Technical Report DIKU-TR-97/21
Department of Computer Science
University of Copenhagen
Universitetsparken 1
DK-2100 KBH Ø
DENMARK

September 1997

Local Search for the Steiner Tree Problem in the Euclidean Plane

Martin Zachariasen

Local Search for the Steiner Tree Problem in the Euclidean Plane

Martin Zachariasen*

September 8, 1997

Abstract

Most heuristics for the Steiner tree problem in the Euclidean plane perform a series of iterative improvements using the minimum spanning tree as an initial solution. We may therefore characterize them as local search heuristics. In this paper, we first give a survey of existing heuristic approaches from a local search perspective, by setting up solution spaces and neighbourhood structures. Secondly, we present a new general local search approach which is based on a list of full Steiner trees constructed in a preprocessing phase. This list defines a solution space on which three neighbourhood structures are proposed and evaluated. Computational results show that this new approach is very competitive from a cost-benefit point of view. Furthermore, it has the advantage of being easy to apply to the Steiner tree problem in other metric spaces and to obstacle avoiding variants.

Keywords: heuristics, survey, local search, Steiner trees

1 Introduction

The Euclidean Steiner tree problem (ESTP) can be stated as follows: Given a set Z of n points in the Euclidean plane, find a shortest network, a *Steiner minimum tree (SMT)*, interconnecting Z. The points in Z are called *terminals*, while any junctions introduced are called *Steiner* points.

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

A shortest network spanning Z without introducing Steiner points is called a *minimum spanning tree (MST)*. In contrast to the Steiner tree problem which is NP-hard, an MST can be constructed in time $O(n \log n)$ [28]. The minimum spanning tree is the basic reference when comparing heuristics for ESTP. The ratio of the length $|SMT(Z)|$ of an SMT to the length $|MST(Z)|$ of an MST spanning the same set of terminals Z cannot be smaller than $\rho = \sqrt{3}/2 \approx 0.866$ [18]. An MST is therefore at most $\frac{2}{\sqrt{3}} - 1 \approx 15.47\%$ longer than an SMT. Recently Arora [3] showed that ESTP belongs to a class of NP-hard problems which have a polynomial-time approximation scheme, i.e., we can find a solution within a factor $1 + \epsilon$ from optimum in polynomial time, for any fixed $\epsilon > 0$.

There has been a major breakthrough in the development of exact algorithms for ESTP during the last few years [44, 42]. Randomly generated instances with 50 terminals can now be solved in a few minutes on a workstation and most 1000 terminal instances in a day. Thus the need for heuristic algorithms may seem less urgent. However, as will be demonstrated in the sequel heuristic algorithms may be able to speed up exact algorithms significantly by providing high quality upper-bounds. In particular, the so-called full Steiner tree approach which is presented in this paper is well suited for this purpose, since it is based on concatenation in a manner similar to the best exact algorithm.

Most heuristics for ESTP may be characterized as local search heuristics. Local search is a general search scheme which has been applied to a wide range of combinatorial optimization problems [1]. A combinatorial optimization problem is given by a finite set $\mathcal{X}$ of solutions, where each solution $\mathbf{x} \in \mathcal{X}$ has cost $f(\mathbf{x})$. The objective is to find a solution $\mathbf{x}^* \in \mathcal{X}$ with minimum (or maximum) cost. The solutions space for ESTP is, as stated above, not finite since Steiner points may be chosen arbitrarily in the plane. However, it is possible give a finite set (with size super-exponential in n) of Steiner point candidates as shown by Melzak [26].

A neighbourhood function $\mathcal{N}$ over $\mathcal{X}$ assigns to every solution $\mathbf{x} \in \mathcal{X}$ a set $\mathcal{N}(\mathbf{x}) \subseteq \mathcal{X}$ of solutions which are "close to" $\mathbf{x}$ in some sense. Neighbourhood functions are usually intimately related to the solution representation used, e.g., the data structure storing and identifying solutions in $\mathcal{X}$. Starting from an initial solution $\mathbf{x}_0 \in \mathcal{X}$ a local search algorithm generates a chain of solutions $\mathbf{x}_0, \mathbf{x}_1, \ldots \mathbf{x}_k$ such that $\mathbf{x}_i \in \mathcal{N}(\mathbf{x}_{i-1})$ for every $i = 1, \ldots k$, continually trying to find better solutions in $\mathcal{X}$.

In this paper, we first give a survey of existing heuristics for ESTP from a local search perspective (Section 3). We classify the heuristics, discuss similarities and differences and present results from the literature on their efficiency. In Section 4 we give a detailed description of local search based on full Steiner trees. We present computational results in Section 5. Concluding remarks are given in Section 6.

2 Definitions and Basic Notions

The definitions and notions used in this paper in general follow those used in the book on the Steiner tree problem by Hwang, Richards and Winter [18] and in the local search book edited by Aarts and Lenstra [1].

We first note that Euclidean distances and Steiner point coordinates are solutions to algebraic equations and may in principle require infinite precision. It is therefore assumed that terminal locations are given as rational numbers (finite-precision) and that distances and Steiner point coordinates are rounded to finite precision.

A *Steiner tree (ST)* is a tree T interconnecting Z fulfilling the following conditions: No two edges meet at an angle less than 120° and edges incident to a Steiner point meet at exactly 120° (angle conditions). The degree of a terminal point is at most three and each Steiner point has exactly degree three (degree conditions). (In their classical exposition on the Steiner tree problem, Gilbert and Pollak [13] defined a Steiner tree as a tree that cannot be shortened by a small perturbation or by "splitting" a terminal by inserting a Steiner point. This definition and the one given in our paper are equivalent and are used interchangeably in the literature.) If, in addition, the number of Steiner points is maximal $(n-2)$, the tree is called a *full Steiner tree (FST)*. An SMT is a Steiner tree and, furthermore, a union of full Steiner trees (Figure 1).

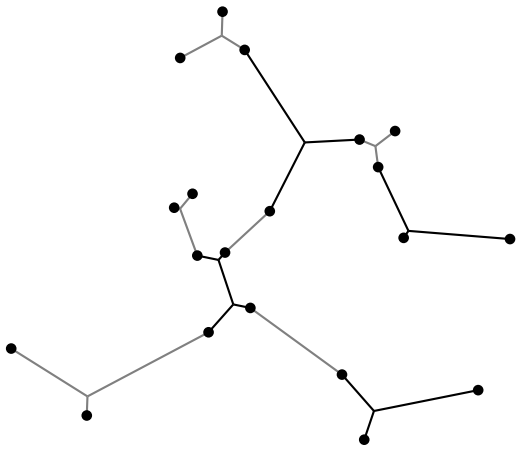


Figure 1: A Steiner minimum tree (each full Steiner tree is indicated).

A *topology* $\mathcal{T}$ is a description (graph) of the connections (edges) between terminals and Steiner points (vertices). A *Steiner topology* is a topology which fulfills the

Steiner tree degree conditions. A topology is *full* if the number of Steiner points is maximal $(n-2)$.

The shortest tree with a topology $\mathcal{T}$ is called the *relatively minimal tree*. A relatively minimal tree is *degenerate* if it has zero-length edges. An *RMT-algorithm* is a procedure for finding a relatively minimal tree; both numerical and combinatorial RMT-algorithms exist [37, 19].

Any Steiner topology can be partitioned into edge-disjoint *full components* each being a full Steiner topology. Given a full topology the locations of all Steiner points fulfilling the angle conditions - if such a configuration exists - can be found in linear time using Hwang's algorithm [17]. We refer to this procedure as an *FST-algorithm*.

Let $|T_H(Z)|$ be the length of a tree $T_H(Z)$ constructed using an heuristic H. The savings over MST

$$\sigma_H(Z) = \frac{|MST(Z)| - |T_H(Z)|}{|MST(Z)|}$$

(in percent) will be our performance measure for the heuristic. When references are made to the *average* reduction, σ_H, over the MST, it is assumed that the terminals have been distributed randomly with uniform distribution in a (unit) square.

Finally we give two definitions related to local search neighbourhoods. A neighbourhood $\mathcal{N}$ is *strongly connected* if any solution in the solution space $\mathcal{X}$ can be reached from any other solution by performing moves via $\mathcal{N}$. A neighbourhood is *weakly optimally connected* if there is a finite chain of moves from any solution in $\mathcal{X}$ to some optimal solution in $\mathcal{X}$.

3 Local Search and Steiner Tree Heuristics

The history of heuristics for ESTP dates back to the early 1970s with the contributions of Chang [8], Thompson [41] and Korhonen [23]. The heuristics by Chang and Thompson may be described as greedy Steiner point insertion algorithms which iteratively reduce the length of an initial MST, at each step inserting a "best" possible Steiner point. Although the notion "local optimization" or "local search" was not used, the analogy is apparent.

The development of general local search methods such as simulated annealing, tabu search and genetic algorithms in the 1980s has opened up a new area of research, but applications to ESTP have had limited success compared to other classical problems (for more details about these meta-heuristics we refer the reader to [1]). To the best of our knowledge, there currently exist three sim-

ulated annealing [24, 6, 14], one tabu search [16], one genetic [15] and one neural network [20] algorithm for ESTP.

In this section we classify existing heuristics for ESTP based on the underlying *solution representation*. By doing so we can set up and characterize neighbourhood structures based on these representations. This classification is obviously not complete, but it captures the essence of the heuristics described. Heuristic approaches which are difficult or impossible to characterize from a local search perspective have not been included in this survey (e.g. [22, 38, 30]). The heuristics are generally presented chronologically and, when available, running times and observed reductions over MST are noted.

The first alternative is to store complete information about the solution tree, that is, information about the topology of the tree and all Steiner point coordinates (Section 3.1). We will refer to this representation as the *Steiner tree* representation, ignoring the fact that the trees may not necessarily fulfill degree and angle conditions - the objective is obviously that these conditions are fulfilled.

A second alternative is to forget all about the topology; given a set of Steiner points S the corresponding tree is the MST over $Z \cup S$ (Section 3.2). This representation is called the *Steiner points* representation (there may be more than one MST, but this fact is usually ignored by the heuristics using this approach).

Conversely, we have the *Steiner topology* representation (Section 3.3) which stores the topology of the tree, but not the locations of Steiner points. These are given by, e.g., the relatively minimal tree.

The last two representation methods first reduce the problem to a pure combinatorial problem by fixing a set of potential Steiner points. The *graph representation* approach maps the problem to the Steiner tree problem in graphs for which several heuristic algorithms exist (Section 3.4). The *full Steiner tree (FST)* approach constructs a list of full Steiner trees in a preprocessing phase and builds an heuristic tree by concatenating FSTs from this list (Section 3.5). A summary of the performance of ESTP-heuristics is given in Section 3.6.

3.1 Steiner Trees

A solution is represented by an unrooted tree T with two types of nodes: Terminals (corresponding to Z) and Steiner points. We may assume that Steiner point coordinates are stored at the respective Steiner point nodes. No restriction is put on the topology of the tree, but the objective is to end up with a Steiner tree which fulfills degree and angle conditions. Note that in a Steiner tree T all leaves are terminals.

Most heuristics from the literature use $MST(Z)$ as the initial solution. Other

options have been suggested, but since the topology of SMTs often is closely related to that of MSTs [44, 45], it is difficult to give any reasonable alternative. However, other options are discussed at the end of this section.

Thompson [41] gave a very simple neighbour selection procedure. Let u be a vertex adjacent to two vertices v and w in the tree. We define a *Steiner point insertion* as follows: First we assume that all angles in the triangle formed by u, v and w are smaller than $120°$, otherwise we do nothing. Delete the edges (u, v) and (u, w) and insert a Steiner point s at its *Steiner position*, by adding the edges (s, u), (s, v) and (s, w) such that they make $120°$ with each other (Figure 2a).

Thompson proposed to select the two edges $\mathbf{a} = (u, v)$ and $\mathbf{b} = (u, w)$, seen as vectors in the plane, for which the scalar product $\mathbf{a} \cdot \mathbf{b}$ was as large as possible. The scalar product can also be used in higher dimensional spaces which was the actual target for Thompson's algorithm (minimum evolutionary trees). This gives preference to long edges meeting at small angles. The new Steiner point is treated as a terminal when making subsequent insertions. The algorithm stops when the improvement drops below a given threshold. Thompson gave no computational results, but indicated that the heuristic seemed to perform well on small instances.

A slightly more sophisticated insertion scheme was given by Chang [8]. He introduced a *generalized* Steiner point insertion: Let u, v and w be any three vertices in the current tree T for which a corresponding Steiner point s exists. Add s to T by connecting it to u, v and w and remove one edge from each of the two cycles created (Figure 2b).

Chang only allowed generalized insertions which enlarged existing full components. That is, an edge in an existing full component could only be removed if the two components were reconnected by the new edges inserted. Chang proved that if the generalized insertion leading to the largest positive reduction in length was performed at every step, the final heuristic tree would have two interesting properties: The degree of every Steiner point would be three and the tree would be an MST over Z and the inserted Steiner points. Chang also noted that the topology of any SMT could be obtained by performing at most $n - 2$ generalized Steiner point insertions.

The method suggested by Chang may be seen as a procedure for constructing a good Steiner topology, since the resulting tree in general has to be adjusted by relocating Steiner points within each full component. Chang used repeated relocation of Steiner points to their Steiner position until the improvement dropped below a given threshold (a better option today would be to use Hwang's FST-algorithm [17] for every full component). The worst-case running time of Chang's heuristic is huge, $O(n^4)$, but the performance is very good, $\sigma_{Chang} \approx 3.0\%$.

Finally we note an early contribution by Korhonen [23]. This heuristic may be characterized as a *tree construction* algorithm and therefore less suited for local

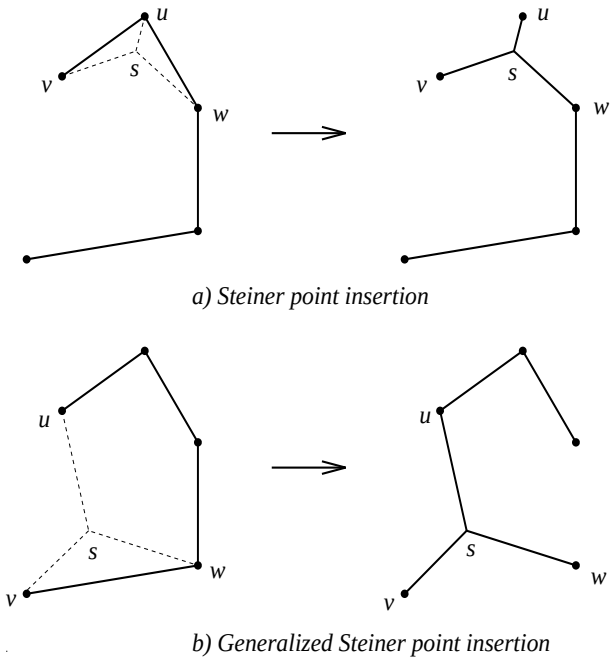


Figure 2: Simple Steiner point insertions.

search. An heuristic tree is grown and the Steiner tree property maintained for every terminal addition step. The terminals are added in the order given by Prim's MST-algorithm. After a terminal has been added the Steiner tree property is reestablished by inserting, deleting and relocating Steiner points. The algorithm is very fast, but its performance moderate, $\sigma_{Korhonen} \approx 2.5\%$.

Discussion and further refinement

Existing heuristics may be put into a local search framework by adding two important components. Firstly, initial trees other than $MST(Z)$ may be used. One choice is random spanning trees and another choice near-optimal MSTs generated by changing Prim's or Kruskal's algorithm as follows: Instead of choosing the top priority edge at every step edges should be chosen with a probability depending on their position in the priority queue.

Secondly, a procedure for deleting Steiner points from the existing tree has to be devised. Otherwise it would not really be possible to continue the search beyond the insertion of $n-2$ Steiner points. No suggestions have been made on this issue in the literature.

One of the main disadvantages of the *Steiner tree* representation is that it is difficult to retain the properties that one is actually looking for, such as degree and angle conditions. A local Steiner point insertion or deletion may have global effects, requiring the relocation of several other Steiner points.

The generalized Steiner insertion by Chang may be extended further by allowing the insertions of k-terminal SMTs (Chang only inserts 3-terminal SMTs) [12, 7].

3.2 Steiner Points

A solution is represented by a set of Steiner points S. New solutions are obtained by adding, deleting or relocating Steiner points in S (note that an SMT can have from 0 to $n-2$ Steiner points). An heuristic tree is obtained by computing $MST(Z \cup S)$. This tree is in general not a Steiner tree - there may even be Steiner points which have degree less than three. We will refer to a *clean-up* procedure as an algorithm which (iteratively) transforms the heuristic tree into a Steiner tree. This is done by deleting and relocating Steiner points in S until $MST(Z \cup S)$ is as close to a Steiner tree as required.

Representing a solution by a set of Steiner points was originally proposed by Smith and Liebman [35]. They also introduced techniques from computational geometry: A triangulation was used to generate a base set of Steiner points S_{base}. Having generated this set, the heuristic solution was constructed by greedy selection of Steiner points from S_{base}. The initial solution was $S = \emptyset$ and a neighbour to the current solution was generated as follows (we assume that $MST(Z \cup S)$ has been computed):

1. Sort S_{base} by the difference $|MST(Z \cup S)| - |MST(Z \cup S \cup \{s\})|$, $s \in S_{base}$, i.e., by length reduction obtained by adding s to S.
2. For each $s \in S_{base}$: Compute $MST(Z \cup S \cup \{s\})$; if improving then $S = S \cup \{s\}$, $S_{base} = S_{base} \setminus \{s\}$.

We note that the algorithm only adds Steiner points to S, that is, never deletes or relocates Steiner points. Also a relatively restricted base set is used for Steiner point candidates. Neighbours are generated in time $O(n^3)$ since an $O(n^2)$ MST-algorithm was used. The overall performance of the algorithm is rather poor, $\sigma_{SL} \approx 2.2\%$.

Suzuki and Iri [40] presented an algorithm in which relocation and deletion of Steiner points is a fundamental element. Starting with a set S of $n/4$ points randomly taken from the convex hull of Z, a neighbour is generated as follows:

1. Find a relatively minimal tree using the topology given by $MST(Z \cup S)$ (obviously only Steiner points are relocated).
2. Delete Steiner points having degree less than three. For every Steiner point with degree greater than three place a new Steiner point in the close neighbourhood of the original Steiner point.

3. Add Steiner points inside all angles smaller than 120° meeting at a terminal.

All added Steiner points are placed randomly but close to the terminals or Steiner points in question. The complexity of the neighbour generation scheme depends on the algorithm used for finding a relatively minimal tree. In the paper a numerical RMT-algorithm is used. The computational requirements are moderate and the performance reasonable, $\sigma_{SI} \approx 2.9\%$.

The only genetic algorithm known for ESTP was given by Hesser, Männer and Stucky [15]. The chromosome (solution) is a bit-string b representing S (the most significant bits of the coordinates of points in S). The bitstring had constant length representing exactly n Steiner points. Thus only relocations were possible. The tree corresponding to a given bitstring was obtained by constructing $MST(Z \cup S)$, removing Steiner points with degree less than three and relocating Steiner points to their Steiner position.

The genetic algorithm used standard crossover and mutation operations. This is not particularly meaningful since this may cut the bit-representation of a Steiner point and share it with another bit-string solution. The method was tested on a single 25-terminal instance (5×5 grid). Apparently it did not perform better than a greedy approach which first generates a base set S_{base} containing n randomly generated Steiner points and then adds candidates from this set to S (similar to Smith and Liebman's algorithm).

Beasley [5] and Beasley and Goffinet [6] presented two heuristics based on the Steiner point approach. The former uses the following neighbour generation strategy (initially $S = \emptyset$):

1. Let L be the set of connected subgraphs of $MST(Z \cup S)$ with exactly four vertices.
2. Sort L by the reduction obtained by replacing the MST-edges by an SMT spanning the same four vertices; for each set of vertices $K \in L$ denote by $S(K)$ the Steiner points in an SMT spanning K.
3. For each set $K \in L$ add $S(K)$ to S, given that no vertex in K has appeared previously.

Since $MST(Z \cup S)$ has bounded degree, L has size $O(n)$ and therefore it takes time $O(n \log n)$ to generate a neighbour. The neighbour is cleaned-up by removing Steiner points with degree less than three and by relocating Steiner points to their optimal positions within each full Steiner tree (using Hwang's FST-algorithm [17]). The algorithm stops when no connected subgraph with four vertices has a

shorter interconnecting tree. The running time of the algorithm is reasonable (the worst-case complexity is not given) and so is the performance, $\sigma_{Beasley} \approx 2.9\%$.

Beasley and Goffinet [6] generate neighbours using Delaunay triangulations (DT). In addition, they use simulated annealing based local search. The initial solution is again $S = \emptyset$ and neighbours are generated using the following algorithm:

1. Construct $DT(Z \cup S)$ and add the Steiner point (if it exists) of every Delaunay triangle to S.
2. Construct $MST(Z \cup S)$.
3. Delete Steiner points with degree smaller than three or greater than four from S. Relocate all Steiner points with degree three to their Steiner position. If a Steiner point s has degree four then delete s from S, construct an SMT for the four incident vertices and add the Steiner points in this SMT to S.
4. Make insertions of Steiner points if any edges meet at an angle less than 120°.
5. If any change was made in step 3 or 4 then goto step 2.

By repeating step 1, each time adding more candidate Steiner points to S, different neighbours can be generated. The neighbour generation procedure is computationally expensive, since several MSTs must be constructed. Unfortunately, it is not obvious why the algorithm stops; the average number of iterations of step 2 is not given either. The number of local search (i.e. simulated annealing) moves is limited, since the *total* number of Delaunay triangulations is only 50, independent of n for $10 \leq n \leq 100$. A temperature reduction factor of 0.7 for simulated annealing strongly indicates that this is the case. The performance is good, $\sigma_{BG} \approx 3.0\%$, but at the cost of a high computational effort.

A more pure simulated annealing algorithm was given by Grimwood [14]. This algorithm uses very little problem specific knowledge and a simple neighbourhood structure. Initially we have $S = \emptyset$ and allow additions, deletions and relocations of Steiner points in S. No clean-up procedure is used.

A new Steiner point candidate is given as the Steiner point of a triangle formed by three distinct points in $Z \cup S$, a total of $O(n^3)$ possibilities. A Steiner point relocation is seen as a deletion followed by an addition, giving a total of $O(n^4)$ neighbours. Additions, deletions and relocations are chosen with equal probability. The length of the new tree is computed by constructing MST over Z and the new set of Steiner points. This simple algorithm is remarkably effective, $\sigma_{Grimwood} \approx 3.0\%$, but computationally very expensive.

Finally we mention a neural network algorithm by Jayadeva and Bhaumik [20]. A self-organizing network is used to locate a fixed number of Steiner points. The approach is computationally very expensive and the solutions produced are significantly worse than those found by Beasley's heuristic [5].

Discussion and further refinement

From a local search point of view the *Steiner points* solution representation has so far been the most successful. It is easy to set up neighbourhood structures and even simple variants perform quite well [14]. An important issue is how to find good Steiner points to insert into the candidate set S. The main drawback is the evaluation procedure, i.e., computing $MST(Z \cup S)$ and performing clean-up, which is computationally costly.

One major advantage is that the heuristic solution is allowed to deviate completely from $MST(Z)$. Also the tree generated is, by construction, an MST over Z and the Steiner points, a property obviously shared by an SMT over Z.

3.3 Steiner Topologies

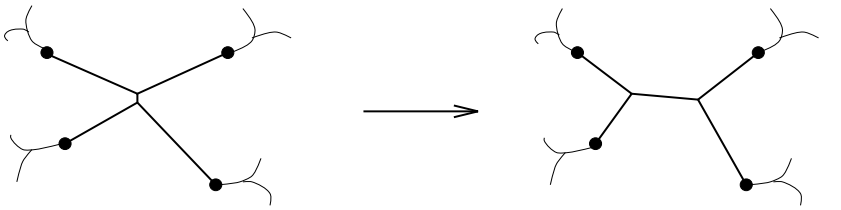
The pure variant of this approach is to store topology information about the current solution only, i.e., the location of Steiner points is given implicitly. A solution is therefore a Steiner topology $\mathcal{T}$ and the corresponding heuristic tree the relatively minimal tree.

Topology-based heuristics were first discussed, but not evaluated experimentally, by Thompson [41]. More specifically, he suggested the following approach: Construct an initial topology $\mathcal{T}_0$ by, e.g., inserting Steiner points into an MST (see Section 3.1). Find the corresponding relatively minimal tree T_0. If this tree is not degenerate (has no zero-length edges) then stop. Otherwise change the topology around zero-length edges between Steiner points (Figure 3a). The relatively minimal tree T_1 for the new topology $\mathcal{T}_1$ is then found and the procedure iterates until no such topology change can be made.

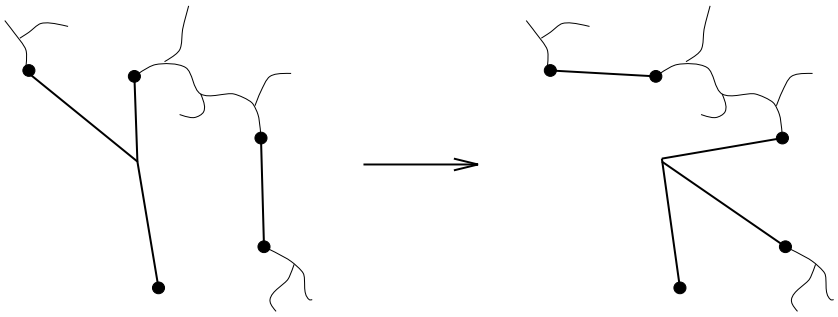
Lundy [24] put Thompson's ideas into a simulated annealing framework; this was also the first simulated annealing algorithm for ESTP. The initial solution is a randomly generated topology with $n-2$ Steiner points (the topology is not Steiner in general since degree conditions for Steiner points are not necessarily fulfilled). The neighbour generation procedure consists of a topology perturbation procedure (Figure 3b) and a Steiner point relocation scheme. The latter may be seen as a simplified RMT-algorithm.

The topology transformation differs from the one suggested by Thompson. Lundy showed that this scheme permits the construction of any topology from any other topology, i.e., the neighbourhood is strongly connected. It was found that

simulated annealing produced better solutions than a multi-start version of the Thompson heuristic, using a similar amount of CPU-time.



a) Thompson's transformation



b) Lundy's transformation

Figure 3: Topology transformations.

Chapeau-Blondeau, Janez and Ferrier [9] gave a fast variant of Thompson's heuristic. The initial topology, which is Steiner, is constructed from $MST(Z)$. For every terminal z with degree d in $MST(Z)$ $d-1$ Steiner points are inserted randomly but geometrically close to z.

The subsequent iterative (local search like) process, which optimizes the topology of the tree and Steiner point locations, is based on the simulation of the dynamics of a fluid film that relaxes under surface forces. Each iteration takes $O(n)$ and since only a constant number of iterations ($= 400$) is made, the overall complexity of the heuristic becomes $O(n \log n)$. The heuristic uses the topology transformation suggested by Thompson (Figure 3a). A change is made if the two connected Steiner points are closer than δ. The parameter δ is slowly decreased, allowing fewer changes at the end of the topology optimization. Considering the running time complexity, the performance of the heuristic is quite good, $\sigma_{CJF} \approx 2.8\%$.

Recently, Dreyer and Overton [11] suggested two heuristics for ESTP. The first heuristic is basically the same as Thompson's, and the second one is a variant of Korhonen's tree construction heuristic, in which a much more involved terminal addition step is used. Only very limited computational results are given and no comparisons are made to other heuristics.

An heuristic using a full Steiner topology solution representation was given by Hürlimann [16]. This is also the only tabu search algorithm known for ESTP.

A full Steiner topology is represented by a $n-3$ component vector $\mathbf{a}$ whose i'th entry is an integer $a_i \in \{0, ..., 2i\},\ 1 \leq i \leq n-3$. The corresponding tree was computed by using Smith's RMT-algorithm [37].

Starting from a full Steiner topology based on the topology of $MST(Z)$, a neighbour is obtained by changing a single component vector entry. Hürlimann proposed to change the value by at most 2 (wrapping around if necessary), which gives $4(n-3)$ neighbours.

The tabu search algorithm is very simple. Whenever a component vector entry has been changed, its value is kept fixed for a certain number of subsequent iterations. The approach was only successful for small problems (≤ 10 terminals), and the author indicates that one of the problems is the neighbourhood structure used.

Discussion and further refinement

The representation of solutions as Steiner topologies has the advantage of giving the Steiner point locations implicitly. Conversely, it has the serious drawback of having a very large, although finite, solution space (the number of different topologies grows super-exponentially). Another drawback is the need for a computationally expensive RMT-algorithm.

3.4 Graph Representation

The ESTP can be mapped to the Steiner tree problem in graphs (GSTP) by laying down a grid on the plane. The granularity of this grid depends on the precision required. All vertices on this grid that are inside the Steiner hull for Z (an area of the plane known to contain an SMT) are mapped to the graph problem as Steiner vertices. Edge weights are (obviously) the corresponding Euclidean distances.

Any local search method for GSTP may then be used to find a good heuristic solution for the graph instance (see [18]). The graph solution is mapped back to the plane and the ESTP solution cleaned-up by adjusting Steiner point locations.

Discussion and further refinement

This approach is particularly interesting when the obstacle avoiding variant is to be solved. Provan [29] gave several theoretical results and Armillotta and Mummolo [2] used a mapping to the graph problem in order to construct a good initial solution (without Steiner points) for the obstacle avoiding problem.

3.5 Full Steiner Trees

This approach first reduces ESTP to a simple selection problem. Construct a list of full Steiner trees (FSTs) $\mathcal{F} = \{F_1, F_2, ..., F_m\}$. Then find a subset $\mathcal{F}^* \subseteq \mathcal{F}$ such that the FSTs in $\mathcal{F}^*$ span all terminals and the length of the resulting tree is a short as possible. A solution can be represented by a 0-1 vector $\mathbf{x} \in \{0,1\}^m$ such that $x_i = 1$ if and only if F_i is selected, $1 \leq i \leq m$.

Smith, Lee and Liebman [34] gave an heuristic in which $\mathcal{F}$ was generated as follows (this is a slightly modified variant, see also [45]): Construct $DT(Z)$ and $MST(Z)$ which is a subgraph of $DT(Z)$. Generate 3-terminal subsets as corners of triangles in $DT(Z)$ with two MST edges and 4-terminal subsets as corners of two edge-sharing triangles in $DT(Z)$ with three connected edges from the MST. For each terminal subset find a shortest FST (if it exists) and append to $\mathcal{F}$. Finally append all MST-edges to $\mathcal{F}$.

The heuristic tree is constructed by greedy selection of FSTs from $\mathcal{F}$ in a manner similar to Kruskal's MST-algorithm. This $O(n \log n)$ heuristic is both theoretically and in practice the fastest heuristic known for ESTP and its performance is also quite good, $\sigma_{SLL} \approx 2.7\%$.

This full Steiner tree algorithm was generalized and elaborated by Zachariasen and Winter [45]. Several FST generation approaches were evaluated and it was shown experimentally that the best of these methods with high probability generates a superset of the FSTs in a corresponding SMT. Using an improved greedy concatenation method, a reduction $\sigma_{ZW} \approx 3.0\%$ could be obtained in time $O(n \log n)$ and a reduction $\sigma_{ZW+} \approx 3.1\%$ in time $O(n^2)$.

Given $\mathcal{F}$ the problem of finding a good heuristic tree is an easily stated combinatorial problem for which local search methods on the set of 0-1 vectors $\{0,1\}^m$ can be devised (Section 4).

Discussion and further refinement

A fixed list of FSTs makes it possible to avoid repeated Steiner point computations (which are expensive floating point operations). It is also an efficient reduction of the original problem to a simple selection problem. The major drawback is that we cannot in general guarantee that $\mathcal{F}$ contains an SMT.

Extensions of the full Steiner tree approach to obstacle avoiding variants have been discussed by Smith [32] and Nielsen [27].

3.6 Summary

In Table 1 we present a summary of heuristics for ESTP, by noting their local search type, average reduction over MST (when available) and running time complexity (when available). The type classification *descent method* in general stands for an iterative best improvement method.

Heuristic	**Type**	σ	**Complexity**
Steiner trees			
Thompson [41]	Descent method	-	-
Chang [8]	Descent method	3.0%	$O(n^4)$
Korhonen [23]	Tree construction	2.5%	Low
Steiner points			
Smith & Liebman [35]	Descent method	2.2%	$O(n^4)$
Suzuki & Iri [40]	Descent method	2.9%	Medium
Männer & Stucky [15]	Genetic algorithm	-	High
Beasley [5]	Descent method	2.9%	Observed $O(n^{1.3})$
Beasley & Goffinet [6]	Simulated annealing	3.0%	Observed $O(n^{2.2})$
Grimwood [14]	Simulated annealing	3.0%	High
Jayadeva & Bhaumik [20]	Neural network	-	High
Steiner topologies			
Lundy [24]	Simulated annealing	-	High
Chapeau-Blondeau et al. [9]	Descent method	2.8%	$O(n \log n)$
Dreyer & Overton [11]	Descent method	-	High
Hürlimann [16]	Tabu search	-	-
Full Steiner trees			
Smith, Lee & Liebman [34]	Tree construction	2.7%	$O(n \log n)$
Zachariasen & Winter [45]	Descent method	3.0%	$O(n \log n)$
– –	Descent method	3.1%	$O(n^2)$

Table 1: Heuristics for ESTP. Type classification and performance. An "-" indicates that no or insufficient data is available to give a reliable estimate of reduction over MST and/or running time complexity.

The table indicates that the *Steiner points* representation has been the most popular and it has also been quite successful. Genetic algorithms, simulated annealing and neural networks have been proposed and evaluated. The simulated annealing based heuristic by Beasley & Goffinet [6] has the best performance when running times are taken into account. The representations *Steiner trees* and *Steiner topologies* have had more limited success, at least from a local search point of view and when running times are considered.

The *full Steiner tree* approach has proven to be a viable ground for the construction of fast greedy heuristics for ESTP [34, 45]. In the following sections we

present several local search algorithms using this approach and show that they perform very well compared to the best known heuristics.

4 Full Steiner Tree Local Search

In this section we describe the novel FST based local search approach. Local search is performed on a preprocessed list of FSTs, $\mathcal{F} = \{F_1, F_2, ..., F_m\}$. Constructing a good and short candidate list $\mathcal{F}$ is obviously essential. A good list is one that contains as many FSTs of an SMT as possible, preferably all of them.

In a previous paper [45] we have shown that such a list, containing only a linear number of FSTs, can be constructed in time $O(n \log n)$. Here we give a summary of a similar generation method for which the *expected* number of FSTs is linear and, furthermore, present a quick and very efficient algorithm for pruning away non-optimal FSTs from $\mathcal{F}$ (Section 4.1). Full Steiner tree local search neighbourhoods are proposed in Section 4.2 and local search methods (meta-heuristics) are given in Section 4.3.

4.1 Generating and Pruning Full Steiner Trees

Our FST generation method is based on two properties *often* true for terminals spanned by an FST in an SMT (the $2 \times m$ ladder case, m odd, solved by Chung and Graham [10] disproves the general validity of these assumptions): Firstly, the terminals are geometrically *close* to each other. If two terminals, z_i and z_j, are spanned by an FST then z_j typically is one of the closest neighbours to z_i, and vice versa.

Secondly, each FST in an SMT spans very few terminals and seldomly more than five. In the study by Winter and Zachariasen [44] less than 1% of the FSTs spanned six or more terminals for randomly generated instances. In the following we let K denote the maximum number of terminals spanned by any FST generated.

Well-known structures from computational geometry, such as the Delaunay triangulation, can be used to generate small subsets of geometrically close terminals [28]. Based on the experimental evidence given in [45] we choose to use the so-called *Gabriel graph* $GG(Z)$ which is an undirected graph with Z as its vertex set. Let $D(z_i, z_j)$, $z_i, z_j \in Z$, denote a disc with $z_i z_j$ as a diameter. The terminals z_i and z_j are adjacent in $GG(Z)$ if and only if $D(z_i, z_j)$ contains no other terminal in Z. The Gabriel graph can be constructed in time $O(n \log n)$ since it is an easily identified subgraph of the Delaunay triangulation,

$DT(Z)$ (Figure 4). The Gabriel graph contains $MST(Z)$ so we have the relation $MST(Z) \subseteq GG(Z) \subseteq DT(Z)$.

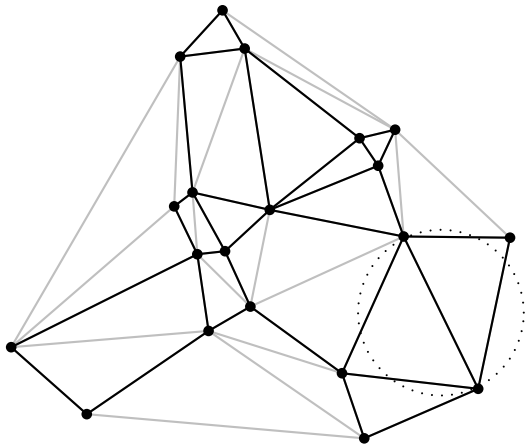


Figure 4: Gabriel graph (black edges) as a subgraph of the Delaunay triangulation (black and gray edges).

We generate subsets of terminals by enumerating all connected subgraphs of $GG(Z)$ with up to K terminals. Based on the results in [45] and preliminary experiments we choose $K = 5$. The expected number of such terminal sets is linear, since the average degree in $GG(Z)$ is bounded by a constant (for randomly generated instances approximately $52n$ subsets are generated [45]). However, for $K \geq 3$ there may be $\Omega(n^2)$ terminal subsets; consider, e.g., the corners of a regular n-gon with one terminal in its center.

FSTs spanning two terminals in an SMT must belong to $MST(Z)$, so these $n-1$ FSTs can be found in time $O(n \log n)$ time since $MST(Z) \subseteq GG(Z)$. Now consider a subset $Z_F \subseteq Z$ containing from 3 to K terminals. A shortest FST F spanning Z_F (if it exists) is obtained by generating all full Steiner topologies and applying Hwang's FST-algorithm [17].

For any two terminals z_i and z_j let $b_{z_i z_j}$ denote the length of the longest edge on the unique path between z_i and z_j in $MST(Z)$. This distance is equal to the so-called *bottleneck Steiner distance* between the two terminals. Let $MST_b(Z_F)$ be an MST over Z_F using bottleneck Steiner distances. If $|MST_b(Z_F)| < |F|$ then F may be discarded, since it cannot appear in any SMT [18].

This pruning test can be performed in total time $O(m \log n)$ and $O(n)$ space by storing $MST(Z)$ as a dynamic search tree [31] allowing longest edge queries in $O(\log n)$ time; recall that there are m FSTs and that each FST only spans

a constant number of terminals. A simpler $O(mn)$ time and $O(n^2)$ space implementation of the pruning test computes $b_{z_i z_j}$ for every pair of terminals in a preprocessing phase by making n depth-first searches in $MST(Z)$; the bottleneck Steiner distances are stored in a matrix. A practical implementation of this latter variant is actually faster than an implementation of the $O(m \log n)$ variant for $n \leq 1000$.

4.2 Local Search Neighbourhoods

Before applying local search we sort the FST list $\mathcal{F} = \{F_1, F_2, ..., F_m\}$ by ratio $|F_i|/|MST_b(Z_{F_i})|, 1 \leq i \leq m$ (smallest first), such that FSTs showing a large reduction over the bottleneck MST form the head of $\mathcal{F}$ (this ratio is always ≤ 1). In addition, we assume that the MST-edges (2-terminal FSTs) form the tail of $\mathcal{F}$ and are sorted by non-decreasing length.

FSTs in $\mathcal{F}$ spanning three or more terminals are called *large FSTs*. By sorting $\mathcal{F}$ as described above the list of large FSTs is $\mathcal{F}' = \{F_1, F_2, \ldots, F_{m'}\}$, $m' = m - (n-1)$, while MST-edges form the list $\mathcal{F}'' = \{F_{m'+1}, F_{m'+2}, \ldots, F_m\}$. In the following we assume that $m' > 0$, otherwise we return $MST(Z)$ as the heuristic solution.

We may define the local search solution space as the set of 0-1 vectors $\hat{\mathcal{X}} = \{0,1\}^m$. A solution $\hat{\mathbf{x}} \in \hat{\mathcal{X}}$ has $\hat{x}_i = 1$ if and only if F_i is selected, $1 \leq i \leq m$. Not all solutions in $\hat{\mathcal{X}}$ represent valid solutions to ESTP. The graph corresponding to a solution $\hat{\mathbf{x}}$ may be disconnected (which is critical) or it may contain cycles (which is less critical). One remedy to this problem is to add a penalty to the cost of solutions which are disconnected, e.g., some large constant times the number of components.

We avoid this problem completely by defining a solution space on large FSTs only, $\mathcal{X} = \{0,1\}^{m'}$. Then any solution vector $\mathbf{x} \in \mathcal{X}$ is a valid solution if we use MST-edges to reconnect disconnected components, if necessary. Recall that MST-edges are presorted in $\mathcal{F}''$ and we therefore can make this reconnection in linear time, $O(n)$, using a fast disjoint-set data structure.

Three different neighbourhood functions are proposed:

Flip neighbourhood $\mathcal{N}_F$

Given a solution $\mathbf{x} \in \mathcal{X}$ a neighbour in $\mathcal{N}_F(\mathbf{x})$ is constructed by flipping the value of exactly one entry in $\mathbf{x}$. This gives a total of m' neighbours; each neighbour is evaluated in time $O(m)$, so evaluating the whole neighbourhood takes time $O(m^2)$. The neighbourhood is obviously strongly connected since we can transform any solution into any other by changing entries that differ, one at a time.

An initial solution is constructed by generating a random 0-1 vector: Every entry has probability 0.5 of containing the value 1.

Insert/delete neighbourhood $\mathcal{N}_I$

When using this neighbourhood all solutions are trees (contain no cycles). An arbitrary vector $\mathbf{x}$ is transformed into a *tree solution* by using a Kruskal like algorithm which runs through the presorted list $\mathcal{F}'$: An FST F_i is added to the tree (forest) only if $x_i = 1$ and no cycle is created when it is added; if $x_i = 1$ and a cycle is created we set $x_i = 0$. Finally MST-edges are added if the corresponding graph is disconnected.

Let $\mathbf{x}$ be a tree solution. We construct a neighbour by flipping one of its entries. The neighbour corresponding to flipping an entry x_i from 0 to 1 is constructed by using the transformation described above - however, F_i is added to the tree *before* any of the other FSTs are added. This is denoted an FST-insertion, since we insert F_i into the tree by pushing other FSTs out such that a tree is obtained (see [45] for more details). If $x_i = 1$ we simply set $x_i = 0$, delete F_i from the tree and reconnect by using MST-edges. This neighbourhood can also be evaluated in time $O(m^2)$.

This neighbourhood is strongly connected since any tree solution can be transformed into any other tree solution by going through the vector of all zeros (which is $MST(Z)$). All 1-entries in the first solution are flipped to 0 (large FSTs are deleted one by one) and then the correct 0-entries are flipped to 1 as given by the second solution (FSTs are inserted one by one).

An initial solution is constructed by generating a random 0-1 vector and transforming it into a tree solution by the procedure given above.

Local insert/delete neighbourhood $\mathcal{N}_L$

This neighbourhood is a greedy variant of $\mathcal{N}_I$ which can be evaluated in time $O(m)$. Let $\mathbf{x}$ be a tree solution and T the corresponding heuristic tree. Assume that for every terminal $z \in Z$ we have access to a list T_z of FSTs spanning z in T. Let F_i be an FST which we would like to insert into T. If the FSTs in T which span some terminal in Z_{F_i} are connected we may perform the insertion *locally* in constant time [45]. Similarly, an FST F_i may be deleted in constant time if the edges in $MST(Z)$ adjacent to some terminal in Z_{F_i} are connected. The size of this neighbourhood is at most m', since some FSTs cannot be inserted or deleted because of the connectedness condition.

It can be shown by a simple counter-example that $\mathcal{N}_L$ is neither strongly nor weakly optimally connected. That is, there exist instances for which some solution cannot be transformed into an optimal solution (with respect to $\mathcal{F}$) by any sequence of $\mathcal{N}_L$ moves.

An initial solution is constructed by using the same procedure as for the insert/delete neighbourhood.

When compared to neighbourhoods proposed for other well-known combinatorial optimization problems, such as the Travelling Salesman Problem, these neighbourhoods are quite simple. However, as will be shown in Section 5, the neighbourhoods are actually very effective - provided that enough CPU time is allocated. The expected (and perhaps the worst-case) complexity for evaluating $\mathcal{N}_F$ and $\mathcal{N}_I$ can be reduced by using dynamic search trees and other sophisticated data structures, but would most likely yield no significant improvement for $n \leq 1000$, the problem size range considered in this study.

4.3 Local Search Methods

We compare three thoroughly studied meta-heuristics the literature, *repeated descent (RD)*, *simulated annealing (SA)* and *tabu search (TS)* [1]. Each of these methods have their advantages and disadvantages, and we will show that they perform quite differently on the three neighbourhoods presented in Section 4.2.

Repeated descent (RD)

This is a multi-start version of iterative improvement. More specifically, we generate a random initial solution and set $i = 1$. Then we increase i until an improving neighbour corresponding to flipping x_i has been found. We move to this new neighbour and increase i (wrapping around if necessary) until a new improving solution is found etc. When no improving neighbour can be found (local optimum) we generate a new initial solution and descend again. This descent method may be characterized as *deterministic first improvement*.

Simulated annealing (SA)

We use the simulated annealing variant by Johnson, Aragon, McGeoch and Schevon [21]. The parameters `INITPROB = 0.4`, `TEMPFACTOR = 0.95`, `SIZEFACTOR = 4` and `MINPERCENT = 2`, which give a relatively fast cooling schedule were used.

Tabu search (TS)

A standard attribute based variant of tabu search is used. Every FST involved in a move is given a tabu tenure chosen randomly from the interval $[5, 8]$ (we found no significant performance differences when changing the interval). Otherwise the deterministic neighbourhood evaluation scheme from *RD* is used.

5 Computational Experience

The full Steiner tree based local search approach was experimentally evaluated on a HP workstation[1] using the programming language C++ and class library LEDA (version 3.4.1) [25]. The random number generator used was the `random_source` class in LEDA. We also used LEDA's native Delaunay triangulation algorithm.

Problems instances were taken from the *OR-Library* [4]. The algorithms were evaluated on the 46 instances by Soukup and Chow (3-62 terminals) [39] and the 180 randomly generated instances by Beasley (10-1000 terminals), 15 instances for each size $10, 20, \ldots, 100, 250, 500$ and 1000 [5]. Optimal solutions are known for all these instances [44, 42].

In the following we first demonstrate the efficiency of the full Steiner tree generation method (Section 5.1) and compare the neighbourhoods and local search methods proposed (Section 5.2). The most promising of these are selected and compared to the best known (local search) heuristic, the Steiner point approach by Beasley and Goffinet [6] (Section 5.3).

5.1 Full Steiner Tree Generation and Pruning

The performance of the full Steiner tree generation method is summarized in Table 2. The algorithm uses the Gabriel graph to find subsets containing up to $K = 5$ terminals and bottleneck Steiner distances to prune FSTs. The simple $O(mn)$ pruning algorithm is used, but the CPU time of this step is negligible for the instances considered. The number of surviving FSTs - which includes the $n - 1$ MST-edges - is small and linear as expected.

5.2 Neighbourhoods and Local Search Methods

We compare neighbourhoods and local search methods on the 15 100-terminal instances. Five independent runs were made on each instance for each neighbourhood/local search method combination. The stopping condition for *SA* is the so-called freezing condition used in [21]. For *RD* and *TS* the stopping condition is given by the maximum number descents `MAXDESC` and maximum number of iterations `MAXITER`, respectively. These parameters were chosen as to make the running times comparable to *SA*, resulting in the parameter values `MAXDESC` $= 10\sqrt{n}$ and `MAXITER` $= 50\sqrt{n}$.

[1] 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).

n	CPU Time Generation (sec)	CPU Time Pruning (sec)	FST Count after Pruning
10	0.09	0.01	20.1
20	0.48	0.03	42.5
30	1.07	0.06	63.9
40	1.60	0.08	93.4
50	2.05	0.11	115.9
60	2.57	0.13	137.6
70	3.11	0.16	157.3
80	3.82	0.20	185.3
90	4.47	0.22	201.9
100	5.32	0.27	240.6
250	15.97	0.88	591.6
500	33.67	2.47	1229.9
1000	73.20	8.42	2413.7

Table 2: Full Steiner tree generation and pruning.

In Table 3 we summarize the results. Note that each number is an average over $15 \times 5 = 75$ runs and that the running times include FST generation and pruning (which takes less than 6 seconds on average and is the same for all combinations).

Two interesting observations follow immediately: Firstly, the neighbourhood $\mathcal{N}_I$ is much better than $\mathcal{N}_F$ and $\mathcal{N}_L$. Secondly, the performance of *TS* is substantially worse than for *RD* and *SA*. Local optima for $\mathcal{N}_F$ have a poor quality, more or less independent of the initial solution (*RD*). *SA* seems to be better at escaping these local optima than *TS*.

The neighbourhood $\mathcal{N}_I$ is in general very good. The reductions obtained by *RD* and *SA* are very close to the average reduction of the optimal solutions, 3.27%. For the neighbourhood $\mathcal{N}_L$ only *RD* achieves a reasonable performance; for this restricted neighbourhood the initial solution is very critical. Running times are lower than for the two other neighbourhoods and this difference increases for larger instances (recall that neighbourhood evaluation only takes $O(m)$ compared to $O(m^2)$ for the other two neighbourhoods).

5.3 Overall Performance Comparison

In this section we make a more thorough investigation of the FST based local search approach using the insert/delete neighbourhood, $\mathcal{N}_I$. Our results are compared to results reported on the same instances by Beasley and Goffinet [6]. The CPU-times in their paper have been "normalized" on basis of the *Linpack* bench-

Neighbourhood and Local Search Method		Reduction over MST (percent)	CPU Time (sec)
$\mathcal{N}_F$	*RD*	2.63 $\pm$ 0.31	32.8
	SA	3.05 $\pm$ 0.37	48.3
	TS	2.25 $\pm$ 0.40	32.3
$\mathcal{N}_I$	*RD*	3.25 $\pm$ 0.36	34.5
	SA	3.24 $\pm$ 0.38	51.3
	TS	3.15 $\pm$ 0.39	48.6
$\mathcal{N}_L$	*RD*	3.17 $\pm$ 0.35	22.7
	SA	2.61 $\pm$ 0.79	33.8
	TS	2.49 $\pm$ 0.81	28.8

Table 3: Neighbourhood and local search method comparison. The second number in the MST-reduction column is standard deviation.

mark as follows: Our HP workstation has a benchmark of approximately 40 and the SGI Indigo machine used in [6] a value between 4 and 12. Accordingly, the CPU-times reported in [6] were divided by 5, in order to make them comparable to ours.

When applied to the 46 Soukup and Chow problem instances, all methods *RD*, *SA* and *TS* found the optimum for 40 instances in every of the five runs made. For four instances (no. 18, 32, 45 and 46) optimum was found at least once out of five runs by some method. Two instances (no. 10 and 40) were never solved to optimality since not all the FSTs of an SMT were generated. For *SA* the average MST reduction was 2.78% (compared to 2.81% for the optimal solutions) and the average running time 1.1 seconds with a maximum of 21.6 seconds. When the maximum FSTs size was increased to $K = 6$ *all* instances were solved to optimality at least once out of five runs by some method.

These results are comparable to those obtained by the heuristic of Beasley and Goffinet. On basis of the solution values presented in their paper [6], they seem to have obtained optimal solutions for 45 instances (only instance no. 18 was not solved to optimality). However, it is not clear if these values are averages over several runs or just the result of one single run; recall that this heuristic is based on simulated annealing so different runs may give different results. The average reduction over MST was 2.81% with an average (normalized) running time of 1.9 seconds.

In Table 4 we present the main computational results of this paper. The three local search methods, using neighbourhood $\mathcal{N}_I$, are evaluated on the 180 randomly generated instances with 10 to 1000 terminals. The results are compared to those reported by Beasley and Goffinet and to the optimal solutions. While the running

times and the running time growth are within the same order of magnitude, FST based local search produces significantly better solutions. For $n \leq 100$ Beasley and Goffinet obtain an average reduction of 3.03% while the values for *RD*, *SA* and *TS* are 3.14%, 3.14% and 3.09%, respectively. Note that the average optimal solution reduction is 3.15%.

n	Beasley & Goffinet		$\mathcal{N}_I$ - *RD*		$\mathcal{N}_I$ - *SA*		$\mathcal{N}_I$ - *TS*		*OPT*
	Reduction over MST (percent)	CPU Time (sec)	Reduction over MST (percent)	CPU Time (sec)	Reduction over MST (percent)	CPU Time (sec)	Reduction over MST (percent)	CPU Time (sec)	Reduction over MST (percent)
10	3.22 ± 1.88	0.7	3.23 ± 1.84	0.2	3.23 ± 1.84	0.3	3.23 ± 1.84	0.3	3.25 ± 1.88
20	3.12 ± 0.97	3.3	3.15 ± 0.96	0.9	3.16 ± 0.96	1.5	3.14 ± 0.94	1.3	3.16 ± 0.99
30	2.95 ± 0.75	7.3	3.06 ± 0.74	2.2	3.06 ± 0.75	4.0	3.02 ± 0.71	3.0	3.07 ± 0.78
40	2.97 ± 0.63	15.5	3.12 ± 0.59	4.4	3.12 ± 0.59	8.2	3.07 ± 0.62	6.0	3.14 ± 0.63
50	2.92 ± 0.42	23.1	3.03 ± 0.40	7.2	3.02 ± 0.40	12.9	3.00 ± 0.41	9.4	3.03 ± 0.41
60	3.18 ± 0.37	31.1	3.27 ± 0.41	9.6	3.27 ± 0.41	16.3	3.21 ± 0.41	13.8	3.27 ± 0.42
70	2.95 ± 0.37	42.1	3.11 ± 0.37	13.2	3.10 ± 0.36	22.1	3.03 ± 0.36	18.8	3.11 ± 0.38
80	2.92 ± 0.69	78.1	3.03 ± 0.65	19.2	3.03 ± 0.65	31.2	2.98 ± 0.62	27.1	3.04 ± 0.67
90	2.95 ± 0.50	93.6	3.11 ± 0.48	23.1	3.10 ± 0.49	37.1	3.02 ± 0.49	33.8	3.12 ± 0.49
100	3.07 ± 0.34	97.7	3.25 ± 0.36	34.5	3.24 ± 0.38	51.3	3.15 ± 0.39	48.6	3.27 ± 0.38
250	-	-	3.17 ± 0.22	338.1	3.17 ± 0.22	365.1	3.08 ± 0.23	432.9	3.21 ± 0.23
500	-	-	3.27 ± 0.17	2332.4	3.30 ± 0.17	1414.9	3.20 ± 0.17	2554.0	3.33 ± 0.18
1000	-	-	3.23 ± 0.13	13904.6	3.28 ± 0.14	5678.5	3.17 ± 0.13	13896.1	3.31 ± 0.14

Table 4: Overall performance comparison. Second numbers in MST-reduction columns are standard deviations.

On larger instances ($n > 100$) solutions within 0.05% from optimum are obtained on average. The observed running time growth is super-quadratic with *SA* being closest to quadratic running time. The relative solution quality deteriorates slightly for larger instances, but by using, e.g., a slower cooling schedule for *SA*, better solutions obviously can be expected at the cost of increased running time. It should be noted that *RD* does perform remarkably well. This indicates that $\mathcal{N}_I$ is a very powerful neighbourhood, that is, the average quality of local optima is high. The poor performance of tabu search may be attributed to the small neighbourhood which makes it necessary to use a short tabu tenure; this again makes it difficult to avoid cycling.

6 Conclusion

The contributions of this paper are twofold: First we gave a comprehensive survey of all known and experimentally evaluated heuristics for ESTP from a local search perspective. The survey is the first unified classification of heuristics for ESTP. We demonstrated that the Steiner points approach so far had been the most popular and successful local search approach. Furthermore, the full Steiner tree (FST) based methods had proven to be very effective in the context of greedy heuristics.

Secondly, we presented several local search neighbourhoods and methods using FSTs. We gave three different neighbourhoods and compared well-known meta-heuristics using these neighbourhoods. Computational experiments showed that the insert/delete neighbourhood compared very favorably to the best known heuristics for ESTP.

We chose to use the most common variants of descent methods, simulated annealing and tabu search. Other meta-heuristics such as iterated descent, genetic algorithms and more sophisticated variants of tabu search may prove to be even more effective. However, there is very little room for improvement as far as quality is concerned - running times may on the other hand be improved.

Local search on FSTs may prove useful as a upper-bounding procedure for exact methods. The best known exact algorithms [44, 42] use the same two-phase scheme employed in this paper: First a list of FSTs $\mathcal{F}$ - in this case known to contain an SMT - is generated and then an SMT is obtained by concatenation of FSTs from $\mathcal{F}$. Better upper-bounding procedures would most likely improve the performance of the branch-and-cut algorithm used in the concatenation phase.

FST based local search can easily be applied to other metrics and higher dimensions, in particular the 3-dimensional Euclidean problem and the plane rectilinear problem. All metric dependence (and dependence on dimension) is restricted to the generation of a good and short FST list $\mathcal{F}$. This requires an effective algorithm for finding subsets of "close" terminals and an algorithm for constructing FSTs (or SMTs) on these subsets. There exist contributions in the literature using this basic approach for both the 3-dimensional Euclidean problem [36] and the plane rectilinear problem [33], but none of them used local search for the concatenation problem.

Extensions to obstacle avoiding variants are also evident. For these problems "closeness" must be defined appropriately, i.e., using shortest paths between terminals which avoid the obstacles. FSTs (or SMTs) on small subsets of terminals must obviously be constructed such that they avoid the obstacles; there exist, e.g., algorithms for the construction of SMTs for three terminals and one polygonal convex obstacle [43].

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

References

[1] E. H. L. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, 1997.

[2] A. Armillotta and G. Mummolo. A Heuristic Algorithm for the Steiner Problem with Obstacles. Technical report, Dipt. di Pregettazione e Produzione Industriale, Univ. degli Studi di Bari, Bari, 1993.

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

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

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

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

[7] P. Berman and V. Ramaiyer. Improved Approximations for the Steiner Tree Problem. *Journal of Algorithms*, 17(3):381–408, 1994.

[8] S. K. Chang. The Generation of Minimal Trees with a Steiner Topology. *J. Assoc. Comput. Mach.*, 19:699–711, 1972.

[9] 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*, to appear.

[10] F. R. K. Chung and R. L. Graham. Steiner Trees for Ladders. *Annals of Discrete Mathematics*, 2:173–200, 1978.

[11] D. R. Dreyer and M. L. Overton. Two Heuristics for the Steiner Tree Problem. Technical Report 724, Computer Science Department, New York University, 1996.

[12] D.-Z. Du and Y. Zhang. On Better Heuristics for Steiner Minimum Trees. *Mathematical Programming*, 57:193–202, 1992.

[13] E. N. Gilbert and H. O. Pollak. Steiner Minimal Trees. *SIAM Journal on Applied Mathematics*, 16(1):1–29, 1968.

[14] G. R. Grimwood. The Euclidean Steiner Tree Problem: Simulated Annealing and Other Heuristics. Master's thesis, Institute of Statistics and Operations Research, Victoria University of Wellington, New Zealand, 1994.

[15] J. Hesser, R. Männer, and O. Stucky. Optimization of Steiner Trees using Genetic Algorithms. In *Proceedings of the Third International Conference on Genetic Algorithm*, pages 231–236, 1989.

[16] T. Hürlimann. The Euclidean Steiner Tree Problem, Implementation of a Heuristic. Technical Report 94-14, Institute of Informatics, University of Fribourg, 1994.

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

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

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

[20] Jayadeva and B. Bhaumik. A Neural Network for the Steiner Minimal Tree Problem. *Biological Cybernetics*, 70:485–494, 1994.

[21] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning. *Operations Research*, 37(6):865–892, 1989.

[22] R. Karp. Probabilistic Analysis of Partitioning Algorithms for the Traveling Salesman Problem in the Plane. *Mathematics of Operations Research*, 2:209–224, 1977.

[23] P. Korhonen. An Algorithm for Transforming a Spanning Tree into a Steiner Tree. In *Survey of Math. Programming, Proc. of the 9-th Int. Math. Program. Symp.*, volume 2, pages 349–357. North-Holland, 1974.

[24] M. Lundy. Applications of the Annealing Algorithm to Combinatorial Problems in Statistics. *Biometrika*, 72:191–198, 1985.

[25] 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.

[26] Z. A. Melzak. On the Problem of Steiner. *Canad. Math. Bull.*, 4(2):143–148, 1961.

[27] M. Nielsen. Abstakte Voronoi-diagrammer i ESTPO-heuristikker. Master's thesis, DIKU, Department of Computer Science, University of Copenhagen, 1994.

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

[29] J. S. Provan. An Approximation Scheme for Finding Steiner Trees with Obstacles. *SIAM Journal on Computing*, 17(5):920–934, 1988.

[30] S. Ravada and A. T. Sherman. Experimental Evaluation of a Partitioning Algorithm for the Steiner Tree Problem in R^2 and R^3. *Networks*, 24:409–415, 1994.

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

[32] J. M. Smith. Generalized Steiner Network Problems in Engineering Design. In *Design Optimization*. Academic Press, 1985.

[33] 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.

[34] 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.

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

[36] 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.

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

[38] J. Soukup. Minimum Steiner Trees, Roots of a Polynomial and Other Magic. *ACM/SIGMAP Newsletter*, 22:37–51, 1977.

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

[40] A. Suzuki and M. Iri. A Heuristic Method for the Euclidean Steiner Problem as a Geometrical Optimization Problem. *Asia-Pacific Journal of Operational Research*, 3(2):109–122, 1986.

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

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

[43] P. Winter and J. M. Smith. Steiner Minimal Trees for Three Points with One Convex Polygonal Obstacle. *Annals of Operations Research*, 33:577–599, 1991.

[44] P. Winter and M. Zachariasen. Euclidean Steiner Minimum Trees: An Improved Exact Algorithm. *Networks*, to appear.

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