Configurations with Few Crossings in Topological Graphs*

Christian Knauer† Étienne Schramm‡ Andreas Spillner§
Alexander Wolff‡

Submitted to CGTA on September 8, 2005
Revised on February 23, 2006

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 κ-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 mixed-integer linear program formulation for each problem and 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

*This paper is based on two preliminary versions [Ch. Knauer, É. Schramm, A. Spillner, A. Wolff, Configurations with Few Crossings in Topological Graphs, in: X. Deng and D.-Z. Du (Eds.), Proc. 16th Annual International Symposium on Algorithms and Computation (ISAAC'05), in: Lecture Notes in Computer Science, vol. 3827, Springer-Verlag, Berlin, 2005, pp. 604–613; Ch. Knauer, É. Schramm, A. Spillner, A. Wolff, Spanning Trees with Few Crossings in Geometric and Topological Graphs, in: Proc. 21st European Workshop on Computational Geometry (EWCG'05), Eindhoven, 2005, pp. 195–198]

†Institute of Computer Science, Freie Universität Berlin. Email: `christian.knauer@inf.fu-berlin.de`

‡Fakultät für Informatik, Universität Karlsruhe, P.O. Box 6980, D-76128 Karlsruhe, Germany. Supported by grant WO 758/4-2 of the German Research Foundation (DFG). WWW: `http://i11www.ira.uka.de/group`

§Institute of Computer Science, Friedrich-Schiller-Universität Jena. Email: `spillner@minet.uni-jena.de` 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 κ-factors, i.e. subgraphs in which every node $v \in V$ has degree κ, 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.*

Note that in our case the embedding of the graph is given and fixed. We do not try to find an embedding such that the number of edge crossings is small. Recently Grigoriev and Bodlaender [2] considered graphs that have an embedding where each edge has a bounded number of crossings. They showed that many optimization problems (like maximum independent set) admit polynomial-time approximation schemes when restricted to such graphs. However, they also showed that it is NP-hard to decide whether a graph has an embedding in the plane with at most one crossing per edge.

Algorithms that find subgraphs with few crossings have applications in VLSI design and pattern recognition [4]. 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 [8] have investigated the problem of reconstructing sets of axis-parallel line segments. 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 [4] 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. Based on this decision algorithm and those for the other configurations, we also give optimization algorithms. In Section 5 we present mixed-integer linear program (MIP) formulations. Both of the approaches in Sections 4 and 5 yield exact solutions, but in general need exponential time. They can be used to solve the above-mentioned problem of Rendl and Woeginger [8]. In our MIP formulations the numbers of variables and constraints depend only linearly on k. This makes the MIP formulations an interesting alternative to the fixed-parameter algorithms of Section 4 for larger values of k.

Finally, in Section 6 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 the corresponding MIP as baseline for our evaluation.

Our fixed-parameter algorithms and the MIP formulations do not exploit the geometry of the embedded graph. Thus they also work in a setting where 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 [3]. 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 [4] 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. Then $\phi(G') = 1$ or $\phi(G') \geq k^d + 1$. Our construction yields the following result. 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 $k^{1-\varepsilon}$.*

Proof. Assume there exists a real $\varepsilon \in (0, 1]$ such that we can compute a factor-$k^{1-\varepsilon}$ approximation in polynomial time. We choose the number of copies d in the construction of G' outlined above such that $d \geq (1-\varepsilon)/\varepsilon$. Then by our assumption we can compute in polynomial time a spanning tree of G' that approximates $\phi(G')$ within a factor of

$$\left(k^{d+1}\right)^{1-\varepsilon} = k^{d+1} \cdot k^{-\varepsilon(d+1)} \leq k^{d+1} \cdot k^{-\varepsilon\left[\frac{1-\varepsilon}{\varepsilon}+1\right]} = k^{d+1} \cdot k^{-1} = k^d.$$

But then we could decide in polynomial time whether there is a crossing-free spanning tree in G. This would contradict the hardness result in [4]. $\square$

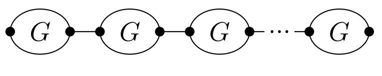


Fig. 1: Graph G': k^d copies of G.

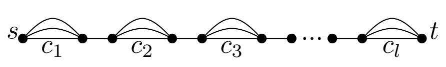


Fig. 2: The basis of the reduction in [5].

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 [4]. 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 show that the problem of deciding whether or not a given *geometric* graph has a crossing-free s–t path is NP-hard by adapting the hardness proof for *topological* graphs of Kratochvíl et al. [5]. Their proof is by reduction from planar 3SAT. Figure 2 reproduces Figure 7 from [5]. Each clause c_i is represented by three edges between the same two vertices. Each edge in such a triplet represents a variable that occurs in c_i. Kratochvíl et al. [5] show that it is possible to draw the edges that correspond to a variable v such that two of these edges cross if and only if they correspond to a negative and a positive occurrence of v. Thus there is a satisfying truth assignment if and only if there is a crossing-free s–t path in the constructed graph.

Our point here is not to go through the whole reduction presented in [5]. We just outline how to modify the reduction such that the result is not merely a topological but a geometric graph. Consider edges in the clause gadgets that correspond to the same variable v. At first glance it seems difficult to make these edges cross in the desired way, i.e. two of them must cross if and only if they correspond to a negative and a positive occurrence of v. The starting point is that each edge in a clause gadget is represented by two straight-line segments as shown in Figure 8 in [5]. The construction in [5] is such that for each variable we can draw a circle that intersects exactly the clause-gadget edges that correspond to the occurrences of this variable. This is indicated in Figure 3 (a). The circle is drawn dotted. The part of the clause-gadget edges inside the circle are the little tips. Tips drawn with bold and thin solid lines correspond to positive and negative occurrences of the variable, respectively. We modify each clause-gadget edge as indicated in Figure 3 (b). Then we replace the gray box in Figure 3 (b) by that in Figure 3 (c). This generates the desired intersection pattern.

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

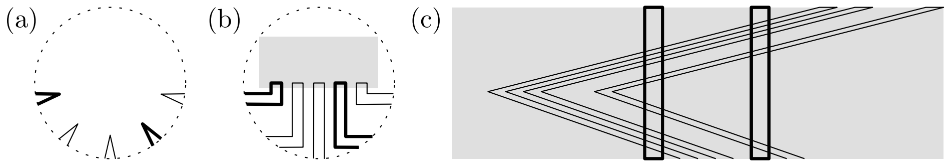


Fig. 3: Details of the reduction.

indicated in Figure 4. 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 an 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 5 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 6. 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 $k^{1-\varepsilon}$.*

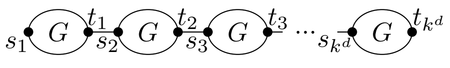


Fig. 4: The graph G'.

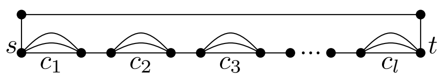


Fig. 5: Adding edges between s and t.

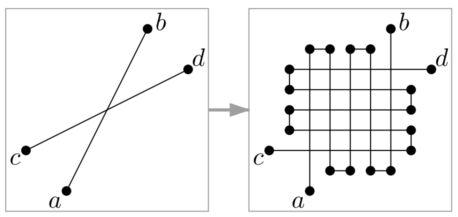


Fig. 6: Punishing the usage of crossings.

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 for every constant $c \in \mathbb{N}$ we can compute a factor-$\max\{1, k-c\}$ approximation. We distinguish two cases.

Case 1: The graph G has at most c crossings. Then we can compute an optimal solution in polynomial time with our fixed-parameter algorithm, see Section 4.

Case 2: The graph G has at least $c+1$ crossings. Then we check for every subset S of $c+1$ crossings from the set of all crossings in G whether there is a way to delete edges participating in the crossings in S such that the resulting subgraph of G is still connected but all the crossings in S have disappeared. If such a set S of $c+1$ crossings exists then we can find a spanning tree of G avoiding all the crossings in S. Such a spanning tree will have at most $k-c-1$ crossings and thus is a factor-$(k-c)$ approximation. If no such set S of $c+1$ crossings exists then every spanning tree of G must have at least $k-c$ crossings and we can find an optimal solution in polynomial time by checking for every subset of at most c crossings from the set of all crossings in G whether there is a spanning tree of G avoiding the crossings in the subset.

Theorem 6 *For every constant $c \in \mathbb{N}$ there is a polynomial-time factor-*$\max\{1, k\text{-}c\}$ *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.*

There are at most 2^k maximal crossing-free subsets of E_X: Let $a(k)$ be an upper bound on the number of maximal crossing-free subsets. Consider an arbitrary edge $e \in E_X$. Let l denote the number of crossings in X that contain e. A maximal crossing-free subset can either contain e or not. Hence, $a(k) \leq 2a(k-l)$. This gives us $a(k) \leq 2^k$. Going through the edges in E_X we can build each of the maximal crossing-free subsets of E_X in $O(k)$ time.

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

If the desired configuration $\mathcal{C}$ is an M-matching, we have $\mathrm{check}_{\mathcal{C}}(n, m) \in O(\sqrt{n}m)$ [7, 10]. Note that a 1-factor is just a special kind of M-matching. For 2-factors we can employ the graph transformation of Tutte [9] and obtain $\mathrm{check}_{\mathcal{C}}(n, m) \in O(n^4)$.

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. Here the problem kernel is the component multigraph $G^\star$ of G' that has a node for each connected component of G'. The edges of $G^\star$ correspond to the edges in E_X. An edge e in E_X is incident to a vertex v of $G^\star$ if and only if e has an endpoint in the connected component represented by v. To decide whether $G' \cup H$ has a spanning tree, s–t path, or cycle for some maximal crossing-free $H \subseteq E_X$, we can do the check (by some variant of breadth-first search) in $G^\star \cup H$. This is faster than doing the check in G' since $G^\star$ has size $O(k)$.

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. Our idea is to check for each subset X' of X whether G has a configuration of the desired type that uses only crossings in X'. This can be done as follows.

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 one that has the minimum weighted number of crossings. Clearly, step 3 dominates the running time of our algorithm. Let $k'' = |X''| \leq k - k'$ and $k' = |X'|$.

Now suppose in step 3 we use a decision algorithm that takes $O(\mathrm{check}_{\mathcal{C}}(n,m)\beta^{k''})$ time after spending $O(m)$ time on preprocessing, then the above procedure takes $O(m+\sum_{k'=0}^{k}\binom{k}{k'}\mathrm{check}_{\mathcal{C}}(n,m)\beta^{k-k'})$ total time. We apply the binomial theorem and conclude as follows.

Theorem 8 *Given a topological graph $G(V,E,X)$, we can compute in $O(m+\mathrm{check}_{\mathcal{C}}(n,m)(1+\beta)^k)$ time a subgraph of configuration $\mathcal{C}$ in G with minimum weighted number of crossings if we can solve the decision problem in $O(\mathrm{check}_{\mathcal{C}}(n,m)\beta^k)$ time after an $O(m)$-time preprocessing.*

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 1.9999992^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 Section 4.3.

Imagine the process of selecting the edges for the set $H \subseteq E_X$ 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 show that it is rather easy to speed up the algorithm of Corollary 2 as long as the crossings in G are not all pairwise disjoint. Let e be an edge such that exactly $z \in \{2,\dots,k\}$ crossings $c_1=\{e,f_1\},\dots,c_z=\{e,f_z\}$ in X share edge e. 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 can still select edges. This leads to the recurrence $T(k) \le 2T(k-z)$ 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 $c \in X$ neither 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. Note that $d=|E(C)|$. 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 consists of 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* $E(C)$.

Proof. First consider the case that $T \subseteq E(C)$ contains at least one edge from S. There are $2^x(2^{d-2x} - 1)$ subsets T of this type.

Next we consider the case that none of the edges in S belongs to T. Suppose there is a maximal crossing-free set H such that $G' \cup H$ is connected and $H \cap S = \emptyset$. Then we consider the graph G'' that is formed by deleting component C from $G' \cup H$. Let r denote the number of components of G''. Since $G' \cup H$ is connected, all the components of G'' are connected in $G' \cup H$ to component C by edges from the crossings in $X(C)$. Since H is maximal crossing-free it contains exactly one edge from each crossing in $X(C)$. Thus the graph G'' has at most x components.

We want to show that there is at least one other maximal crossing-free set $\hat{H}$ such that $G' \cup \hat{H}$ is connected and $\hat{H} \cap S = \emptyset$. Then for each edge set $T \subset R$ which leads to a maximal crossing-free set H that connects G' there would be another edge set $\hat{T} \subset R$ which also leads to a maximal crossing-free set $\hat{H}$ that connects G'. Hence it would suffice to check only $2^x - 1$ subsets T of $E(C)$, one could be omitted. We know $r \leq x$.

If $r < x$, then there are two distinct crossings $c_1 = \{e_1, f_1\}$ and $c_2 = \{e_2, f_2\}$ in $X(C)$ such that edges e_1 and e_2 are in H and connect the same component of G'' with C. Then $\hat{H} = (H \setminus \{e_1\}) \cup \{f_1\}$ is another maximal crossing-free subset of E_X that connects G'.

If $r = x$, then we consider two sub-cases:

If there is a crossing $c = \{e, f\}$ in $X(C)$ such that e is in H and both edges e and f are incident to vertices in the same component of G'', then $\hat{H} = (H \setminus \{e\}) \cup \{f\}$ is another maximal crossing-free subset of E_X that connects G'.

Otherwise, i.e. if in every crossing $c = \{e, f\}$ in $X(C)$ edges e and f are incident to vertices in distinct components of G'', we construct a bipartite graph B with one independent vertex set representing the crossings in $X(C)$ and the other independent vertex set representing the components of G''. A crossing c in $X(C)$ is connected by an edge in B to a component C'' of G'' if and only if there is an edge in c which is incident to a vertex of C''. Thus there is a one-to-one correspondence between the edges in R and the edges of the bipartite graph B. Now it is easy to see that the edges in $H \cap R$ correspond to a perfect matching M in B. Note that the vertices of B which correspond to the crossings in $X(C)$ have all degree two. Hence we can construct another perfect matching $\hat{M}$ in B: From any crossing c_1 we follow the edge of B which is not in the perfect matching and reach a component C''_1 of G''. From C''_1 we follow the edge in the perfect matching to a crossing c_2 and so on. After traversing at most $2x$ edges we will eventually return to crossing c_1. The traversed edges of B form a cycle where edges in the perfect matching and edges not in the perfect matching alternate. Changing the roles of the edges in the cycle gives us the desired perfect matching $\hat{M} \neq M$ and thus a maximal crossing-free edge set $\hat{H} \neq H$.

To summarize the proof: we need only check $2^x(2^{d-2x} - 1) + 2^x - 1 = 2^{d-x} - 1$ subsets of $E(C)$. □

The result of Lemma 1 leads to a new search-tree algorithm. In the beginning of this section we used edges involved in many crossings to guide the branching. In order to cope with the case that all crossings are pairwise disjoint, we use a component of hopefully small degree d to guide the branching. Then Lemma 1 yields the recurrence $T(k) \leq (2^{d-x}-1)T(k-(d-x))$ for the size of the search tree. First observe that for every positive integer l it holds that $(2^l-1)^{1/l} < (2^{l+1}-1)^{1/(l+1)}$ since $(2^l-1) < 2^l < (2+1/(2^l-1))^l$. Now suppose we have an upper bound δ on d. Then $T(k) = (2^\delta-1)^{k/\delta}$ solves the above recurrence. This can be shown by induction on k:

$$\begin{aligned} T(k) &\leq (2^{d-x}-1)\,T(k-(d-x)) \\ &\leq (2^{d-x}-1)(2^\delta-1)^{(k-d+x)/\delta} \\ &= (2^\delta-1)^{k/\delta}\cdot\frac{2^{d-x}-1}{(2^\delta-1)^{(d-x)/\delta}} \\ &= (2^\delta-1)^{k/\delta}\cdot\left[\frac{(2^{d-x}-1)^{1/(d-x)}}{(2^\delta-1)^{1/\delta}}\right]^{d-x} \\ &\leq (2^\delta-1)^{k/\delta}. \end{aligned}$$

Hence, we have $T(k) \in O(\beta_\delta^k)$, where $\beta_\delta = \sqrt[\delta]{2^\delta-1} < 2$.

Now we describe how to guarantee the existence of a component with small degree. The intuition behind this is simple: if G' has many, i.e. more than αk connected components for some $\alpha \in (0,1]$, then by the pigeon-hole principle there must be a component of small degree. Otherwise, if G' has few connected components (i.e. at most αk), then we can afford to enumerate all possibilities to connect G' by a crossing-free edge set H. The choice of α has to balance the running times in the two cases.

We first consider the case that G' has more than αk connected components. Let δ denote the minimum degree of a component of G'. Then we double-count the edge–component incidences. This yields $4k \geq 2|E_X| > \delta\alpha k$, and hence $\delta < 4/\alpha$. In the worst case $\delta = \lfloor 4/\alpha \rfloor$. The resulting search tree has size

$$T_1(k) \in O\big(b_1(\alpha)^k\big)\,,\ \text{where } b_1(\alpha) = \beta_{\lfloor 4/\alpha \rfloor}.$$

We need $O(k)$ time for generating and checking each of the at most $T_1(k)$ nodes and leaves of the search tree. This yields a total running time of $O(m + kT_1(k))$.

Next we consider the case that G' has at most αk connected components. Set $l = \lfloor \alpha k \rfloor$. Observe that we can make G' connected without crossing edges if and only if there are $l-1$ crossings in X such that G' becomes connected by using one edge from each of these $l-1$ crossings. Thus we simply go through each of the $\binom{k}{l-1}$ subsets of $l-1$ crossings from X. For each such subset we enumerate all 2^{l-1} ways to choose one edge per crossing. This results in $T_2(k) < \binom{k}{l}2^l$ subsets H for which we can check in $O(k)$ time each whether $G' \cup H$ is connected. This yields a total running time of $O(m + kT_2(k))$. In order to simplify T_2 we apply Sterling's formula to $\binom{k}{l} = k!/(l!(k-l)!)$. This results in

$$T_2(k) \in O\big(b_2(\alpha)^k\big)\,,\ \text{where } b_2(\alpha) = \frac{2^\alpha}{\alpha^\alpha\cdot(1-\alpha)^{1-\alpha}}.$$

In order to balance the expressions $T_1(k)$ and $T_2(k)$ in the two running times, we determine a value of α that minimizes $\max_{\alpha\in(0,1)}\{b_1(\alpha), b_2(\alpha)\}$. (The graphs of

b_1 and b_2 do not intersect since b_1 is a stair-case function, i.e. not continuous.) The minimum is attained at $\alpha \approx 0.227$. This value of α yields $T_1(k) = O(\beta_{17}^k) = O(1.9999992^k)$ and $T_2(k) = O(1.99965^k)$, so the total running time is dominated by the first case. We sum up our result in the following theorem.

Theorem 9 *Given a topological graph $G(V, E, X)$, we can decide in $O(m + k\beta_{17}^k)$ time whether G has a crossing-free spanning tree, where $\beta_{17} < 1.9999992$.*

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 αk connected components. The following randomized algorithm allows us to choose α larger than in the deterministic algorithm above. 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 topological 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 $T_2'(k) = O(b_2'(\alpha)^k)$ iterations suffice to guarantee a probability of success greater than $1/2$, where $b_2'(\alpha) = 2^\alpha$. This results in a running time of $O(m + kT_2'(k))$ if G' has at most αk connected components. Otherwise we use the deterministic algorithm with running time $O(m + kT_1'(k))$, where $T_1'(k) \in O(b_1(\alpha)^k)$ and $b_1(\alpha) = \beta_{\lfloor 4/\alpha \rfloor}$.

Again we must choose α to balance the two running times. Setting $\alpha = 4/5$ yields $T_1(k) \in O(\beta_4^k) = O(1.968^k)$ and $T_2'(k) \in O(2^{4k/5}) = O(1.742^k)$.

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 ($\beta_4 < 1.968$).*

Before we turn to other configurations, note that for the dual optimization problem of finding a spanning forest of G consisting of as few trees as possible, we only have to find a maximal crossing-free subgraph of G consisting of as few components as possible. So by checking every crossing-free subset of E_X we can easily find one that yields an optimal solution in $O(m + k2^k)$ time. With an argument similar to the one that led to Theorem 9 one can show that it is not even necessary to check *each* crossing-free subset of E_X. This yields the following theorem.

Theorem 11 *Given a topological graph $G(V, E, X)$, we can compute in $O(m+k\beta_{17}^k)$ time a spanning forest of G consisting of as few trees as possible.*

4.3 s–t Paths and Cycles

First we consider the problem of finding a crossing-free path between two given vertices s and t. Suppose C is the connected component of G' that contains vertex

s. We select an edge e of E_X as the edge by which the crossing-free s–t path leaves C. Edge e connects C to another component of G' and the new bigger component thus created will then play the role of C and so on. If we again imagine this process of selecting edges as a search tree then we reach a leaf of the search tree if vertices s and t become vertices of the same component (then we have found a crossing-free path between them) or if we get stuck because we cannot leave the actual component C by edges from E_X.

Suppose vertices s and t do not belong to the same connected component of G'. We delete all components that have no vertex incident to an edge in E_X. Suppose the components to which s and t belong are still present.

Again it is rather easy to speed up the algorithm if crossings in X are not pairwise disjoint. As long as there is an edge e such that exactly z crossings in X share edge e and $2 \leq z \leq k$ we obtain the recurrence $T(k) \leq 2T(k-z)$ which solves to $T(k) \in O(2^{k/z})$.

Next we consider the case that the crossings in X are pairwise disjoint. 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'. Furthermore we can assume that there is no connected component C of degree 1 since either C contains s or t and then we *must* use the only edge from E_X incident to C, or C contains neither s nor t and then C can be deleted.

Now let C denote the connected component of G' that contains vertex s. 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 $x = |X(C)|$. The key observation is that if there is a crossing-free s–t path at all then there is one that uses exactly one of the edges in $E(C)$.

First we consider the case that $d - 2x \geq 1$. We obtain the recurrence $T(k) \leq dT(k-d+x)$. If d is an even number then $x \leq d/2 - 1$ and the recurrence solves to $T(k) \in O(4^{k/3}) \subseteq O(1.59^k)$. If d is not an even number then $x \leq (d-1)/2$ and the recurrence solves to $T(k) \in O(3^{k/2}) \subseteq O(1.733^k)$.

Second we consider the case that $d = 2x$. We can assume that there is no crossing c in $X(C)$ such that both edges in c go from C to the same connected component of G' since with respect to the existence of a crossing-free s–t path the edges in c would be equivalent and we could use one of the two edges if necessary.

Now if there are two crossings $c_1 = \{e_1, f_1\}$ and $c_2 = \{e_2, f_2\}$ in $X(C)$ such that e_1 and e_2 go from C to the same connected component of G', then it is sufficient to check only for one of e_1 and e_2 whether there is a crossing-free s–t path that uses this edge. Hence we obtain the recurrence $T(k) \leq (d-1)T(k-d/2)$ which solves to $T(k) \in O(3^{k/2})$.

Otherwise the edges in $E(C)$ go from C to pairwise distinct connected components of G'. Consider any crossing $c = \{e, f\}$ in $X(C)$. Let A and B denote the connected components of G' to which edges e and f respectively go from C. Since every connected component of G' has degree at least two, components A and B each have a vertex incident to an edge from $E_X \setminus E(C)$. Now observe that if there is a crossing-free s–t path that uses edge e and also goes through component B then there is another crossing-free s–t path that uses edge f to go directly to component B. Hence if we check edge e we can delete component B and all edges in E_X incident to B, and if we check edge f we can delete component A and all edges in E_X incident to A. Thus we get rid of at least $x + 1$ crossings for each edge in $E(C)$ checked and obtain the recurrence $T(k) \leq dT(k - d/2 - 1)$ which solves to $T(k) \in O(4^{k/3})$. This yields the following theorem.

Theorem 12 *Given a topological graph $G(V, E, X)$, we can decide in $O(m + k\beta_2^k)$ time whether G has a crossing-free s–t path ($\beta_2 = \sqrt{3} < 1.733$).*

Finally we present a simple way to employ the result of Theorem 12 for finding crossing-free cycles. For every edge e in E_X we check whether there is a crossing-free cycle that contains edge e. Let A and B be the connected components of G' that have a vertex which is incident to edge e and let X' be the set of crossings in X that do not contain edge e. Observe that there is a crossing-free cycle that contains edge e if and only if there is a crossing-free path from a vertex in component A to a vertex in component B using only edges in G' and in the crossings X'. Thus we obtain the following theorem.

Theorem 13 *Given a topological graph $G(V, E, X)$, we can decide in $O(m + k^2\beta_2^k)$ time whether G has a crossing-free cycle.*

5 MIP Formulations

In this section we give MIP formulations for the problem of computing a subgraph G' of a given topological graph $G(V, E)$ with the minimum weighted number of crossings among all subgraphs of a certain configuration. As it will turn out, the numbers of variables and constraints in our MIP formulations depend only linearly on k. This makes the MIP formulations an interesting alternative to the fixed-parameter algorithms in Section 4 for larger values of k. We first give a formulation for spanning trees. In Section 6 we use this formulation to measure the performance of a heuristic for the same problem on small to medium-size instances.

We introduce a continuous variable $x_{ee'}$ that will be forced to be 1 if both e and e' are in the spanning tree, and 0 otherwise. Our objective function is

$$\min! \sum_{\{e,e'\}\in X} \mu_{ee'}\, x_{ee'}. \tag{1}$$

We have the following constraints. We introduce a continuous variable y_e for each edge $e \in E$ that will be 1 if e is in the spanning tree and 0 otherwise. For each pair $\{e, e'\} \in X$ we require:

$$x_{ee'} \geq 0 \quad \text{and} \quad x_{ee'} \geq y_e + y_{e'} - 1 \tag{2}$$

Given our objective function (1) and the fact that y_e will be forced to lie in $\{0, 1\}$, constraint (2) is equivalent to $x_{ee'} = \min\{y_e, y_{e'}\}$. Thus $x_{ee'}$ indeed serves as an indicator whether e and e' cross. It remains to make sure that the graph $G'(V, E')$ with $E' = \{e \in E \mid y_e = 1\}$ is connected.

We model connectivity by fixing an arbitrary vertex $s \in V$ as sink and then introducing flow between s and every other vertex $t \in V \setminus \{s\}$. The flow is modeled by a 0–1 variable f_e^t for each edge $e \in E$. First we make sure that for each choice of t, the source s and the sink t have exactly one edge with flow:

$$\sum_{e \text{ incident to } s} f_e^t \quad = \sum_{e \text{ incident to } t} f_e^t \quad = \quad 1 \tag{3}$$

Note that if a graph contains an s–t path, then that graph also contains an s–t path that visits each vertex at most once. So we simply ensure that each vertex

$v \in V \setminus \{s,t\}$ has either zero or two incident edges with flow. To do this we need an auxiliary 0–1 variable h_v^t for each v. Now we can model our special kind of flow conservation in each vertex $v \in V \setminus \{s,t\}$.

$$\sum_{e \text{ incident to } v} f_e^t \quad = \quad 2h_v^t \tag{4}$$

Finally, for each edge $e \in E$ we lower-bound the "global" decision variable y_e (that decides whether e goes into the spanning subgraph G') by the "local" flow f_e^t (that goes through e from s to t):

$$y_e \geq f_e^t \tag{5}$$

Given our objective function and constraint (2), this is equivalent to setting $y_e = \max\{f_e^t \mid t \in V \setminus \{s\}\}$. This completes our MIP formulation for spanning trees. It consists of $O(nm + k)$ variables and constraints.

All of the remaining formulations use only $O(m + k)$ constraints and variables. For s–t paths we only need flow from s to a single target t. Thus the formulation can be simplified by making y_e a 0–1 variable, replacing f_e^t by y_e, and dropping constraint (5).

For cycles we drop constraint (3) in the formulation for s–t paths and require flow conservation (constraint (4)) to hold for *all* $v \in V$.

For the remaining configurations we do not need the auxiliary variable h_v^t any more. In a 2-factor each vertex must lie on a cycle, thus constraint (4) becomes

$$\sum_{e \text{ incident to } v} y_e = 2.$$

In a 1-factor each vertex must be matched, so constraint (4) becomes

$$\sum_{e \text{ incident to } v} y_e = 1.$$

For M-matchings constraint (4) becomes

$$\sum_{e \text{ incident to } v} y_e \leq 1.$$

Additionally we need a constraint that makes sure that the required number of edges is in the matching:

$$\sum_{e \in E} y_e \geq M.$$

Finally we observe that among all cycles or s–t paths with the minimum weighted number of crossings we can get a cycle or an s–t path with the maximum number of edges by subtracting the term $\sum_{e \in E} y_e/|E|$ from the objective function. For getting a subgraph with the minimum number of edges we add the same term. Similarly we can get a subgraph whose total edge length is shortest among all subgraphs that have the minimum weighted number of crossings for a given configuration. By swapping the integral and the fractional part of the objective function we can also find a matching that has the minimum weighted number of crossings among all maximum matchings.

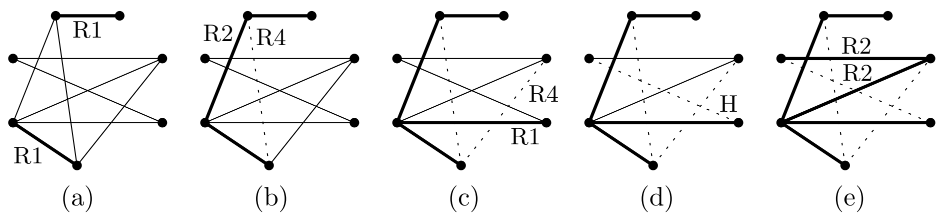


Fig. 7: An example run of our heuristic.

6 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 topological graphs. To simplify our description we assume that each pair of edges in the input graph G crosses at most once. We also assume that we know for each edge the edges it crosses—for geometric graphs this is easy to find out.

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 process stops as soon as the selected edges span G.

The heuristic applies the rules in an arbitrary order to the input graph until no more rule can be applied. Then a heuristic decision is taken. We decided to delete an edge e that maximizes $A(e) + 3S(e)$ among all active edges, where $A(e)$ and $S(e)$ are the numbers of active and selected edges that e crosses, respectively. The intuition behind this particular choice is that we want to use edges for the spanning tree that cross only few edges among those already selected. At the same time we want to avoid edges that cross too many of the still active egdes since some of those may also become tree edges later on. Our experiments show that this works quite well. After each heuristic decision again all rules are applied exhaustively.

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.

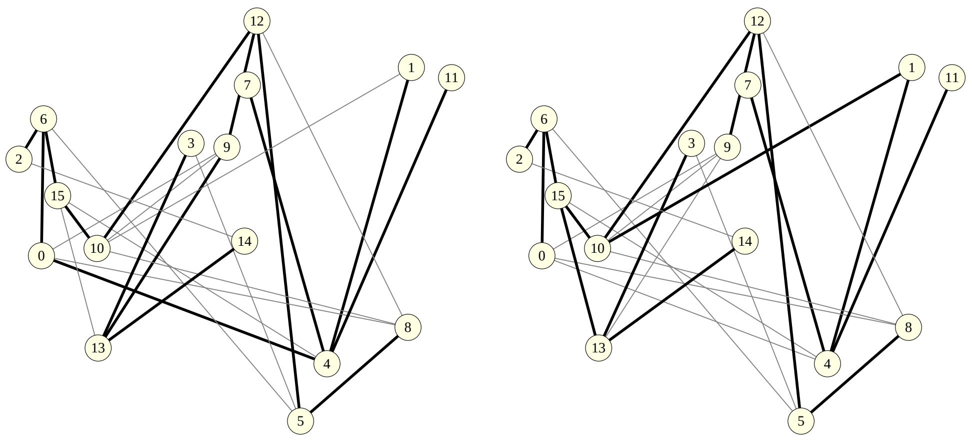
(R1) If an edge has no crossings with other edges, it is selected.

(R2) If an edge is a cut edge, it is selected.

(R3) If both endpoints of an edge belong to the same connected component, then this edge is deleted.

(R4) 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.

For an example run of our heuristic, see Figures 7 (a)–(e). From left to right each of the five figures shows a snapshot of the graph that results from applying rules and heuristic decisions. Active edges are depicted thin and solid, selected edges bold, and deleted edges dotted. Whenever an edge changes its status, the edge is labeled by the reason for its status change ("H" means heuristic decision).



Heuristic: 5 crossings MIP: 4 crossings

Fig. 8: Solutions for a 16-node random geometric graph.

Our heuristic always finds a spanning subgraph G' of G since rule (R2) makes sure that no cut edge is deleted. Due to rule (R1) G' may have cycles. If we insist on a tree, we can simply compute a breadth-first tree of G'. Clearly this tree also spans the underlying graph G and has no more crossings than G'. A brute-force implementation of our heuristic runs in $O(nm^3)$ time.

We have implemented the heuristic (except for rule (R4)) in C++ using the LEDA graph library [6]. 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 Section 5. 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 geometric graphs with 20 nodes and $24, 26, \ldots, 80$ edges as follows. First, edges were drawn randomly until the desired graph size was obtained. The graph was discarded if it was not connected. Finally the coordinates were chosen uniformly from the unit square. To these graphs we applied our heuristic and the MIP solver, see Figure 8, which shows the resulting spanning trees of the same random graph with 16 vertices and 26 edges. The bold line segments represent edges in the spanning tree, the thin segments are the non-tree edges. Figure 9 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). For random graphs with 30 vertices and 120 edges, the MIP solver often ran more than 48 hours without finding any solution that was at least as good as the one found by the heuristic. This is why we chose to show only results for random graphs with 20 vertices.

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

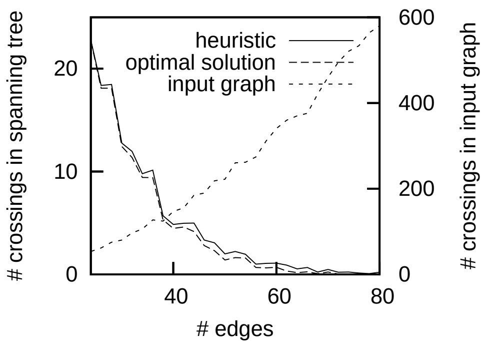


Fig. 9: Performance of heuristic and MIP on 20-node random geometric graphs.

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.

straight-line. We point out that we were not interested in any particularly *nice* embedding, but rather in an instance of an existing network with real-world coordinates. Our aim was to see whether our heuristic performed in a completely different way on these real-world instances as compared to the synthetic instances described above.

The results are given in Table 1. Figures 10 and 11 show the results of the heuristic on the Lufthansa World and Air Canada flight-connection graphs. Figures 12 and 13 show clippings of the previous two graphs. Note that YYZ is the airport code of Toronto. As in Figure 8, bold and thin line segments represent tree and non-tree edges, respectively.

Given our inapproximability results in Section 2 we were surprised to see how well our simple heuristic performs both on random and on real-world data: 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.

7 Conclusion

The main achievement of this paper is the design of the non-trivial fixed-parameter algorithms that decide whether a graph with k crossings has a plane spanning tree, an s–t path, or a cycle. The difficulty was to show that these configurations allow algorithms where the base in the exponential part of the running time is strictly less than 2, namely 1.9999992 for trees and $\sqrt{3}$ for paths and cycles. The main open question is whether such algorithms also exist for other configurations such as matchings, and whether the running times of our algorithms can be further reduced. We are also interested in finding an explanation for the amazingly good performance of our simple heuristic. How would a bad example look like? The reader is invited to

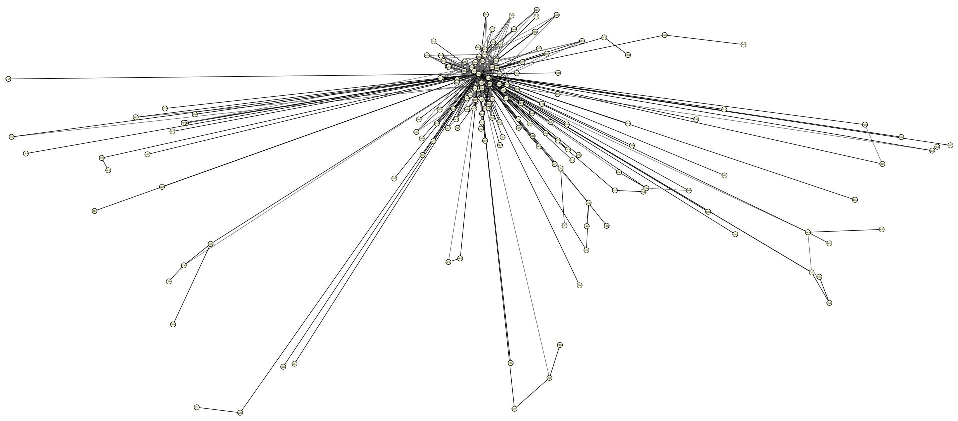


Fig. 10: Lufthansa World flight-connection graph.

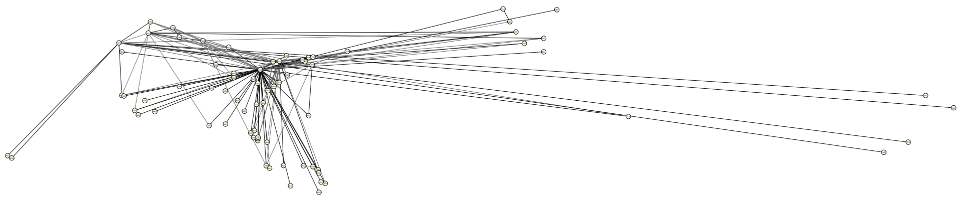
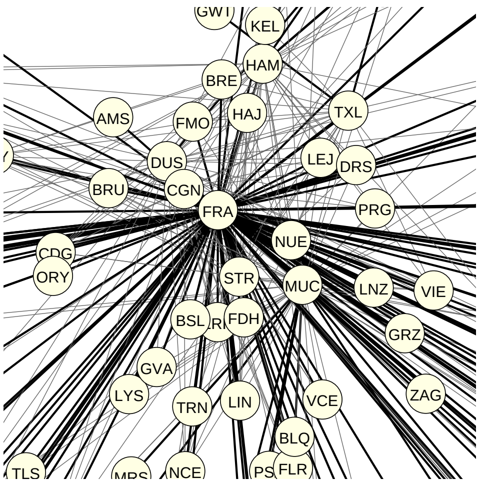


Fig. 11: Air Canada flight-connection graph.



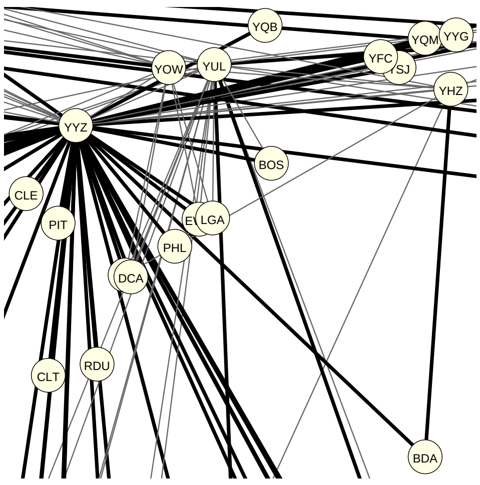


Fig. 12: Clipping of Figure 10.

Fig. 13: Clipping of Figure 11.

play around with our Java applet at `http://i11www.ira.uka.de/few_crossings`.

Acknowledgments

We thank Jiong Guo for pointing out that for the search-tree algorithms in Section 4 the case of disjoint crossings is actually the hardest. We also thank Georg Kliewer for supplying us with real-world data and Markus Völker for implementing the Java applet. Last but not least we wish to express our gratitude to two anonymous referees whose comments helped to improve the presentation of our results.

References

[1] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer, 1999.

[2] A. Grigoriev and H. L. Bodlaender. Algorithms for graphs embeddable with few crossings per edge. In M. Liśkiewicz and R. Reischuk, editors, *Proc. 15th Int. Symp. on Fundamentals of Computation Theory (FCT'05)*, volume 3623 of *Lecture Notes in Computer Science*, pages 378–387. Springer-Verlag, 2005.

[3] D. Halperin. Arrangements. In *Handbook of Discrete and Computational Geometry*, chapter 24, pages 529–562. CRC Press, 2004.

[4] K. Jansen and G. J. Woeginger. The complexity of detecting crossingfree configurations in the plane. *BIT*, 33:580–595, 1993.

[5] J. Kratochvíl, A. Lubiw, and J. Nešetřil. Noncrossing subgraphs in topological layouts. *SIAM J. Disc. Math.*, 4(2):223–244, 1991.

[6] K. Mehlhorn and S. Näher. LEDA: a platform for combinatorial and geometric computing. *Commun. ACM*, 38(1):96–102, 1995.

[7] S. Micali and V. V. Vazirani. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *Proc. 21th IEEE Symp. Found. Comp. Sci. (FOCS'80)*, pages 17–27, 1980.

[8] F. Rendl and G. Woeginger. Reconstructing sets of orthogonal line segments in the plane. *Discrete Mathematics*, 119:167–174, 1993.

[9] W. T. Tutte. A short proof of the factor theorem for finite graphs. *Canad. J. Math.*, 6:347–352, 1954.

[10] 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.