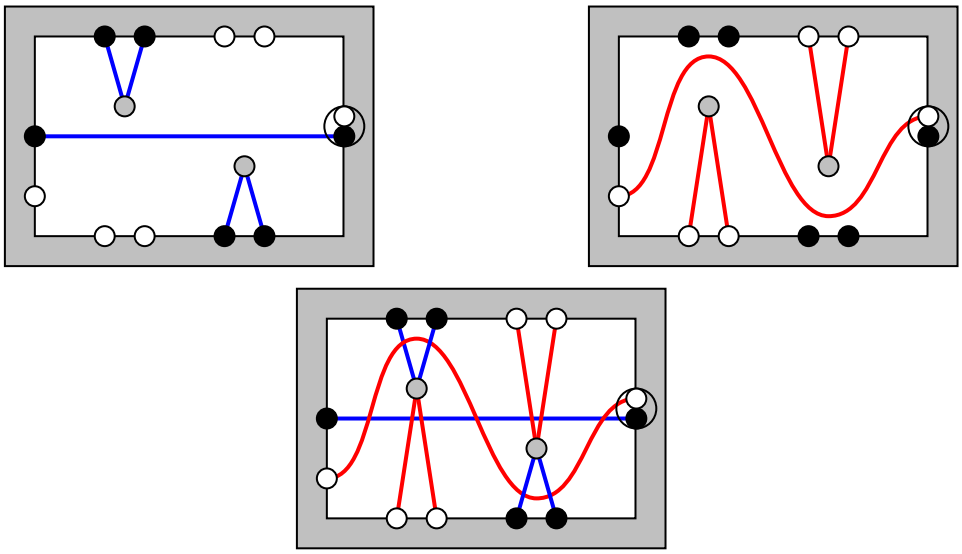


Figure 7.6: Two simultaneously planar basic graphs (top) where a pair of edges (the horizontal and the bent edge) crosses multiple times in an optimal drawing of their simultaneous graph (bottom). The gray region denotes a dense triconnected subgraph identical in both basic graphs. When merging the two nodes on the right-hand side, we observe multiple crossings on adjacent edges.

Proof. Figure 7.6 depicts a simultaneous graph $\mathcal{G}$ consisting of two basic graphs. In particular, both basic graphs have unique embeddings as they are triconnected apart from the "spikes"—i.e., single chains of length 2—which can only be uniquely embedded as well, given the triconnected part of the graph is dense enough. $\mathcal{G}$ has simultaneous crossing number zero but all realizing drawings involve multiple phantom crossings on a pair of edges; this edge pair can even be adjacent. □

Proposition 7.23. *There are simultaneous graphs $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $k \geq 3$ whose optimal drawings require that pairs of edges, even adjacent ones, cross $\Omega(|\mathcal{V}|)$ times.*

Proof. Figure 7.7 depicts a simultaneous graph $\mathcal{G}$ consisting of three basic graphs. As before, they have unique embeddings. $\mathcal{G}$ has simultaneous crossing number zero but all realizing drawings involve multiple phantom crossings on a pair of edges; this edge pair can even be adjacent. By enlarging the example at the zigzag line, we obtain linearly many phantom crossings between these two edges. Note that we require the additional basic graph to fix the inner gray nodes at their relative position: Without it, we could let the spikes of different basic graphs cross in order to sort them such that only 3 (or 2, for adjacent edges) phantom crossings are required between the edge pair. □

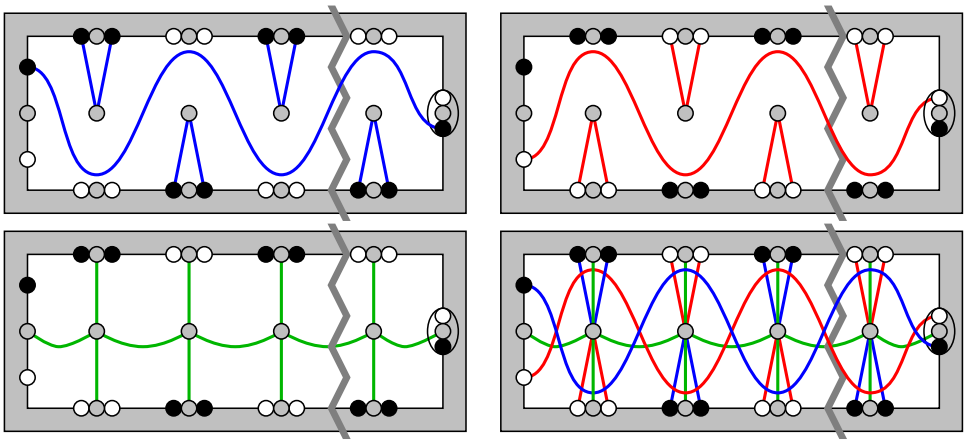


Figure 7.7: Three simultaneously planar basic graphs where a pair of edges crosses multiple times in an optimal drawing of their simultaneous graph (bottom right). The gray region denotes a dense triconnected subgraph identical in both basic graphs. When merging the two nodes on the right-hand side, we observe multiple crossings on adjacent edges. By enlarging the example at the zigzag line, we obtain linearly many crossings of this edge pair.

For a simultaneous graph $\mathcal{G}$ neither the traditional crossing number is bounded by the simultaneous crossing number nor the other way around. In fact, Figure 7.6 shows an example of a simultaneously planar (no proper crossings) but not planar (traditional crossing number greater than zero) graph. On the other hand, assume a non-planar graph G and construct a simultaneous graph $\mathcal{G}$ by defining basic graphs $G_1 = G_2 = G$. Then the simultaneous crossing number is twice the traditional crossing number of $\mathcal{G}$.

Using this idea, we develop an upper bound to the simultaneous crossing number:

Lemma 7.24. *Let $\mathcal{G}$ be a simultaneous graph with k basic graphs. Then* $\operatorname{simcr}(\mathcal{G}) \leq k \cdot \operatorname{cr}(\mathcal{G})$.

Proof. Starting with a simultaneous graph $\mathcal{G}$ with k basic graphs, we define a new simultaneous graph $\mathcal{G}'$ by a set of k basic graphs $G'_1 := \ldots G'_k := \mathcal{G}$. By construction we have $\operatorname{simcr}(\mathcal{G}') = k \cdot \operatorname{cr}(\mathcal{G})$, where $\operatorname{cr}(\mathcal{G})$ is the ordinary crossing number of $\mathcal{G}$. On the other hand $\operatorname{simcr}(\mathcal{G}) \leq \operatorname{simcr}(\mathcal{G}')$ as every crossing pair in $\mathcal{G}$ costs at most as much as in $\mathcal{G}'$. □

Corollary 7.25. *The simultaneous crossing number* $\operatorname{simcr}(\mathcal{G})$ *—and therefore the number of proper crossings in any optimal drawing of $\mathcal{G}$—is polynomial in the number of vertices and basic graphs for any simultaneous graph $\mathcal{G}$.*

A large difference to the traditional problem is the lack of upper bounds for the *cumulative crossing count* $\operatorname{ccc}(\mathcal{G})$, i.e., the sum of phantom and proper crossings, in the solution of a SimCM$^+$ instance. As we have seen in Figures 7.6 and 7.7, two edges can cross more than once. Therefore, we cannot

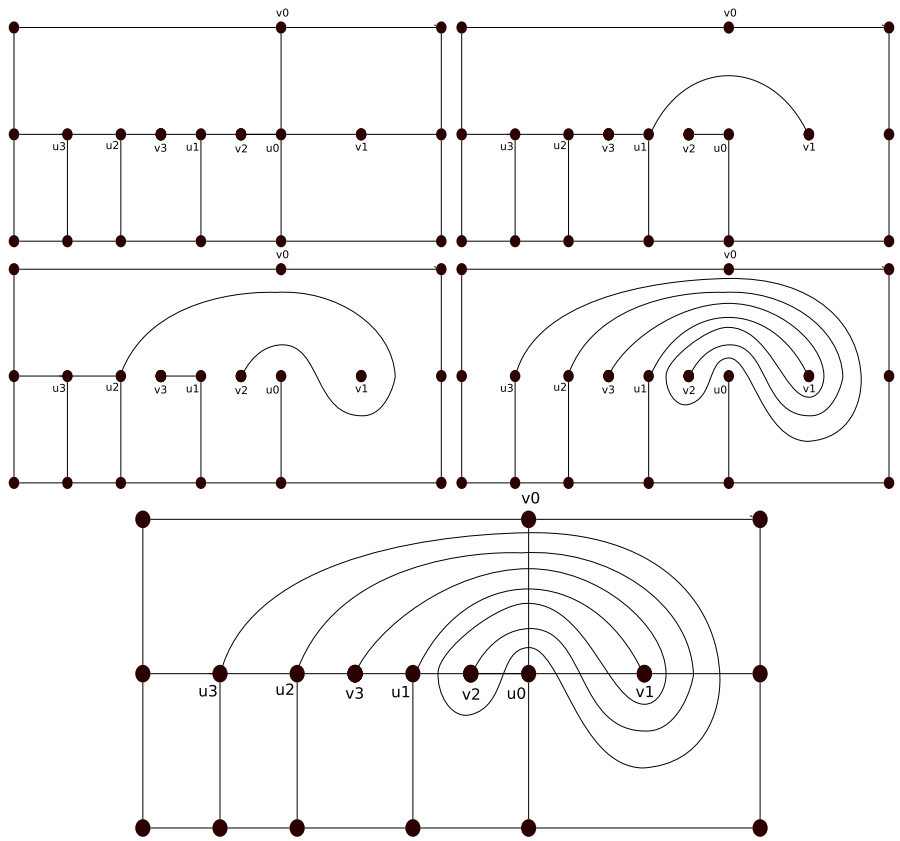


Figure 7.8: A simultaneously planar simultaneous graph (shown in the bottommost picture) with $n+1$ basic graphs. Edge (u_0, v_0) is involved in $2^n - 1$ phantom crossings. The pictures show the case $n = 3$.

bound the number of crossings per edge by $|\mathcal{E}| - 1$ in an optimal drawing as in traditional crossing minimization. Furthermore we have:

Proposition 7.26. *There are simultaneous graphs $\mathcal{G}$ whose optimal drawings require that an edge is involved in an exponential number of phantom crossings.*

Proof. Figure 7.8 shows a simultaneous graph with this property in any optimal drawing. The graph is adapted from Kratochvíl and Matoušek [103] where it was used in the context of string graphs. □

We know that the traditional crossing number problem is NP-complete. Since the simultaneous crossing number for a single basic graph (i.e., $k = 1$ and $\mathcal{G} = G_1$) is equal to the ordinary crossing number, we have NP-hardness for SimCM. But the fact that an exponential number of phantom crossings may be necessary for any drawing realizing $\text{simcr}(\mathcal{G})$ raises the question of NP-membership of SimCM. We can prove this membership by showing a relation to the NP-complete *Weak Realizability* problem [132].

Definition 7.27 (Weak Realizability)**.** Given a graph $G = (V, E)$ and a set of edge pairs $\mathcal{R} \subseteq E^{\{2\}}$, does there exist a drawing of G where all crossing pairs lie in $\mathcal{R}$?

Note the difference between this realizability problem and the problem of strong realizability, cf. Definition 5.1: While in the latter we ask for a drawing with exactly the crossing edge pairs specified by $\mathcal{R}$, we now only specify *allowed crossings*, and ask for a drawing which only uses (some of) these allowed crossings.

Lemma 7.28. *SimCM reduces NP-many-one to Weak Realizability.*

Proof. Given a simultaneous graph $\mathcal{G}$ and a positive integral number h, we can test $\operatorname{simcr}(\mathcal{G}) \leq h$ in the following way. We guess $\ell \leq h$ pairs of edges with non-zero crossing cost and whose crossing costs sum up to at most h. Our guessing includes the order in which each edge is crossed. It is allowed to guess the same edge pairs multiple times. Each edge e involved in these crossing pairs is split into a path by introducing a new dummy vertex for each guessed crossing. The new dummy vertices have degree four as they are simultaneously used in the paths for both edges involved in the corresponding crossing. The guessed crossing order is maintained by the construction of the paths. This transformation can be realized in time polynomially in $|\mathcal{E}|$ and h.

We further define the set of allowed crossing pairs by the set of edge pairs that create a phantom crossing. Path edges inherit their original edge's crossing costs. Edges of the same constructed path are, by construction, not allowed to cross each other.

Notice that our guessing requires only a polynomial number of crossings as h and ℓ, the number of proper crossings, are polynomial (cf. Corollary 7.25). When we correctly guess the crossings (including the order for each edge) that correspond to a solution of the original SimCM problem, the constructed Weak Realizability problem solves the original SimCM problem as it will find the corresponding phantom crossings. □

Since we have NP-hardness for SimCM, we can conclude:

Theorem 7.29. *SimCM is NP-complete.*

Corollary 7.30. *SimCM$^+$ is NP-hard.*

This relationship between SimCM and Weak Realizability allows us to use a similar result for Weak Realizability (see [120, Theorem 2]) to show that there are at most exponentially many crossings per edge in an optimal drawing. Even more, we have an exponential bound on the number of all crossings of the drawing.

Theorem 7.31. *The cumulative crossing count* $\operatorname{ccc}(\mathcal{G})$ *in an optimal solution to SimCM$^+$ for a simultaneous graph $\mathcal{G}$ is at most exponential. There are at most $(4m)^{12m+24}$ phantom crossings where m is the number of edges in $\mathcal{G}$ while the number of proper crossings is polynomial.*

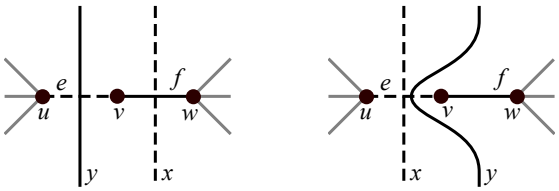


Figure 7.9: In general, a vertex with degree two cannot be shrunk by the traditional reduction techniques, cf. text. Solid edges are in G_1, dashed edges in G_2. All edges except for e and f are also in G_3.

A main difference to the traditional problem lies in the fact that the actual number of crossings is not proportional to the simultaneous crossing number as a higher number of phantom crossings but lower number of proper crossings is preferred to a higher number of proper crossings (independent of the total number of crossings). Thus, the overall number of crossings in a drawing of $\mathcal{G}$ that realizes SimCM or SimCM$^+$ cannot be bounded by the crossing number, the simultaneous crossing number, or by the number of crossings in some other drawing of $\mathcal{G}$.

7.3.3 Preprocessing

We discussed preprocessing routines for the traditional crossing number problem in Chapter 4; thereby we in particular concentrated on the SPR-tree based non-planar core reduction. However, these graph reduction techniques cannot be used for simultaneous crossing minimization in general. As we shall see, even trivial reductions are not always possible. We describe two valid reduction techniques for the SimCM and SimCM$^+$ problems:

Biconnected Components. Trivially, the simultaneous crossing number of a graph is the sum of the simultaneous crossing numbers of the connected components, when computed separately. Furthermore, as for the traditional crossing number problem, it is also valid to solve the problem for each biconnected component separately: The obtained drawings of the blocks can be glued together at their common vertices, without introducing additional crossings. This implies that edges with a degree one vertex can be removed from the problem instance recursively, as they will never cause a crossing in a drawing that realizes SimCM$^+$.

Chain Reduction. A *2-chain* is a pair of edges $e = \{u, v\}$ and $f = \{v, w\}$, in which the common vertex v has degree two. For the traditional crossing number problem, we could merge them into a single edge $g = \{u, w\}$. In general, this would be invalid for simultaneous graphs, cf. Figure 7.9: Assume that $e \in E_1 \setminus E_2$ and $f \in E_2 \setminus E_1$. Let there be two edges $x \in E_1 \cap E_3 \setminus E_2$

and $y \in E_2 \cap E_3 \setminus E_1$. Depending on the relative order in which they cross the 2-chain, they may induce either two phantom crossings, or one phantom and one proper crossing. Replacing e and f by some edge $g = \{u, w\}$ can never reflect both situations.

Nevertheless, we can define a valid chain reduction based on a subset relation. Let $\mathcal{B}(e)$ and $\mathcal{B}(f)$ be the sets of basic graphs that contain e and f, respectively. If $\mathcal{B}(e) \subseteq \mathcal{B}(f)$, we can replace the edges by $g = \{u, w\}$ with $\mathcal{B}(g) := \mathcal{B}(e)$. Clearly, this reduction can be performed recursively on the newly generated edge, in order to reduce even longer chains.

7.3.4 Heuristic Crossing Minimization

We consider the planarization heuristic as outlined in Section 2.3; we already discussed it in more detail in the context of extending it for the hypergraph and minor crossing number in Section 7.2. Recall that the heuristic consists of two steps: We first identify a maximum, maximal, or at least large planar subgraph; then we reinsert the temporarily removed edges one after another. Thereby each edge is inserted such that all new crossings lie on this edge; the crossings are replaced by dummy nodes, such that the intermediate graph always stays planar.

Planar Subgraph. If $\mathcal{G}$ is planar in the traditional sense, it is also simultaneously planar. Hence we can use any known algorithm to solve the maximum or maximal planar subgraph problem, as it is done for the traditional planarization approach. The resulting graph may not be maximally simultaneously planar, in the sense that we could insert additional edges without introducing proper crossings. But it is maximally simultaneously planar in the sense that it does not contain any phantom crossing and the insertion of any edge would lead to at least one phantom or proper crossing.

Edge Insertion. As described before, the reinsertion of an edge e into a planar graph G can be considered in two different settings: We may fix a combinatorial embedding of G and insert e into this embedding along a shortest-path in the dual graph of G. This can result in an unnecessary high number of crossings if the chosen embedding is disadvantageous. Hence, the preferable setting is to insert e optimally over all possible combinatorial embeddings, which can also be done in linear time, using SPR-trees.

We can adapt both algorithms for simultaneous crossing minimization by modifying the crossing cost calculations. Given an edge e and a planar simultaneous graph $\mathcal{G}$, we want to add e to $\mathcal{G}$ such that all introduced crossings lie on e and the simultaneous crossing number of the resulting graph is minimized. In traditional crossing minimization, the crossing cost for each edge f already in $\mathcal{G}$ is independent of e; in the unweighted case it is fixed to 1. However, in simultaneous crossing minimization the cost for crossing f depends

on e: the cost is given by $\mathrm{cc}(e, f)$, i.e., the number of basic graphs which contain both e and f. As mentioned above, we also want to minimize the number of phantom crossings and set the cost for edges f that do not have a common basic graph with e to some small positive number ε. Hence—for each reinsertion step and independently on whether we solve the insertion problem for a fixed embedding or over all embeddings—the crossing cost for each edge f in $\mathcal{G}$ must be calculated anew to reflect the current cost depending on the inserted edge e.

Clearly, the quality of the solution depends on the order in which the edges are inserted. As discussed in [76], there are several strong post-processing strategies, based on removing and reinserting edges after the first heuristic solution. All of them can also be used for the simultaneous crossing minimization.

7.3.5 Exact Crossing Minimization & Testing Planarity

As discussed above, a pair of edges may cross several times, even an exponential number of times. This renders the variable concept of OECM useless: The idea of uniquely specifying that an edge f crosses some edge e before an edge g crosses e breaks when f crosses e multiple times, sometimes before, sometimes after g crosses e. Generally, there is no way to uniquely model multiple crossings per edge pairs with a cubic number of binary linear-ordering variables. Hence we will consider the SECM formulation and extend it to solve the phantom crossing aware simultaneous crossing minimization problem.

Variables and Expansion of Edges. For the traditional crossing number, we are able to bound the number of crossings per edge by the number of graph edges and by any heuristic solution. We use this bound ℓ to expand the graph and solve the simple crossing number on $G^{[\ell]}$ or $G^{[\ell-1]}$. As discussed in Section 7.3.2, there are no bounds of practical interest in case of the SimCM$^+$ problem. We define $\mathrm{ccc}(\mathcal{G}, \mathcal{D})$ as the sum of phantom and proper crossing in the drawing $\mathcal{D}$ of $\mathcal{G}$. We cannot even use $\mathrm{ccc}(\mathcal{G}, \mathcal{D})$, for some heuristically computed $\mathcal{D}$, as a valid upper bound for the number of crossings per edge: The optimal drawing $\mathcal{D}^*$ might have fewer proper crossings than $\mathcal{D}$, but require many more phantom crossings, possibly resulting in $\mathrm{ccc}(\mathcal{G}, \mathcal{D}^*) > \mathrm{ccc}(\mathcal{G}, \mathcal{D})$.

Hence we consider a maximum expansion with exponentially many segments per edge. This drawback, although terrifying on first sight, turns out to be of little relevance in practice, since the column generation scheme does not generate this enormous amount of additional variables in any of our tests. We observe that now the use of the column generation scheme is not only an enhancement to improve the running time, but that it is a compulsory element of the algorithm.

The original approach allows to leave out the variables for adjacent edges e and f. Additionally, it may use constraints that allow at most one crossing

per pair of original edges. Both these possibilities are no longer valid.

Crossing Costs and Column Generation Scheme. The main idea for computing a solution to the SimCM$^+$ problem with our ILP is to use the crossing cost $\mathrm{cc}(e, f)$ as the coefficient of the variable $x_{e,f}$ in the objective function, for each pair of edges e, f. Since phantom crossings correspond to $\mathrm{cc}(e, f) = 0$, the ILP would not solve SimCM$^+$ correctly. We set the crossing costs for phantom crossing to some small value $\hat{\varepsilon} > 0$. Clearly, as we know there can be exponentially many phantom crossings in an optimal drawing, $\hat{\varepsilon}$ would have to be exponentially small such that $k\hat{\varepsilon} < 1$ for each arising number of phantom crossings k. Conceptually, we can do the following: We start the algorithm assuming that the number of phantom crossings will not be too large, and choose some small $\hat{\varepsilon}$. When inspecting a fractional solution, we can check if the objective function is perturbed too strongly by these values: Whenever it is, we choose $\hat{\varepsilon} := \hat{\varepsilon}/2$ and re-solve. Yet, for all practical purposes the BIP will not allow too large crossing numbers to be computed anyhow, due to the problem's NP-hardness. So this does not become an issue in practice.

The $\hat{\varepsilon}$-approach comes with further certain challenges regarding the column generation scheme: The central idea of the combinatorial column generation scheme is to start with a primary segment per edge, adding secondary segments when necessary. It is crucial for the algorithm that these additional segments are a bit cheaper to cross than the primary segment of an edge, say by some small enough $\varepsilon > 0$. Hence we have to be careful about mixing these two different epsilons. Note that also crossings with cost $\hat{\varepsilon}$ have to be reduced by ε for secondary segments. Hence we require $\varepsilon < \hat{\varepsilon}$ to ensure positive costs. We cannot choose $\varepsilon \ll \hat{\varepsilon}$ due to the risk of numerical instabilities.

The bounding schemes can easily become unstable, i.p., simply rounding the objective function up will in general not give a valid lower bound for $\mathrm{simcr}(G)$: In the original column generation approach, using a suitable ε, the objective value of the ILP can always be rounded up to obtain the unskewed integer objective value, i.e., any solution of the ILP is of the form $\lceil \mathrm{cr}(G) - \ell' \cdot \varepsilon - \ell'' \cdot \varepsilon^2 \rceil = \mathrm{cr}(G)$. Hereby, ℓ' and ℓ'' reflect the number of crossings between primary and secondary, and between two secondary segments, respectively. We loose this property by the introduction of $\hat{\varepsilon}$, since crossings with such cost have to be rounded down to obtain the graph theoretic crossing cost 0. Hence the ILP solution is of the form $\mathrm{simcr}(\mathcal{G}) - \ell' \cdot \varepsilon - \ell'' \cdot \varepsilon^2 + \hat{\ell} \cdot \hat{\varepsilon}$ which can be greater than $\mathrm{simcr}(\mathcal{G})$.

Overall, we can in fact use both ε and $\hat{\varepsilon}$ simultaneously, choosing, e.g., $2\varepsilon = \hat{\varepsilon}$, but we must adapt all bounding schemes accordingly. By choosing the epsilons carefully, we can change the $\lceil . \rceil$-function into a rounding scheme which transforms any value within the interval $[a - 0.5, a + 0.5)$ into the integral value a, and still use the combinatorial column generation scheme. Alternatively, we can compute the unskewed objective function (i.e., without

any ε or $\hat{\varepsilon}$) based on the fractional solution and use this for bounding purposes.

With the use of $\hat{\varepsilon}$ and a conceptually exponentially large graph expansion, it is clear that neither SECM's Kuratowski-constraints, nor their separation, has to be modified and we obtain:

Theorem 7.32. *The* SECM *formulation, when modified as described above, solves SimCM$^+$ to provable optimality.*

Corollary 7.33. *The* SECM *formulation, when modified as described above, can be used for the NP-complete problem of testing simultaneous planarity: A graph is simultaneous planar if and only if the 0/1-ILP has an optimal solution with* $\operatorname{simcr}(\mathcal{G}) = 0$.

7.4 Further Crossing Numbers

In [25], Buchheim et al. presented how the SECM formulation can be extended to solve the *bimodal crossing number*. This crossing number is defined as the traditional crossing number, but—considering some fixed orientation of the given graph—requires that the ingoing and the outgoing edges appear consecutively in the resulting drawing.

In order to solve this crossing number, we modify the given graph by expanding each original vertex v into two connected vertices v_i and v_o, similar to the inverse minor operation described in Section 7.2. The node v_i is incident to all ingoing edges of v, and v_o is incident to all outgoing edges of v_o. We can then forbid that the new edges $\{v_i, v_o\}$ (for all nodes v) are crossed, simply by fixing the corresponding variables to 0. By shrinking the node pairs into their original counterparts after the computation, we obtain a bimodal drawing of the graph with the minimum number of crossings.

The same modification is also possible for the OECM formulation. Since OECM is preferable over SECM, this yields a formulation for the bimodal crossing number preferable to the one presented in [25].

We want to conclude this section with noting that there are also many other crossing numbers which are of theoretical or practical interest. Some of these, like crossing numbers on surfaces of higher genus, are hard to formulate similar to OECM or SECM, as the planarity classification based on Kuratowski subdivisions no longer holds.

The probably best-known other crossing number variants on the plane, which up to now are of mainly theoretical interest, are the *pairwise crossing number* $\operatorname{pcr}(G)$ and the *odd crossing number* $\operatorname{ocr}(G)$, see [119]. They are defined similar to the traditional crossing number, but in the former we do not count the crossings, but the number of edge pairs that cross; i.e., we only care if two edges e, f cross of not, but we do not care if they cross once or multiple times. For the latter crossing number we only count the pairs of edges which cross an odd number of times.

Clearly we have

$$\operatorname{ocr}(G) \leq \operatorname{pcr}(G) \leq \operatorname{cr}(G)$$

for all graphs G. For a long time, it has been an open question whether equality holds. In 2005, Pelsmajer et al. [124] showed that there are graphs for which $\operatorname{ocr}(G) \neq \operatorname{cr}(G)$. For the pairwise crossing number the answer is still unknown. The OECM formulation seems inappropriate for computing the pairwise crossing number, as its variable structure cannot describe multiple crossings of the same edge pair. The subdivision strategy of SECM on the other hand, may be suitable for such an extension. Considering the fact that the counter-example presented in [124] for the odd crossing number is quite small (when edge weights are allowed), it might be an interesting approach to have an optimal pairwise crossing minimizer, when seeking for counter-examples—if there are any.

Part III

Experiments & Outlook

Explanatory Note

crossing: *(Road Construction)*
A place where two or more routes of transportation form a junction or intersection.

In this part of the thesis, we will investigate the practical performance of the algorithms, the influence of different strategies, etc. In Chapter 8, we report on experiments regarding our approaches for the traditional crossing number on real-world graphs. Afterwards, we look at our algorithms for other crossing numbers. Chapter 10 will give an overview on ongoing research: We investigate how our ILP approach can be applied to graph theoretic questions regarding the crossing number of special graph classes. We conclude with a summary and outlook in Chapter 11.

We practically evaluated all relevant algorithms, schemata and routines described in Part II of this thesis. They were implemented using the free open-source[1] *Open Graph Drawing Framework* (OGDF) [118]. This framework is mainly developed here at the Chair for Algorithm Engineering at the Dortmund University of Technology. At the time of this writing, all of the algorithms described in this thesis, apart from the branch-and-cut-and-price algorithms themselves, are contained in the publicly available package. Furthermore, our code uses the free open-source[2] Abacus branch-and-cut-and-price framework [1, 94] in conjunction with the commercial LP-solver *Ilog CPLEX 9.0* [90]. Until specified otherwise, we ran our experiments on an AMD Opteron 2.4 GHz, 32bit, 2GB RAM per process, under Debian Linux. Although the machine offers 4 cores, we used only a single one per process.

[1] GPL – GNU General Public License, v2 and v3.
[2] LGPL – GNU Lesser General Public License, v2.1 and above.

Chapter 8

Experiments: Crossing Number

crossing: *(Nature)*
A shallow area in a stream that can be forded.

In this chapter, we concentrate on the traditional crossing number. First we introduce the *Rome graph library* used for benchmarking and discuss the efficiency of our non-planar core preprocessing scheme. The central part of the experiments is our investigation on the practical performance of SECM and OECM, as well as the effectiveness of our column generation schemata. After obtaining provable optimal solutions for most Rome graphs, we can use these to study the quality of the currently best known crossing number heuristic.

8.1 The Rome Graph Library

We consider the *Rome* graph library. This benchmark set was collected and introduced by Di Battista et al. at the eponymous University of Rome III [45]. It has been widely used to evaluate various graph drawing algorithms since then, in particular also crossing number heuristics, see, e.g. [76].

The set contains 11,389 graphs that consist of 10 to 100 nodes and 9 to 158 edges. These graphs were generated from a core set of 112 real-world graphs originating from database design and software engineering applications. Most of the graphs are sparse, which is a common property in most application areas of automatic graph drawing. The average ratio between the number of edges and the number of nodes of the graphs is about 1.35. The benchmark set contains 8,249 non-planar graphs; their average degree is 2.69.

To understand the test set better, it is worth looking at Figure 8.1. The library consists of many planar graphs and non-planar graphs for which we

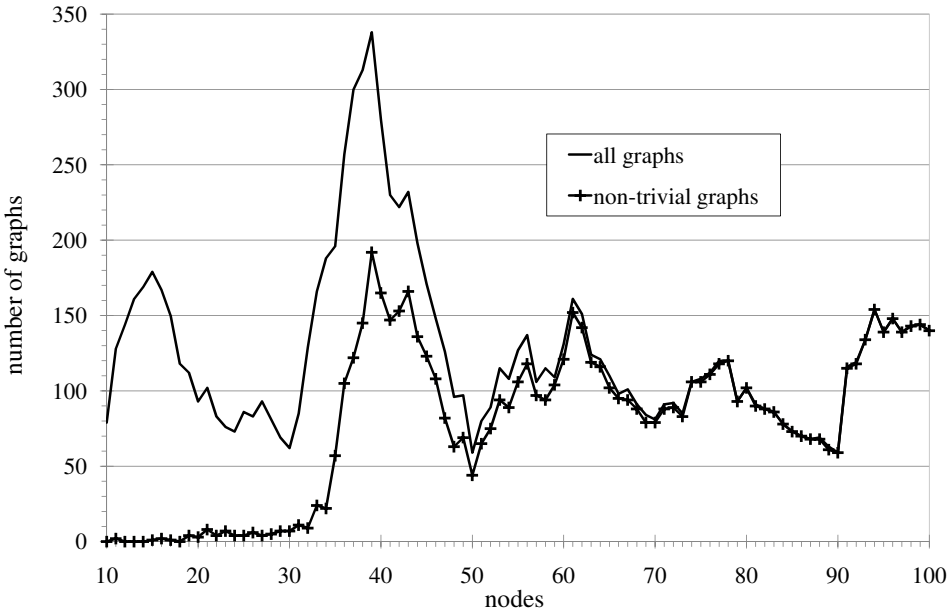


Figure 8.1: Number of non-trivial graphs compared to all graphs in the benchmark set.

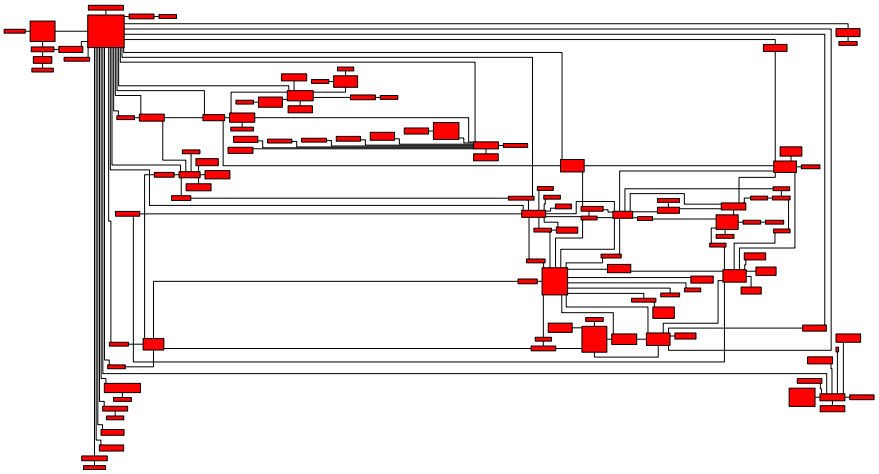
know that their crossing number is 1, based on the primal heuristic; we call these graphs *trivial*, since they are of no interest for us. As we can see, there are only very few graphs with up to 32 nodes which are non-trivial.

8.2 Non-Planar Core Reduction

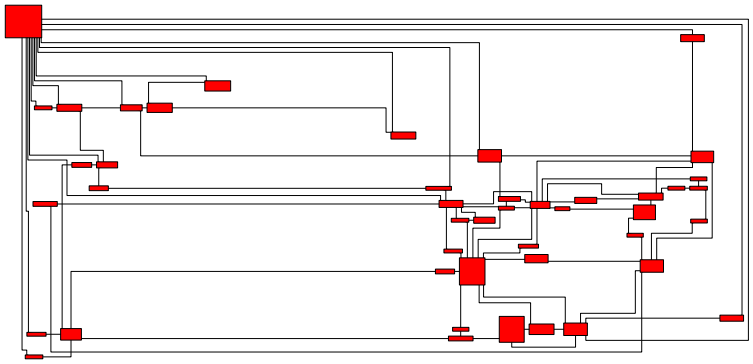
Figure 8.2 serves as a motivation. We see a database scheme of a retirement fund company with 117 nodes and 153 edges (ignoring the graph's second connected component which is a simple chain of four nodes). It contains a single non-planar block with 45 nodes and 80 edges. The non-planar core of this graph has only 21 nodes and 47 edges. The graph becomes simple enough that our heuristic directly computes a drawing with 7 crossings; our ILP can easily prove that this is in fact the optimal value.

In our experiments, we consider the 8,249 non-planar graphs in the Rome library; their average node degree is 2.69. We find that all these graphs have a single non-planar block whose non-planar core is the skeleton of a single R-node. The average node degree in these non-planar blocks is 2.89.

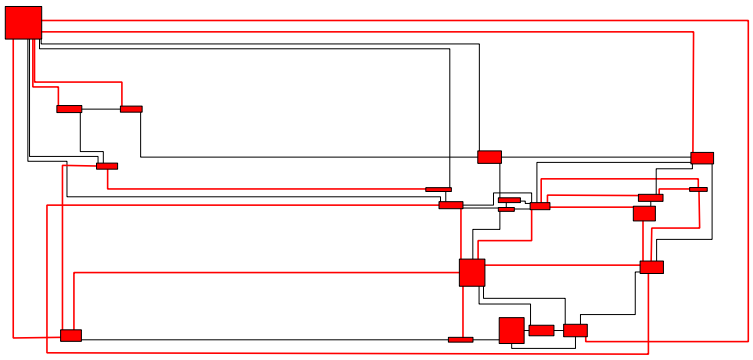
Figure 8.3 shows the average relative size of the non-planar core $\mathcal{C}$ compared to the non-planar block and compared to the whole graph. Here, the size of a graph is simply the number of its edges. The size of each circle in the diagram reflects the number of graphs at the data point. Hence, larger circles correspond to statistically more reliable data points. It turns out that, on av-



(a) Non-planar component: 117 nodes, 153 edges



(b) Non-planar block: 45 nodes, 80 edges



(c) Non-planar core: 21 nodes, 47 edges

Figure 8.2: An example graph, coming from a retirement company, cf.text. Note that the crossed, left-most virtual edge in the core has weight 2.

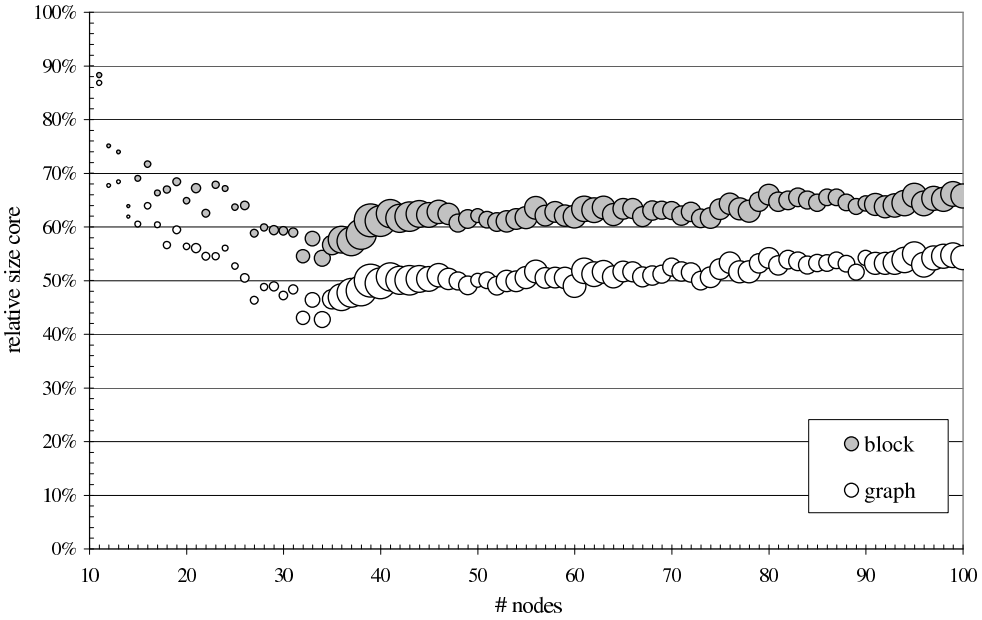


Figure 8.3: Relative size (w.r.t. the original graph size) of the non-planar core for the Rome graphs.

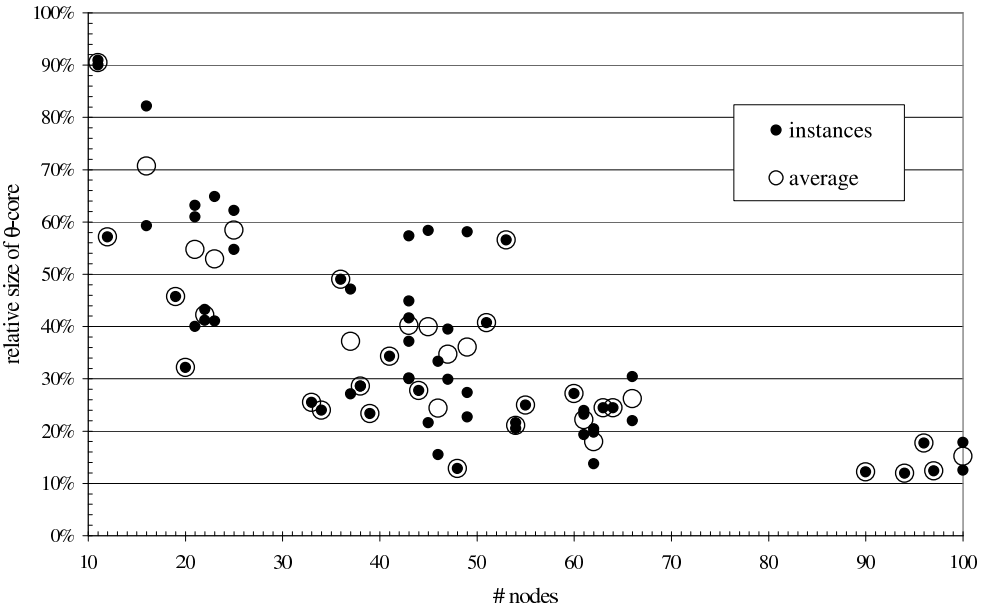


Figure 8.4: Relative size (w.r.t. the original graph size) of the θ-core for the Rome graphs (if unsolved).

erage, the size of the non-planar core is only 2/3 of the size of the non-planar block. Compared to the whole graph, the size of the non-planar core reduces to about 55% on average. This shows that the new preprocessing technique provides a significant improvement for reducing the size the real-world graphs.

Furthermore, we want to clarify that not only exact approaches, but also heuristics profit from the application of the non-planar core. Not only does it reduce their running time, but also the computed number of crossings decreases, as the problem tends to get easier for the heuristics.

We also analyzed the improved reduction strategy for the thickness presented in the end of Section 4.3.2. We call the set of graphs obtained after exhaustively applying this reduction the "θ-core" of the graph. For only 64 of the Rome graphs the $\theta-$-core is not empty, and hence the thickness problem can be solved for 99.4% of the graphs (11,464 out of 11,528) simply by planarity testing and preprocessing. Figure 8.4 shows the relative size of the θ-core for all 64 unsolved instances (filled circles) along with the average relative size grouped by number of nodes (unfilled circles); the average number of edges in the θ-core of the unsolved instances is 21.7.

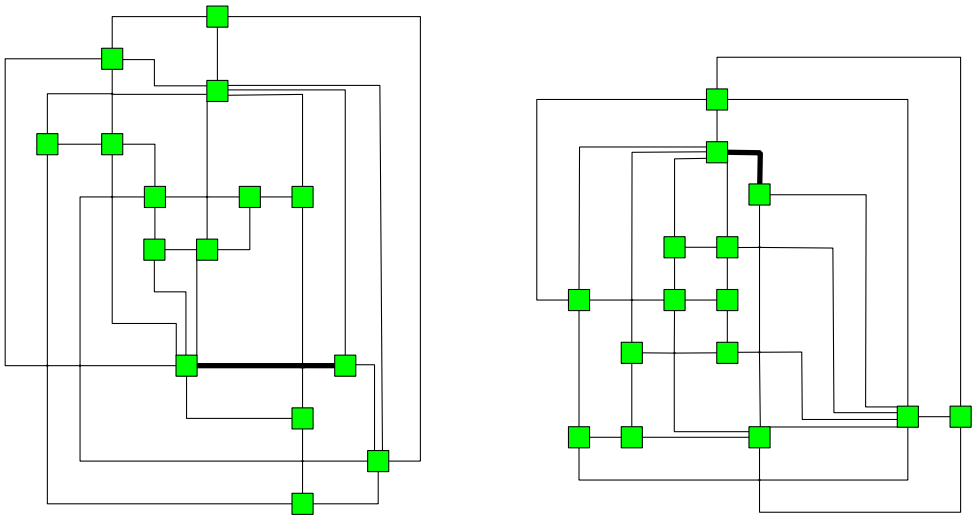
8.3 Exact Crossing Minimization

Figure 8.5 serves as a motivation. We see a Rome instance with originally 37 nodes and 53 edges. Its non-planar core consists of 15 nodes and 29 edges, which is the graph depicted in the figure. While the heuristic requires 9 crossings, our algorithms finds the provable optimal solution with only 5 crossings.

Developments and Progress. We do not only want to show the current applicability of our ILP approaches, but also the chronological progress over the last few years. Table 8.1 shows the algorithms and their key properties for all milestones of the algorithmic development or the ILP formulations.

The first implementation (S) is a special case as it was not implemented by the author of this thesis but by Dietmar Ebner as part of his Diploma thesis. The corresponding experiments were conducted on a Intel Pentium 4 (2.4 GHz) with 1 GB RAM. All experiments but the ones for this first variant were performed on the AMD Opteron described in the beginning of this part of the thesis. We note that between S and S+, there has been a full re-implementation of the former by this thesis' author, using COIN-OR [40]. This implementation performed similar to S+ but was discontinued: At that time, COIN-OR did not offer a column generation framework as mature as Abacus. Parts of the code of this intermediate version where later developed into S+. Table 8.2 summarizes the main parameters used in this study, if not specified otherwise.

The *success ratio* of our algorithm is the percentage of how many instances



(a) Heuristic solution: 8 crossings (b) Optimal solution: 5 crossings

Figure 8.5: Rome graph example (No. 11627): a non-planar core with 15 nodes and 29 edges. The thick edge is the only one with weight 2.

Algorithm	ILP	Column Gen.	Graph library	ILP library
Generation 1: pure branch-and-cut				
S [24]	SECM	(none)	Leda [112], AGD [2]	(direct)
	Note: This is the only implementation which, due to the direct use of CPLEX (v8.1), was able to profit from CPLEX's own integer cuts. Preprocessing: consider only blocks, shrink chains into edges. Implementation done by Dietmar Ebner.			
Generation 2: branch-and-cut-and-price, non-planar core reduction				
S+ [30]	SECM	(none)	OGDF	ABACUS
	Note: full re-implementation. Implementation identical to SA/SC below, but column generation turned off.			
SA [30]	SECM	Algebraic Pr.	OGDF	ABACUS
SC [23, 30]	SECM	Combinatorial	OGDF	ABACUS
	Note: Column generation criterion applied to fractional values instead of rounded solution.			
Generation 3: tuning regarding column generation; efficient Kuratowski extraction (Section 6.1.2); new formulation. — These are the algorithms as described in this thesis.				
SC+ [37]	SECM	Combinatorial	OGDF	ABACUS
OC [37]	OECM	Combinatorial	OGDF	ABACUS

Table 8.1: Exact crossing minimization algorithms.

Algorithm	Description
Heuristics	
S	• AGD's default planarization heuristic. • Settings as suggested in [76].
others	• OGDF's default planarization heuristic (equivalent to AGD's heuristic, but supports post-processing option *incremental*, cf. text on page 157). • Settings equivalent to above, but for the initial upper bound we use: 10 independent runs of the heuristic, 100 permutations of the insertion order, and *incremental* post-processing.
Kuratowski Separation	
S	• Runs SimpleExtract to extract a Kuratowski subdivision K. • Adds corresponding cut if violated. • Removes one of the edges of K from P, and iterates the above steps.
S+, SA, SC	• We perform 50 independent runs of SimpleExtract. • We add the 20 most violated cuts. • If no violated cut was found, we try up to 250 further extractions until at least one violated constraint is found.
SC+, OC	• We perform 20 independent runs of MultiExtract. • In each run, we extract at most 100 subdivisions and buffer the 30 most violated corresponding cuts. • We add the 40 most violated cuts over all buffered cuts. • If no violated cut was found, we try up to 80 more runs until at least one violated constraint is found.
Algebraic Pricing	
SA	• For each edge pair (in random order), we identify up to 5 variables with smallest reducted cost. • We stop after identifying 300 such variables. • We activate the 50 variables with smallest reduced cost.

Table 8.2: Main algorithmic parameters used in this study.

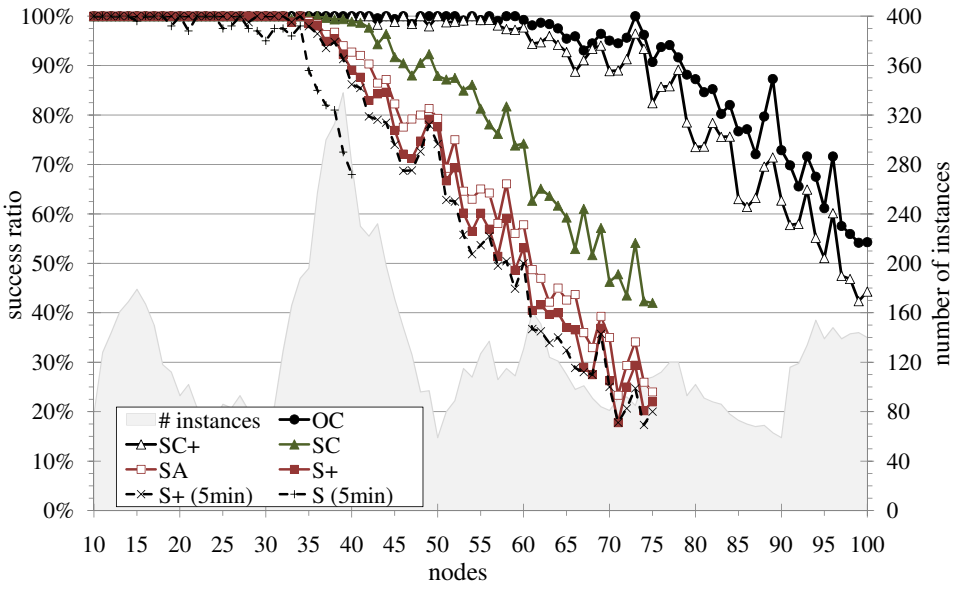


Figure 8.6: Success ratio of the various algorithms after 30 minutes of computation (if not otherwise stated).

it can solve within the time limit. In Figure 8.6 we consider the various milestones of the algorithmic and formulation-specific development. We show an overview over the success ratios w.r.t. the number of nodes of the graphs, within a time bound of 30 minutes per graph instance. As only 5 minutes were considered for S in [24], we also give the results for S+ after 5 minutes. Observe that for S+, the difference between both time bounds is rather small.

Table 8.3 shows some corresponding statistical data. Note that only graphs with up to 40 nodes were considered for S; the 2nd generation algorithms were run for up to 75 nodes. Only the 3rd generation algorithms were efficient enough to reasonably consider all Rome graphs.

Column Generation Strategies. In order to compare our two different column generation schemes for SECM in a fair way, we use the 2nd generation algorithms SA and SC. Since the success ratio drops rapidly for large graphs, we restrict ourselves to the graphs with up to 75 nodes. For each computation scheme, we apply a 30 minute time limit per instance.

Figure 8.7 shows the percentage of graphs for which the exact crossing number was computed. The size of the circles denotes the number of graphs per node count. Hence, larger circles correspond to statistically more reliable data points. While S+—the ILP without column generation—can only solve a third of the large graphs, SC can still solve about 50%. Note that even after only 5 minutes, SC can already solve more graphs than S+ and SA after 30. Furthermore, there is no instance which can be solved by either S+ or SA,

Alg.	*max*	at least 100%	at least 95%	$\|V\|$ 38–40	$\|V\|$ 58–60	$\|V\|$ 73–75	$\|V\|$ 88–90	$\|V\|$ 98–100
S *	40	14	35	73.8%	—	—	—	—
S+ *	75	27	36	90.7%	48.4%	20.7%	—	—
S+	75	32	36	92.2%	53.6%	23.9%	—	—
SA	75	32	38	94.5%	60.0%	28.0%	—	—
SC	75	32	42	99.2%	76.6%	46.1%	—	—
SC+	100	38	60	99.9%	97.4%	90.8%	67.9%	44.5%
OC	100	42	66	100%	99.7%	95.7%	80.0%	54.8%

Table 8.3: Success ratios of the various algorithms after 30 minutes (5 minutes, if marked with *) of computation. *max* gives the number of nodes of the largest considered graphs. The two *at least X %* columns give the maximum number of nodes N such that for each class of graph size $|V| \leq N$ at least $X\%$ were solved. Conversely, the $|V| = Y$–Z columns give the success ratio over the graphs classes with size Y–Z.

but remains unsolved by SC.

As it turns out, the number of edges in the non-planar core is much more influential than the number of nodes in the original graph. Figure 8.8 shows the relationship between those two properties: While we can clearly see a correlation, the variance is quite high. When we look at Figure 8.9 we can see the clear dependence of a successful computation on the number of core edges.

For the following observations on the number of required variables, we only consider non-trivial graphs that are solved to provable optimality. Since SC is able solve many more instances than SA, we will in particular consider the *common set*, which is the set of instances solved by SA and therefore also by SC. Figure 8.10 shows the number of variables used by SA and SC, relative to the full (potential) variable set. While SC needs more variables on average for all the graphs it can solve, it uses less variables than SA when compared on the common set. This shows that SC is able to solve graphs which require a larger variable set, whereby SA is too slow to tackle such graphs successfully.

Figure 8.11 shows the actual numbers of generated variables (compared on the common set). Note that the number of variables generated by SC stays very close to the number of the initially generated variables. While SA can only solve graphs with roughly 10,000 potential variables on average, a similar statistic for all the graphs solved by SC shows an average of between 30,000 and 40,000 potential variables.

According to the above statistics, it is obvious that the running time of SC is superior to SA's, and that both are more efficient than no column generation at all. Yet, it is impressive that—over the set of instances solved by S+—even the maximum running time of SC is always far below S+'s

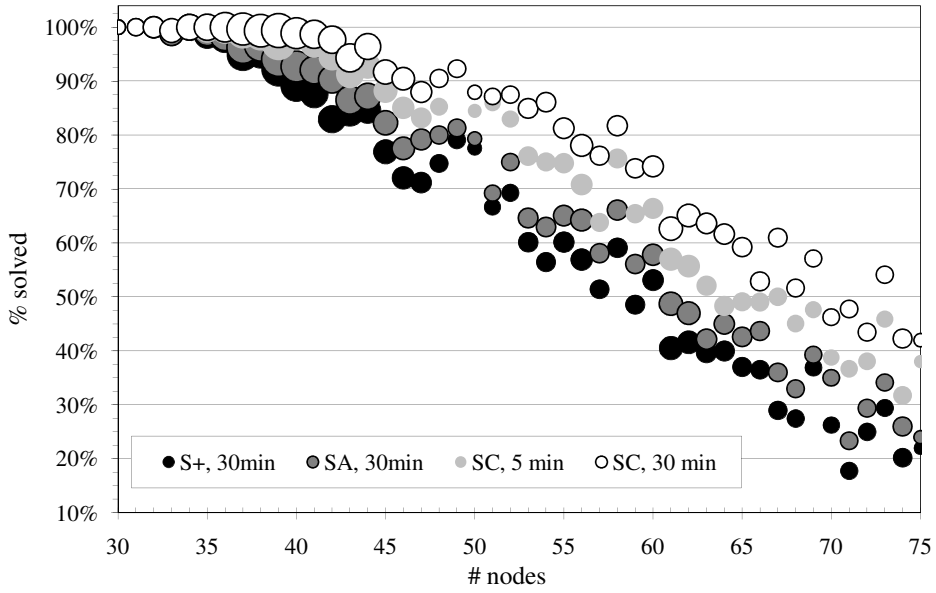


Figure 8.7: Percentage of graphs solved by algorithms of Generation 2.

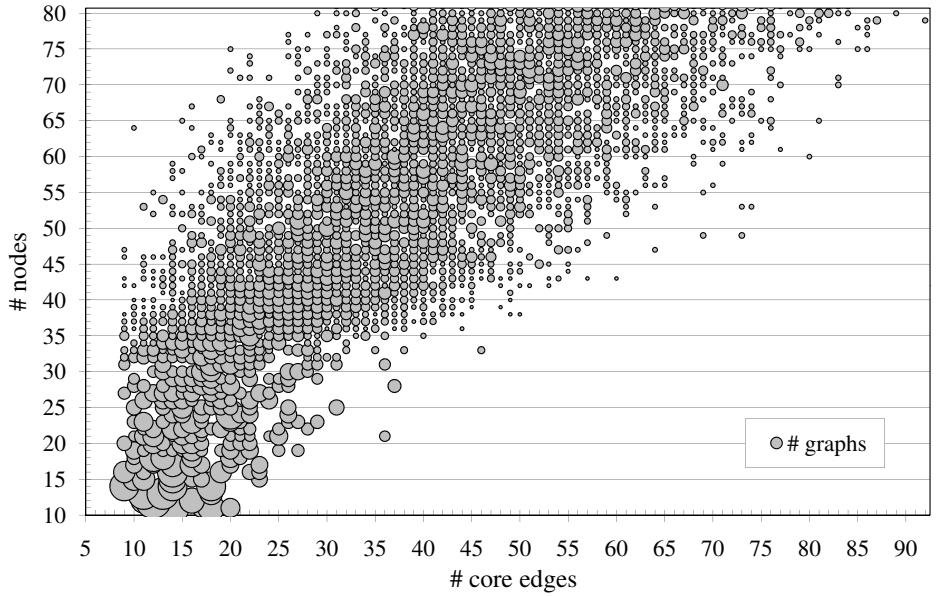


Figure 8.8: Correlation between number of nodes and number of core edges.

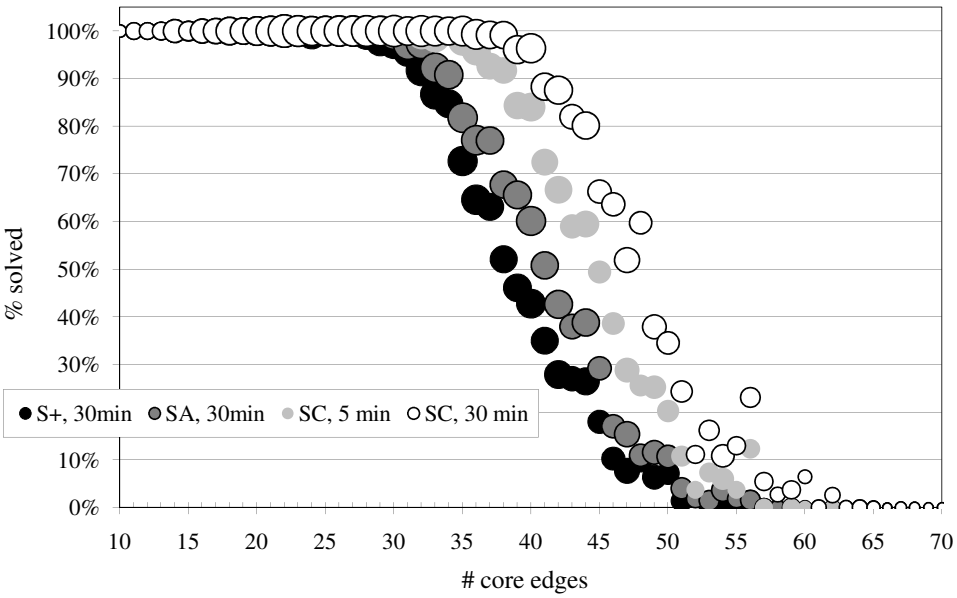


Figure 8.9: Success ratio of the 2nd generation algorithms.

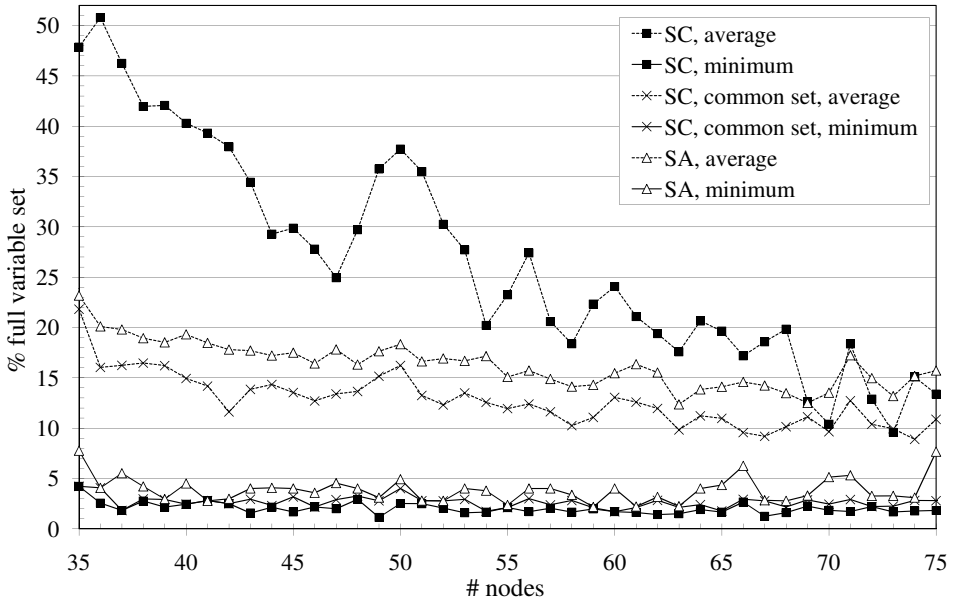


Figure 8.10: Number of generated variables, relative to the full variable set.

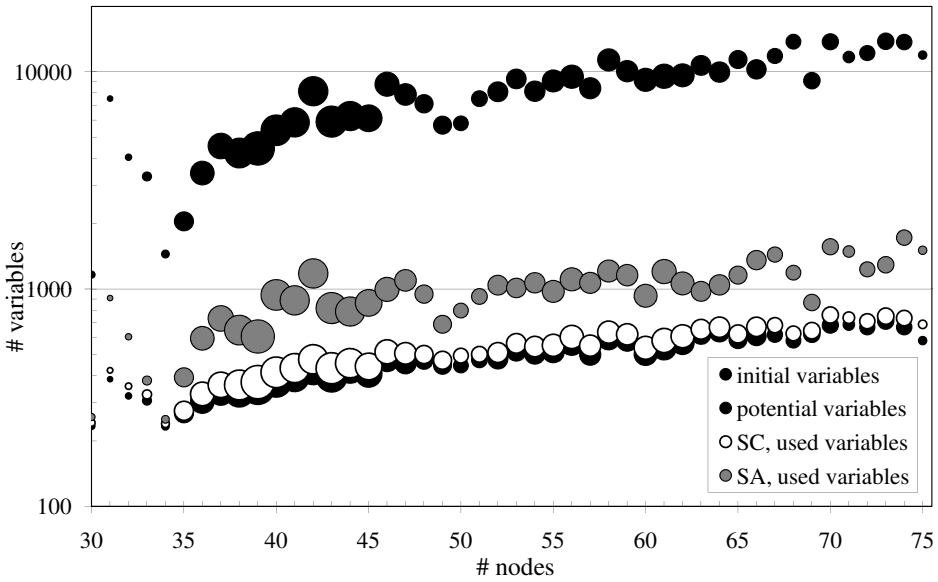


Figure 8.11: Number of generated variables (common set).

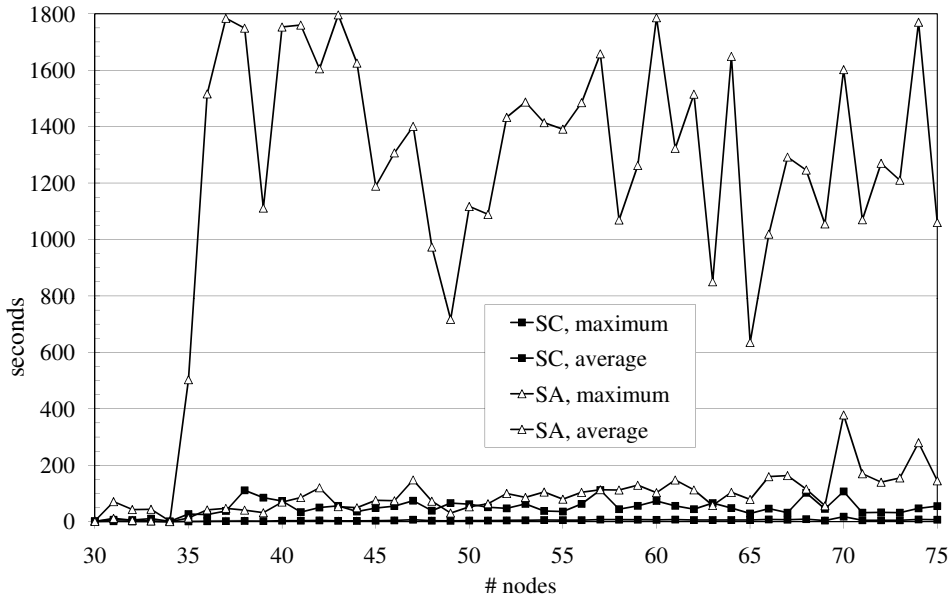


Figure 8.12: Running time (common set).

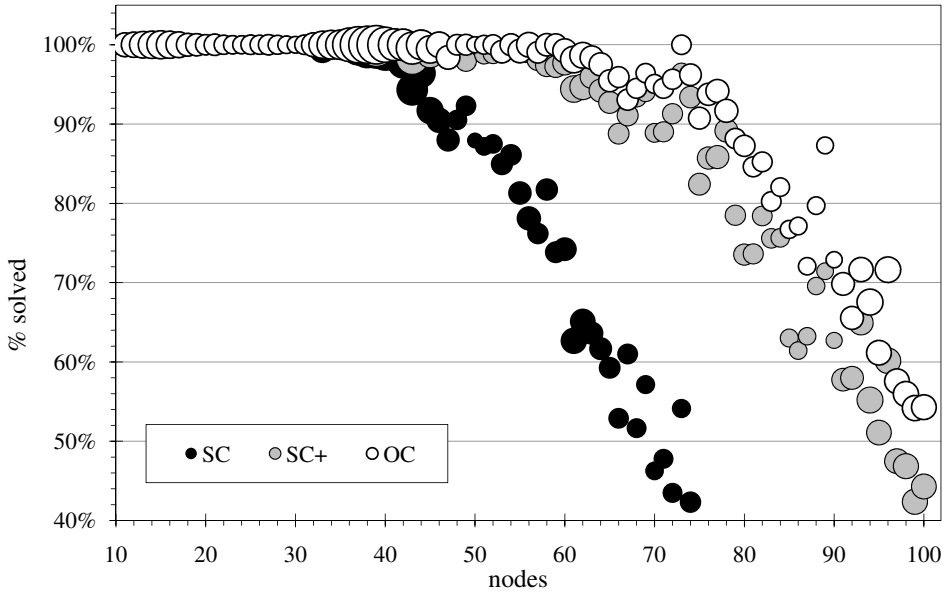


Figure 8.13: Success ratio of the 3rd generation algorithms (and SC).

average. When comparing SA to SC on the common set (Figure 8.12), we see that SA's average is about equal to SC's maximum time. The average running time of SC over all successfully solved graphs is under 5 minutes.

Subdivision-based vs. Ordering-based. We now compare our two ILP formulations SECM and OECM with each other. For the former, we choose its strongest algorithmic variant SC+ that is—as far as possible—equivalent to OC, except for the underlying ILP model and column generation scheme. As we can see in Figure 8.13, both algorithms clearly outperform SC, which drops below a success ratio of 50% for graphs of 70 nodes. While OC can solve virtually all graphs with up to 60 nodes to provable optimality within the time limit, SC already drops to a 70% success ratio for graphs of size 60. The experiments also show that the linear-ordering based ILP formulation of OC is able to solve more and larger graphs than SC+: While SC+ can only solve 84.4% of all non-trivial graphs within 30 minutes, OC finds and proves an optimal solution in 89.2% of all these instances, i.e., 93.3% over all benchmark instances. Even when OC has a time limit of only 10 and 5 minutes per non-trivial instance, it can still solve 85.9% and 83.4%, respectively, and thus produces results comparable to 30 minutes of SC+ in a 3–6x shorter period of time.

There are only 19 instances solved by SC+ but not by OC, within 30 minutes, but 361 instances which OC solves but SC+ does not, cf. Figure 8.14. Most importantly, we can now solve over 50% of the largest graphs of the Rome library. Figure 8.15 further illustrates the strength of OC; it shows the

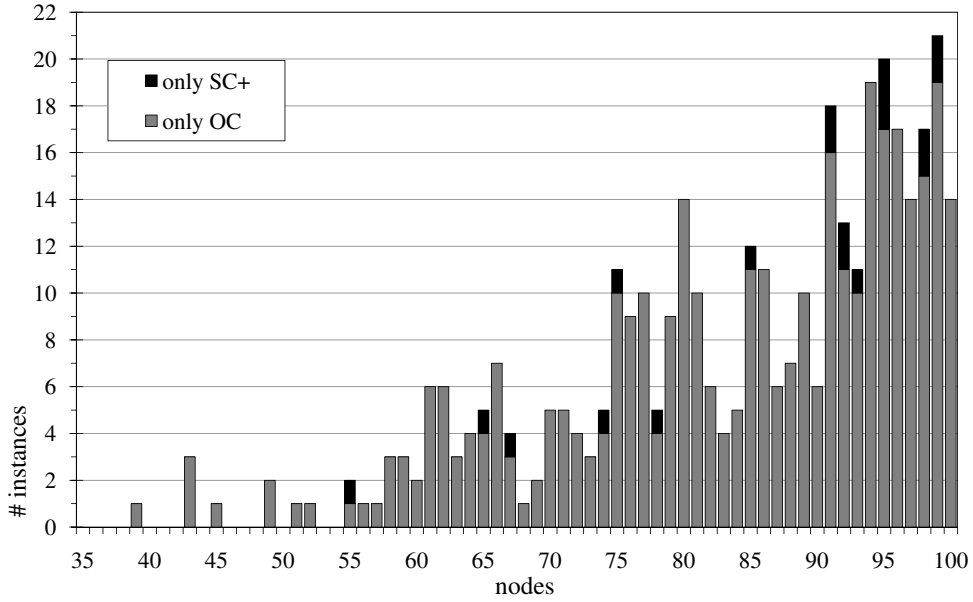


Figure 8.14: The number of instances only solved by either SC+ or OC, but not by both.

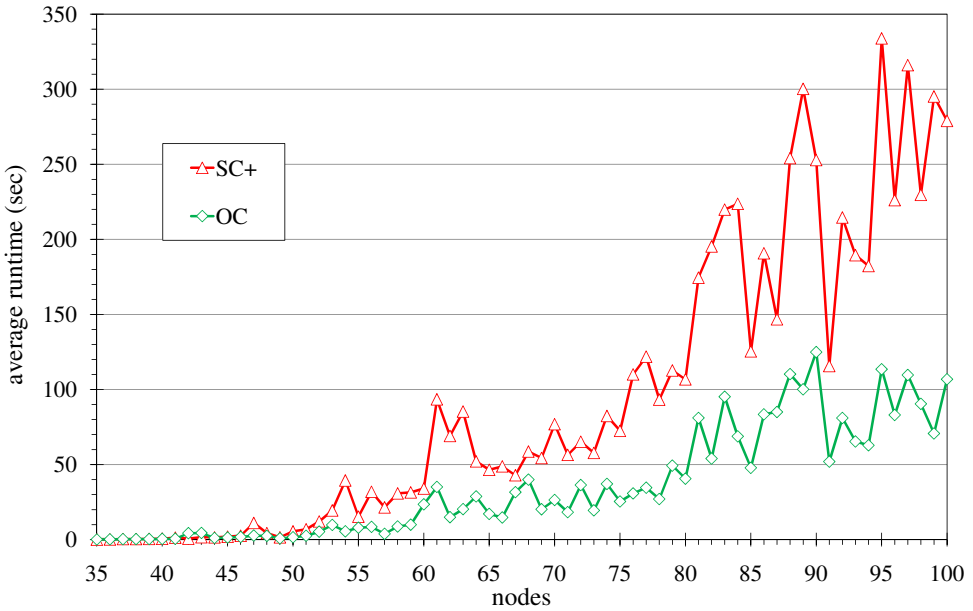


Figure 8.15: The required running time, averaged over the instances solved by both SC+ and OC.

average running times for graphs solved by both approaches. Even for large graphs, OC only requires roughly 100 seconds on average.

Figure 8.16 shows the dependency of the solvability on the crossing number: We see that OC can solve all but 6 graphs with a crossing number of up to 20. It even solves a graph with a crossing number of 37. In contrast to this, SC+ solves only all but 7 graphs with a crossing number of at most 12.

Finally, Figure 8.17 shows a comparison of the number of required variables for the instances solved by both approaches: Both algorithms start with the same initial variable set, but OC requires far less additional variables during the computation of the optimal solution. This seems to be the main reason why OC is faster and more efficient than SC and SC+.

Root-node vs. Branching & Separation. Considering the strongest algorithm OC, we investigate its algorithmic behavior. We are in particular interested in finding out if our formulation is strong enough to solve many problems in the root node of the branch-and-bound tree. In Figure 8.18, we consider all graphs that are solved to provable optimality in 30 minutes. We only have to branch very rarely; even for the largest graphs, we solve 80% of them within the root note. If we do need to branch, we require less than 8 subproblems on average. In Figure 8.19, we analogously look at the instances that remain unsolved after 30 minutes. Even in these cases, we do not branch often, but regularly time out while still in the root node. We observe that this is especially the case when the graphs get larger. Interestingly, using a time-out limit of one hour gives virtually the same results. It seems that already solving the root node can become too expensive for large, complex graphs.

For Figure 8.20, we consider all graphs that are solved to optimality and require branching. We observe that the rounded lower bound at the root node is nearly always corresponding to the optimal solution, while the heuristic upper bound is relatively far away. Yet, this statistic is skewed in the sense that it does not and cannot include the instances for which no optimality proof is found due to weak lower bounds, although the upper bounds might be tight.

Finally, Figure 8.21 shows the average number of required constraints. We note that separating the simple triangle constraints does not influence the overall running time in any statistically relevant way for the Rome graphs. They are only useful when considering complete graphs, cf. Section 10.2.1. We therefore do not use them in these experiments. Also note that virtually all separated Kuratowski constraints are $K_{3,3}$ subdivisions.

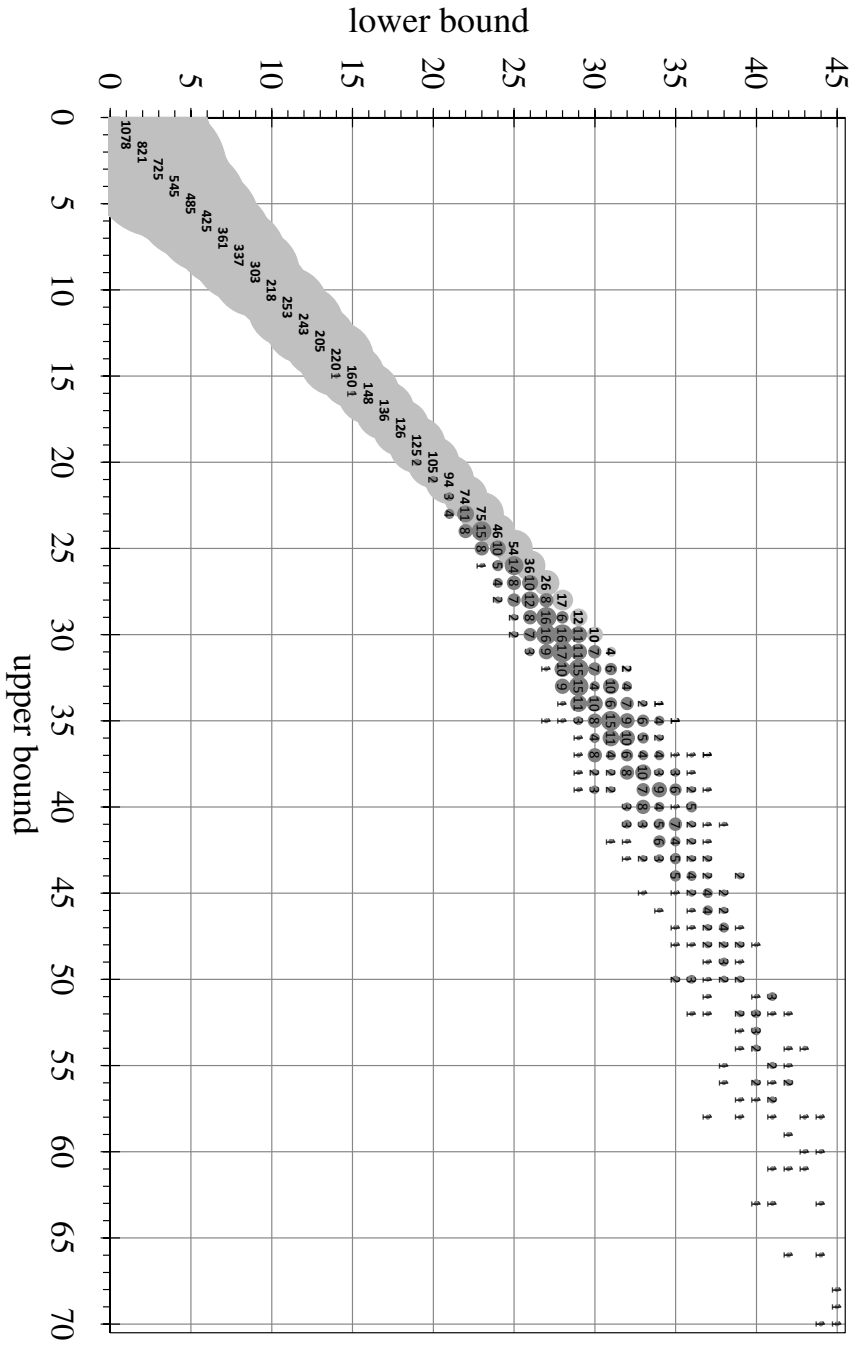


Figure 8.16: The number of instances per combination of lower and upper bound after 30 minutes of OC, over all graphs of the Rome library. 9 instances are not shown as their lower or upper bounds do not fit into this diagram.

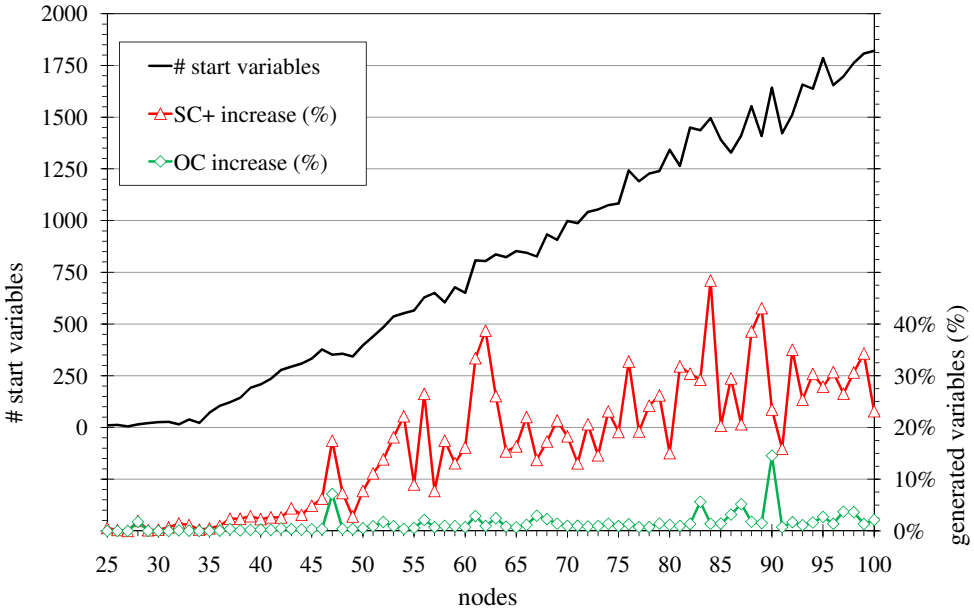


Figure 8.17: The average factor by which the number of variables increases compared to the number of start variables (which is identical for SC+ and OC). The diagram also shows the average number of start variables per graph size.

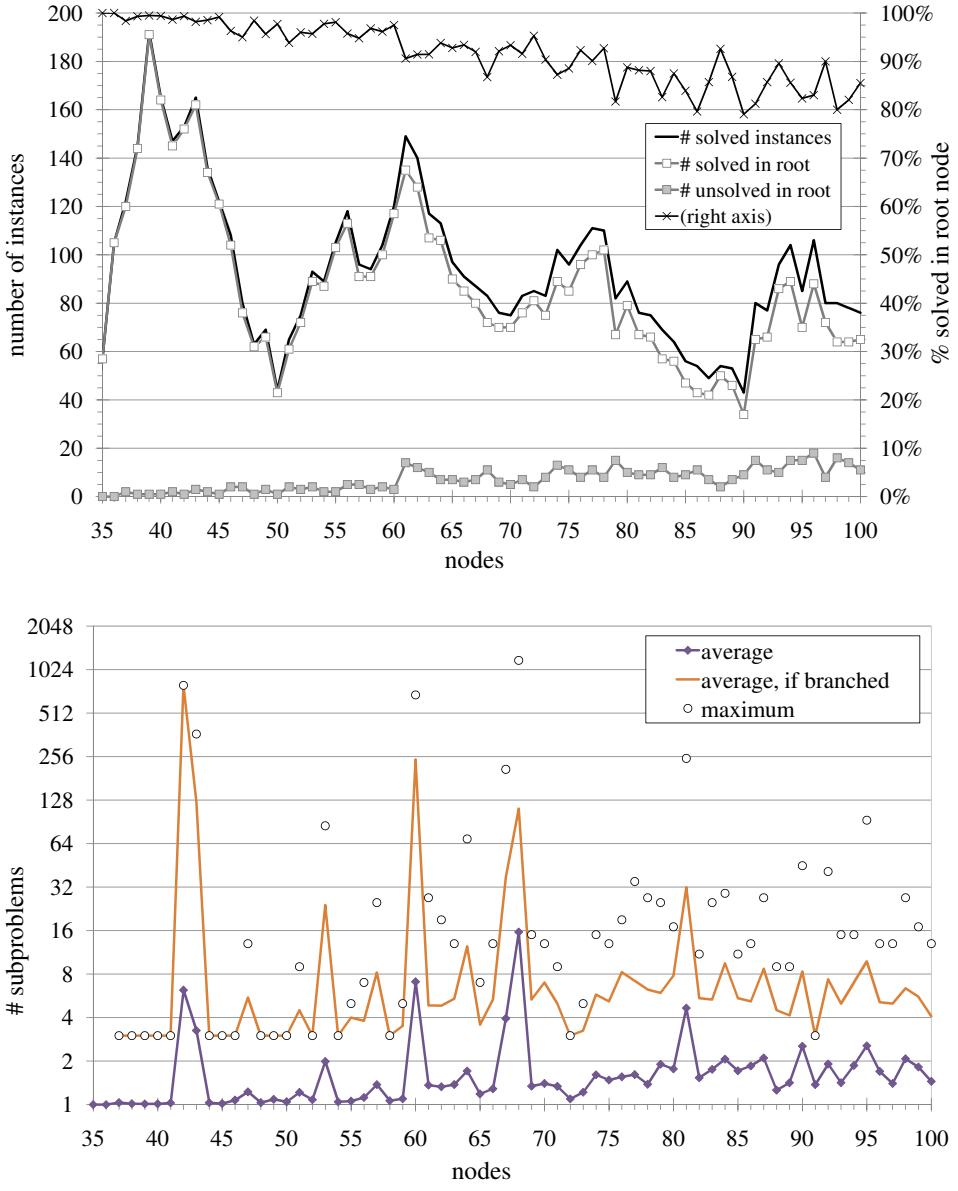


Figure 8.18: Considering the graphs that are solved within 30 minutes: (top) necessity for branching, (bottom) number of subproblems.

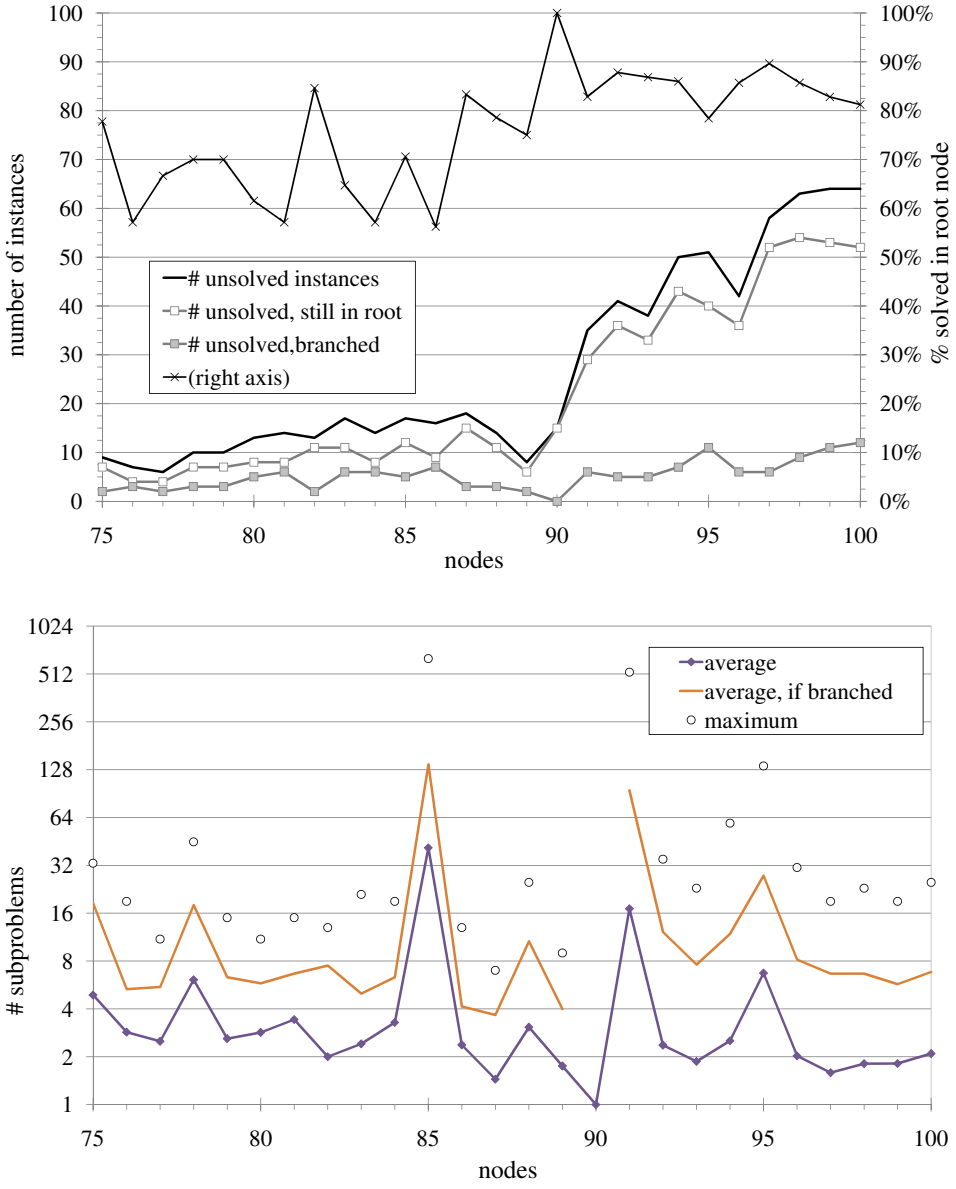


Figure 8.19: Considering the graphs that are unsolved after 30 minutes: (top) necessity for branching, (bottom) number of subproblems.

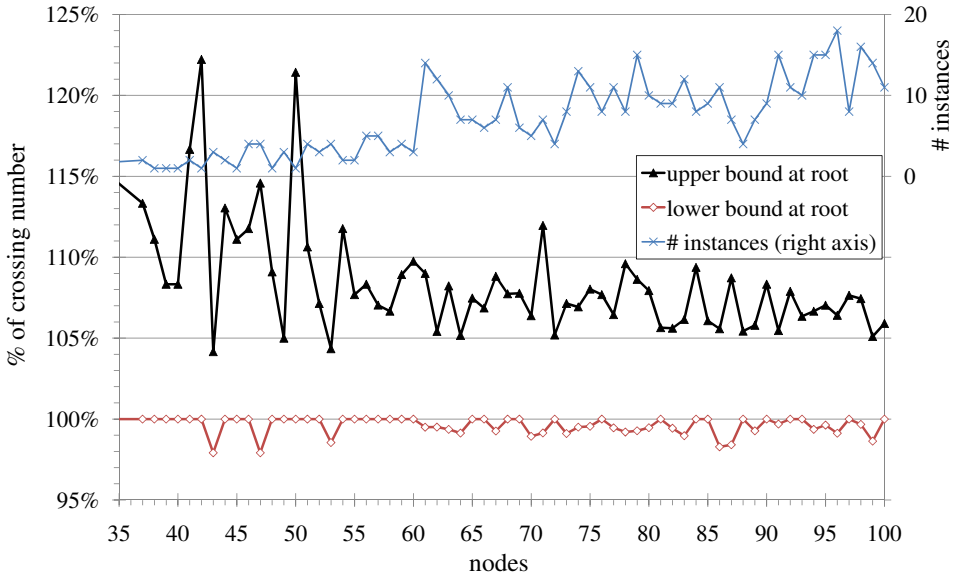


Figure 8.20: Gaps between the root bounds and the crossing number. We only consider solved graphs that require branching.

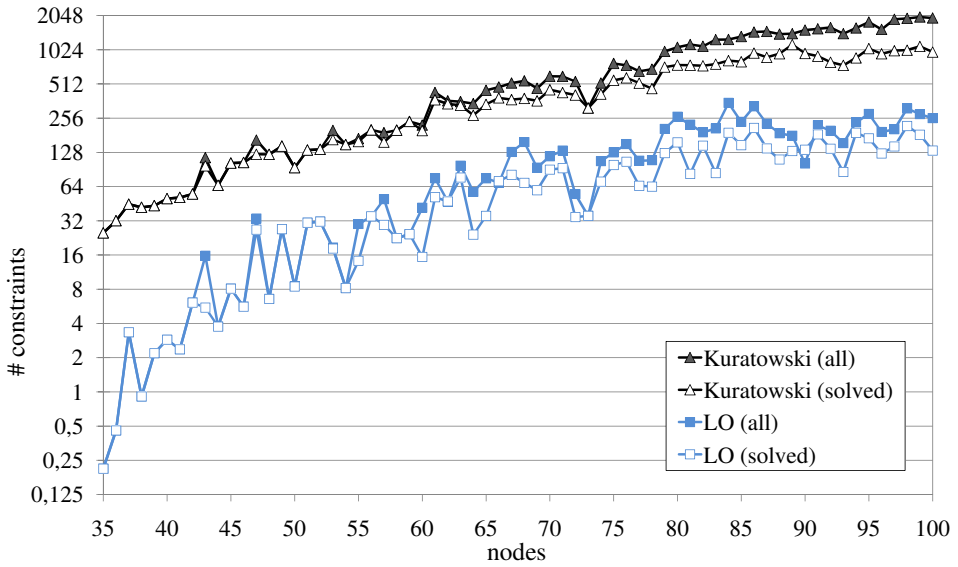


Figure 8.21: Kuratowski- and LO-constraints generated by OC, over all graphs, and over all that are solved to provable optimality.

8.4 Comparison with Heuristic

Having the knowledge of exact values for most Rome graph instances, we are now in the position to evaluate the quality of the currently best heuristic. You may also see [73] for a more thorough description and comparison of the quality of such heuristics, as well as a comparison with the exact crossing numbers computed by the algorithms described herein. While the cited thesis especially discusses the comparison between a long-running version of the planarization heuristic and the exact values, we will focus on more traditional, but weaker, parameters for the heuristic, as would be done in most semi-interactive graph drawing applications. In particular, we will use the parameter settings recommended in [76], augmented with the improved post-processing strategy called *incremental*: After each edge insertion step, we remove and re-insert all other edges again one-by-one. The traditional post-processing strategy would try to re-insert all edges only after all initial insertion steps, cf. [73]. This is the same setting we used for our initial upper bound in SC+ and OC. Hence, we simultaneously analyze how often our algorithm was improving the upper bound or merely had to prove its optimality. More generally, we can also compare the initial upper bound *IUB* to the final upper bound *FUB* obtained after 30 minutes of OC. Recall Figure 8.16 for an overview on the deviation between the final upper and lower bounds.

Figure 8.22 shows the average number of crossings with respect to the number of nodes for both the IUB and the FUB. It turns out that the IUB is very close to the FUB. We know that OC solves almost all instances with up to 60 nodes to optimality, and so we can conclude that the heuristic performs very well on these graphs. The dashed line shows the relative improvement achieved by OC. For graphs over 50 nodes, the heuristic is usually 6–8% away from OC's final upper bound. The small circles in the lower part of the diagram give the maximal absolute deviation between the FUB and the IUB; the largest improvement was 13 crossings achieved for a graph with 91 nodes. We further observe in Figure 8.23 that the heuristic solution is optimal for virtually all instances with up to 3 crossings. The absolute deviation from the upper bound of OC grows linear for graphs with up to 24 crossings by about 1/12 per crossings—the experiments cannot be used to deduce any relationship for larger numbers. It is very interesting to see this observation in the context of the analogous statistic based on experiments with the 2nd generation algorithm SC, as published in [30]: Therein, we suggested the very same slope, although it was only observable for up to 12 crossings as SC's success ratio was not sufficient for higher crossing numbers.

We consider the absolute improvement achieved by OC in more detail. Figure 8.24 shows the deviation between the FUB and the IUB per graph size: The larger the circle is, the more graphs had the difference given on the vertical axis. It is interesting to consider the same statistic with respect to the FUB (see Figure 8.25). Finally, see Figure 8.26: For graphs with crossing

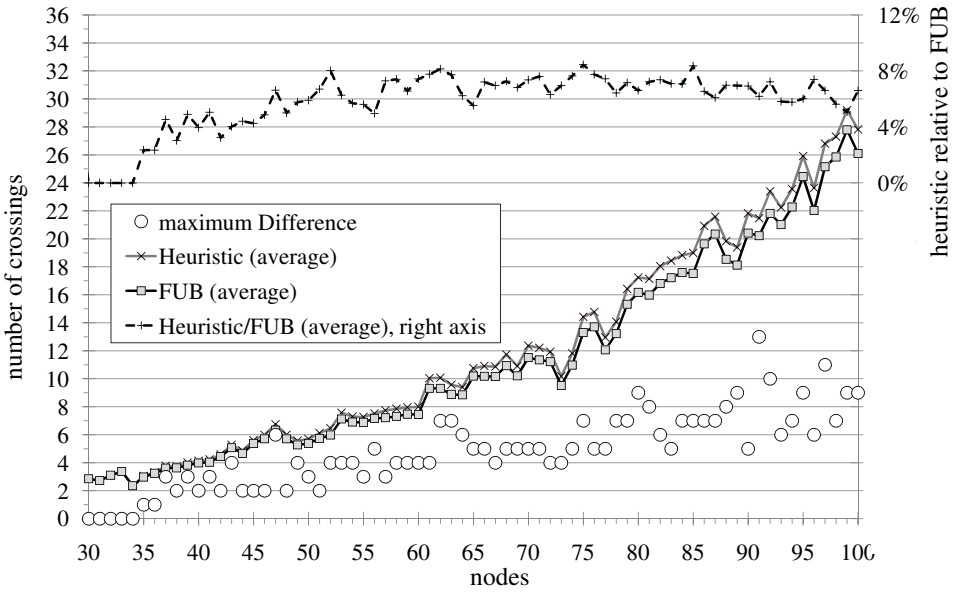


Figure 8.22: Average number of crossings achieved by heuristic and OC.

number at most 3, the heuristic solves nearly all instances to optimality—note that there is a single non-trivial graph G with $\mathrm{cr}(G) = 1$ where the heuristic could not find such a solution. We can again observe that the higher the crossing number, the more often the heuristic was not optimal. The diagram is truncated at FUB= 40. The results for graphs with higher final bound are only few and statistically useless, since OC timed out early.

We conclude that, although there is still room for improvement when the crossing numbers get higher, the currently best known heuristics are very good in practice and solve many instances to their optimal value. This observation if further strengthened in [73], where the solutions of the heuristics over multiple, more time-consuming runs are investigated. Using the solutions of such a heuristic would again speed up our exact algorithm, as we could prune more branch-and-bound nodes early.

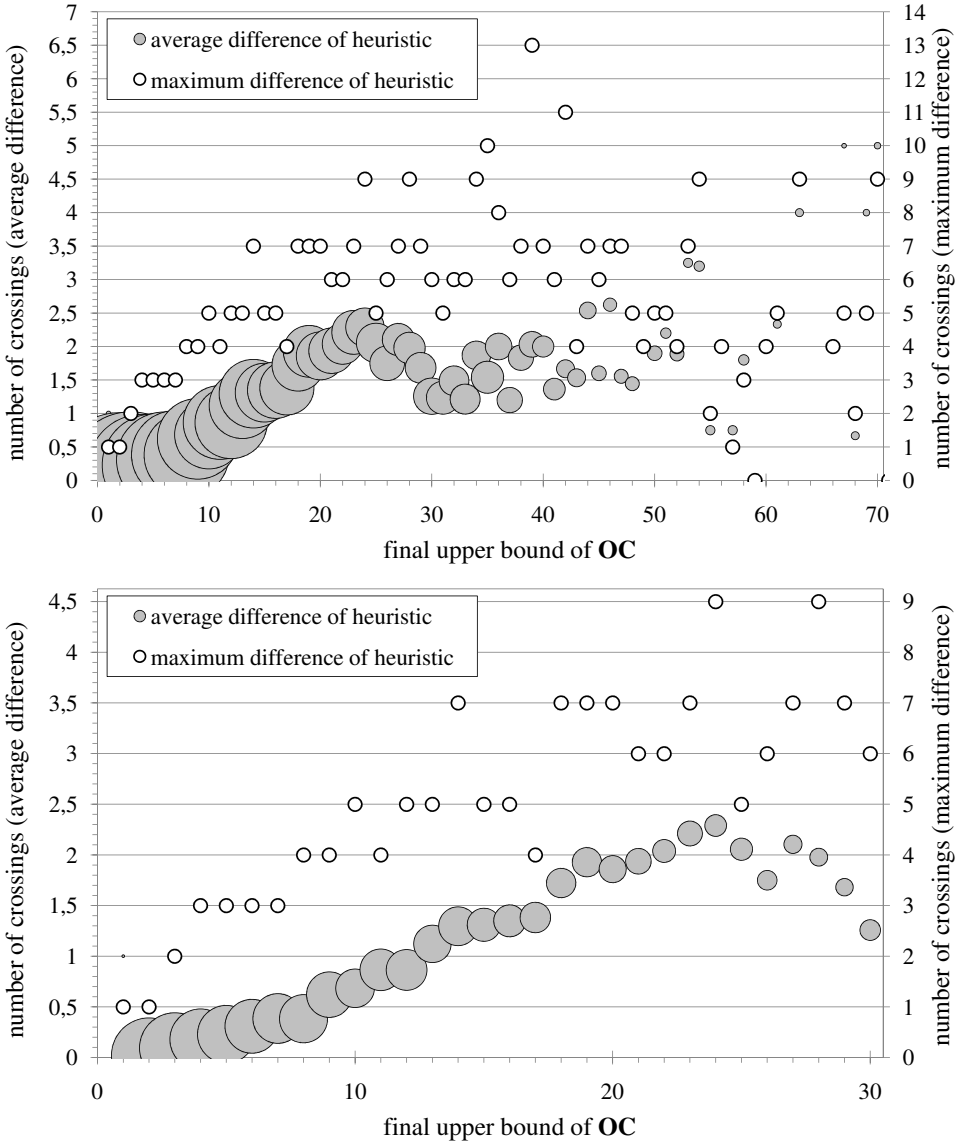


Figure 8.23: Difference in the computed crossing numbers between the IUB and the FUB. (top) overview, (bottom) zoomed.

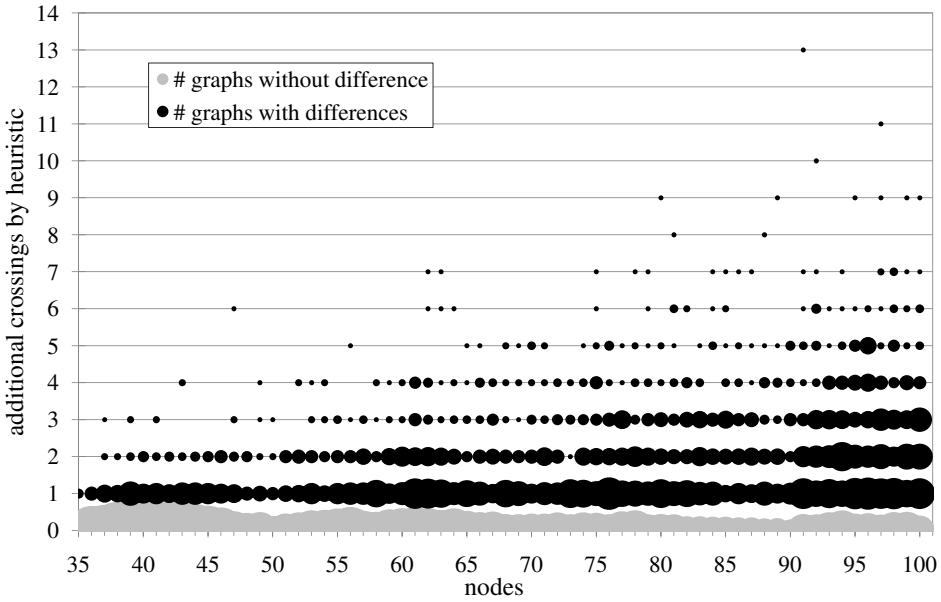


Figure 8.24: Absolute improvement of OC by number of nodes.

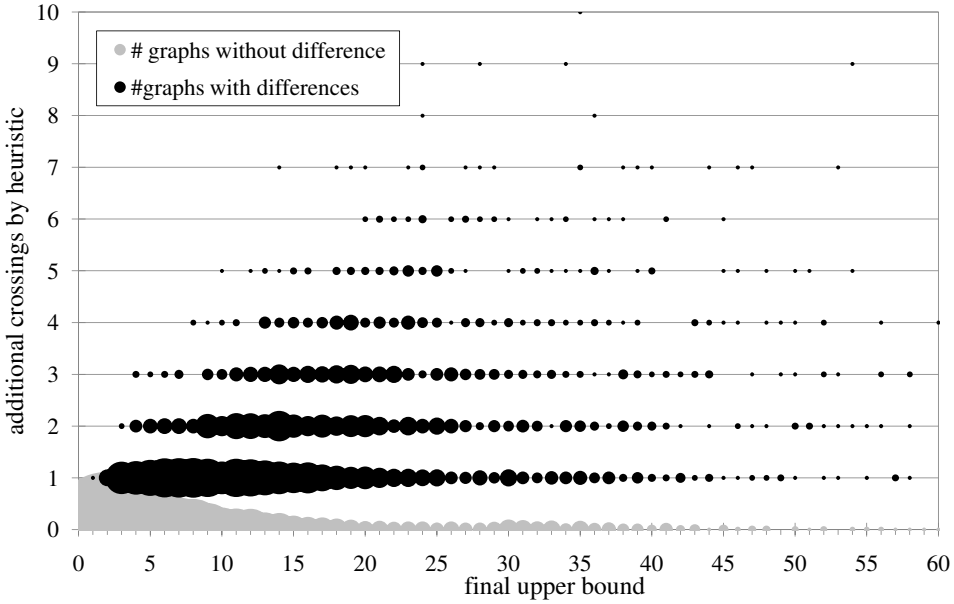


Figure 8.25: Absolute improvement of OC by FUB.

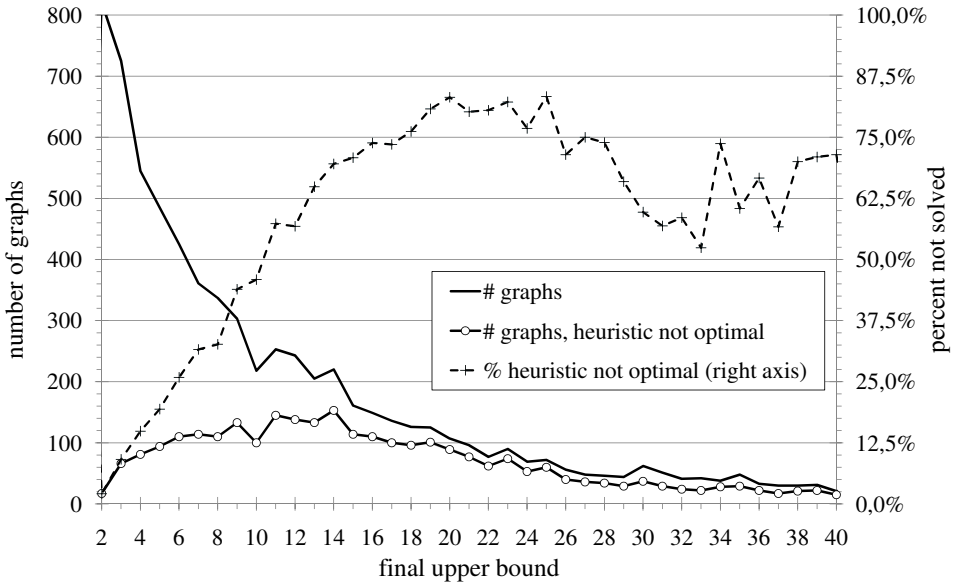


Figure 8.26: Quality of the heuristic solution.

Chapter 9

Experiments: Other Crossing Numbers

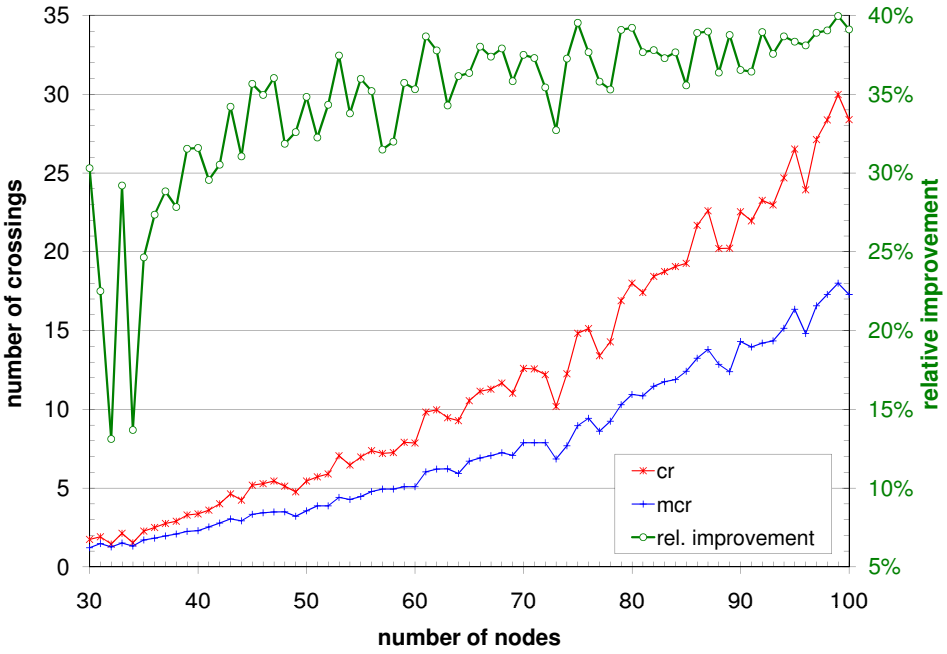
crossing: *(Architecture)*
A drainage structure that passes over, under, or through a location used for the passage of people, livestock, or vehicles.

9.1 Minor and Hypergraph Crossing Number

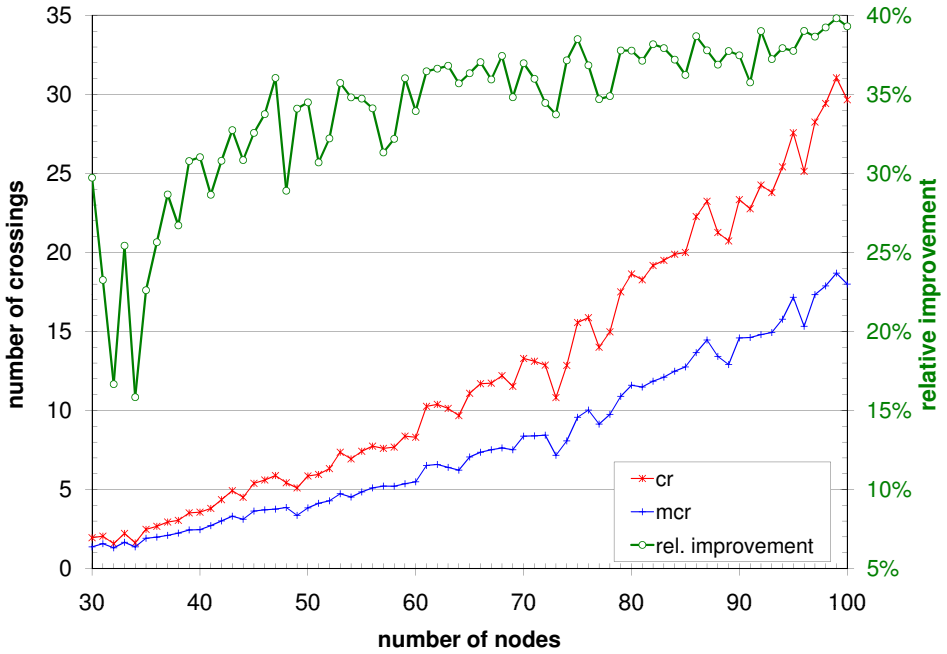
In Section 7.2, we described heuristic approaches to solve the minor and hypergraph crossing numbers. As noted, the sketched exact ILP approach is of no practical relevance. Hence, this section focuses on the former.

Minor crossing number. The first group of experiments deals with the minor crossing number. We use the Rome benchmark set introduced in the previous chapter and restrict ourselves to the non-planar graphs with at least 30 nodes. Our main focus is to investigate how the minor crossing number compares to the traditional crossing number in real-world settings. Figure 9.1(a) shows the average crossing numbers and minor crossing numbers per graph size. We can see that the minor crossing minimization leads to roughly 35% less crossings on average. While this diagram shows the results for variable embeddings, the diagram looks nearly identical when considering random fixed embeddings, although the absolute crossing numbers are of course a bit higher (cf. Figure 9.1(b)).

In both cases, for the large graphs, the realizing graphs have about 10% more nodes than the original graphs, and roughly 8% of the graphs' nodes are substituted by expansion trees. See Figure 9.2 for the absolute values. On a Pentium 4 Windows PC (3.4 GHz) and 2 GB RAM, all graphs can be solved in clearly under a second for the fixed embedding case and under 30



(a) variable embedding



(b) fixed embedding

Figure 9.1: Minor crossing number results for the Rome graphs.

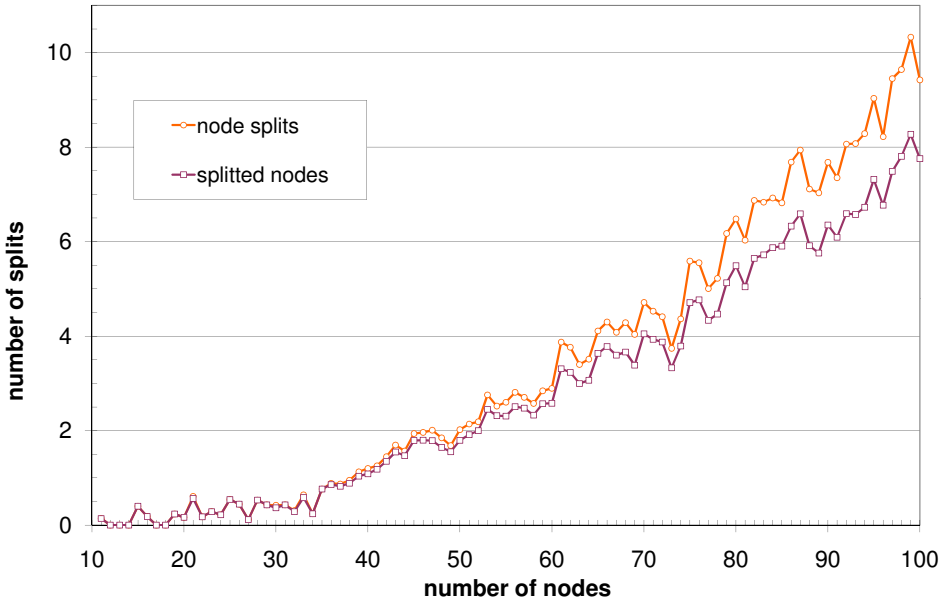


Figure 9.2: Splits in the minor crossing number solutions for the Rome graphs.

seconds for the variable case. For the latter, the 100-node graphs require 5.5 seconds on average.

Hypergraph/Circuit Crossing Number. The second set of experiments deals with the hypergraph crossing numbers and the circuit crossing number. The tests were run on an Intel Xeon E5430 2.67GHz with 2 GB. We choose all hypergraphs from the established synthesized sequential benchmark set SYNTH [22], and the ISCAS'85, ISCAS'89, and ITC'99 benchmark sets of real-world electrical networks. In the latter, we considered all graphs with up to 1000 gates. This leads to hypergraphs with up to 1781 nodes and 2750 edges in their point-based transformations.

Table 9.1 shows the heuristically computed point-based and tree-based hypergraph crossing numbers for these instances (*phcr*, *thcr*), as well as their circuit crossing numbers (*ccr*). Recall that the latter considers the tree-based setting under the restriction that the I/O ports have to satisfy the outer face properties DR1 and DR2 described in Section 7.2.7. All crossing numbers are computed using 10 permutations, i.e., the full edge insertion step is performed 10 times, and the best solution is chosen. The column ccr_1 shows the results for the circuit crossing number after only one permutation.

Table 9.2 gives the required running times of our algorithm for computing *phcr*, *thcr* and ccr_1. We omit the running times for *ccr* as they are simply 10 times higher than ccr_1. We see the clear dependence of the running time on the size of the resulting planarization: The higher the crossing number, the more dummy nodes we have to insert and consider. We observe that

Table 9.1:

				FIX				VAR				[52]	
	ID	Gts	$\lvert E' \rvert$	phcr	thcr	ccr_1	ccr	phcr	thcr	ccr_1	ccr	ccr	red.
Synth	add6	229	531	51	43	94	68	49	37	70	64	112	42.9%
	adr4	147	340	40	29	40	40	33	25	63	41	74	44.6%
	alu1	85	188	10	10	42	42	10	9	36	36	60	40.4%
	alu2	189	448	115	89	181	128	117	80	140	121	243	50.2%
	alu3	218	514	172	109	190	166	160	108	226	183	331	44.7%
	co14	145	334	19	18	36	33	19	17	33	32	52	38.5%
	dk17	168	392	89	59	159	138	92	56	131	116	188	38.3%
	dk27	78	175	12	11	43	40	10	9	43	34	45	24.4%
	dk48	194	452	74	59	204	170	73	56	171	166	290	42.8%
	mish	215	423	0	0	12	11	0	0	11	10	49	79.6%
	radd	121	275	13	12	38	29	12	12	30	24	37	35.1%
	rckl	338	790	99	90	192	186	89	85	164	157		
	rd53	134	320	88	62	85	69	80	54	75	75	126	40.5%
	vg2	185	421	35	27	92	86	30	25	74	74	131	43.5%
	x1dn	186	423	35	29	85	78	24	24	80	74	134	44.8%
	x9dn	203	459	39	33	115	100	35	33	77	77	158	51.3%
	z4	125	291	33	24	48	43	29	24	47	41	66	37.9%
	Z9sym	438	1081	1615	818	1074	917	1563	700	824	763	1802	57.7%
Iscas'85	c17	4	16	0	0	1	1	0	0	1	1		
	c432	153	489	331	175	453	409	303	166	471	368		
	c499	170	578	543	218	708	574	493	209	578	546		
	c880	357	1086	548	369	910	716	521	336	787	701		
	c1196	516	1525	3612	1500	2195	1998	3613	1388	2008	1765		
	c1238	495	1536	5074	1838	2460	2353	4868	1640	2252	2068		
	c1355	514	1578	569	225	756	684	492	214	708	561		
	c1908	855	2353	1800	867	1571	1382	1768	809	1335	1328		
Iscas'89	s27	12	33	0	0	1	1	0	0	1	1		
	s208	102	276	30	22	39	38	31	21	34	34	162	79.0%
	s208a	111	300	35	21	40	31	25	18	29	28		
	s298	127	385	244	87	140	116	226	80	128	116	428	72.9%
	s344	164	448	64	43	109	89	60	37	94	76		
	s349	165	453	71	42	94	80	60	42	83	76		
	s382	173	500	226	82	145	122	201	89	129	110	357	69.2%
	s386	158	511	835	300	449	360	815	278	385	310	904	65.7%
	s400	177	518	238	95	149	140	239	93	138	121	400	69.8%
	s420	210	562	84	60	120	83	79	57	96	84		
	s420a	233	632	91	52	87	83	88	47	78	75		
	s444	196	569	229	90	125	119	224	83	118	105		
	s510	210	640	1291	565	857	764	1244	497	716	678		
	s526	208	674	662	262	327	288	627	233	296	275		
	s526a	209	675	631	252	324	264	631	233	279	272		
	s641	374	932	237	118	394	394	227	120	425	385		
	s713	389	999	226	122	404	374	219	106	384	354		
	s820	275	1037	6021	1075	1566	1378	5491	963	1603	1346		
	s832	273	1047	6262	1161	1673	1527	6015	1058	1360	1360		
	s838	420	1122	244	144	281	269	218	139	277	247		
	s838a	477	1296	217	112	187	170	197	116	172	158		
	s953	401	1173	2364	1317	1936	1728	2212	1259	1671	1555		
	s1196	533	1549	3908	1728	2104	2016	3876	1600	2005	1805		
	s1423	726	1964	620	273	402	386	577	257	381	342		
Itc'99	b01	40	127	57	33	37	31	56	30	32	29		
	b02	25	74	14	8	16	11	11	8	11	9		
	b03	137	407	153	56	80	73	145	56	72	66		
	b04s	570	1670	1595	607	694	645	1504	477	758	612		
	b05s	872	2750	3091	1339	1567	1472	3017	1294	1399	1312		
	b06	46	147	66	32	48	45	62	32	52	41		
	b07s	403	1194	1073	495	564	543	1049	486	484	484		
	b08	150	455	316	168	250	196	295	149	232	181		
	b09	156	449	219	94	101	101	227	89	124	92		
	b10	166	521	438	231	355	299	437	215	313	300		
	b11s	462	1428	3131	1164	1414	1182	2939	1187	1366	1216		
	b13s	309	899	203	111	173	148	168	95	175	137		

				FIX			VAR		
	ID	Gts	$\|E'\|$	phcr	thcr	ccr_1	phcr	thcr	ccr_1
Synth	add6	229	531	0.26	0.24	0.14	16.72	14.21	4.98
	adr4	147	340	0.14	0.12	0.06	5.68	5.05	1.29
	alu1	85	188	0.04	0.03	0.02	0.87	0.62	0.78
	alu2	189	448	0.54	0.5	0.13	46.16	31.9	9.15
	alu3	218	514	0.9	0.7	0.2	79.41	62.89	12.34
	co14	145	334	0.09	0.08	0.05	2.78	2.18	0.91
	dk17	168	392	0.35	0.3	0.12	25.8	22.8	7.7
	dk27	78	175	0.04	0.04	0.02	1.02	0.89	0.38
	dk48	194	452	0.29	0.42	0.21	24.15	21.56	14.43
	mish	215	423	0.0	0.0	0.1	0.0	0.0	0.43
	radd	121	275	0.05	0.04	0.04	0.98	1.13	0.56
	rckl	338	790	0.62	0.6	0.4	51.45	48.2	18.89
	rd53	134	320	0.29	0.24	0.08	29.64	19.28	1.73
	vg2	185	421	0.13	0.14	0.1	6.91	4.66	5.85
	x1dn	186	423	0.12	0.11	0.1	5.12	4.95	6.34
	x9dn	203	459	0.19	0.2	0.15	6.96	6.82	4.34
	z4	125	291	0.1	0.1	0.04	4.88	4.21	0.72
	Z9sym	438	1081	115.7	86.67	7.81	17305.6	8462.18	473.87
Iscas'85	c17	4	16	0.0	0.0	0.0	0.0	0.0	0.0
	c432	153	489	1.9	1.29	0.41	214.5	99.48	23.92
	c499	170	578	9.06	3.32	1.55	1143.6	417.55	145.22
	c880	357	1086	10.73	6.9	3.18	2842.73	936.08	471.94
	c1196	516	1525	1977.37	871.58	75.37	115963	46354	3720.44
	c1238	495	1536	3548.5	1284.16	117.85	161832	79042.9	8968.46
	c1355	514	1578	21.85	6.59	2.99	2272.43	1359.76	240.23
	c1908	855	2353	409.58	77.91	20.16	44384.6	16817.6	1999.33
Iscas'89	s27	12	33	0.0	0.0	0.0	0.0	0.0	0.0
	s208	102	276	0.08	0.09	0.04	5.37	4.13	0.48
	s208a	111	300	0.08	0.07	0.03	4.32	3.14	0.44
	s298	127	385	1.93	0.74	0.13	204.15	55	5.5
	s344	164	448	0.24	0.22	0.09	19.66	13.64	3.27
	s349	165	453	0.28	0.27	0.14	21.54	16.2	4.72
	s382	173	500	0.99	0.66	0.17	114.81	55.88	4.6
	s386	158	511	9.01	7.65	0.51	1503.18	593.6	25.37
	s400	177	518	1.12	0.76	0.18	137.1	49.05	7.18
	s420	210	562	0.37	0.41	0.13	43.77	26.13	5.67
	s420a	233	632	0.38	0.37	0.16	38.31	30.05	5.3
	s444	196	569	1.06	0.78	0.19	134.78	55.39	7.63
	s510	210	640	28.18	16.66	1.4	4199.45	1122.45	164.35
	s526	208	674	7.58	3.93	0.38	1177.34	315.69	26.93
	s526a	209	675	6.37	3.75	0.6	1213.15	437.1	30.38
	s641	374	932	1.64	1.38	0.97	177.96	131.2	65.63
	s713	389	999	2.07	1.48	1.1	226.59	154.51	49.45
	s820	275	1037	3288.78	287.49	27.27	72036.2	14218.4	1129.8
	s832	273	1047	4960.48	354.96	15.9	100122	21556.7	2058.63
	s838	420	1122	3.36	2.24	0.59	492.93	326.83	24.35
	s838a	477	1296	1.83	1.39	0.73	494.33	256.95	19.6
	s953	401	1173	487.33	170.66	14.47	27788.7	16452.2	1918.05
	s1196	533	1549	2747.56	757.75	106.63	159485	61260.1	4333.75
	s1423	726	1964	14.01	7.53	2.23	3104.34	1140.83	90.63
Itc'99	b01	40	127	0.09	0.08	0.01	3.64	2.5	0.38
	b02	25	74	0.01	0.01	0.0	0.24	0.25	0.03
	b03	137	407	0.45	0.3	0.06	37.83	17.1	1.77
	b04s	570	1670	147.14	29.39	3.79	18064.2	3928.51	407
	b05s	872	2750	1379.09	261.75	20.58	138015	59115.6	4218.66
	b06	46	147	0.1	0.06	0.02	4.76	2.36	0.46
	b07s	403	1194	78.78	14.57	1.34	5710.17	1961.42	138.64
	b08	150	455	2.19	1.32	0.3	228.91	112.16	11.86
	b09	156	449	1.6	0.54	0.1	163.69	40.39	2.57
	b10	166	521	5.78	2.73	0.46	699.3	280.95	36.79
	b11s	462	1428	1395.11	197.56	11.88	58743.1	18403.3	2016.01
	b13s	309	899	1.24	0.76	0.3	210.25	110.41	12.84

Table 9.2: Test results for hypergraphs and electrical wiring schemes. Table 9.1 gives the achieved crossing numbers, this table the corresponding running times (in seconds). *Gts* gives the number of gates, E' denotes the edges in $\Lambda(\mathcal{H})$. See the main text for a description of the other columns.

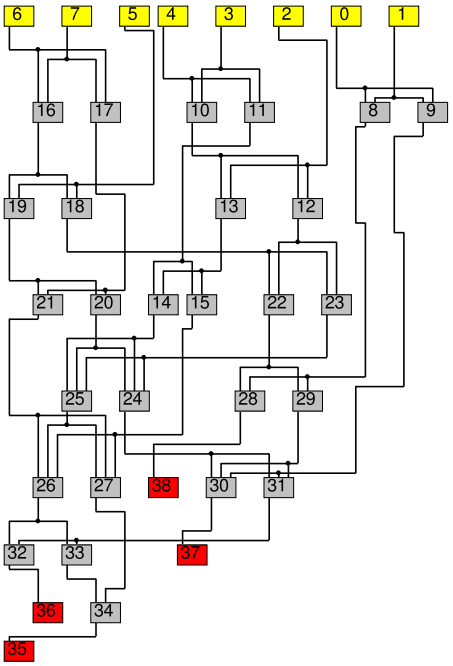
		SYNTH	ISCAS'85	ISCAS'89	ITC'99	all
crossings	VAR:FIX (phcr)	91.8%	94.0%	93.8%	94.1%	93.4%
	VAR:FIX (thcr)	91.8%	93.1%	93.8%	94.0%	93.2%
	VAR:FIX (ccr)	92.0%	91.0%	92.3%	92.4%	91.9%
	thcr:phcr (VAR)	80.1%	46.2%	47.9%	47.8%	55.5%
	thcr:ccr (VAR)	50.9%	55.5%	68.9%	87.5%	65.7%
runtime	VAR/FIX (phcr)	58.4×	102.7×	110.3×	85.9×	89.3×
	VAR/FIX (thcr)	49.7×	109.6×	81.8×	95.6×	84.2×
	VAR/FIX (ccr)	39.7×	75.8×	49.1×	72.3×	59.2×
	phcr/thcr (VAR)	1.2×	2.2×	2.3×	2.5×	2.1×
	ccr/thcr (VAR)	28.7×	2.1×	1.5×	1.1×	8.3×

Table 9.3: Benchmark-wise average percentages and ratios of the results for hypergraph and circuit crossing numbers. *A:B (C)* gives the percentage of A in terms of B, in the setting C. *A/B (C)* gives the ratio between A and B, in the setting C.

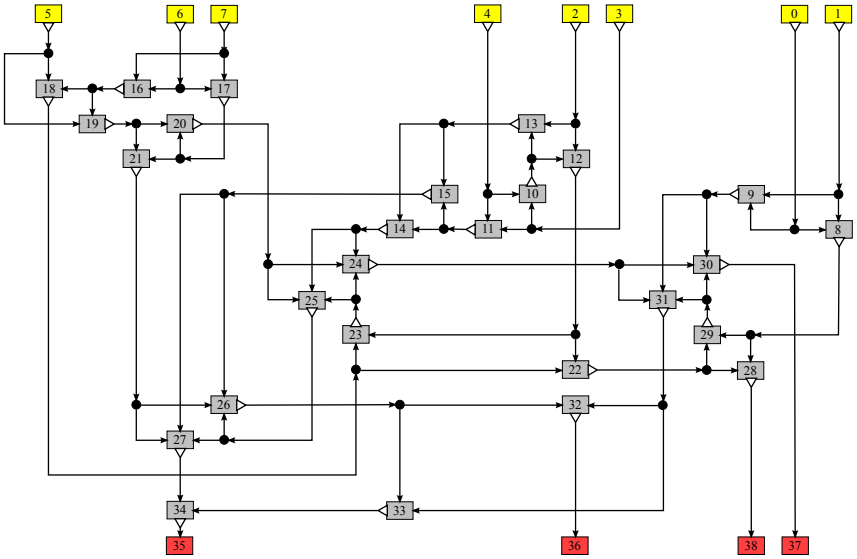
computing *phcr* therefore is more time consuming than the implementation-wise more complex algorithm for *thcr*, which allows smaller crossing numbers. Table 9.3 summarizes some statistics, averaged over the benchmark sets. The column *all* gives the average over the results for the individual benchmark sets.

Obviously, and as known for the traditional crossing number, the algorithm clearly benefits from multiple permutations and from considering all embeddings (VAR) instead of fixing one (FIX) in the edge insertion step. Consider the results of the algorithm variant VAR. Generally, we can clearly see the benefit of considering the tree-based hypergraph crossing number compared to the point-based one: it reduces the crossing number to 55.5% on average. Regarding the circuit crossing number, we observe that the additional restriction requires 52% more crossings than the pure tree-based hypergraph crossing number.

The right-most columns in Table 9.1 give the results presented in [52], which summarized the currently best known solutions, as well as our improvement on these numbers in percent. Our algorithm for *ccr* clearly outperforms the prior algorithms. The best results we obtained for each circuit require 42% less crossings on average for the SYNTH instances, and even 71.3% less crossings for the ISCAS'89 instances. See Figure 9.3 for an example: Figure 9.3(a) is a re-print of an example in [51], which has 30 crossings, instead of 118 resulting from naïvely drawing the graph using the IDs of the nodes for ordering them within layers; note that in 9.3(a) one out-port would require either an additional crossing or a switch between nodes 28 and 29, in order to lie on the outer face. Figure 9.3(b) shows a drawing corresponding to the planarization computed by our algorithm. We require only 18 crossings. Note that in this example, we do not require to split any nodes, as the hypernodes have at most degree 3.



(a) Layout from [51]; nodes recolored and background grid removed: 30 crossings



(b) A drawing realizing our computed planarization: 18 crossings

Figure 9.3: Example graph *rd84* from [51] (not part of the official SYNTH benchmark set).

9.2 Simultaneous Crossing Number

We implemented both heuristic approaches—based on the edge insertion over a fixed embedding and over all embeddings, respectively—as well as the exact algorithm. The main intention of the experiments is to investigate the behavior of the simultaneous crossing minimization concept.

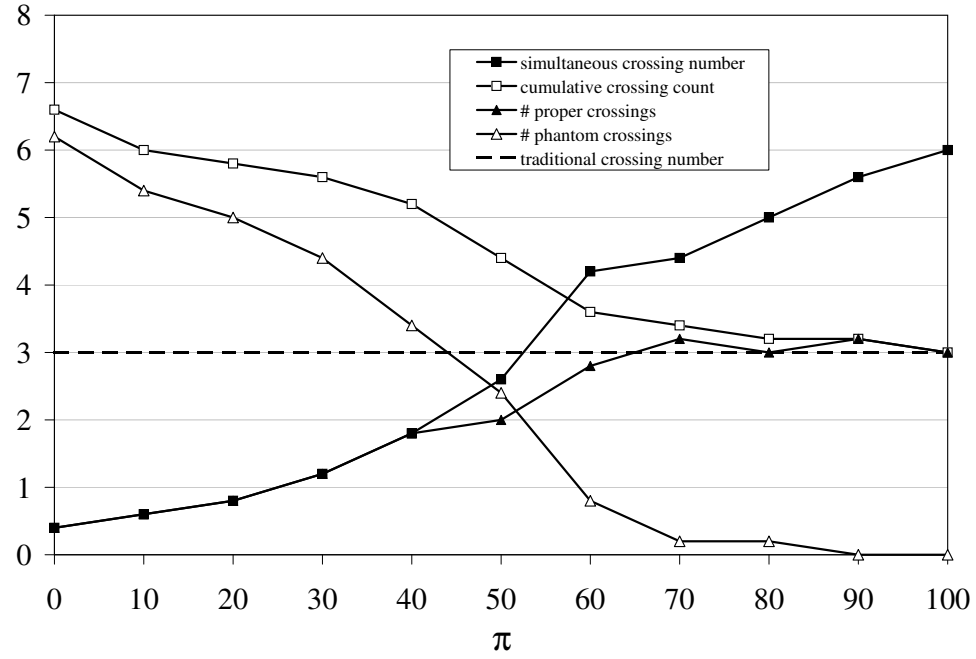
First, we created a set of 10 random graphs with varying size. We then interpreted each graph G of this set as a simultaneous graph of two basic graphs by randomly choosing the basic graphs for each edge of G: We set the probability for every edge to belong to both basic graphs to some percentage π. All edges which do not belong to both basic graphs had an equal chance to belong to either basic graph. We used different values for π: 0, 10, 20, ..., 90, and created a set of 10 simultaneous graphs for each original graph and each value of π. This resulted in 1000 simultaneous graphs.

We applied the heuristic algorithms to this test set and computed the simultaneous crossing number, together with the number of phantom and proper crossings. Assume a simultaneous graph $\mathcal{G}$ generated from G with $\pi = 100$: All edges belong to both basic graphs and thus this instance reflects the traditional crossing minimization problem: There are no phantom crossings, we have $\mathrm{scr}(\mathcal{G}) = 2 \cdot \mathrm{ccc}(\mathcal{G})$, and for optimal solutions we have $\mathrm{ccc}(\mathcal{G}) = \mathrm{cr}(G)$. We performed similar experiments with the exact algorithm. These were run on smaller and less dense graphs, as the algorithm's running time is highly dependent on the crossings number (cf. Chapter 8), in our case on the cumulative crossing count $\mathrm{ccc}(\mathcal{G})$.

Generally, we encountered the same effects throughout all instances and algorithms. Figure 9.4 and Figure 9.5 show two representative examples. The former one was computed via the exact algorithm, the second one via the heuristic which optimizes the edge insertions over all planar embeddings. The heuristic-specific parameters are the ones proposed in [76].

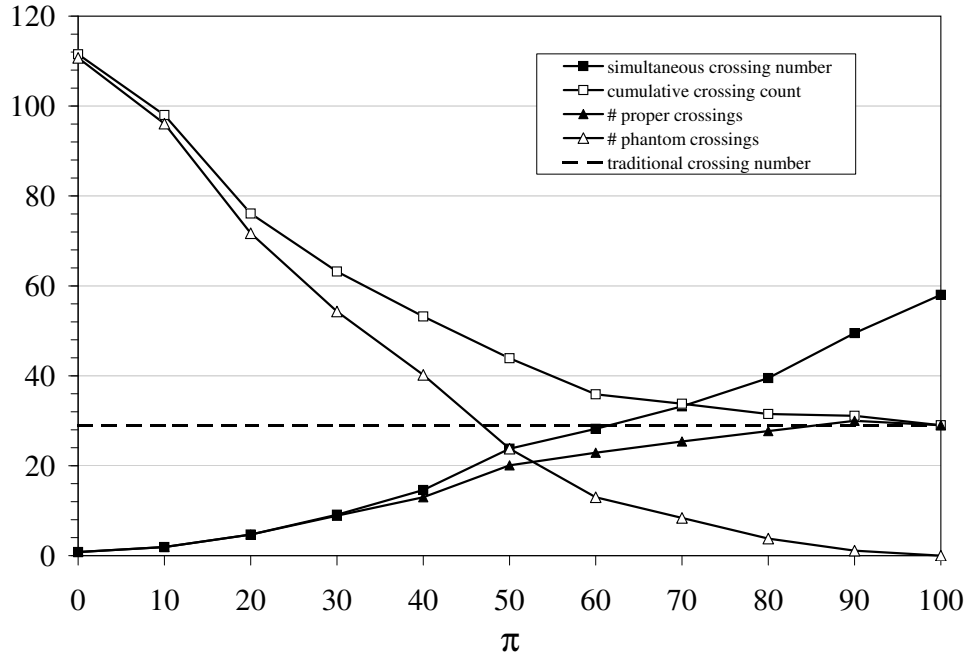
The simultaneous crossing number and the number of proper crossings increase as π gets larger. On the other hand, the number of phantom crossings strongly decreases at the same time, and thus the total number of crossings slightly decreases. Interestingly, while the simultaneous crossing number of course monotonically increases, this does not hold in general for the number of proper crossings due to their differing crossing costs.

Over all values for π, the runtime of the heuristic algorithms is in general roughly four times larger than for the traditional heuristics, most runtime loss is due to the more complex crossing cost calculation for each pair of edges. For the tested ILP instances, the heuristic always finds the optimal solution. Hence the exact branch-and-cut algorithm only has to prove this optimality. The running time is thereby ranging from 10 to $1,000$ seconds for graphs with a crossing count of up to 7.



π	simcr	*prop*	*phan*	ccc
0	0.4	0.4	6.2	6.6
10	0.6	0.6	5.4	6.0
20	0.8	0.8	5.0	5.8
30	1.2	1.2	4.4	5.6
40	1.8	1.8	3.4	5.2
50	2.6	2.0	2.4	4.4
60	4.2	2.8	0.8	3.6
70	4.4	3.2	0.2	3.4
80	5.0	3.0	0.2	3.2
90	5.6	3.2	0.0	3.2
100	6.0	3.0	0.0	3.0

Figure 9.4: (Exact) The figure shows the computed simultaneous crossing number (simcr), the number of proper (*prop*) and phantom (*phan*) crossings and the cumulative crossing count (ccc) for different values of π. The underlying graph G has 10 nodes and 23 edges. The results were computed using the exact algorithm. For each value π, the values were averaged over 5 simultaneous graphs generated from G.



π	simcr	*prop*	*phan*	ccc
0	0.8	0.8	110.7	111.5
10	1.9	1.9	96.1	98.0
20	4.7	4.7	71.7	76.4
30	9.1	8.9	54.3	63.2
40	14.6	13.0	40.2	53.2
50	23.8	20.1	23.8	43.9
60	28.2	22.9	13.0	35.9
70	33.2	25.4	8.4	33.8
80	39.5	27.7	3.8	31.5
90	49.5	30.0	1.1	31.1
100	58.0	29.0	0.0	29.0

Figure 9.5: (Heuristic) The figure shows the computed simultaneous crossing number (simcr), the number of proper (*prop*) and phantom (*phan*) crossings and the cumulative crossing count (ccc) for different values of π. The underlying graph G has 33 nodes and 70 edges. The results were computed heuristically. For each value π, the values were averaged over 10 simultaneous graphs generated from G.

Chapter 10

Solving Special Graph Classes

crossing: *(Travel)*
A voyage across a body of water (usually across the Atlantic Ocean).

This section does not so much reflect finished research, but is more meant as an outlook on a research direction, enabled by the results described prior in this thesis.

The SECM and OECM formulations were developed with "real-world" graphs in mind, i.e., general, medium-sized, comparably sparse graphs. In this section, we will discuss how we can tune our 0/1-ILPs to solve special graph classes which usually have larger crossing numbers. Interestingly, these graph classes can all be considered structurally "simple" in the sense that they are highly symmetric and may have properties like low degree, thickness, genus, etc. Yet, and despite the fact that there has been dedicated graph theoretical research for all the considered graph classes, they turn out to be hard w.r.t. the crossing number problem. In particular, we often have construction schemata which are only conjectured to give optimal solutions.

We will showcase some useful properties and observations for special graph classes which can be turned into additional constraints. Using such strengthenings, we can hope to solve graph instances to provable optimality, for which current purely theoretical research has no proof yet. Although the current results are not necessarily ground breaking yet, they show that we can improve a lot on the standard ILP formulation when including knowledge of the specific graph type.

The aim of this approach is many-fold: We can prove that conjectured crossing numbers hold for certain graphs; we can solve base cases which could be used to develop construction schemata, to extract conjectures, or even as a foundation for theoretical inductive proofs of larger graphs; we can potentially disprove conjectures or falsify non-rigorous proofs for specific graph classes. We also hope that by showing how to integrate certain types of crossing

number observations (e.g. required crossings for substructures, symmetries, etc.) we provide an incentive to do research regarding such results. They can help to speed up our ILPs significantly, which in turn helps to solve theoretically hard crossing number problems.

10.1 General Concepts

Usually, we will start with an upper bound for which we hope that it constitutes an optimal solution, and tune the ILP to prove that no better solution exists. The constraints described below can either be constructed in the beginning—adding them to a pool from which automatic separation is possible—or we may generate them in a dedicated separation routine by manually enumerating and testing them on the fly. Generally, we can classify the additional constraints as follows:

Substructure constraints. The often most useful category of constraints is based on specifically structured subgraphs or subdivisions. If we can, e.g., detect some K_6 subgraph in the given graph, we can construct a corresponding Kuratowski constraint and require $\mathrm{cr}(K_6) = 4$ crossings. This is beneficial compared to enumerating multiple K_5 constraints on this structure.

The main purpose of these constraints is to raise the lower bound of the relaxations in order to prune the problem early due to bound clashes with the supposedly optimal upper bound.

Symmetry constraints. At certain problem sizes, pure cutting will not be sufficient for proofs; branching will occur. The main purpose of symmetry(-breaking) constraints is to avoid enumerating distinct but isomorphic subproblems.

Knowledge constraints. The most general class of constraints that has the potential for strong improvements in practice is also the one requiring the most graph theoretical insight: In certain graphs we may know properties of the optimal drawings (e.g. that it is sufficient to investigate solutions with at most one crossing per edge). Such findings can be turned into constraints of high practical value.

In the following, we will discuss some constraints of these types for specific graph classes.

10.2 Special Classes

10.2.1 Complete Graphs

Bounds and Kleitman's argument. The best-known bound regarding complete graphs is Guy's crossing number conjecture [80]:

$$\mathrm{cr}(K_n) \stackrel{?}{=} Z(n) := \frac{1}{4}\left\lfloor\frac{n}{2}\right\rfloor\left\lfloor\frac{n-1}{2}\right\rfloor\left\lfloor\frac{n-2}{2}\right\rfloor\left\lfloor\frac{n-3}{2}\right\rfloor,$$

which can be written as

$$Z(n) = \frac{1}{64}(n-1)^2(n-3)^2$$

for odd n. Based on the combinatorial bound[1]

$$\mathrm{cr}(K_n) \geq \frac{\binom{n}{p}\mathrm{cr}(K_p)}{\binom{n-4}{p-4}}$$

we can observe that the conjecture holds for $\mathrm{cr}(K_{2k})$ if it is true for $\mathrm{cr}(K_{2k-1})$, i.e., proofs are only interesting for K_n with odd n.

Guy presented a drawing scheme for arbitrary K_n realizing the crossing number conjectured above, hence we have at least an upper bound. He proved the conjectured equality for $n \leq 10$, and Pan and Richter [123] later extended the proof to $n \leq 12$. We know from [99] that

$$0.8594 \cdot Z(n) \leq \mathrm{cr}(Kn).$$

Recall Kleitman's argument that the parity of $\mathrm{cr}(K_n)$, n odd, has to be the same as of $Z(n)$. When proving that the conjecture holds for some K_n and K_{n+1} (n odd), we only have to show that $\mathrm{cr}(K_n) > Z(n) - 2$.

Kuratowski Constraints. Clearly, each K_n contains n K_{n-1} subgraphs, and generally $\binom{n}{m}$ K_m subgraphs. When computing K_n for the smallest unknown n, we can therefore consider all K_m constraints ($5 \leq m < n$, Section 5.4.3) for which we know that they are facet-defining in $\mathcal{K}_n$. Analogously, we have multiple complete bipartite subgraphs $K_{m,m'}$, with $m + m' \leq n$, in K_n for which we can construct Kuratowski constraints if $\mathrm{cr}(K_{m,m'})$ is known, cf. below.

Apart from subgraphs, we can also generate all $n\binom{n-1}{2}$ K_{n-1} subdivisions in K_n: We select any node v that shall not be a Kuratowski node; the other nodes induce a K_{n-1} subgraph. We then select one edge e in this subgraph and replace it by the path of length 2 over v. For this substructure we can force at least $\mathrm{cr}(K_{n-1})$ crossings.

[1] This arises from removing four vertices from the drawing and counting the crossings over all such possible modifications.

While we can store the subgraph constraints in a pool, we check the subdivision constraints on the fly via a separation routine due to their high number. We only add the constraints with highest violation.

Theoretically, we could also separate any K_m, $m < n$, subdivision but their enumeration, even if only done for promising edges, takes too long for practical purposes.

Node/Kuratowski Symmetry Constraints. Assume we are given any optimal drawing of some K_n with node labels $v_1, v_2, \ldots, v_n$. We can arbitrarily permute these labels and still obtain an optimal drawing. Hence we can establish a canonical ordering of the vertices based on the *responsibility* of a node v, i.e., the number of crossings on edges incident to v. We use the shorthand $X(v) = \sum_{\{e,f\} \in \mathcal{CP}: v \in e} x_{e,f}$ to denote this concept. Using any fixed labeling of the nodes we can require:

$$X(v_1) \geq X(v_2) \geq \ldots \geq X(v_n) \tag{10.1}$$

This will not influence the value of the optimal solutions but is only used for breaking the symmetry in the solution space. Its advantage only comes into play when branching.

An analogous idea is to establish a canonical ordering of the crossings on the K_{n-1} subgraphs. We can observe that this is dual to the node symmetry constraints above: Consider any optimal solution. If $G - v$ is the K_{n-1} subgraph with minimal number of crossings, we can deduce that v has the highest responsibility among all nodes, as each edge is either contained in $G - v$ or incident to v. Therefore (10.1) is also a canonical ordering with respect to the K_{n-1} subgraphs.

Edge Symmetry Constraints. Alternatively to the node symmetry constraints above, we can establish *edge symmetry constraints*. Note that they are not compatible, i.e., we cannot use both constraint classes simultaneously. Consider any optimal drawing of some K_n and fix a vertex v_1. Using the relabeling argument of above, we can establish a canonical ordering with respect to its incident edges. We use the shorthand $X(e) = \sum_{f: \{e,f\} \in \mathcal{CP}} x_{e,f}$ for the number of crossings over the edge e:

$$X(\{v_1, v_2\}) \geq X(\{v_1, v_3\}) \geq \ldots \geq X(\{v_1, v_n\}) \tag{10.2}$$

Furthermore, we can require that v_1 is the node with the highest responsibility:

$$X(v_1) \geq X(v_i) \quad \forall 2 \leq i \leq n. \tag{10.3}$$

Experiments show that this class of constraints is more efficient than the node symmetry constraint above.

Subgraph Knowledge Constraints. Sometimes we have certain knowledge about the crossing configuration of certain substructures in the context of the whole graph. Below are such results obtained and used in [123], as sub-steps when proving $\mathrm{cr}(K_{11}) = 100$ analytically. It is trivial to formulate such results as linear constraints; one has only to be careful when trying to use them in conjunction with symmetry breaking constraints like the ones above.

Theorem 10.1 ([123]). *1. For $n \leq 8$, every optimal drawing of K_n contains an optimal drawing of K_{n-1}.*

2. A good optimal drawing of K_9 contains a good drawing of K_8 with at most 20 crossings. Any good drawing of K_8 with at most 20 crossings contains an optimal drawing of K_7.

3. A good drawing of K_{11} with fewer than 100 crossings contains a good drawing of K_{10} with at most 62 crossings. Any good drawing of K_{10} with at most 62 crossings contains an optimal drawing of K_9.

Note on Complete Bipartite Graphs

After preliminary experiments, we did not try to tackle the crossing number problem for complete bipartite graphs: Similar to complete graphs, there is a drawing construction by Zarankiewicz [147] (predating Guy's conjecture) requiring

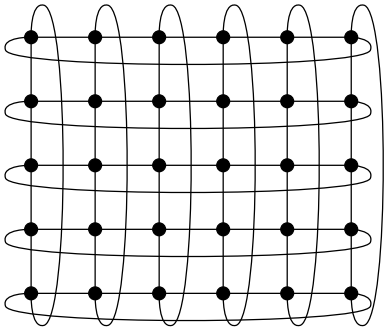
$$Z(n,m) := \left\lfloor \frac{n}{2} \right\rfloor \left\lfloor \frac{n-1}{2} \right\rfloor \left\lfloor \frac{m}{2} \right\rfloor \left\lfloor \frac{m-1}{2} \right\rfloor$$

crossings; he gave a proof for $\mathrm{cr}(K_{n,m}) = Z(n,m)$, which was later shown to be flawed by Guy [79]. Anyhow, we know that the conjecture holds for $K_{2k,m}$ if it holds for $K_{2k-1,m}$, i.e., proving the conjecture is interesting only for complete bipartite graphs with odd partition set cardinalities. Furthermore, we know that $\mathrm{cr}(K_{n,m}) \geq 0.8 \cdot Z(n,m)$ using straight-forward combinatorial arguments; this was later improved to a factor of roughly 0.8001 by Nahas [116]. Kleitman proved the conjecture rigorously for $K_{n,m}$ with $n \leq 6$. For $n = 7$, the conjecture is known to be true for $m \leq 10$.

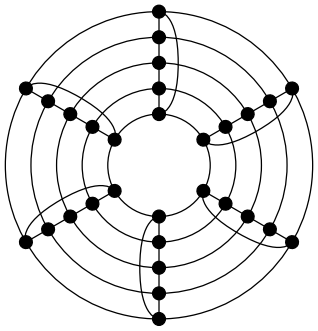
Hence the first interesting complete bipartite graphs in terms of proving Zarankiewicz's conjecture would be $K_{7,11}$ and $K_{9,9}$ with supposedly 225 and 256 crossings, respectively. This seems to be beyond the reach of our ILP approach.

10.2.2 Toroidal Grids

Definition and Bounds. A toroidal grid $T_{n,m}$ is a 4-regular (rectangular) $n \times m$ grid graph where the opposing sides are joined with each other; cf. Figure 10.1(a). The graph, though non-planar, can be embedded on the



(a) Drawn as a rectangular grid with joined sides.



(b) Drawn with the supposedly optimal drawing paradigm.

Figure 10.1: Toroidal grid $T_{6,5}$.

torus; cf. Section 4.3.4. The graph therefore is "simple" in the sense that it has genus 1 and thickness 2, independent of its specific size.

Formally, $T_{n,m}$ has the vertices $v_{i,j}$ for $1 \leq i \leq n$ and $1 \leq j \leq m$. Each vertex $v_{i,j}$ is incident to the vertices $v_{i\pm1 (\text{mod } n), j\pm1 (\text{mod } m)}$.

There is a drawing paradigm which requires $n(m-2)$ crossings (assuming $n \geq m$), cf. Figure 10.1(b). Despite the fact that this drawing and the counting of the arising crossings are very simple, the optimality of these drawings can only be conjectured; see [87, 121] for details. The smallest toroidal grid with unknown crossing number is $T_{8,8}$ with supposedly 48 crossings.

Subgrid constraints. Consider the graph $T_{n,m}$ with $n \geq m$. By ignoring a column or a rows arising in Figure 10.1(a), we obtain a subdivision of a $T_{n-1,m}$ or $T_{n,m-1}$, respectively. Analogous to Kuratowski constraints we can specify constraints on these substructures requiring $(n-1)(m-2)$ or $n(m-3)$ crossings, respectively. By ignoring multiple rows and/or column, we can generate further—smaller—toroidal grid subdivisions and corresponding constraints.

Kuratowski constraints. When trying to solve $T_{8,8}$, we can enumerate multiple $K_{4,4}$ subdivisions; these require at least 4 crossings. Clearly, this is the most complex complete bipartite graph that any toroidal grid can contain as a subdivision. Figure 10.2 shows such a subdivision within a $T_{8,8}$, found by Petr Hliněný.

Symmetry constraints. Similar to the complete graphs, we can pick any vertex, say $v_{i,i}$, in $T_{n,m}$ and require it to have the highest responsibility among all nodes. Note that we cannot order all nodes according to their responsibility. Furthermore, assume w.l.o.g. that $0 < i < \min(n, m)$. We can "orient"

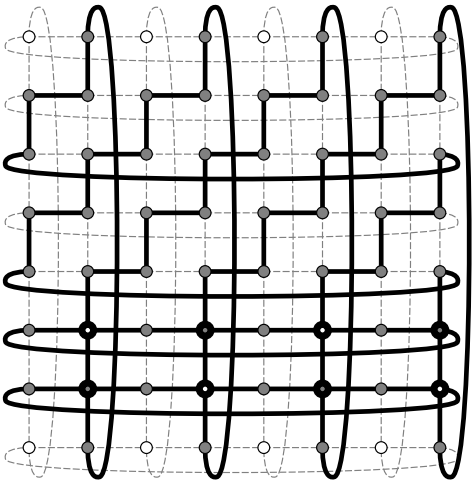


Figure 10.2: A $K_{4,4}$ subdivision in a $T_{8,8}$.

our drawing by requiring an order of the number of crossings w.r.t. the vertical and horizontal neighbors of $v_{i,i}$:

$$X(\{v_{i,i}, v_{i-1,i}\}) \geq X(\{v_{i,i}, v_{i+1,i}\}), \tag{10.4}$$
$$X(\{v_{i,i}, v_{i,i-1}\}) \geq X(\{v_{i,i}, v_{i,i+1}\}). \tag{10.5}$$

If $n = m$ we can furthermore require:

$$X(\{v_{i,i}, v_{i-1,i}\}) \geq X(\{v_{i,i}, v_{i,i-1}\}). \tag{10.6}$$

10.2.3 Generalized Petersen Graphs

Definition and Bounds. A generalized Petersen graph $P_{n,m}$ is a cubic graph (each vertex has degree 3) and is constructed a follows: We start with the *main cycle*, i.e., a cycle on nodes $v_1, \ldots, v_n$. For each vertex v_i we introduce a node v'_i incident to v_i. Then, for each $1 \leq i \leq n$ we connect v'_i with $v'_{i+m(\mathrm{mod}\ n)}$. These latter edges form one or more *secondary cycles*. The eponymous Petersen graph itself is $P_{5,2}$, cf. Figure 10.3.

This graph class has been studied with respect to many graph measures; it was considered in the context of crossing numbers in multiple papers [56, 57, 110, 127, 129, 131]. The known results only regard general bounds or special cases. One of the most outstanding results was to show the crossing number for $P_{n,3}$ for general n [127]. A main complexity when considering generalized Petersen graphs is that there are no simple closed formulae like for the special graph classes above, and there are no established conjectures for most parameter settings.

In our context, there is few room for subgraph constraints similar to the classes investigated before: Since the generalized Petersen graphs are cubic,

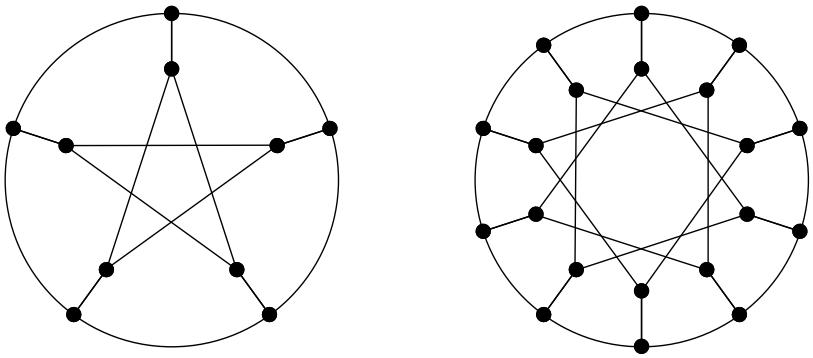


Figure 10.3: Petersen graph $P_{5,2}$, and generalized Petersen graph $P_{10,3}$.

they can never contain a subdivision of a non-planar complete graph or a complete bipartite graph that is more complex than a $K_{3,3}$. In general, it is also not possible to find subgraphs or subdivisions which are themselves generalized Petersen graphs.

Node Symmetry constraints. When considering an optimal drawing of $P_{n,m}$, we can label an arbitrary vertex of the main cycle as v_1. There are two possible labeling orientations of this cycle. Hence, we can select any valid labeling and break the symmetry (a) by requiring that v_1 is the vertex with highest responsibility on that cycle

$$X(v_1) \geq X(v_i) \quad \forall i \neq 1, \tag{10.7}$$

and (b) by choosing an orientation on the main cycle:

$$X(\{v_1, v_n\}) \geq X(\{v_1, v_2\}). \tag{10.8}$$

10.3 Current Results

Complete Graphs. Our current results with the above approach for complete graphs are double-edged: On the one hand we clearly see the benefits of the additional efforts, on the other hand we are not yet able to solve the currently smallest unproven complete graph K_{13}. Experimental runs with this particular graph do not finish within months of computation due to the large number of required branchings. Our unaugmented OECM formulation can only solve graphs up to $\mathrm{cr}(K_7) = 9$ without the special constraints. This graph is trivially solvable using K_6 constraints. Considering the augmented algorithm, we can solve $\mathrm{cr}(K_9) = 36$ within about 15 minutes.

Finally, we were also able to prove $\mathrm{cr}(K_{11}) = 100$, only months after it was proven formally by Richter and Pan in 2007 [123]. Hence our computation—requiring roughly two weeks on a single CPU—verifies their highly complex

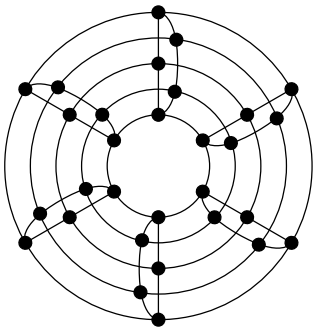


Figure 10.4: Toroidal grid $T_{n,m}$ with $n \geq m$ and m odd: one crossing per edge.

proof. Note that we have to switch to the 64bit mode the aforementioned AMD Opteron offers: Our algorithm requires more than the 2 GB of memory which is the limit for 32bit applications. The machine's physical 32 GB RAM are sufficient.

Toroidal Grid. The toroidal grids also constitute hard problems not only for the theoreticians but also for the ILP approach. This somehow comes as a surprise, as their crossing numbers only increase moderately with their size. We can compute $\mathrm{cr}(T_{7,7}) = 35$ rather easily, but already for the known $\mathrm{cr}(T_{8,7}) = 40$ our algorithm starts to branch heavily such that it cannot prove the bound within reasonable time. In order to tackle the unproven $\mathrm{cr}(T_{8,8}) \stackrel{?}{=} 48$, we seem to require some more graph theoretic insight in the graph structure, and to identify some additional strengthening constraints. An interesting aspect would be the following conjecture:

Conjecture 10.2. The optimal drawing of a toroidal grid $T_{n,m}$, with $n \geq m$ and m odd, requires at most one crossing per edge.

Rationale. When considering the supposedly optimal drawing scheme, we can redraw the cycles of length m such that each edge is crossed at most once, cf. Figure 10.4. □

Hence, if the conjectured crossing number is correct, this conjecture would be a corollary. On the other hand, if we could prove this conjecture, we could restrict our ILP to the simple crossing number, which is practically much simpler, and try to prove $T_{9,9}$.

Generalized Petersen Graphs. We know from [127] that $\mathrm{cr}\,(P_{n,3})$ does not offer a completely closed formula, but

Theorem 10.3 ([127])**.** *The crossing number of the generalized Petersen graph $P_{3k+h,3}$ is $k+h$ if $h \in \{0,2\}$ and $k+3$ if $h = 1$, for each $k \geq 3$, with the single exception of $P_{9,3}$, whose crossing number is 2.*

h	k										*proof*
	1	2	3	4	5	6	7	8	9	10	
0	0	3	7	10	14	16	18	20	22	24	10
1	0	4	8	10	12	14	16	18	20	22	37
2	3	5	10	12	14	16	18	20	22	24	11
3	1	4	8	10	12	14	16	18	20	22	43

Table 10.1: Computed crossing numbers for some generalized Petersen graphs $P_{4k+h+1,4}$. The column *proof* gives the largest k for which we proved our conjectured crossing number; we stopped the computation for a specific h when the last proof required more than 10,000 seconds; the largest graph proved this way is $P_{176,4}$ with 88 crossings.

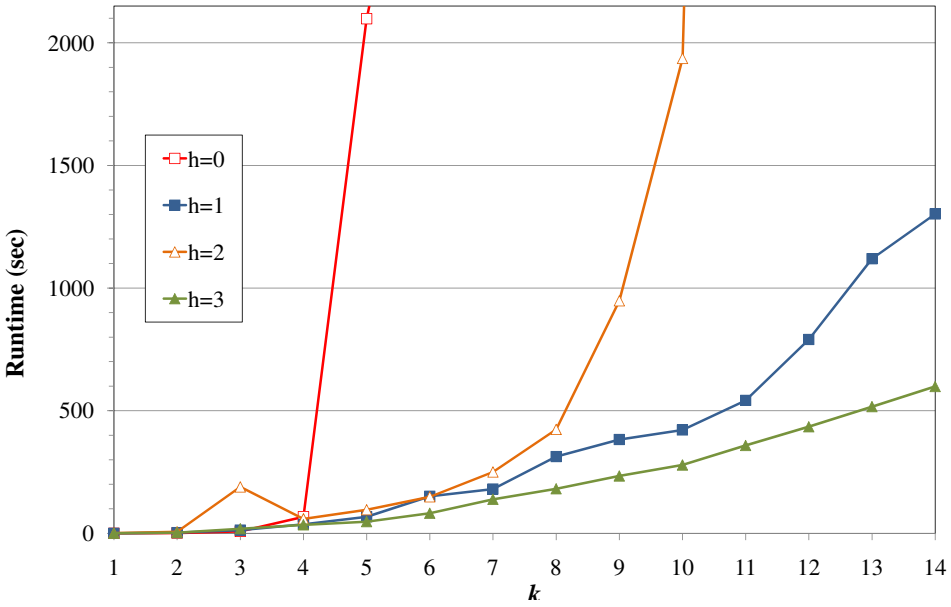


Figure 10.5: The runtime performance for $P_{4k+h+1,4}$, depending on h and k.

In our experiments, we tackled the generalized Petersen graphs $\mathrm{cr}(P_{n,4})$, for which currently not even conjectures exist. There is a paper proving the crossing number of the single graph $P_{4,10}$ [131]. By our computation, cf. Table 10.1, we can prove all such graphs with $n \leq 44$ and multiple beyond. We can see that for small n the crossing number behaves seemingly chaotic, constituting exceptional cases like $P_{9,3}$ for $P_{n,3}$. Only for $n \geq 14$, a pattern seems to emerge. We therefore conjecture:

Conjecture 10.4. The crossing number of the generalized Petersen graph $P_{4k+h+1,4}$ is $2k+4$ if $h \in \{0,2\}$ and $2k+2$ if $h \in \{1,3\}$, for each $k \geq 3$, with the single exceptions of $P_{13,4}$ and $P_{17,4}$, whose crossing numbers are 7 and 10, respectively.

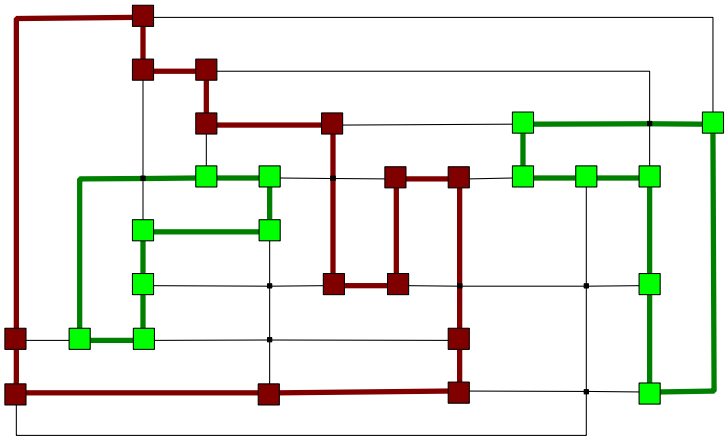
It is very interesting to investigate the required runtime for these proofs, as they are highly dependent on h (Figure 10.5): $P_{4i,4}$ ($h = 3$) can be solved very quickly—for these graphs, the secondary cycles decompose into i disjoint cycles, and the running time increases moderately with the graph size. The performance for $P_{4i+2,4}$ ($h = 1$) is the second best, as they contain $i/2$ disjoint secondary cycles. We have the worst running time for $P_{4i+1,4}$ ($h = 0$), as the secondary cycle does not decompose; furthermore the performance degrades quickly for larger graphs. We conclude with two exemplary crossing minimal drawings of $P_{14,4}$ and $P_{21,4}$, respectively, see Figure 10.6.

10.4 Extracting Proofs

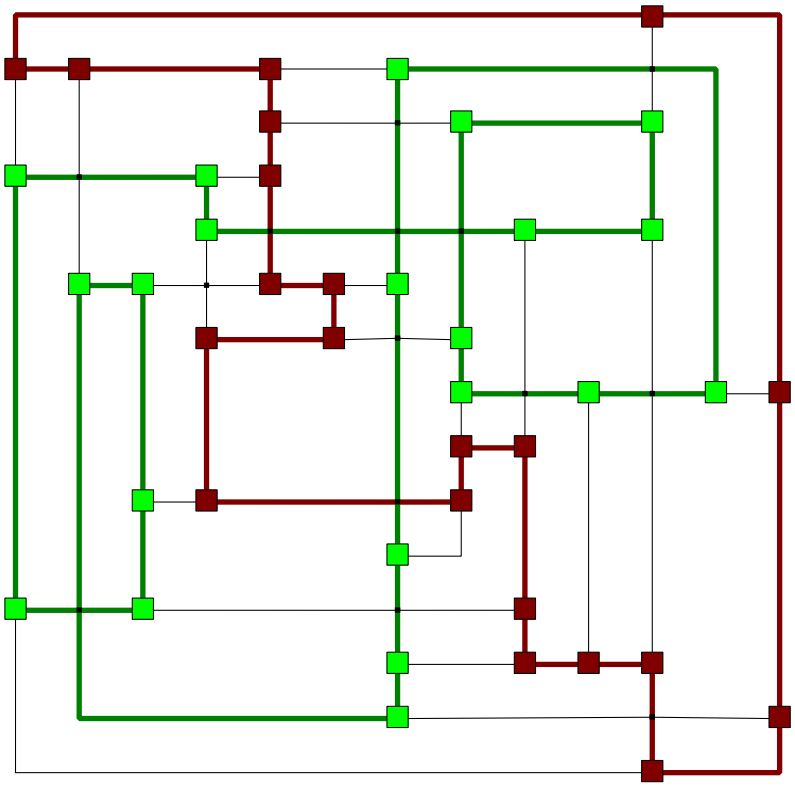
Theoretically, we have a formal proof for a crossing number after computing it via our ILP. Nonetheless, this does not constitute a proof that can be easily verified, and ignores the fact of possible bugs in the code of any necessary component. Even the fact that we have two distinct algorithms (based on SECM and OECM, respectively) seems not satisfiable enough, due to shared code and conceptual ideas.

Therefore we may want to extract a proof that is (comparably) easily verifiable, using code that is itself easily verifiable and has nothing in common with the code of the original algorithm. In order to establish a formal proof, one would have to perform the following steps:

1. Solve the problem using one of the branch-and-cut-and-price algorithms described in Chapters 5 and 6; store the optimal solution OPT.
2. For each leaf node in the branch-and-bound tree (or the root node if no branching was necessary):
   (a) Extract the branch information (variable fixings, K_5-branches), if any.
   (b) Extract the necessary variables and constraints from the final LP model.



(a) $\mathrm{cr}(P_{14,4}) = 8$



(b) $\mathrm{cr}(P_{21,4}) = 14$

Figure 10.6: Provably crossing minimal drawings of two generalized Petersen graphs. Thick edges denote the main and secondary cycles.

   (c) Translate the formal variables and constraints into a human readable format, e.g., by specifying the type of the constraint, listing the edges of the corresponding Kuratowski subdivision, etc.

3. Publish the information extracted above.

To validate the proof, one then has to perform the following steps:

1. Validate the branch information, in order to ensure that the multitude of subproblems describes the original problem completely.

2. For each subproblem:

   (a) Run a validation program that proves the graph theoretic soundness of the constraints, e.g., checking that the edge lists resemble Kuratowski constraints.

   (b) Run a validation program that proves that the LP relaxation of the subproblem induces that the solution cannot be better than the stored optimal solution we try to validate. Therefore it reinterprets the human readable variables and constraints in terms of our 0/1-ILP, adds the branching information and solves a single LP relaxation (via any trusted LP-solver). The objective function has to be larger than $OPT - 1$ (or $OPT - 2$ for K_{2k-1} due to Kleitman's parity argument).

   Note that this is sufficient as the introduction of additional constraints or variables could never decrease the objective function.

Chapter 11

Conclusion and Outlook

crossing: *(Road Construction)*
A place where two or more routes of transportation form a junction or intersection.

After summarizing some main aspects of current crossing number research, this thesis's Part II introduced a preprocessing scheme based on SPR-trees. We showed that we can compute a non-planar core for arbitrary graphs in linear time, which behaves invariant not only w.r.t. the crossing number but also w.r.t. the graph's skewness, thickness, coarseness and genus.

Afterwards, we presented the thesis's central ILP formulations for the traditional crossing number problem: one based on subdividing the edges of the given graph and one based on linear ordering. We discussed their common concepts and defined the crossing number polytope. We were able to show that several classes of Kuratowski constraints define facets of this polytope.

The exponential size of the ILPs required us to develop branch-and-cut algorithms facilitating sophisticated separation routines, in particular efficient Kuratowski extractions. Even when generating constraints on the fly, the number of variables is still prohibitive for computing the crossing number of larger graphs. Therefore we investigated several possibilities of generating variables dynamically as well. While a traditional pricing approach based on reduced costs only allowed moderate improvements, we presented application specific combinatorial column generation schemes that are crucial for the algorithms' success. These schemes detect the necessity for additional variables based on combinatorial and graph theoretic arguments; in contrast to traditional pricing, they have the valuable property that the LP relaxations always correspond to lower bounds of the full problem.

We discussed related crossing number concepts and how to apply heuristic or exact approaches to them. We described the minor-monotone and the hypergraph crossing numbers, and showed a previously unexplored relationship between them. We showed how to use corresponding heuristic algorithms to

find drawings of electrical circuits with clearly less crossings than previously reported. We also investigated the novel concept of simultaneous crossing numbers, and presented complexity results as well as heuristic and exact algorithms.

We implemented the presented approaches and showed experimentally that the exact crossing number algorithm is practical for medium sized general graphs with crossing numbers of up to 20 and beyond. Based on these investigations, we conclude that the OECM formulation, using combinatorial column generation, is the practically strongest one. Yet, SECM has benefits regarding its adaptability to other crossing number concepts. Knowing the exact crossing numbers, we can deduce that state-of-the-art heuristics for the problem are very close to optimality if the crossing number is not too large.

In the last chapter, we discussed how to use our algorithms to solve open problems in the realm of formerly purely theoretic questions regarding the crossing number of special graph classes. Initial experiments are promising, as we could verify the crossing number of the largest proven complete graph K_{11}. For the first time, we were able to show crossing numbers for generalized Petersen graphs $P_{n,4}$; we computed the numbers for all $n \leq 44$ and multiple beyond, and came up with a conjecture for the crossing number with general n.

We want to conclude with giving some more open questions that arose during the development of this thesis's material:

Non-planar Core Reduction. The reduction strategy is strong if the biconnected components of a graph contain only few non-planar triconnected components. If a graph is non-planar and triconnected, the reduction strategy will not modify the graph. Can we decompose R-nodes based on 4-connected components and analyze these components in order to shrink the graph further?

Additional Constraints. In practice, triangle constraints seldom lead to improved running times. Can we prove any properties of this constraint class? Can we identify additional classes of planarity constraints that are either practically more efficient or easier to separate than the general Kuratowski constraints used in SECM and OECM?

Polyhedral Strength of the Approaches. Can we show whether SECM or OECM is stronger from the polyhedral point of view? See Section 5.5 for a discussion why this question seems hard to answer.

Pairwise Crossing Number. Can we extend our ILP to solve the pairwise crossing number problem (Section 7.4), and to thereby probably find examples where $\mathrm{pcr}(G) \neq \mathrm{cr}(G)$.

Toroidal Grids. Recall Conjecture 10.2: Can we restrict the drawing of toroidal grids $T_{n,m}$, with $n \geq m$ and m odd, to have at most one

crossing per edge, without losing optimality?

Special Graph Classes. As noted before, special graph classes offer a wide field for research; see the beginning of Chapter 10 for an overview on the aims. In order to solve more such special graphs or graph classes, we have to identify and graph theoretically prove additional special properties of their optimal drawings. We can then try to turn this knowledge into linear constraints. Furthermore, it would be nice to be able to perform a simple minimal-proof extraction for clean-room verification, as sketched in Section 10.4.

Thrackles. A *thrackle drawing* [146] of a graph is a good drawing of a graph where each pair of non-adjacent edges crosses exactly once. John Conway asked if there exists a graph with more edges then nodes that allows a thrackle drawing. It is conjectured that this is not the case. In fact, the thrackle conjecture holds, if it can be shown that the one-point union of two even-length cycles cannot be thrackled.

By fixing all x-variables to 1 in OECM and eliminating the objective function, we could reuse our formulation to solve the realizability problem for such graphs. This could be used within a search framework, to identify a graph that allows such a drawing. This concept, of course, would only allow to disprove the conjecture.

Part IV

Backmatter

Curriculum Vitae

Personal Data

Name	(Dipl.Ing.) Markus CHIMANI
Date of birth	January 8th, 1980
Place of birth	Vienna, Austria
Citizenship	Austrian
Marital status	single
Address	Chair XI – Algorithm Engineering Faculty of Computer Science Dortmund University of Technology Otto-Hahn-Str. 14 44227 Dortmund, Germany
Telephone (office)	+49 / 231 / 755 7706
eMail	`markus.chimani@tu-dortmund.de`

Education

since 2005	PhD-student of Computer Science at Dortmund University of Technology (former name: University Dortmund), Chair XI (Algorithm Engineering). Topic: "Computing Crossing Numbers".
1999–2004	Master student of Computer Science at Vienna University of Technology. Focus on algorithmics and computer graphics. *April 27th, 2004:* Diploma (≈ Master of Science), with distinction: Avarage (first part:) 1.1, (second part:) 1.0 (best possible average: 1.0); below minimum duration of study. *Diploma thesis:* "Bend-Minimal Orthogonal Drawing of Non-Planar Graphs" (in English).
1998	Realgymnasium GRg XIX, 1990 Vienna, Austria. finished with distinction; Matura (≈ high school diploma).

Experience

since 2005	Research associate with teaching responsibilities at Dortmund University of Technology/University Dortmund; developer of parts of OGDF [118].
2004	Research visit at MERL (Mitsubishi Electric Research Laboratories) in Boston, MA, USA (6 months). Research projects: "Human Guided Optimization" and "Collaborative Interface Design".
2000–2004	Multiple tutorships (student assistant) for Introduction-to-Programming, Algorithms-and-Datastructures 1 and 2, and Compiler-Engineering at TU Vienna
2003, 2004	Developer of parts of AGD [2] at TU Vienna.
2000,2001,2002	Internships at Generali AG; software development (3× 2 months).
1999	Software developer at BEKO (4 months).
1998–1999	Military service (8 months).

Research Interests

- (Computation of) Crossing numbers and other non-planarity measures, graph drawing
- Network optimization
- Integer linear programming, branch-and-cut, column generation
- Combinatorial optimization
- Graph theory

Languages

Languages	German, English
Programming	C/C++, Java, Python, Pascal/Delphi, (Visual)Basic, Modula, Fortran, Prolog, Smalltalk

Publications

Refereed Conference Papers

2009

- M. Chimani, C. Gutwenger, P. Mutzel, C. Wolf. *Inserting a Vertex into a Planar Graph.* ACM-SIAM Symposium on Discrete Algorithms 2009, New York (SODA'09), pp. 375–383, ACM Press, 2009.

2008

- M. Chimani, C. Gutwenger, M. Jansen, K. Klein, P. Mutzel. *Computing Maximum C-Planar Subgraphs.* to appear in: 16th International Symposium on Graph Drawing 2008, Heraklion (GD'08).

- M. Chimani, P. Mutzel, I. Bomze. *A New Approach to Exact Crossing Minimization.* 16th Annual European Symposium on Algorithms 2008, Karlsruhe (ESA'08), LNCS 5193, pp. 284–296, Springer, 2008.

- M. Chimani, M. Kandyba, I. Ljubić, P. Mutzel. *Strong Formulations for the 2-Node-Connected Steiner Network Problems.* 2nd Annual International Conference on Combinatorial Optimization and Applications 2008, St. John's, Newfoundland (COCOA'08), LNCS 5165, pp. 190–200, Springer, 2008.

- M. Chimani, C. Gutwenger, P. Mutzel, H.-M. Wong. *Layer-Free Upward Crossing Minimization.* 7th International Workshop on Experimental Algorithms 2008, Cape Cod (WEA'08), LNCS 5038, pp. 55–68, Springer, 2008.

- M. Chimani, M. Jünger, M. Schulz. *Crossing Minimization meets Simultaneous Drawing.* 2008 IEEE Pacific Visualization Symposium, Kyoto (PacificVis'08), pp. 33–40, 2008.

- M. Chimani, M. Kandyba, I. Ljubic, P. Mutzel. *Obtaining Optimal k-Cardinality Trees Fast.* Workshop on Algorithm Engineering & Experiments 2008, San Francisco (ALENEX'08), SIAM, pp. 27–36, 2008.

2007

- M. Chimani, C. Gutwenger. *Algorithms for the Hypergraph and the Minor Crossing Number Problems.* 18th Annual International Symposium on Algorithms and Computation (ISAAC'07), LNCS 4835, pp. 184–195, Springer, 2007.

- M. Chimani, P. Mutzel, J.M. Schmidt. *Efficient Extraction of Multiple Kuratowski Subdivisions.* 15th International Symposium on Graph Drawing (GD'07), LNCS 4875, pp. 159–170, Springer, 2007.

- M. Chimani, M. Kandyba, M. Preuss. *Hybrid Numerical Optimization for Combinatorial Network Problems.* 4th International Workshop on Hybrid Metaheuristics (HM'07), LNCS 4771, pp. 185–200, Springer, 2007.

- M. Chimani, M. Kandyba, P. Mutzel. *A New ILP Formulation for 2-Root-Connected Prize-Collecting Steiner Networks.* 15th Annual European Symposium on Algorithms (ESA'07), LNCS 4698, pp. 681–692, Springer, 2007.

2006

- M. Chimani, C. Gutwenger, P. Mutzel. *Experiments on Exact Crossing Minimization using Column Generation.* 5th International Workshop on Experimental Algorithms (WEA'06), LNCS 4007, pp. 303–315, Springer, 2006.

- C. Gutwenger, M. Chimani. *Non-Planar Core Reduction of Graphs.* 13th International Symposium on Graph Drawing (GD'05), LNCS 3843, pp. 223–234, Springer, 2006.

2005

- C. Rich, C. Sidner, N. Lesh, A. Garland, S. Booth, M. Chimani. *Diamond-Help: A Collaborative Interface Framework for Networked Home Appliances.* IEEE International Conference on Distributed Computing Systems Workshops (ICDCS'05) / 5th International WorkShop on Smart Appliances and Wearable Computing (IWSAWC'05), pp. 514–519, IEEE Computer Society Press, 2005.

- M. Chimani, N. Lesh, M. Mitzenmacher, C. Sidner, H. Tanaka. *A Case Study in Large-Scale Interactive Optimization.* International Conference on Artificial Intelligence and Applications (AIA'05), IASTED-Proc., pp. 24–29, Acta Press, Anaheim CA, 2005.

- M. Chimani, G.W. Klau, R. Weiskircher. *Non-Planar Orthogonal Drawings with Fixed Topology.* SofSem'05: Theory and Practice of Computer Science, LNCS 3381, pp. 96–105, Springer, 2005.

Journal Articles

- M. Chimani, C. Gutwenger, P. Mutzel, H.-M. Wong. *Layer-Free Upward Crossing Minimization.* Submitted.

- M. Chimani, M. Kandyba, I. Ljubic, P. Mutzel. *Obtaining Optimal k-Cardinality Trees Fast.* to appear in: Journal of Experimental Algorithms.

- M. Chimani, C. Gutwenger. *Non-Planar Core Reduction of Graphs.* Discrete Mathematics. In print.

- M. Chimani, C. Gutwenger, P. Mutzel. *Experiments on Exact Crossing Minimization using Column Generation.* Submitted.

- C. Buchheim, M. Chimani, D. Ebner, C. Gutwenger, M. Jünger, G.W. Klau, P. Mutzel, R. Weiskircher. *A Branch-and-Cut Approach to the Crossing Number Problem.* Discrete Optimization, Vol. 5(2), Special Issue in Memory of George B. Dantzig, pp. 373–388, 2008.

- M. Chimani, C. Gutwenger, P. Mutzel. *On the Minimum Cut of Planarizations.* Electronic Notes in Discrete Mathematics, Vol. 28, pp. 177–184, 2007.

- C. Rich, C. Sidner, N. Lesh, A. Garland, S. Booth, M. Chimani. *Diamond-Help: A New Interaction Design for Networked Home Appliances.* Personal and Ubiquitous Computing, Vol. 10(2), pp. 187–190, Springer, 2006.

Posters and Demos

- M. Chimani, P. Hliněný, P. Mutzel. *Approximating the Crossing Number of Apex Graphs.* to appear in: 16th International Symposium on Graph Drawing (GD'08).

- M. Chimani, C. Gutwenger, M. Jünger, K. Klein, P. Mutzel, M. Schulz. *The Open Graph Drawing Framework.* 15th International Symposium on Graph Drawing (GD'07).

- C. Rich, C. Sidner, N. Lesh, A. Garland, S. Booth, M. Chimani. *Diamond-Help: A New Interaction Design for Networked Home Appliances.* Third International Conference on Appliance Design 2005 (3AD), Finalist of Design Competition.

- C. Rich, C. Sidner, N. Lesh, A. Garland, S. Booth, M. Chimani. *DiamondHelp: A Collaborative Task Guidance Framework for Complex Devices*. 20th National Conference on Artificial Intelligence; Intelligent Systems Demonstrations (AAAI'05), pp. 1700–1701, 2005.

Bibliography

[1] Abacus: A Branch-And-CUt System. University of Cologne, Faculty of Mathematics and Natural Sciences, Chair of Computer Science (Prof. Jünger), `http://www.informatik.uni-koeln.de/abacus/`, 2007.

[2] AGD: Algorithms for Graph Drawing. Max-Planck institute Saarbrücken, Vienna University of Technology, University of Cologne, University Trier, `http://www.ads.tuwien.ac.at/AGD/`, 2000.

[3] V. B. Alekseev and V. S. Gončakov. The thickness of an arbitrary complete graph. *Mathematics of the USSR-Sbornik*, 30(2):187–202, 1976.

[4] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2006.

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

[6] C. Batini, M. Talamo, and R. Tamassia. Computer aided layout of entity-relationship diagrams. *Journal of Systems and Software*, 4:163–173, 1984.

[7] L. W. Beineke and G. Chartrand. The coarseness of a graph. *Compositio Mathematica*, 19:290–298, 1968.

[8] L. W. Beineke and R. K. Guy. The coarseness of the complete bipartite graph. *Canadian Journal of Mathematics*, 21:1086–1096, 1969.

[9] L. W. Beineke and F. Harary. The thickness of the complete graph. *Canadian Journal of Mathematics*, 17:850–859, 1965.

[10] L. W. Beineke, F. Harary, and J. W. Moon. On the thickness of the complete bipartite graph. *Proceedings of the Cambridge Philosophical Society*, 60:1–5, 1964.

[11] S. N. Bhatt and F. T. Leighton. A framework for solving VLSI graph layout problems. *Journal of Computer and System Sciences*, 28:300–343, 1984.

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

[13] D. Bokal. On the crossing numbers of cartesian products with paths. *Jounal of Combinatorial Theory, Series B*, 2007(3):381–384, 97.

[14] D. Bokal, É. Czabarkab, L. A. Székely, and I. Vrt'o. Graph minors and the crossing number of graphs. *Electronic Notes in Discrete Mathematics*, 28:169–175, 2007.

[15] D. Bokal, G. Fijavz, and B. Mohar. The minor crossing number. *SIAM Journal on Discrete Mathematics*, 20:344–356, 2006.

[16] K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976.

[17] K. J. Börözky, J. Pach, and G. Tóth. Planar crossing numbers of graphs embeddable in another surface. *International Journal of Foundations of Computer Science*, 17(5):1005–1016, 2006.

[18] J. M. Boyer, P. F. Cortese, M. Patrignani, and G. Di Battista. Stop minding your P's and Q's: Implementing a fast and simple DFS-based planarity testing and embedding algorithm. In *Proc. GD '03*, volume 2912 of *LNCS*, pages 25–36. Springer, 2004.

[19] J. M. Boyer and W. J. Myrvold. Stop minding your P's and Q's: A simplified O(n) planar embedding algorithm. In *Proc. SODA '99*, pages 140–146. SIAM, 1999.

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

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

[22] F. Brglez, D. Bryan, and K. Kozminski. Combinatorial profiles of sequential benchmark circuits. In *Proc. Circuits and Systems*, pages 1929–1934, 1989.

[23] C. Buchheim, M. Chimani, D. Ebner, C. Gutwenger, M. Jünger, G. W. Klau, P. Mutzel, and R. Weiskircher. A branch-and-cut approach to the crossing number problem. *Discrete Optimization, Special Issue in Memory of George B. Dantzig*, 5(2):373–388, 2008.

[24] C. Buchheim, D. Ebner, M. Jünger, G. W. Klau, P. Mutzel, and R. Weiskircher. Exact crossing minimization. In *Proc. GD '05*, volume 3843 of *LNCS*, pages 37–48. Springer, 2006.

[25] C. Buchheim, M. Jünger, A. Menze, and M. Percan. Bimodal crossing minimization. In *Proc. COCOON '06*, volume 4112 of *LNCS*, pages 497–506. Springer, 2006.

[26] S. Cabello and B. Mohar. Crossing and weighted crossing number of near planar graphs. In *Proc. GD '08*, LNCS. Springer. to appear.

[27] N. Chiba, T. Nishizeki, S. Abe, and T. Ozawa. A linear algorithm for embedding planar graphs using *PQ*-Trees. *Journal of Computer and System Sciences*, 30(1):54–76, 1985.

[28] M. Chimani and C. Gutwenger. Algorithms for the hypergraph and the minor crossing number problems. In *Proc. ISAAC '07*, volume 4835 of *LNCS*, pages 184–195. Springer, 2007.

[29] M. Chimani and C. Gutwenger. Non-planar core reduction of graphs. *Discrete Mathematics*, 2008. In press. A preliminary version appeared in *Proc. GD '05*, LNCS 3843, pp. 223–234.

[30] M. Chimani, C. Gutwenger, and P. Mutzel. Experiments on exact crossing minimization using column generation. In *Proc. WEA '06*, volume 4007 of *LNCS*, pages 303–315. Springer, 2006.

[31] M. Chimani, C. Gutwenger, and P. Mutzel. On the minimum cut of planarizations. *Electronic Notes in Discrete Mathematics*, 28:177–184, 2007.

[32] M. Chimani, C. Gutwenger, P. Mutzel, and C. Wolf. Inserting a vertex into a planar graph. submitted, 2008.

[33] M. Chimani, C. Gutwenger, P. Mutzel, and H.-M. Wong. Layer-free upward crossing minimization. In *Proc. WEA '08*, volume 5038 of *LNCS*, pages 55–68. Springer, 2008.

[34] M. Chimani, P. Hliněný, and P. Mutzel. Approximating the crossing number of apex graphs. submitted, 2008.

[35] M. Chimani, M. Jünger, and M. Schulz. Crossing minimization meets simultaneous drawing. In *Proc. PacificVis '08*, pages 33–40, 2008.

[36] M. Chimani, M. Kandyba, I. Ljubić, and P. Mutzel. Obtaining optimal k-cardinality trees fast. In *Proc. ALENEX '08*, pages 27–36, 2008.

[37] M. Chimani, P. Mutzel, and I. Bomze. A new approach to exact crossing minimization. In *Proc. ESA '08*, LNCS. Springer, 2008. to appear.

[38] M. Chimani, P. Mutzel, and J. M. Schmidt. Efficient extraction of multiple Kuratowski subdivisions (TR). Technical Report TR07-1-002, June 2007, Dortmund University of Technology, June 2007.

[39] M. Chimani, P. Mutzel, and J. M. Schmidt. Efficient extraction of multiple Kuratowski subdivisions. In *Proc. GD '07*, volume 4875 of *LNCS*, pages 159–170. Springer, 2008.

[40] Coin-OR: COmputational INfrastructure for Operations Research. `www.coin-or.org`.

[41] G. B. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954.

[42] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.

[43] G. Demoucron, Y. Malgrange, and R. Pertuiset. Graphes planaires: Reconnaissance et construction de representations planaires topologiques. *Rev. Francaise Recherche Operationelle*, 8:33–47, 1964.

[44] G. Di Battista, P. Eades, R. Tamassia, and I.G. Tollis. *Graph Drawing: algorithms for the visualization of graphs.* Prentice Hall, 1999.

[45] G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of four graph drawing algorithms. *Computational Geometry: Theory and Applications*, 7(5–6):303–325, 1997.

[46] G. Di Battista and R. Tamassia. On-line planarity testing. *SIAM Journal on Computing*, 25:956–997, 1996.

[47] R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics.* Springer-Verlag, Heidelberg, 3 edition, 2005.

[48] Garcia-Moreno E. and G. Salazar. Bounding the crossing number of a graph in terms of the crossing number of a minor with small maximum degree. *Journal of Graph Theory*, 36:168–173, 2001.

[49] D. Ebner. Optimal crossing minimization using integer linear programming. Master's thesis, Vienna University of Technology, 2005.

[50] C. Erten and S. G. Kobourov. Simultaneous embedding of planar graphs with few bends. In *Proc. GD '04*, volume 3383 of *LNCS*, pages 195–205. Springer, 2005.

[51] T. Eschbach, W. Günther, and B. Becker. Orthogonal circuit visualization improved by merging the placement and routing phases. In *Proc. VLSID '05*, pages 433–438, 2005.

[52] T. Eschbach, W. Günther, and B. Becker. Orthogonal hypergraph drawing for improved visibility. *Journal of Graph Algorithms and Applications*, 10(2):141–157, 2006.

[53] A. Estrella-Balderrama, E. Gassner, M. Jünger, M. Percan, M. Schaefer, and M. Schulz. Simultaneous geometric graph embeddings. In *Proc. GD '07*, volume 4875 of *LNCS*, pages 280–290. Springer, 2008.

[54] G. Even, S. Guha, and B. Schieber. Improved approximations of crossings in graph drawing. In *Proc. STOC '00*, pages 296–305, 2000.

[55] S. Even and R. E. Tarjan. Computing an *st*-Numbering. *Theoretical Computer Science*, 2(3):339–344, 1976.

[56] G. Exoo, F. Harary, and J Kabell. The crossing numbers of some generalized Petersen graphs. *Mathematica Scandinavica*, 48:184–188, 1981.

[57] S. Fiorini. On the crossing number of generalized Petersen graphs. *Annals of Discrete Mathematics*, 30:225242, 1986.

[58] F. Frati. Embedding graphs simultaneously with fixed edges. In *Proc. GD '06*, volume 4372 of *LNCS*, pages 108–113. Springer, 2007.

[59] F. Frati, M. Kaufmann, and S. Kobourov. Constrained simultaneous and near-simultaneous embeddings. In *Proc. GD '07*, volume 4875 of *LNCS*, pages 268–279. Springer, 2008.

[60] H. de Fraysseix and P. Ossona de Mendez. On cotree-critical and DFS cotree-critical graphs. *Journal of Graph Algorithms an Applications*, 7(4):411–427, 2003.

[61] H. de Fraysseix, P. Ossona de Mendez, and P. Rosenstiehl. Trémaux Trees and planarity. *International Journal of Foundations of Computer Science*, 17(5):1017–1030, 2006.

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

[63] M. R. Garey and D. S. Johnson. The rectilinear steiner tree problem is NP-complete. *SIAM Journal on Applied Mathematics*, 32:826–834, 1977.

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

[65] E. Gassner, M. Jünger, M. Percan, M. Schaefer, and M. Schulz. Simultaneous graph embeddings with fixed edges. In *Proc. WG '06*, volume 4271 of *LNCS*, pages 325–335. Springer, 2006.

[66] M. Geyer, M. Kaufmann, and I. Vrt'o. Two trees which are self-intersecting when drawn simultaneously. In *Proc. GD '05*, volume 3843 of *LNCS*, pages 201–210. Springer, 2006.

[67] E. Di Giacomo and G. Liotta. A note on simultaneous embedding of planar graphs. In *Proc. EWCG '05*, pages 207–210, 2005.

[68] I. Gitler, P. Hliněný, J. Leanos, and G. Salazar. The crossing number of a projective graph is quadratic in the face-width. *Electronic Notes in Discrete Mathematics*, 29:219–223, 2007.

[69] A. J. Goldstein. An efficient and constructive algorithm for testing whether a graph can be embedded in a plane. In *Proc. Graph and Combinatorics Conference '63*, 1963.

[70] R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64:275–278, 1958.

[71] A. Grigoriev and H. Bodlaender. Algorithms for graphs embeddable with few crossings per edge. *Algorithmica*, 49(1):1–11, 2007.

[72] M. Grohe. Computing crossing numbers in quadratic time. In *Proc. STOC '01*, 2001.

[73] C. Gutwenger. *Applications of SPQR-Trees in the Planarization Approach for Drawing Graphs*. PhD thesis, Dortmund University of Technology, 2008. to appear.

[74] C. Gutwenger, K. Klein, and P. Mutzel. Planarity testing and optimal edge insertion with embedding constraints. In *Proc. GD '06*, volume 4372 of *LNCS*, pages 126–137. Springer, 2007.

[75] C. Gutwenger and P. Mutzel. A linear time implementation of SPQR trees. In *Proc. GD '00*, volume 1984 of *LNCS*, pages 77–90. Springer, 2001.

[76] C. Gutwenger and P. Mutzel. An experimental study of crossing minimization heuristics. In *Proc. GD '03*, volume 2912 of *LNCS*, pages 13–24. Springer, 2004.

[77] C. Gutwenger, P. Mutzel, and R. Weiskircher. Inserting an edge into a planar graph. *Algorithmica*, 41(4):289–308, 2005.

[78] R. K. Guy. A coarseness conjecture of Erdös. *Journal of Combinatorial Theory*, 3:38–42, 1967.

[79] R. K. Guy. The decline and fall of zarankiewicz's theorem. In *Proof Techniques in Graph Theory, Proc. 2nd Ann Arbor Graph Theory Conference '68*, pages 63–69. Academic Press, 1969.

[80] R. K. Guy. Crossing numbers of graphs. In *Proc. Graph Theory and Applications*, LNM, pages 111–124. Springer, 1972.

[81] R. K. Guy and L. W. Beineke. The coarseness of the complete graph. *Canadian Journal of Mathematics*, 20:888–894, 1968.

[82] R. K. Guy, T. Jenkyns, and Schaer J. The toroidal crossing number of complete graphs. *Journal of Combinatorial Theory*, 4:376–390, 1968.

[83] F. Harary. Research problem. *Bulletin of the American Mathematical Society*, 67:542, 1961.

[84] F. Harary. *Graph Theory*. AddisonWesley, Reading, MA, 1969.

[85] P. Hliněný. Crossing number is hard for cubic graphs. *Journal of Combinatorial Theory, Series B*, 96:455–471, 2006.

[86] P. Hliněný and G. Salazar. On the crossing number of almost planar graphs. In *Proc. GD '05*, volume 4372 of *LNCS*, pages 162–173. Springer, 2006.

[87] P. Hliněný and G. Salazar. Approximating the crossing number of toroidal graphs. In *Proc. ISAAC '07*, volume 4835 of *LNCS*, pages 148–159. Springer, 2007.

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

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

[90] Ilog CPLEX, v.9.0. `www.ilog.com`.

[91] D. S. Johnson and H. O. Pollak. Hypergraph planarity and the complexity of drawing venn diagrams. *Journal of Graph Theory*, 11(3):309–325, 1987.

[92] M. Jünger and P. Mutzel. Solving the maximum weight planar subgraph. In *Proc. IPCO '93*, pages 479–492, 1993.

[93] M. Jünger and P. Mutzel. The polyhedral approach to the maximum planar subgraph problem: New chances for related problems. In *Proc. GD '94*, volume 894 of *LNCS*, pages 119–130. Springer, 1995.

[94] M. Jünger and S. Thienel. The ABACUS system for branch-and-cut-and-price-algorithms in integer programming and combinatorial optimization. *Software: Practice & Experience*, 30(11):1325–1352, 2000.

[95] K. Kawarabayashi and B. Reed. Computing crossing number in linear time. In *Proc. STOC '07*, pages 382–380, 2007.

[96] D.J. Kleitman. The crossing number of $K_{5,n}$. *Journal of Combinatorial Theory*, 9:315–323, 1970.

[97] D.J. Kleitman. A note on the parity of the number of crossings of a graph. *Journal of Combinatorial Theory, Series B*, 21(1):88–89, 1976.

[98] H. Klemetti, I. Lapinleimu, E. Mäkinen, and M. Sieranta. A programming project: Trimming the spring algorithm for drawing hypergraphs. *SIGCSE Bulletin*, 27(3):34–38, 1995.

[99] E. de Klerk, J. Maharry, D.V. Pasechnik, R.B. Richter, and G. Salazar. Improved bounds for the crossing numbers of $K_{m,n}$ and K_n. *SIAM Journal on Discrete Mathematics*, 20(1):189–202, 2006.

[100] A. Kotzig. On certain decompositions of a graph. *Mat.-Fyz. Časopis*, 5:144–151, 1955.

[101] J. Kratochvíl. String graphs II: Recognizing string graphs is NP-hard. *Journal of Combinatorial Theory, Series B*, 52:67–78, 1991.

[102] J. Kratochvíl. Crossing number of abstract topological graphs. In *Proc. GD '98*, volume 1547 of *LNCS*, pages 238–245. Springer, 1998.

[103] J. Kratochvíl and J. Matoušek. String graphs requiring exponential representations. *Journal of Combinatorial Theory*, B 53:1–4, 1991.

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

[105] J. Kyncl. The complexity of several realizability problems for abstract topological graphs. In *Proc. GD '07*, volume 4875 of *LNCS*, pages 137–158. Springer, 2008.

[106] Glebsky L. and G. Salazar. The crossing number of $C_m \times C_n$ is as conjectured for $m \geq m(m+1)$. *Journal of Graph Theory*, 47(1):53–72, 2004.

[107] A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. In *Theory of Graphs: Int't Symp.*, pages 215–232. Gordon and Breach, 1967.

[108] I. Ljubić, R. Weiskircher, U. Pferschy, G. Klau, P. Mutzel, and M. Fischetti. An algorithmic framework for the exact solution of the prize-collecting Steiner tree problem. *Mathematical Programming, Series B*, 105(2–3):427–449, 2006.

[109] E. Mäkinen. How to draw a hypergraph. *International Journal of Computer Mathematics*, 34:177–185, 1990.

[110] D. McQuillan and R. B. Richter. On the crossing numbers of certain generalized Petersen graphs. *Discrete Mathematics*, 104:311–320, 1992.

[111] K. Mehlhorn and P. Mutzel. On the embedding phase of the Hopcroft and Tarjan planarity testing algorithm. *Algorithmica*, 16(2):233–242, 1996.

[112] K. Mehlhorn and S. Näher. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press, 1999.

[113] P. Mutzel. *The Maximum Planar Subgraph Problem*. PhD thesis, University of Cologne, 1994.

[114] P. Mutzel, T. Odenthal, and M. Scharbrodt. The thickness of graphs: A survey. *Graphs and Combinatorics*, 14(1):59–73, 1998.

[115] P. Mutzel and T. Ziegler. The constrained crossing minimization problem. In *Proc. GD '99*, volume 1731 of *LNCS*, pages 175–185. Springer, 1999.

[116] N. Nahas. On the crossing number of $k_{m,n}$. *Electronic Journal of Combinatorics*, 10, 2003. Note 8.

[117] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Discrete Mathematics and Optimization. Wiley-Interscience, 1999.

[118] OGDF: Open Graph Drawing Framework. Dortmund University of Technology, Faculty of Computer Science, Chair for Algorithm Engineering, `www.ogdf.net`, 2007.

[119] J. Pach and G. Tóth. Which crossing number is it anyway? *Journal of Combinatorial Theory, Series B*, 80(2):225–246, 2000.

[120] J. Pach and G. Tóth. Recognizing string graphs is decidable. In *Proc. GD '01*, volume 2265 of *LNCS*, pages 247–260. Springer, 2002.

[121] J. Pach and G. Tóth. Crossing number of toroidal graphs. In *Proc. GD '05*, volume 3843 of *LNCS*, pages 334–342. Springer, 2006.

[122] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33(1):60–100, 1991.

[123] S. Pan and R. B. Richter. The crossing number of K_{11} is 100. *Journal of Graph Theory*, 56:128–134, 2007.

[124] M. Pelsmajer, M. Schaefer, and D. Štefankovič. Odd crossing number is not crossing number. In *Proc. GD '05*, volume 3843 of *LNCS*, pages 386–396. Springer, 2006.

[125] M. Pelsmajer, M. Schaefer, and D. Štefankovič. Crossing number of graphs with rotation systems. In *Proc. GD '06*, LNCS. Springer, 2007.

[126] H. C. Purchase. Which aesthetic has the greatest effect on human understanding? In *Proc. GD '97*, volume 1353 of *LNCS*, pages 248–261. Springer, 1997.

[127] R. B. Richter and G. Salazar. The crossing number of $P(N, 3)$. *Graphs and Combinatorics*, 18:381–394, 2002.

[128] G. Salazar. On the crossing number of $C_m \times C_n$. *Journal of Graph Theory*, 28(3):163–170, 1998.

[129] G. Salazar. On the crossing numbers of loop networks and generalized Petersen graphs. *Discrete Mathematics*, 302(1-3):243–253, 2005.

[130] G. Sander. Layout of directed hypergraphs with orthogonal hyperedges. In *Proc. GD '03*, volume 2912 of *LNCS*, pages 381–386. Springer, 2004.

[131] M.L. Saražin. The crossing number of the generalized Petersen graph $p(10, 4)$ is four. *Mathematica Slovaca*, 4747:189192, 1997.

[132] M. Schaefer, E. Sedgwick, and D. Štefankovič. Recognizing string graphs in NP. *Journal of Computer and System Sciences*, 67(2):365–380, 2003.

[133] J. M. Schmidt. Effiziente Extraktion von Kuratowski-Teilgraphen. Master's thesis, University Dortmund, March 2007. (in German).

[134] A. Schrijver. *Theory of Linear and Integer Programming.* Discrete Mathematics and Optimization. Wiley-Interscience, 1998.

[135] F. Shahrokhi, O. Sykora, L. A. Szekely, and I. Vrt'o. Drawing graphs on surfaces with few crossings. *Algorithmica*, 16:118–131, 1996.

[136] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. Sys. Man. Cyb.*, 11(2):109–125, 1981.

[137] L. A. Székely. A successful concept for measuring non-planarity of graphs: the crossing number. *Discrete Mathematics*, 276(1-3):331–352, 2004.

[138] C. Thomassen. The graph genus problem is np-complete. *Journal of Algorithms*, 10(4):568–576, 1989.

[139] P. Turán. A note of welcome. *Journal of Graph Theory*, 1:7–9, 1977.

[140] I. Vrt'o. Crossing numbers of graphs: A bibliography. `ftp://ftp.ifi.savba.sk/pub/imrich/crobib.pdf`, 2008.

[141] J. Širáň. Additivity of the crossing number of graphs with connectivity 2. *Periodica Mathematica Hungarica*, 15(4):301–305, 1984.

[142] K. Wagner. Über eine Eigenschaft der ebenen Komplexe. *Mathematische Annalen*, 114(1):570–590, 1937.

[143] R. Weiskircher. *New Applications of SPQR-Trees in Graph Drawing*. PhD thesis, Saarland University, 2002.

[144] L. A. Wolsey. *Integer Programming*. Discrete Mathematics and Optimization. Wiley-Interscience, 1998.

[145] D. R. Wood and J. A. Telle. Planar decompositions and the crossing number of graphs with an excluded minor. In *Proc. GD '06*, volume 4372 of *LNCS*, pages 150–161. Springer, 2007.

[146] D. R. Woodall. *Combinatorial Mathematics and its Applications*, chapter Thrackles and deadlock, pages 335–348. Academic Press, London, 1969.

[147] K. Zarankiewicz. The solution of a certain problem on graphs of P. Turán. *Bulletin de l'Academie Polonaise des sciences, Cl. III*, 1:167–168, 1953.

[148] W. P. Zheng. On the crossing number of $K_m \times P_n$. *Graphs and Combinatorics*, 23:327–336, 2007.

[149] T. Ziegler. *Crossing Minimization in Automatic Graph Drawing*. PhD thesis, Saarland University, Germany, 2001.

Index