Fixed-Parameter Tractability for Non-Crossing Spanning Trees

Magnús M. Halldórsson[1,⋆], Christian Knauer[2], Andreas Spillner[3], and Takeshi Tokuyama[4,⋆⋆]

[1] Dept. of Computer Science, Reykjavik University
mmh@ru.is
[2] Institute of Computer Science, Freie Universität Berlin
christian.knauer@inf.fu-berlin.de
[3] Institute of Computer Science, Friedrich-Schiller-Universität Jena
spillner@minet.uni-jena.de
[4] Graduate School of Information Sciences, Tohoku University, Sendai, 980-8579 Japan
tokuyama@dais.is.tohoku.ac.jp

Abstract. We consider the problem of computing non-crossing spanning trees in topological graphs. It is known that it is NP-hard to decide whether a topological graph has a non-crossing spanning tree, and that it is hard to approximate the minimum number of crossings in a spanning tree. We consider the parametric complexities of the problem for the following natural input parameters: the number k of crossing edge pairs, the number μ of crossing edges in the given graph, and the number ι of vertices in the interior of the convex hull of the vertex set. We start with an improved strategy of the simple search-tree method to obtain an $O^*(1.93^k)$ time algorithm. We then give more sophisticated algorithms based on graph separators, with a novel technique to ensure connectivity. The time complexities of our algorithms are $O^*(2^{O(\sqrt{k})})$, $O^*(\mu^{O(\mu^{2/3})})$, and $O^*(2^{O(\sqrt{\iota})})$. By giving a reduction from 3-SAT, we show that the $O^*(2^{\sqrt{k}})$ complexity is hard to improve under a hypothesis of the complexity of 3-SAT.

1 Introduction

A *topological graph* is a graph with an embedding of its edges as curve segments in the plane such that each pair of edge curves intersects at most once. We refer to the embeddings of the vertices also as vertices, and to the geometric curve segments as *curves*. A topological graph is said to be *non-crossing* if none of the edge curves cross. We consider non-crossing subgraph problems that involve finding a non-crossing subgraph satisfying some property: spanning tree, s–t path, and cycle. All of these problems are known to be NP-hard [10,6]. In this article we focus on the non-crossing spanning tree problem (NCST). The corresponding minimization problem may be of interest when focusing on finding structures in the

⋆ Work done at University of Iceland, and supported by the Icelandic Research Fund.
⋆⋆ Supported by the project *New Horizons in Computing*, MEXT Japan.

F. Dehne, J.-R. Sack, and N. Zeh (Eds.): WADS 2007, LNCS 4619, pp. 410–421, 2007.

drawing of an embedded graph. Removing as many edges and crossings as possible makes it easier to recognize the structure of the graph in terms of connectivity.

Let G be a topological graph on n vertices and m edges. A *crossing* is a pair of edges that meet in a non-vertex point, and a *crossing edge* is one that participates in some crossing. A *crossing point* is a non-vertex point that is contained in at least two edge curves. Note that if d edges intersect in a crossing point, they create $\binom{d}{2} = d(d-1)/2$ crossings. Let X be the set of crossings in G, and let E_X be the set of crossing edges. Let $k = |X|$ be the number of crossings and let $\mu = |E_X|$ be the number of crossing edges. Observe that $\mu/2 \leq k \leq \mu(\mu-1)/2$. We assume without loss of generality that the curves intersect only in individual points, not in curve segments. Note that sometimes [8], a topological graph is allowed to have multiple crossings between a pair of edges, and our theory can be easily modified to that definition as long as the number of multiple crossings between each pair of edges is bounded by a constant.

A very naive method for a noncrossing subgraph problem is to exhaustively check the noncrossing properties for all subgraphs with the requested properties. This needs exponential time in the number m of edges of the graph. However, if k is small and the problem is polynomial time solvable without the noncrossing condition (e.g., spanning tree, cycle and s-t path), we have the following better strategy: For every crossing pair of edges, we delete one of the crossing edges to have a non-crossing subgraph. We have 2^k possible combinations of deletions, and it takes polynomial time for each fixed combination to find a spanning tree (for example) in the subgraph if it is connected. We can see that if G has a noncrossing spanning tree, we can find one by the above method. Thus, it is clear that the problem is computed in $O^*(2^k)$ time, where the O^*-notation hides polynomial terms. Recently, Knauer et al. [8] gave algorithms for NCST with improved time complexity of $O^*(1.9999992^k)$. This left the question of how far down the complexity can be brought down.

Our results. We give a number of results that answer many of the open questions about the fixed-parameter tractability of non-crossing subgraph problems.

We first give an improved $O^*(1.928^k)$-time algorithm for NCST. This is based on a compact *kernel* for the problem, and on a new set of reduction rules that takes advantage of limited recurrences for low-degree vertices. This approach actually applies to a generalized problem, involving arbitrary pairwise conflicts on the edges.

One of the main contributions of this paper is an algorithm for NCST with an asymptotic improvement in the time complexity to $2^{O(\sqrt{k})}$ (we ignore polynomial time preprocessing), see Section 3. This is based on finding a cyclic separator in a related planar graph. Thus turns out to be best possible, under the *exponential time hypothesis* that 3-SAT does not have a $2^{o(n)}$-time algorithm (where n is the number of variables), as shown in Section 5.

We also present fixed-parameter algorithms for two further parameters. For the parameter μ, the number of crossing edges, we give a $\mu^{O(\mu^{2/3})}$-time algorithm.

A *geometric graph* is a topological graph whose edges correspond to the straight-line segments that connect their endpoints. For geometric graphs, we

use another measure to design a fixed-parameter algorithm. Consider the vertex set of the embedded graph as a set of points in the plane. Then we can refer to the points that lie in the interior of the convex hull of the point set as *inner* points. The number ι of inner points has been used successfully to parameterize some hard geometric problems on points in the plane, including Minimum Weight Triangulation problem (MWT) [4,13]. For this parameter, we give an algorithm that solves NCST for geometric graphs in $O^*(\iota^{O(\sqrt{\iota})})$ time. Note that it is easy to come up with geometric graphs where ι is small but k is large. We also show that it is unlikely that a $2^{o(\sqrt{\iota})}$-time algorithm exists.

2 Improved Search-Tree Algorithm

We first give a simple search-tree method to find a non-crossing spanning tree in a topological graph with k crossings in time 1.9276^k (plus polynomial time preprocessing) if one exists. This improves on the previous bounds of 1.99999^k, as well as on the 1.968^k bound for a Monte-Carlo algorithm [8]. Although it will be greatly improved asymptotically to $2^{O(\sqrt{k})}$ in Section 3, we feel the above result is valuable since the search-tree algorithm is preferable in practice for the range of k that the problem is solvable in real feasible time, and our improved method gives little additional burden to programmers who want to implement a search-tree method. We reduce the original problem to a compact kernel problem, and then introduce some simple rules for a naive search-tree algorithm to obtain the improved time complexity.

Kernel. A kernel is a reduced problem instance, whose solution can be "easily" turned into a solution of the original instance. To form a kernel for NCST we use edge contractions, where contracting the edge uv in a graph G results in the graph where the vertices u and v have been merged into a single vertex that has all the neighbors of that either of its original vertices had.

Edges that cross are said to be *crossing edges*; if they share an endpoint v, we say they are *tangled*, more specifically, they are *tangled at* v.

To form a small kernel, we contract all non-crossing edges of the graph G yielding a new topological graph G'. More precisely, for each connected component of the induced subgraph of G by the non-crossing edges, we select any spanning tree, and contract it. The other edges in the connected component are deleted. It is clear that the kernel is obtained in polynomial (indeed, linear) time.

Note that this does not affect the crossing properties of the crossing edges. However, it can lead to non-tangled pairs to become tangled. A planar subgraph H' of G' maps to a subgraph H of G; adding the contracted edges to H still retains planarity. Hence, there is a bijective mapping between maximal planar subgraphs of G and G'.

Every edge in G' is crossing, thus the number of edges in G' is at most μ. Since the graph G' is necessarily connected and non-acyclic, the number of vertices in G' is at most its number of edges. We further delete all loop edges in G' even if they are crossing. This resolves some crossings, but does not affect the problem solution because of the property of a spanning tree.

Proposition 1. *A kernel for* NCST *with at most μ edges and vertices can be computed in linear time.*

Search-tree approach. We give reduction rules that result in an efficient search tree for a non-crossing connected spanning subgraph. A non-crossing spanning trees can then be easily found.

In most nodes of the search tree we select an edge e for *branching*: either a solution contains e or it does not. If it contains e it cannot contain the edges C_e crossing e. Hence, we obtain two subproblems: $G - \{e\}$ and $G - C_e$. In either of the subproblems, we eliminate all crossings incident on e, and apply the available contractions. The measure, $T(k)$, of a subproblem is the number of search tree leaves in terms of the number k of crossings. In subproblem $G - C_e$, the number of crossings is reduced by one, for a measure of $T(k-1)$. We want to show that the measure of $G - \{e\}$ is less.

We select branching edges in the following order of preference:

1. If there is an edge crosses two or more edges, then we choose such an edge. Crossing number k is reduced by at least two in $G - \{e\}$ (also in $G - C_e$).
2. For tangled parallel edges, we can pick either of them, yielding the same subproblem, since neither is twice-crossing (otherwise, we should apply the rule 1). This allows us to contract both edges, reducing k by one.
3. Consider a node v of degree ≤ 3. At least one edge e incident on v is not tangled with either of the other incident edges; otherwise, one of them would be twice-crossing. We branch on e and obtain on one branch a degree-2 node. For a degree-2 node with two incident tangled edges, branching on either edges yields the same subproblem after contractions. Otherwise, we branch on one of the incident edges, obtaining on one branch a degree-1 node. A degree-1 node must be connected in a spanning tree, thus only one choice is then possible. Hence, a problem with a node of degree at most 3 has a measure of at most $T(k-1) + T(k-2) + T(k-3)$.
4. Consider a degree-4 node v with an edge untangled at v. Let e be an edge incident on v that is not tangled with the other edges incident on v. When we branch on e, the non-included case leaves us with v being of degree-3. We then apply the degree-3 case above.
5. When none of the above rules apply, we branch on an arbitrary edge.

In each case, except when we reach the last rule, we measure the decrease in the number of crossings. This allows us to bound the size of the search tree.

Let us first consider what happens when we reach the last rule. In that case, all nodes are of degree at least four. Further, only nodes that have two tangled incident edge pairs have degree 4, while the others are of degree at least 5. Thus, each edge that is tangled at node v appears untangled at the other endpoint, since there are no tangled parallel edges and no twice-crossing edges. Thus, no two degree 4 nodes are adjacent to each other. We claim that the number of nodes, n, is at most $9\mu/20$. Let a denote the number of degree 4 nodes, and note that all neighbors of degree-4 nodes are of degree at least 5. Therefore, counting edge incidences, $\mu \geq \frac{4a+5(n-a)}{2} = \frac{5n-a}{2}$. But clearly, $\mu \geq 4a$. Combining the

two inequalities, we have that $\mu \geq (20/9)n$. We contract an edge, eliminating a vertex, in each round. Hence, the depth of the recursion is at most $n - 1 \leq (9/20)\mu = (9/10)k$, for a time complexity of $2^{0.9k}$.

Let us now evaluate the effects of the other branching rules. In each rule, we perform one or more branches, yielding a set of subproblems measured in terms of the number of crossings remaining. We express each case as a recurrence relation:

$$T(k) \leq \max \begin{cases} 2\,T(k-2), & \text{Twice-crossing edge} \\ T(k-1), & \text{Tangled parallel edges} \\ T(k-1) + T(k-2) + T(k-3), & \text{Degree-3 case} \\ T(k-1) + T(k-2) + T(k-3) + T(k-4), & \text{Degree-4 case} \\ 2^{0.9k}. & \text{Dense case} \end{cases}$$

The worst case is the degree-4 case, which yields $T(k) \leq 1.9276^k \approx 2^{0.9468k}$.[1]

Generalized structures. Our arguments do not use planarity in any way, except indirectly as prescribing conflicts between edges. Thus, the approach works more generally for finding spanning forests of graphs with conflicts between edges. More generally, we can formulate the Conflict-Free Spanning Tree (CFST) problem, where we are given a graph G and a conflict graph H defined on the edge set $E(G)$. We are to determine whether there exists a subset of mutually non-conflicting edges forming a spanning tree. In NCST, the conflicts are given by the crossings, and $|E(H)| = k$. For another example, the algorithm can be applied to layouts of graphs on surfaces of higher genus.

Theorem 1. *Given graphs G and H, CFST can be solved in time $O^*(1.9276^{|E(H)|})$.*

3 Separator-Based Algorithm

We describe here our algorithm for the non-crossing spanning tree problem. The approach bears some similarity to the algorithm of Deineko et al [1] for the Hamilton cycle problem in planar graphs. that has complexity $O^*(2^{O(\sqrt{n})})$. Our method is based on a cycle separator theorem of Miller.

Proposition 2. *(Miller [12]) Let G' be an embedded triangulated planar graph on n vertices. Then, there is a linear time algorithm that finds in G' a simple cycle C of at most $\sqrt{8n}$ vertices that partitions $G' - C$ into a vertex set A that lies within the region inside of C, and a vertex set B that lies outside of C, with $|A| \leq 2n/3$ and $|B| \leq 2n/3$.*

Before applying the above theorem, we resolve the multiplicities of the kernel. The *multiplicity* of a crossing is the number of pairs of edges that meet in the same point. Large multiplicity can confuse good algorithms, especially those based on separators, and the same can be said of high-degree vertices.

[1] 1.9276 represents the positive-valued solution of the equation $x^4 = 1 + x + x^2 + x^3$.

Fortunately, we can assume without loss of generality that crossings are of unit multiplicity and vertices of maximum degree 3. We omit details, but the basic idea is to clip edges at high-degree vertices and to replace the clipped stars by binary trees and to wiggle edge curves in order to avoid degenerate crossings. We have the following theorem:

Theorem 2. *Suppose there is an algorithm that solves* NCST *on degree-3 graphs with unit crossing multiplicity in time $T(k, \mu, n)$. Then, there is an algorithm for* NCST *for general topological graphs running in time $O(T(k, \mu, n))$.*

Given a topological graph H, we form an associated triangulated plane graph $P = P_H$ as follows. We replace each crossing point of H by a vertex and the curve of each crossing edge by line segments connecting the vertices and the crossing points. Finally, we arbitrarily triangulate the graph. The edges of the resulting graph P are therefore of three kinds: non-crossing edges from H, segments of crossing edges (connected a crossing point to either another crossing point or to an original vertex), and newly introduced "dummy" edges. Observe that the number $n(P_H)$ of vertices in H equals $\mu + k$.

The idea of our algorithm is as follows. In the preprocessing step, we find a kernel, as guaranteed by Proposition 1, and apply the multiplicity reduction of Theorem 2 to ensure each crossing point involves exactly two crossing edges.

The main algorithm finds a cycle separator in the derived plane graph P_H, and solves the two resulting subgraphs of H recursively, under all possible ways of constraining one subsolution to contribute to the connectedness of the whole solution. More precisely, if C is a cycle separator of P_H, we partition its nodes into C_v, a set of vertices of H, and C_c, a set of crossing points in H. The algorithm tries all $2^{|C_c|}$ ways of breaking the crossings of C_c. Consider one such decision vector D, and let D_v be the set of vertices of the chosen crossing edges that are on the inside of the cycle C. Consider now the set $S = C_v \cup D_v$. This set can be topologically arranged on a circle C', such that no edges cross the circle. Let H_A be the subgraph of H induced by vertices on the inside of *or on* the circle C', and H_B the subgraph on the outside of or on C'. Thus, $V(H_A) \cap V(H_B) = S$.

Given H_A and H_B, the algorithm examines all the ways that the vertices of S can be connected inside C' (i.e. within H_B) while maintaining planarity. Namely, if we view S as being an ordered set, we seek, in combinatorial terminology, a *non-crossing partition* of S. A partition of an ordered set is non-crossing if no two blocks "cross" each other, i.e. whenever a and b belong to one block and x and y to another, they are not arranged in the order $axby$. For each non-crossing partition Π, we form a star forest $X = X_\Pi$ with the leaves of each star corresponding to a block of the partition and a new node as the root of the star. Let $H'_B = H_B \cup X$. The algorithm recursively solves H'_B, yielding a non-crossing forest F_B in H_B. By induction, crossing edges in G have either all of its segments in H in F_B or none. The algorithm then recursively solves $H'_A = H_A \cup F_B$, giving a non-crossing spanning tree in G.

Theorem 3. *The algorithm solves* NCST *in time $2^{O(\sqrt{k})} + O(m)$ and polynomial space.*

Proof. We first show that the correctness of the algorithm. Suppose that the input graph G contains a non-crossing spanning tree T. Let T_A (T_B) be the restriction of T to H_A (H_B). Each tree of the forest T_A contains some nodes of S; for the purpose of the solution of H_B, all that matters is that it connects those vertices together. Thus, if we replace each tree U of T_A by a star with nodes in $S \cap U$ as leaves, the resulting union, joined with T_B, induces a connected tree spanning all the nodes. Hence, by induction, the first recursive call of the algorithm returns a spanning tree of H'_B, whose restriction to H_B is the forest F_B. Now, $F_B \cup T_A$ is connected and spans $F_B \cup H_A$. Hence, the second recursive call will also result in a non-crossing spanning tree T' of $H'_A = H_A \cup F_B$. The nodes of F_B are the nodes of H_B; hence, we have spanned all of G. Thus, the algorithm correctly computes a non-crossing spanning tree. On the other hand, if G does not contain a non-crossing spanning tree, the second recursive call never finds a non-crossing spanning tree.

Next, we analyze the complexity. Let $\nu = n(P_H) = \mu + k$ to be the *measure* of the problem. By Proposition 2, the algorithm finds a cycle separator in P of size at most $z = \sqrt{8\nu}$. We have at most 2^z ways of resolving the crossing edges on the separator. The size of S and the cycle C' is still z. The number of non-crossing partitions of S equals the Catalan number $C_z = \frac{1}{z+1}\binom{2z}{z} < 4^z$.Thus, there are less than 8^z cases considered by the algorithm.

Each case involves two subproblems. The larger of the subproblems is of measure M of at most $2\nu/3 + z$. A more careful analysis actually shows that most of the cases involve smaller subproblems. The measure of the smaller subproblem is at most $(\nu - M) + 2z$. The time complexity for any subproblem, aside from recursive calls, is linear in the size of the graph. Thus, the complexity of the algorithm is bounded by $T(\nu) = O(8^z/z^{3/2}) \cdot (T(2\nu/3 + z) + T(\nu/3 + z)) + O(\nu)$. This leads to $T(\nu) = O(2^{18\sqrt{\nu}})$. Since $\nu = \mu + k \leq 3k$, $T(\nu) = 2^{O(\sqrt{k})}$. QED.

The parameter μ. A straightforward $O^*(2^\mu)$ algorithm for NCST follows by considering all subsets of the set of crossing edges, and $O^*(2^{0.552\mu})$ can be obtained by the search-tree method (omitted in this version). We further give the following asymptotic improvement by combining the search method and the separator-based method:

Theorem 4. *NCST can be solved in $\mu^{O(\mu^{2/3})} + O(m)$ time and polynomial space.*

Proof. We split the computation into two cases, depending on the size of μ relative to $\nu = \mu + k$. Let $R(\mu)$ be the number of subproblems in an instance with μ crossing edges. If $\nu < 2\mu^{4/3}$ (Case 1), then the separator-based algorithm gives $R(\mu) < 2^{c\sqrt{\nu}} < 2^{2c\mu^{2/3}}$. Otherwise (Case 2), $\nu \geq 2\mu^{4/3}$. Then there exists an edge that participates in at least $2\mu^{1/3}$ crossings. We branch on that edge, resulting in two subproblems: one without that edge, and the other without all the edges crossing it. This gives the recurrence $R(\mu) \leq R(\mu-1) + R(\mu-2\mu^{1/3}) + 1$. The time complexity follows from this recurrence using Case 1 as the induction basis.

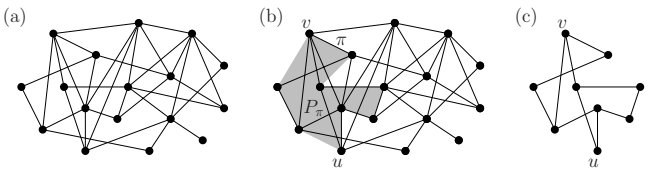

Fig. 1. A geometric graph, the polygon P_π and the subgraph G_π

4 Dynamic Programming Approach for the Parameter ι

A necessary prerequisite to successfully parameterize a problem with the number of inner points is that we can solve the problem in polynomial time for sets of points in convex position. For geometric graphs whose vertices are in convex position, it is easy to see that NCST can be solved using dynamic programming in $O(n^3)$ time. So this parameter could be viewed as a measure that tries to capture for each input its "distance from triviality" [3]. The key observation that provides a unified view on many of the problems mentioned in the introduction is that we can reformulate them as the search for a certain kind of triangulation. For NCST, we are given a geometric graph $G = (V, E)$ and the goal is to find a triangulation $\mathcal{T}$ of V such that the graph formed by those edges of G contained in the triangulation $\mathcal{T}$ is connected. Then, we can easily find a noncrossing spanning tree by using only those edges.

We describe the subproblem considered in our dynamic programming algorithm to solve NCST. The subproblem is defined by a crossing-free path π that starts at an outer vertex u, visits some inner vertices and ends at another outer vertex v. Such a path π splits the convex hull of V into two polygonal regions. Note that π is not necessarily a path in the input graph, but any noncrossing path connecting vertices by line segments is fine; indeed, it is a path in the (unknown) triangulation we are searching for. By P_π we denote the polygonal subregion to the left of π. An example is given in Figure 1(b), where P_π is shaded. The subgraph G_π induced by π consists of all those vertices and edges of G that are contained in P_π. This is illustrated in Figure 1(c).

We now describe what we actually want to compute for each P_π. It is not enough to decide whether or not there is a crossing-free spanning tree in G_π. Intuitively, we need a list of those crossing-free spanning forests of G_π where each tree in the forest shares at least one vertex with the path π. However, it is costly to consider the complete list of such spanning forests. Instead, it suffices to know which vertices on π belong to the same tree in the spanning forest. We can handle this by considering partitions of the set of vertices of the path π. For each such path π we have a collection of subproblems: one for each partition of the vertices of π. For such a subproblem we must decide whether or not there is a spanning forest of G_π such that every tree in the forest has at least one vertex on π and vertices on π in a component of the partition belong to the same tree in the forest.

The key fact for the analysis is that the existence of small simple cycle separators in planar triangulated graphs implies that we can restrict ourselves to subproblems defined by paths with $O(\sqrt{\iota})$ vertices [9]. Thus, the number of polygonal regions P_π considered in the algorithm is bounded by $n^2\iota^{O(\sqrt{\iota})}$ (selecting two outer vertices and $O(\sqrt{\iota})$ inner vertices), and there are $\iota^{O(\sqrt{\iota})}$ possible partitions for the vertices of the path π of each region. In the DP table we record whether there is a triangulation containing a feasible forest for each partition of each such polygonal region. Thus, the table size is $O(n^2\iota^{O(\sqrt{\iota})})$.

It remains to sketch how we process a subproblem in P_π by using information for smaller polygons stored in the dynamic programming table. We check every triangle Δ that is contained in P_π, shares an edge with the path π, and does not contain a vertex of V in its interior. Checking Δ means to decide whether a suitable triangulation for the subproblem containing Δ exists. By removing Δ from P_π we have one or two subpolygons, and this leads us to one or two smaller subproblems. We remark that we discard the choice of Δ if it generates a subpolygon with too many interior points on its boundary. It is routine to see that we can now solve the subproblem for P_π by referring the dynamic programming table. Thus, we have the following theorem:

Theorem 5. *Given a geometric graph G with n vertices we can decide in $O^*(\iota^{O(\sqrt{\iota})})$ time and $O^*(\iota^{O(\sqrt{\iota})})$ space whether or not G admits a crossing-free spanning tree.*

The time and space complexities are $O(n^3\iota^{O(\sqrt{\iota})})$ and $O(n^2\iota^{O(\sqrt{\iota})})$ if we consider polynomial factors of n. We can also compute a crossing-free spanning tree (not only decision) if exists in the same time and space complexities.

5 Hardness Results

We show here that the results of Section 3 are in some sense best possible, assuming the well-known *Exponential time hypothesis*, which is that 3-SAT cannot be solved in sub-exponential time. This hypothesis was formalized by Impagliazzo, Paturi, and Zane [5]. Evidence was given there and in later papers for support of the hypothesis. We are interested in the NCST_κ problem, where we decide whether an input geometric graph $G = (V, E)$ with k crossings has a crossing free spanning tree, and we use $\kappa(G) = \lceil\sqrt{k}\rceil$ as the parameter. We want to relate the question of whether there is an algorithm solving NCST_κ in $O^*(2^{o(\kappa(G))})$ time to an open question concerning the $\mathsf{3SAT}_\nu$ (3-SAT with the parameter ν):

> Instance: Exact 3-SAT formula (CNF formula with exactly three literals per clause) F.
> Parameter: The number $\nu(F)$ of variables occurring in F.
> Problem: Decide whether F is satisfiable.

The exponential time hypothesis is that $\mathsf{3SAT}_\nu$ cannot be solved in time $O^*(2^{o(\nu(F))})$. If we take the closure of $\mathsf{3SAT}_\nu$ under so called *subexponential reduction families (serf)* (cf. [2]) we obtain the class S[1]. Our goal is to show

that NCST_κ is S[1]-hard. S[1]-hardness can be also shown for the parameter $\sqrt{l}$, but we omit it because of space limitation.

To achieve the S[1]-hardness, it suffices to give a *parameter preserving polynomial time reduction* from $\mathsf{3SAT}_\nu$ to NCST_κ . Such a reduction transforms a given instance F of $\mathsf{3SAT}_\nu$ in polynomial time into a instance G of NCST_κ such that $\kappa(G) \in O(\nu(F))$. We can give such a reduction through some intermediate problems. The first is $\mathsf{3SAT}_\mu$, which has the same instance and problem as $\mathsf{3SAT}_\nu$ but the parameter is the number $\mu(F) = 3m$ where m is the number of clauses of F. $\mathsf{3SAT}_\mu$ is known to be S[1]-complete (cf. [2]).

With every 3-CNF formula F we can associate a bipartite graph $H(F) = ((V, C), E)$. The vertices in V represent the variables occurring in F. The vertices in C represent the clauses of F. A variable is connected to a clause by an edge in E iff the variable occurs in this clause. Lichtenstein [11] gives a polynomial time algorithm that computes for every 3-CNF formula F a 3-CNF formula F' such that (1) formula F is satisfiable iff formula F' is satisfiable, (2) the associated bipartite graph $H(F')$ is planar, and (3) Formula F' has $O((\mu(F))^2)$ clauses.

This immediately gives a parameter preserving polynomial time reduction from $\mathsf{3SAT}_\mu$ to the following $\mathsf{planar\ 3SAT}_{\mu'}$.

> Instance: Exact 3SAT formula F such that the graph $H(F)$ is planar.
> Parameter: $\mu'(F) = \lceil\sqrt{m}\rceil$ where m is the number of clauses of F.
> Problem: Decide whether F is satisfiable.

Moreover, it is shown in [11] that we can restrict to instances F of $\mathsf{planar\ 3SAT}_{\mu'}$ where the bipartite graph $H(F)$ has a drawing satisfies the following conditions: Every vertex of $H(F)$ that represents a variable in F lies on a horizontal line, no edge crosses the horizontal line, and no vertex representing a clause lies on the horizontal line. Hence $\mathsf{planar\ 3SAT}_{\mu'}$ with these properties is S[1]-hard.

Thus, it suffices to give a polynomial time reduction from this restricted version of $\mathsf{planar\ 3SAT}_{\mu'}$ to NCST_κ . We remark that this reduction was also given in [7] in the context of NP-hardness and approximation hardness.

Our reduction maps a given instance F of $\mathsf{planar\ 3SAT}_{\mu'}$ to a geometric graph G_F such that G_F has a crossing-free spanning tree iff F is satisfiable. The overall structure of G_F is indicated in the left picture of Figure 2 for $F = (x_1 \vee x_2 \vee \overline{x}_4) \wedge (x_2 \vee \overline{x}_3 \vee x_4) \wedge (x_1 \vee x_3 \vee x_4)$.

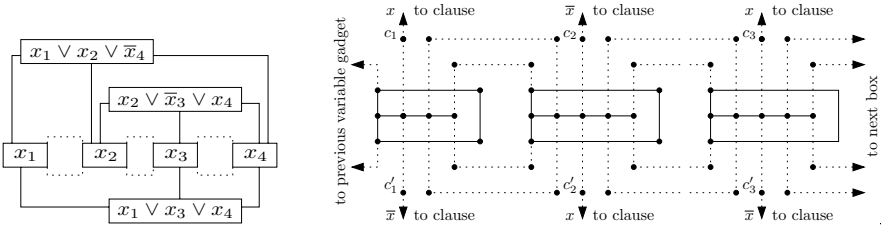

Fig. 2. Overall structure of G_F (left), and a part of a variable gadget (right)

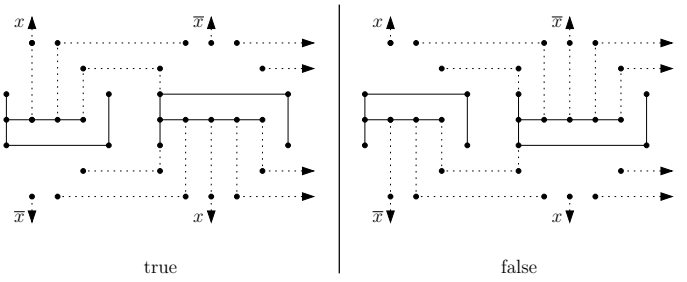

Fig. 3. Spanning trees encoding true and false for a variable

We have a gadget for every variable occurring in F. These gadgets are arranged along a horizontal line ℓ. We further have a gadget for every clause in F which is connected with every variable occurring in the clause. This gadget looks like a three-legged comb.

Now let's have a closer look at the gadgets. The leftmost part of the gadget for a variable x is shown as the right picture in Figure 2. The gadget for x consists of at most twice as many boxes as there are clauses in F that contain x. Three of these boxes are drawn with solid edges in Figure 2. The dotted edges that emanate from the boxes fulfill three tasks. First they connect consecutive boxes within one variable gadget. Second they connect the first and last box of variable gadgets that are consecutive on the line ℓ. Third they connect boxes to clause gadgets. Each dotted edge that connects a variable gadget to a clause gadget is associated with a literal. This literal will be true if the dotted edge is part of the spanning tree of G_F.

The intended way of simulating the truth assignment of the variable x is indicated in Figure 3. The Boolean values of x correspond to the two ways in which a crossing-free spanning tree can be chosen among the edges of the gadget of x. Note that only every other box can be connected to a clause gadget above (below) ℓ. This way we ensure that according to the value of x either only the dotted edges associated to positive literals or only the dotted edges associated to negative literals can connect x to clause gadgets. Not all points of type c_i or c'_i in a variable gadget are used—only those where the variable is in fact connected to a clause gadget in G_F. A clause gadget is just a vertex of degree three connecting to the corresponding literals.

It remains to argue that our reduction is parameter preserving. We charge the crossings in one box of a variable gadget to a clause that is connected to this box or its predecessor or its successor. At least one of these boxes must be connected to a clause, otherwise we could omit them. This way a clause is charged only a constant number of times and every time we charge the clause we charge it only with a constant number of crossings. Hence, the number of crossings in G_F is in $O(m)$ where m is the number of clauses of F. But this gives $\kappa(G_F) \in O(\mu'(F))$, as desired.

6 Concluding Remarks

As we have claimed in the introduction, we can apply our method to several other problems such as non-crossing s–t paths and cycles. We can also deal with the optimization problems, minimizing either the number of components in a non-crossing spanning forest or the number of crossing edges in a spanning tree. These extensions will be given in the full paper.

Acknowledgement. The authors gratefully acknowledge Alexander Wolff for his valuable suggestions and help in organizing this joint research.

References

1. Deineko, V.G., Klinz, B., Woeginger, G.J.: Exact algorithms for the Hamilton cycle problem in planar graphs. Inf. Process. Lett. 34, 269–274 (2006)
2. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
3. Guo, J., Hüffner, F., Niedermeier, R.: A structural view on parameterizing problems: Distance from triviality. In: Downey, R.G., Fellows, M.R., Dehne, F. (eds.) IWPEC 2004. LNCS, vol. 3162, pp. 162–173. Springer, Heidelberg (2004)
4. Hoffmann, M., Okamoto, Y.: The minimum weight triangulation problem with few inner points. Computational Geometry 34(3), 149–158 (2006)
5. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity. Journal of Comput. Syst. Sci. 63, 512–530 (2001)
6. Jansen, K., Woeginger, G.J.: The complexity of detecting crossingfree configurations in the plane. BIT 33, 580–595 (1993)
7. Knauer, C., Schramm, É., Spillner, A., Wolff, A.: Spanning trees with few crossings in geometric and topological graphs. In: Proc. European Workshop on Computational Geometry, pp. 195–198 (2005)
8. Knauer, C., Schramm, É., Spillner, A., Wolff, A.: Configurations with few crossings in topological graphs. Computational Geometry 37(2), 104–114 (2007)
9. Knauer, C., Spillner, A.: A fixed-parameter algorithm for the minimum weight triangulation problem based on small graph separators. In: Fomin, F.V. (ed.) WG 2006. LNCS, vol. 4271, pp. 49–57. Springer, Heidelberg (2006)
10. Kratochvíl, J., Lubiw, A., Nešetřil, J.: Noncrossing subgraphs in topological layouts. SIAM J. Disc. Math. 4(2), 223–244 (1991)
11. Lichtenstein, D.: Planar formulae and their uses. SIAM J. Computing 11, 329–343 (1982)
12. Miller, G.L.: Finding small simple cycle separators for 2-connected planar graphs. J. Comput. Syst. Sci. 32, 265–279 (1986)
13. Spillner, A.: A faster algorithm for the minimum weight triangulation problem with few inner points. In: Proc. Workshop Algorithms and Complexity in Durham (ACID'05), pp. 135–146. KCL Publications (2005)