

A Branch-and-Cut Approach to the Crossing Number Problem[★]

Christoph Buchheim¹, Markus Chimani², Dietmar Ebner³,
Carsten Gutwenger², Michael Jünger¹, Gunnar W. Klau⁴,
Petra Mutzel², and René Weiskircher⁵

Abstract

The crossing number of a graph is the minimum number of edge crossings in any drawing of the graph in the plane. Extensive research has produced bounds on the crossing number and exact formulae for special graph classes, yet the crossing numbers of graphs such as K_{11} or $K_{9,11}$ are still unknown. Finding the crossing number is NP-hard for general graphs and no practical algorithm for its computation has been published so far. We present an integer linear programming formulation that is based on a reduction of the general problem to a restricted version of the crossing number problem in which each edge may be crossed at most once. We also present cutting plane generation heuristics and a column generation scheme. As we demonstrate in a computational study, a branch-and-cut algorithm based on these techniques as well as recently published preprocessing algorithms can be used to successfully compute the crossing number for small to medium sized general graphs.

Key words: Crossing number, branch-and-cut, column generation
1991 MSC: 05C10, 90C35, 90C57

[★] This work was partially supported by the Marie Curie RTN ADONET 504438 funded by the EU.

¹ Department of Computer Science, University of Cologne, Germany
{buchheim,mjuenger}@informatik.uni-koeln.de

² Department of Computer Science, University of Dortmund, Germany
{markus.chimani,carsten.gutwenger,petra.mutzel}@cs.uni-dortmund.de

³ Institute of Computer Languages, Vienna University of Technology, Austria
ebner@complang.tuwien.ac.at

⁴ Department of Mathematics and Computer Science, Free University Berlin and DFG Research Center MATHEON, Germany, gunnar@math.fu-berlin.de

⁵ CSIRO Mathematical and Information Sciences, Melbourne, Australia
rene.weiskircher@csiro.au

1 Introduction

Crossing minimization is among the oldest and most fundamental problems arising in the area of automatic graph drawing and *VLSI* design and — at the same time — very easy to formulate:

“Given a graph $G = (V, E)$, draw it in the plane with a minimum number of edge crossings”.

A drawing of G is a mapping of each node $v \in V$ to a distinct point and each edge $e = (v, w) \in E$ to a curve connecting the incident nodes v and w without passing through any other node. Common points of two edges that are not incident nodes are called *crossings*. The minimum number of crossings among all drawings of G is denoted by $\text{cr}(G)$.

The main goal in automatic graph drawing is to obtain a layout that is easy to read and understand. The definition of layout quality often depends on the particular application and is hard to measure. However, the number of edge crossings is among the most important criteria [41]. Figure 1 shows a comparison of different drawings for the same graph preferring different aesthetic criteria.

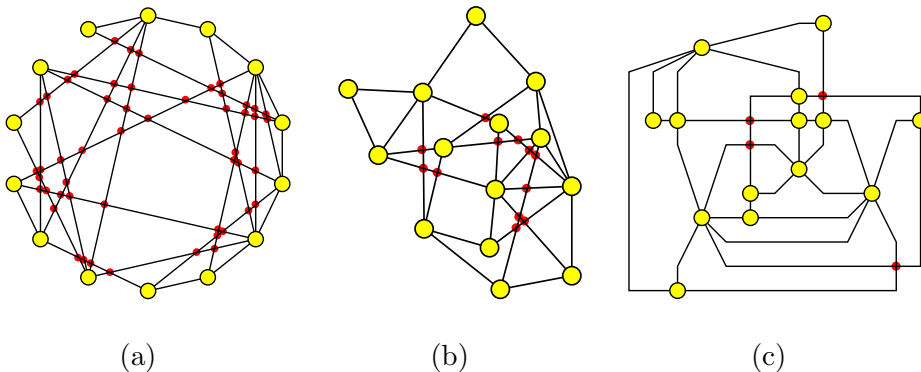


Fig. 1. Three drawings of the same graph with 51 (a), 12 (b), and 4 crossings (c). Most aesthetic criteria, for example, number of bends, uniformity of edge lengths, or drawing area, favor the first two drawings, while the last drawing is preferable with respect to the number of edge crossings

In fact, the crossing minimization problem is even older than the area of automatic graph drawing. It goes back to P. Turán, who proposed the problem in his “Notes of Welcome” in the first issue of the *Journal of Graph Theory* [42]. While working in a labor camp during the Second World War, he noticed that crossings of the rails 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 graphs $K_{m,n}$.

In 1953, K. Zarankiewicz [43] and K. Urbaník independently claimed a solution to this problem by providing a drawing rule for complete bipartite graphs $K_{m,n}$ with $Z(m, n) = \lfloor \frac{m}{2} \rfloor \lfloor \frac{m-1}{2} \rfloor \lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor$ crossings. About ten years later, their proof of optimality was shown to be wrong and it is still unknown whether the conjecture holds. The situation for complete graphs K_n is similar. Their crossing number is conjectured to be $Z(n) = \frac{1}{4} \lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor \lfloor \frac{n-2}{2} \rfloor \lfloor \frac{n-3}{2} \rfloor$, which has been verified for graphs of up to ten nodes by Guy [22]. Both conjectures are based on a drawing rule and therefore serve as an upper bound for $\text{cr}(K_n)$ and $\text{cr}(K_{m,n})$.

Recently, de Klerk et al. [30,31] devised a method for computing asymptotic lower bounds for $\text{cr}(K_{m,n})$ and $\text{cr}(K_n)$ based on semidefinite programming. They show that

$$\lim_{n \rightarrow \infty} \frac{\text{cr}(K_{m,n})}{Z(m, n)} \geq 0.8594 \frac{m}{m-1} \quad \text{and} \quad \lim_{n \rightarrow \infty} \frac{\text{cr}(K_n)}{Z(n)} \geq 0.83 .$$

It is well known that the general crossing minimization problem is *NP*-hard [16]. More precisely, it is shown that the *crossing number problem* is *NP*-complete:

“Given a graph G and a non-negative integer K , decide whether there is a drawing of G with at most K edge crossings.”

However, for fixed K , we can obtain a polynomial time algorithm by examining all possible configurations with up to K crossings. Clearly, this algorithm is not appropriate in practical applications for larger values of K . Recently, Grohe could show that this problem can be solved in time $O(|V|^2)$ [17]. Even though the exponent is independent of K , the constant factor of his algorithm grows doubly exponentially in K . Therefore, this method is also of little relevance in practice.

The search for approximation algorithms did not lead to significant results either. While there is no known polynomial time approximation algorithm with any type of quality guarantee for the general problem, Bhatt and Leighton could derive an algorithm for graphs with *bounded degree* that approximates the number of crossings *plus the number of nodes* in polynomial time [2]. Due to the complexity of the crossing minimization problem, many restricted versions have been considered in the literature. However, in most cases, e.g., for bipartite, linear, and circular drawings, the problem remains *NP*-hard [14,37,36].

The most prominent and practically successful heuristic for the crossing minimization problem is the *planarization approach* [1]. This technique is used by our algorithm as a primal heuristic and is explained in detail in Section 3.3.

Contribution and Structure. In this paper, we present the first algorithm that is able to compute the crossing number of general sparse graphs of moderate size. We state computational results on a popular benchmark set of graphs, the so-called Rome library [10]. The approach uses a new integer linear programming formulation of the problem combined with strong heuristics and problem reduction techniques. This enables us to compute the crossing number for nearly all graphs on up to 40 nodes in the Rome library within a time limit of five minutes per graph. In Section 2, we show how to reduce the problem to the easier problem of computing crossing-minimal drawings where each edge is involved in at most one crossing. We present an integer linear programming formulation for the reduced problem and a branch-and-cut algorithm to compute provably optimal solutions for this formulation in Section 3. An optional preprocessing technique for reducing the size of the input graph is explained in Section 4. Section 5 summarizes the computational results obtained with our new approach for the crossing number problem for Rome library graphs. Conclusions and further work are presented in Section 6.

2 Crossing Restricted Drawings

The area of crossing minimization is closely related to the field of planarity testing, i.e., to decide whether a given graph G can be drawn in the plane without any edge crossings. This task can be performed surprisingly fast, more precisely in linear time [24,4]. One of the ground-breaking results in this research area was *Kuratowski's theorem* [33], which provides a full characterization of planar graphs based on the complete graph K_5 and the complete bipartite graph $K_{3,3}$. In the following, a *subdivision* of a graph G is obtained by repeatedly replacing its edges by a path of length two.

Theorem 1 (Kuratowski's theorem) *A finite graph is planar if and only if it contains no subgraph that is a subdivision of K_5 or $K_{3,3}$.*

As a consequence of Theorem 1, at least two edges in every Kuratowski subdivision, i.e., a subdivision of K_5 or $K_{3,3}$, have to cross in every planar drawing of a graph G . As we describe in Section 3, we can obtain inequalities from this observation that fully characterize the set of *realizable* crossing configurations (corresponding to drawings in the plane).

Unfortunately, even the problem to decide whether there is a drawing for a given set of edge crossings is *NP*-complete [32]. This variant is known as the *realizability problem* and can be stated as follows:

“Given a set of edge pairs D , does there exist a drawing of G such that two edges $e, f \in E$ cross each other if and only if $(e, f) \in D$?”

However, if we know for each edge the *order* of edges crossing it, it is easy to solve the problem by placing dummy nodes on all chosen crossings and testing the resulting graph for planarity.

One way to work around the realizability problem is the reduction to crossing restricted drawings: a drawing is called *crossing restricted* (or *CR-drawing*) if each edge crosses at most one other edge. Not surprisingly, there are graphs that do not admit any crossing restricted drawing. Pach and Tóth [40] showed the following more general theorem:

Theorem 2 *Let $G = (V, E)$ be a simple graph drawn in the plane so that every edge is crossed by at most k others. If $0 \leq k \leq 4$, then we have*

$$|E| \leq (k + 3)(|V| - 2) . \tag{1}$$

They could further prove that this bound cannot be improved for $0 \leq k \leq 2$ and that for *any* $k \geq 1$ the following inequality holds:

$$|E| \leq \sqrt{16.875 k} |V| \approx 4.108\sqrt{k} |V| \tag{2}$$

Furthermore, Bodlaender and Grigoriev proved that it is *NP*-complete to decide whether there is a crossing restricted drawing for a given graph G [3]. If there is such a drawing, we denote the minimum number of crossings among all crossing restricted drawings of G by $\text{crr}(G)$.

Even if there is a crossing restricted drawing for G , its crossing number $\text{crr}(G)$ does not necessarily coincide with $\text{cr}(G)$. Consider the graph in Figure 2. The left drawing shows an optimum drawing with two crossings while the right drawing shows an optimum drawing among all crossing restricted drawings.

Given a graph $G = (V, E)$, we create a graph $G^* = (V^*, E^*)$ by replacing every edge $e \in E$ with a path of length $|E|$. Then for any non-negative number K the graph G can be drawn with K crossings if and only if there is a *crossing restricted drawing* of G^* with K crossings. Therefore, it is sufficient to solve the crossing minimization problem for crossing restricted drawings on an extended graph in order to solve the general crossing minimization problem — clearly at significant computational expense. Since the transformation obviously can be done in polynomial time, the *NP*-completeness of the corresponding decision problem for crossing restricted drawings follows immediately from the *NP*-completeness for the general crossing number problem [16].

It is well-known that every graph G admits a *good drawing* with a minimum number of crossings. A good drawing is a drawing that satisfies the following conditions:

- (1) no edge crosses itself

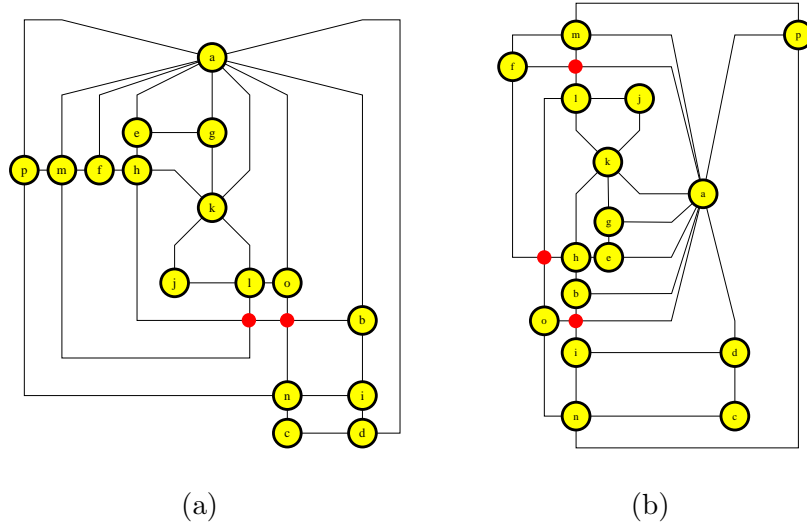


Fig. 2. A crossing minimum drawing of a graph with two crossings (a) and an optimum crossing restricted drawing of the same graph with three crossings (b). Both drawings were produced with our exact algorithm presented in this paper

- (2) adjacent edges do not cross each other
- (3) non-adjacent edges cross each other at most once

Therefore it is sufficient to replace every edge $e = (v, w) \in E$ with a path of length $|E| - |\delta(v)| - |\delta(w)| - 1$, while $\delta(v)$ denotes the degree of node v . We can further decrease the number of required dummy edges by using any upper bound for $\text{cr}(G)$, since no single edge can cross more than $\text{cr}(G)$ other edges in any optimum solution.

3 ILP Formulation, Branch-and-Cut, and Column Generation

In order to solve the general crossing minimization problem with the approach outlined in the previous section, we present an integer linear programming formulation for the problem of finding a CR-drawing of a graph G with a minimum number of edge crossings. In Section 3.1, we discuss the set of variables and a family of simple constraints that model crossing restrictedness. In Section 3.2, we introduce the Kuratowski constraints that ensure realizability. Then we present three components of our algorithm that are crucial for its performance: a powerful primal heuristic (Section 3.3), a column generation scheme (Section 3.4), and a preprocessing technique (Section 4).

3.1 Variables and CR-Constraints

The first step in developing our ILP is the choice of the variable space. We will use binary variables corresponding to potential edge crossings in the drawing of G . As discussed in Section 2, using one integer variable per edge pair does not yield a tractable formulation in general, since even the problem of realizability of the given crossing configuration is NP-complete. In particular, with this set of variables no polynomial set of constraints can characterize the feasible solutions, even if integrality is assumed, unless P equals NP. The same holds for every set of constraints that can be separated in polynomial time.

Therefore, we restrict ourselves to CR-drawings, for which feasibility can be checked easily. Let $G = (V, E)$ be a graph and let D denote a set of unordered pairs of edges of G . In analogy to the notation for drawings of G , we call D *crossing restricted* if for every $e \in E$ there is at most one $f \in E$ such that $(e, f) \in D$. Furthermore, D is called *realizable* if there is a drawing of G such that there is a crossing between the edges e and f if and only if $(e, f) \in D$.

Next, let D be crossing restricted and let G_D be the graph that is obtained by introducing a dummy node for each pair of edges $(e, f) \in D$. More precisely, we introduce dummy nodes on both e and f and identify them. Notice that G_D is only well-defined if D is crossing restricted, as otherwise it would not be clear where to place the dummy nodes. We can obtain the following simple characterization:

Lemma 3 *Let D be crossing restricted. Then D is realizable if and only if the graph G_D is planar.*

Using a linear time planarity testing and embedding algorithm, we can thus test in time $O(|V| + |D|)$ whether D is realizable, and compute a realizing drawing in the affirmative case.

Lemma 3 shows that the realizability problem is easy if we restrict ourselves to CR-drawings. In this case we can use the following natural set of variables: for each unordered pair of edges (e, f) , we use a binary variable $x_{e,f}$ set to 1 if and only if the two edges e and f cross each other. In other words, the matrix x models the incidence vectors of a subset $D \subseteq E^2$. Since Lemma 3 requires D to be crossing restricted, we have to model this property by linear constraints. Any given edge e may accept at most one crossing, i.e., we have to introduce the inequality

$$\sum_{f \in E} x_{e,f} \leq 1 \tag{3}$$

for each edge $e \in E$. In the following, we call (3) the *CR-constraint* for e .

3.2 Kuratowski Constraints and Separation

In order to complete the desired ILP formulation for crossing minimization restricted to CR-drawings, we have to model realizability: a given solution of the ILP specified so far has to be excluded by a linear constraint if the corresponding crossing restricted set D is not realizable. The key is again Lemma 3, by which it suffices to ensure planarity of G_D . In other words, G_D must not contain a subdivision of K_5 or $K_{3,3}$, as stated by Theorem 1. This is modeled by the *Kuratowski constraints* introduced in this section.

In the construction of G_D , an edge e was split up whenever there was an edge f with $(e, f) \in D$. For both split edges e_1 and e_2 , we set $\hat{e}_1 = \hat{e}_2 = e$. If an edge e is not split, we set $\hat{e} = e$. Moreover, for every subgraph $H = (V', E')$ of G_D , let $\hat{H} = \{\hat{e} \mid e \in E'\} \subseteq E$. Less formally, \hat{H} contains all edges of G involved in the subgraph H of G_D .

Proposition 4 *Let $D \subseteq E^2$ be crossing restricted and let H be any subdivision of K_5 or $K_{3,3}$ in G_D . Then every realizable crossing restricted set satisfies*

$$C_{D,H} : \sum_{(e,f) \in \hat{H}^2 \setminus D} x_{e,f} \geq 1 - \sum_{(e,f) \in \hat{H}^2 \cap D} (1 - x_{e,f}). \quad (4)$$

The inequalities (4) are called *Kuratowski constraints*.

PROOF. Suppose (4) is violated for some realizable crossing restricted set D' with incidence vector x . Since $x_{e,f} \in \{0, 1\}$ for all $e, f \in E$, inequality (4) can only be violated if the left hand side is zero and the right hand side is one, which means that

$$x_{e,f} = \begin{cases} 0 & \text{for all } (e, f) \in \hat{H}^2 \setminus D, \\ 1 & \text{for all } (e, f) \in \hat{H}^2 \cap D. \end{cases}$$

By definition of the vector x , this implies that $\hat{H}^2 \cap D' = \hat{H}^2 \cap D$, in other words, that G_D and $G_{D'}$ coincide on the subgraph induced by \hat{H} . Consequently, the K_5 - or $K_{3,3}$ -subdivision H is also a subgraph of $G_{D'}$. From Kuratowski's Theorem we derive that $G_{D'}$ is not planar. This contradicts the realizability of D' by Lemma 3. \square

Next, we show that the Kuratowski constraints suffice to give a complete characterization of realizability of crossing restricted sets:

Theorem 5 *Let $G = (V, E)$ be a simple graph. A subset of E^2 is crossing restricted and realizable if and only if its incidence vector x satisfies all CR-*

constraints and the Kuratowski constraints $C_{D,H}$ for every crossing restricted set $D \subseteq E^2$ and every forbidden subgraph H in G_D .

PROOF. Let $D' \subseteq E^2$ and let x be the incidence vector of D' . It is easy to see that the CR-constraints hold if and only if D' is crossing restricted. Hence, for the rest of the proof, we assume that D' is crossing restricted. It remains to show that D' is realizable if and only if all Kuratowski constraints $C_{D',H}$ hold.

If D' is realizable, then every Kuratowski constraint is satisfied according to Proposition 4. Thus we have to show that at least one Kuratowski constraint is violated if D' is not realizable. By Lemma 3, the graph $G_{D'}$ is not planar if D' is not realizable, hence $G_{D'}$ contains a subdivision H of K_5 or $K_{3,3}$. We claim that $C_{D',H}$ is violated.

It follows from the definition of x that every $x_{e,f} \in \hat{H}^2 \setminus D'$ is zero, hence the left hand side of $C_{D',H}$ is zero. Furthermore, every $x_{e,f} \in \hat{H}^2 \cap D'$ is one, hence

$$\sum_{(e,f) \in \hat{H}^2 \cap D'} (1 - x_{e,f}) = 0,$$

and the right hand side of $C_{D',H}$ is one. This completes the proof. \square

For every crossing restricted and realizable set $D \subseteq E^2$, we can compute a corresponding drawing in polynomial time. Thus we can reformulate the crossing minimization problem for crossing restricted drawings as follows:

“Given a graph $G = (V, E)$, find a realizable crossing restricted set $D \subseteq E^2$ of minimum cardinality”.

This leads to the following ILP-formulation. We use $x(F)$ as an abbreviation for $\sum_{(e,f) \in F} x_{e,f}$:

$$\begin{aligned} \min \quad & x(E^2) \\ \text{s.t.} \quad & \sum_{f \in E} x_{e,f} \leq 1 && \text{for all } e \in E \\ & x(\hat{H}^2 \setminus D) - x(\hat{H}^2 \cap D) \geq 1 - |\hat{H}^2 \cap D| && \text{for every crossing restricted} \\ & && \text{set } D \subseteq E^2 \text{ and every} \\ & && \text{forbidden subgraph } H \text{ of } G_D \\ & x_{e,f} \in \{0, 1\} && \text{for all unordered pairs } (e, f) \in E^2 \end{aligned}$$

It is clearly impractical to generate all constraints $C_{D,H}$ in advance. Instead, we embed the given formulation into a branch-and-cut framework, separating violated inequalities dynamically. This leads to a separation problem: given a (possibly fractional) LP-solution, how can we decide whether one of the Kuratowski constraints is violated?

For integer solutions, this problem can be solved easily: the LP-solution gives rise to a crossing restricted set $D \subseteq E^2$. If G_D is planar, we know from Lemma 3 that D is realizable, hence no Kuratowski inequality is violated. Otherwise, the separation problem is reduced to the search for a Kuratowski subdivision H in G_D ; by the proof of Theorem 5, the corresponding constraint $C_{D,H}$ is violated. The latter problem can be solved in linear time [9,5].

For fractional LP-solutions, we can solve the separation problem only heuristically. For this, we round all LP-solutions to integers, then we can apply the algorithm for integer solutions mentioned above.

3.3 Primal Heuristics

The most prominent and practically successful method for solving the crossing minimization problem heuristically is the *planarization approach*. This approach was introduced by Batini, Talamo and Tamassia in [1] and can be viewed as a general framework which addresses the problem with a two step strategy. Each step aims at solving a particular optimization problem for which various solution methods are possible. Let G be the graph for which we want to find a crossing minimal drawing. Then, the two steps to be executed are:

- (1) Compute a planar subgraph of G that contains as many edges as possible.
- (2) Reinsert the edges not contained in the planar subgraph. During this edge insertion process, edge crossings that occur when inserting an edge are replaced by dummy nodes with degree four, i.e., if two edges cross, both edges are split and the two nodes produced by the split are identified. Hence, the graphs constructed during the edge insertion step are planar. The objective is to keep the number of dummy nodes (and thus the number of crossings in the final drawing) as small as possible.

The outcome of the planarization procedure is a planar graph G_p which consists of the original nodes of G and additional dummy nodes. If we replace the dummy nodes in a planar drawing of G_p by edge crossings, we obtain a drawing of G . Hence, we also say that G_p is a *planarized representation* of G . Both, finding a planar subgraph of maximum size and re-inserting edges with a minimum number of crossings are NP-hard optimization problems; see [35,15,38].

Many algorithms for the computation of planar subgraphs have been published, including an exact branch-and-cut algorithm by Jünger and Mutzel [28], a $4/9$ -approximation algorithm by Călinescu et al. [7], algorithms based on incremental planarity testing [12,34], and fast heuristics, e.g., [6,26,27,13,25]. In our experiments, we used the algorithm by Jayakumar et al. [26]. Although the algorithm has quadratic worst case running time, it is very quick in practice. Moreover, it can easily be randomized, so that calling the algorithm several times improves the quality of the results significantly.

The planar subgraph computed in the first step is a good starting point for finding a drawing of G with few crossings. In practice, we expect that only a small number of edges has to be re-inserted. However, the choice of the edge insertion technique may have a considerable impact on the quality of the final solution. Gutwenger and Mutzel [20] have conducted an extensive study on crossing minimization heuristics including different methods for edge insertion and postprocessing techniques.

Usually, the edges are inserted individually one by one. The simple approach for inserting a single edge is to fix an embedding and to compute a shortest path in the geometric dual graph. However, the choice of the embedding may have a large influence on the number of edge crossings. A more sophisticated algorithm introduced by Gutwenger, Mutzel, and Weiskircher [21] allows to insert an edge with the minimum number of crossings among all embeddings in linear running time. After all edges have been inserted, a straight-forward postprocessing technique can further reduce the number of crossings. It repeatedly removes edges with many crossings and tries to find a better insertion path. A further observation is that the order in which the edges are inserted also affects the final number of crossings. Calling the complete edge insertion process several times and inserting the edges in different, randomly chosen orders may significantly improve the solution.

Let E' be the edges of the planar subgraph. We remark that the original *edge insertion problem (EIP)* does not allow crossings between edges in E' . To our knowledge, there is no practically applicable exact algorithm known for solving the EIP. Since we are only interested in a solution with few crossings, we also include edges of E' in the postprocessing step, so that the final solution may not be a valid solution for the EIP. However, it is straight-forward to modify the branch-and-cut algorithm presented in this paper so that it also solves the EIP to optimality. We simply have to exclude the edges in E' from being re-inserted in the postprocessing step of the primal heuristic and forbid crossings between edges of E' by fixing the respective crossing variables to 0.

Primal Heuristic in the Branch-and-Cut Approach. After obtaining a fractional solution of the current LP-relaxation, we can deduce two possibly

distinct integer solutions – one where all fractional values are interpreted as 0, and one based on a rounding scheme where all values below a certain threshold are interpreted as 0, and values above that threshold are interpreted as 1. We can then interpret these integer solutions as partial planarizations, onto which we can apply our primal heuristic. Since the obtained partial planarizations are highly redundant, we do not run the primal heuristic after each ILP relaxation step, but only when the induced integer solutions have actually changed.

3.4 Column Generation

An important drawback of the ILP introduced above is the large number of variables. It contains one variable for each pair of edge segments. In the worst case, we have to replace every original edge by $\Omega(m)$ segments, so that the total number of variables is as huge as $\Omega(m^4)$. This is clearly impractical for large or even medium-sized graphs. However, this situation arises mainly because of the reduction to CR-drawings, which is crucial for our ability to model the problem at all. Yet the number of variables with value one in an optimum solution of the ILP is only a small fraction of the total number of variables, as it equals the minimum number of crossings. The latter is always in $O(m^2)$, but usually much smaller; see Section 5 for the crossing numbers of graphs in the Rome library.

In consideration of this, it is an obvious idea to use column generation, i.e., to start with a small subset of the given variables and add further variables only according to need. In contrast to the well-known generic pricing scheme first introduced by Dantzig and Wolfe [8], our column generation criteria are of a combinatorial rather than an algebraic nature. More precisely, our decisions to activate variables are not based on reduced costs but have a very natural interpretation, as explained in the following.

We always start with only one segment per edge. Diverging from the model introduced above, we do not add any CR-constraint for these initial edge segments. This will guarantee that there is always a feasible solution for our current ILP formulation, which simplifies the implementation of our approach.

Now the column generation step is performed as follows. Assume that the current, possibly fractional, LP-value of the variable $x_{e,f}$ is $\bar{x}_{e,f}$. Then we first check whether any of the CR-constraints for initial edge segments (which were not included, as described above) is violated, i.e., whether

$$\sum_f \bar{x}_{e,f} > 1$$

holds for some edge segment e . Here, the sum is taken over all edge segments f

such that the variable $x_{e,f}$ is active. Every initial segment e with a violated CR-constraint is extended by adding a new segment, belonging to the same original edge of G . More formally, we split up the initial segment e , thus obtaining a new segment e' , and to activate every variable corresponding to a crossing between e' and one of the segments f with $\bar{x}_{e,f} > 0$. In other words, all segments crossing e with some fraction in the current LP-solution are now allowed to cross e' , too.

In contrast to the initial segments, every new segment obtained this way will be accompanied by its CR-constraint; the latter is added to the ILP immediately whenever a new edge segment is introduced. As a last step, we decrease the objective function coefficient of all new variables by some small value $\varepsilon > 0$, such that new edge segments will always be preferred over the initial ones when crossings are distributed—notice that e and e' would be equivalent otherwise. By this, we make sure that the initial segment e is only overloaded if necessary, i.e., its CR-constraint is only violated if all edge segments belonging to the same original edge as e together do not suffice to host all crossings assigned by the LP-solution.

As long as the number of segments for f with $\bar{x}_{e,f} > 0$ is greater than one, we continue to activate new variables. However, we do not always have to extend the segment e : if e has been extended by some segment e' in an earlier step and if one of the segments f with $\bar{x}_{e,f} > 0$ is not allowed to cross e' , then we only activate the variable $x_{e',f}$ instead of further extending the segment e , again with an objective function coefficient decreased by ε .

Using the described column generation strategy, the Kuratowski constraints could be left unchanged without making the ILP invalid. However, if there is any active constraint $C_{D,H}$ in our ILP such that e is contained in $\hat{H} \cap D$, the strength of this constraint degrades, as potential crossings can be shifted from e to e' now. In order to avoid this, we duplicate all Kuratowski constraints, adding copies with e replaced by e' .

Notice that the column generation step must always be performed before separation of new constraints. Moreover, if any variables have been activated, we have to proceed to the next LP solution, skipping the separation step. The reason is that the separation requires validity of all CR-constraints, as it is based on searching for Kuratowski subdivisions in the graph G_D . The latter is only well-defined if D is crossing restricted.

The correctness of this column generation scheme follows from the fact that we never add any CR-constraints for the initial edge segments. This implies that every solution of the crossing number problem on the original graph corresponds to a feasible solution of our current LP relaxation, at every point of our optimization process. However, we cannot construct planarizations from

these solutions as long as some of these CR-constraints are violated. So by introducing new segments with slightly lower objective function coefficients, we aim at obtaining a solution satisfying all CR-constraints without explicitly introducing them. As soon as all CR-constraints are satisfied and the solution is integer, we therefore get an optimal solution of the crossing number problem.

The computational results presented in Section 5 show that the presented column generation strategy performs very well in practice; the number of variables activated during the entire optimization process is only a very small fraction of the number of potential variables. The running time decreases significantly compared to the branch-and-cut-approach without column generation.

4 Preprocessing

Before starting the actual branch-and-cut algorithm, we apply graph reduction techniques that try to reduce the size of the input graph without affecting its crossing number. Such techniques are most promising for sparse graphs, so we expect a significant reduction of the size of the input graph in many practical applications. It is well known that it is sufficient to compute the crossing number for each block of the graph separately. If B_1, \dots, B_k are the blocks of G , i.e., its maximally 2-connected subgraphs, then

$$\text{cr}(G) = \sum_{i=1}^k \text{cr}(B_i) .$$

Recent work by Gutwenger and Chimani [18] shows that also for 2-connected graphs further reductions are possible. Consider a decomposition of G into two subgraphs S and K such that $S \cup K = G$ and S shares no edges and exactly two nodes s and t with K . We call S an *st-component* of G ; if, in addition, the graph $S \cup (s, t)$ is planar, then S is called a *planar st-component* of G . Obviously, a planar *st-component* can be drawn planar with s and t on the external face. A key observation is that we can always draw a Jordan curve in such a drawing which exactly crosses the edges of a minimum *st-cut* in S .

Let λ be the cardinality of a minimum *st-cut* in S , and let K^* be the graph obtained by inserting a single edge $e_{st} = (s, t)$ with weight λ into K . The edge e_{st} can be seen as a placeholder for the planar *st-component* S . If each crossing with e_{st} counts as λ crossings, we can extend a drawing of the reduced graph K^* to a drawing of G with the same number of crossings. In this drawing, an edge crossing e_{st} in the drawing of K^* is routed across the edges of a minimum *st-cut* instead; see Figure 3. On the other hand, it can be shown

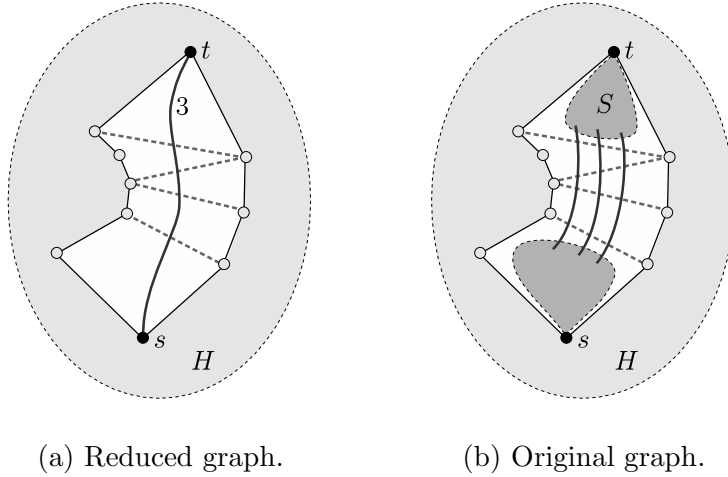


Fig. 3. Reduction of a planar st -component S to an edge with weight $\lambda = 3$.

that the crossing number of K^* is a lower bound for the number of crossing in any drawing of G . Hence, the reduced graph K^* has the same crossing number as G .

Applying this reduction strategy repeatedly leads to a weighted graph with the same crossing number as G . In this case, a crossing between two edges with weight w_1 and w_2 counts as $w_1 \cdot w_2$ many crossings. This is equivalent to replacing an edge with weight w by a bundle of w parallel edges. A very efficient way to apply this strategy exhaustively uses the decomposition of a graph into its triconnected components [23], which can be represented by the data structure SPQR-tree [11,19].

The SPQR-tree \mathcal{T} of a 2-connected graph reflects its 3-connectivity structure which is comprised of

- serial structures (S -nodes);
- parallel structures (P -nodes); and
- triconnected structures (R -nodes).

With each node μ of \mathcal{T} , a *skeleton* graph G_μ is associated. According to the type of μ , its skeleton graph is either a cycle of at least three nodes (S-node), a bundle of at least three parallel edges (P-node), or a triconnected simple graph (R-node).

A skeleton can be seen as a sketch of G in the following sense. An edge (u, v) in G_μ is either a *real* edge corresponding to an edge (u, v) in G , or a *virtual* edge corresponding to a uv -component of G . The respective uv -component of a virtual edge is determined by a neighbor ν of μ whose skeleton G_ν contains an edge (u, v) as well. Hence, the skeletons of two adjacent nodes μ and ν can be merged by contracting the edge (μ, ν) , identifying the corresponding

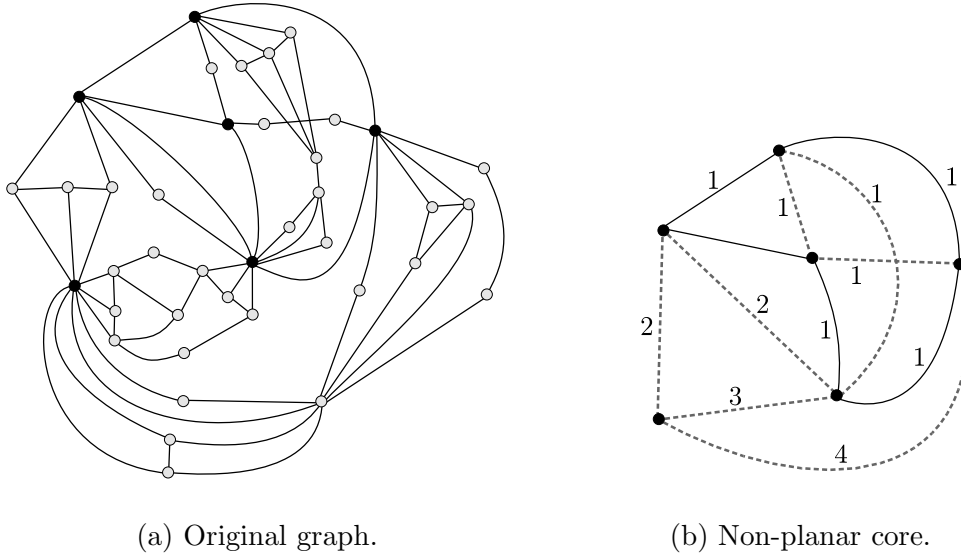


Fig. 4. An example for the non-planar core of a graph.

virtual edges in the skeletons, and finally removing the resulting virtual edge. Exhaustively merging skeletons recreates the graph G .

Obviously, only skeletons of R-nodes can be non-planar. If we determine which R-nodes have a non-planar skeleton, it is easy to find planar st -components of maximal size. We just look for the smallest subtree \mathcal{S} of \mathcal{T} that contains all R-nodes with non-planar skeletons. Then, \mathcal{S} induces a minimal reduction of G with respect to the reduction strategy described above. The reduced graph is obtained from \mathcal{S} by merging the skeletons of its tree nodes. Figure 4 gives an example of this reduction strategy.

The entire reduction algorithm, including the construction of the SPQR-tree and the computation of the edge weights in the reduced graph, can be implemented to run in $O(|V| + |E|)$ time; see [18]. Observe that the edge weights can be computed in linear time, since we only look for minimum st -cuts in planar graphs. The resulting weighted graph is called the *non-planar core* of G . In our experiments, we observed that every non-planar block of a graph in the Rome library was reduced to the skeleton of a single non-planar R-node. However, this seems to be merely a special property of the graphs in this particular benchmark set.

5 Experiments

We implemented the presented algorithms as part of the open-source C++ library *Open Graph Drawing Framework (OGDF)* [39]. We use the free branch-

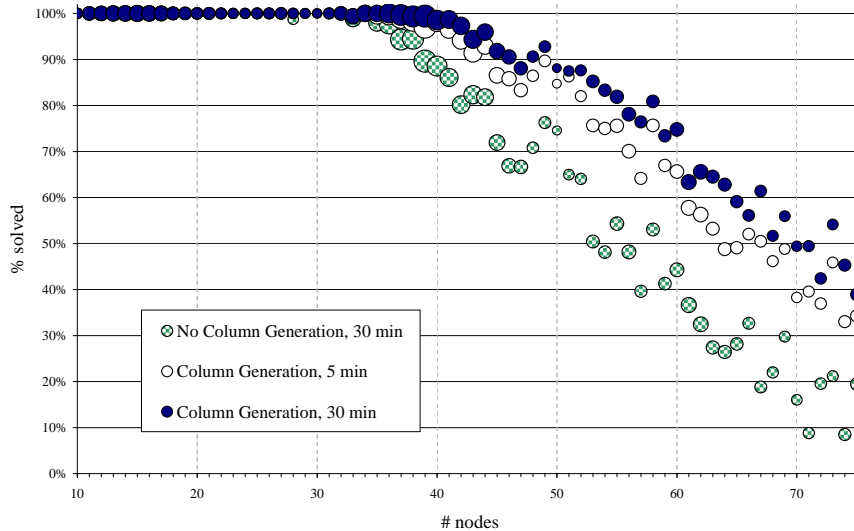


Fig. 5. The percentage of graphs solved to provable optimality, categorized by the number of nodes. The size of the data points represents the number of graphs falling into the respective classes.

and-cut-and-price framework *ABACUS* [29], in conjunction with the commercial optimization library *CPLEX* (version 9). The tests were performed on a single AMD Opteron CPU with 2.4 GHz, and 32 GB RAM shared between 4 CPUs. As it turned out, the memory consumption seldomly exceeded 1 GB.

To test the performance of our new algorithm, we used a benchmark set of graphs of the University of Rome III, introduced in [10]. 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 life” graphs used in 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 from the benchmark set is about 1.35.

Due to the complexity of the crossing minimization problem, we only consider graphs of up to 75 nodes. As it turns out, the number of edges in the non-planar core is of often more important than the number of nodes in the original graph. Figure 5 shows the percentage of graphs we could solve to provable optimality within 5 and 30 minutes, respectively. Using the number of edges in the non-planar core (Figure 6) as the x-axis, we can observe the clear dependence on that parameter.

To understand the test set better, it is worth looking at Figure 7. The Rome graph library consists of many planar graphs and non-planar graphs for which we know that their crossing number is 1, based on the primal heuristic; we call these graphs *trivial*, since they are of no interest for our algorithm. As we can

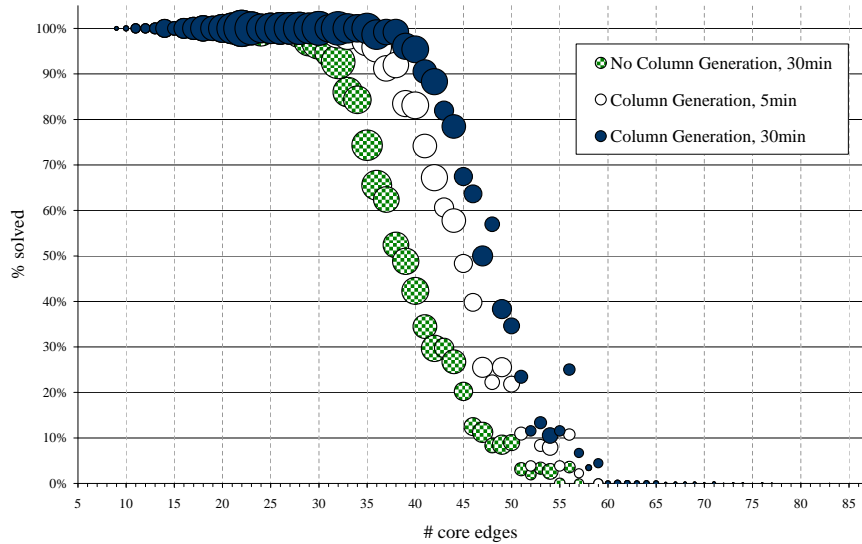


Fig. 6. The percentage of graphs solved to provable optimality, categorized by the number of core edges. The size of the data points represents the number of graphs falling into the respective classes.

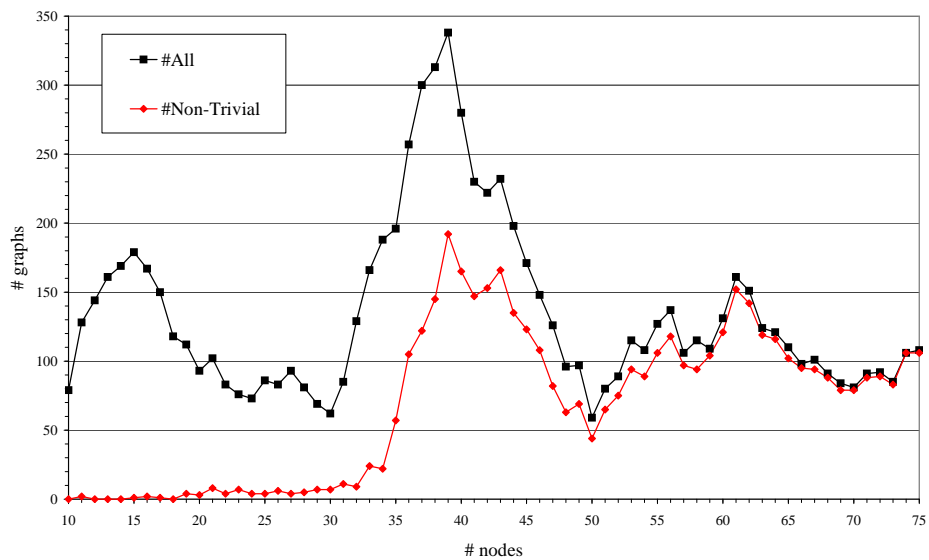


Fig. 7. The number of graphs compared to the number of non-trivial graphs for the Rome library.

see, there are only very few graphs with up to 33 nodes which are non-trivial.

Now that we can solve many instances to provable optimality, it is interesting to see the quality of the commonly used planarization heuristics, described in Section 3.3. As it turns out, the heuristic often directly computes the optimum crossing number, especially if the graph or the crossing number is relatively

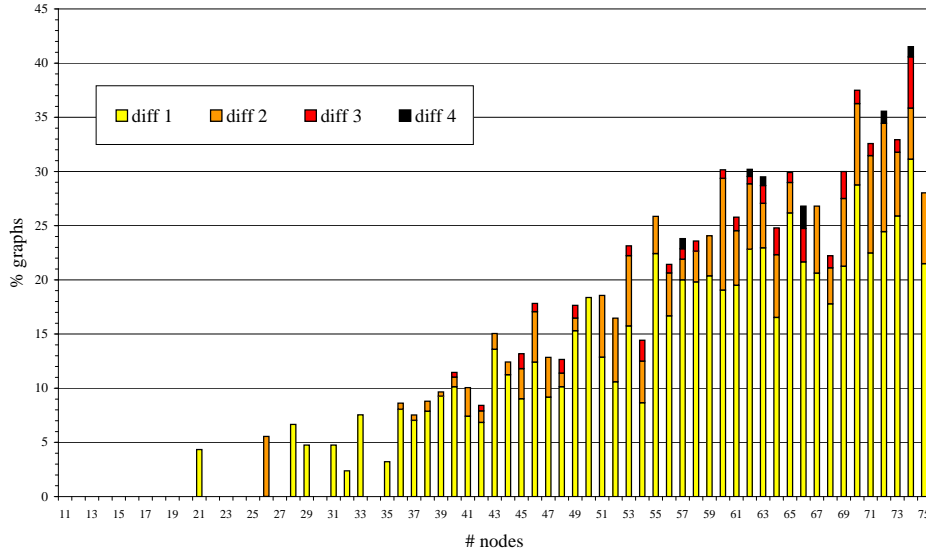


Fig. 8. Percentage of graphs for which we could improve on the heuristic solution. The stacked bar labeled “diff n ” refers to the instances with a decrease of n in the number of crossings compared with the result of the heuristic.

small. Figure 8 shows that even for the large graphs, we could only improve the result for less than half of the non-trivial graphs within 30 minutes. This statistic also includes results where the branch-and-cut algorithm produced lower crossing numbers than the pure heuristics during the computation, but could not prove optimality within the time bound.

Figure 9 demonstrates the efficiency of our column generation scheme, based on the successfully optimized test instances. Basically all graphs with a crossing number of two were detected as such by the heuristics. Only two graphs with actual crossing number of 1 have been estimated with 2 crossings by the heuristic. In all those cases where the heuristic gives an upper bound of two, our algorithm does not allow any edge splitting at all, since it is enough to prove the non-existence of any better solution, i.e., a solution with one crossing. Hence in these cases the column generation scheme did of course not reduce the number of variables; but since these cases are quite easy to compute anyway, this is of no concern for us. Notice that for a crossing number of 3, after obtaining an upper bound of 3, the starting set of variables already constitutes 25% of all possible variables. Figure 10 shows the discrepancy between the number of variables we would have had to generate without our column generation scheme and the number of variables we actually generated. As shown in Figure 11, we typically need about 40% additional variables, compared to the starting set where each edge pair is represented by exactly one segment. It also shows that this percentage is correlated with the number of edges in the non-planar core.

The runtime analysis (Figures 12 and 13) shows that the performance mainly

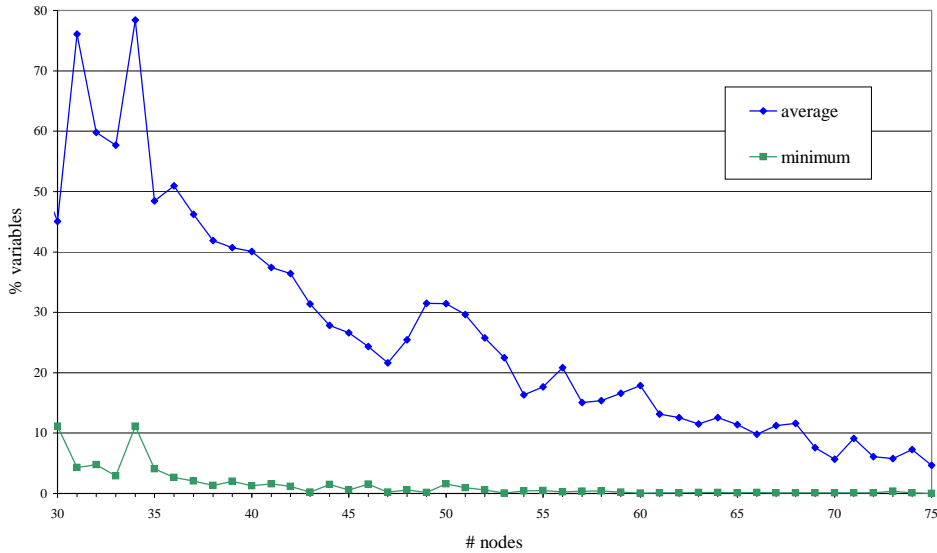


Fig. 9. The number of variables used by the column generation scheme relative to the full number of variables, by the number of nodes.

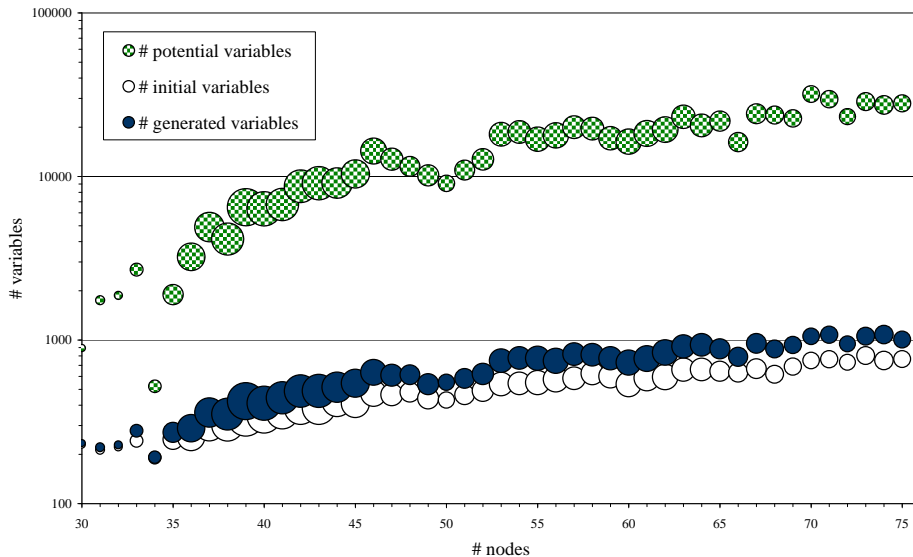


Fig. 10. The number of variables by the number of nodes for the following three categories: potential variables; those generated when optimality was established; those generated at the beginning of the computation.

depends on the crossing number of the graph. The gentle slope in Figure 12 is mainly due to the fact that the larger graphs tend to have higher crossing numbers. The potentially interesting data points referring to the minimum time needed for the non-trivial class all turn out to be constantly under one second. Notice that the apparent convergence between the maximum and the minimum for the graphs with crossing number 12 is due to the fact that only four such graphs could be solved with proven optimality.

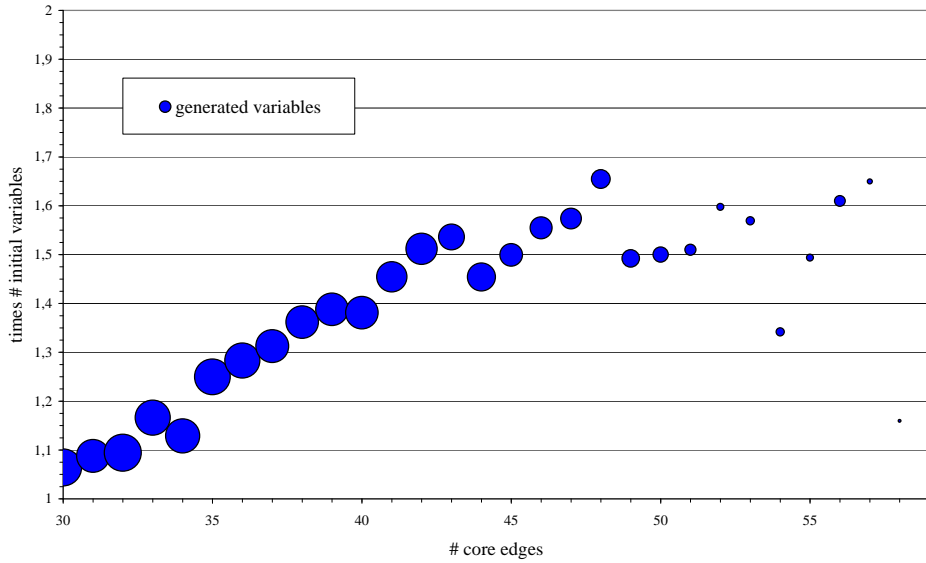


Fig. 11. The number of variables for all successfully optimized test instances, relative to the number of variables in the initial variable set.

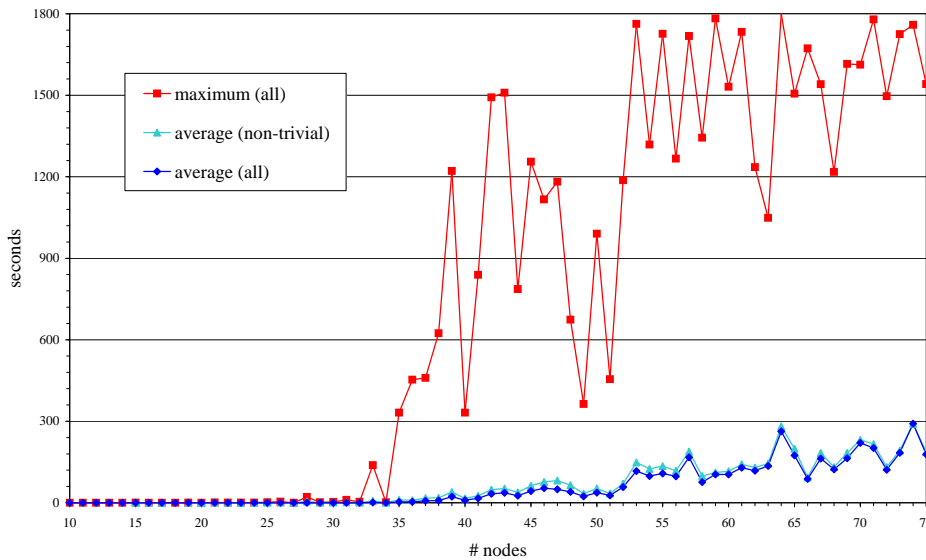


Fig. 12. The running time of all test instances solved to provable optimality relative to the number of nodes.

In the last statistics (Figure 14), we show the distribution of the final upper and lower bound obtained after the 30 minutes time limit for all graphs. The main diagonal shows the number of graphs solved to proven optimality. As we can see, we could prove the crossing number for all graphs with up to 5 crossings. For scaling purposes, the first diagram does not show two data points corresponding to a single graph each. They are the lower-bound/upper-bound pairs 16/51 and 20/62.

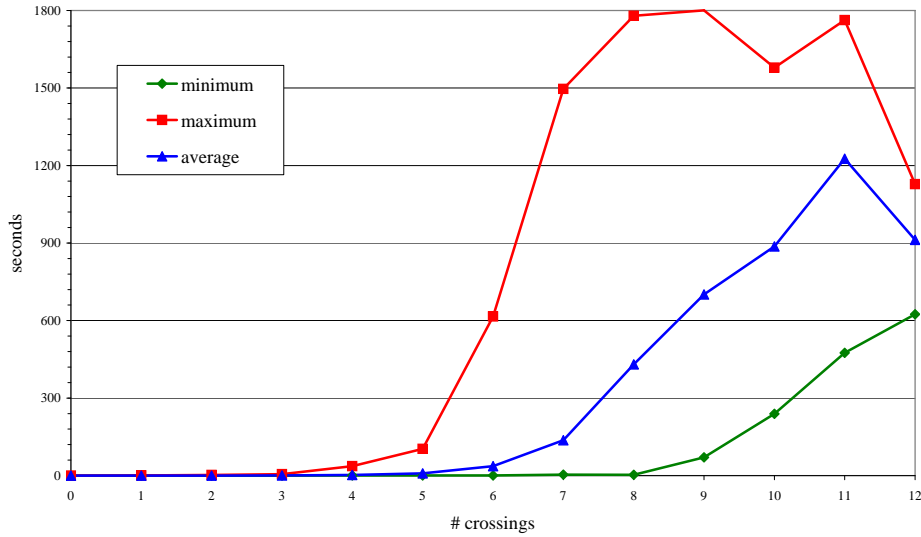


Fig. 13. The running time of all test instances solved to provable optimality relative to the crossing number.

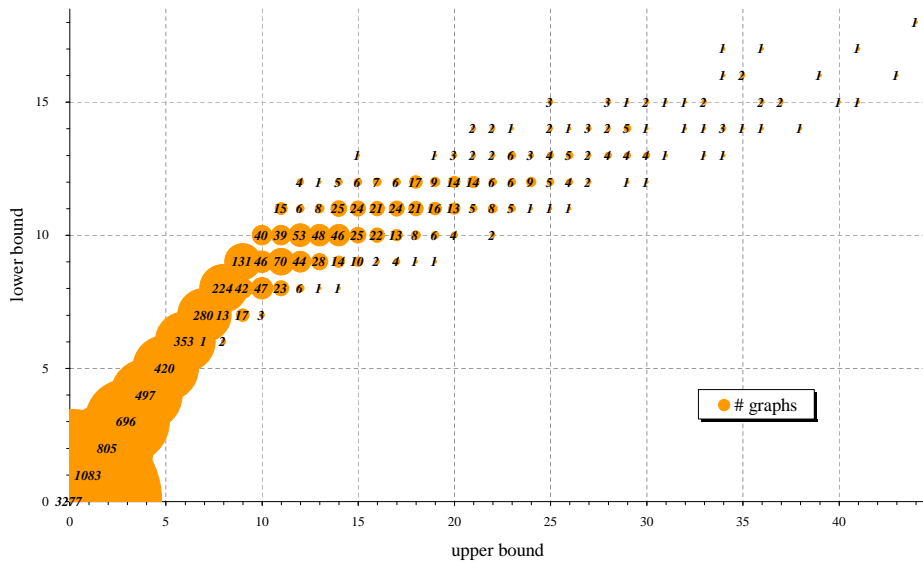


Fig. 14. Final lower compared to final upper bound over all graphs. The number at the data points gives the number of graphs falling into the corresponding category.

We also applied our algorithm to complete graphs K_n . By this, we managed to verify the crossing number conjecture up to $n = 8$. In view of Figure 13, this is very surprising, as the number of crossings is 18 for K_8 . The resulting planarizations are different from those generated by Zarankiewicz's rule. We are convinced that we will be able to compute the crossing numbers of much larger complete graphs using specialized versions of our algorithm, e.g., by using the knowledge of $cr(K_{n-1})$ in order to add stronger inequalities for K_n .

6 Conclusion

We presented the first algorithm for computing the crossing number and a corresponding planarization for small to medium-sized graphs. Our system combines a sophisticated problem-reduction algorithm using SPQR-trees with a branch-and-cut approach that reduces the general crossing number problem to the restricted version where each edge is crossed at most once. We introduced a column generation scheme that reduces the number of necessary variables tremendously and thus makes larger instances solvable. Our approach works well for benchmark graphs of up to 40–50 core edges and can also solve larger instances provided the crossing number is not too high. In the future, we hope to improve the cutting strategy by a more thorough polyhedral investigation. Moreover, we will try to contribute to open questions concerning the crossing numbers of complete (bipartite) graphs by specializing our approach to these instances.

References

- [1] C. Batini, M. Talamo, and R. Tamassia. Computer aided layout of entity-relationship diagrams. *Journal of Systems and Software*, 4:163–173, 1984.
- [2] 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.
- [3] H. Bodlaender and A. Grigoriev. Algorithms for graphs embeddable with few crossings per edge. Research Memoranda 036, Maastricht : METEOR, Maastricht Research School of Economics of Technology and Organization, 2004.
- [4] 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.
- [5] 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.
- [6] J. Cai, X. Han, and R. E. Tarjan. An $O(m \log n)$ -time algorithm for the maximal planar subgraph problem. *SIAM Journal on Computing*, 22(6):1142–1162, December 1993.
- [7] G. Călinescu, C. G. Fernandes, U. Finkler, and H. Karloff. A better approximation algorithm for finding planar subgraphs. *Journal of Algorithms*, 27(2):269–302, May 1998.
- [8] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.

- [9] H. de Fraysseix and P. O. de Mendez. On cotree-critical and dfs cotree-critical graphs. *Journal of Graph Algorithms and Applications*, 7(4):411–427, 2003.
- [10] 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.
- [11] G. Di Battista and R. Tamassia. On-line maintenance of triconnected components with SPQR-trees. *Algorithmica*, 15:302–318, 1996.
- [12] G. Di Battista and R. Tamassia. On-line planarity testing. *SIAM Journal on Computing*, 25:956–997, 1996.
- [13] H. N. Djidjev. A linear algorithm for the maximal planar subgraph problem. In *Proceedings of the 4th Workshop on Algorithms and Data Structures*, volume 955 of *Lecture Notes in Computer Science*, pages 369–380. Springer-Verlag, 1995.
- [14] P. Eades and N. C. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11(4):379–403, 1994.
- [15] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, NY, 1979.
- [16] M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 4:312–316, 1983.
- [17] M. Grohe. Computing crossing numbers in quadratic time. In *STOC 2001: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, 2001.
- [18] C. Gutwenger and M. Chimani. Non-planar core reduction of graphs. In P. Eades and P. Healy, editors, *Graph Drawing 2005*, Lecture Notes in Computer Science. Springer-Verlag, 2005. To appear.
- [19] C. Gutwenger and P. Mutzel. A linear time implementation of SPQR trees. In J. Marks, editor, *Graph Drawing 2000*, volume 1984 of *Lecture Notes in Computer Science*, pages 77–90. Springer-Verlag, 2001.
- [20] C. Gutwenger and P. Mutzel. An experimental study of crossing minimization heuristics. In G. Liotta, editor, *Graph Drawing 2003*, volume 2912 of *Lecture Notes in Computer Science*, pages 13–24. Springer-Verlag, 2004.
- [21] C. Gutwenger, P. Mutzel, and R. Weiskircher. Inserting an edge into a planar graph. *Algorithmica*, 41(4):289–308, 2005.
- [22] R. K. Guy. Crossing numbers of graphs. In *Graph Theory and Applications (Proceedings)*, Lecture Notes in Mathematics, pages 111–124. Springer-Verlag, 1972.
- [23] J. E. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM Journal on Computing*, 2(3):135–158, 1973.

- [24] J. E. Hopcroft and R. E. Tarjan. Efficient planarity testing. *Journal of the ACM*, 21(4):549–568, 1974.
- [25] W.-L. Hsu. A linear time algorithm for finding a maximal planar subgraph based on PC-trees. In *COCOON: Annual International Conference on Computing and Combinatorics*, pages 787–797, 2005.
- [26] R. Jayakumar, K. Thulasiraman, and M. N. S. Swamy. $O(n^2)$ algorithms for graph planarization. *IEEE Transactions on Computer-Aided Design*, 8:257–267, 1989.
- [27] M. Jünger, S. Leipert, and P. Mutzel. A note on computing a maximal planar subgraph using PQ-trees. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(7):609–612, 1998.
- [28] M. Jünger and P. Mutzel. Maximum planar subgraphs and nice embeddings: Practical layout tools. *Algorithmica*, 16(1):33–59, 1996.
- [29] 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.
- [30] 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*. To appear.
- [31] E. de Klerk, D. V. Pasechnik, and A. Schrijver. Reduction of symmetric semidefinite programs using the regular *-representation. *Mathematical Programming*. To appear.
- [32] J. Kratochvíl. String graphs. II.: Recognizing string graphs is NP-hard. *Journal of Combinatorial Theory, Series B*, 52(1):67–78, 1991.
- [33] C. Kuratowski. Sur le problème des courbes gauches en topologie. *Fundamenta Mathematicae*, 15:271–283, 1930.
- [34] J. A. La Poutré. Alpha-algorithms for incremental planarity testing. In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing*, pages 706–715, 1994.
- [35] P. Liu and R. Geldmacher. On the deletion of nonplanar edges of a graph. In *Proceedings of the 10th Southeastern Conference on Combinatorics, Graph Theory, and Computing*, pages 727–738, Boca Raton, FL, 1977.
- [36] S. Masuda, T. Kashiwabara, K. Nakajima, and T. Fujisawa. On the NP-completeness of a computer network layout problem. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 292–295, 1987.
- [37] S. Masuda, K. Nakajima, T. Kashiwabara, and T. Fujisawa. Crossing minimization in linear embeddings of graphs. *IEEE Transactions on Computers*, 39(1):124–127, 1990.

- [38] P. Mutzel and T. Ziegler. The constrained crossing minimization problem. In J. Kratochvil, editor, *Graph Drawing '99*, volume 1731 of *Lecture Notes in Computer Science*, pages 175–185. Springer-Verlag, 1999.
- [39] OGDF – Open Graph Drawing Framework. University of Dortmund, Chair of Algorithm Engineering and Systems Analysis. Web site under construction.
- [40] J. Pach and G. Tóth. Graphs drawn with few crossings per edge. *Combinatorica*, 17(3):427–439, 1997.
- [41] H. C. Purchase. Which aesthetic has the greatest effect on human understanding? In G. Di Battista, editor, *Graph Drawing '97*, volume 1353 of *Lecture Notes in Computer Science*, pages 248–261. Springer-Verlag, 1997.
- [42] P. Turán. A note of welcome. *Journal of Graph Theory*, 1:7–9, 1977.
- [43] 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.