Algorithmica (1999) 25: 418–437

Algorithmica

Concatenation-Based Greedy Heuristics for the Euclidean Steiner Tree Problem

M. Zachariasen[1] and P. Winter[1]

Abstract. We present a class of $O(n \log n)$ heuristics for the Steiner tree problem in the Euclidean plane. These heuristics identify a small number of subsets with few, geometrically close, terminals using minimum spanning trees and other well-known structures from computational geometry: Delaunay triangulations, Gabriel graphs, relative neighborhood graphs, and higher-order Voronoi diagrams. Full Steiner trees of all these subsets are sorted according to some appropriately chosen measure of quality. A tree spanning all terminals is constructed using greedy concatenation. New heuristics are compared with each other and with heuristics from the literature by performing extensive computational experiments on both randomly generated and library problem instances.

Key Words. Heuristics, Steiner trees.

1. Introduction. Given a set Z of n *terminals* in the Euclidean plane, a shortest network which interconnects Z is called a *Steiner minimum tree* (*SMT*). An SMT may contain additional intersection points, *Steiner points*. This *Steiner tree problem* is NP-hard and has been a subject for extensive investigation [9]. The most effective exact algorithm is currently able to solve most problem instances with up to 1000 terminals in a day [24], [23]. If less CPU time is available or larger problem instances have to be solved, heuristics are called for.

An SMT is a union of *full Steiner trees* (*FSTs*). An FST F spanning a subset Z_k of k terminals in Z has $k-2$ Steiner points. Each Steiner point has three edges making 120° with each other. Every terminal in F has degree one (is a leaf in F). If two or three FSTs share a terminal in an SMT, then the edges meet at the terminal at an angle which is at least 120°. Experience has shown that FSTs in an SMT seldomly span more than five terminals [24].

A *minimum spanning tree* (*MST*) for the terminals in Z is a shortest network spanning Z without introducing Steiner points. An MST for Z can be constructed in $O(n \log n)$ time [14], and is a good approximation to an SMT. This is a consequence of the Steiner ratio theorem [9]: Let $SMT(Z)$ and $MST(Z)$ denote an SMT and an MST, respectively, spanning the same set of terminals Z. Then the ratio $|SMT(Z)|/|MST(Z)|$ is always greater than or equal to $\sqrt{3}/2$. Consequently, any MST algorithm, seen as an approximation algorithm for the Steiner tree problem, has a $2/\sqrt{3} \approx 1.1547$ performance ratio.

[1] Department of Computer Science, University of Copenhagen, DK-2100 Copenhagen Ø, Denmark. {martinz,pawel}@diku.dk.

Received October 27, 1997; revised May 7, 1998. Communicated by D. T. Lee.

The length of the MST is therefore a natural reference for the performance of other approximation algorithms, since there is little point in constructing algorithms which produce solutions worse than the MST. It was for a long time an open problem whether there exist approximation algorithms with performance ratios strictly less than $2/\sqrt{3}$. Arora [1] showed that there exists a polynomial-time approximation scheme for the Euclidean traveling salesman problem and other geometric problems—among these the Euclidean Steiner tree problem. This means that we can find in polynomial time (in the number of terminals but not in $1/\varepsilon$) a solution within a factor $1+\varepsilon$ from the optimum for every fixed $\varepsilon > 0$. This result was obtained by a clever partitioning of the plane and applying dynamic programming.

The practical usefulness of the Arora algorithm has yet to be proven. Less sophisticated heuristics[2] on average produce much better solutions than their worst-case performance ratio $2/\sqrt{3}$ indicates. All heuristics described in the sequel have performance ratio $2/\sqrt{3}$. Therefore, their performance will be measured on an experimental basis by computing the reduction over the MST (the most commonly used measure in the literature) and, when available, the excess from the SMT. When references are made to the *average* reduction over the MST, it is assumed that the terminals have been distributed randomly with uniform distribution in a (unit) square. For this distribution and $n = 100$ the average reduction of SMT over MST is approximately 3.2%—the asymptotic value is probably slightly larger [24].

Starting with an MST, a simple approach suggested by Thompson [21] is to look for edges meeting at angles less than 120° and insert Steiner points at appropriate positions (Figure 1(a)). These Steiner point insertions are continued until improvements fall below some threshold value. Chang [6] gave a slightly more general variant: select three vertices (terminals or Steiner points) of the current tree, insert a new Steiner point and corresponding edges, and remove the longest edge on any cycle created (Figure 1(b)). Other MST-based methods have been proposed. The fastest of them, due to Beasley [4], has subquadratic observed running time and a 2.9% average reduction over the MST. Beasley and Goffinet [5] used the approach from [4] in a simulated annealing framework. Their heuristic obtains a 3.0% average reduction at the expense of a huge running time.

The application of geometric structures to heuristics for the Euclidean Steiner tree problem was initiated by Smith and Liebman [17]. A triangulation of Z was used to aid the identification of small subsets of terminals. The triangulation was constructed by using the convex hull for Z. The worst-case running time was $O(n^4)$ due to a very elaborate subset selection procedure. The observed running time was close to being quadratic. The average reduction was rather poor, approximately 2.2%.

A much more efficient heuristic was given by Smith et al. [16]. A similar approach has also been applied to the three-dimensional Euclidean Steiner tree problem [18] and to the plane rectilinear case [15]. This first $O(n \log n)$ heuristic is based on the *Delaunay* triangulation (DT). The average reduction was approximately 2.7%. This is quite impressive, worst-case running time taken into consideration. Since the class of heuristics presented in this paper all originate from the ideas used by this heuristic, we present one of its variants here.

[2] All heuristics for the Steiner tree problem are in fact approximation algorithms since they construct solutions which are no worse than the MST; however, for historic reasons we use the word "heuristics" in the following.

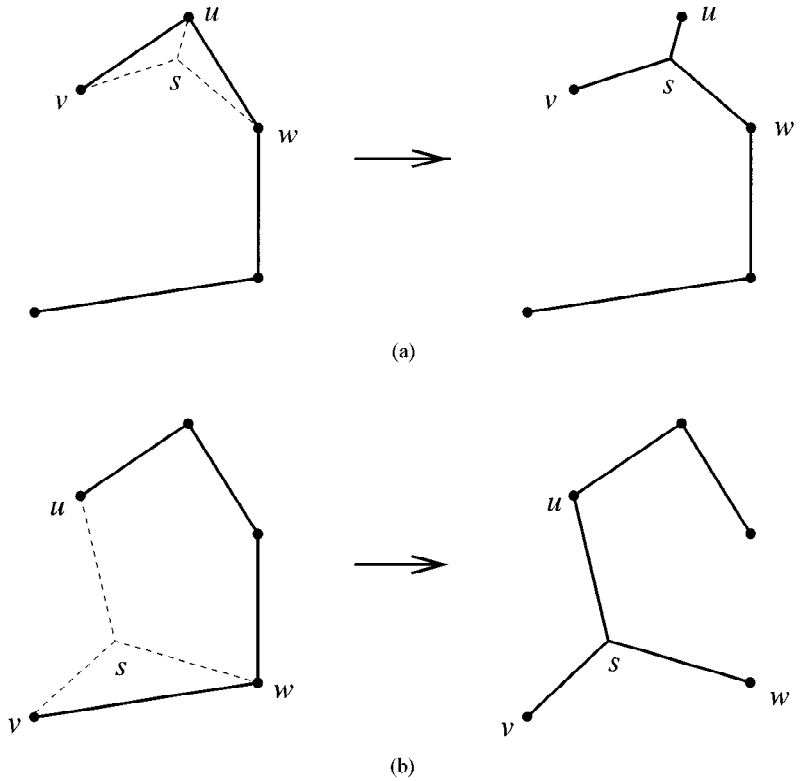

Fig. 1. Simple Steiner point insertions. (a) Steiner point insertion. (b) Generalized Steiner point insertion.

A linear number of subsets with two, three, or four terminals is identified as follows: two-terminal subsets are the MST edges (MST is a subgraph of DT), three-terminal subsets are corners of triangles in the DT with two MST edges, and four-terminal subsets are corners of two edge-sharing triangles in the DT with three connected edges from the MST. For each set Z_k containing k terminals, $k = 2, 3, 4$, the shortest FST is constructed (if it exists); let it be denoted by $FST(Z_k)$. All generated FSTs are placed on a priority queue Q with the priority $|FST(Z_k)|/|MST(Z_k)|$ (smallest first).

The tree T spanning all terminals is then constructed by picking FSTs from Q in a manner similar to Kruskal's MST algorithm. An FST is only added to T if it does not create a cycle. Since a fast disjoint-set data structure is used, the overall worst-case complexity of the concatenation of small FSTs remains $O(n \log n)$.

The new class of heuristics will follow the general outline of the DT heuristic. We show that it is possible to obtain a 3.0% reduction using $O(n \log n)$ time, albeit with a slightly larger constant factor than Smith et al.'s heuristic. In Section 2 we present some well-known structures from computational geometry and discuss their application to the Steiner tree problem. Identification of small subsets of terminals (which are likely to be spanned by a single FST in an SMT) using these structures is discussed in Section 3.

Determination of the shortest FST for every such subset and pruning away nonoptimal FSTs is covered in Section 4. Concatenation of FSTs using a greedy approach is discussed in Section 5. In Section 6 the performance of the heuristics is compared by performing extensive experiments on randomly generated problem instances, including instances from the *OR-Library* [3]. The results are also compared with other heuristics with similar worst-case or observed running time. Concluding remarks are given in Section 7.

2. Proximity Structures. In this section we present some well-known structures from computational geometry which capture proximity relations for a set of terminals Z.

2.1. *Voronoi Diagrams*. Let z_i and z_j denote two distinct terminals. Furthermore, let $H(z_i, z_j)$ denote the set of points not farther from z_i than from z_j. $H(z_i, z_j)$ is a half-plane. Let

$$V(z_i, Z) = \bigcap_{z_j \in Z \setminus z_i} H(z_i, z_j)$$

be the *Voronoi region* of z_i. $V(z_i, Z)$ is convex and its interior is the locus of points closer to z_i than to any other terminal. Hence,

$$V(z_i, Z) = \{q \in E^2 \mid |z_i q| \leq |z_j q|, \forall z_j \in Z \setminus z_i\}.$$

Let $P(z_i, Z)$ denote the boundary of $V(z_i, Z)$. The union of these boundaries for all terminals in Z forms the *Voronoi diagram* for Z, denoted by $VD(Z)$. Its edges are called *Voronoi edges*. Points where Voronoi edges meet are called *Voronoi points*.

The kth *order Voronoi diagram* $VD_k(Z)$, $1 \leq k < n$, is a partition of the plane into regions $V(Z_k, Z)$, $Z_k \subseteq Z$, $|Z_k| = k$. The interior of $V(Z_k, Z)$ is the locus of points closer to every terminal in Z_k than to any terminal in $Z \setminus Z_k$. Hence,

$$V(Z_k, Z) = \{q \in E^2 \mid |z_i q| \leq |z_j q|, \forall z_i \in Z_k, \forall z_j \in Z \setminus Z_k\}.$$

Hence $VD_1(Z) = VD(Z)$. Note that $V(Z_k, Z)$ may be empty. In fact, at most $O(n^3)$ regions of all orders k, $1 \leq k \leq n-1$, are nonempty; furthermore, the Voronoi diagrams of all orders up to Kth order can be determined in $O(K^2 n \log n)$ time [10].

2.2. *Delaunay Triangulations*. The straight-line dual of the Voronoi diagram for Z is a triangulation of Z, called the Delaunay triangulation and denoted by $DT(Z)$. This is one of the most important triangulations capturing proximity relations. It can also be defined as the unique triangulation such that the circumcircle of each triangle does not contain any other terminal in its interior. Triangles of $DT(Z)$ tend to be as "equilateral" as possible in the sense that the smallest internal angle in all its triangles is maximized over all triangulations.

An edge (z_i, z_j) belongs to $DT(Z)$ if and only if there is a circle passing through z_i and z_j and containing no other terminal in its interior (Figure 2(a)).

$DT(Z)$ has a number of interesting properties. It can be constructed in time $\Theta(n \log n)$ and contains at least one MST for Z. The MST for Z can be determined in time $\Theta(n)$ once $DT(Z)$ is given [14].

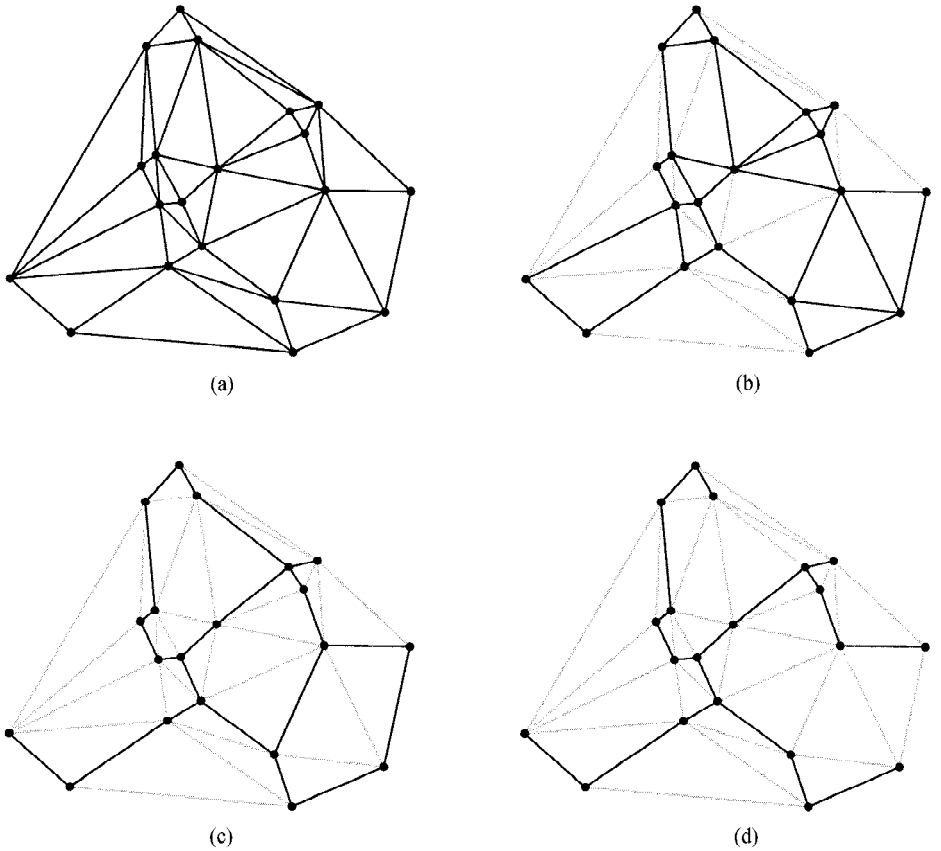

Fig. 2. Proximity structures: (a) Delaunay triangulation, (b) Gabriel graph, (c) relative neighborhood graph, and (d) minimum spanning tree.

2.3. *Gabriel Graphs.* Let z_i and z_j denote two distinct terminals. Let $D(z_i, z_j)$ denote a disk with $z_i z_j$ as its diameter. A *Gabriel graph* $GG(Z)$ has Z as its vertex set. A pair of terminals z_i and z_j is adjacent iff $D(z_i, z_j)$ contains no other terminal (Figure 2(b)). $GG(Z)$ can be constructed in $\Theta(n \log n)$ time by removing from $DT(Z)$ edges not intersecting their dual Voronoi edges [11]. Consequently, $GG(Z)$ is a subgraph of $DT(Z)$. In fact, it contains at least one MST for Z.

2.4. *Relative Neighborhood Graphs.* Let z_i and z_j denote two distinct terminals. Let $L(z_i, z_j)$ be a lune obtained as an intersection of two disks with radius $|z_i z_j|$ and centered at z_i and z_j, respectively. A relative neighborhood graph $RNG(Z)$ has Z as its vertex set. A pair of terminals z_i and z_j is adjacent iff $L(z_i, z_j)$ contains no other terminal [22] (Figure 2(c)). A $\Theta(n \log n)$ plane sweep algorithm for the construction of $RNG(Z)$ has been suggested by Supowit [19]. $RNG(Z)$ is a subgraph of $DT(Z)$. It contains at least one MST for Z.

3. Subsets of Terminals. The first phase of all our heuristics is to identify low-cardinality subsets of terminals $Z_k \subseteq Z$, $2 \leq |Z_k| \leq K$, such that the shortest FST for Z_k (if it exists) is a good candidate for a subtree of an SMT for Z. We assume that the maximum number of terminals of any FST to be generated is given as a fixed integer K, $2 \leq K \leq n$. Computational experience reported in Section 6 indicates that K should not be greater than 6 for randomly generated problem instances.

Subsets with two terminals are identified by taking all $n-1$ pairs of terminals joined by an edge in an MST for Z. The rationale behind this strategy is due to the fact that FSTs spanning two terminals in an SMT for Z must belong to an MST for Z.

HIGHER-ORDER VORONOI DIAGRAMS. A subset Z_k of Z, $3 \leq k \leq K$, is selected iff the Voronoi region $V(Z_k, Z)$ is nonempty. The family of these subsets (together with two-terminal subsets) is denoted by $HVD(K)$.

The total number of nonempty Voronoi regions in Voronoi diagrams of all orders up to Kth order is $O(K^2(n-K))$ [10].

DELAUNAY TRIANGULATIONS. A subset Z_k of Z, $3 \leq k \leq K$, is selected iff the subgraph of $DT(Z)$ induced by Z_k is the Delaunay triangulation of Z_k. The family of these subsets (together with two-terminal subsets) is denoted by $DT^{\triangle}(K)$.

A larger family of subsets, denoted by $DT^{\star}(K)$, is obtained by taking subsets Z_k of Z, $3 \leq k \leq K$, such that the subgraph of $DT(Z)$ induced by Z_k is connected.

GABRIEL GRAPHS. A subset Z_k of Z, $3 \leq k \leq K$, is selected iff the subgraph of $DT(Z)$ induced by Z_k is the Delaunay triangulation of Z_k and is connected in $GG(Z)$. The family of these subsets (together with two-terminal subsets) is denoted by $GG^{\triangle}(K)$.

A larger family of subsets, denoted by $GG^{\star}(K)$, is obtained by taking subsets Z_k of Z, $3 \leq k \leq K$, such that the subgraph of $GG(Z)$ induced by Z_k is connected.

RELATIVE NEIGHBORHOOD GRAPHS. A subset Z_k of Z, $3 \leq k \leq K$, is selected iff the subgraph of $DT(Z)$ induced by Z_k is the Delaunay triangulation of Z_k and is connected in $RNG(Z)$. The family of these subsets (together with two-terminal subsets) is denoted by $RNG^{\triangle}(K)$.

A larger family of subsets, denoted by $RNG^{\star}(K)$, is obtained by taking subsets Z_k of Z, $3 \leq k \leq K$, such that the subgraph of $RNG(Z)$ induced by Z_k is connected.

MINIMUM SPANNING TREES. A subset Z_k of Z, $3 \leq k \leq K$, is selected iff the subgraph of $DT(Z)$ induced by Z_k is the Delaunay triangulation of Z_k and it is connected in $MST(Z)$. The family of these subsets (together with two-terminal subsets) is denoted by $MST^{\triangle}(K)$.

A larger family of subsets, denoted by $MST^{\star}(K)$, is obtained by taking subsets Z_k of Z, $3 \leq k \leq K$, such that the subgraph of $MST(Z)$ induced by Z_k is connected.

It is well-known that

$$MST^{\triangle}(K) \subseteq RNG^{\triangle}(K) \subseteq GG^{\triangle}(K) \subseteq DT^{\triangle}(K)$$

and

$$MST^{*}(K) \subseteq RNG^{*}(K) \subseteq GG^{*}(K) \subseteq DT^{*}(K).$$

Furthermore,

$$MST^{\triangle}(K) \subseteq MST^{*}(K), \quad RNG^{\triangle}(K) \subseteq RNG^{*}(K),$$
$$GG^{\triangle}(K) \subseteq GG^{*}(K), \quad DT^{\triangle}(K) \subseteq DT^{*}(K).$$

The total number of terminal subsets in each of the sets $MST^{\triangle}(K)$, $RNG^{\triangle}(K)$, $GG^{\triangle}(K)$, and $DT^{\triangle}(K)$ is $O(3^{K-3}n)$ since there are $O(n)$ triangles and each triangle has at most three neighboring triangles. Assuming that K is a constant we get $O(n)$ terminal subsets in time $O(n \log n)$.

The *expected* number of terminal subsets in each of the sets $MST^{\star}(K)$, $RNG^{\star}(K)$, $GG^{\star}(K)$, and $DT^{\star}(K)$ is $O(6^{K-1}n)$ since each terminal in $DT(Z)$ has *average* degree six. Assuming that K is a constant the expected number of terminal subsets is $O(n)$ and they can be generated in expected time $O(n \log n)$.

4. Full Steiner Tree Generation. Since we only consider small subsets of terminals ($K \leq 6$), the processing time needed for each subset of terminals is bounded by a constant. We compute a shortest FST for a given subset of terminals, instead of an SMT, for the following reasons:

- It is much faster and less complicated to compute a shortest FST or to conclude that no FST exists.
- An FST does exist for a small fraction of subsets of terminals (especially when $K \geq 5$) while all subsets of terminals have an SMT.
- SMTs will often be obtained by concatenation of smaller FSTs.

It is well-known that an SMT for Z_k is completely inside the convex hull $CH(Z_k)$. Even a smaller region, called the *Steiner hull* of Z_k and denoted by $SH(Z_k)$, can be obtained in the following iterative way. Initially, $SH(Z_k)$ is identical with $CH(Z_k)$. If $SH(Z_k)$ contains a pair z_i, z_j of terminals appearing consecutively on the boundary, and a third terminal z_q, $z_q \neq z_i, z_j$, such that the interior angle $\angle z_i z_q z_j$ of $\triangle z_i z_q z_j$ is greater than or equal to 120° and $\triangle z_i z_q z_j$ contains no other terminals, then replacing $z_i z_j$ by $z_i z_q$ and $z_q z_j$ on the boundary yields a smaller region completely containing SMT for Z_k. This process is continued for as long as possible. It can be shown that the order in which the edges are replaced is immaterial [9]. If the interior of $SH(Z_k)$ is not connected, Z_k is said to have *degenerate configuration*; the original problem instance can be decomposed into smaller problem instances (Figure 3(a)). If the interior of $SH(Z_k)$ is connected and all terminals are on its boundary, Z_k is said to have *Steiner configuration* (Figure 3(b)).

Properties of FSTs with three or four terminals are well understood [9]. In particular, a necessary condition for the existence of such FSTs is that all terminals are on the boundary of their convex hull. We say that such sets of terminals have a *convex configuration*.

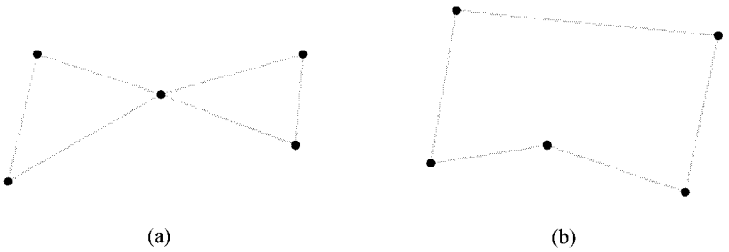

Fig. 3. Terminal configuration examples: (a) degenerate and (b) Steiner.

A convex configuration is not necessary for the existence of FSTs with five or more terminals.

Experiments show that we may discard terminal sets which do not form a Steiner configuration since they are very unlikely to be spanned by a single FST in an SMT for Z [24]. In a set of one hundred problem instances (each with one hundred terminals), all sets with five terminals and all but *one* set with six terminals had Steiner configurations. There were two sets of seven terminals which did not have a Steiner configuration, but since we restrict our attention to sets with up to six terminals, these sets would not be considered anyway.

The advantage of considering Steiner configurations, in addition to discarding less promising FSTs, is that it is much faster to compute shortest FSTs for sets with five and six terminals. An FST for k terminals is computed by generating all possible full topologies for k terminals and determining the FST (if it exists) for each topology in $O(k)$ time using Hwang's algorithm [8]. There are 15 (resp. 120) different topologies for $k = 5$ (resp. $k = 6$). If the configuration is Steiner, only 5 (resp. 14) different topologies need to be considered [9].

For each terminal set Z_k the following computations/tests are made:

1. Compute the Steiner hull for Z_k. If the configuration of Z_k is degenerate or not Steiner, then discard Z_k.
2. Find a shortest FST for Z_k, denoted by $FST(Z_k)$, by generating all admissible full topologies and applying Hwang's algorithm. If no FST exists, then discard Z_k.
3. Compute $MST(Z_k)$. If $|MST(Z_k)| \leq |FST(Z_k)|$, then discard Z_k.

5. Greedy Concatenation. Given a list $\mathcal{F}$ of FSTs, we would like to construct a short tree spanning Z using these FSTs. For any FST $F \in \mathcal{F}$ we denote by Z_F the set of terminals spanned by F and by $\mathcal{M}_{Z_F}$ the subgraph of $MST(Z)$ induced by Z_F. We assume in our complexity analysis that $\mathcal{F}$ contains $O(n)$ FSTs. Relatively limited knowledge is required about these FSTs—a fact that makes the approach easy to generalize to other metrics (including obstacle-avoiding variants).For each FST $F \in \mathcal{F}$ we only assume that the following information is available:

- Terminal set Z_F spanned by F.
- Length $|F|$ of F.

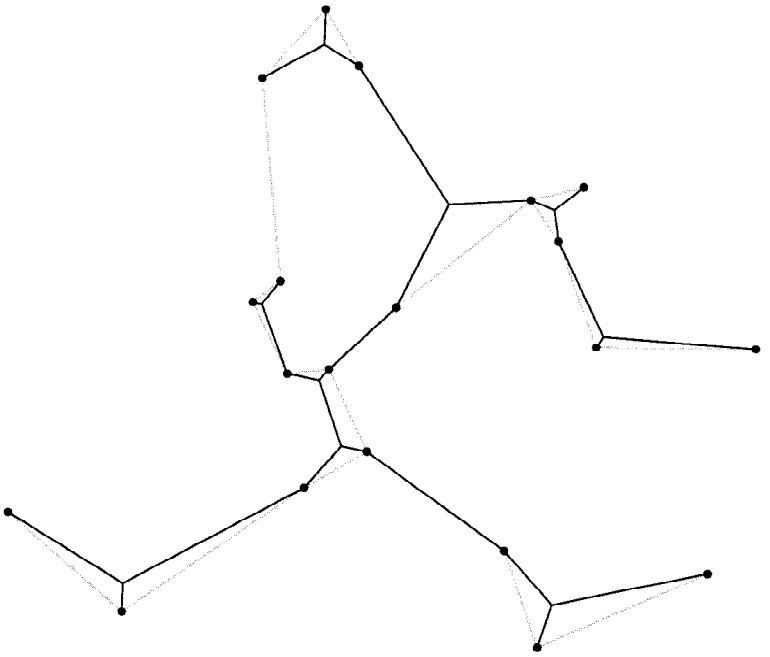

Fig. 4. SMT with an FST inducing disconnected subgraphs of the MST (gray edges).

- Length $|MST(Z_F)|$ of an MST spanning Z_F.
- Information about the connectedness of $\mathcal{M}_{Z_F}$, i.e., whether Z_F induces a connected subgraph of $MST(Z)$ or not.

No knowledge about the location of Steiner points nor about the connections between terminals and Steiner points is needed during the concatenation. However, in order to output the final tree, it is obviously necessary that we have access to this information.

Experimental evidence, presented in Section 6, shows that terminals of *most* FSTs in an SMT induce connected subgraphs of $MST(Z)$. One must therefore use MSTs as backbones when constructing good approximations to SMTs. However, in order to obtain high-quality solutions, it is sometimes necessary to deviate from the MST (Figure 4), as previously pointed out by Chang [6]. We therefore split our greedy construction into two phases:

1. **Greedy concatenation (Kruskal):** Sort the FSTs according to a *priority measure* and construct an initial solution using a variant of Kruskal's MST algorithm, as explained in Section 5.1. Only FSTs with terminals inducing connected subgraphs of $MST(Z)$ are allowed to be used in this phase.
2. **Greedy insertion:** Try to improve the initial solution by inserting the remaining FSTs, as explained in Section 5.2.

We have tried to avoid the greedy insertion altogether to simplify the procedure. However, it seems very difficult to define a priority measure which both gives special priority to connected induced subgraphs of the MST *and* some preference to other promising FSTs.

5.1. *Constructing the Initial Tree Using Kruskal.* This initial phase is basically the heuristic of Smith et al. [16]. The FSTs are sorted according to the ratio priority $|F|/|MST(Z_F)|$ (smallest first), such that FSTs showing a large relative reduction over the MST have top priority. Another possibility is to use the difference priority $|MST(Z_F)| - |F|$ (largest first). Our experiments show that the ratio priority is to be preferred as the difference priority gives preference to large (in terms of length) FSTs; there is no guarantee that such FSTs are SMTs for their terminals. The ratio priority is more size independent, although with some preference for smaller FSTs. Also, if a small "bad" FST is added in this initial phase, it is likely to be replaced by a larger FST during the greedy FST insertion phase (Section 5.2).

Thus, we sort the FSTs by ratio and assume in the following that the FSTs are indexed according to this priority: $\mathcal{F} = \{F_1, F_2, \ldots, F_m\}$, where $m = O(n)$ is the number of FSTs. We also assume that the MST edges (two-terminal FSTs), denoted by $\mathcal{M}$ and sorted in nondecreasing order, form the tail of $\mathcal{F}$. The initial tree T_0 is constructed by using the following algorithm (for an example see Figure 5):

function *Kruskal*($\mathcal{F}$)
  $T_0 = \emptyset$
  forall $F \in \mathcal{F}$ **do**
    if ($\mathcal{M}_{Z_F}$ is connected) **and** (F does not create a cycle in T_0) **then** $T_0 = T_0 \cup F$
  return T_0
end

That is, we scan $\mathcal{F}$ and add FSTs to the tree (forest) if no cycle is created. Since the MST edges form the tail of $\mathcal{F}$ we always obtain a valid tree, i.e., T_0 is connected and acyclic.

The initial tree T_0 will never be longer than $MST(Z)$ since only FSTs which induce connected subgraphs of $MST(Z)$ are added. Furthermore, since we only improve T_0 in the second phase, the final tree will necessarily also have this property. An $MST(Z)$ is known to have degree at most six for every terminal [13] and the same will hold for T_0. This can be seen from the fact that every FST in T_0 spanning k terminals replaces exactly $k - 1$ edges in $MST(Z)$; all terminals in an FST are leaves and thus the degree of every terminal in T_0 is at most its degree in $MST(Z)$.

5.2. *Greedy FST Insertion.* In this section we present an approach which has some similarity to Chang's generalized Steiner point insertions [6]. However, we restrict our attention to the sorted list $\mathcal{F}$ of FSTs and define an *FST insertion* as follows: Let T be the current tree, initially $T = T_0$. Assume that the FSTs in T appear in the same order

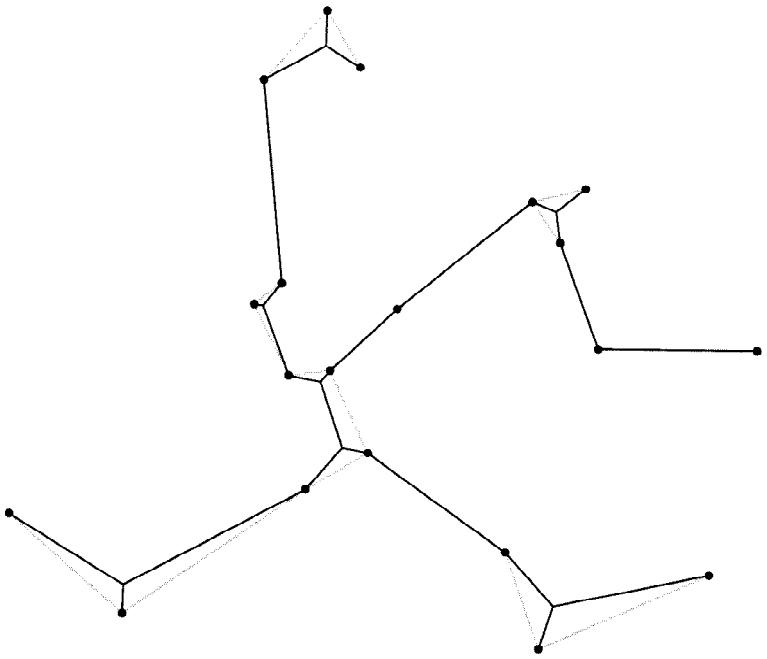

Fig. 5. Initial tree using Kruskal. Observe that each FST induces a connected subgraph of the MST (gray edges).

as in $\mathcal{F}$. An FST $F_i \in \mathcal{F} \backslash T$ is inserted into T by using the following algorithm:

function *Insert*(T, F_i, $\mathcal{M}$)
  $T' = F_i$
  forall $F \in T$ **do**
    if ($|Z_F| \geq 3$) **and** (F does not create a cycle in T') **then** $T' = T' \cup F$
  forall $F \in \mathcal{M}$ **do**
    if (F does not create a cycle in T') **then** $T' = T' \cup F$
  return T'
end

Thus, we first add F_i to an empty tree, then all FSTs in T with three or more terminals (avoiding cycles), and finally MST edges in order to guarantee connectivity. An FST-insertion requires $O(n)$ time.[3]

The actual behavior of the algorithm is that it inserts F_i into T by pushing some of the FSTs in T out and reconnecting the components by adding edges from $MST(Z)$.

[3] Using a fast disjoint set data structure [20], the amortized time per FST is actually $O(\alpha(m, n))$, where $\alpha(m, n)$ is the inverse of Ackermann's function. This is an extremely slow-growing function and for all practical purposes a constant.

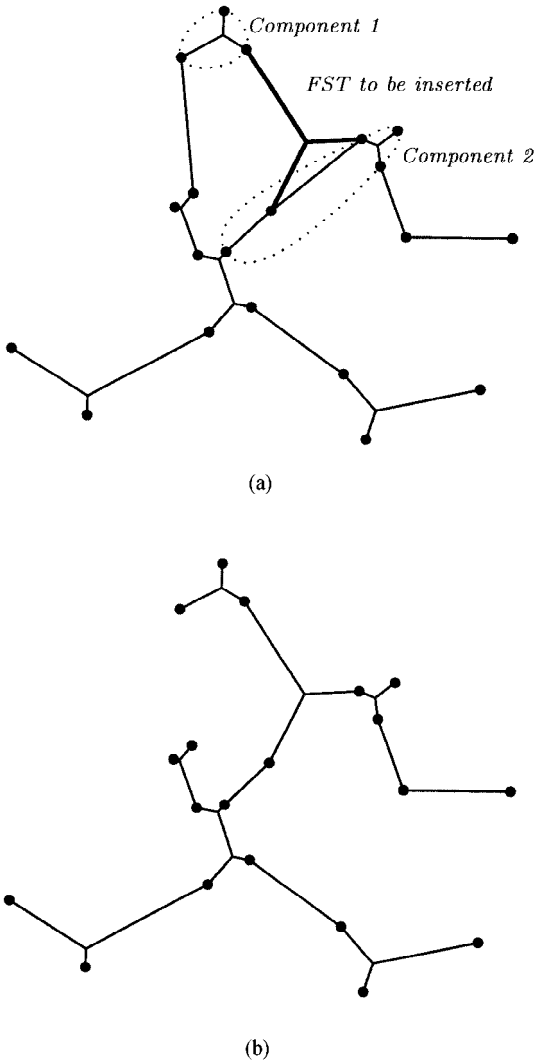

Fig. 6. FST-insertion: tree (a) before and (b) after insertion.

More precisely, if an FST F_i with k terminals is inserted into T, $k-1$ cycles are created (Figure 6(a)). Each such cycle visits one or more FSTs from T; one single FST in T may, if it spans three or more terminals, be a part of more than one cycle. For each cycle, only the FST which has the highest ratio is a candidate for deletion. Thus there is a natural preference for keeping low-ratio FSTs, in particular FSTs spanning three or more terminals (Figure 6(b)).

If the resulting tree T' is shorter than T we set $T = T'$ and try to insert the next FST from $\mathcal{F} \backslash T$. Scanning through $\mathcal{F}$ once yields an $O(n^2)$ greedy improvement algorithm. Preliminary experiments, presented in Section 6, show that this method is very effective

but at the cost of high running times. We will now show how to cut the running time down to $O(n \log n)$ while preserving most of the power of the $O(n^2)$ heuristic.

The obvious solution is to allow at most $O(\log n)$ FST insertions: pick the first $O(\log n)$ FSTs from $\mathcal{F}$ and perform the insert operation iteratively for each of these FSTs. Unfortunately, the following arguments show that the total expected relative improvement will diminish when n becomes large.

Assuming that the terminals are randomly distributed in a unit square, the expected length of an SMT is $\Theta(\sqrt{n})$. This follows directly from the constant factor relationship between the length of an $SMT(Z)$, an $MST(Z)$, and a Traveling Salesman tour $TSP(Z)$ through the same set of points Z. More specifically we have $\frac{1}{2}|TSP(Z)| \leq |MST(Z)| \leq |TSP(Z)|$ (the first inequality is the performance bound on the so-called double MST heuristic for TSP and the other inequality is obvious). Similarly we have $(\sqrt{3}/2)|MST(Z)| \leq |SMT(Z)| \leq |MST(Z)|$ (the first inequality is the Steiner ratio theorem and the second is again obvious). These inequalities yield $(\sqrt{3}/4)|TSP(Z)| \leq |SMT(Z)| \leq |TSP(Z)|$ and by using the classic result of Beardwood et al. [2] which states that the expected length of a TSP tour through a set of points distributed randomly in a unit square is $\Theta(\sqrt{n})$, the same results follows for an SMT.

When an FST F_i spanning k terminals is inserted, at most $k-1$ FSTs spanning three or more terminals are deleted from T. In addition, only a constant number of *new* MST edges is added to T since only MST edges which span terminals in removed FSTs are new candidates for being used to reconnect the tree. Thus, the total number of FSTs (including MST edges) deleted or added is bounded by a constant. Each FST has bounded length and therefore the length reduction is bounded by a constant independent of n. The total length reduction obtained by performing $O(\log n)$ FST insertions thus has the same order of magnitude. Now, this implies that the expected relative improvement over the initial solution drops to zero as n goes to infinity. This indicates that we must perform $\Omega(\sqrt{n})$ insertions to ensure that the effect of the greedy improvement does not disappear for large values of n.

One remedy to this problem is to insert F_i *locally* in constant time. For any tree T, define T_z, $z \in Z$, to be the set of FSTs in T which span z. Assume that T has at most six FSTs spanning each terminal; in particular, this holds for T_0 (see Section 5.1).[4] Consider the forest $\bar{T} = \bigcup_{z \in Z_{F_i}} T_z$, i.e., FSTs in T which share a terminal with F_i. Obviously, this forest contains at most $6K$ FSTs. Let $\bar{Z} = \bigcup_{F \in \bar{T}} Z_F$ be the terminals spanned by $\bar{T}$ and let $\mathcal{M}_{\bar{Z}}$ be the subgraph of $MST(Z)$ induced by $\bar{Z}$.

The number of components in the forest $\bar{T}$ is bounded by the number of terminals in F_i (there are two components in Figure 6(a)). If there is only one component, then *all* FSTs through cycles created by inserting F_i into T are in $\bar{T}$. We replace the subtree $\bar{T}$ by $\bar{T}' = Insert(\bar{T}, F_i, \mathcal{M}_{\bar{Z}})$ provided that $\bar{T}'$ spans $\bar{Z}$ and is connected. Note that $\bar{T}'$ is not necessarily connected since we use edges from $\mathcal{M}_{\bar{Z}}$ to reconnect the components, not edges from $MST(\bar{Z})$.

By representing T and $MST(Z)$ appropriately, a local insertion can be performed in constant time. Although not very likely to happen we must also ensure that the size of T_z

[4] In an SMT there can be at most *three* such FSTs and, furthermore, for randomly generated instances the probability that there are *exactly* three FSTs is zero.

never becomes greater than six. This can be checked just before T is updated, discarding the insertion if the bound is exceeded.

If $\bar{T}$ or $\bar{T}'$ are disconnected the local insertion of F_i is discarded. However, it may still be possible to perform an $O(n)$-insertion. The overall greedy improvement algorithm is the following: Set $T = T_0$. For each FST $F_i \in \mathcal{F} \backslash T$ we first try a local insertion. If $\bar{T}$ and $\bar{T}'$ are connected and $|\bar{T}'| < |\bar{T}|$ we update T and go to the next FST. Otherwise we make the call *Insert*$(T, F_i, \mathcal{M})$—provided that we have made less than $C\lceil \log n \rceil$ $O(m)$-insertions already. Here C is a constant determined as follows. Preliminary experiments showed that if the number of components in $\bar{T}$ was small, it was more likely that an insertion was improving. We therefore make $C_i \lceil \log n \rceil$ insertions for every number i of components in $\bar{T}$ $(i = 1, \ldots, K)$; setting $C_i = K - i + 1$ proved to give the right balance between running time and solution quality.[5] Still, it should be noted that the main reduction in length comes from the local insertions since these may be performed for every FST; the other $O(\log n)$ insertions only have a small, although not negligible, effect.

6. Computational Experience. The new class of greedy concatenation-based heuristics was experimentally evaluated on an HP9000 workstation[6] using the programming language C++ and class library LEDA (version 3.4.1) [12]. The random number generator used was the `random_source` class in LEDA. We also used LEDA's native Delaunay triangulation algorithm. The higher-order Voronoi diagram implementation was based on Lee's algorithm [10] using the first-order Voronoi diagram algorithm in LEDA.

In Section 6.1 we compare terminal subset generation methods discussed in Section 3. The most promising of these are selected and we compare the new heuristics with our own implementation of the heuristic by Smith et al. [16] by performing experiments on a large number of randomly generated instances with up to 10,000 terminals (Section 6.2). Finally, the new heuristics are compared with other heuristics from the literature by measuring their performance on a series of library problem instances (Section 6.3).

6.1. *Terminal Subset Generation Methods.* The terminal subset generation methods described in Section 3 compared experimentally 100 problem instances (each with 100 terminals). The terminals were drawn randomly with uniform distribution from a unit square.

The average number of subsets generated with cardinality $k = 3, 4, 5, 6$ is given in Table 1. As could be expected, fewer subsets are generated when the connected induced subgraphs are restricted to adjacent triangles of DT. A very large number of subsets is generated for $DT^\star$ and $GG^\star$. Also, a relatively large number of three and four terminal subsets is generated by *HVD*, but this is less critical since FST computations are much more expensive for five and six terminal subsets.

Table 1 also presents the corresponding counts of surviving FSTs. We immediately note that a much larger fraction of terminal subsets survives for the triangulation-based

[5] Example: for $K = 5$ and $n = 1000$ at most $(5 + 4 + 3 + 2 + 1)\lceil \log 1000 \rceil = 105$ insertions are made.

[6] 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).

Table 1. Terminal subset generation methods.*

	$k=3$			$k=4$			$k=5$			$k=6$		
Method	TS	FST	NI	TS	FST	NI	TS	FST	NI	TS	FST	NI
HVD	459	253	0.01	623	161	0.14	777	88	0.08	924	40	0.05
$DT^{\triangle}$	186	151	0.54	273	116	0.34	530	98	0.08	1,150	80	0.03
$GG^{\triangle}$	134	116	0.55	193	93	0.34	334	73	0.09	636	56	0.03
$RNG^{\triangle}$	78	65	1.01	98	40	0.53	134	24	0.36	186	14	0.08
$MST^{\triangle}$	57	45	1.80	61	20	1.23	67	8	0.61	73	3	0.12
$DT^{\star}$	1,052	507	0.00	4,316	794	0.00	18,717	1389	0.00	83,930	2,375	0.01
$GG^{\star}$	431	220	0.01	1,188	253	0.00	3,507	326	0.00	10,796	403	0.01
$RNG^{\star}$	184	90	0.48	327	62	0.19	623	50	0.08	1,241	39	0.01
$MST^{\star}$	120	54	1.30	159	24	0.89	219	10	0.33	310	5	0.05

*For each cardinality k the table gives the average number of terminal subsets (TS), average number of surviving full Steiner trees (FST) and average number of not identified FSTs in SMTs (NI). Terminal subsets are generated as explained in Section 3 using higher-order Voronoi diagrams (HVD), Delaunay triangulations (DT), Gabriel graphs (GG), relative neighborhood graphs (RNG), and minimum spanning trees (MST).

methods. For $DT^{\triangle}$ a total (for $3 \leq k \leq 6$) of 21% survive, compared with only 5% for $DT^{\star}$. The same numbers for $GG^{\triangle}$ and $GG^{\star}$ are 26% and 8%, respectively. For *HVD* the number is 19%, slightly lower than for $DT^{\triangle}$.

The ratio of surviving terminal subsets is not the only measure of quality. More important is the issue of how well FSTs in SMTs are represented. Since SMTs are known for all instances in the testbed [24], we may count the number of FSTs in each SMT which have *not* been identified. These average counts are also given in Table 1 and are in general quite low. They should be compared with an average of 29.2 two-terminal FSTs (MST-edges), 19.9 three-terminal FSTs, 7.5 four-terminal FSTs, 1.6 five-terminal FSTs, and 0.2 FSTs with six or more terminals in the same 100 problem instances. Another interesting observation is that 10 of these SMTs had a maximum FST size (number of terminals) of four, 65 a maximum of five, 20 a maximum of six, and only 5 had an FST with seven or more terminals.

There is a (natural) correspondence between the total number of FSTs generated and counts of not identified FSTs. The methods *HVD*, $DT^{\star}$, and $GG^{\star}$ which generate a large number of subsets (the last two in particular) also have a higher probability of "covering" the SMT. This does not mean that $DT^{\triangle}$ and $GG^{\triangle}$ are poor at identifying FSTs in SMTs—on average only one FST is missed. Another interesting observation is that $RNG^{\triangle}$ and especially $MST^{\triangle}$ are significantly worse at identifying FSTs in SMTs. Finally, on the basis of the statistics for $MST^{\star}$ we can conclude that an average of 2.6 FSTs do not induce connected subgraphs of the MST. This should be compared with an average total of 58.4 FSTs in the corresponding SMTs.

The performance of a heuristic using a given subset generation method depends on the concatenation method used. First we compare all generation methods using the $O(n \log n)$ greedy concatenation method described in Section 5.2. In Table 2 the reduction over MST is given for each generation method and $K = 3, 4, 5, 6$ (maximum subset cardinality). The table clearly shows that the ability to cover FSTs in SMTs is not the only factor that determines the performance of greedy concatenation heuristics.

Table 2. Heuristic performance for terminal subset generation methods.

	$K = 3$		$K = 4$		$K = 5$		$K = 6$	
Method	RED*	CPU*	RED	CPU	RED	CPU	RED	CPU
HVD	2.79	2.13	3.06	4.13	3.09	7.66	3.09	14.57
$DT^{\triangle}$	2.77	0.17	3.06	0.38	3.09	1.66	3.10	7.33
$GG^{\triangle}$	2.78	0.17	3.07	0.33	3.09	1.25	3.10	4.67
$RNG^{\triangle}$	2.69	0.19	3.05	0.27	3.09	0.66	3.09	1.62
$MST^{\triangle}$	2.56	0.09	2.92	0.14	2.97	0.36	2.97	0.71
$DT^{\star}$	2.78	0.41	3.03	2.22	3.05	28.15	3.06	346.07
$GG^{\star}$	2.80	0.25	3.05	0.79	3.08	6.22	3.09	50.26
$RNG^{\star}$	2.74	0.21	3.06	0.36	3.09	1.38	3.09	6.26
$MST^{\star}$	2.63	0.11	2.95	0.18	2.99	0.54	3.00	1.59

*RED: reduction over MST (%). CPU: total CPU time (sec.).

Triangulation-based methods, in particular $DT^{\triangle}$ and $GG^{\triangle}$, perform *better* than their subgraph connectivity-based counterparts. This may seem surprising but is just an indicator that the greedy concatenation has had less bad choices. The CPU times are more or less directly proportional to the number of subsets generated—perhaps except for *HVD* which has a relatively high overhead due to the complex algorithm for constructing higher-order Voronoi diagrams. Increasing K from 4 to 5 increases the running time by a factor between 2 (for *HVD*) and 13 (for $DT^{\star}$). Also note that selecting $K = 3$ is a very bad alternative. The improvement in solution quality when going from $K = 5$ to $K = 6$ on the other hand is negligible.

In the following we identify heuristics using the $O(n \log n)$ concatenation method by its generation method and maximum terminal subset cardinality K. For instance, $GG^{\triangle}(5)$ uses terminal subsets with up to five terminals identified as vertices of adjacent triangles of the *DT* forming connected induced subgraphs of the Gabriel graph.

On the basis of the results presented in Table 2, we selected $GG^{\triangle}(4)$ and $GG^{\triangle}(5)$ as the most "promising" alternatives. This is motivated by the excellent performance of these two heuristics and by their limited use of CPU time. $RNG^{\triangle}(5)$ is also very efficient, but requires a special nonstandard algorithm if one demands $O(n \log n)$ running time [19].

Finally we present some results on other alternatives for greedy concatenation. Table 3 compares four variants using generation method $GG^{\triangle}$ for $K = 4, 5$ and $n = 100$. The first variant returns the initial solution obtained by using Kruskal without making any

Table 3. Greedy concatenation method comparison for $GG^{\triangle}$.

	$K = 4$		$K = 5$	
Greedy concatenation method	RED*	CPU*	RED	CPU
(1) Kruskal	2.83	0.19	2.86	1.05
(2) Kruskal + local insertions	2.98	0.27	3.00	1.17
(3) Kruskal + local insertions + $O(\log n)$ insertions	3.07	0.33	3.09	1.25
(4) Kruskal + $O(n)$ insertions	3.11	0.54	3.13	1.53

*RED: reduction over MST (%). CPU: total CPU time (sec.).

FST insertions (Section 5.1). The second only makes local FST insertions while the third makes local FST insertions and $O(\log n)$ FST insertions (this is the variant evaluated in Table 2). The last variant makes $O(n)$ FST insertions and therefore takes $O(n^2)$ time as opposed to the other three variants which require $O(n \log n)$ time.

FST insertions significantly improve the quality of the heuristic solution at relatively limited cost. The performance of the third variant is not much worse than the fourth variant. However, this difference increases for larger values of n as will be shown in Section 6.3; the higher running time complexity of the latter also becomes much more evident.

6.2. *Comparison to Smith et al.'s Heuristic.* In this section we compare $GG^{\triangle}(4)$ and $GG^{\triangle}(5)$ to our own implementation of the heuristic by Smith et al. [16]. In this original version, called *SLL*, triangles in the DT with two MST edges for which the corresponding FST exists are put on a priority queue (similar to $MST^{\triangle}(3)$). The Kruskal-based concatenation first tries to add a four-terminal FST for the triangle in question and its nearest adjacent triangle[7]—if this FST does not exist or if it does create a cycle in the current tree, the three-terminal FST is added.

A very simple modification of this heuristic, called SLL^{+}, simply puts all triangles with two MST edges for which an FST exists and all four-terminal FSTs constructed from adjacent triangles with three connected MST edges on the priority queue (just like $MST^{\triangle}(4)$). The heuristic tree is constructed using Kruskal.

We present computational results in Table 4. Each number is an average taken over 100 instances. SLL^{+} outperforms *SLL* at very little extra computational cost (for very small instances the opposite seems to be the case as shown in Section 6.3). $GG^{\triangle}(4)$ and $GG^{\triangle}(5)$ obtain reductions which are more than 0.3% better than *SLL*–at a *constant* factor of 6 and 22 times the running time of *SLL*. For $GG^{\triangle}(4)$ approximately one-third of this extra time is spent generating FSTs and two-thirds performing greedy improvement. For

Table 4. Randomly generated instances.

	SLL		SLL^{+}		$GG^{\triangle}(4)$		$GG^{\triangle}(5)$	
n	RED*	CPU*	RED	CPU	RED	CPU	RED	CPU
50	2.76	0.06	2.89	0.06	3.11	0.15	3.13	0.56
100	2.71	0.11	2.83	0.13	3.07	0.33	3.09	1.25
500	2.68	0.31	2.84	0.37	3.04	2.05	3.07	7.60
1,000	2.70	0.68	2.83	0.81	3.02	4.37	3.05	16.04
5,000	2.72	3.96	2.85	4.61	3.01	25.67	3.04	89.00
10,000	2.71	8.44	2.85	9.75	3.00	54.69	3.02	186.40
Avg.	2.71		2.85		3.04		3.07	

*RED: reduction over MST (%). CPU: total CPU time (sec.).

[7] Based on the distance between corresponding Voronoi vertices.

$GG^{\triangle}(5)$ we have just the opposite: two-thirds are used by FST generation and one-third by greedy improvement.

This may at first seem to be expensive, but the new heuristics still are $O(n \log n)$ and we actually are very close to optimum: For $n = 50$, the average SMT reduction is 3.23%; out of 100 instances $GG^{\triangle}(4)$ found the optimal solution for 10 instances and $GG^{\triangle}(5)$ the optimal solution for 15 instances. For $n = 100$ the average SMT reduction is 3.20%; no optimal solutions have been found by either heuristic. On average we are therefore within 0.1% from optimum for small instances ($n \leq 100$) and (most likely) within 0.2% from optimum for larger instances.

6.3. *Comparison to Other Heuristics*. First we make a detailed comparison to the heuristics by Beasley [4] (*BE*92) and Beasley and Goffinet [5] (*BG*94). The CPU times in these two papers have been "normalized" using the *Linpack* benchmark.[8]

The heuristics were evaluated on instances which are available from the *OR-Library* [3]. These instance are randomly generated problem instances with 10–1000 terminals, 15 instances for each size; optimal solutions are known for all these instances [24], [23]. In addition, a single 10,000 terminals instance is available from the *OR-Library*.

We ran SLL^{+}, $GG^{\triangle}(4)$ and $GG^{\triangle}(4)^{+}$ (the $O(n^2)$ variant of $GG^{\triangle}(4)$ which makes $O(n)$ FST insertions) on the same set of problem instances. In Table 5 we compare the results with *BE*92. The overall tendency as far as solution quality is concerned is clear: SLL^{+} falls behind by a large margin, while $GG^{\triangle}(4)$ and $GG^{\triangle}(4)^{+}$ both are better than *BE*92. While the observed running time growth of *BE*92 is $O(n^{1.317})$ the heuristic $GG^{\triangle}(4)$ has a worst-case running time of $O(n \log n)$ with a relatively small constant. For $n = 1000$ the running time of $GG^{\triangle}(4)$ is less that one-seventh of *BE*92 and the heuristic solutions produced are also better.

It would have been interesting to make a thorough comparison between our new heuristics and *BE*92 on larger instances. Beasley [4] reports a 3.00% reduction in (normalized) time 4093.38 seconds on one 10,000 terminals instance. When we applied SLL^{+}, $GG^{\triangle}(4)$, and $GG^{\triangle}(4)^{+}$ to the same instance we obtained reductions of 2.85%, 2.98%, and 3.16%, respectively. The corresponding CPU times were 9.89, 54.22, and 5533.93 seconds. Thus the new heuristics compare very favorably when running times are taken into account.

While the variance of the MST reduction is similar for *BE*92, $GG^{\triangle}(4)$, and $GG^{\triangle}(4)^{+}$, the CPU-time variance shows a completely different picture. The iterative nature of *BE*92 makes the running time less predictable, e.g., for $n = 1000$, the ratio between the maximum and minimum running time is 3.00, while it is only 1.05 and 1.11 for $GG^{\triangle}(4)$ and $GG^{\triangle}(4)^{+}$, respectively. Also, while $GG^{\triangle}(4)$ uses 1 minute on an average 10,000 terminal instance, *BE*92 spends more than an hour on a similar instance (based on the result for the single 10,000 terminal instance discussed above; if this instance is representative the reported running time growth of $O(n^{1.317})$ actually seems to be closer to being quadratic for larger instances).

[8] Our HP workstation has a Linpack benchmark of approximately 40, the Cray X-MP/28 used in [4] a value between 50 and 200 and the SGI Indigo machine used in [5] a value between 4 and 12. Accordingly, the CPU times in these two papers have been multiplied by 1.5 and 0.2, respectively, in order to make them comparable with ours.

Table 5. Comparison on instances from the *OR-Library*.

	*BE*92		*SLL*$^+$		*GG*$^\triangle$(4)		*GG*$^\triangle$(4)$^+$		*OPT*
n	RED*	CPU*	RED	CPU	RED	CPU	RED	CPU	RED
10	3.14 ± 1.86	0.07	2.91 ± 1.82	0.01	3.17 ± 1.91	0.02	3.17 ± 1.91	0.02	3.25 ± 1.88
20	3.02 ± 1.01	0.17	2.91 ± 1.04	0.01	3.10 ± 1.00	0.04	3.10 ± 0.97	0.04	3.16 ± 0.99
30	2.87 ± 0.72	0.26	2.73 ± 0.72	0.02	2.94 ± 0.78	0.08	2.96 ± 0.77	0.08	3.07 ± 0.78
40	3.02 ± 0.63	0.50	2.87 ± 0.54	0.03	3.03 ± 0.63	0.11	3.04 ± 0.63	0.11	3.14 ± 0.63
50	2.84 ± 0.40	0.49	2.72 ± 0.39	0.04	2.93 ± 0.36	0.14	2.93 ± 0.36	0.17	3.03 ± 0.41
60	2.95 ± 0.40	0.72	2.75 ± 0.37	0.04	3.08 ± 0.46	0.19	3.10 ± 0.43	0.22	3.27 ± 0.42
70	2.84 ± 0.36	0.72	2.65 ± 0.33	0.05	2.92 ± 0.36	0.21	2.97 ± 0.33	0.29	3.11 ± 0.38
80	2.82 ± 0.62	1.00	2.64 ± 0.61	0.06	2.87 ± 0.65	0.25	2.92 ± 0.65	0.36	3.04 ± 0.67
90	2.94 ± 0.45	1.22	2.85 ± 0.50	0.07	2.96 ± 0.49	0.29	3.01 ± 0.51	0.45	3.12 ± 0.49
100	2.95 ± 0.37	1.47	2.80 ± 0.34	0.08	3.08 ± 0.43	0.34	3.14 ± 0.41	0.56	3.27 ± 0.38
250	2.95 ± 0.21	4.32	2.79 ± 0.23	0.17	3.00 ± 0.22	0.92	3.07 ± 0.24	2.91	3.21 ± 0.23
500	3.05 ± 0.17	10.28	2.89 ± 0.17	0.37	3.13 ± 0.19	2.03	3.22 ± 0.17	11.52	3.33 ± 0.18
1000	3.02 ± 0.13	31.76	2.87 ± 0.12	0.80	3.05 ± 0.12	4.32	3.18 ± 0.14	48.84	3.31 ± 0.14
Avg.	2.95 ± 0.72		2.80 ± 0.70		3.02 ± 0.74		3.06 ± 0.73		3.18 ± 0.73

*RED: reduction over MST (%) and standard deviation. CPU: total CPU time (sec.). OPT: optimal solution reduction.

The heuristic *BG*94 [5] has a performance that is somewhere between *GG*$^\triangle$(4) and *GG*$^\triangle$(4)$^+$ but at the cost of a huge running time. Results are only reported for $n \leq 100$ and for these instances the average reduction is 3.03%. For *GG*$^\triangle$(4) and *GG*$^\triangle$(4)$^+$ the reductions obtained on the same instances are 3.01% and 3.03%, respectively. However, the (normalized) running time for *BG*94 ($n = 100$) is more than 100 times larger than for both *GG*$^\triangle$(4) and *GG*$^\triangle$(4)$^+$. Also, the observed running time growth is higher, namely $O(n^{2.19})$.

Finally, Chapeau-Blondeau et al. [7] recently suggested a new $O(n \log n)$ heuristic. The (normalized) running times reported are slightly higher than those for *SLL* and *SLL*$^+$ and the average reduction for $n = 1000$ is only 2.78%. Thus this heuristic does not perform better than *SLL*$^+$.

7. Concluding Remarks. We presented a class of $O(n \log n)$ heuristics for the Steiner tree problem in the Euclidean plane. The new heuristics first generated a short list of FSTs constructed on small subsets of terminals. The geometrically close terminals spanned by each FST were identified by using well-known structures from computational geometry.

Heuristic trees were constructing by greedy concatenation. Extensive experiments showed that the new heuristics performed better than any other known $O(n \log n)$ heuristic. In fact, the heuristic solutions were better than those obtained by most other heuristics with higher (or unknown) complexities.

The approach can be easily generalized to other metrics and higher dimensions, including obstacle-avoiding variants. A local search approach using full Steiner tree concatenation has recently been suggested by Zachariasen [25]. Solutions within 0.05% from optimum could be obtained by using the same basic FST-insertion scheme.

References

[1] S. Arora. Polynomial Time Approximation Schemes for Euclidean TSP and Other Geometric Problems. In *Proc. 37th Ann. Symp. on Foundations of Computer Science*, pages 2–13, 1996.

[2] J. Beardwoord, J. H. Halton, and J. M. Hammersley. The Shortest Path Through Many Points. *Proceedings of the Cambridge Philosophical Society*, 55:299–327, 1959.

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

[4] J. E. Beasley. A Heuristic for Euclidean and Rectilinear Steiner Problems. *European Journal of Operational Research*, 58:284–292, 1992.

[5] J. E. Beasley and F. Goffinet. A Delaunay Triangulation-Based Heuristic for the Euclidean Steiner Problem. *Networks*, 24:215–224, 1994.

[6] S. K. Chang. The Generation of Minimal Trees with a Steiner Topology. *Journal of the Association for Computing Machinery*, 19:699–711, 1972.

[7] F. Chapeau-Blondeau, F. Janez, and J.-L. Ferrier. A Dynamic Adaptive Relaxation Scheme Applied to the Euclidean Steiner Minimal Tree Problem. *SIAM Journal on Optimization*, 7(4):1037–1053, 1997.

[8] F. K. Hwang. A Linear Time Algorithm for Full Steiner Trees. *Operations Research Letters*, 4(5):235–237, 1986.

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

[10] D. T. Lee. On *k*-Nearest Neighbour Voronoi Diagrams in the Plane. *IEEE Transactions on Computers*, C-31(6):478–487, 1982.

[11] D. W. Matula and R. R. Sokal. Properties of Gabriel Graphs Relevant to Geographic Variation Research and the Clustering of Points in the Plane. *Geographical Analysis*, 12:205–222, 1980.

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

[13] C. H. Papadimitriou and U. V. Vazirani. On Two Geometric Problems Related to the Travelling Salesman Problem. *Journal of Algorithms*, 5:231–246, 1984.

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

[15] J. M. Smith, D. T. Lee, and J. S. Liebman. An $O(n \log n)$ Heuristic for the Rectilinear Steiner Minimal Tree Problem. *Engineering Optimization*, 4:179–192, 1980.

[16] J. M. Smith, D. T. Lee, and J. S. Liebman. An $O(n \log n)$ Heuristic for Steiner Minimal Tree Problems on the Euclidean Metric. *Networks*, 11:23–29, 1981.

[17] J. M. Smith and J. S. Liebman. Steiner Trees, Steiner Circuits and the Interference Problem in Building Design. *Engineering Optimization*, 4:15–36, 1979.

[18] J. M. Smith, R. Weiss, and M. Patel. An $O(N^2)$ Heuristic for Steiner Minimal Trees in E^3. *Networks*, 25:273–289, 1995.

[19] K. J. Supowit. The Relative Neighbourhood Graph, with an Application to Minimum Spanning Trees. *Journal of the Association for Computing Machinery*, 30(3):428–448, 1983.

[20] R. E. Tarjan. *Data Structures and Network Algorithms*. CBMS–NSF Regional Conference Series in Applied Mathematics, Vol. 44. CBMS, Washington, DC, 1983.

[21] E. A. Thompson. The Method of Minimum Evolution. *Annals of Human Genetics*, 36:333–340, 1973.

[22] G. T. Toussaint. The Relative Neighbourhood Graph of a Finite Planar Set. *Pattern Recognition*, 12(4):261–268, 1980.

[23] D. M. Warme. Personal communication, 1997.

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

[25] M. Zachariasen. Local Search for the Steiner Tree Problem in the Euclidean Plane. *European Journal of Operational Research*, to appear.