Technical Report DIKU-TR-97/29
Department of Computer Science
University of Copenhagen
Universitetsparken 1
DK-2100 KBH Ø
DENMARK

December 1997

Rectilinear Full Steiner Tree Generation

Martin Zachariasen

Rectilinear Full Steiner Tree Generation

Martin Zachariasen*

December 19, 1997

Abstract

The fastest exact algorithm (in practice) for the rectilinear Steiner tree problem in the plane uses a two-phase scheme: First a small but sufficient set of full Steiner trees (FSTs) is generated and then a Steiner minimum tree is constructed from this set by using simple backtrack search, dynamic programming or an integer programming formulation. FST generation methods can be seen as problem reduction algorithms and are also useful as a first step in providing good upper- and lower-bounds for large instances. Currently, the time needed to generate FSTs poses a significant overhead for FST based exact algorithms. In this paper we present a very efficient algorithm for the rectilinear FST generation problem which removes this overhead completely. Based on information obtained in a preprocessing phase, the new algorithm "grows" FSTs while applying several new and important optimality conditions. For randomly generated instances approximately $4n$ FSTs are generated (where n is the number of terminals). The observed running time is quadratic and the FSTs for a 10000 terminal instance can on average be generated within 10 minutes.

1 Introduction

The rectilinear Steiner tree problem (RSTP) asks for a shortest interconnection of a set Z of n terminals (points in the plane) using only horizontal and vertical lines. Alternatively we may say that we would like to interconnect Z using the rectilinear (or Manhattan) distance metric L_1. This NP-hard problem [5] has important applications in, e.g., VLSI-design. Many exact algorithms and

*Department of Computer Science, University of Copenhagen, DK-2100 Copenhagen Ø, Denmark. E-mail: martinz@diku.dk.

heuristics have been proposed for the problem; for an extensive survey see the book by Hwang, Richards and Winter [11].

The Steiner points for a rectilinear Steiner minimum tree (SMT) may be confined to the vertices of the grid graph for Z [9]. An SMT is a union of *full Steiner trees (FSTs)*, in which every leaf is a terminal (having degree one) and all other nodes (having degree three or more) are Steiner points. The FSTs of an SMT are also denoted *full components*. Furthermore, our definition of FSTs should be compared to the definition of FSTs in [18, 23] where FSTs are required to span terminal sets for which every SMT is a full topology. The SMT shown in Figure 1 consists of 7 FSTs; each FST spans from 2 to 6 terminals.

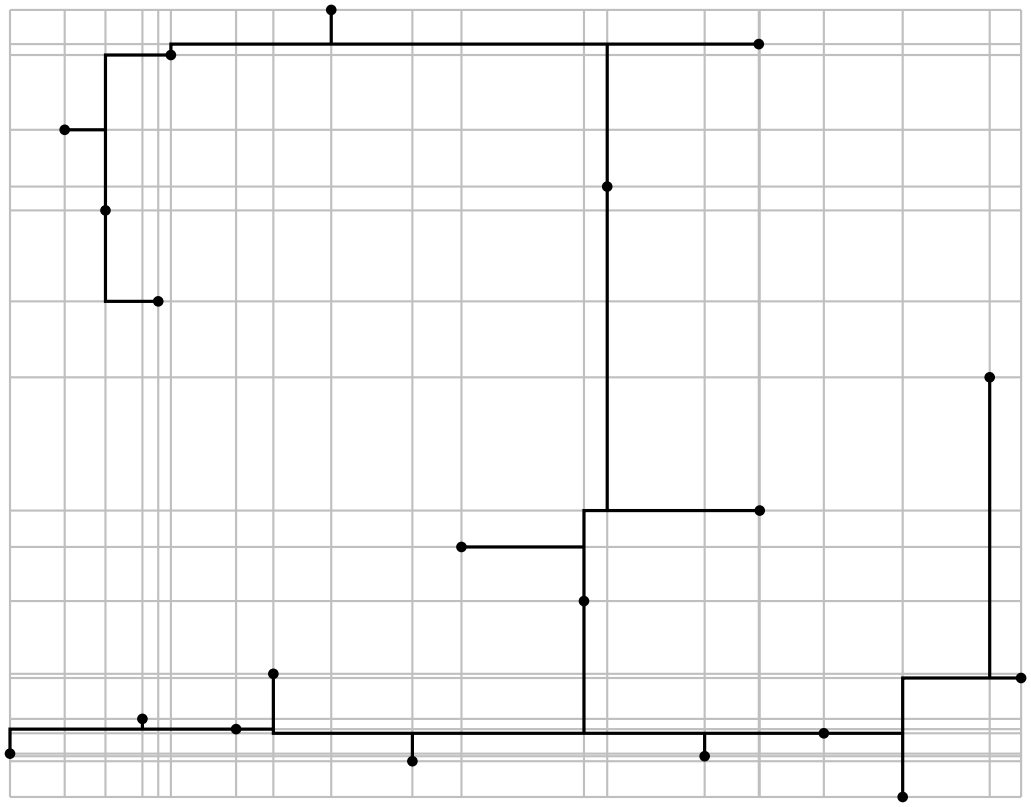


Figure 1: Grid graph and Steiner minimum tree.

Recently Warme [23] made a breakthrough in the construction of exact algorithms for the problem. The new algorithm uses a two-phase scheme originally suggested by Winter [24] for the Euclidean problem and later applied to the rectilinear problem by Salowe and Warme [18]. The idea is following: In the first (FST generation) phase we generate a (small) set of FSTs $\mathcal{F}$ containing at least

one SMT identified as a subset. In the second (FST concatenation) phase we find a subset $\mathcal{F}^* \subseteq \mathcal{F}$ with minimum total length such that the FSTs in $\mathcal{F}^*$ interconnect Z. Warme [23] noticed that FST concatenation is equivalent to finding a minimum spanning tree in the hypergraph $H = (Z, \mathcal{F})$ and formulated this problem as an integer program. He solved this problem using branch-and-cut, allowing the exact solution of rectilinear Steiner tree problem instances with up to 1000 terminals.

However, the first phase (FST generation) now seems to be the bottleneck for a large fraction of the problem instances considered by Warme [23]. Salowe and Warme [18] gave the first FST generation algorithm for the rectilinear problem. Although being very fast for small instances ($n \leq 100$), the exponential running time growth of the algorithm made it impractical for larger instances (a 400 terminal instance required several days of CPU-time[1]). Warme [23] improved this algorithm significantly, reducing the observed running time to $O(n^3)$. FSTs for a 1000 terminal instance could be generated in a few hours. In this paper we reduce the observed running time to $O(n^2)$; the algorithm generates FSTs for a 1000 terminal instance in less than 10 seconds and for a 10000 terminal instance in less than 10 minutes.

A precise description of the topology of rectilinear FSTs was given by Hwang [10]. This characterization and other properties of rectilinear FSTs are given in Section 3. In Section 4 we give an overview of the FST generation algorithm which is based on "growing" FSTs. Section 5 describes the important preprocessing phase while Section 6 presents the tests performed while growing FSTs. Theoretical bounds on the expected number of FSTs generated are given in Section 7. Computational results are reported in Section 8 and concluding remarks are given in Section 9.

2 Definitions and Basic Notions

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 $\|uv\|_p = (|u_x - v_x|^p + |u_y - v_y|^p)^{1/p}$. As special cases we have the rectilinear (or Manhattan) L_1-distance $|uv| = \|uv\|_1$, the Euclidean L_2-distance $\|uv\| = \|uv\|_2$ and $\|uv\|_\infty = \max(|u_x - v_x|, |u_y - v_y|)$.

Define $\mathcal{C}_p(u, r)$ to be the interior of an L_p-circle centred at u with radius $r > 0$: $\mathcal{C}_p(u, r) = \{x \in \Re^2 \mid \|ux\|_p < r\}$. An L_p-lune $\mathcal{L}_p(u, v)$ is the set of points in the plane which are closer to both u and v than u and v are to each other, i.e., $\mathcal{L}_p(u, v) = \mathcal{C}_p(u, \|uv\|_p) \cap \mathcal{C}_p(v, \|uv\|_p)$. Finally we define $\mathcal{R}(u, v)$ to be the

[1] All running times measured on a workstation comparable to the one used in this study.

interior of the smallest axis-aligned rectangle that contains u and v; thus u and v are opposite corners of the rectangle.

In general we use the terminology in [18] for geometric primitives related to rectilinear Steiner trees. A rectilinear Steiner tree consists of vertical and horizontal *segments*. Segments intersect only at their endpoints which are either terminals (belonging to Z), *corner points* (having degree two), *T-nodes* (having degree three) or *cross-nodes* (having degree four). A *Steiner point* is either a T-node or a cross-node.

A *line* is a sequence of one or more adjacent, colinear segments with no terminals in its relative interior (where relative interior is used in the usual geometric sense). A *complete line* is a line of maximal length. One horizontal and one vertical complete line incident to a common corner point form a *corner*; the complete lines are the *legs* of the corner. A set of segments incident to a common line l are said to *alternate* along l if each intersection point forms a distinct T-node and no two successive segments are on the same side of l.

A segment uv is oriented according to the direction of vector $\vec{uv}$. We may restrict our attention to four directions $\alpha = 0, 1, 2, 3$, corresponding to East (positive x-axis), North (positive y-axis), West (negative x-axis) and South (negative y-axis), respectively. Define $\alpha - 1$ to be the direction corresponding to α turned 90° clockwise and $\alpha + 1$ to be the direction α turned 90° counter-clockwise; for a given direction α we thus say that $\alpha + 1$ points to the left and $\alpha - 1$ points to the right. Similarly the direction $\alpha + 2$ $(= \alpha - 2)$ points in the opposite direction of α.

Let $MST(Z)$ be a minimum spanning tree for Z using distance metric L_1 and $z_i, z_j \in Z$ a pair of terminals. The *bottleneck Steiner distance* $b_{z_i z_j}$ between z_i and z_j is equal to the length of the longest edge on the (unique) path between z_i and z_j in $MST(Z)$. Note that no edge on the path between z_i and z_j in an SMT for Z can be longer than $b_{z_i z_j}$.

3 Full Steiner Tree Properties

Hwang [10] proved that there always exists an SMT for which every FST has one of the two generic forms shown in Figure 2: An FST spanning k terminals consists of a corner (also denoted the backbone) given by a *root* z_0 and a *tip* z_{k-1}. The root is incident to the *long* leg and the tip incident to the *short* leg of the corner[2]. There are two main types (i) and (ii) and two degenerate cases of type (i):

[2]The terminology *short* leg and *long* leg is not meant to connote geometric length, according to the L_1 metric. Rather, the long leg can have more incident segments than the short leg.

- Type (i) has $k-2$ alternating segments incident to the long leg and no segment incident to the short leg. The first degenerate case (i′) has a zero-length short leg, i.e., the corner is degenerated into a line. The second degenerate case (i″) is a cross spanning exactly four terminals (the two alternating incident segments are on the same line); the two Steiner points are degenerated into one Steiner point having degree four.
- Type (ii) has $k-3$ alternating segments incident to the long leg and one segment incident to the short leg.

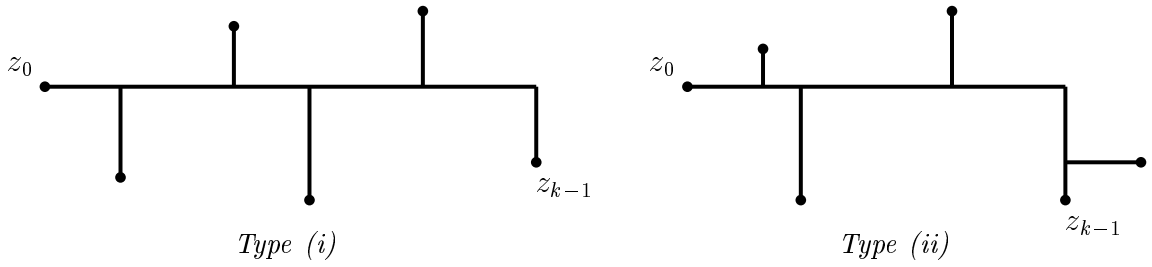


Figure 2: Generic full Steiner trees.

If $k \geq 5$ the two legs are uniquely identified; otherwise they may be interchanged except when $k = 4$ and both segments are attached to one single leg (type (i)). However, the important observation is that every FST has a terminal which may be identified as root.

In the following every FST is assumed to have Hwang topology. Note that this gives an upper bound of $O(2^n)$ on the total number of FSTs, i.e., for every subset of terminals at most four Hwang topologies exist, one for each of the four root candidates having minimum or maximum x- or y-coordinate in the subset.

We say that the long leg has direction α if the vector from the root to the corner point of the FST points in direction α. An FST is oriented according to the direction of the long leg and the position of the tip (left/right of the long leg when looking in direction α). In the following we use the FSTs shown in Figure 2 as our generic FSTs, i.e., we assume w.l.o.g. that the long leg has direction $\alpha = 0$ and the tip is to the right of the long leg when looking in direction α.

Since we are only interested in FSTs that may be part of an SMT for Z, we need a series of simple but strong *necessary* conditions for an FST to be part of an SMT. Assume that F is an FST spanning the terminal set $Z_F \subseteq Z$ of length $|F|$. If F is a subtree of an SMT for Z then F must be an SMT for Z_F. Otherwise it would be possible to shorten the SMT spanning Z. Furthermore we may assume that there exists no union of smaller FSTs spanning Z_F and having

total length $|F|$. That is, we can disregard an FST spanning Z_F for which a concatenation of smaller FSTs spanning Z_F with the same total length exists.

Bottleneck Steiner distances (defined in Section 2) provide very effective conditions that must be fulfilled. The conditions given here are essentially the same as those known for the Steiner problem in graphs [11]. Let $z_i, z_j \in Z_F$. The longest edge on the (unique) path between z_i and z_j in F cannot be longer than $b_{z_i z_j}$. One implication of this condition is that no edge in F — and hence in the SMT for Z — can be longer than the longest edge in $MST(Z)$.

Another powerful condition based on bottleneck Steiner distances is the following: $|F|$ cannot be greater than the length of a minimum spanning tree over Z_F using distances $b_{z_i z_j}$ for every pair of terminals.

3.1 Empty Regions

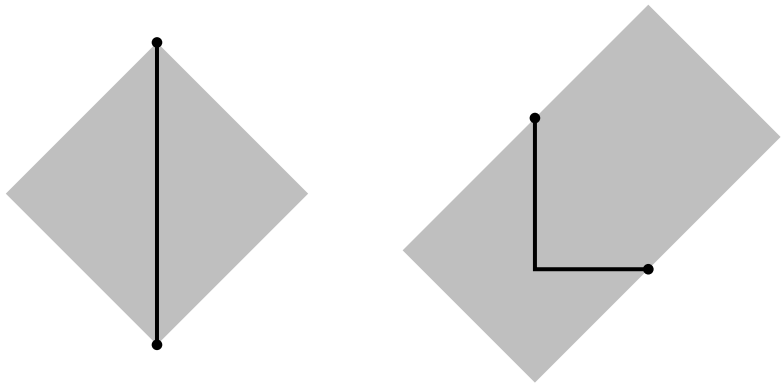
In this section we will generalize the concept of *lunes* from the Euclidean Steiner tree problem to the rectilinear case. A Euclidean lune $\mathcal{L}_2(u, v)$ for u and v is the set of points in the plane which are closer to both u and v than u and v are to each other (see also Section 2) . The so-called *lune property* states that if uv is a segment in an SMT then no other point (terminal, Steiner point or interior segment point) of the SMT, except for the segment uv itself, can be in $\mathcal{L}_2(u, v)$: Assume on the contrary that such a point x exists. Delete the segment uv and form a shorter tree by adding the segment ux or vx depending which component of the SMT x falls into when uv is deleted.

The same result applies immediately to the rectilinear problem. Let uv be a segment or a corner in a rectilinear SMT, i.e., an edge between nodes u and v in an SMT. Then no other point of the SMT can lie in $\mathcal{L}_1(u, v)$. When uv is a (horizontal or vertical) segment we call this the *empty diamond property* (Figure 3a) and when uv is a corner the *empty skew diamond property* (Figure 3b). In Figure 4 the areas covered by empty diamonds in FSTs are shown.

Let uw and vw denote two perpendicular segments sharing a common endpoint w. The nodes u, v and w may be any combination of terminals, Steiner points and corner points. Assume w.l.o.g. that the segments are oriented as in Figure 5a.

Lemma 1 *Assume that uw and vw are segments in an SMT. Then no other point (and in particular no terminal) of the SMT can lie in the interior of the smallest axis-aligned rectangle $\mathcal{R}(u, v)$ containing u and v.*

Proof. Assume on the contrary that $x \in \mathcal{R}(u, v)$, where x is some point of the SMT. The unique path P from x to w in the SMT visits either u or v first — or none of them — before reaching w.



a) Empty diamond *b) Empty skew diamond*

Figure 3: The empty (skew) diamond property.

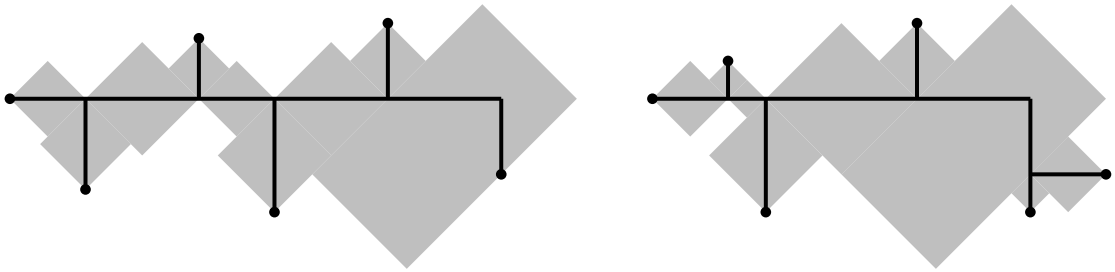
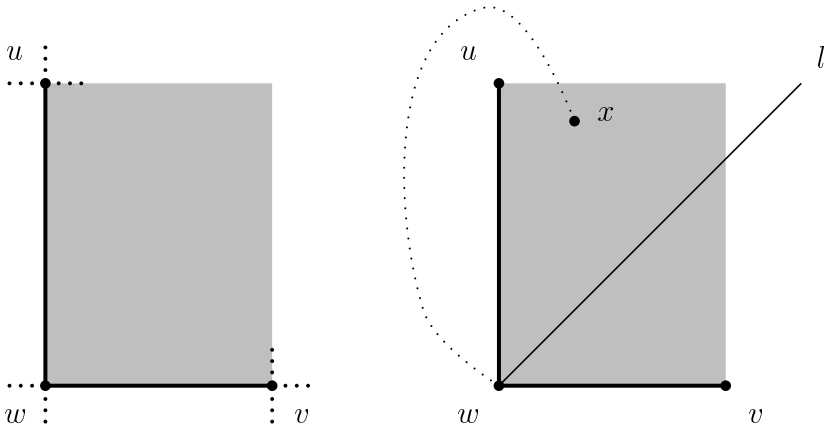


Figure 4: Empty diamonds in FSTs.

Assume that P reaches u first. Delete the segment uw and reconnect the tree by adding a vertical segment from x to a point y on the interior of vw. The resulting tree is shorter since $|xy| < |uw|$. A similar result is obtained if P reaches v first.

Now assume that P reaches neither u nor v before reaching w. Let l be the line bisecting the perpendicular angle (Figure 5b). The point x is either above, below or on the line l. If x is above l then $|ux| < |uw|$ so by deleting uw and adding ux a shorter tree is obtained. Similarly if p is below l we obtain a shorter tree by deleting vw and adding vx. Finally if p is on l we note the following: The path P consists of vertical and horizontal segments only and therefore there exists a point $x' \in P$ for which $x' \in \mathcal{R}(u, v)$ and which is either above or below l, i.e., not precisely on l. By repeating the arguments above for x' we again obtain a shorter tree. ■



a) Empty corner rectangle *b) Proof illustration*

Figure 5: Empty corner rectangle and proof illustration.

The condition given in Lemma 1 is a very strong necessary condition for any pair of adjacent and perpendicular segments in an SMT and is denoted the *empty corner rectangle property* (Figure 6).

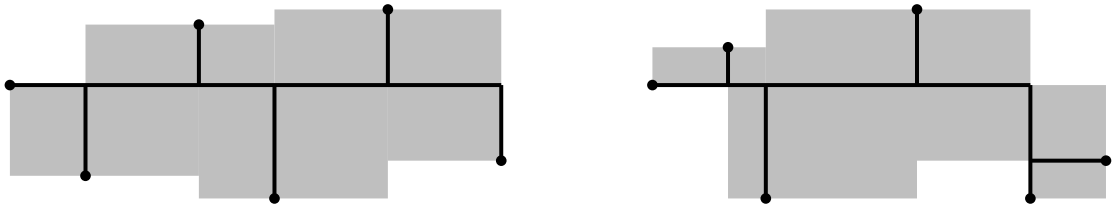
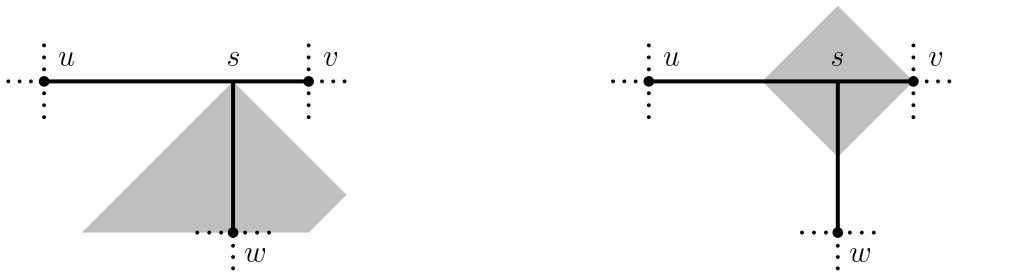


Figure 6: Empty corner rectangles in FSTs.

Let uv be a segment and assume that we would like to attach exactly one terminal to one side of uv and no terminal on the other side. In particular, uv may be the short leg of a type (ii) FST. The empty corner rectangle property then implies that at most one candidate need to be considered; we say that there is a *unique* candidate. More precisely, if z_j is attached to uv via Steiner point s_j then $|z_j s_j|$ must be minimal among all candidates. Even if two or more terminals have the same minimal distance, only one (arbitrarily chosen) candidate need to be considered: Assume that the segment $z_j s_j$ was chosen but the SMT contains uv and another segment $z_l s_l$ such that $|z_l s_l| = |z_j s_j|$. Then we may simply delete $z_l s_l$ from the SMT and add the segment $z_j s_j$ without disconnecting the tree. Otherwise there would be a path from z_l to, say u, in the SMT. By deleting the segment uv (in addition to $z_l s_l$) and adding the segments $z_j z_l$ and $z_l v$ (or $z_j v$), we can prove that the tree cannot be optimal.

Other types of empty regions can be obtained by looking at configurations like those in Figure 7. Assume that the node s (typically but not necessarily a Steiner point) is adjacent to the nodes u, v and w such that us and vs are colinear. We obtain a triangular-like empty region given by all points that are no further from w than $|ws|$ and no further to some point on the segment uv than $|ws|$ (Figure 7a). This region was used by Salowe and Warme [18] (their Theorem 6).

Another empty region is $\mathcal{C}_1(s, r_{min})$, where $r_{min} = \min(|us|, |vs|, |ws|)$ (Figure 7b). Both regions shown in Figure 7 are usually, but not always, covered by diamonds or corner rectangles and have not been used in the current implementation.



a) Triangular empty region *b) Circle empty region*

Figure 7: Other empty regions.

3.2 Corner-flipped Topologies

A rectilinear FST can always, except in degenerate cases, be transformed into equal-length trees by flipping corners and sliding segments [17, 18]. Figure 8 shows a sequence of such flips and slides. In particular, an FST can be transformed into the so-called *corner-flipped* version of itself in which the backbone

essentially is a flipped version of the original backbone corner. The Hwang topology type of the new FST depends on the type of the original FST and on the parity (odd/even) of the number of segments incident to the long leg (Figure 9).

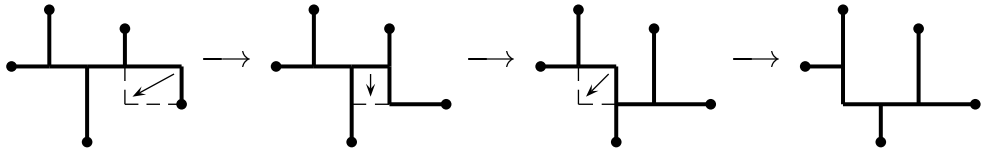


Figure 8: Flip/slide sequence for obtaining the corner-flipped topology.

The topologies corresponding to type (i)-even (i.e., type (i) FST with an even number of terminals incident to the long leg) and type (ii)-odd (i.e., type (ii) FST with an odd number of terminals incident to the long leg) are equivalent since any non-degenerate type (i)-even FST (not type (i$'$) nor type (i$''$)) can be transformed into an equal length type (ii)-odd FST and vice versa. More importantly, any FST having direction α of the long leg can be transformed into an equal-length FST having the opposite direction $\alpha + 2$ of the long leg. Thus we only need to consider two perpendicular directions, say $\alpha = 0$ and $\alpha = 1$, for the direction of the long leg when generating FSTs.

For $k \leq 3$ and type (ii) with $k = 4$ only one direction (e.g., $\alpha = 0$) actually suffices. This can be seen by simple case study, i.e., by transforming any such FST with long leg direction $\alpha = 1$ into an equal length FST with long leg direction $\alpha = 0$. However, some care is needed in degenerate cases in which a zero-length long leg has to be accepted (consider, e.g., a type (i$'$) FST spanning three terminals with long leg direction $\alpha = 1$).

3.3 Short Leg Upper Bounds

The length of the short leg in an FST F can be upper bounded on basis of the length of other segments in F. We will give four simple upper bounds on d_s which is the length of the short leg for a type (i) FST and the distance from the corner point to the Steiner point on the short leg for a type (ii) FST. All bounds are obtained by corner flipping and segment sliding.

The sequence of segments alternating along the long leg of an FST (in the direction from the root to the corner point) are denoted by $z_1 s_1$, $z_2 s_2$, etc.

Upper bound (A) Let $z_i s_i$ be a segment attached to the long leg such that z_i is on the same side of the long leg as the tip z_{k-1}. If $d_s \geq |z_i s_i|$ it is possible to split F into two smaller FSTs sharing the terminal z_i. This can be achieved by flipping and sliding until the terminal z_i is hit. Thus we have $d_s < |z_i s_i|$.

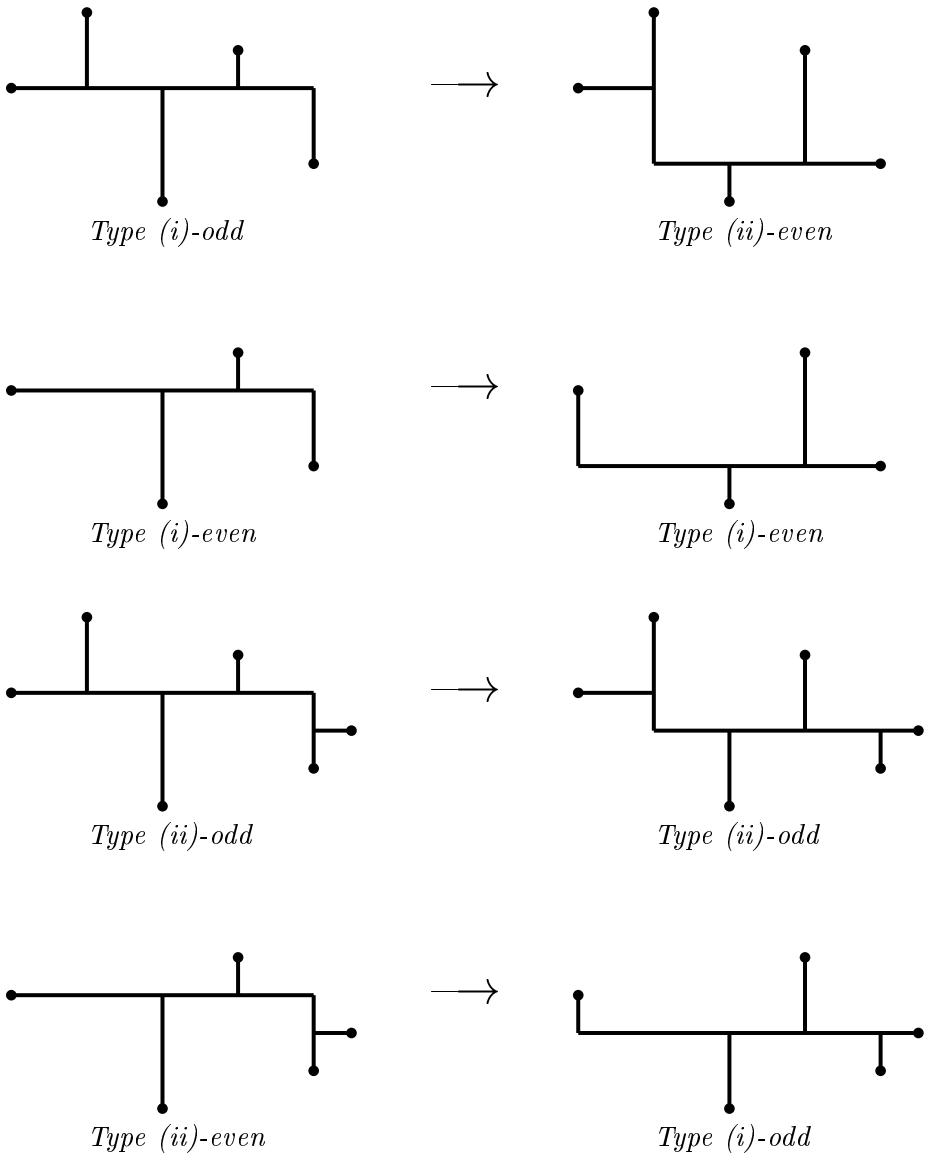


Figure 9: Corner-flipped topologies.

We can actually state an even stronger condition based on the non-intersecting property of the SMT: For type (i) the interior of the smallest axis-aligned rectangle $\mathcal{R}(z_0, z_{k-1})$ containing the root z_0 and the tip z_{k-1} contains no terminal. For type (ii) we similarly have that $\mathcal{R}(z_0, z_j)$ must be empty where z_j is the terminal attached to the short leg. This is denoted the *empty inner rectangle property* (Figure 10) and is obtained by applying upper bound (A) and the empty corner rectangle property to both F and its corner-flipped topology.

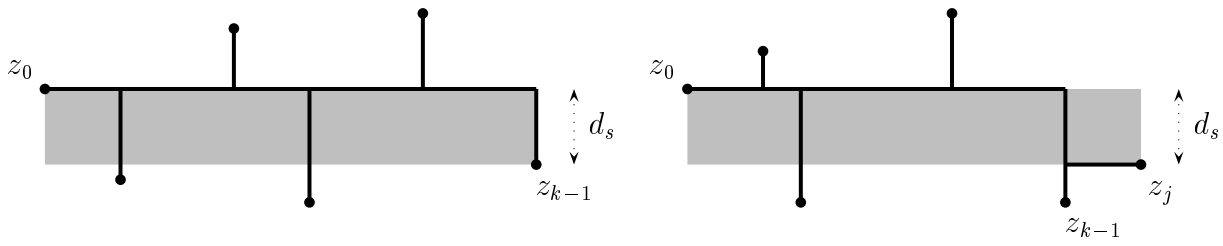


Figure 10: Empty inner rectangle.

Upper bound (B) When an FST F is transformed into its corner-flipped version the length of segments attached to the long leg on the opposite side of the tip increases by exactly d_s, except when the segment is the closest to the original root (Figure 9). Assume that we have an upper bound d_{UB} on the length of such a segment $z_i s_i$, $i > 1$ (it will be described later how such an upper bound can be obtained). Then we have $d_s \leq d_{UB} - |z_i s_i|$.

Upper bound (C) Let $s_i s_{i+1}$ be a segment on the long leg and z_i and z_{i+1} the corresponding terminals such that z_{i+1} is on the opposite side of the long leg as the tip (Figure 11). Then, $d_s \leq |s_i s_{i+1}|$.

Transform F as shown in Figure 11. Since F is an SMT we must have $|s_i s_{i+1}| \geq |s_{i+1} s'_{i+1}| = d_s$ since otherwise it would be possible to shorten the tree by deleting the segment $s_{i+1} s'_{i+1}$ and adding the segment $s_i s_{i+1}$.

Upper bound (D) Let s_i, s_{i+1} and s_{i+2} be successive Steiner points on the long leg and z_i, z_{i+1} and z_{i+2} the corresponding terminals such that z_i and z_{i+2} are on the opposite side of the long leg as the tip (Figure 12). Then, $d_s \leq |s_i s_{i+2}| - \min(|z_i s_i|, |z_{i+2} s_{i+2}|)$.

Assume w.l.o.g. that $|z_i s_i| \leq |z_{i+2} s_{i+2}|$. Transform F as shown in Figure 12. Since F is an SMT we must have $|s_i s_{i+2}| \geq |z'_{i+2} s'_{i+2}| = \min(|z_i s_i|, |z_{i+2} s_{i+2}|) + d_s$ since otherwise it would be possible to shorten the tree by deleting the segment $z'_{i+2} s'_{i+2}$ and adding the segment $z_i z'_{i+2}$.

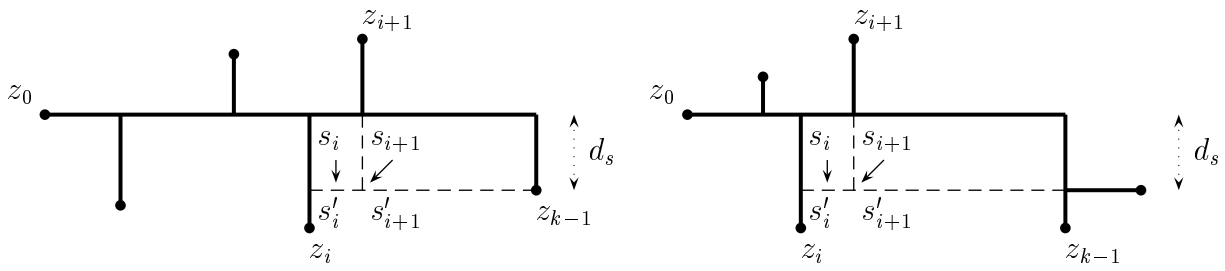


Figure 11: Violation of short leg upper bound (C).

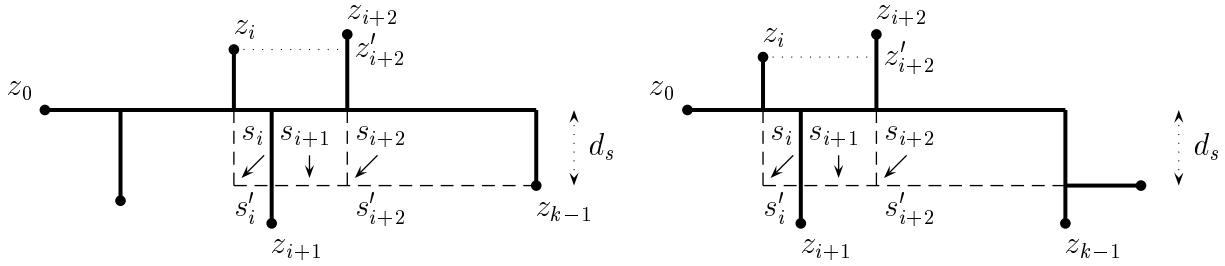


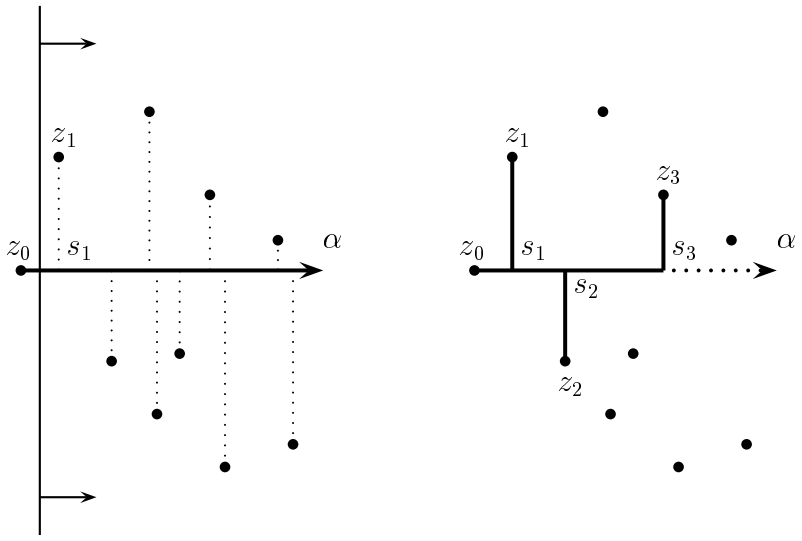
Figure 12: Violation of short leg upper bound (D).

Naturally, the upper bounds (C) and (D) can be generalized to include more than three successive Steiner points on the long leg, but we found that the effect of these bounds diminished when larger subsets of Steiner points were considered (note that the terminal which is closest to the backbone determines the effect of the bounds). By studying the conditions for optimality for small SMTs given by Hanan [9] we can prove that the upper bounds (C) and (D) are actually *necessary* conditions for the optimality of FST spanning five or fewer terminals. Since more than 90% of the FSTs generated span five or fewer terminals (see Section 8), these simple tests are very effective for guaranteeing the optimality of a large fraction of the FSTs generated.

4 Full Steiner Tree Generation Algorithm

The FST generation algorithm works by "growing" FSTs. For a given terminal z_0 and direction α we try to grow an FST with z_0 as root and having direction α of the long leg l_0 (seen as a half-line originating in z_0). The algorithm can be visualized by sweeping a line perpendicular to l_0 forth and back along l_0. The recursive algorithm works as follows: Let $\{z_1, \ldots, z_{i-1}\}$ be the current list of terminals attached to l_0 via Steiner points $\{s_1, \ldots, s_{i-1}\}$ such that the segments

$\{z_1s_1, \ldots, z_{i-1}s_{i-1}\}$ alternate along l_0 (initially this list is empty). Denote by T_{i-1} the corresponding *partial* tree; note that if $\{z_1s_1, \ldots, z_{i-1}s_{i-1}\}$ is non-empty then T_{i-1} has the form of a valid FST and that the segment $z_{i-1}s_{i-1}$ in this case is the short leg of the backbone (Figure 13b).



a) Line-sweep *b) Partial FST*

Figure 13: Growing an FST.

Now we seek a terminal z_i on the opposite side of the long leg as z_{i-1}. If the new tree $T_i = T_{i-1} \cup \{s_{i-1}s_i\} \cup \{z_is_i\}$ survives a series of FST tests (Section 6) it is stored permanently as a type (i) FST candidate. Also the type (ii) FST obtained by attaching a single segment z_js_j to the short leg z_is_i is evaluated. The direction from z_j to s_j is required to be $\alpha + 2$.

Finally the tree T_i is grown (recursively) if it survives a series of partial-tree tests (also described in Section 6). For example, if there is a shorter interconnection of the vertices $z_0, z_1, z_2, \ldots, z_i, s_i$ then there is no need to try to grow T_i any further; we say that the partial tree T_i is non-optimal.

In order to speed up this process, i.e., avoid attaching terminals to the leg l_0 which cannot constitute a (partial) FST in an SMT, we perform FST independent preprocessing (Section 5). This preprocessing primarily uses empty regions to find upper bounds on the length of segments attached to a backbone. In addition it identifies terminals which may be attached to short backbone legs.

5 FST Independent Preprocessing

In this section we describe a $O(n^2)$ time and space preprocessing phase which is used to reduce the average complexity of the FST growing phase. The main purpose of the preprocessing phase is to reduce the set of terminals that can be attached to a backbone (long or short leg). This will be accomplished by using bottleneck Steiner distances, empty diamonds (Figure 3a) and empty corner rectangles (Figure 5a).

The first step of the preprocessing phase is to sort the terminals according to each direction α. For a given terminal z_i we thus assume that its successor ${z_i}^{\alpha}$ in direction α is available in constant time.

Then we compute the bottleneck Steiner distance $b_{z_i z_j}$ for every pair of terminals $z_i, z_j \in Z$. Since this distance is equal to the length of the longest edge on the path between z_i and z_j in $MST(Z)$, this can be done in time $O(n^2)$; the space needed is obviously $O(n^2)$ also. That is, the MST can be computed in $O(n \log n)$ time by using, e.g., nearest neighbour graphs [7]; bottleneck Steiner distances from a given terminal to all other terminals can be found in $O(n)$ time using depth-first traversal of the MST. An alternative is to set up $MST(Z)$ as a dynamic search tree in time $O(n \log n)$ using only $O(n)$ space such that longest edge queries can be answered in amortized time $O(\log n)$ [19]. This and other alternatives will be discussed in Section 8.

Finally we determine, for every pair of terminals z_i and z_j, whether $\mathcal{R}(z_i, z_j)$ is empty (contains no other terminals). We use a simple $O(n^2)$ time and space algorithm which makes one line sweep for every terminal. A boolean matrix is used to store empty rectangle information. Alternatively the algorithm by Güting et al. [8] could have been used; this algorithm uses time $O(n \log n + k)$ where k is the number of empty rectangles, but still $O(n^2)$ in the worst-case.

5.1 Long Leg Terminal Candidates

Let (z_i, α) be any (terminal, direction) pair. Consider a segment $z_i s_i$ having direction α which attaches z_i to a backbone via Steiner point s_i. First we will look for an upper bound $d_{UB0}(z_i, \alpha)$ on the length of $z_i s_i$ such that if $z_i s_i$ is a part of an SMT then $|z_i s_i| \leq d_{UB0}(z_i, \alpha)$.

We use the condition that the empty diamond property must be fulfilled for $z_i s_i$, that is, no terminal can be in $\mathcal{L}_1(z_i, s_i)$. Draw two perpendicular 45° lines through z_i and let Q be the quadrant in direction α given by these two lines (Figure 14). The distance from z_i to the closest terminal z_j in Q is then a valid upper bound. If there exists no terminal in Q we set $d_{UB0}(z_i, \alpha) = \infty$.

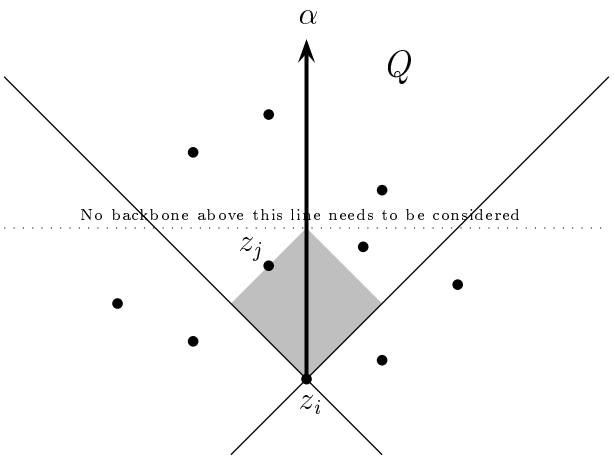


Figure 14: Long leg terminal segment upper bound.

Finding the closest such terminal for every (z_i, α) pair can be accomplished in $O(n \log n)$ time [7]. A simple alternative which is $O(n^2)$ but in practice very fast since the terminals are assumed to be sorted in each direction is to start from z_i and make a sweep in direction α until it can be concluded that the closest terminal in Q has been found. We choose to use this simple alternative since there are other parts of the preprocessing phase which require $O(n^2)$ time anyway.

5.2 Short Leg Terminal Candidates

In this phase we identify candidates which may be attached to the short leg of an FST backbone. Recall that a short leg has either zero or one attached terminal; this makes it possible to prune the candidate list quite effectively. Let (z_i, α) be any (terminal, direction) pair. Assume that z_i is a tip and incident to a short leg which points in direction α. Since we only need to grow FSTs in two perpendicular directions (say, $\alpha = 0$ and $\alpha = 1$) a terminal can only be attached to one side of a short leg pointing in direction α. We would like to determine an ordered list $Z_s(z_i, \alpha)$ of terminals that may be attached to the left or right (depending on α) of the short backbone leg.

The key observations are the following: Let $z_j s_j$ be a segment attached to the short leg of a backbone with tip z_i. Assume that the direction from z_j to s_j is β (Figure 15). Then we must have

- $|z_i s_j| \leq \min(d_{UB0}(z_i, \alpha), b_{z_i z_j})$
- $|z_j s_j| \leq \min(d_{UB0}(z_j, \beta), b_{z_i z_j})$
- $\mathcal{R}(z_i, z_j)$ is empty (contains no terminals)

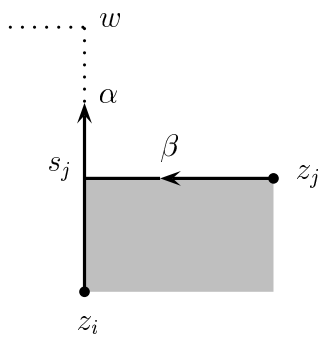


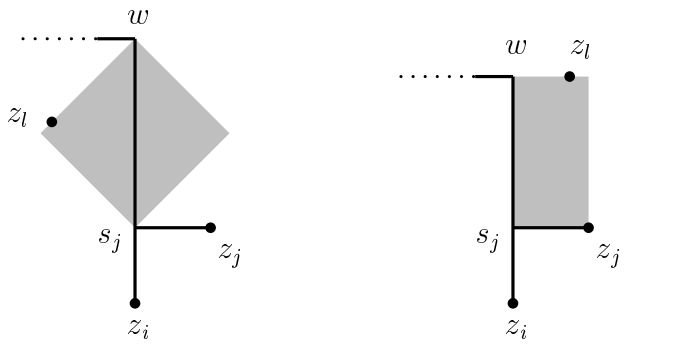
Figure 15: Short leg terminal candidates.

The short leg candidates $Z_s(z_i, \alpha)$ are identified by making a sweep from z_i in direction α. We use the first condition above to stop the scan when $|z_i s_j| > d_{UB0}(z_i, \alpha)$. One implication of the third condition is that the distance $|z_j s_j|$ for any accepted terminal z_j must be smaller than or equal to the shortest such distance seen during the sweep, otherwise the corner rectangle would be non-empty. The first two conditions are easily checked in constant time by using precomputed information.

Once $Z_s(z_i, \alpha)$ is determined, upper bounds $d_{UB1}(z_i, \alpha)$ on the length of short legs (with tip z_i and pointing in direction α) which have exactly one attached terminal are obtained. Recall that $d_{UB0}(z_i, \alpha)$ already is an upper bound on the length of a short leg without any attached terminal.

If $Z_s(z_i, \alpha) = \emptyset$ we set $d_{UB1}(z_i, \alpha) = 0$; this means that no short leg with one attached terminal exists. Otherwise we seek the longest $z_i w$ such that there is a $z_l \in Z_s(z_i, \alpha)$ having both an empty lune $\mathcal{L}_1(s_l, w)$ and an empty corner rectangle $\mathcal{R}(z_l, w)$. Let z_j be the last terminal accepted into $Z_s(z_i, \alpha)$, i.e., $|z_i s_j|$ is maximum. Then it is sufficient to check the empty regions for z_j only, since for all other $z_l \in Z_s(z_i, \alpha) \setminus \{z_j\}$ the regions $\mathcal{L}_1(s_j, w)$ and $\mathcal{R}(z_j, w)$ are covered by $\mathcal{L}_1(s_l, w)$ and $\mathcal{R}(z_l, w)$, respectively.

The upper bound $d_{UB1}(z_i, \alpha)$ is found by making a sweep from s_j (or equivalently z_j) in direction α. The largest possible lune $\mathcal{L}_1(s_j, w)$ (Figure 16a) is found by using the algorithm described in Section 5.1. The empty corner rectangle property is equivalent to testing whether there exists a terminal z_l and corner point w such that $|z_l w| < |z_j s_j|$; when such a terminal is encountered during the sweep we set $d_{UB1}(z_i, \alpha) = |z_i w|$ and stop (Figure 16b).



a) Empty diamond *b) Empty corner rectangle*

Figure 16: Short leg upper bound.

6 Growing Full Steiner Trees

The basic idea of the FST growing algorithm was presented in Section 4. In this section we give a more detailed description of the algorithm which uses information obtained in the preprocessing phase. The main algorithm makes a call to the recursive procedure *grow_FST* $(\{z_0\},\ \alpha)$ for every root $z_0 \in Z$ and $\alpha \in \{0, 1\}$ (Figure 17).

The procedure performs four different types of tests; these will now be described in detail. Tests for degeneracies and duplicate tests for small FSTs have not been included in the description.

Let β be the direction from z_i to the Steiner point s_i on the long leg. If z_i is to the left when looking in direction α from z_0 then $\beta = \alpha - 1$ and otherwise $\beta = \alpha + 1$.

6.1 Distance Tests

The upper bounds d_{UB0} and d_{UB1} obtained in the preprocessing phase are used to eliminate terminals from consideration. We must have

$$|z_i s_i| \leq \max(d_{UB0}(z_i, \beta), d_{UB1}(z_i, \beta))$$

otherwise the segment $z_i s_i$ can neither be attached to the long leg directly nor be a short leg in a type (ii) FST.

This test only depends on the root z_0 and on the direction α of the long leg and not on the current partial tree T_{i-1}. This actually allows us to make a (short) list of candidates before calling *grow_FST* $(\{z_0\},\ \alpha)$. However, since the FST

```
procedure grow_FST (T_{i-1}, α)
  // T_{i-1} has segments z_1 s_1, ..., z_{i-1} s_{i-1} attached to the long leg
  z_i = z_{i-1}^α // successor to z_{i-1} in direction α
  while (z_i ≠ nil) do
    // Assume that z_i is attached to the long leg via Steiner point s_i
    if (z_i passes distance tests (Section 6.1)) then
      if (s_{i-1} s_i passes long leg segment tests (Section 6.2)) then
        T_i = T_{i-1} ∪ {s_{i-1} s_i} ∪ {z_i s_i} // new partial tree
        if (short leg candidate z_j exists) then
          // Assume that z_j is attached to short leg z_i s_i via Steiner point s_j
          if (T_i ∪ {z_j s_j} passes type (ii) FST tests (Section 6.4)) then
            Save T_i ∪ {z_j s_j} as a permanent type (ii) FST
          endif
        endif
        if (z_i s_i passes attachment tests (Section 6.3)) then
          if (T_i passes type (i) FST tests (Section 6.4)) then
            Save T_i as a permanent type (i) FST
          endif
          Update short leg upper bound and longest edge information
          grow_FST (T_i, α) // make recursive call
        endif
      endif
    endif
    z_i = z_i^α // get next terminal candidate
  enddo
end
```

Figure 17: FST growing algorithm.

growing algorithm typically stops well before reaching the end of the candidate list, we do not construct this list before calling *grow_FST* $(\{z_0\}, \alpha)$. Instead we add candidates to the list when needed, that is, whenever we are at the end of the list.

When a terminal candidate is added to the candidate list the corresponding short leg candidate (if it exists) is also identified. This is done by scanning the short leg candidate list $Z_s(z_i, \beta)$ constructed in the preprocessing phase.

6.2 Long Leg Segment Tests

This series of tests depends on the partial tree T_{i-1} and the new long leg segment $s_{i-1}s_i$. A long leg segment $s_{i-1}s_i$ fails this test if the tree $T_{i-1} \cup \{s_{i-1}s_i\}$ cannot be a subtree of *any* larger FST that also spans z_i. We assume that z_i is either connected directly to s_i or via a short leg Steiner point.

The first and obvious condition is that z_i must be on the opposite side of the long leg as z_{i-1}. The empty diamond property must be satisfied for $s_{i-1}s_i$ and $\mathcal{R}(z_{i-1}, z_i)$ should contain no terminals. The former condition can be checked by maintaining an upper bound on the length of the segment $s_{i-1}s_i$ based on the previous terminal candidates seen for z_i (see also Section 5.1).

Finally we have the following strong condition: The longest edge on the path between s_i and any terminal z_l, $l \leq i-1$ in $T_{i-1} \cup \{s_{i-1}s_i\}$ cannot be longer than $b_{z_i z_l}$. This condition holds since the same (longest) edge will also appear on the path between z_i and z_l in any tree having $T_{i-1} \cup \{s_{i-1}s_i\}$ as a subtree. By updating this longest edge information dynamically while growing the tree this test can be performed very efficiently.

6.3 Attachment Tests

These tests check if the partial tree $T_i = T_{i-1} \cup \{s_{i-1}s_i\} \cup \{z_i s_i\}$ can be a subtree of any larger FST (including T_i itself). This is done by testing whether $|z_i s_i| \leq d_{UB0}(z_i, \beta)$ and if $|z_i s_i| \leq b_{z_i z_l}$ for all z_l, $l \leq i-1$.

6.4 FST Tests

These tests check the optimality of a specific FST candidate. The most efficient tests are based on the short leg upper bounds presented in Section 3.3. These upper bounds are dynamically updated while growing the tree so this test can be performed in constant time for a specific FST candidate.

The empty diamond property (resp. empty corner rectangle property) is tested for every segment (resp. pair of adjacent segments), also in the corner-flipped topology. Note that these conditions are satisfied by construction for every edge in the primary topology, except for segments in or connected to the short leg in a type (ii) FST. Furthermore, the empty inner rectangle property is checked.

Finally the longest edge test and minimum spanning tree test using bottleneck Steiner distances are performed (Section 3). It should be noted that we also tried to compute an upper bound on the FST length by using one of the available heuristics for the rectilinear Steiner tree problem, but since all FSTs spanning up to five terminals are optimal by construction (Section 3.3) this only had a negligible effect on the number of surviving FSTs.

7 Expected Number of Full Steiner Trees

In this section we give a theoretical bound on the expected number of surviving FSTs for randomly generated instances (recall that FSTs are assumed to have Hwang topology). Our interest in the *expected* number of surviving FSTs is due to the fact that current worst-case bounds are exponential.

Fößmeier and Kaufmann [4] constructed an infinite series of instances for which the number of FSTs fulfilling a so-called *tree star* condition is *exponential*. An FST F with Steiner points $\{s_1, s_2, \ldots, s_k\}$ is a tree star if all corner rectangles are empty and $MST(Z \cup \{s_1, s_2, \ldots, s_k\})$ contains every edge of F. The latter condition is equivalent to the following condition which is much faster to verify when bottleneck Steiner distances are given (see also Section 3): For every pair of terminals z_i and z_j spanned by F the longest edge on the unique path between z_i and z_j is not longer than $b_{z_i z_j}$. Fößmeier and Kaufmann also presented experimental evidence showing that the average number of tree stars for randomly generated instances is almost linear. Thus there is a huge gap between the worst-case bound (which is exponential) and the average number of surviving FSTs for randomly generated instances (which is almost linear).

Computational experience reported in Section 8 indicates that even when using the strong screening tests described in the previous sections it does not seem possible to prove a worst-case polynomial upper bound. However, we prove that the expected number of FSTs fulfilling a weak version of the tree star condition and spanning up to K terminals is $O(n(\log\log n)^{K-2})$, that is, almost linear (where $K \geq 3$ is a constant). Thus this is also a bound on the number of FSTs fulfilling the original tree star condition given above. Note that the trivial worst-case bound is $O(n^K)$.

Unfortunately, we have not been able to prove that the expected *total* number of

FSTs is polynomial. But our bound is a significant improvement on two bounds given by Salowe and Warme [18]. They gave an $O(n \log n)$ bound on the expected number of FSTs spanning exactly three terminals and an $O(n^2)$ bound on the expected number of FSTs spanning up to K terminals (where K is a constant). In addition, they gave an $O(1)$ bound on the expected number of FSTs spanning $\Omega(n)$ terminals.

We first give a bound on the expected length of the longest MST-edge (Section 7.1). This is also a bound on the longest SMT-edge and it holds for any metric L_p for $1 \leq p \leq \infty$; similar proof techniques as in [3] are applied. Then we present some previously known properties for empty rectangles (Section 7.2) and finally give our main result in Section 7.3.

7.1 Bounding the Longest MST-edge

The probability that there exist long MST-edges is bounded by the following theorem, which is proved in the Appendix.

Theorem 1 *Let Z be a set of $n > 4$ terminals randomly distributed with uniform distribution within the unit square. Let C be a constant and $B = C\sqrt{\log n/n}$. The probability that there exists an MST-edge (z_i, z_j) under the L_p metric, $1 \leq p \leq \infty$, such that $\|z_i z_j\|_p > B$ is bounded by $n^{2-C^2/16}$.*

Corollary 1 *With high probability the longest edge in an MST (and SMT) under any metric L_p, $1 \leq p \leq \infty$, is $O(\sqrt{\log n/n})$. Also the* expected *length of the longest edge in an MST (and SMT) is $O(\sqrt{\log n/n})$.*

Proof. Choose $C = \sqrt{41}$, such that $2 - C^2/16 < -1/2$. Then the probability that there exists an MST-edge longer than $C\sqrt{\log n/n}$ is $n^{2-C^2/16} \to 0$ as $n \to \infty$.

The expected length of the longest MST-edge is bounded by

$$\begin{aligned} & (1 - n^{2-C^2/16}) \times C\sqrt{\log n/n} \; + \; n^{2-C^2/16} \times \sqrt{2} \\ \leq \; & C\sqrt{\log n/n} + \sqrt{2/n} \\ \leq \; & (C+2)\sqrt{\log n/n} \end{aligned}$$

■

7.2 Empty Rectangle Properties

Two terminals $z_i, z_j \in Z$ make an *empty rectangle* if $\mathcal{R}(z_i, z_j)$ contains no other terminal from Z. Empty rectangles are related to so-called *maximal points* in a

point set. A maximal point is a point which is not dominated by any other point in the set, i.e., there is no point which has both a greater x- and y-coordinate.

Consider a point $z_i \in Z$. Let $Z_i \subset Z$ be the set of points dominated by z_i. These are all points in Z located in the third quadrant of z_i. Then all maximal points in Z_i are exactly those that make an empty rectangle with z_i. By making appropriate transformations we see that the problem of determining empty rectangles is equivalent to finding maximal points.

Bentley et al. [2] proved that the expected number of maximal points in a point set with n uniformly distributed points is $O(\log n)$. Thus we have:

Lemma 2 *Let $z_i \in Z$ be a terminal. The expected number of terminals making empty rectangles with z_i is $O(\log n)$.*

7.3 Expected Number of FSTs

The bound on the expected number of FSTs is obtained by using the bound on the longest MST-edge and the bound on the number of terminals making an empty rectangle with a given terminal. Denote by b_{max} the largest bottleneck Steiner distance (which is the same as the length of the longest MST-edge).

Assume that z_i and z_{i+1} are two successive terminals attached to the long leg of an FST in an SMT for Z (z_i may be the root of the FST). Then both the horizontal and vertical distance between z_i and z_{i+1} is at most $2b_{max}$ (more precisely the distance along the long leg is at most b_{max} and the perpendicular distance at most $2b_{max}$). Also z_i and z_{i+1} define an empty rectangle.

Since the expected value of b_{max} is bounded by $O(\sqrt{\log n/n})$ (Corollary 1) the expected number of candidates for z_{i+1} when z_i is given is $O((\sqrt{\log n/n})^2 \times n) = O(\log n)$. In addition, z_i and z_{i+1} must make an empty rectangle and thus by Lemma 2 the expected number of candidates for z_{i+1} is $O(\log \log n)$.

Let K be a constant, $K \geq 3$. Since any type (i) FST can be obtained by first choosing a root and then growing the FST from this root, selecting alternating segments along the long leg in order, we obtain the bound $O(n(\log \log n)^{K-1})$ on the expected number of type (i) FSTs spanning up to K terminals. For type (ii) the distance between last terminal attached to the long leg and the tip in the direction perpendicular to the long leg may be as large as $3b_{max}$; however, this does not change the $O(\log \log n)$ bound. On the other hand, the terminal attached to the short leg is uniquely given once the backbone is constructed. Thus we obtain the stronger bound $O(n(\log \log n)^{K-2})$ on the number of type (ii) FSTs.

We can obtain the same bound for type (i) FSTs by "growing" the FST in a different order. Having chosen the root we *first* choose the tip of the FST. Since both the vertical and horizontal distance to this terminal is at most $(K-1)b_{max}$ there are only $O(\log\log n)$ candidates for the tip since K is a constant. When choosing the terminals along the long leg the *last* terminal is now uniquely given (since we have already chosen the tip). The main result of this section follows:

Theorem 2 *Assume that the terminals are randomly distributed with uniform distribution within the unit square. Let K be a constant, $K \geq 3$. Then the expected number of FSTs (i.e., Hwang topologies) spanning up to K terminals for which no edge is longer than the longest MST-edge and all corner rectangles are empty is bounded by $O(n(\log\log n)^{K-2})$.*

Since any FST fulfilling the tree star condition also fulfills the conditions in this theorem we have the same bound on the number of tree stars.

8 Computational Experience

The new rectilinear FST generator was experimentally evaluated on an HP9000 workstation[3] using the programming language C++ and class library LEDA (version 3.4.1) [14]. The random number generator used was the `random_source` class in LEDA.

The test-bed consists of problem instances from three sources: Public library instances (Section 8.1), randomly generated instances (Section 8.2) and a series of constructed pathological instances producing an exponential number of surviving FSTs (Section 8.3).

8.1 Public Library Instances

The first series of problems instances is from the OR-Library [1]. FSTs for each of the 46 instances by Soukup and Chow [20] (3-62 terminals) were generated within 0.2 seconds and the number of surviving FSTs similar to those given by Warme [23]. Results for the other series of instances from the *OR-Library* (randomly generated instances, 15 for each size for $n \leq 1000$ and one 10000 terminal instance) are given in Table 1.

[3]Machine: HP 9000 Series 700 Model 735/99. Processor: 99 MHz PA-RISC 7100. Main memory: 96 MB. Performance: 3.27 SPECint95 (109.1 SPECint92) and 3.98 SPECfp95 (169.9 SPECfp92). Operating system: HP-UX 9.0. Compiler: GNU C++ 2.7.2 (optimization flag -O3).

The total number of FSTs is almost linear with relatively small variation. The average FST size (number of terminals) increases very slowly and stabilizes below four terminals. The CPU-time for the preprocessing phase is — as expected — quadratic. Interestingly, the CPU-time for the FST growing phase is *sub-quadratic* and for $n = 10000$ the CPU-time for the preprocessing phase clearly dominates the CPU-time for the FST growing phase. We discuss this interesting observation in detail in Section 8.2. The total CPU-times are only fractions of the CPU-times reported in [23]; recall that the FST generation algorithm in [23] required a few hours on a 1000 terminal instance. In addition, the FST-counts reported here are approximately 15% smaller.

n	FST-count		FST-size		CPU-prep		CPU-grow		CPU-total	
10	23	± 7	2.79	± 0.29	0.00	± 0.00	0.01	± 0.00	0.01	± 0.00
20	62	± 16	3.17	± 0.36	0.01	± 0.01	0.02	± 0.01	0.04	± 0.02
30	103	± 23	3.34	± 0.32	0.02	± 0.00	0.04	± 0.01	0.06	± 0.01
40	135	± 22	3.30	± 0.25	0.03	± 0.01	0.06	± 0.02	0.09	± 0.01
50	168	± 24	3.32	± 0.20	0.05	± 0.01	0.07	± 0.02	0.12	± 0.02
60	225	± 32	3.51	± 0.27	0.06	± 0.01	0.11	± 0.03	0.16	± 0.03
70	254	± 31	3.44	± 0.21	0.07	± 0.01	0.12	± 0.02	0.19	± 0.02
80	293	± 34	3.49	± 0.23	0.09	± 0.01	0.15	± 0.03	0.23	± 0.03
90	326	± 41	3.44	± 0.24	0.10	± 0.01	0.16	± 0.03	0.25	± 0.03
100	386	± 63	3.55	± 0.24	0.11	± 0.01	0.20	± 0.05	0.31	± 0.05
250	963	± 76	3.52	± 0.12	0.44	± 0.01	0.56	± 0.06	1.00	± 0.06
500	2006	± 112	3.61	± 0.11	1.28	± 0.05	1.37	± 0.14	2.65	± 0.17
1000	4172	± 220	3.69	± 0.11	4.21	± 0.04	3.50	± 0.30	7.71	± 0.30
10000	40933		3.65		363.75		82.24		445.99	

Table 1: *OR-Library* instances. FST-count: Number of FSTs generated (including MST-edges). FST-size: Average number of terminals spanned by generated FSTs. CPU-prep: Preprocessing CPU-times. CPU-grow: FST-growing CPU-times. All CPU-times are in seconds. Second numbers in each column are standard deviations.

The FST generator was also evaluated on 26 instances from *TSPLIB* [15] (198-7397 terminals). This library is a collection of instances for the Traveling Salesman Problem (TSP), mainly plane real-world Euclidean problem instances. The 26 selected instances are the same as those chosen in a study by Reinelt [16] on heuristics for the TSP; in addition we have chosen the instance *pla7397*, the largest TSP instance solved to optimality to date. These instances are quite representative for the whole *TSPLIB* collection.

Computational results are presented for each instance in Table 2. For many of the instances the FST-count, FST-size and total CPU-time is lower than the average for randomly generated instances of the same size. The only exceptions are the instances *rat783* and *fnl4461* which have their points distributed in a random and uniform fashion. Thus instances with a less random and less uniform distribution

are in general easier, in particular *rl1323*, *u1432* and *rl5934* which have many co-linear and equidistant terminals.

Instance	FST-count	FST-size	CPU-prep	CPU-grow	CPU-total
d198	595	2.80	0.29	0.25	0.54
lin318	1378	3.37	0.51	0.83	1.34
fl417	1624	3.04	0.91	0.73	1.64
pcb442	914	2.57	1.02	0.41	1.43
att532	2267	3.76	1.34	1.54	2.88
u574	1733	3.11	1.58	1.16	2.74
p654	2103	2.89	1.84	1.43	3.27
rat783	4560	4.08	2.56	4.01	6.57
pr1002	3154	3.05	3.85	1.78	5.63
u1060	3453	3.13	4.41	2.11	6.52
pcb1173	3308	3.13	5.21	1.97	7.18
d1291	4266	2.73	5.83	2.19	8.02
rl1323	2541	2.64	6.75	1.69	8.44
fl1400	8689	3.16	7.36	5.32	12.68
u1432	3604	2.60	7.08	1.75	8.83
fl1577	5665	2.89	9.30	3.05	12.35
d1655	4142	2.66	9.63	2.29	11.92
vm1748	4646	3.17	10.49	3.83	14.32
rl1889	3659	2.69	11.92	2.71	14.63
u2152	5855	2.63	15.21	2.79	18.00
pr2392	6782	2.95	14.46	4.55	19.01
pcb3038	10694	3.34	28.06	8.92	36.98
fl3795	12993	2.86	49.08	8.57	57.65
fnl4461	29229	4.59	61.84	46.74	108.58
rl5934	11584	2.60	115.40	12.04	127.44
pla7397	20497	2.76	174.69	22.92	197.61

Table 2: *TSPLIB* instances.

8.2 Randomly Generated Instances

One hundred instances were generated for each size $1000, 2000, \ldots, 10000$. Terminals were drawn with uniform distribution from the unit square. It should be noted that we did not choose the coordinates from a 10000×10000 grid which is common in the literature, since this would impose a significant number of co-linear terminals for larger instances. However, as far as the number of surviving FSTs is concerned, preliminary experiments showed that it did not seem to make any noticeable difference whether a grid was used or not.

Computational results presented in Table 3 and Figures 18 and 19 show a very regular pattern. The total number of FSTs grows almost linearly, the FST preprocessing time quadratically and the FST growing time sub-quadratically. In fact, the FST growing time seems to be n times some poly-logarithmic factor.

This behaviour can be explained by studying two interesting statistics. The first is the average number of short leg candidates (Section 5.2). This number increases extremely slowly: From $n = 1000$ to $n = 10000$ the number of candidates increases from 0.89 to 0.90 candidates. Note that the analysis in Section 7 gives a theoretical upper bound of $O(\log \log n)$.

Secondly, we have the average number of long leg candidates considered for a given root and direction (Section 6.1). Surprisingly, this number also grows very slowly: From $n = 1000$ to $n = 10000$ this number increases from 5.28 to 5.59. Currently, we do not have any tight theoretical upper bound on this value, but this shows that the FST growing procedure is cut-off very early and that the running time therefore is close to being linear.

n	FST-count		FST-size		CPU-prep		CPU-grow		CPU-total	
1000	4092	± 188	3.66	± 0.09	4.10	± 0.07	3.36	± 0.26	7.46	± 0.29
2000	8286	± 244	3.68	± 0.06	13.60	± 0.30	8.35	± 0.41	21.95	± 0.49
3000	12405	± 277	3.67	± 0.04	29.39	± 0.15	14.52	± 0.44	43.91	± 0.48
4000	16626	± 348	3.69	± 0.04	49.11	± 0.57	21.88	± 0.75	70.99	± 1.09
5000	20793	± 412	3.69	± 0.04	79.38	± 0.38	29.87	± 0.87	109.26	± 0.91
6000	24970	± 504	3.69	± 0.04	121.39	± 1.17	39.61	± 1.17	161.01	± 1.62
7000	29126	± 552	3.69	± 0.04	171.33	± 0.98	49.24	± 1.45	220.57	± 1.88
8000	33374	± 582	3.69	± 0.04	232.10	± 2.63	60.93	± 1.88	293.03	± 3.50
9000	37550	± 564	3.69	± 0.03	304.28	± 6.16	73.01	± 2.82	377.29	± 8.12
10000	41652	± 590	3.69	± 0.03	359.51	± 7.87	84.65	± 3.77	444.16	± 10.45

Table 3: Randomly generated instances.

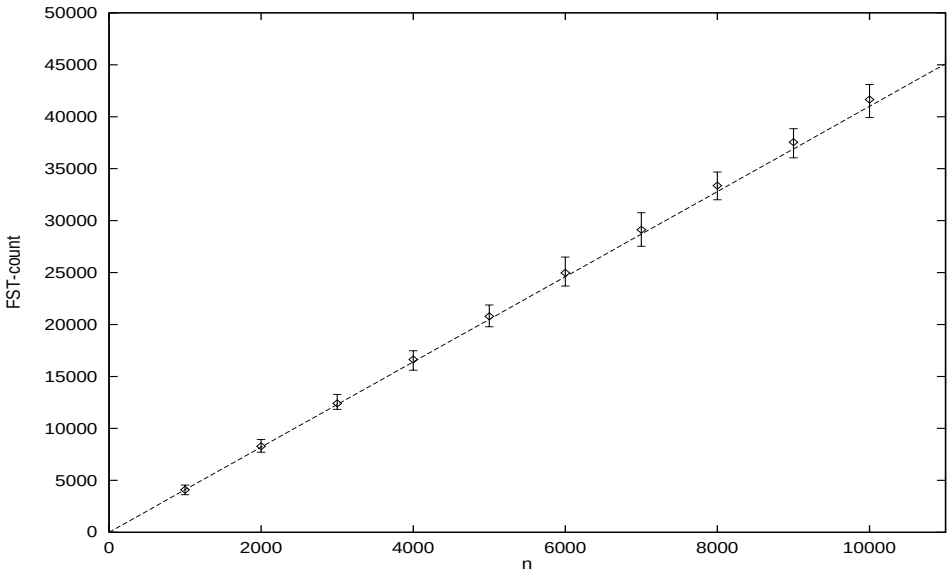


Figure 18: Total FST-count.

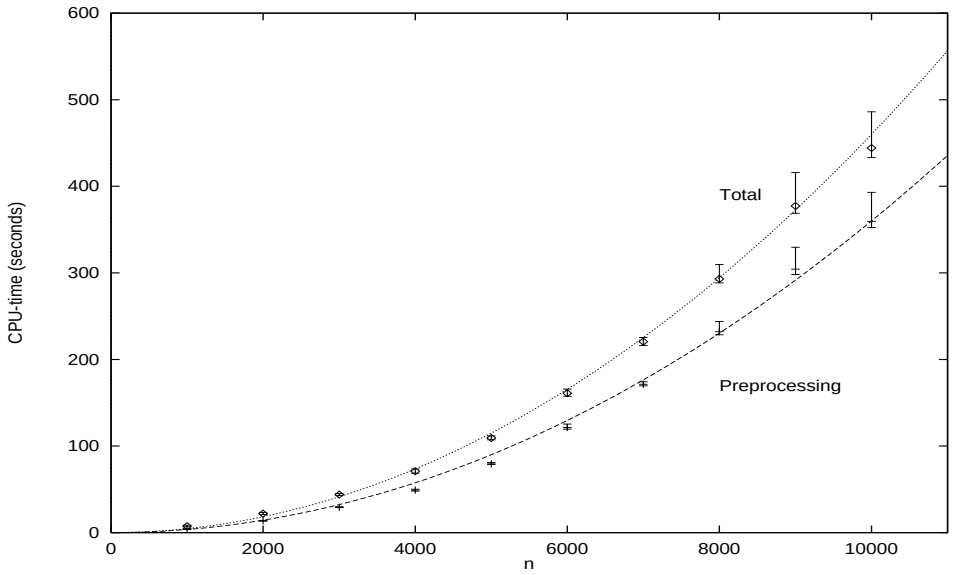


Figure 19: CPU-times: Preprocessing and total.

The average number of terminals spanned by the FSTs is approximately 3.7. The FST-size distribution for $n = 10000$ is shown in Figure 20. Approximately 90% of the FSTs span five or fewer terminals and less than 1% span more than ten terminals. From $n = 1000$ to $n = 10000$ the relative (i.e., normalized by n) increase in the total number of FSTs is approximately 2%. This increase mainly comes from a relative increase in the number of large FSTs: There are only 1% more FSTs spanning three terminals while there are 5% more FSTs spanning, e.g., six terminals.

For large instances the running time of the preprocessing phase clearly dominates the running time of the FST growing phase and we may therefore ask if it is possible to reduce the former without increasing the latter too much. The dominating part of the preprocessing phase is the $\Theta(n^2)$ time and space procedure for computing bottleneck Steiner distances.

By using the (dynamic) search tree data structure by Sleator and Tarjan [19], this part of preprocessing phase can be reduced to $O(n \log n)$ time and $O(n)$ space, but longest edge queries then take $O(\log n)$ amortized time instead of $O(1)$ (matrix lookup). We used the `dynamic_trees` class in LEDA in order to evaluate this option.

For $n = 1000$ the running time of the preprocessing phase unfortunately remained approximately the same while the running time of the FST growing phase increased from 3 to approximately 15 seconds. Thus the large constant factor involved in the search tree data structure does not make this approach favourable for this instance size.

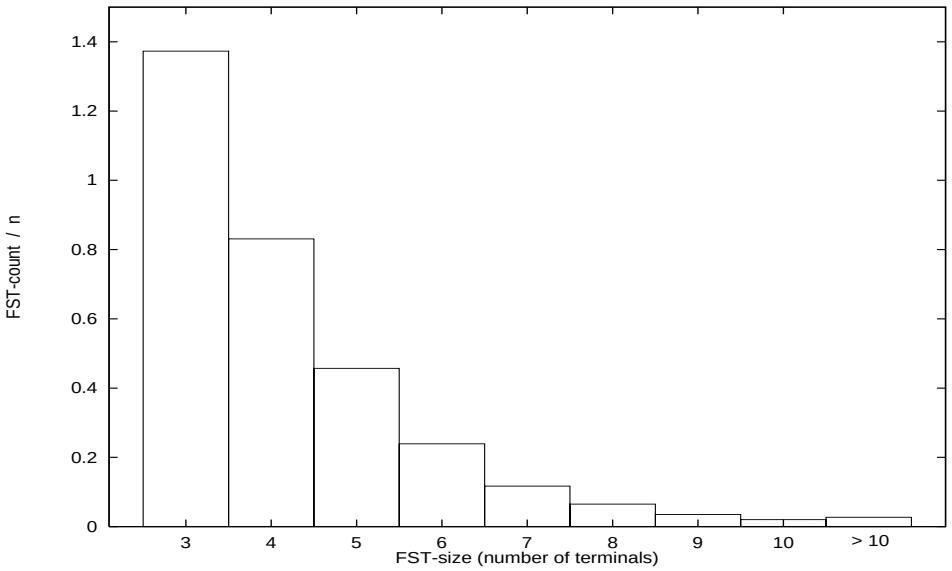


Figure 20: FST size distribution for $n = 10000$.

When $n = 10000$ the running time of the preprocessing phase was halved while the running time FST growing phase approximately doubled. The total CPU-time was reduced from 7.5 minutes to 6.0 minutes. Consequently, for large instances it is possible to reduce the total running time, but this requires the application of relatively advanced data structures. We leave this topic open for future research.

8.3 Pathological Instances

Fößmeier and Kaufmann [4] constructed an infinite series of "bad" instances with respect to the so-called tree star condition (see Section 7). They showed that the number of tree stars was $\Omega(1.32^n)$ for the series of instances given in Figure 21. The base case consists of the 12 leftmost terminals; the 10 rightmost of these are repeatedly scaled down and added to the right.

In Table 4 we give statistics for the first six of these instances with 12, 22, 32, 42, 52 and 62 terminals. Apparently, the number of surviving FSTs is exponential, approximately $\Omega(1.06^n)$. This slower growth was to be expected because all FSTs generated by the new generator are tree stars and, in addition, a number of other optimality tests are fulfilled. These computational results present evidence that it may be difficult to find simple conditions which will guarantee a polynomial number of surviving FSTs.

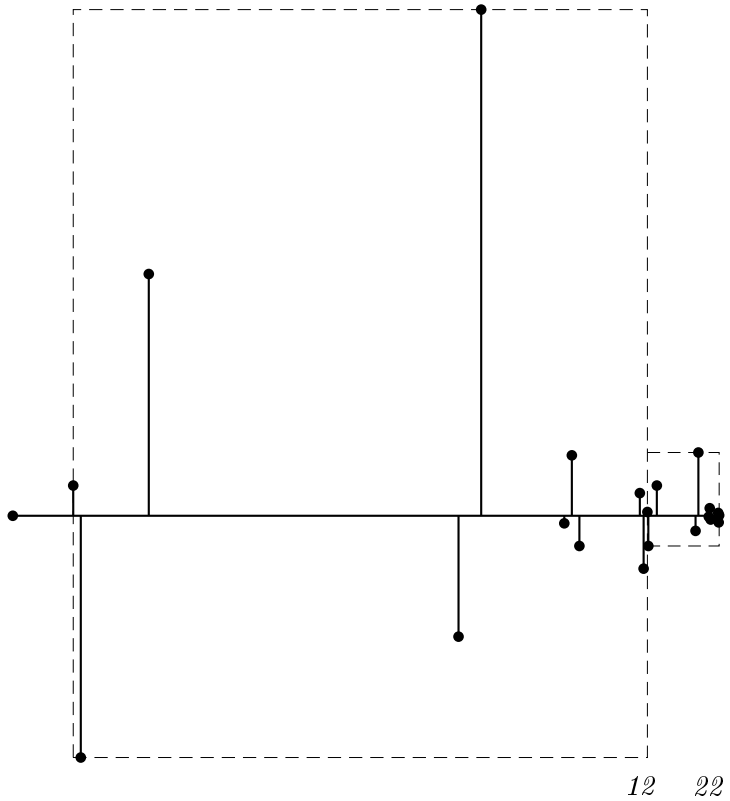


Figure 21: Pathological instances. The small rectangle is a scaled-down version of the large rectangle. The infinite series of instances is constructed by repeatedly adding a scaled-down copy of the terminals inside the current rectangle to the right of the rectangle.

n	FST-count	FST-size	CPU-prep	CPU-grow	CPU-total
12	108	5.37	0.00	0.08	0.08
22	251	7.71	0.02	0.32	0.34
32	475	10.88	0.03	1.09	1.12
42	853	15.10	0.04	3.32	3.36
52	1551	20.09	0.05	10.13	10.18
62	2865	25.69	0.09	28.72	28.81

Table 4: Pathological instances.

9 Conclusion

We presented a new algorithm for generating rectilinear full Steiner trees (FSTs). The algorithm outperforms previous approaches [18, 23] by orders of magnitude. New optimality conditions for rectilinear FSTs were presented. Furthermore, we gave a tighter upper bound on the expected number of FSTs spanning up to K terminals for randomly generated instances (where K is a constant).

The proposed algorithm is very fast and has applications both with respect to exact algorithms and heuristics:

FST based exact algorithms The first and natural application is to use the generator in conjunction with the integer programming based concatenation method by Warme [23]. This would on average allow the exact computation of rectilinear Steiner trees for 100 terminals in less than 10 seconds and for 500 terminals in less than 15 minutes. Also, tight lower bounds (and in some cases optimal solutions) for problems with a few thousand terminals could be computed in less than an hour. Computational results will be presented in a forthcoming paper [22].

FST based heuristics The generator can be used as a first phase for greedy and local search based heuristics. The approach presented in [27, 26] can easily be extended to the rectilinear problem.

Grid graph reduction We can obtain a grid reduction method by overlaying the generated FSTs on the Hanan grid (Figure 22). On average, the number of Steiner points left in the grid is almost linear, approximately $3n$ (e.g., for $n = 1000$ only 0.3% of the Steiner points are retained). The reduced problem may be transformed into the corresponding graph problem; exact algorithms and heuristics for the graph problem can then be applied.

Winter [25] proposed a series of reduction tests for the rectilinear Steiner tree problem. Computational experiments showed that it was possible to reduce the total number of Steiner points to $15 - 20\%$ of the original number (for $n \leq 25$). For this problem size the FST generation method retains $10 - 15\%$ of the Steiner points. By combining the both techniques even better reductions may be obtained.

The best performing heuristic for the rectilinear Steiner tree problem is the iterated 1-Steiner heuristic [12]. An efficient implementation of this heuristic runs in time $O(n^3)$ [6], but requires approximately 30 minutes for a 300 terminal problem (on a SUN4 workstation). Since this heuristic is based on iteratively

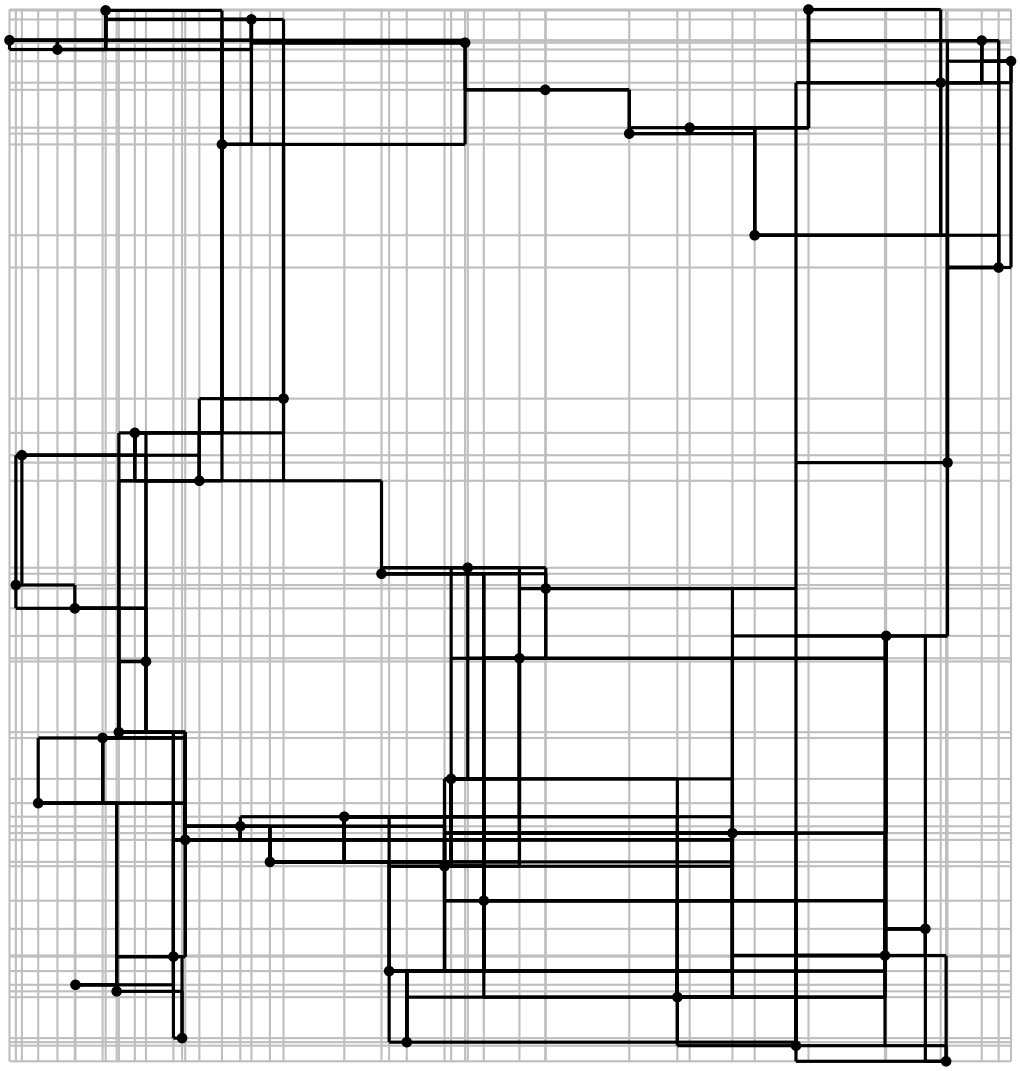


Figure 22: FST generation as a grid reduction algorithm.

choosing Steiner point candidates from the Hanan grid, a simple and practical way of speeding it up would be to preprocess the Hanan grid by generating FSTs. This would reduce the number of Steiner points from $O(n^2)$ to approximately $O(n)$, cutting the running time down by a factor of $O(n)$.

As noted in Section 8.2 the running time of the generator can be further reduced for large instances ($n > 1000$) by using advanced data structures, e.g., for answering bottleneck Steiner distance queries [19, 21]. This would also reduce memory requirements from $\Theta(n^2)$ to $\Theta(n)$.

Obtaining a tight upper bound on the expected (total) number of surviving FSTs is still the most prominent open theoretical problem. The bound given in this paper on the expected number of FSTs spanning up to K terminals is a significant improvement over previous bounds, but it still not entirely satisfactory.

Acknowledgement
The author would like to thank David M. Warme and Pawel Winter for valuable comments and suggestions.

References

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

[2] J. L. Bentley, K. T. Kung, M. Schkolnick, and C. D. Thompson. On the Average Number of Maxima in a Set of Vectors and Applications. *J. Assoc. Comput. Mach.*, 25(4):536–543, 1978.

[3] M. T. Dickerson, R. L. S. Drysdale, S. A. McElfresh, and E. Welzl. Fast Greedy Triangulation Algorithms. Technical Report PCS-TR94-215, Department of Mathematics and Computer Science, Darthmouth College, 1994.

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

[5] 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.

[6] J. Griffith, G. Robins, J. S. Salowe, and T. Zhang. Closing the Gap: Near-Optimal Steiner Trees in Polynomial Time. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(11):1351–1365, 1994.

[7] L. J. Guibas and J. Stolfi. On Computing All North-East Nearest Neighbors in the L_1 Metric. *Information Processing Letters*, 17:219–223, 1983.

[8] R.-H. Güting, O. Nurmi, and T. Ottmann. Fast Algorithms for Direct Enclosures and Direct Dominances. *Journal of Algorithms*, 10:170–186, 1989.

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

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

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

[12] A. B. Kahng and G. Robins. A New Class of Iterative Steiner Tree Heuristics with Good Performance. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(7):893–902, 1992.

[13] Z.-C. Liu and D.-Z. Du. On Steiner Minimal Trees with L_p Distance. *Algorithmica*, 7(2/3):179–191, 1992.

[14] 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.

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

[16] G. Reinelt. *The Traveling Salesman, Computational Solutions for TSP Applications*. Lecture Notes in Computer Science 840. Springer-Verlag, 1994.

[17] D. S. Richards and J. S. Salowe. A Linear-Time Algorithm to Construct a Rectilinear Steiner Minimal Tree for k-Extremal Point Sets. *Algorithmica*, 7(2/3):247–276, 1992.

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

[19] D. D. Sleator and R. E. Tarjan. A Data Structure for Dynamic Trees. *Journal of Computer and System Sciences*, 26:362–391, 1983.

[20] J. Soukup and W. F. Chow. Set of Test Problems for the Minimum Length Connection Networks. *ACM/SIGMAP Newsletter*, 15:48–51, 1973.

[21] A. K. Tsakalidis. The Nearest Common Ancestor in a Dynamic Tree. *Acta Informatica*, 25:37–54, 1988.

[22] D. M. Warme, P. Winter, and M. Zachariasen. Exact Algorithms for Plane Steiner Tree Problems: A Computational Study. In D.-Z. Du, J. M. Smith, and J. H. Rubinstein, editors, *Advances in Steiner Trees*, Kluwer Academic Publishers, Boston, to appear.

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

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

[25] P. Winter. Reductions for the Rectilinear Steiner Tree Problem. *Networks*, 26:187–198, 1995.

[26] M. Zachariasen. Local Search for the Steiner Tree Problem in the Euclidean Plane. Technical Report 97/21, DIKU, Department of Computer Science, University of Copenhagen, 1997.

[27] M. Zachariasen and P. Winter. Concatenation-Based Greedy Heuristics for the Euclidean Steiner Tree Problem. Technical Report 97/20, DIKU, Department of Computer Science, University of Copenhagen, 1997.

Appendix

In this appendix we prove Theorem 1. The unit square is denoted by $\mathcal{U}$ and the area of a planar region R denoted by $Area(R)$.

Let $z_i, z_j \in Z$ be any pair of terminals in $\mathcal{U}$ sharing an edge in some MST for Z using distance metric L_p, $1 \leq p \leq \infty$. First we note that $\|z_i z_j\|_p \leq \sqrt{2}\|z_i z_j\|$ (e.g., see [13]). Let $\mathcal{D}(z_i, z_j)$ be the interior of the square having the line segment $z_i z_j$ as a diameter. We have $\mathcal{D}(z_i, z_j) \subseteq \mathcal{L}_p(z_i, z_j)$ (Figure 23) and since $\mathcal{L}_p(z_i, z_j)$ is empty (contains no terminals from Z) we also have that $\mathcal{D}(z_i, z_j)$ is empty.

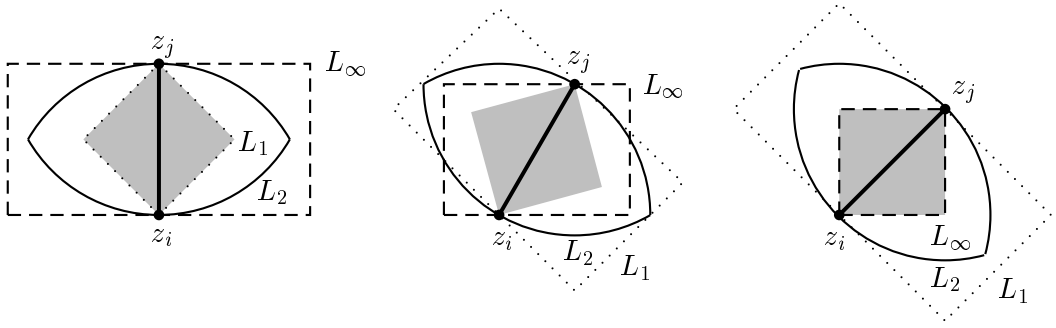


Figure 23: Lunes for L_1, L_2 and L_∞ metrics. The square $\mathcal{D}(z_i, z_j)$ is also shown (shaded).

The area covered by $\mathcal{D}(z_i, z_j)$ is $Area(\mathcal{D}(z_i, z_j)) = \|z_i z_j\|^2/2$. No matter how z_i and z_j are located in $\mathcal{U}$, at least one of the two triangles on each side of the segment $z_i z_j$ will be entirely inside $\mathcal{U}$. Thus at least half of $\mathcal{D}(z_i, z_j)$ is inside $\mathcal{U}$, or $Area(\mathcal{D}(z_i, z_j) \cap \mathcal{U}) \geq \|z_i z_j\|^2/4$.

Now we bound the probability that z_i and z_j are further apart than distance B:

$$\begin{aligned}
& Pr(\ (z_i, z_j) \text{ is an MST-edge and } \|z_i z_j\|_p > B\ ) \\
\leq\ & Pr(\ (z_i, z_j) \text{ is an MST-edge } \mid \sqrt{2}\|z_i z_j\| > B\ ) \\
\leq\ & Pr(\ \mathcal{D}(z_i, z_j) \text{ is empty } \mid \|z_i z_j\| > B/\sqrt{2}\ ) \\
\leq\ & (1 - (B/\sqrt{2})^2/4)^{n-2} \\
\leq\ & e^{-B^2(n-2)/8} \\
=\ & e^{-C^2 \log n (n-2)/8n} \\
\leq\ & n^{-C^2/16}
\end{aligned}$$

Finally we bound the probability that there exists an MST-edge with terminals further apart than distance B:

$$Pr(\ \exists \text{ MST-edge } (z_i, z_j) \text{ such that } \|z_i z_j\|_p > B\ ) \ \leq\ \binom{n}{2} n^{-C^2/16} \ \leq\ n^{2-C^2/16}$$

■