# Configurations with Few Crossings in Topological Graphs

Christian Knauer[1], Étienne Schramm[2⋆],
Andreas Spillner[3], and Alexander Wolff[2⋆]

[1] Institute of Computer Science, Freie Universität Berlin.
Email: christian.knauer@inf.fu-berlin.de
[2] Fakultät für Informatik, Universität Karlsruhe, P.O. Box 6980, D-76128 Karlsruhe.
WWW: i11www.ira.uka.de/algo/group
[3] Institute of Computer Science, Friedrich-Schiller-Universität Jena.
Email: spillner@minet.uni-jena.de

**Abstract.** In this paper we study the problem of computing subgraphs of a certain configuration in a given topological graph $G$ such that the number of crossings in the subgraph is minimum. The configurations that we consider are spanning trees, $s$–$t$ paths, cycles, matchings, and $\kappa$-factors for $\kappa \in \{1, 2\}$. We show that it is NP-hard to approximate the minimum number of crossings for these configurations within a factor of $k^{1-\varepsilon}$ for any $\varepsilon > 0$, where $k$ is the number of crossings in $G$. We then show that the problems are fixed-parameter tractable if we use the number of crossings in the given graph as the parameter. Finally we present a simple but effective heuristic for spanning trees.

## 1 Introduction

An undirected graph $G(V, E)$ that is embedded in the plane such that no two edges share an unbounded number of points is called a *topological graph*. If all edges are straight-line embedded, then $G$ is called a *geometric* graph. A *crossing* $\{e, e'\}$ is a pair of edges in $G$ such that $e \cap e' \not\subseteq V$. We call $\mu_{ee'} = |(e \cap e') \setminus V|$ the *multiplicity* of the crossing $\{e, e'\}$. Note that $\mu \equiv 1$ for geometric graphs. Let $X \subseteq \binom{E}{2}$ be the set of pairs of crossings in $E$. Note that $c$ edges intersecting in a single non-endpoint give rise to $\binom{c}{2}$ crossings. We will use $n$, $m$, and $k$ as shorthand for the cardinalities of $V$, $E$, and $X$, respectively. We define the *weighted number of crossings* of $G$ as $\sum_{\{e,e'\}\in X} \mu_{ee'}$.

In this paper we study the problem of computing subgraphs of a certain configuration in a given topological graph such that the weighted number of crossings in the subgraph is minimum. The configurations that we consider are spanning trees, $s$–$t$ paths, cycles, matchings, and $\kappa$-factors, i.e. subgraphs in which every node $v \in V$ has degree $\kappa$, for $\kappa \in \{1, 2\}$. In the version of matching that we consider the number $M$ of desired matching edges is part of the input. We will refer to this version as *$M$-matching*.

⋆ Supported by grant WO 758/4-2 of the German Science Foundation (DFG).

Algorithms that find subgraphs with few crossings have applications in VLSI design and pattern recognition [3]. For example, a set of processors (nodes) on a chip and a number of possible wire connections (edges) between the processors induce a topological graph $G$. A spanning tree in $G$ with few crossings connects all processors to each other and can help to find a wire layout that uses few layers, thus reducing the chip's cost.

There is also a connection to matching with geometric objects. Rendl and Woeginger [7] have investigated the following problem. Given a set of $2n$ points in the plane they want to decide whether there is a perfect matching (i.e. a 1-factor) where the matched points are connected by axis-parallel line segments. They give an $O(n \log n)$-time algorithm for this problem and show that the problem becomes NP-hard if the line segments are not allowed to cross.

Kratochvíl et al. [5] have shown that for topological graphs it is NP-hard to decide whether they contain a crossing-free subgraph for any of the configurations mentioned above. Later Jansen and Woeginger [3] have shown that for spanning trees, 1- and 2-factors the same even holds in geometric graphs with just two different edge lengths or with just two different edge slopes.

These results do not rule out the existence of efficient constant-factor approximation algorithms. However, as we will show in Section 2, such algorithms do not exist unless $\mathcal{P} = \mathcal{NP}$. In Section 3 we complement these findings by a simple polynomial-time factor-$(k - c)$ approximation for any constant integer $c$. This result being far from satisfactory, we turn our attention to other possible ways to attack the problems: in Section 4 we show that the problems under consideration are fixed-parameter tractable with $k$ being the parameter. While there are simple algorithms that show tractability, it is not at all obvious how to improve them. It was a special challenge to beat the $2^k$-term in the running time of the simple fixed-parameter algorithm for deciding the existence of a crossing-free spanning tree. We also give optimization algorithms.

Finally, in Section 5 we give a simple heuristic for computing spanning trees with few crossings. Due to our findings in Section 2 our heuristic is unlikely to have a constant approximation factor. However, it performs amazingly well, both on random examples and on real-world instances. We use a mixed-integer programming (MIP) formulation (see [4]) as baseline for our evaluation.

Our fixed-parameter algorithm and the MIP formulation for the 1-factor problem can both be used to solve the above-mentioned problem of Rendl and Woeginger [7]. In our MIP formulation the numbers of variables and constraints depends only linearly on $k$. This makes the MIP formulation superior to the FPT algorithm for large values of $k$. Neither of our exact methods exploits the geometry of the embedded graph. Thus they also work if we are given an abstract graph $G$ and a set $X$ of crossings, and we view a crossing simply as a set of two edges not supposed to be in the solution at the same time.

In the whole paper we assume that the set of crossings $X$ in the input graph has already been computed. Depending on the type of curves representing the graph edges this can be done using standard algorithms [2]. Whenever we want to stress that $X$ is given, we use the notation $G(V, E, X)$.

## 2 Hardness of Approximation

For each of the configurations mentioned in the introduction we now show that it is hard to approximate the problem of finding subgraphs of that configuration with the minimum number of crossings in a given geometric graph $G$. The reductions are simple and most of them follow the same idea.

We begin with the problem of finding a spanning tree with as few crossings as possible. We already know [3] that the problem of deciding whether or not $G$ has a crossing-free spanning tree is NP-hard. Our reduction employs this result directly. Given a graph $G$ with $k$ crossings and a positive integer $d$, we build a new graph $G'$ by arranging $k^d$ copies of $G$ along a horizontal line and by then connecting consecutive copies by a single edge as in Figure 1. The new graph $G'$ has $k^{d+1}$ crossings. Now if $G$ has a crossing-free spanning tree then $G'$ has a crossing-free spanning tree. Otherwise every spanning tree in $G'$ has at least $k^d$ crossings. Let $\phi(G)$ be 1 plus the minimum number of crossings in a spanning tree of $G$. Since we can choose $d$ arbitrarily large we have the following theorem. All theorems in this section hold for any $\varepsilon \in (0, 1]$.

**Theorem 1.** *It is NP-hard to approximate $\phi(G)$ within a factor of $k^{1-\varepsilon}$.*

We also consider a kind of dual optimization problem: Find a crossing-free spanning forest in $G$ with as few trees as possible. Let $\phi'(G)$ denote the minimum number of trees in a spanning forest of $G$. Since there is a spanning forest with one tree if and only if there is a crossing-free spanning tree in $G$, we immediately have the following theorem.

**Theorem 2.** *It is NP-hard to approximate $\phi'(G)$ within a factor of $k^{1-\varepsilon}$.*

Next let us briefly consider the problems of finding $M$-matchings, 1- and 2-factors in $G$ with as few crossings as possible. Again we already know that the related decision problems are NP-hard [3]. Let $\eta(G)$ denote 1 plus the minimum number of crossings in a subgraph of $G$ of the desired configuration. Arguing along the same lines as for spanning trees we obtain the following theorem. Note that in this reduction we do *not* connect the copies of $G$ in $G'$.

**Theorem 3.** *It is NP-hard to approximate $\eta(G)$ within a factor of $k^{1-\varepsilon}$.*

Now we turn to the problem of finding a path between two given vertices $s$ and $t$ (an $s$–$t$ path for short) with as few crossings as possible. We can show that the problem of deciding whether or not a given geometric graph has a crossing-free $s$–$t$ path is NP-hard by a simple adaption of the reduction from planar 3SAT presented in [5] for topological graphs. Figure 2 reproduces Figure 7 from [5]. Every clause is represented by a triple of edges between two vertices. Each edge in such a triple represents a variable occurring in the corresponding clause. It is shown in [5] that it is possible to draw the edges in such a way that occurrences of variables that cannot be set true simultaneously correspond to edges that intersect. Thus there is a satisfying truth assignment if and only if there is a

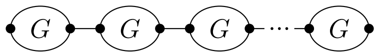



**Fig. 1.** Graph $G'$: $k^d$ copies of $G$.

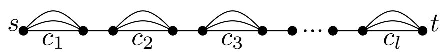



**Fig. 2.** The basis of the reduction in [5].

crossing-free $s$–$t$ path in the constructed graph. It is not hard to see that we can substitute a sequence of straight line segments for the drawing of every edge in the topological graph.

Now we apply the same trick as in the case of spanning trees to turn the NP-hardness of the decision problem into a hardness-of-approximation result. This is indicated in Figure 3. If there is a crossing-free $s$–$t$ path in $G$ then there is a crossing-free $s_1$–$t_{k^d}$ path in $G'$. If there is at least one crossing in every $s$–$t$ path in $G$ then there are at least $k^d$ crossings in every $s_1$–$t_{k^d}$ path in $G'$. Let $\gamma(G)$ denote 1 plus the minimum number of crossings in a $s$–$t$ path in $G$. Then we have the following theorem.

**Theorem 4.** *It is NP-hard to approximate $\gamma(G)$ within a factor of $k^{1-\varepsilon}$.*

In [5] it is shown that it is even possible to draw the edges of the graph in Figure 2 in such a way that in addition to the crossings which ensure the consistency of the chosen truth setting we can make the edges in every clause pairwise intersecting. Thus we can choose at most one edge in every clause gadget and the constructed graph does not contain a crossing-free cycle. Now we connect vertices $s$ and $t$ by an extra sequence of edges as indicated in Figure 4 and easily obtain the following corollary.

**Corollary 1.** *It is NP-hard to decide whether or not a geometric graph contains a crossing-free cycle.*

Unfortunately it seems impossible to apply the same trick again to obtain the hardness of approximation, since there may be cycles that do not pass through vertices $s$ and $t$ and that have only few crossings. Thus we have to punish the usage of a crossing in forming a cycle in the graph. To achieve this goal for every crossing in the constructed graph we make the sequences of straight line edges cross many times. This is indicated in Figure 5. By choosing the number of bends large enough we obtain the following theorem where $\zeta(G)$ denotes 1 plus the minimum number of crossings in a cycle in $G$.

**Theorem 5.** *It is NP-hard to approximate $\zeta(G)$ within a factor of $k^{1-\varepsilon}$.*

## 3 Approximation Algorithms

After this long list of negative results on approximability let us now give a positive remark. Trivially, any spanning tree in a geometric graph with $k$ crossings $(k+1)$-approximates $\phi(G)$. However, with just a little more effort we can compute a factor-$k$ approximation by checking whether all edges in $\bigcup X$ are cut

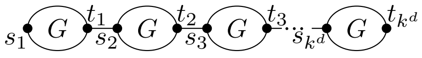



**Fig. 3.** The graph $G'$.

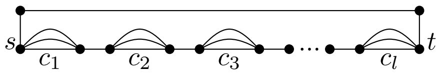



**Fig. 4.** Adding edges between $s$ and $t$.

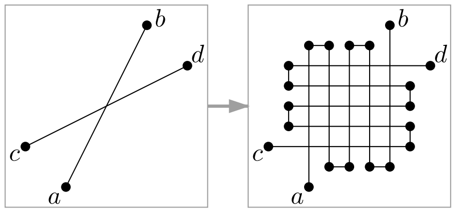



**Fig. 5.** Punishing the usage of crossings.

edges. If yes, then *every* spanning tree of $G$ has $k$ crossings (and hence is optimal). Otherwise we can compute a spanning tree that avoids one of the non-cut edges, which yields a factor-$k$ approximation. Along the same lines we obtain factor-$(k - c)$ approximations for every constant $c > 0$ in polynomial time.

**Theorem 6.** *For every constant integer $c \in (0, k)$ there is a polynomial-time factor-$(k - c)$ approximation for $\phi(G)$.*

# 4 Fixed-Parameter Algorithms

In this section we present fixed-parameter algorithms using the total number $k$ of crossings as the parameter. The intuition behind the concept of fixed-parameter algorithms [1] is to find a quantity associated with the input such that the problem can be solved efficiently if this quantity is small. The number $k$ suggests itself naturally since on the one hand the problems under consideration become trivial if $k = 0$ and on the other hand the reductions in Section 2 employ graphs with many crossings.

## 4.1 A Simple General Approach

We assume that the input graph $G$ has a subgraph of the desired configuration and we only try to find one with the minimum weighted number of crossings. For example, when looking for spanning trees we assume that the input graph is connected. We set $E_X = \bigcup X$. Thus $E_X$ contains exactly those edges that participate in a crossing. Note that $|E_X| \leq 2k$. Now we can proceed as follows:

1. Form the crossing-free graph $G'$ by removing all edges in $E_X$ from $G$.
2. For all crossing-free subsets $H \subseteq E_X$ check whether the graph $G' \cup H$ has a subgraph of the desired configuration.

The graph $G'$ can be constructed in $O(m)$ time. Let $\text{check}_\mathcal{C}(n, m)$ be the time needed for checking whether $G' \cup H$ has a subgraph of configuration $\mathcal{C}$. Since $\text{check}_\mathcal{C}(n, m) = \text{poly}(n, m)$ for all the configurations we consider, the two-step procedure shows that the corresponding decision problems can be solved in $O(m + \text{check}_\mathcal{C}(n, m)\, 4^k)$ time and thus are all fixed-parameter tractable. However, it is easy to do better.

**Observation 1** *To check the existence of a crossing-free configuration in $G$ it suffices to go through all maximal (w.r.t. the subgraph relation) crossing-free subgraphs of $G$ and check whether one of them has a subgraph of the desired configuration.*

By induction on $k$ we get that $E_X$ has at most $2^k$ maximal crossing-free subsets $H$. We perform step 2 only on these.

**Theorem 7.** *Given a topological graph $G(V, E, X)$ and a configuration $\mathcal{C}$, we can decide in $O(m+\text{check}_{\mathcal{C}}(n, m)\, 2^k)$ time whether $G$ has a crossing-free subgraph of configuration $\mathcal{C}$.*

If the desired configuration $\mathcal{C}$ is an $M$-matching we have $\text{check}_{\mathcal{C}}(n, m) \in O(\sqrt{n}m)$ [6, 9]. Note that 1-factors are only a special kind of $M$-matching. For 2-factors we can employ the graph transformation of Tutte [8] and obtain $\text{check}_{\mathcal{C}}(n, m) \in O(n^4)$.

Observe that in step 2 the only interesting connected components of $G'$ are those that contain an endpoint of an edge from $E_X$. However, there are at most $4k$ such connected components. For the configurations spanning tree, $s$–$t$ path and cycle this observation yields a reduction to a *problem kernel* [1], i.e. to a problem whose size depends only on the parameter $k$, but not on the size of the input. Now it is clear that for these configurations generating and checking a subset $H$ can be done in $O(k)$ time.

**Corollary 2.** *Given a topological graph $G(V, E, X)$, we can decide in $O(m + k2^k)$ time whether $G$ has a crossing-free spanning tree, $s$–$t$ path or cycle.*

Finally we want to present a simple approach to deal with the corresponding optimization problems, i.e. the problem of finding a desired configuration with minimum weighted number of crossings. For every subset $X'$ of the set of crossings $X$ we do the following.

1. Compute the graph $G'' = G' \cup \bigcup X'$.
2. Compute the subset of crossings $X''$ from $X$ that do not share an edge with any crossing in $X'$.
3. Decide whether there is a crossing-free subset $H$ of $\bigcup X''$ such that $G'' \cup H$ contains a desired configuration.

We filter out those subsets of crossings $X'$ for which we get a positive answer in step 3. Among the filtered out subsets we can easily keep track of the one which yields a minimum weighted number of crossings. Suppose in step 3 we use a decision algorithm running in $O(\text{check}_{\mathcal{C}}(n, m)\beta^k)$ time.

**Theorem 8.** *Given a topological graph $G(V, E, X)$, we can compute in $O(m + \sum_{j=0}^{k} \binom{k}{j} \text{check}_{\mathcal{C}}(n, m)\beta^{k-j}) = O(m + \text{check}_{\mathcal{C}}(n, m)(1 + \beta)^k)$ time a subgraph of configuration $\mathcal{C}$ in $G$ with minimum weighted number of crossings.*

### 4.2 Spanning Trees

In this section we want to improve the $2^k$-term in the running time of the simple decision algorithm. It will turn out that we barely achieve this goal. We will get the exponential term in the running time down to $1.9999996^k$. While this improvement seems marginal, the fact that we managed to beat the trivial algorithm is of theoretical interest. Moreover, we think that our methods can be applied in a wider scenario. A similar approach for $s$–$t$ paths and cycles yields $1.733^k$, see the long version of this article [4].

Imagine the process of selecting the edges for set $H$ as a search tree. Branchings in the tree correspond to possible choices during the selection process. By selecting edges in $E_X$ to be in $H$ or not to be in $H$ we reduce the number of crossings from which we can still select edges. The leaves of the search tree correspond to particular choices of $H$. Let $T(k)$ denote the maximum number of leaves in the search tree for input graphs with $k$ crossings. Note that $T(k)$ also bounds the number of interior nodes of the search tree.

First we will see that it is rather easy to speed up the algorithm as long as the crossings in $G$ are not pairwise disjoint. Let $e$ be an edge such that exactly $z$ crossings $c_1 = \{e, f_1\}, \dots, c_z = \{e, f_z\}$ in $X$ share edge $e$ and $2 \leq z \leq k$. If we select edge $e$ to be in $H$ then none of the edges $f_1, \dots, f_z$ can be in $H$. If we select edge $e$ not to be in $H$ then we can select edges $f_1, \dots, f_z$ as if crossings $c_1, \dots, c_z$ would not exist. Thus for both choices there are only $k - z$ crossings left from which we still can select edges. This leads to the recurrence $T(k) \leq 2T(k - z)$ which solves to $T(k) \in O(2^{k/z})$.

It remains to consider the case that the crossings in $X$ are pairwise disjoint. Up to now we concentrated on the set $E_X$. It was only after selecting an edge $e$ to be in $H$ that we took a look at the connected components of $G'$ that are possibly connected by $e$. Now we also take the connected components of $G'$ into consideration to guide the selection process. Observe that any connected component $C$ (in order to make $G'$ connected) must be connected by at least one edge from $E_X$ to the rest of $G'$. This puts some restriction on which crossing-free subsets of the edges in $E_X$ with an endpoint in $C$ need to be checked. After introducing some notation, Lemma 1 will make this more precise.

We can assume that for every crossing $c \in X$ none of the two edges in $c$ connects vertices in the same connected component of $G'$, since such an edge cannot help to make $G'$ connected and thus need not be selected to be in $H$.

We define the degree of a connected component of $G'$ as the number of edges in $E_X$ with one endpoint in this component. Now consider some connected component $C$ of $G'$. Let $d$ denote the degree of $C$ and $E(C)$ the set of edges in $E_X$ incident to a vertex of $C$. Let $X(C)$ denote the set of crossings contained in $E(C)$. We set $x = |X(C)|$, $R = \bigcup X(C)$ and $S = E(C) \setminus R$. Then $S$ contains the $d - 2x$ edges in $E(C)$ that do not cross any other edge in $E(C)$. To connect $C$ with the rest of $G'$ we select subsets $T$ of $E(C)$ such that $T$ contains exactly one edge from each crossing in $X(C)$ and a subset of the edges in $S$.

**Lemma 1.** *It suffices to check $2^{d-x} - 1$ subsets of $E(C)$.*

For the proof of Lemma 1 refer to [4]. The result leads to the recurrence $T(k) \leq (2^{d-x} - 1)T(k - (d - x))$ which solves to $T(k) \in O((2^{d-x} - 1)^{k/(d-x)})$. Of course, this solution is of little use if we cannot guarantee the existence of a component $C$ with appropriately bounded degree $d$. It turns out that we can do that if there is a constant $\alpha$ $(0 < \alpha < 1)$ such that $G'$ has more than $\alpha k$ connected components: Let $\delta$ denote the minimum degree of a component of $G'$. Then we have $4k \geq 2|E_X| > \delta\alpha k$ and hence $4/\alpha > \delta$. It is desirable to choose $\alpha$ as large as possible. We will use $\alpha = 0.211$ for reasons that will become clear soon. Then we can guarantee the existence of a component with degree at most 18 and obtain $T(k) \in O(\beta_{18}^k)$, where $\beta_i = \sqrt[i]{2^i - 1} < 2$. For each of the at most $T(k)$ nodes and leaves of the search tree, we need $O(k)$ time.

It remains to treat the case that the crossings are pairwise disjoint and $G'$ has at most $\alpha k$ connected components. Set $l = \lfloor \alpha k \rfloor$. Observe that we can make $G'$ connected without crossing edges iff there are $l$ crossings in $X$ such that $G'$ becomes connected by using one edge from each of these $l$ crossings. Thus we simply check every subset of $l$ crossings from $X$, which can be done in $O(\binom{k}{l} 2^l k)$ time. Using $\alpha = 0.211$, this is in $O(k 1.985^k)$.

**Theorem 9.** *Given a topological graph $G(V, E, X)$, we can decide in $O(m + k\beta_{18}^k)$ time whether $G$ has a crossing-free spanning tree ($\beta_{18} < 1.9999996$).*

If we are willing to resort to a randomized algorithm we can improve the result of Theorem 9. We are looking for a new way to treat the case that the crossings are pairwise disjoint and $G'$ has at most $\alpha k$ connected components. Observe that if $G$ has a connected crossing-free spanning subgraph at all, then there are at least $2^{(1-\alpha)k}$ maximal crossing-free subsets $H \subseteq E_X$ such that $G' \cup H$ is connected. This can be seen as follows: Suppose there is a crossing-free subset $F \subseteq E_X$ that makes $G'$ connected. Then we can choose such an $F$ with $|F| < \alpha k$. Let $X_F = \{c \in X \mid c \cap F = \emptyset\}$. Since the elements of $X$ are pairwise disjoint we have $|X_F| > (1 - \alpha)k$. If we select one edge from each crossing in $X_F$ and add these edges to $F$ the resulting set of edges is still crossing-free. There are at least $2^{(1-\alpha)k}$ possible ways to select edges. Thus there are at least $2^{(1-\alpha)k}$ maximal crossing-free subsets $H \subseteq E_X$ such that $G' \cup H$ is connected.

This suggests the following randomized algorithm: From each element of $X$ we randomly select one edge and check if the resulting crossing-free graph is connected. If the given geometric graph $G$ has a crossing-free connected spanning subgraph, the probability of success is at least $2^{(1-\alpha)k}/2^k = 2^{-\alpha k}$. Thus $O(2^{\alpha k})$ iterations suffice to guarantee a probability of success greater than $1/2$. The running time is in $O(k 2^{\alpha k})$. Setting $\alpha = 4/5$ yields the following theorem.

**Theorem 10.** *There is a Monte Carlo algorithm with one sided error, probability of success greater than $1/2$ and running time in $O(m + k\beta_4^k)$ which tests whether a given topological graph has a crossing-free spanning tree ($\beta_4 < 1.968$).*

For the dual optimization problem of finding a spanning forest of $G$ consisting of as few trees as possible we can proceed similarly as in the algorithm for the decision problem above. Thus we can solve the dual optimization problem in $O(m + k\beta_{18}^k)$ time.

## 5 Heuristic

Due to our inapproximability results in Section 2, we cannot hope to find a constant-factor approximation for the number of crossings in any of the configurations we consider. Instead, we now describe a simple heuristic for computing spanning trees with few crossings in geometric graphs. Our heuristic uses a set of rules that simplify the input graph without changing the number of crossings of an optimal spanning tree. Initially all edges are *active*. During the process, edges can be deleted or *selected*. The solution will consist of the edges that are selected during the process. The heuristic applies the rules to the input graph until no more rule can be applied. Then a heuristic decision is taken. We decided to delete the edge $e$ that maximizes $A(e) + 3S(e)$, where $A(e)$ and $S(e)$ are the numbers of active and selected edges that $e$ crosses, respectively.

We now specify the rules. They are only applied to active edges. Connected components refer to the graph induced by all nodes and the selected edges.

1. If an edge has no crossings with other edges, it is selected.
2. If an edge is a cut edge, it is selected.
3. If both endpoints of an edge belong to the same connected component, then this edge is deleted.
4. If two edges $e_1$ and $e_2$ connect the same connected components, and if every edge crossed by $e_1$ is also crossed by $e_2$, then $e_2$ is deleted.

Our heuristic always finds a spanning tree since rule 2 makes sure that no cut edge is deleted. A brute-force implementation runs in $O(nm^3)$ time.

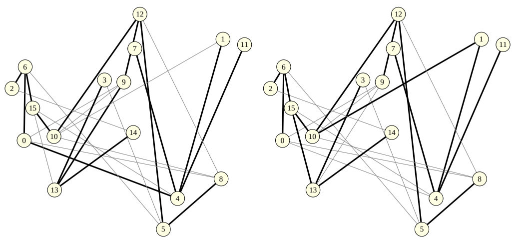
We have implemented the heuristic (except for rule 4) in C++ using the LEDA graph library. It can be tested via a Java applet at `http://i11www.ira.uka.de/few_crossings`. To compute optimal solutions at least for small graphs, we also implemented the MIP formulation described in [4]. We used the MIP solver *Xpress-Optimizer* (2004) by Dash Optimization with the C++ interface of the BCL library. Both heuristic and MIP were run on an AMD Athlon machine with 2.6 GHz and 512 MB RAM under Linux-2.4.20.

We generated random graphs with 20 nodes and $24, 26, \ldots, 80$ edges as follows. Edges were drawn randomly until the desired graph size was obtained. The edge set was discarded if it was not connected. Vertex coordinates were chosen uniformly from the unit square. To these graphs we applied our heuristic and the MIP solver. Figure 6 shows spanning trees of a random graph with 16 vertices and 26 edges. Figure 7 shows the average number of crossings of the spanning trees found by the heuristic and the MIP solver, as well as the number of crossings in the input graph. For each data point, we generated 30 graphs. The average was taken only over those which were solved by the MIP solver within three hours (at least 27 of the 30 graphs per data point).

As real-world data we used three graphs whose vertices correspond to airports and whose edges correspond to direct flight connections in either direction. The flight-connection graphs had each very few high-degree nodes and many leaves. We used the Mercator projection for planarization and then embedded the edges straight-line. The results are given in Table 1.

| Data set | | | | Heuristic | | MIP | |
|---|---|---|---|---|---|---|---|
| | nodes | edges | crossings | crossings | time [sec.] | crossings | time [sec.] |
| Lufthansa Europe | 68 | 283 | 1760 | 66 | 0.2 | 66 | 304.1 |
| Air Canada | 77 | 276 | 1020 | 83 | 0.1 | 83 | 379.7 |
| Lufthansa World | 163 | 696 | 8684 | 128 | 1.8 | 121 | 59.4 |

**Table 1.** Number of crossings of spanning trees in airline graphs.



Heuristic: 5 crossings MIP: 4 crossings

**Fig. 6.** Solutions for a 16-node random graph.

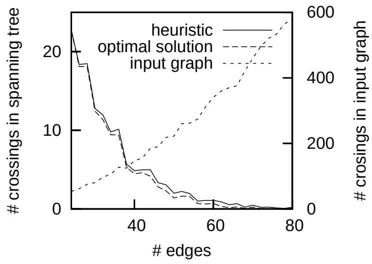



**Fig. 7.** Performance of heuristic and MIP on 20-node random graphs.

Given our inapproximability results in Section 2 we were surprised to see how well our simple heuristic performs: in 77 % of the random graphs and in two of the three real-world instances the heuristic performed optimally. For random graphs it used at most five edge crossings above optimal.

**Acknowledgments.** We thank Jiong Guo, Georg Kliewer, and Markus Völker.

## References

1. R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer, 1999.
2. D. Halperin. Arrangements. In *Handbook of Discrete and Computational Geometry*, chapter 24, pages 529–562. CRC Press, 2004.
3. K. Jansen and G. J. Woeginger. The complexity of detecting crossingfree configurations in the plane. *BIT*, 33:580–595, 1993.
4. C. Knauer, É. Schramm, A. Spillner, and A. Wolff. Configurations with few crossings in topological graphs. Techical Report 2005-24, Universität Karlsruhe, Sept. 2005. `http://www.ubka.uni-karlsruhe.de/cgi-bin/psview?document=/ira/2005/24`.
5. J. Kratochvíl, A. Lubiw, and J. Nešetřil. Noncrossing subgraphs in topological layouts. *SIAM J. Disc. Math.*, 4(2):223–244, 1991.
6. S. Micali and V. V. Vazirani. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *Proc. IEEE Symp. Found. Comp. Sci.*, pp. 17–27, 1980.
7. F. Rendl and G. Woeginger. Reconstructing sets of orthogonal line segments in the plane. *Discrete Mathematics*, 119:167–174, 1993.
8. W. T. Tutte. A short proof of the factor theorem for finite graphs. *Canad. J. Math.*, 6:347–352, 1954.
9. V. V. Vazirani. A theory of alternating paths and blossoms for proving correctness of the $O(|V|^{1/2}|E|)$ general graph matching algorithm. *Combinatorica*, 14:71–91, 1994.