**Technical Report DIKU-TR-98/11**
**Department of Computer Science**
**University of Copenhagen**
**Universitetsparken 1**
**DK-2100 KBH Ø**
**DENMARK**

**April 1998**

# Exact Algorithms for Plane Steiner Tree Problems: A Computational Study

David M. Warme, Pawel Winter and Martin Zachariasen

# Exact Algorithms for Plane Steiner Tree Problems: A Computational Study

D.M. Warme
*System Simulation Solutions, Inc.*
*Alexandria, VA 22314, USA*
E-mail: warme@s3i.com

P. Winter
*Department of Computer Science*
*University of Copenhagen, Denmark*
E-mail: pawel@diku.dk

M. Zachariasen
*Department of Computer Science*
*University of Copenhagen, Denmark*
E-mail: martinz@diku.dk

**Abstract**

We present a computational study of exact algorithms for the Euclidean and rectilinear Steiner tree problems in the plane. These algorithms — which are based on the generation and concatenation of full Steiner trees — are much more efficient than other approaches and allow exact solutions of problem instances with more than 2000 terminals. The full Steiner tree generation algorithms for the two problem variants share many algorithmic ideas and the concatenation part is identical (integer programming formulation solved by branch-and-cut). Performance statistics for randomly generated instances, public library instances and "difficult" instances with special structure are presented. Also, results on the comparative performance on the two problem variants are given.

# Contents



# 1 Introduction

The Euclidean and rectilinear Steiner tree problems in the plane are by far the most studied geometric Steiner tree problem variants. The classical Euclidean problem has roots more than two centuries back while the rectilinear was first considered by Hanan [13] in the 1960's. The interest in the latter came mainly from applications in, e.g., VLSI design. In the 1970's both problems were shown to be NP-hard [11, 12] and this virtually shattered the hope of finding efficient (polynomial time) algorithms for these problems.

Informally, we ask for a shortest interconnection — a Steiner minimum tree (SMT) — of a set $Z$ of $n$ terminals (points in the plane) with respect to a given distance function. Let $u = (u_x, u_y)$ and $v = (v_x, v_y)$ be a pair of points in the Cartesian plane $\Re^2$. The distance in the $L_p$-metric, $1 \leq p \leq \infty$, between $u$ and $v$ (or simply the $L_p$-distance) is $\|uv\|_p = (|u_x - v_x|^p + |u_y - v_y|^p)^{1/p}$. In this work we consider the Steiner tree problem with the Euclidean $L_2$-distance and the rectilinear (or Manhattan) $L_1$-distance. Hanan [13] proved that for the rectilinear version, it is always possible to find an SMT with edges belonging

to the *Hanan grid*. The Hanan grid is obtained by drawing horizontal and vertical lines through all terminals. As a consequence, Steiner points appear at intersections of these lines only.

Euclidean SMTs (ESMTs) and rectilinear SMTs (RSMTs) are unions of *full Steiner trees* (FSTs) whose terminals are incident with one FST-edge each. An Euclidean FST (EFST) and a rectilinear FST (RFST) spanning $k$ terminals, $2 \leq k \leq n$, has $k-2$ Steiner points (except when $k=4$; RFSTs can then have one Steiner point incident with four edges). Steiner points in EFSTs have three incident edges meeting at 120°. Steiner points in RSMTs are incident with 3 edges (except for the already mentioned case).

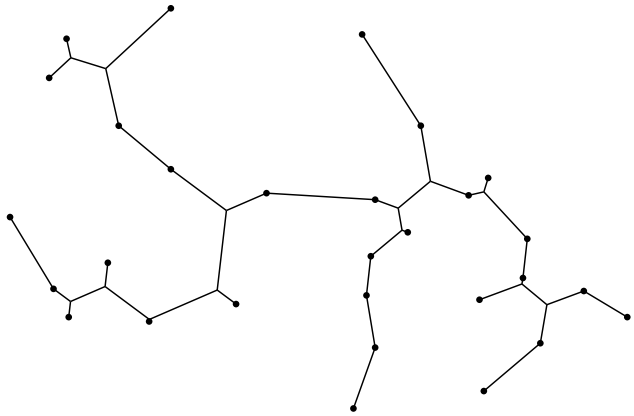
A minimum spanning tree (MST) for the terminals in $Z$ is a shortest network spanning $Z$ without introducing Steiner points. Euclidean MSTs (EMSTs) and rectilinear MSTs (RMSTs) for $Z$ can be constructed in $O(n \log n)$ time [18]. The length of an EMST (resp. RMST) exceeds the length of an ESMT (resp. RSMT) by at most a factor of $2/\sqrt{3}$ (resp. 3/2) [15].

First exact algorithms for the Euclidean and rectilinear Steiner tree problems (see Hwang et al. [15] for references) are based on a straightforward common framework. Subsets of terminals are considered one by one. For each subset, all its FSTs are determined one by one, and the shortest is retained. Several tests can be applied to these shortest FSTs in order to identify and prune away those that cannot be in any SMT. Surviving FSTs are then concatenated in all possible ways to obtain trees spanning all terminals. The shortest of them is an SMT.

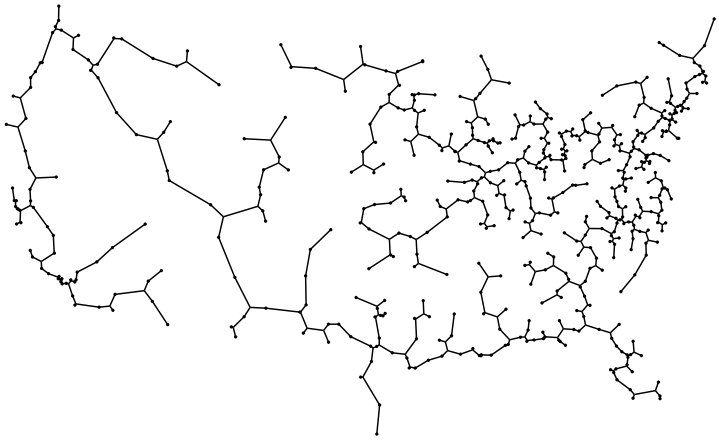
The bottleneck of this approach is the generation of FSTs. Winter [25] suggested a departure from the above general framework. He observed that substantial improvements are available if EFSTs are generated across various subsets of terminals. Retained EFSTs are not necessarily minimal. However, pruning tests are so powerful that only very few EFSTs survive. Similar strategy was recently applied to the generation of RFSTs by Zachariasen [27]. Spectacular speed-ups have been achieved in both cases. As a consequence, the concatenation of FSTs became a bottleneck of FST-based algorithms for both the Euclidean and the rectilinear Steiner tree problems.

Recent result of Warme [24] improved the concatenation dramatically. He noticed that the concatenation of FSTs can be formulated as a problem of finding a minimum spanning tree in a hypergraph with terminals as vertices and subsets spanned by FSTs as (hyper)edges. He solved this problem using branch-and-cut. Instances of the Euclidean and rectilinear Steiner tree problem with as many as 2000 terminals can today be solved in a reasonable amount of time. In Figure 1 we illustrate the dramatic progress in the performance of exact algorithms.

The purpose of this work is to present computational results using the EFST generator of [25, 26] and the RFST generator of [27] with the FST concatenator of [24]. We present the main algorithmic ideas, but the reader is referred to the above mentioned papers for further details. The work is organized as follows:



29 cities in the United States and Canada



532 cities in the United States

Figure 1: Euclidean Steiner minimum tree examples. The top instance was published in Scientific American in 1989 and was at that time "close to the limit of computing capabilities" [2]. The bottom instance, `att532` from the *TSPLIB* collection, was solved in a few hours on a workstation using the algorithm presented in this paper.

The algorithms for generating EFSTs and RFSTs are described in Section 2. A survey of FST concatenation algorithms is presented in Section 3. Computational results are given in Section 4 and concluding remarks in Section 5.

# 2 Generating Full Steiner Trees

The algorithms given by Winter [25] and Zachariasen [27] for generating EFSTs and RFSTs, respectively, share many algorithmic ideas. The simpler RFST algorithm is given first. The EFST algorithm is then described using the same framework.

Before we present the general framework, we note that 2-terminal FSTs in any SMT can be restricted to edges of an arbitrarily chosen MST [15]. Thus we only describe the generation of FSTs with three or more terminals.

## 2.1 Generation of RFSTs

We assume that terminals are in general position, i.e., no pair of terminals has the same $x$- or $y$-coordinates. This assumption simplifies the description by avoiding some straightforward but tedious special cases.

Hwang [14] proved that there always exists an RSMT in which every RFST has a very restricted shape shown in Figure 2. Any such RFST spanning $k$ terminals has a *root terminal* $z_0$ and a *tip terminal* $z_t$. The root and the tip are connected by a *backbone*. The backbone consists a *long leg* (incident with the root) and a perpendicular *short leg* (incident with the tip). The remaining terminals are attached to the long leg by alternating straight line segments called *branches*. At most one branch can be attached to the short leg (away from the root) as shown in Figure 2b. Steiner points are all on the backbone at points xdvi adwhere branches are attached.

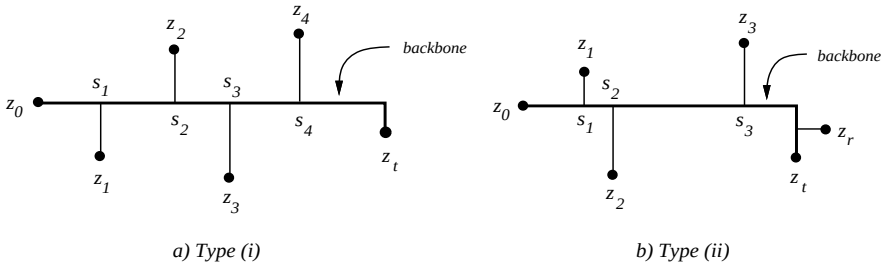



Figure 2: Two types of RFSTs

The generation of RFSTs with three or more terminals is as follows. For a given root $z_0$, long legs are grown in each of the four possible directions. For a

given direction (to the right, say), only terminals with greater $x$-coordinate are considered.

Suppose that the long leg from $z_0$ has been extended to the $x$-coordinate $x_k$ of the terminal $z_k$, and the branches alternate below and above the long leg. Several tests (described in Subsection 2.3 and in [27]) can be applied to check whether such an alternating sequence can appear in at least one RSMT. If not, the $z_k$-branch is replaced by another (alternating) branch farther to the right or backtracking occurs.

Assume that the long leg from $z_0$ to $z_k$ cannot be pruned away. Each terminal $z_t$ to the right of $z_k$ is attached to the long leg as a tip (by extending the long leg and attaching the alternating short leg). Several tests can be applied to check if this RFST can occur in any RSMT. Furthermore, each terminal $z_r$ to the right of the tip with its $y$-coordinate between $y_0$ and $y_t$ is attached to the short leg. Again, most of such RFSTs can be pruned away by one of several efficient pruning tests.

## 2.2 Generation of EFSTs

Consider an EFST for $k$ terminals, $3 \leq k \leq n$, as shown in Figure 3. A path between 2 arbitrary terminals (one considered as a root and the other as a tip) goes through one or more Steiner points and can be considered as a backbone. At each Steiner point, the long leg turns 60° either to the left or to the right. Hence, contrary to the rectilinear case, the long leg does not have a fixed direction. Furthermore, in the rectilinear case, the branches are line segments appearing in the alternating fashion. This is not so in the Euclidean case. Branches can involve more than one terminal and consecutive branches do not need to alternate. In conclusion, the generation of EFSTs is more complicated than the generation of RFSTs. On the other hand, pruning tests for the Euclidean case are more efficient so that the number of surviving EFSTs is (on average) smaller than the number of surviving RFSTs (for the same number of terminals).

In order to explain how EFSTs with three or more terminals are generated, we restrict our attention to 1-terminal branches. Assume that $z_0$ is selected as the root, and $k$ terminals $z_1$, $z_2$, ..., $z_k$ have been attached to the long leg (in that order) making 60° left turns at each Steiner point (Figure 4).

The first complication (when compared with the rectilinear case) arises due to the fact that the locations of Steiner points are not known. The first Steiner point $s_1$ on the long leg must be located somewhere on the *Steiner arc* $\widehat{z_0 z_1}$ of $z_0$ and $z_1$. It is determined as follows. Consider the equilateral triangle with the line segment $z_0 z_1$ as one of its sides, and with its third corner to the right of $z_0 z_1$ (when looking from $z_0$ toward $z_1$). This third corner is referred to as the *equilateral point* and is denoted by $e_1$. Consider the circle $C(z_0, z_1, e_1)$ circumscribing this equilateral triangle. The arc from $z_0$ to $z_1$ (clockwise) is the Steiner arc $\widehat{z_0 z_1}$.

Possible locations of the Steiner point $s_2$ adjacent to $s_1$ and $z_2$ (with the long

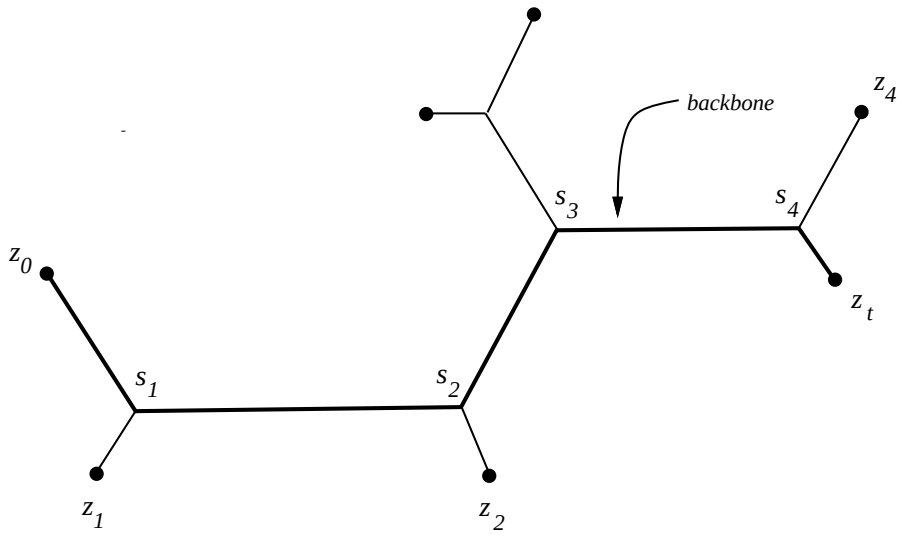



Figure 3: EFST

leg making a 60° left turn at $s_2$) are on the Steiner arc $\widehat{e_1 z_2}$. The corresponding equilateral point is denoted by $e_2$. The Steiner arc $\widehat{e_1 z_2}$ can be reduced; the edge connecting $s_1$ and $s_2$ must overlap with the line segment from $e_1$ to $s_2$. Hence, only projections of feasible locations of $s_1$ seen from $e_1$ are feasible locations of $s_2$ on $\widehat{e_1 z_2}$.

Steiner arcs for Steiner points $s_3, s_4, ..., s_k$ are determined in analogous manner. Consider the feasible subarc of the Steiner arc $\widehat{e_{k-1} z_k}$ ($k = 3$ in Figure 5). Let $e_k$ denote the associated equilateral point. Consider the region $R_k$ bounded by half-lines rooted at $e_k$ through the extreme points of the (pruned) Steiner arc $\widehat{e_{k-1} z_k}$ (Figure 5). Any terminal $z_t \in R_k \setminus C(e_{k-1}, z_k, e_k)$ yields an EFST. More precisely, the intersection of the line segment $z_t e_k$ with $\widehat{e_{k-1} z_k}$ is the location of $s_k$. Given the location of $s_k$, the location of $s_{k-1}$ is given as the intersection of the line-segment $s_k e_{k-1}$ with $\widehat{e_{k-2} z_{k-1}}$. Locations of $s_{k-2}, s_{k-3}, ..., s_1$ can be determined successively in the same manner.

It can be shown that the length of the EFST is equal to the length of the line-segment $z_t e_k$. Hence, the locations of Steiner points could in principle be determined only for EFSTs belonging to the ESMT.

Once all EFSTs for a given long leg have been identified, the long leg is extended by adding one more branch. If no extension is possible, the current long leg is redirected by requiring 60° right turn at $s_k$. This calls for the recomputation of $e_k$ (now to the left of the line segment from $e_{k-1}$ to $z_k$). Other equilateral points $e_1, e_2, ..., e_{k-1}$ are not affected. Once all extensions of this long leg have been considered, $z_k$ is replaced by another terminal. If all terminals have been

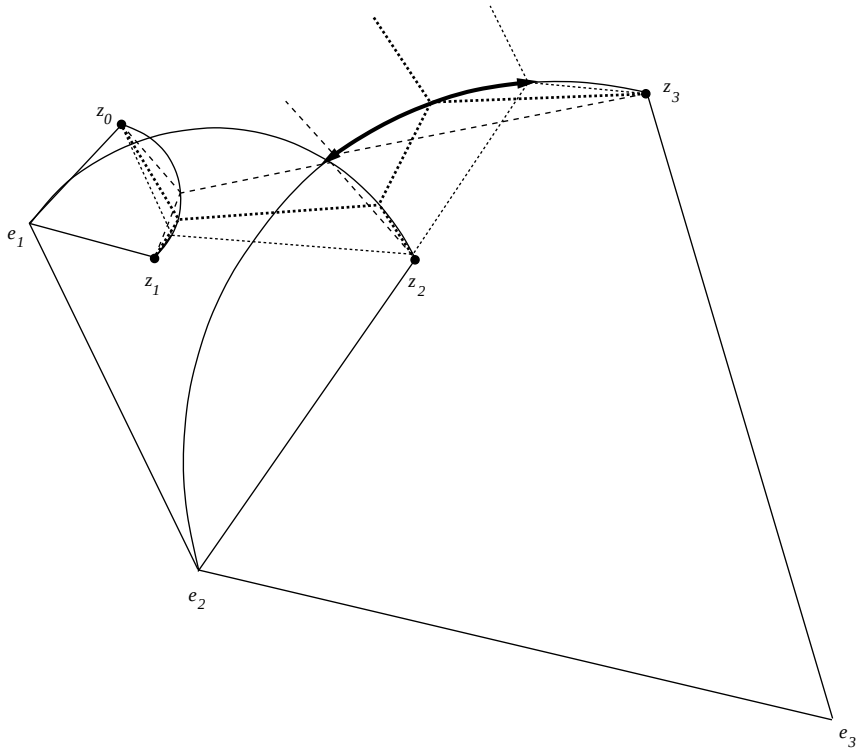



Figure 4: Long leg construction

considered, backtracking occurs.

The efficiency of the above approach stems from the fact that when long legs are expanded as well as when tips are attached, very powerful pruning tests can be applied. Most long legs never involve more than 2-3 Steiner points. Steiner arcs for these long legs are usually very narrow so that at most one or two tips can be attached (if any).

A serious complication when generating EFSTs is that it is not sufficient to grow long legs with 1-terminal branches. In fact, branches with arbitrary many terminals can be attached. These branches are long legs. We omit a detailed description of how such long legs are generated. Suffice it to say that whenever a long leg is not pruned away, it is saved. Long legs are extended either by attaching a terminal (as described above) or by attaching a saved disjoint long leg. Only very few EFSTs of the latter kind survive the pruning tests.

Another rather technical issue which we do not cover here is how to avoid multiple generation of the same EFST.

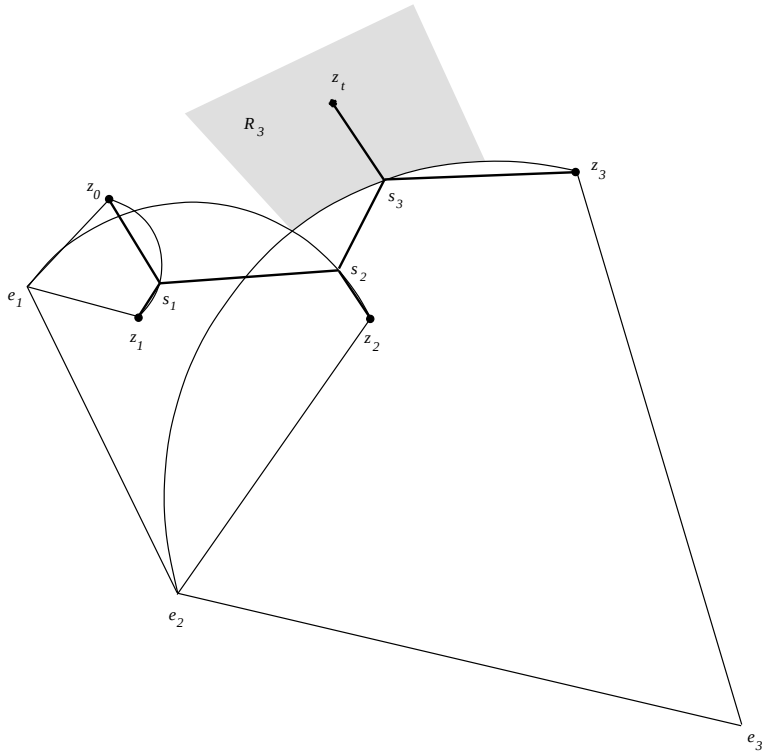



Figure 5: EFST construction

## 2.3 Pruning

In this subsection we give a brief description of some tests that make it possible to prune away the majority of FSTs. We focus on pruning tests that are common for both the Euclidean and the rectilinear case. The description is kept on a general level. The reader is referred to Winter and Zachariasen [26] and to Zachariasen [27] for details. Furthermore, some additional tests, not described below, can be found in these two papers.

### 2.3.1 Lune Property

A *lune* $L_{uv}$ of a line segment $uv$ is the intersection of two circles both with radius $\|uv\|_p$ and centred at $u$ and $v$, respectively (Figure 6). A necessary condition for the line segment $uv$ to be in any SMT is that its lune contains no terminals.

Edges in rectilinear backbones are known since the locations of Steiner points are fixed. As backbones are expanded, it is straightforward to check if the lunes of the end-points of added line segments are empty. The same applies when tips

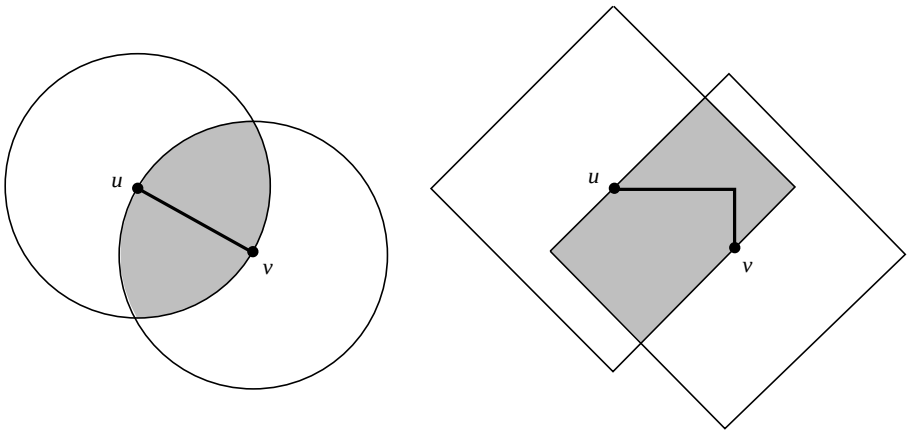



Figure 6: Euclidean and rectilinear lunes

(possibly with side branches) are added.

The situation is more complicated in the Euclidean case. Locations of Steiner points are not fixed. Fortunately, they are restricted to Steiner arcs. Consider the most recent Steiner arc of the long leg. If the corresponding Steiner point $s_k$ is assumed to be one of the extreme points of this Steiner arc, then the location of $s_{k-1}$ on the long leg can be determined. If one of the lunes $L_{s_k s_{k-1}}$ or $L_{s_k z_k}$ contains a terminal $z$, the Steiner arc $e\widehat{_{k-1}z_k}$ can be narrowed until none of them contains $z$.

Several other tests involving more complex empty regions are available, in particular for the rectilinear case. These tests will not be covered here; the reader is referred to Zachariasen [27] for details.

### 2.3.2 Bottleneck Steiner Distances

Construct an MST for the set of terminals $Z$. Let $b_{z_i z_j}$ denote the length of the longest edge on the unique path from a terminal $z_i$ to a terminal $z_j$. We refer to $b_{z_i z_j}$ as the *bottleneck Steiner distance*. Bottleneck Steiner distances for all pairs of terminals can be determined in $O(n^2)$ time. Consider an Euclidean or rectilinear SMT for $Z$. It is straightforward to show that no edge on the path between a pair of terminals $z_i$ and $z_j$ can be longer than $b_{z_i z_j}$.

When growing long legs in the rectilinear case, longest edges on paths from the terminal of the added branch to other terminals can be determined. If any of these edges has length greater than the corresponding bottleneck Steiner distance, the long leg can be pruned away. Similar arguments apply when a tip (possibly with a branch attached to the short leg) is added to the long leg.

The use of bottleneck Steiner distances when generating EFSTs is more complicated since the locations of Steiner points are not fixed. However, as already mentioned, they are restricted to Steiner arcs. It is therefore possible to determine good lower and upper bounds for the lengths of edges incident with Steiner points. If a lower bound for such an edge is greater than the bottleneck Steiner distance between a pair of terminals forced to use this edge, then the long leg can be pruned away. If an upper bound is greater than the bottleneck Steiner distance, then it is usually possible to narrow the Steiner arc (upper bounds are typically computed using the extreme points of Steiner arcs). It is outside the scope of this work to give a detailed description of how these bounds are determined and how the arcs are narrowed. The interested reader is referred to Winter and Zachariasen [26].

### 2.3.3 Upper Bounds

Several good heuristics for both the rectilinear and the Euclidean Steiner tree problems are available (see Hwang et al. [15]). They can be used to exclude some long legs and FSTs. Once again, the situation is simpler for the rectilinear case than for the Euclidean case.

Consider a rectilinear long leg with $s_k$ as the most recent Steiner point. Use one of the rectilinear heuristics to determine a tree spanning $s_k$ and the terminals attached to the long leg. If this tree is shorter than the tree given by the long leg, then the long leg cannot occur in the RSMT. Similarly, when a tip is added to the long leg (possibly with a branch attached to the short leg), and the resulting RFST can be shown to be non-optimal for its terminals (by generating a shorter tree using a heuristic), then the RFST can be pruned away. Similarly, if the MST for the terminals (using bottleneck Steiner distances between terminals) is shorter than the RFST, then the RFST can be pruned away. This test can be very efficient since bottleneck Steiner distances between pairs of terminals are often much smaller than $L_1$-distances.

Similar tests are available for the Euclidean case when a tip is added to a long leg. However, when a long leg is expanded, the location of the most recent Steiner point $s_k$ is not fixed, and the determination of upper bounds becomes more complicated. Once again, the reader is referred to Winter and Zachariasen [26] for a description of how upper bounds are determined and how they are used to prune away long legs or to narrow the Steiner arcs.

# 3 Concatenating Full Steiner Trees

In contrast to the FST generation problem, the FST concatenation problem is purely combinatorial and essentially metric-independent. Given a set of FSTs $\mathcal{F} = \{F_1, F_2, ..., F_m\}$, known to contain a subset whose union is an SMT for $Z$, the problem is to identify a subset $\mathcal{F}^* \subseteq \mathcal{F}$ such that the FSTs in $\mathcal{F}^*$

interconnect $Z$ and have minimum total length.

This concatenation problem has been solved using simple backtrack search (Section 3.1) and by applying dynamic programming (Section 3.2). However, backtrack search has a very steep running time growth and can only be applied for moderately sized problems. Dynamic programming has better worst-case running time properties, but is in practice even slower (and consumes much more memory).

Warme [24] presented an integer programming formulation for the concatenation problem which was solved by branch-and-cut (Section 3.3). Warme proved that the concatenation problem is equivalent to finding a minimum spanning tree (MST) in the hypergraph $H = (Z, \mathcal{F})$ and showed that the general hypergraph MST problem is NP-hard. This algorithm has increased the solvable range by more than an order of magnitude (see computational results in Section 4).

## 3.1 Backtrack Search

Backtrack search is a simple, yet reasonably effective "dumb" algorithm. Starting with a partial solution consisting of a single FST $F_i \in \mathcal{F}$, we seek the shortest-length tree interconnecting $Z$ and containing $F_i$. This is done by recursively adding FSTs to the solution until it interconnects $Z$ or it can be concluded that it cannot be optimal, e.g., if a cycle is created. In this case the search backtracks and some other FSTs are added. Obviously, it is only necessary to try FSTs spanning a particular terminal as the initial FST.

The cut-off tests applied during the search determine the practical behaviour of the algorithm, which otherwise runs in time $\Theta(2^m)$. Winter [25] used only relatively simple cut-off tests and observed that the concatenation phase began to dominate the generation phase already at $n \approx 15$. Cockayne and Hewgill [6, 7] improved the concatenation phase considerably by applying *problem decomposition*, *FST compatibility* and *FST pruning*. Similar ideas were also applied to the rectilinear problem by Salowe and Warme [20] but with substantially less success. More recently, Winter and Zachariasen [26] improved FST compatibility and pruning substantially and this allowed the exact solution of randomly generated Euclidean instances with up to 140 terminals (on average one hour on a workstation).

Problem decomposition reduces the initial concatenation problem to several smaller concatenation problems. Define $G(\mathcal{F})$ to be an undirected graph with $Z$ as its vertices; two nodes (terminals) $z_i$ and $z_j$ of $G(\mathcal{F})$ are adjacent if and only if there exists an FST in $\mathcal{F}$ spanning $z_i$ and $z_j$. If $G(\mathcal{F})$ has an articulation point, it can be split into two subproblems at that node [6] — each biconnected component corresponds to a subproblem for which concatenation may be done separately. More sophisticated decomposition methods, based on vertex separators, have been proposed [6, 20, 23]. Problem decomposition has been the single most important algorithmic improvement for concatenation based on backtrack search, but the effect of decomposition diminishes for larger problem instances,

in particular for the rectilinear problem. Also, for problem instances with special structure, such as regular lattices for the Euclidean problem, decomposition has virtually no effect.

The notion of *FST compatibility* was introduced by Cockayne and Hewgill [7] (a slightly different definition is given here). Two FSTs $F_i, F_j \in \mathcal{F}$ are *incompatible* if they cannot appear simultaneously in any SMT (e.g., if they span two or more common terminals), and otherwise they are *compatible*. A series of compatibility tests are applied to every pair of FSTs; the result is stored in a compatibility matrix which is used during backtrack search, as described later.

Let $F_i, F_j \in \mathcal{F}$ be two FSTs sharing at least one terminal. In the current implementation we used the following series of compatibility tests:

- $F_i$ and $F_j$ share at most one common terminal $z_c$.
- The angle between the two edges incident with $z_c$ is at least 120° (resp. 90°) for the Euclidean (resp. rectilinear) problem.
- $F_i \cup F_j$ passes longest edge test (described in Section 2.3.2).
- $F_i \cup F_j$ passes MST-BSD test (described in Section 2.3.3).

These tests can be performed with a fairly low computational overhead and they dominate many of the tests proposed in the literature, some of which are apparently metric-dependent [7, 20, 26].

The notion of FST compatibility can be used to remove FSTs from candidacy. Sophisticated variants of *FST pruning* can remove more than half of the initial list of FSTs [9, 26] at moderate computational effort — and have approximately doubled the solvable range. One effective pruning technique is the following: Consider an FST $F_i \in \mathcal{F}$ and the set $\mathcal{F}_i$ of FSTs compatible to this FST. If the graph $G(\{F_i\} \cup \mathcal{F}_i)$ is disconnected, then $F_i$ cannot be augmented to a tree spanning $Z$ and may be eliminated from further consideration.

Furthermore, FST compatibility can be used during backtrack search. At any point during the search only FSTs which are compatible to all FSTs in the current solution need to be considered. This reduces the search tree drastically without any significant computational overhead (recall that the compatibility matrix is computed in a preprocessing phase). Salowe and Warme [20] gave a strategy for scheduling the FSTs to add during the search in a "most promising" fashion; these ideas were elaborated by Warme [23]. However, the effect of using various scheduling techniques has been much less prominent than the effect of problem decomposition and FST compatibility.

## 3.2 Dynamic Programming

One of the earliest exact algorithms for the Steiner tree problem in graphs is the dynamic programming algorithm of Dreyfus and Wagner [8]. By reducing the rectilinear problem to the Steiner tree in graphs this algorithm solves

the rectilinear problem in time $O(n^2 3^n)$. Apart from the divide-and-conquer algorithm of Smith [21] which runs in time $n^{O(\sqrt{n})}$ (but with a huge constant factor), dynamic programming algorithms provide the best worst-case bounds for the rectilinear problem (the running time bounds given in this section do not generalize to the Euclidean problem as explained below).

Ganley and Cohoon [10] gave the first FST based dynamic programming algorithm. It uses the well-known fact that an SMT is either an FST or can split into two SMTs joined at a terminal; note that at least one of these smaller SMTs can again be assumed to be an FST. Subsets of $Z$ are enumerated in order of increasing cardinality — SMTs for each subset are computed and stored in a lookup table. Since rectilinear FSTs can be computed in linear time[1] finding the shortest joined SMT dominates the running time which is $O(n3^n)$. By proving an $O(n1.62^n)$ bound on the number of FSTs with Hwang topology, Ganley and Cohoon reduced the running time to $O(n^2 2.62^n)$. More recently, Fößmeier and Kaufmann [9] improved the former bound to $O(n1.38^n)$ by using additional necessary conditions on rectilinear FSTs, thereby improving the latter bound to $O(n^2 2.38^n)$.

Although dynamic programming provides the best worst-case bounds for the concatenation problem the practical behaviour seems to be inferior to backtrack search [9]. In addition, huge memory requirements make the approach impractical for instance sizes larger than $n \approx 40$.

These running time bounds do not generalize to the Euclidean problem, since no upper bound except from $O(2^n)$ is known for the total number of Euclidean FSTs. Furthermore, no polynomial time algorithm exists for computing a shortest FST for a set of terminals — this may be compared to the linear time algorithm for the rectilinear problem.

## 3.3 Integer Programming

In this section an integer programming formulation is given for the concatenation problem. Fundamental valid inequalities and polytope properties are presented and important details on the branch-and-cut algorithm are highlighted. For further details, we refer to the paper by Warme [24].

Consider a *hypergraph* $H = (Z, \mathcal{F})$ with the set of terminals $Z$ as its vertices and the set of FSTs $\mathcal{F}$ as its hyperedges, that is, each FST $F_i \in \mathcal{F}$ is considered to be a subset of $Z$ which is denoted by $Z(F_i)$. A *chain* in $H$ from $z_0 \in Z$ to $z_k \in Z$ is a sequence $z_0, F_1, z_1, F_2, z_2, \ldots, F_k, z_k$ such that all vertices and hyperedges are distinct and $z_{i-1}, z_i \in Z(F_i)$ for $i = 1, 2, \ldots, k$. A spanning tree in $H$ is is a subset of hyperedges $\mathcal{F}' \subseteq \mathcal{F}$ such that there is a *unique* chain between every pair of vertices $z_i, z_j \in Z$. The uniqueness implies that there can be no pair of hyperedges $F_i, F_j \in \mathcal{F}'$ that share two or more vertices, i.e., we

[1] Assuming that the terminals have been sorted in x- and y-direction in a preprocessing phase.

have $|Z(F_i) \cap Z(F_j)| \leq 1$ for all $F_i, F_j \in \mathcal{F}'$. The problem of finding a minimum spanning tree (MST) in $H$ where each hyperedge $F_i \in \mathcal{F}$ has weight equal to its length $|F_i|$ is equivalent to solving the concatenation problem.

The hypergraph MST problem is NP-hard, even when all hyperedges span at most a constant number $K \geq 3$ of vertices. In fact, Tomescu and Zimand [22] have shown that the existence of a spanning tree in a $K$-uniform hypergraph (in which all hyperedges span exactly $K$ vertices) is an NP-complete problem for $K \geq 3$. For $K = 2$ the problem reduces to the MST problem in ordinary graphs, which can be solved in polynomial time.

Now we give the integer programming (IP) formulation. Let $c$ be a vector in $\Re^m$ whose components are $c_i = |F_i|$. Denote by $x$ an $m$-dimensional *binary* vector. The IP formulation is then

$$\min cx \tag{1}$$

$$\text{s.t.} \quad \sum_{F_i \in \mathcal{F}} (|Z(F_i)| - 1)x_i = n - 1 \tag{2}$$

$$x_i + x_j \leq 1, \quad F_i \text{ incompatible with } F_j \tag{3}$$

$$\sum_{F_i \in \mathcal{F}} \max(0, |Z(F_i) \cap S| - 1)x_i \leq |S| - 1, \quad \forall S \subseteq Z,\ 2 \leq |S| < n \tag{4}$$

The objective (1) is to minimize the total length of the chosen FSTs subject to the following constraints: Equation (2) enforces the correct number and cardinality of hyperedges to construct a spanning tree. Constraints (3) add compatibility information to the formulation (Section 3.1). Finally, constraints (4) eliminate cycles by extending the standard notion of subtour elimination constraints; these constraints also ensure, in conjunction with equation (2), that the chosen FSTs interconnect $Z$.

Warme [24] proved several fundamental properties of the corresponding polytope. Let $\mathcal{K}_n$ be the complete hypergraph with $n$ vertices. This graph has $2^n - n - 1$ hyperedges (all hyperedges span two or more vertices). Let $ST_n \subset \{0,1\}^{2^n-n-1}$ denote the set of incidence vectors of spanning trees of $\mathcal{K}_n$. That is, each spanning tree of $\mathcal{K}_n$ defines a binary vector in which an element is 1 if and only if the corresponding hyperedge is chosen.

Then $conv(ST_n)$, the convex hull of the incidence vectors of all spanning trees, is the *spanning tree in hypergraph polytope*, STHGP($n$). Warme proved that $conv(ST_n)$ has dimension $2^n - n - 2$, that all spanning trees are extreme points and that the cycle elimination constraints (4) are facet defining for $n \geq 3$. In particular the last result explains the strength of this formulation (as evidenced by the computational results in Section 4).

The integer program is solved via branch-and-cut. Lower bounds are provided by linear programming (LP) relaxation, i.e., by relaxing integrality of every component $x_i$ of $x$ to $0 \leq x_i \leq 1$. Obviously the number of compatibility constraints (3) is bounded by $O(m^2)$, but the number of cycle elimination constraints (4) is exponential in $n$. The latter constraints are dynamically added

by separation methods. This separation problem can be solved in polynomial time (in $n$ and $m$) by finding minimum cuts in certain graphs [24]. However, heuristic separation methods are also used whenever applicable in order to speed up convergence to LP-optimum.

Enumeration is done using *best choice* branching. Let $x_i$ be a possible non-integral branch variable. Assume that the two LP-subproblems corresponding to $x_i = 0$ and $x_i = 1$ yield objective values $z_i^0$ and $z_i^1$, respectively. The $x_i$ that maximizes the value of $\min(z_i^0, z_i^1)$ is chosen as the new branch variable. Note that solving the $x_i = 1$ subproblem can be skipped if $z_i^0$ is already too low to be maximum. Furthermore, the process can terminate immediately if both $z_i^0$ and $z_i^1$ meet or exceed the best known integer solution.

Nodes in the branching tree are chosen using a *best node first* selection strategy, such that the outstanding node having the lowest objective value is processed next. This may (theoretically) produce a large number of outstanding nodes, but in practice the tightness of the formulation produces only a fairly small number of nodes.

# 4 Computational Experience

## 4.1 Experimental Conditions

The computational study was made on an HP9000 workstation[2]. The rectilinear and Euclidean FST generators were programmed in C++ using the class library LEDA version 3.4.1 [17]; the `random_source` class in LEDA was used for generating pseudo-random numbers. The FST concatenator was programmed in C using CPLEX version 5.0 to solve all the linear programming (LP) relaxations.

The test bed mainly consists of three sets of problem instances. The first set is a collection of 60 randomly generated instances from the *OR-Library* [1], 15 instances for each problem size 100, 250, 500 and 1000. All instances in the *OR-Library* with 100 and fewer terminals have previously been solved as Euclidean instances [26] and all instances with 1000 and fewer terminals have previously been solved as rectilinear instances [24].

The second set is a selection of 26 public library instances taken from *TSPLIB* [19] (ranging from 198 to 7397 terminals). These are the same instances as studied by Zachariasen [27]; none of these have previously been solved as rectilinear or Euclidean problems. *TSPLIB* is a collection of instances for the Traveling Salesman Problem (TSP), mainly plane real-world Euclidean problem instances. All instances are given as (the coordinates of) points in the plane.

Thirdly, the average behaviour of the exact algorithm is studied on a large set of randomly generated instances. Fifty instances were generated for each size

[2]Machine: HP 9000 Model C160. Processor: 160 MHz PA-RISC 8000. Main memory: 256 MB. Performance: 10.4 SPECint95 and 16.3 SPECfp95. Operating system: HP-UX 10.20. Compiler: GNU C++ 2.7.2.1 (optimization flag -O3).

100, 200, 300, 400 and 500. Terminals were drawn with uniform distribution from the unit square.

Computational results on FST generation, FST concatenation and SMT properties for these three sets are presented in Sections 4.2, 4.3 and 4.4, respectively. In addition, we present results for two small sets of instances (one for the rectilinear and one for the Euclidean problem) for which the full Steiner tree approach performs relatively badly — at least when compared to the average case. These results are given in Section 4.4.

In order to ease the comparison between the rectilinear and the Euclidean problem, statistics for both problems are presented in the same table. Also, we use the same table layout for each of the three main instance classes. However, while results for *OR-Library* and *TSPLIB* instances are presented for each instance, only averages (and standard deviations) are given for the randomly generated instances. Thus it is possible both to study algorithmic behaviour and solution properties for specific instances and to identify general tendencies for increasing problem sizes.

## 4.2 Generation

As observed in previous studies (e.g. [20, 26]) the number of FSTs surviving the pruning tests described in Section 2 is almost *linear*. For randomly generated instances (Table 1 and 3; Figure 7), approximately $4.0n$ rectilinear and $2.3n$ Euclidean FSTs survive. For problems with more structure (in particular instances with many co-linear and equidistant terminals) *fewer* rectilinear FSTs survive (Table 2). These instances are on the other hand very difficult to solve as Euclidean problems; for many of these instances the Euclidean FST generation did not finish within the allotted CPU time of approximately one week.

Generated rectilinear FSTs on average span less than four terminals while largest FSTs span 14 terminals (Table 3). Euclidean FSTs on average only span three terminals and at most 9 terminals. Figure 10 illustrates this more clearly: For a given FST size, approximately twice as many rectilinear FSTs are generated — the ratio increases for larger FSTs.

We also present statistics on incompatibility based on the tests described in Section 3.1. These tests are essentially metric-independent. Recall that two FSTs are incompatible if they cannot appear simultaneously in any SMT and that tests for incompatibility are only applied to FST pairs which share at least one terminal. The percentage of FST pairs sharing at least one terminal which are incompatible will be used as "measure of incompatibility". Interestingly, this measure is extremely independent of the metric (rectilinear/Euclidean), terminal set distribution and instance size. More than two-thirds of all FST pairs sharing at least one terminal are incompatible. The effect of adding incompatibility constraints to the integer programming formulation is small compared to the speed-up obtained when using compatibility in conjunction with backtrack search. In some cases it is even negative since larger LPs need to be solved.

| $n$ | | Rectilinear | | | | | Euclidean | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Count | Size | | Incomp | CPU | Count | Size | | Incomp | CPU |
| 100 | (1) | 397 | 3.60 | (11) | 66.95 | 1.9 | 239 | 3.05 | (7) | 69.37 | 208.9 |
| 100 | (2) | 508 | 3.84 | (17) | 66.48 | 4.6 | 239 | 2.97 | (8) | 65.86 | 264.7 |
| 100 | (3) | 331 | 3.42 | (12) | 68.98 | 1.3 | 219 | 2.97 | (8) | 66.34 | 214.7 |
| 100 | (4) | 349 | 3.48 | (10) | 71.42 | 1.4 | 231 | 2.96 | (6) | 70.67 | 193.2 |
| 100 | (5) | 337 | 3.41 | (9) | 67.15 | 1.4 | 211 | 2.85 | (5) | 65.51 | 177.5 |
| 100 | (6) | 495 | 3.91 | (12) | 67.87 | 4.9 | 225 | 3.05 | (8) | 65.80 | 233.7 |
| 100 | (7) | 490 | 3.89 | (11) | 69.22 | 4.1 | 245 | 2.99 | (7) | 64.46 | 333.1 |
| 100 | (8) | 361 | 3.51 | (9) | 69.32 | 1.5 | 221 | 2.96 | (8) | 68.14 | 177.2 |
| 100 | (9) | 396 | 3.69 | (11) | 68.24 | 2.6 | 237 | 2.95 | (7) | 67.38 | 206.1 |
| 100 | (10) | 365 | 3.46 | (9) | 65.21 | 1.7 | 225 | 2.93 | (6) | 63.49 | 188.3 |
| 100 | (11) | 336 | 3.28 | (8) | 67.14 | 1.2 | 235 | 3.01 | (6) | 65.89 | 153.8 |
| 100 | (12) | 357 | 3.37 | (9) | 65.32 | 1.4 | 210 | 2.82 | (6) | 66.74 | 114.7 |
| 100 | (13) | 401 | 3.87 | (13) | 72.13 | 3.0 | 221 | 2.95 | (8) | 68.30 | 135.3 |
| 100 | (14) | 312 | 3.15 | (9) | 63.44 | 0.8 | 224 | 2.92 | (8) | 66.38 | 179.9 |
| 100 | (15) | 356 | 3.39 | (10) | 67.90 | 1.4 | 233 | 3.09 | (7) | 68.05 | 240.8 |
| 250 | (1) | 936 | 3.65 | (21) | 69.00 | 6.5 | 550 | 2.87 | (7) | 65.00 | 1303.4 |
| 250 | (2) | 889 | 3.34 | (10) | 65.35 | 3.9 | 527 | 2.84 | (8) | 64.74 | 935.3 |
| 250 | (3) | 870 | 3.37 | (12) | 66.53 | 3.8 | 575 | 3.01 | (9) | 71.43 | 1279.2 |
| 250 | (4) | 911 | 3.48 | (11) | 68.34 | 4.4 | 537 | 2.84 | (7) | 63.29 | 1129.4 |
| 250 | (5) | 914 | 3.45 | (11) | 66.16 | 4.3 | 537 | 2.88 | (8) | 67.53 | 1031.2 |
| 250 | (6) | 909 | 3.40 | (10) | 66.76 | 4.1 | 501 | 2.80 | (7) | 65.22 | 975.3 |
| 250 | (7) | 914 | 3.47 | (11) | 68.21 | 4.0 | 533 | 2.87 | (7) | 65.36 | 1054.2 |
| 250 | (8) | 1102 | 3.65 | (17) | 66.81 | 8.2 | 631 | 3.09 | (7) | 67.46 | 2046.8 |
| 250 | (9) | 922 | 3.43 | (12) | 66.01 | 4.2 | 537 | 2.85 | (8) | 64.00 | 1228.6 |
| 250 | (10) | 1139 | 3.72 | (14) | 67.60 | 8.8 | 573 | 2.99 | (9) | 66.47 | 1518.6 |
| 250 | (11) | 980 | 3.67 | (17) | 65.74 | 7.0 | 590 | 3.00 | (8) | 66.31 | 1280.4 |
| 250 | (12) | 951 | 3.62 | (18) | 68.77 | 5.7 | 557 | 2.89 | (7) | 66.23 | 1085.7 |
| 250 | (13) | 1009 | 3.49 | (9) | 66.43 | 5.1 | 627 | 3.06 | (8) | 67.83 | 1565.5 |
| 250 | (14) | 992 | 3.51 | (12) | 67.82 | 5.3 | 575 | 2.98 | (7) | 68.28 | 1152.3 |
| 250 | (15) | 988 | 3.53 | (12) | 65.47 | 6.0 | 574 | 2.99 | (8) | 68.11 | 1301.6 |
| 500 | (1) | 1927 | 3.57 | (15) | 67.10 | 14.0 | 1271 | 3.16 | (10) | 71.51 | 5461.8 |
| 500 | (2) | 2249 | 3.86 | (15) | 69.79 | 20.7 | 1184 | 2.98 | (7) | 68.15 | 6620.6 |
| 500 | (3) | 2159 | 3.77 | (15) | 69.51 | 15.8 | 1248 | 3.14 | (10) | 70.22 | 6572.3 |
| 500 | (4) | 1891 | 3.62 | (15) | 68.04 | 11.1 | 1159 | 3.01 | (8) | 66.39 | 5768.3 |
| 500 | (5) | 1875 | 3.46 | (15) | 68.03 | 9.5 | 1080 | 2.89 | (7) | 67.12 | 4112.7 |
| 500 | (6) | 2070 | 3.68 | (14) | 67.40 | 14.3 | 1191 | 3.06 | (9) | 69.22 | 6378.0 |
| 500 | (7) | 1953 | 3.51 | (10) | 67.61 | 10.1 | 1118 | 2.91 | (7) | 65.42 | 4738.5 |
| 500 | (8) | 2027 | 3.59 | (14) | 67.95 | 12.0 | 1092 | 2.92 | (8) | 65.78 | 5027.0 |
| 500 | (9) | 2011 | 3.63 | (13) | 66.91 | 13.2 | 1191 | 3.01 | (8) | 69.30 | 5445.7 |
| 500 | (10) | 1953 | 3.48 | (11) | 67.45 | 10.3 | 1208 | 3.00 | (8) | 67.20 | 5326.4 |
| 500 | (11) | 2068 | 3.55 | (12) | 67.21 | 12.2 | 1150 | 2.96 | (8) | 67.52 | 6330.9 |
| 500 | (12) | 1912 | 3.59 | (13) | 68.77 | 12.2 | 1086 | 2.88 | (8) | 65.69 | 5255.1 |
| 500 | (13) | 1879 | 3.48 | (17) | 66.71 | 9.6 | 1077 | 2.86 | (6) | 65.10 | 4085.0 |
| 500 | (14) | 2126 | 3.68 | (17) | 67.71 | 14.3 | 1305 | 3.14 | (10) | 69.40 | 7511.8 |
| 500 | (15) | 1962 | 3.61 | (13) | 67.24 | 12.4 | 1176 | 2.99 | (9) | 68.48 | 5277.4 |
| 1000 | (1) | 4176 | 3.70 | (12) | 68.70 | 31.1 | 2197 | 2.90 | (9) | 65.26 | 20933.1 |
| 1000 | (2) | 4012 | 3.61 | (17) | 67.75 | 25.5 | 2280 | 2.96 | (10) | 66.73 | 23147.0 |
| 1000 | (3) | 4023 | 3.58 | (16) | 67.11 | 26.7 | 2185 | 2.93 | (9) | 65.98 | 22887.5 |
| 1000 | (4) | 4092 | 3.62 | (18) | 67.96 | 26.8 | 2476 | 3.07 | (10) | 69.53 | 24932.6 |
| 1000 | (5) | 4025 | 3.69 | (16) | 67.73 | 32.5 | 2214 | 2.94 | (10) | 67.11 | 23218.7 |
| 1000 | (6) | 4336 | 3.81 | (21) | 69.91 | 31.8 | 2313 | 2.98 | (10) | 67.80 | 24286.8 |
| 1000 | (7) | 3998 | 3.64 | (14) | 68.50 | 29.3 | 2236 | 2.91 | (8) | 66.48 | 20443.2 |
| 1000 | (8) | 4294 | 3.77 | (17) | 69.10 | 35.2 | 2266 | 2.92 | (8) | 66.05 | 23819.0 |
| 1000 | (9) | 4481 | 3.82 | (14) | 68.41 | 43.4 | 2393 | 3.01 | (9) | 67.72 | 26676.5 |
| 1000 | (10) | 4002 | 3.56 | (15) | 67.97 | 26.4 | 2239 | 2.92 | (9) | 67.17 | 20917.7 |
| 1000 | (11) | 4042 | 3.62 | (20) | 67.63 | 30.0 | 2311 | 2.96 | (10) | 67.16 | 23504.0 |
| 1000 | (12) | 4690 | 3.95 | (18) | 69.75 | 48.9 | 2430 | 3.03 | (8) | 67.97 | 28115.9 |
| 1000 | (13) | 3876 | 3.58 | (16) | 68.72 | 26.3 | 2253 | 2.94 | (9) | 67.61 | 19564.7 |
| 1000 | (14) | 4336 | 3.75 | (14) | 68.77 | 32.6 | 2309 | 2.97 | (10) | 66.33 | 25001.8 |
| 1000 | (15) | 4151 | 3.63 | (14) | 68.34 | 28.3 | 2331 | 2.99 | (10) | 68.62 | 23183.7 |

Table 1: FST-generation, *OR-library* instances. Count: Number of FSTs generated, including MST-edges. Size: Average number of terminals spanned in generated FSTs (resp. maximum number of terminals spanned). Incomp: Percentage of FST pairs sharing at least one terminal which are incompatible. CPU: CPU-time (seconds).

| Instance | Rectilinear | | | | | Euclidean | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Count | Size | | Incomp | CPU | Count | Size | | Incomp | CPU |
| d198 | 275 | 2.38 | (5) | 37.08 | 0.6 | 807 | 4.95 | (21) | 87.74 | 4153.9 |
| lin318 | 998 | 3.32 | (10) | 63.28 | 4.5 | 1235 | 3.54 | (8) | 71.84 | 10299.9 |
| fl417 | 1192 | 2.93 | (17) | 46.94 | 4.9 | 3715 | 6.84 | (24) | 82.12 | 81803.9 |
| pcb442 | 558 | 2.27 | (7) | 39.43 | 1.2 | - | - | - | - | - |
| att532 | 2239 | 3.76 | (16) | 69.76 | 19.4 | 1246 | 3.02 | (9) | 67.92 | 7403.8 |
| u574 | 1506 | 3.06 | (9) | 62.31 | 4.9 | 1340 | 2.98 | (10) | 68.66 | 5347.9 |
| p654 | 933 | 2.55 | (11) | 57.84 | 3.2 | - | - | - | - | - |
| rat783 | 3899 | 4.02 | (25) | 68.31 | 45.7 | 1954 | 3.15 | (11) | 69.27 | 21258.2 |
| pr1002 | 2198 | 2.90 | (11) | 59.18 | 7.6 | 2322 | 2.96 | (8) | 66.92 | 15447.9 |
| u1060 | 2818 | 3.04 | (9) | 59.04 | 11.9 | 3630 | 4.10 | (13) | 77.00 | 59230.9 |
| pcb1173 | 3001 | 3.07 | (12) | 65.67 | 11.5 | 3052 | 4.61 | (36) | 89.96 | 72887.3 |
| d1291 | 1393 | 2.09 | (7) | 15.51 | 5.8 | - | - | - | - | - |
| rl1323 | 1957 | 2.49 | (9) | 52.63 | 7.7 | 2351 | 2.67 | (7) | 71.72 | 8569.4 |
| fl1400 | 5870 | 3.02 | (11) | 40.61 | 32.7 | - | - | - | - | - |
| u1432 | 1431 | 2.00 | (2) | 0.00 | 4.8 | - | - | - | - | - |
| fl1577 | 3822 | 2.71 | (7) | 47.40 | 12.5 | - | - | - | - | - |
| d1655 | 2219 | 2.34 | (7) | 44.85 | 7.7 | - | - | - | - | - |
| vm1748 | 4015 | 3.12 | (15) | 66.59 | 18.1 | 3329 | 2.77 | (10) | 66.56 | 28653.0 |
| rl1889 | 2920 | 2.56 | (11) | 58.98 | 11.3 | 3579 | 3.13 | (16) | 82.81 | 33193.8 |
| u2152 | 2173 | 2.01 | (3) | 1.53 | 9.6 | - | - | - | - | - |
| pr2392 | 4792 | 2.83 | (9) | 55.59 | 16.1 | 5923 | 3.12 | (10) | 71.48 | 101522.3 |
| pcb3038 | 9356 | 3.28 | (12) | 64.48 | 44.1 | - | - | - | - | - |
| fl3795 | 7770 | 2.62 | (7) | 36.28 | 45.7 | - | - | - | - | - |
| fnl4461 | 27959 | 4.59 | (34) | 71.05 | 543.5 | 12246 | 3.19 | (12) | 67.90 | 965730.9 |
| rl5934 | 8168 | 2.38 | (8) | 49.15 | 71.0 | - | - | - | - | - |
| pla7397 | 10595 | 2.46 | (9) | 53.84 | 100.9 | - | - | - | - | - |

Table 2: FST-generation, *TSPLIB* instances. See Table 1 for captions.

| $n$ | Rectilinear | | | | | Euclidean | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Count | Size | | Incomp | CPU | Count | Size | | Incomp | CPU |
| 100 | 363.3 | 3.42 | (9.6) | 67.44 | 1.7 | 220.0 | 2.92 | (7.0) | 66.81 | 162.3 |
| | ±47.7 | ±0.21 | ±1.7 | ±2.53 | ±0.8 | ±21.7 | ±0.16 | ±1.4 | ±3.79 | ±49.6 |
| 200 | 797.4 | 3.60 | (12.3) | 68.08 | 5.0 | 454.5 | 2.95 | (7.7) | 66.61 | 781.0 |
| | ±78.9 | ±0.18 | ±2.9 | ±2.20 | ±1.8 | ±33.2 | ±0.12 | ±1.2 | ±2.93 | ±202.2 |
| 300 | 1196.6 | 3.62 | (13.4) | 68.24 | 7.8 | 671.8 | 2.94 | (8.0) | 66.75 | 1740.7 |
| | ±90.1 | ±0.15 | ±2.6 | ±1.68 | ±2.5 | ±36.4 | ±0.09 | ±1.1 | ±1.96 | ±394.9 |
| 400 | 1599.9 | 3.61 | (13.6) | 68.10 | 10.7 | 910.3 | 2.96 | (8.3) | 67.20 | 3171.4 |
| | ±127.1 | ±0.15 | ±2.3 | ±1.62 | ±3.7 | ±36.9 | ±0.07 | ±1.1 | ±1.49 | ±593.3 |
| 500 | 1997.6 | 3.62 | (13.9) | 68.30 | 13.5 | 1139.8 | 2.96 | (8.6) | 67.16 | 5194.8 |
| | ±125.0 | ±0.12 | ±2.3 | ±1.30 | ±3.6 | ±55.7 | ±0.09 | ±1.4 | ±1.86 | ±985.4 |

Table 3: FST-generation, randomly generated instances. Averages over 50 instances for each size; standard deviations on the second line of each row. See Table 1 for captions.

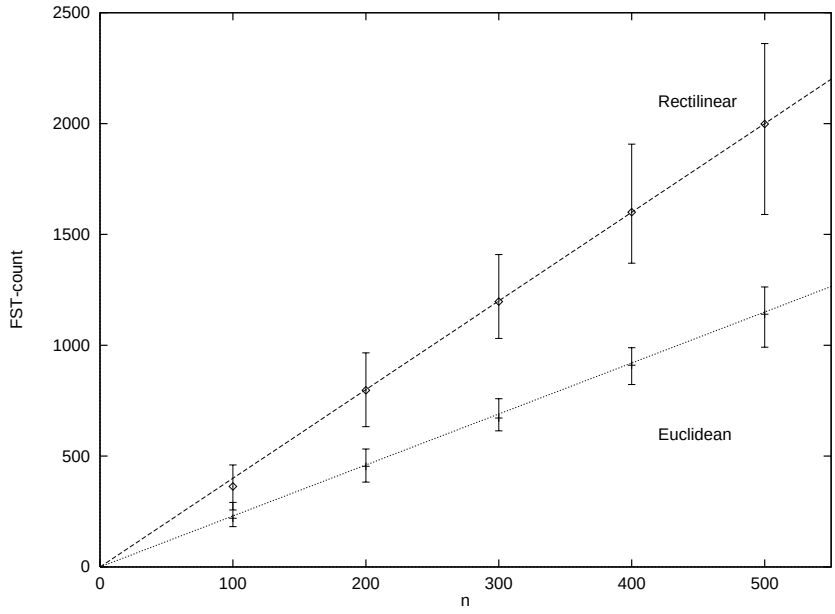



Figure 7: FST generation, randomly generated instances. Total FST-count, including MST-edges; average, minimum and maximum.

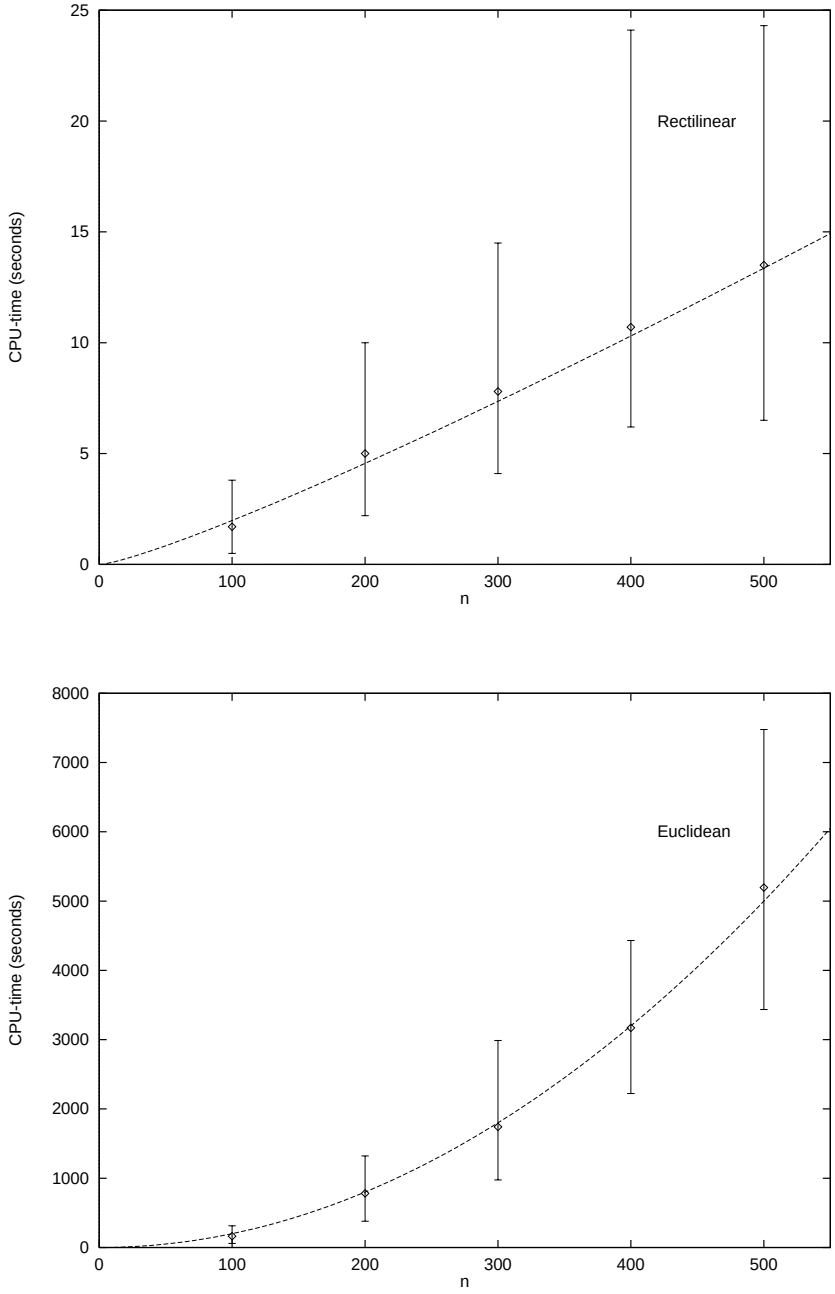



Figure 8: FST generation, randomly generated instances. CPU-time (seconds); average, minimum and maximum. Note different scaling on $y$-axes.

Another measure of incompatibility could have been the number of incompatible FST pairs normalized by either $n$ or $m$ (number of FSTs) — or by the square of these numbers. However, this would not have given a good basis for comparing the two problem variants because of the larger number of rectilinear FSTs.

Rectilinear FSTs are generated in a fraction of the CPU-time needed for generating Euclidean FSTs (more than 100 times faster). However, as noted in Section 2, the generation of Euclidean FSTs is much more complicated and involves heavy use of floating point operations and trigonometric functions. When comparing the CPU-times with those given in [26, 27], the following should be noted: First of all a faster workstation has been used in this study (20-30% faster). The code for the Euclidean generator has been optimized to make fewer calls to trigonometric functions and the upper bounding procedure has been improved. These improvements have reduced the running time by 30-40%. The running times reported for the rectilinear generator are *greater* than those reported in [27]. This comes from the fact that incompatibility information has been computed in this study. This increases the running time by a factor of approximately five — thus the computation of incompatibility information completely dominates the generation of rectilinear FSTs.

### 4.3 Concatenation

Statistics on the performance of the branch-and-cut algorithm for solving the FST concatenation problem are given in Table 4, 5, 6 and Figure 9. We present root LP/optimal IP value gap, number of branch-and-bound nodes and total number of LPs solved, i.e., number of separation iterations.

The strength of the LP relaxation is indicated by a very small LP/IP gap; for many instances the root LP solution is integral and no branching is needed. Only two instances have a gap of more than 0.1% and few instances require more than 10 branch-and-bound nodes. However, the computational effort required to solve the FST concatenation problem using integer programming shows a very large variance compared to the FST generation algorithms. The running time growth is clearly exponential (Figure 9).

One subtle observation regarding Table 4 is that even though the rectilinear FST generator used in the current study generates significantly fewer FSTs than the FST generator by Warme [24], this does not necessarily have a positive effect on the time to solve the concatenation problem. For many instances the branch-and-cut algorithm requires *more* separation iterations (LPs) and branch-and-bound nodes (Nds). The reason for this behaviour is as yet unclear.

Out of 26 instances in the TSPLIB selection, 19 have been solved as rectilinear problems and 13 as Euclidean problems using a cut-off of approximately one week. Instance `pr2392` was the largest instance solved, both for the Euclidean and rectilinear problem. It is interesting to note that instances with many co-linear and equidistant terminals are easy to solve as rectilinear prob-

lems while instances with a more uniform and less structured distribution are easier to solve as Euclidean problems. Also note that all Euclidean instances for which FSTs have been generated are solved using only a few hours for the concatenation (except for `fnl4461` which was not solved within one week).

## 4.4 Steiner Minimum Tree Properties

In this section we present some structural properties of rectilinear and Euclidean SMTs (Table 7, 8, 9 and Figure 10). Similar statistics have been presented for Euclidean SMTs spanning up to 150 terminals [26]. No such statistics have previously been given for rectilinear SMTs spanning more than 50 terminals.

The number of FSTs in an average RSMT spanning $n$ terminals is approximately $0.5n$ while an ESMT has approximately $0.6n$ FSTs. An ESMT has more MST-edges, $0.3n$, compared to $0.2n$ in an RSMT. This difference is also reflected in the size of the FSTs. RFSTs on average span 2.95 terminals while EFSTs span 2.70 terminals; furthermore, for $n = 500$ the largest RFST spans 7 terminals and the largest EFST 6 terminals (see also Figure 10). The size of the largest FST grows very slowly for both problems, apparently with a growth that is $o(\log n)$. For large instances ($n = 500$) the reduction over the MST is 11.5% for the rectilinear problem and 3.3% for the Euclidean problem.

In order to test the limits of FST based exact algorithms for plane Steiner tree problems, we performed a series of tests on seemingly "difficult" instances. Fößmeier and Kaufmann [9] constructed an infinite series of (rectilinear) instances for which the number of FSTs fulfilling a so-called *tree star* condition is *exponential*. Zachariasen [27] noted that the rectilinear FST generator actually produced a super-polynomial number of FSTs for this series of instances. In Table 10 we present statistics on the first five instances in this series (12-52 terminals). The number of FSTs and the total CPU-time grows rapidly although the structure of the optimal solutions (number of FSTs and average size) does not differ radically from randomly generated instances. The total CPU-time needed to solve the 52 terminal instance is more than 10 minutes; this is almost 200 times the CPU-time needed to solve a randomly generated 100 terminal instance (Table 9).

As previously noted [26], regular lattices are difficult to solve as Euclidean problems using FST based exact algorithms. Polynomial time algorithms for regular lattice problems have recently been given by Brazil et al. [3, 4, 5]. In Table 11 we present data for solving regular lattices spanning $2 \times 7$, $3 \times 7$, ..., $7 \times 7$ terminals. Again the CPU-times are orders of magnitude larger than for solving randomly generated instances of similar size. In particular, it is interesting to note that the LP relaxation is much weaker for these instances and that relatively heavy branching is required. One explanation is the large number of symmetric near-optimal solutions.

| $n$ | | Rectilinear | | | | Euclidean | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Gap | Nds | LPs | CPU | Gap | Nds | LPs | CPU |
| 100 | (1) | 0.000 | 1 | 33 | 5.1 | 0.000 | 1 | 4 | 0.5 |
| 100 | (2) | 0.000 | 1 | 15 | 2.4 | 0.000 | 1 | 8 | 0.8 |
| 100 | (3) | 0.000 | 1 | 22 | 2.8 | 0.000 | 1 | 30 | 1.6 |
| 100 | (4) | 0.000 | 1 | 7 | 1.1 | 0.000 | 1 | 3 | 0.2 |
| 100 | (5) | 0.000 | 1 | 6 | 0.8 | 0.000 | 1 | 2 | 0.2 |
| 100 | (6) | 0.000 | 1 | 7 | 6.0 | 0.000 | 2 | 6 | 0.4 |
| 100 | (7) | 0.000 | 1 | 55 | 6.4 | 0.000 | 1 | 3 | 0.6 |
| 100 | (8) | 0.001 | 1 | 22 | 3.2 | 0.000 | 1 | 3 | 0.5 |
| 100 | (9) | 0.000 | 1 | 7 | 4.4 | 0.000 | 1 | 26 | 1.3 |
| 100 | (10) | 0.000 | 1 | 22 | 2.6 | 0.000 | 1 | 6 | 0.5 |
| 100 | (11) | 0.109 | 5 | 20 | 2.2 | 0.000 | 1 | 3 | 0.3 |
| 100 | (12) | 0.000 | 1 | 27 | 1.9 | 0.000 | 1 | 3 | 0.3 |
| 100 | (13) | 0.000 | 1 | 8 | 3.0 | 0.000 | 1 | 3 | 0.3 |
| 100 | (14) | 0.046 | 2 | 72 | 5.3 | 0.000 | 1 | 4 | 0.7 |
| 100 | (15) | 0.000 | 1 | 10 | 4.4 | 0.000 | 1 | 4 | 0.4 |
| 250 | (1) | 0.000 | 1 | 24 | 7.8 | 0.000 | 3 | 10 | 2.2 |
| 250 | (2) | 0.009 | 3 | 60 | 18.7 | 0.000 | 1 | 12 | 2.3 |
| 250 | (3) | 0.000 | 1 | 102 | 30.7 | 0.000 | 1 | 42 | 4.5 |
| 250 | (4) | 0.013 | 2 | 36 | 22.7 | 0.000 | 2 | 22 | 3.3 |
| 250 | (5) | 0.000 | 1 | 272 | 37.4 | 0.000 | 1 | 129 | 23.1 |
| 250 | (6) | 0.003 | 2 | 110 | 44.3 | 0.003 | 5 | 12 | 2.1 |
| 250 | (7) | 0.000 | 1 | 44 | 17.1 | 0.000 | 1 | 4 | 1.6 |
| 250 | (8) | 0.053 | 5 | 41 | 26.1 | 0.000 | 1 | 18 | 4.8 |
| 250 | (9) | 0.000 | 1 | 604 | 242.7 | 0.000 | 3 | 23 | 2.6 |
| 250 | (10) | 0.060 | 3 | 95 | 63.8 | 0.000 | 3 | 30 | 4.0 |
| 250 | (11) | 0.017 | 3 | 32 | 18.9 | 0.000 | 1 | 6 | 1.7 |
| 250 | (12) | 0.005 | 2 | 58 | 38.3 | 0.000 | 4 | 11 | 3.1 |
| 250 | (13) | 0.028 | 8 | 325 | 330.7 | 0.000 | 2 | 34 | 6.2 |
| 250 | (14) | 0.000 | 1 | 12 | 9.8 | 0.001 | 1 | 85 | 7.4 |
| 250 | (15) | 0.000 | 1 | 58 | 21.4 | 0.000 | 1 | 38 | 4.0 |
| 500 | (1) | 0.004 | 2 | 38 | 65.1 | 0.000 | 1 | 52 | 17.4 |
| 500 | (2) | 0.009 | 1 | 50 | 63.8 | 0.004 | 2 | 68 | 28.9 |
| 500 | (3) | 0.000 | 1 | 216 | 966.4 | 0.000 | 1 | 180 | 138.7 |
| 500 | (4) | 0.000 | 1 | 169 | 367.3 | 0.000 | 3 | 182 | 52.3 |
| 500 | (5) | 0.034 | 9 | 455 | 2112.7 | 0.000 | 2 | 210 | 105.0 |
| 500 | (6) | 0.000 | 1 | 28 | 48.6 | 0.000 | 1 | 47 | 22.0 |
| 500 | (7) | 0.006 | 1 | 31 | 66.1 | 0.000 | 3 | 50 | 15.2 |
| 500 | (8) | 0.001 | 1 | 142 | 505.8 | 0.000 | 2 | 80 | 37.6 |
| 500 | (9) | 0.017 | 4 | 52 | 80.9 | 0.000 | 4 | 60 | 47.9 |
| 500 | (10) | 0.000 | 1 | 66 | 98.9 | 0.000 | 2 | 24 | 12.9 |
| 500 | (11) | 0.000 | 2 | 206 | 1186.4 | 0.007 | 1 | 215 | 158.0 |
| 500 | (12) | 0.002 | 2 | 70 | 184.9 | 0.000 | 2 | 284 | 118.3 |
| 500 | (13) | 0.005 | 2 | 52 | 86.7 | 0.000 | 2 | 52 | 24.3 |
| 500 | (14) | 0.038 | 4 | 112 | 474.7 | 0.000 | 1 | 75 | 46.7 |
| 500 | (15) | 0.007 | 2 | 44 | 40.0 | 0.000 | 2 | 615 | 396.5 |
| 1000 | (1) | 0.047 | 25 | 232 | 2780.9 | 0.000 | 4 | 93 | 117.2 |
| 1000 | (2) | 0.000 | 2 | 124 | 616.5 | 0.000 | 2 | 299 | 347.9 |
| 1000 | (3) | 0.001 | 2 | 1099 | 34313.0 | 0.000 | 1 | 27 | 31.9 |
| 1000 | (4) | 0.011 | 7 | 490 | 1172.3 | 0.000 | 1 | 157 | 107.8 |
| 1000 | (5) | 0.004 | 3 | 1330 | 34970.7 | 0.000 | 1 | 66 | 73.0 |
| 1000 | (6) | 0.032 | 16 | 5560 | 351933.9 | 0.000 | 2 | 328 | 3006.8 |
| 1000 | (7) | 0.006 | 2 | 1310 | 39551.3 | 0.001 | 5 | 56 | 72.3 |
| 1000 | (8) | 0.005 | 5 | 743 | 28679.8 | 0.003 | 5 | 1160 | 21956.0 |
| 1000 | (9) | 0.004 | 5 | 86 | 544.4 | 0.000 | 2 | 60 | 102.4 |
| 1000 | (10) | 0.013 | 6 | 1202 | 48770.7 | 0.000 | 2 | 33 | 44.4 |
| 1000 | (11) | 0.012 | 5 | 485 | 1341.6 | 0.000 | 1 | 114 | 72.1 |
| 1000 | (12) | 0.010 | 4 | 2107 | 214567.0 | 0.000 | 2 | 503 | 513.6 |
| 1000 | (13) | 0.000 | 1 | 4300 | 263917.3 | 0.000 | 1 | 35 | 35.0 |
| 1000 | (14) | 0.022 | 17 | 5416 | 540376.5 | 0.000 | 2 | 448 | 1579.3 |
| 1000 | (15) | 0.010 | 2 | 285 | 2701.5 | 0.000 | 2 | 472 | 569.5 |

Table 4: FST-concatenation, *OR-library* instances. Gap: Root LP objective value vs. optimal value (gap in percent). Nds: Number of branch-and-bound nodes. LPs: Number of LPs solved. CPU: CPU-time (seconds).

| Instance | Rectilinear | | | | Euclidean | | | |
|---|---|---|---|---|---|---|---|---|
| | Gap | Nds | LPs | CPU | Gap | Nds | LPs | CPU |
| d198 | 0.000 | 1 | 63 | 4.4 | 0.000 | 1 | 47 | 10.4 |
| lin318 | 0.046 | 7 | 130 | 451.7 | 0.000 | 1 | 154 | 499.8 |
| fl417 | 0.018 | 52 | 331 | 267.2 | 0.000 | 1 | 75 | 48.9 |
| pcb442 | 0.000 | 1 | 13 | 7.0 | - | - | - | - |
| att532 | 0.014 | 5 | 633 | 16110.9 | 0.000 | 1 | 431 | 1698.8 |
| u574 | 0.003 | 1 | 91 | 69.1 | 0.002 | 5 | 36 | 19.9 |
| p654 | 0.024 | 5 | 74 | 27.1 | - | - | - | - |
| rat783 | 0.008 | 6 | 126 | 209.6 | 0.000 | 1 | 131 | 73.4 |
| pr1002 | 0.010 | 14 | 69 | 143.1 | 0.000 | 1 | 21 | 61.3 |
| u1060 | 0.016 | 113 | 660 | 3807.8 | 0.005 | 157 | 533 | 1220.9 |
| pcb1173 | - | - | - | - | 0.004 | 1 | 2088 | 9002.1 |
| d1291 | 0.000 | 1 | 44 | 43.0 | - | - | - | - |
| rl1323 | 0.024 | 3 | 87 | 86.9 | 0.000 | 1 | 24 | 46.9 |
| fl1400 | - | - | - | - | - | - | - | - |
| u1432 | 0.000 | 1 | 1 | 0.4 | - | - | - | - |
| fl1577 | 0.002 | 1 | 268 | 701.5 | - | - | - | - |
| d1655 | 0.000 | 15 | 270 | 513.9 | - | - | - | - |
| vm1748 | 0.004 | 3 | 168 | 402.0 | 0.000 | 1 | 89 | 186.7 |
| rl1889 | 0.011 | 4 | 2353 | 14147.6 | 0.000 | 1 | 2366 | 11580.1 |
| u2152 | 0.000 | 1 | 7 | 4.4 | - | - | - | - |
| pr2392 | 0.000 | 3 | 130 | 693.9 | 0.000 | 8 | 92 | 1109.2 |
| pcb3038 | - | - | - | - | - | - | - | - |
| fl3795 | - | - | - | - | - | - | - | - |
| fnl4461 | - | - | - | - | - | - | - | - |
| rl5934 | - | - | - | - | - | - | - | - |
| pla7397 | - | - | - | - | - | - | - | - |

Table 5: FST-concatenation, *TSPLIB* instances. See Table 4 for captions.

| $n$ | Rectilinear | | | | Euclidean | | | |
|---|---|---|---|---|---|---|---|---|
| | Gap | Nds | LPs | CPU | Gap | Nds | LPs | CPU |
| 100 | 0.008 | 1.1 | 15.6 | 2.3 | 0.001 | 1.1 | 5.9 | 0.5 |
| | ±0.028 | ±0.4 | ±14.3 | ±1.4 | ±0.009 | ±0.4 | ±6.4 | ±0.4 |
| 200 | 0.005 | 1.4 | 42.6 | 15.9 | 0.001 | 1.2 | 23.1 | 3.6 |
| | ±0.010 | ±0.9 | ±41.6 | ±16.2 | ±0.002 | ±0.6 | ±29.8 | ±3.4 |
| 300 | 0.008 | 1.7 | 71.3 | 73.4 | 0.000 | 1.5 | 35.8 | 7.3 |
| | ±0.010 | ±1.2 | ±63.8 | ±122.9 | ±0.000 | ±0.7 | ±48.6 | ±11.5 |
| 400 | 0.012 | 3.0 | 98.9 | 161.9 | 0.000 | 1.7 | 88.5 | 30.1 |
| | ±0.013 | ±2.8 | ±91.6 | ±297.3 | ±0.000 | ±1.1 | ±125.6 | ±42.7 |
| 500 | 0.010 | 3.3 | 155.5 | 504.4 | 0.000 | 2.0 | 89.7 | 54.7 |
| | ±0.012 | ±3.7 | ±153.4 | ±803.6 | ±0.001 | ±1.3 | ±95.7 | ±90.9 |

Table 6: FST-concatenation, randomly generated instances. Averages over 50 instances for each size; standard deviations on the second line of each row. See Table 4 for captions.

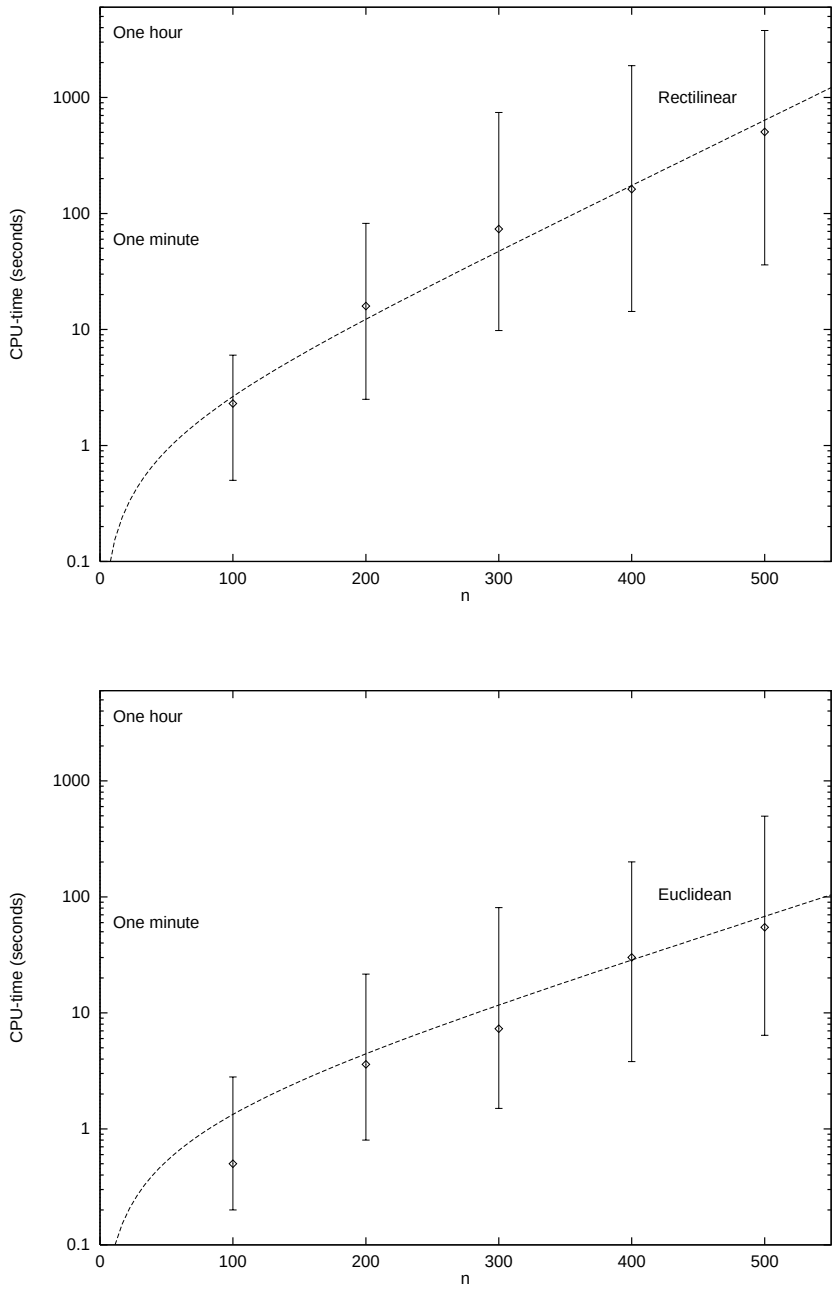



Figure 9: FST concatenation, randomly generated instances. CPU-time (seconds); average, minimum and maximum. Note the logarithmic scaling on $y$-axes.

| $n$ | | Rectilinear Count | | Size | | Red | CPU | Euclidean Count | | Size | | Red | CPU |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 100 | (1) | 48 | (18) | 3.06 | (9) | 12.11 | 7.0 | 55 | (22) | 2.80 | (5) | 3.24 | 209.4 |
| 100 | (2) | 45 | (12) | 3.20 | (8) | 12.45 | 7.0 | 56 | (26) | 2.77 | (5) | 3.48 | 265.5 |
| 100 | (3) | 53 | (17) | 2.87 | (4) | 11.63 | 4.2 | 60 | (29) | 2.65 | (5) | 3.42 | 216.4 |
| 100 | (4) | 51 | (19) | 2.94 | (5) | 10.80 | 2.5 | 55 | (25) | 2.80 | (5) | 3.25 | 193.4 |
| 100 | (5) | 55 | (23) | 2.80 | (7) | 10.21 | 2.2 | 62 | (34) | 2.60 | (5) | 3.31 | 177.7 |
| 100 | (6) | 50 | (22) | 2.98 | (7) | 12.89 | 10.8 | 60 | (32) | 2.65 | (5) | 3.40 | 234.0 |
| 100 | (7) | 48 | (12) | 3.06 | (6) | 12.69 | 10.5 | 54 | (23) | 2.83 | (5) | 3.98 | 333.7 |
| 100 | (8) | 52 | (19) | 2.90 | (7) | 11.27 | 4.7 | 61 | (34) | 2.62 | (5) | 3.57 | 177.7 |
| 100 | (9) | 52 | (19) | 2.90 | (6) | 13.56 | 7.0 | 51 | (21) | 2.94 | (6) | 3.50 | 207.4 |
| 100 | (10) | 54 | (22) | 2.83 | (6) | 11.40 | 4.4 | 61 | (32) | 2.62 | (6) | 3.37 | 188.8 |
| 100 | (11) | 52 | (20) | 2.90 | (7) | 9.97 | 3.4 | 59 | (29) | 2.68 | (6) | 2.82 | 154.1 |
| 100 | (12) | 49 | (14) | 3.02 | (5) | 12.49 | 3.3 | 58 | (23) | 2.71 | (5) | 2.69 | 115.0 |
| 100 | (13) | 46 | (13) | 3.15 | (6) | 12.27 | 6.0 | 62 | (34) | 2.60 | (5) | 2.66 | 135.6 |
| 100 | (14) | 56 | (23) | 2.77 | (5) | 10.63 | 6.2 | 55 | (27) | 2.80 | (5) | 3.57 | 180.7 |
| 100 | (15) | 51 | (20) | 2.94 | (9) | 11.43 | 5.8 | 65 | (39) | 2.52 | (5) | 2.78 | 241.2 |
| 250 | (1) | 120 | (36) | 3.08 | (7) | 10.86 | 14.3 | 149 | (70) | 2.67 | (5) | 3.08 | 1305.6 |
| 250 | (2) | 127 | (43) | 2.96 | (6) | 11.66 | 22.6 | 147 | (81) | 2.69 | (6) | 2.99 | 937.6 |
| 250 | (3) | 138 | (54) | 2.80 | (6) | 11.23 | 34.4 | 153 | (80) | 2.63 | (5) | 3.28 | 1283.7 |
| 250 | (4) | 132 | (47) | 2.89 | (7) | 11.35 | 27.1 | 145 | (68) | 2.72 | (6) | 3.23 | 1132.7 |
| 250 | (5) | 131 | (50) | 2.90 | (8) | 12.00 | 41.7 | 148 | (78) | 2.68 | (7) | 3.48 | 1054.3 |
| 250 | (6) | 129 | (46) | 2.93 | (6) | 10.58 | 48.4 | 149 | (74) | 2.67 | (5) | 2.93 | 977.4 |
| 250 | (7) | 130 | (47) | 2.92 | (7) | 10.60 | 21.1 | 144 | (70) | 2.73 | (6) | 2.80 | 1055.8 |
| 250 | (8) | 126 | (43) | 2.98 | (6) | 12.70 | 34.2 | 151 | (75) | 2.65 | (5) | 3.65 | 2051.6 |
| 250 | (9) | 125 | (41) | 2.99 | (6) | 11.84 | 247.0 | 148 | (77) | 2.68 | (5) | 3.09 | 1231.2 |
| 250 | (10) | 125 | (37) | 2.99 | (6) | 12.51 | 72.6 | 150 | (80) | 2.66 | (5) | 3.40 | 1522.6 |
| 250 | (11) | 126 | (46) | 2.98 | (7) | 12.03 | 25.9 | 143 | (69) | 2.74 | (5) | 3.31 | 1282.1 |
| 250 | (12) | 130 | (52) | 2.92 | (6) | 11.41 | 44.0 | 149 | (78) | 2.67 | (6) | 3.45 | 1088.8 |
| 250 | (13) | 123 | (39) | 3.02 | (7) | 12.09 | 335.8 | 151 | (75) | 2.65 | (4) | 3.29 | 1571.7 |
| 250 | (14) | 128 | (49) | 2.95 | (7) | 11.75 | 15.2 | 142 | (70) | 2.75 | (7) | 2.99 | 1159.8 |
| 250 | (15) | 125 | (38) | 2.99 | (6) | 12.09 | 27.3 | 143 | (73) | 2.74 | (7) | 3.15 | 1305.6 |
| 500 | (1) | 253 | (79) | 2.97 | (6) | 11.52 | 79.1 | 286 | (134) | 2.74 | (6) | 3.42 | 5479.2 |
| 500 | (2) | 244 | (74) | 3.05 | (7) | 12.80 | 84.5 | 285 | (123) | 2.75 | (5) | 3.51 | 6649.5 |
| 500 | (3) | 248 | (86) | 3.01 | (7) | 12.40 | 982.1 | 288 | (139) | 2.73 | (6) | 3.37 | 6711.0 |
| 500 | (4) | 266 | (106) | 2.88 | (8) | 11.26 | 378.4 | 291 | (141) | 2.71 | (6) | 3.50 | 5820.6 |
| 500 | (5) | 256 | (90) | 2.95 | (7) | 11.08 | 2122.2 | 300 | (157) | 2.66 | (6) | 2.87 | 4217.7 |
| 500 | (6) | 261 | (95) | 2.91 | (7) | 11.69 | 63.0 | 287 | (133) | 2.74 | (5) | 3.37 | 6400.0 |
| 500 | (7) | 248 | (77) | 3.01 | (7) | 11.74 | 76.3 | 290 | (139) | 2.72 | (6) | 3.38 | 4753.8 |
| 500 | (8) | 257 | (99) | 2.94 | (7) | 11.50 | 517.8 | 304 | (158) | 2.64 | (6) | 3.17 | 5064.7 |
| 500 | (9) | 259 | (93) | 2.93 | (6) | 11.15 | 94.1 | 294 | (146) | 2.70 | (6) | 3.38 | 5493.6 |
| 500 | (10) | 258 | (97) | 2.93 | (7) | 11.53 | 109.2 | 284 | (134) | 2.76 | (6) | 3.60 | 5339.4 |
| 500 | (11) | 257 | (91) | 2.94 | (5) | 11.67 | 1198.6 | 297 | (156) | 2.68 | (5) | 3.25 | 6488.8 |
| 500 | (12) | 260 | (99) | 2.92 | (7) | 11.21 | 197.2 | 302 | (153) | 2.65 | (6) | 3.21 | 5373.4 |
| 500 | (13) | 259 | (93) | 2.93 | (6) | 11.66 | 96.3 | 288 | (141) | 2.73 | (6) | 3.37 | 4109.3 |
| 500 | (14) | 247 | (84) | 3.02 | (7) | 12.02 | 489.0 | 285 | (136) | 2.75 | (6) | 3.27 | 7558.5 |
| 500 | (15) | 261 | (97) | 2.91 | (7) | 11.22 | 52.5 | 294 | (147) | 2.70 | (7) | 3.22 | 5674.0 |
| 1000 | (1) | 521 | (188) | 2.92 | (7) | 11.84 | 2812.0 | 580 | (278) | 2.72 | (6) | 3.45 | 21050.3 |
| 1000 | (2) | 496 | (164) | 3.01 | (7) | 11.43 | 642.0 | 587 | (296) | 2.70 | (6) | 3.40 | 23494.8 |
| 1000 | (3) | 520 | (197) | 2.92 | (6) | 11.16 | 34339.7 | 609 | (318) | 2.64 | (5) | 3.17 | 22919.3 |
| 1000 | (4) | 499 | (184) | 3.00 | (8) | 11.61 | 1199.1 | 572 | (279) | 2.75 | (8) | 3.30 | 25040.5 |
| 1000 | (5) | 522 | (205) | 2.91 | (7) | 11.34 | 35003.2 | 604 | (305) | 2.65 | (6) | 3.10 | 23291.7 |
| 1000 | (6) | 507 | (188) | 2.97 | (9) | 11.57 | 351965.7 | 582 | (279) | 2.72 | (6) | 3.23 | 27293.6 |
| 1000 | (7) | 523 | (206) | 2.91 | (7) | 11.33 | 39580.6 | 587 | (292) | 2.70 | (6) | 3.26 | 20515.5 |
| 1000 | (8) | 517 | (186) | 2.93 | (7) | 11.80 | 28715.0 | 586 | (288) | 2.70 | (5) | 3.42 | 45775.0 |
| 1000 | (9) | 501 | (167) | 2.99 | (7) | 12.10 | 587.8 | 576 | (272) | 2.73 | (6) | 3.37 | 26778.9 |
| 1000 | (10) | 507 | (169) | 2.97 | (7) | 11.81 | 48797.1 | 598 | (305) | 2.67 | (6) | 3.36 | 20962.1 |
| 1000 | (11) | 511 | (181) | 2.95 | (8) | 11.36 | 1371.6 | 582 | (294) | 2.72 | (6) | 3.14 | 23576.2 |
| 1000 | (12) | 492 | (164) | 3.03 | (7) | 12.71 | 214615.9 | 570 | (281) | 2.75 | (6) | 3.58 | 28629.5 |
| 1000 | (13) | 518 | (181) | 2.93 | (8) | 11.43 | 263943.6 | 591 | (290) | 2.69 | (6) | 3.19 | 19599.6 |
| 1000 | (14) | 517 | (179) | 2.93 | (8) | 11.74 | 540409.9 | 587 | (287) | 2.70 | (6) | 3.48 | 26581.2 |
| 1000 | (15) | 512 | (184) | 2.95 | (8) | 11.58 | 2729.8 | 576 | (280) | 2.73 | (8) | 3.24 | 23753.2 |

Table 7: SMT-properties, *OR-library* instances. Count: Number of FSTs in SMT (resp. number of MST-edges). Size: Average number of terminals spanned by FSTs in SMT (resp. maximum number of terminals spanned). Red: Reduction over MST in percent. CPU: Total CPU-time (seconds).

| Instance | Rectilinear | | | | | | Euclidean | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Count | | Size | | Red | CPU | Opt | | Count | | Size | CPU |
| d198 | 171 | (146) | 2.15 | (4) | 3.66 | 5.0 | 102 | (54) | 2.93 | (8) | 2.91 | 4164.4 |
| lin318 | 227 | (148) | 2.40 | (4) | 8.90 | 456.2 | 208 | (129) | 2.52 | (5) | 4.77 | 10799.7 |
| fl417 | 237 | (111) | 2.76 | (5) | 11.94 | 272.1 | 180 | (47) | 3.31 | (7) | 3.34 | 81852.8 |
| pcb442 | 392 | (347) | 2.12 | (5) | 3.99 | 8.2 | - | - | - | - | - | - |
| att532 | 272 | (93) | 2.95 | (8) | 11.44 | 16130.3 | 310 | (159) | 2.71 | (6) | 3.36 | 9102.6 |
| u574 | 370 | (211) | 2.55 | (7) | 8.92 | 74.0 | 344 | (175) | 2.67 | (7) | 3.15 | 5367.7 |
| p654 | 584 | (526) | 2.12 | (5) | 5.89 | 30.4 | - | - | - | - | - | - |
| rat783 | 414 | (163) | 2.89 | (9) | 12.59 | 255.3 | 448 | (221) | 2.75 | (6) | 3.52 | 21331.6 |
| pr1002 | 689 | (437) | 2.45 | (5) | 8.61 | 150.7 | 581 | (278) | 2.72 | (6) | 3.05 | 15509.2 |
| u1060 | 674 | (393) | 2.57 | (8) | 11.31 | 3819.8 | 599 | (274) | 2.77 | (12) | 3.25 | 60451.8 |
| pcb1173 | - | - | - | - | - | - | 802 | (522) | 2.46 | (10) | 3.18 | 81889.4 |
| d1291 | 1250 | (1213) | 2.03 | (4) | 1.80 | 48.8 | - | - | - | - | - | - |
| rl1323 | 1150 | (1005) | 2.15 | (6) | 5.39 | 94.5 | 1075 | (899) | 2.23 | (5) | 1.65 | 8616.3 |
| fl1400 | - | - | - | - | - | - | - | - | - | - | - | - |
| u1432 | 1431 | (1431) | 2.00 | (2) | 0.00 | 5.2 | - | - | - | - | - | - |
| fl1577 | 1189 | (879) | 2.33 | (5) | 10.59 | 714.0 | - | - | - | - | - | - |
| d1655 | 1508 | (1368) | 2.10 | (4) | 3.57 | 521.6 | - | - | - | - | - | - |
| vm1748 | 1320 | (1009) | 2.32 | (8) | 8.93 | 420.1 | 1258 | (888) | 2.39 | (6) | 2.81 | 28839.6 |
| rl1889 | 1605 | (1381) | 2.18 | (5) | 5.48 | 14159.0 | 1485 | (1190) | 2.27 | (8) | 2.02 | 44773.9 |
| u2152 | 2141 | (2131) | 2.00 | (3) | 0.22 | 14.0 | - | - | - | - | - | - |
| pr2392 | 1712 | (1163) | 2.40 | (7) | 7.75 | 710.1 | 1490 | (869) | 2.60 | (5) | 3.61 | 102631.5 |
| pcb3038 | - | - | - | - | - | - | - | - | - | - | - | - |
| fl3795 | - | - | - | - | - | - | - | - | - | - | - | - |
| fnl4461 | - | - | - | - | - | - | - | - | - | - | - | - |
| rl5934 | - | - | - | - | - | - | - | - | - | - | - | - |
| pla7397 | - | - | - | - | - | - | - | - | - | - | - | - |

Table 8: SMT-properties, *TSPLIB* instances. See Table 7 for captions.

| $n$ | Rectilinear | | | | | | Euclidean | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Count | | Size | | Opt | CPU | Count | | Size | | Red | CPU |
| 100 | 51.7 | (19.4) | 2.93 | (5.7) | 11.30 | 4.0 | 58.2 | (29.0) | 2.71 | (5.3) | 3.16 | 162.8 |
| | ±3.8 | ±4.6 | ±0.14 | ±1.0 | ±1.03 | ±1.8 | ±3.2 | ±4.2 | ±0.09 | ±0.9 | ±0.43 | ±49.7 |
| 200 | 102.0 | (35.8) | 2.95 | (6.3) | 11.51 | 20.9 | 116.2 | (57.7) | 2.72 | (5.5) | 3.25 | 784.6 |
| | ±4.3 | ±5.1 | ±0.08 | ±0.8 | ±0.81 | ±16.2 | ±4.7 | ±6.7 | ±0.07 | ±0.6 | ±0.34 | ±202.3 |
| 300 | 153.5 | (54.7) | 2.95 | (6.6) | 11.47 | 81.2 | 177.2 | (88.5) | 2.69 | (5.9) | 3.25 | 1748.0 |
| | ±5.4 | ±6.5 | ±0.07 | ±1.4 | ±0.51 | ±123.6 | ±4.8 | ±8.2 | ±0.05 | ±0.9 | ±0.23 | ±401.2 |
| 400 | 205.6 | (73.3) | 2.94 | (6.8) | 11.54 | 172.6 | 234.6 | (116.8) | 2.70 | (6.0) | 3.20 | 3201.5 |
| | ±6.7 | ±8.1 | ±0.06 | ±1.0 | ±0.45 | ±298.3 | ±5.9 | ±9.3 | ±0.04 | ±0.9 | ±0.20 | ±596.6 |
| 500 | 256.4 | (92.2) | 2.95 | (7.1) | 11.53 | 517.9 | 293.1 | (145.2) | 2.70 | (6.0) | 3.25 | 5249.5 |
| | ±6.6 | ±7.5 | ±0.05 | ±1.0 | ±0.45 | ±803.9 | ±6.9 | ±10.5 | ±0.04 | ±0.7 | ±0.20 | ±1013.3 |

Table 9: SMT-properties, randomly generated instances. Averages over 50 instances for each size; standard deviations on the second line of each row. See Table 7 for captions.

| $n$ | Generation | | | | Concatenation | | | | SMT | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Count | Size | Incomp | CPU | Gap | Nds | LPs | CPU | Count | Size | Opt | CPU |
| 12 | 107 | 5.38 | 91.27 | 0.6 | 0.000 | 1 | 7 | 0.1 | 4 | 3.75 | 11.90 | 0.7 |
| 22 | 243 | 7.77 | 91.88 | 5.2 | 0.000 | 1 | 16 | 1.3 | 8 | 3.62 | 11.47 | 6.6 |
| 32 | 460 | 11.03 | 94.32 | 28.1 | 0.000 | 1 | 47 | 7.3 | 12 | 3.58 | 11.43 | 35.4 |
| 42 | 831 | 15.34 | 96.16 | 131.9 | 0.000 | 1 | 51 | 32.8 | 16 | 3.56 | 11.42 | 164.7 |
| 52 | 1522 | 20.35 | 97.44 | 585.7 | 0.000 | 1 | 104 | 138.7 | 20 | 3.55 | 11.42 | 724.4 |

Table 10: Difficult rectilinear instances. See text for instance descriptions and Tables 1, 4 and 7 for captions.

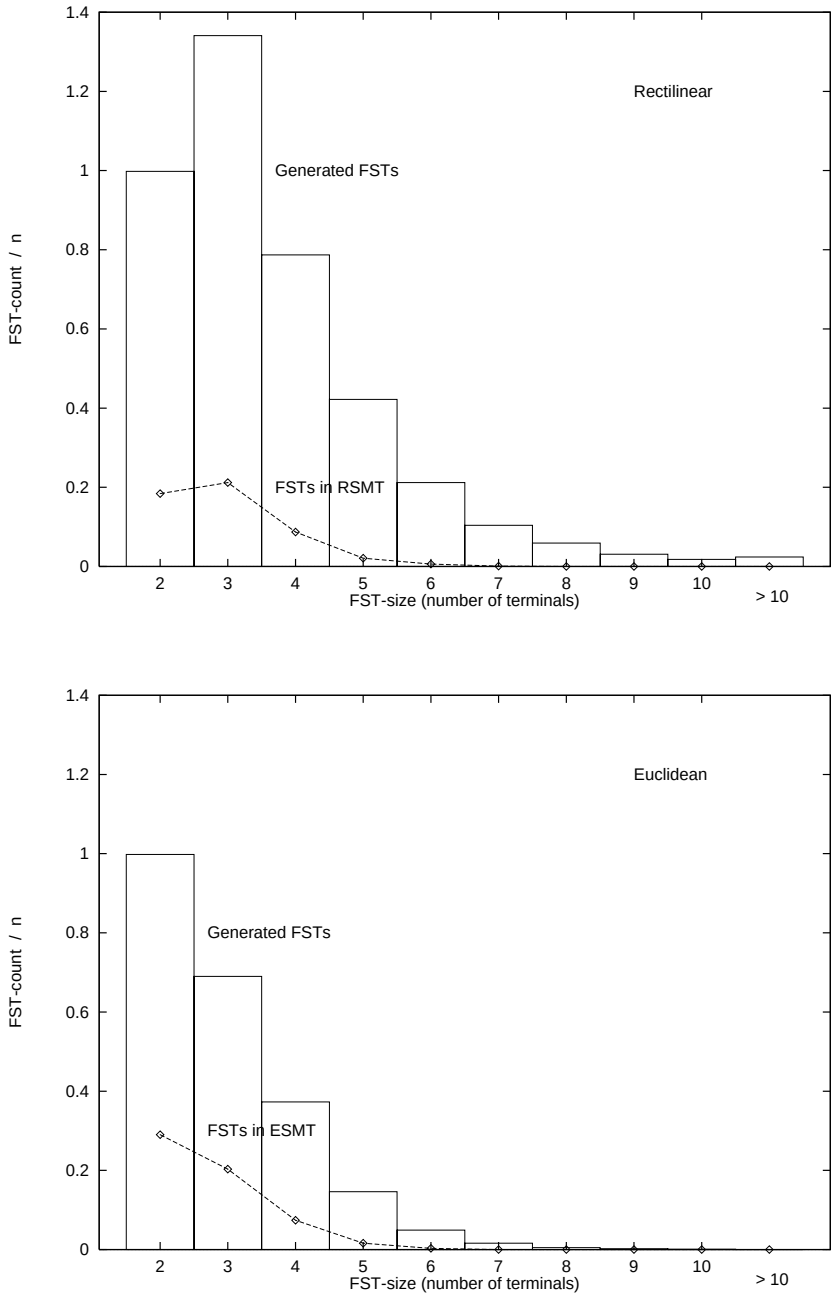



Figure 10: FST-size distribution, randomly generated instances ($n = 500$).

| | Generation | | | | Concatenation | | | | SMT | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | Count | Size | Incomp | CPU | Gap | Nds | LPs | CPU | Count | Size | Opt | CPU |
| 14 | 96 | 5.50 | 90.30 | 52.2 | 0.000 | 1 | 1 | 0.1 | 1 | 14.00 | 7.13 | 52.3 |
| 21 | 216 | 5.67 | 79.93 | 236.6 | 0.000 | 1 | 1 | 0.1 | 8 | 3.50 | 8.04 | 236.7 |
| 28 | 376 | 5.83 | 76.89 | 679.3 | 0.071 | 7 | 11 | 0.7 | 9 | 4.00 | 8.34 | 680.0 |
| 35 | 565 | 6.04 | 75.14 | 1697.8 | 0.230 | 26 | 72 | 7.6 | 12 | 3.83 | 8.20 | 1705.4 |
| 42 | 798 | 6.42 | 75.06 | 4049.6 | 0.094 | 25 | 126 | 18.5 | 15 | 3.73 | 8.50 | 4068.0 |
| 49 | 1057 | 6.76 | 75.24 | 8747.7 | 0.203 | 92 | 383 | 144.8 | 18 | 3.67 | 8.37 | 8892.4 |

Table 11: Difficult Euclidean instances. See text for instance descriptions and Tables 1, 4 and 7 for captions.

# 5 Conclusion

This paper presented an extensive computational study on exact algorithms for the Euclidean and rectilinear Steiner tree problems in the plane. Optimal solutions to problem instances with more than 2000 terminals have been obtained. The branch-and-cut FST concatenation algorithm is orders of magnitude faster than algorithms based on backtrack search or dynamic programming. However, for the rectilinear problem, the concatenation phase is still the bottleneck.

The Euclidean FST generator is remarkably effective on (uniformly) randomly generated problem instances. For structured (real-world) instances, such as those in *TSPLIB*, the generator is less efficient. One reason is simply that fewer FSTs can be pruned away by using current tests. In order to solve larger, structured Euclidean problems, the Euclidean FST generator must be improved. Often such instances contain terminal subsets which can be mapped into other subsets by translation and rotation (and sometimes scaling). If such congruent subsets could be identified, FSTs generated for one subset could essentially be copied to all congruent subsets. However, whether this is a practical option for improving the FST generator is still an open question.

The FST concatenator can, on the other hand, most likely be improved. We are currently investigating three options: Firstly, FST pruning can be applied to the list of generated FSTs. If faster and perhaps even more powerful tests than those already suggested [9, 26] can be applied, it may be worthwhile to reduce the list of FSTs before entering the concatenation phase. But as noted in Section 4.3 the effect of reducing the FST list need not have a positive effect on the concatenation.

Another option is early branching, i.e., branching in the root node before an LP optimal solution has been obtained. The problems that require the most time to concatenate spend the vast majority of their time generating constraints that improve the lower bound only minutely before LP-optimum is reached. Orders of magnitude reduction in total concatenation time are possible when good heuristics are used to detect such *tailing off* in the convergence rate and choosing a good branching variable instead.

Thirdly, other types of valid inequalities may be added to the formulation. For other well-studied problems (e.g., TSP) the inclusion of new types of valid

inequalities has formed the basis for substantially improved exact algorithms.

FST concatenation can be avoided altogether by enumerating all so-called full topologies for $Z$ (see, e.g., [15]). A shortest tree for a given full topology or any of its degenerate topologies can be determined in $O(n^2)$ time using the *luminary algorithm* of Hwang and Weng [16] or by using a numerical algorithm suggested by Smith [21]. Since the number of full topologies is superexponential in the number of terminals, these methods can only be effective if the number of full topologies can be reduced by using pruning tests similar to those applied in this paper.

# References


[1] J. E. Beasley. OR-Library: Distributing Test Problems by Electronic Mail. *Journal of the Operational Research Society*, 41:1069–1072, 1990.

[2] M. W. Bern and R. L. Graham. The Shortest-Network Problem. *Scientific American*, pages 66–71, January 1989.

[3] M. Brazil, T. Cole, J. H. Rubinstein, D. A. Thomas, J. F. Weng, and N. C. Wormald. Minimal Steiner Trees for $2^k$ x $2^k$ Square Lattices. *Journal of Combinatorial Theory, Series A*, 73:91–110, 1996.

[4] M. Brazil, J. H. Rubinstein, D. A. Thomas, J. F. Weng, and N. C. Wormald. Full Minimal Steiner Trees on Lattice Sets. *Journal of Combinatorial Theory, Series A*, 78:51–91, 1997.

[5] M. Brazil, J. H. Rubinstein, D. A. Thomas, J. F. Weng, and N. C. Wormald. Minimal Steiner Trees for Rectangular Arrays of Lattice Points. *Journal of Combinatorial Theory, Series A*, 79:181–208, 1997.

[6] E. J. Cockayne and D. E. Hewgill. Exact Computation of Steiner Minimal Trees in the Plane. *Information Processing Letters*, 22:151–156, 1986.

[7] E. J. Cockayne and D. E. Hewgill. Improved Computation of Plane Steiner Minimal Trees. *Algorithmica*, 7(2/3):219–229, 1992.

[8] S. E. Dreyfus and R. A. Wagner. The Steiner Problem in Graphs. *Networks*, 1:195–207, 1971.

[9] U. Fößmeier and M. Kaufmann. On Exact Solutions for the Rectilinear Steiner Tree Problem. Technical Report WSI-96-09, Universität Tübingen, 1996.

[10] J. L. Ganley and J. P. Cohoon. Improved Computation of Optimal Rectilinear Steiner Minimal Trees. *International Journal of Computational Geometry and Applications*, 7(5):457–472, 1997.

[11] M. R. Garey, R. L. Graham, and D. S. Johnson. The Complexity of Computing Steiner Minimal Trees. *SIAM Journal on Applied Mathematics*, 32(4):835–859, 1977.

[12] M. R. Garey and D. S. Johnson. The Rectilinear Steiner Tree Problem is $NP$-Complete. *SIAM Journal on Applied Mathematics*, 32(4):826–834, 1977.

[13] M. Hanan. On Steiner's Problem with Rectilinear Distance. *SIAM Journal on Applied Mathematics*, 14(2):255–265, 1966.

[14] F. K. Hwang. On Steiner Minimal Trees with Rectilinear Distance. *SIAM Journal on Applied Mathematics*, 30:104–114, 1976.

[15] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner Tree Problem.* Annals of Discrete Mathematics 53. Elsevier Science Publishers, Netherlands, 1992.

[16] F. K. Hwang and J. F. Weng. The Shortest Network under a Given Topology. *Journal of Algorithms*, 13:468–488, 1992.

[17] K. Mehlhorn and S. Näher. LEDA - A Platform for Combinatorial and Geometric Computing. Max Planck Institute for Computer Science http://www.mpi-sb.mpg.de/LEDA/leda.html, 1996.

[18] F.P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, second edition, 1988.

[19] G. Reinelt. TSPLIB - A Traveling Salesman Problem Library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

[20] J. S. Salowe and D. M. Warme. Thirty-Five-Point Rectilinear Steiner Minimal Trees in a Day. *Networks*, 25(2):69–87, 1995.

[21] W. D. Smith. How to Find Steiner Minimal Trees in Euclidean $d$-Space. *Algorithmica*, 7(2/3):137–177, 1992.

[22] I. Tomescu and M. Zimand. Minimum Spanning Hypertrees. *Discrete Applied Mathematics*, 54:67–76, 1994.

[23] D. M. Warme. Practical Exact Algorithms for Geometric Steiner Problems. Technical report, System Simulation Solutions, Inc., Alexandria, VA 22314, USA, 1996.

[24] D. M. Warme. A New Exact Algorithm for Rectilinear Steiner Minimal Trees. Technical report, System Simulation Solutions, Inc., Alexandria, VA 22314, USA, 1997.

[25] P. Winter. An Algorithm for the Steiner Problem in the Euclidean Plane. *Networks*, 15:323–345, 1985.

[26] P. Winter and M. Zachariasen. Euclidean Steiner Minimum Trees: An Improved Exact Algorithm. *Networks*, 30:149–166, 1997.

[27] M. Zachariasen. Rectilinear Full Steiner Tree Generation. Technical Report 97/29, DIKU, Department of Computer Science, University of Copenhagen, 1997.