Large Euclidean Steiner Minimum Trees in an Hour

Pawel Winter
Dept. of Computer Science
University of Copenhagen
DK-2100 Copenhagen Ø
DENMARK

Martin Zachariasen
Dept. of Computer Science
University of Copenhagen
DK-2100 Copenhagen Ø
DENMARK

October 4, 1996

Abstract

The Euclidean Steiner tree problem asks for a shortest network interconnecting a set of terminals in the plane. Over the last decade the maximum problem size solvable within one hour (for randomly generated problem instances) has increased from 10 to approximately 50 terminals. We present a new exact algorithm, called `geosteiner96`. It has several algorithmic modifications which improve both the generation and the concatenation of full Steiner trees. On average, `geosteiner96` solves randomly generated problem instances with 50 terminals in less than 2 minutes, and problem instances with 100 terminals in less than 8 minutes. In addition to computational results for randomly generated problem instances, we present computational results for (perturbed) regular lattice instances and public library instances.

1 Introduction

The *Euclidean Steiner tree problem* asks for a shortest network spanning a given set Z of n *terminals* in the plane. Contrary to the minimum spanning tree problem, connections in *Steiner minimum trees* (SMTs) are not required to be between the terminals only. Additional intersections, called *Steiner points*, can be introduced to obtain shorter spanning networks.

The Euclidean Steiner tree problem has received considerable attention in the literature. The reader is referred to [13] for a comprehensive survey. We mention here only some basic properties of SMTs needed in the sequel.

- Steiner points are incident with exactly 3 edges making 120° with each other. We refer to this property as the *angle condition*.
- SMTs for n terminals have at most $n - 2$ Steiner points.
- SMTs are unions of *full Steiner trees* (FSTs). Terminals spanned by an FST have degree 1. FSTs have two Steiner points less than they have terminals. If two FSTs

share a terminal z, then the two edges incident with z (one from each FST) make at least 120° with each other. No terminal can therefore be in more than three FSTs.

A possible approach to find an SMT is to generate all FSTs (pruning as many non-optimal ones as possible), and apply exhaustive concatenation of the surviving FSTs.

Consider any subset Z_k of k terminals, $2 \leq k \leq n$. All FSTs for Z_k have $k-2$ Steiner points. Clearly, only the shortest one (if any) can appear in an SMT for Z. Unfortunately, the problem of finding a shortest FST for a given subset Z_k of terminals seems to be as difficult as the Euclidean Steiner tree problem itself. In order to find a shortest FST for Z_k, it is necessary to consider all possible *full topologies*, i.e., all possible ways of interconnecting k terminals and $k-2$ Steiner points such that the *degree condition* (all terminals have degree 1 and all Steiner points have degree 3) is satisfied. For instance, for $k = 4$, there are three different full topologies (Figure 1).

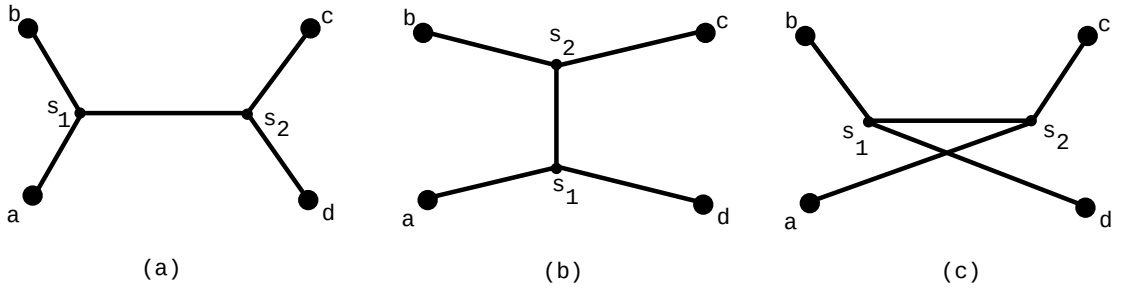

Figure 1: Different full topologies for 4 terminals

Given a full topology $\mathcal{T}_k$ for Z_k, its unique FST (if it exists) can be determined in $O(k)$ time and space. The basic idea of this FST-algorithm, discovered by Melzak [16], is as follows. If $k = 2$, there is only one full topology (with no Steiner points). The corresponding FST is the line segment between the two terminals. A full topology $\mathcal{T}_k$ with k terminals, $k \geq 3$, has at least one pair of terminals a and b adjacent to a common Steiner point s_{ab}. Let v denote the third point adjacent to s_{ab}. The edges from a and b must make 120° at s_{ab}. The location of s_{ab} in the corresponding FST is therefore very restricted. More specifically, consider the *equilateral points* e_{ab} and e_{ba}, which are third corners of equilateral triangles with the line segment from a to b as their common side. Assume that e_{ab} and e_{ba} are respectively to the left and to the right of the line segment ab. Draw circles circumscribing these two triangles. The Steiner point s_{ab} must be located on one of the two 120° *Steiner arcs* $\widehat{ab}$ and $\widehat{ba}$, respectively to the right and to the left of ab. Assume that s_{ab} is on $\widehat{ab}$ (Figure 2).

Consider a smaller full topology $\mathcal{T}_{k-1}$ where the terminals a and b, and the Steiner point s_{ab}, are replaced by e_{ab} acting as a terminal adjacent to v. $\mathcal{T}_{k-1}$ is full, and has one terminal and one Steiner point less than $\mathcal{T}_k$. A necessary condition for the existence of the FST for $\mathcal{T}_k$ with s_{ab} on $\widehat{ab}$ is that the FST for $\mathcal{T}_{k-1}$ exists. The existence of this smaller FST can be determined by such repeated *merging* (unless $k = 2$, in which case the smaller FST is a line segment; it is refereed to as the *Simpson line*). Once the FST for $\mathcal{T}_{k-1}$ is known, so is the location of v. Furthermore, if the line segment $e_{ab}v$ intersects the arc $\widehat{ab}$, the FST for $\mathcal{T}_k$ exists. The Steiner point s_{ab} must then be at this intersection.

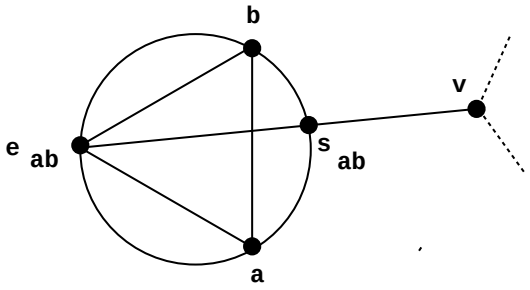

Figure 2: FST construction

The remaining Steiner points in the FST for $\mathcal{T}_k$ have the same locations as in the FST for $\mathcal{T}_{k-1}$. If the FST for $\mathcal{T}_k$ with s_{ab} on $\widehat{ab}$ does not exist, the other possibility (s_{ab} on $\widehat{ba}$) must be tried. This algorithm requires $O(2^k)$ time. However, Hwang [12] noticed that it is always possible to select a, b, and s_{ab} in $\mathcal{T}_k$ such that v is a terminal (i.e., $k = 3$), or v is adjacent to a terminal, or v is adjacent to a Steiner point (other than s_{ab}) with its two other neighbours being terminals. In each of these three cases, the locations of terminals can be used to exclude in constant time one of the Steiner arcs $\widehat{ab}$ and $\widehat{ba}$ as a location for s_{ab}. This yields an $O(k)$ time and space FST-algorithm for a given full topology $\mathcal{T}_k$.

The number $f(k)$ of full topologies (each defining a topologically different tree) with k terminals and $k-2$ Steiner points is given by

$$f(k) = \frac{(2k-4)!}{2^{k-2}(k-2)!}$$

The function f is a superexponential in k, i.e., it increases faster than an exponential function. The total number of full topologies for n terminals is given by

$$f^*(n) = \sum_{k=2}^{n} \left(\begin{array}{c} n \\ k \end{array} \right) f(k)$$

It is obvious that even for relatively small sets of terminals, the number of full topologies is of such a magnitude that their complete enumeration, followed by the linear FST-algorithm, is completely out of the question. Winter [20] suggested, instead of enumerating full topologies, to enumerate equilateral points. This approach has several advantages. First of all, the same equilateral point is generated repeatedly during the enumeration of full topologies of all subsets of terminals. This redundancy is avoided when generating equilateral points. Secondly, particular locations of terminals involved in the construction of an equilateral point e_{ab} often make it impossible to place a Steiner point s_{ab} on the corresponding Steiner arc such that the angle condition is satisfied. This can be discovered as the equilateral points are generated (non-existence of an FST for a given full topology is not discovered until the reconstruction phase). Furthermore, even if the Steiner point s_{ab} can be placed on the Steiner arc, all its feasible locations may lead to a non-optimal tree which cannot be a part of an SMT (or the arc can be narrowed resulting in subsequent exclusions of equilateral points involving e_{ab}).

As will be explained in Section 4, it is possible to rule out most of the equilateral points at a very early stage of the generation process. For each surviving equilateral point e_{ab}, terminals not involved in the construction of e_{ab} are considered one by one. If for a particular terminal z, the line segment $e_{ab}z$ intersects the Steiner arc $\widehat{ab}$ corresponding to e_{ab}, a full Steiner tree for terminals of e_{ab} and the terminal z has been identified. This FST can be submitted to additional optimality tests as explained in Section 5.

In order to emphasize the superiority of the generation of FSTs via the enumeration of equilateral points rather than the enumeration of full topologies, we merely mention here one computational result from Section 7. For randomly generated problem instances with 100 terminals, the average number of surviving FSTs of all sizes was 84.7. The total time to generate them was 431.6 seconds. It is left to the reader to determine the value of $f^*(100)$; the total number of full topologies for all subsets of 100 terminals.

Having identified all FSTs that may appear in an SMT, we want to select a subset of these such that all terminals are spanned and the length of the resulting tree is as short as possible. This will be referred to as the concatenation procedure. Basically, the procedure performs exhaustive backtrack search, but the amount of work required can be reduced substantially by preprocessing.

The advantage of using preprocessing was first demonstrated in [10] and a similar method has also been used in the rectilinear Steiner tree problem [17]. For each pair of FSTs a number of tests are performed in order to decide whether they may appear simultaneously in an SMT. This information is stored in a *compatibility matrix*, which is used to limit the size of the backtrack search tree. Furthermore, this information may be used to reduce the list of potential FSTs: An FST may become required to be included into an SMT on the basis of the compatibility information or it may be deleted if it cannot appear in every SMT. In Section 5 some new and strong compatibility tests are presented and the concatenation procedure is described in Section 6.

Hwang and Weng [14] suggested another approach to find an SMT. It requires the enumeration of full topologies for the entire set Z of terminals (i.e., there is no need to enumerate full topologies of subsets of Z). For a given full topology $\mathcal{T}$ for Z, let $D(\mathcal{T})$ denote the set of topologies which can be obtained from $\mathcal{T}$ by any sequence of collapsing pairs of adjacent vertices. The *luminary algorithm* determines, in $O(n^2)$ time, the unique minimum length (not necessarily full) tree satisfying the angle condition among all trees with topologies in $D(\mathcal{T})$. Since any topology satisfying the degree condition (each terminal has degree at most 3 and each Steiner point has degree 3) can be obtained from some (not necessarily unique) full topology for all terminals, applying the luminary algorithm to each full topology for Z will yield an SMT for Z. The advantage of this approach is that there is no need for the concatenation. However, the number of full topologies for n terminals is $f(n)$, still a very large number. Consequently, the overall complexity of the algorithm is $O(n^2 f(n))$. No implementation of the luminary approach is available. However, it seems unrealistic to expect to solve problem instances with 100 terminals in a reasonable amount of time. Nevertheless, a combination of the ideas suggested in our paper and the approach based on the luminary algorithm may prove very powerful.

2 Definitions

Let p and q be two points in the plane. The *equilateral point* e_{pq} of p and q is the third corner of the equilateral triangle with the line segment pq as one of its sides, and such that the sequence of points $\{p, e_{pq}, q\}$ makes a right turn at e_{pq}. Points p and q are called the *base points* of e_{pq}. Note that e_{pq} and e_{qp} are distinct equilateral points.

If p and q are terminals, then e_{pq} is said to be an equilateral point of *first order*. In order to simplify some arguments in the sequel, terminals will also be referred to as equilateral points of *zero order*. If base points p or q of e_{pq} are equilateral points of order $ORD(p)$ and $ORD(q)$ respectively, then $ORD(e_{pq}) = \max\{ORD(p), ORD(q)\} + 1$. The set of terminals involved in the (recursive) construction of an equilateral point e will be denoted by $Z(e)$. Note that $Z(e) = \{e\}$ if e is of zero order (i.e., if e is a terminal).

The *equilateral circle* of p and q is the circle circumscribing the equilateral triangle $\triangle pe_{pq}q$ and is denoted by C_{pq}. Its radius is denoted by r_{pq}. The center of C_{pq} is denoted by o_{pq}.

Let pq be a line segment from a point p to a point q. Its length is denoted by $|pq|$. The *direction* d_{pq} of pq is defined as the counterclockwise angle between the positive x-axis and pq. The direction of the line segment $o_{pq}p$ is denoted by γ_{pq}. The counterclockwise arc from p to q on C_{pq} is called the *Steiner arc* from p to q. It is denoted by $\widehat{pq}$. Note that $\angle po_{pq}q = 2\angle pe_{pq}q = 120°$. A circle with center in p and radius r is denoted by $C(p, r)$.

Our interest in equilateral points and Steiner arcs is due to the fact that if two terminals p and q are adjacent to a common Steiner point s_{pq}, and the sequence of points $\{p, s_{pq}, q\}$ makes a left turn at s_{pq}, then the location of s_{pq} is restricted to the Steiner arc $\widehat{pq}$. If p and q are Steiner points on Steiner arcs $\widehat{ac}$ and $\widehat{bd}$, and the sequence of points $\{p, s_{pq}, q\}$ makes a left turn, then the location of s_{pq} is restricted to the Steiner arc $\widehat{e_{ac}e_{bd}}$ where e_{ac} is the equilateral point with base points a and c, and e_{bd} is the equilateral point with base points b and d. Note that a, b, c, d can be terminals or equilateral points.

Consider an arbitrary path P between a pair of terminals z_i and z_j, and with other terminals (if any) as intermediate vertices. The *Steiner distance* between z_i and z_j along P is the length of the longest edge in P. The *bottleneck Steiner distance* $b_{z_iz_j}$ between z_i and z_j is the minimum Steiner distance between z_i and z_j taken over all paths between z_i and z_j.

3 Overview of the Exact Algorithm

3.1 Generation of Equilateral Points

An equilateral point e_{pq} which survives pruning tests (described in Section 4) is appended to a list $\mathcal{E}$. Initially, $\mathcal{E}$ contains all terminals (equilateral points of zero order).

New equilateral points are generated in the following manner. For each equilateral point $p \in \mathcal{E}$, an attempt to construct the equilateral point e_{pq} for every $q \in \mathcal{E}$ is made. If $Z(p) \cap Z(q) = \emptyset$, $|Z(p)| + |Z(q)| < n$, and e_{pq} passes all pruning tests, then e_{pq} is appended to $\mathcal{E}$. Even if e_{pq} passes all pruning tests, the location of the corresponding Steiner point

s_{pq} is usually restricted to a subarc of the Steiner arc $\widehat{pq}$. When all equilateral points in $\mathcal{E}$ have been processed as p, the entire procedure is repeated. However, for a given p, only equilateral points added after the last scan of p are used as q.

3.2 Generation of Full Steiner Trees

Once all equilateral points have been determined, FSTs can be generated in a straightforward manner. Let $e_{pq} \in \mathcal{E}$. For every terminal z, $z \notin Z(e_{pq})$, the corresponding FST exists if and only if the Simpson line $e_{pq}z$ intersects the (reduced) Steiner arc $\widehat{pq}$. As will be explained in Section 5, additional pruning tests can be applied to identify non-optimal FSTs.

3.3 Concatenation of Full Steiner Trees

The SMT is the shortest union of FSTs which spans all terminals. Starting from a single FST, a tree is grown by adding FSTs which are compatible to those already in the tree. This procedure is continued until either all terminals have been spanned, the length of the current tree is longer than a previously generated tree spanning all terminals, or no FST can be added. FSTs are removed from the tree in a last-in first-out manner until all possibilities have been evaluated (backtrack search). Details are given in Section 6.

4 Equilateral Points

4.1 Projections

Suppose that p and q are two equilateral points, and p is of non-zero order. Let a and c denote the base points of p, i.e., $p = e_{ac}$. In this subsection we examine how the location of a and c can be used to identify infeasible locations for the Steiner point s_{pq} on the Steiner arc $\widehat{pq}$ (similar arguments apply if q is an equilateral point of non-zero order). We need to distinguish between the following six subcases (Figure 3):

I: $0° \leq \angle qpa < 240°$. It is impossible to connect p with any point on $\widehat{pq}$ by a line segment intersecting $\widehat{ac}$. In other words, it is impossible to place adjacent Steiner points on $\widehat{pq}$ and $\widehat{ac}$ without violating the angle condition at s_{ac}. Hence, in this case, there is no feasible location for s_{pq} on $\widehat{pq}$. The equilateral point e_{pq} can be pruned.

II: $240° \leq \angle qpa < 300°$, and c is not in the interior of C_{pq}. It is impossible to place adjacent Steiner points on $\widehat{pq}$ and $\widehat{ac}$ without violating the angle condition (when c is on the boundary of C_{pq}, these two Steiner points will overlap). Hence, also in this case the equilateral point e_{pq} can be pruned.

III: $240° \leq \angle qpa < 300°$, and c is in the interior of C_{pq}. Consider the projection q' of c onto $\widehat{pq}$ in direction from p to c. The Steiner point on $\widehat{pq}$ cannot be placed on $\widehat{q'q}$ without violating the angle condition. Hence, the Steiner arc $\widehat{pq}$ can be reduced to

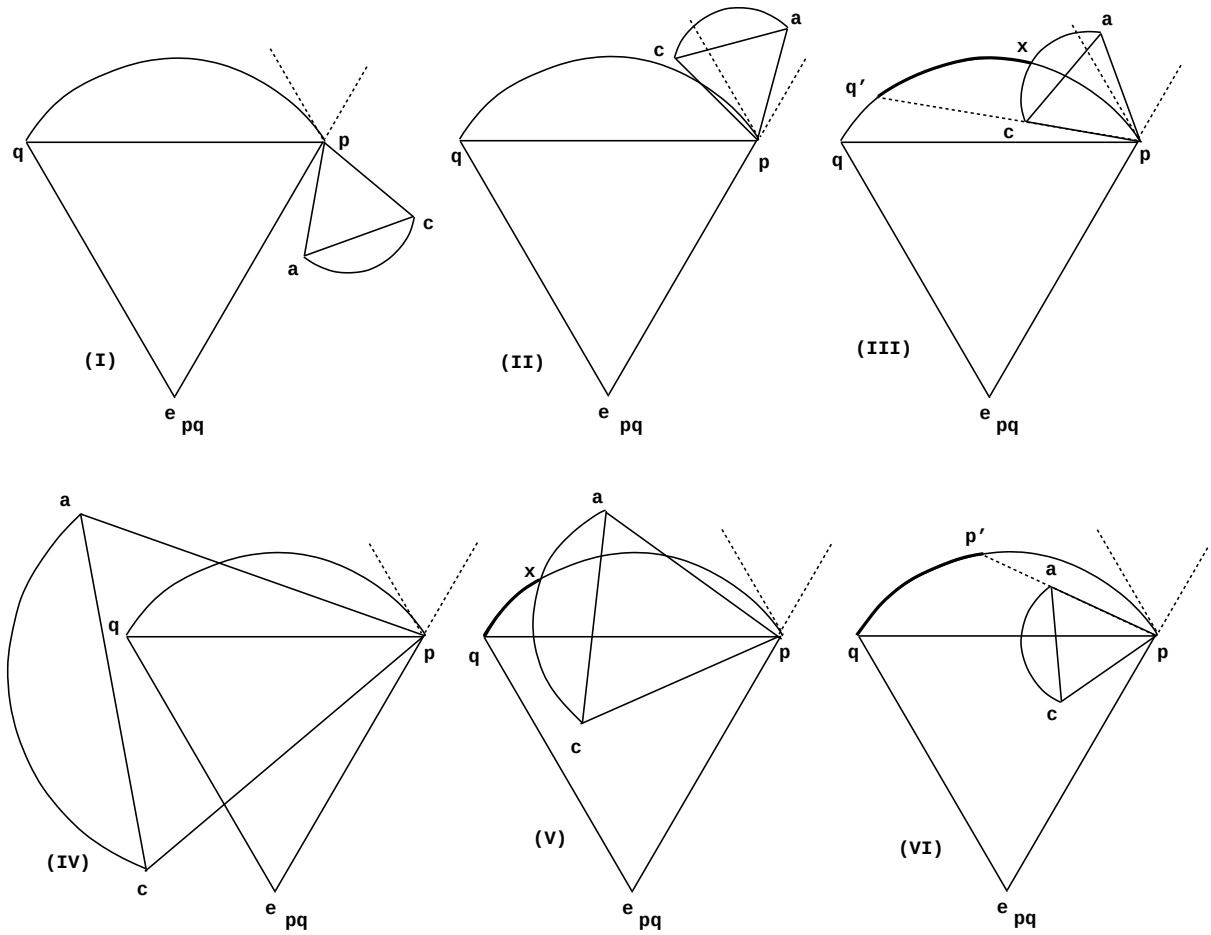

Figure 3: Projections: six subcases

$\widehat{pq'}$. The equilateral circles C_{pq} and C_{ac} intersect at p and at another point x on $\widehat{pq'}$. The angle condition at Steiner points on $\widehat{pq'}$ and $\widehat{ac}$ will be fulfilled only if the Steiner point on $\widehat{pq'}$ is on $\widehat{xq'}$. Hence, the Steiner arc $\widehat{pq'}$ can be reduced to $\widehat{xq'}$.

IV: $300° \leq \angle qpa < 360°$, and q is in the interior of C_{ac}. The intersection between C_{pq} and C_{ac} (other than at p) is not on $\widehat{pq}$. Consequently, it is impossible to place adjacent Steiner points on $\widehat{pq}$ and $\widehat{ac}$ without violating the angle condition. Hence, the equilateral point e_{pq} can be pruned.

V: $300° \leq \angle qpa < 360°$, q is not in the interior of C_{ac}, and a is not in the interior of C_{pq}. The intersection x of C_{pq} and C_{ac} (other than p) is somewhere on $\widehat{pq}$. The Steiner arc on $\widehat{pq}$ can be reduced to $\widehat{xq}$.

VI: $300° \leq \angle qpa < 360°$, q is not in the interior of C_{ac}, and a is in the interior of C_{pq}. Let p' be the projection of a onto $\widehat{pq}$ in direction from p to a. The Steiner arc $\widehat{pq}$ can be reduced to $\widehat{p'q}$.

In order to carry out the above tests and reductions efficiently, we need to:

- Determine $\angle qpa$. Since $d_{pq} = \gamma_{pq} + 150°$ and $d_{pa} = \gamma_{ac} + 30°$, it follows immediately that $\angle qpa = d_{pq} - d_{pa} = \gamma_{pq} - \gamma_{ac} + 120°$.
- Determine the projection q' of c onto $\widehat{pq}$ in the direction from p through c. In order to determine q' it is sufficient to determine $\angle po_{pq}q'$. Using straightforward geometric properties of equilateral points yields

$$\angle po_{pq}q' = 2\angle pe_{pq}q' = 2(d_{e_{pq}q'} - d_{e_{pq}p}) =$$

$$2(d_{pc} - 60° - d_{o_{pq}p} - 30°) = 2(d_{pa} + \angle apc - 60° - \gamma_{pq} - 30°) =$$

$$2(d_{o_{ac}a} + 30° + 2\alpha_{ac} - 60° - \gamma_{pq} - 30°) = 2(\gamma_{ac} + 2\alpha_{ac} - \gamma_{pq} - 60°)$$

  where $\alpha_{ac} = \angle ao_{ac}c$.
- Determine the intersection x of C_{pq} and C_{ac} other than p under the assumption that this intersection appears on $\widehat{pq}$. In order to determine x, it is sufficient to determine $\angle po_{pq}x = 2\angle po_{pq}o_{ac}$.

We assumed so far that the Steiner point s_{ac} can be located anywhere on $\widehat{ac}$. However, tests applied to e_{ac} can reduce this Steiner arc to a proper subarc $\widehat{a'c'}$. Above arguments apply if $\widehat{ac}$ is replaced by $\widehat{a'c'}$.

4.2 Lune Property

Suppose that an SMT is known to contain a line segment uv where u and v can be terminals or Steiner points. The *lune* L_{uv} of the line segment uv is defined as the intersection of two circles $C(u, |uv|)$ and $C(v, |vu|)$ (Figure 4a). It is well-known [11] that a necessary condition for the line segment uv to be in any SMT is that $L(u, v)$ contains no terminals.

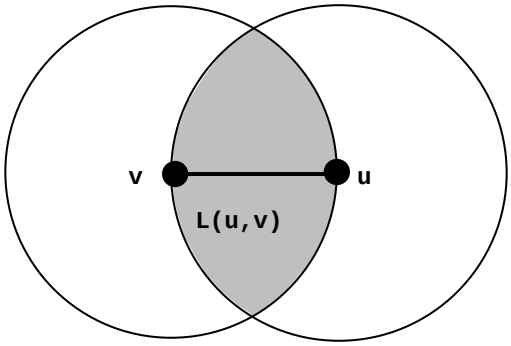

Figure 4: Lune $L(u, v)$

Suppose that p and q are equilateral points of any order (Figure 5). Let $\widehat{p_iq_i}$ be a feasible subarc of $\widehat{pq}$. If q is an equilateral point of non-zero order based on a and c, let a_i and c_i be the points on the Steiner arc $\widehat{ac}$ of q such that the projections of a_i and c_i on $\widehat{pq}$ are respectively p_i and q_i. If q is an equilateral point of zero order, then $a_i = c_i = q$. Suppose that there is a terminal z, such that $z \in L(a_i, p_i)$ (Figure 5a). Consequently, $\widehat{p_iq_i}$ can be reduced by moving p_i toward q. We are interested in a point a_i' on $\widehat{a_ic_i}$ and its projection

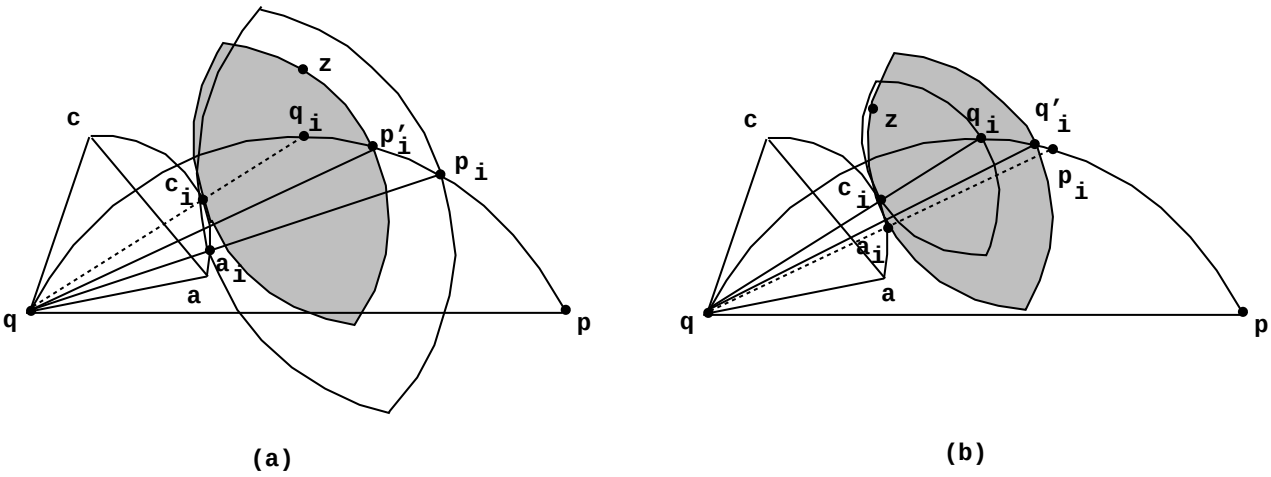

Figure 5: Lune-based reduction

p'_i on $\widehat{p_iq_i}$ such that $|a'_ip'_i| = |a'_iz|$ or $|a'_ip'_i| = |p'_iz|$ ($a'_i = q$ if q is of zero order). If there is more than one such p'_i, we are interested in the one closest to p_i.

Let $\alpha_{p_i} = \angle po_{pq}p_i$ and $\alpha_{a_i} = \angle ao_{ac}a_i$ ($\alpha_{a_i} = 0°$ if q is of zero order). Since $\widehat{p_iq_i}$ is given, α_{p_i} is known. Furthermore, $\alpha_{a_i} = 2\angle aqp_i$ if q is of non-zero order. It can be easily verified that

$$|qp_i| = \sqrt{3}r_{pq}\cos\frac{\alpha_{p_i}}{2} - r_{pq}\sin\frac{\alpha_{p_i}}{2}$$

$$|qa_i| = \sqrt{3}r_{ac}\cos\frac{\alpha_{a_i}}{2} + r_{ac}\sin\frac{\alpha_{a_i}}{2}$$

with $r_{ac} = 0$ if q is an equilateral point of zero order. The distance between p_i and a_i is then given by $|p_ia_i| = |qp_i| - |qa_i|$.

Let z be a terminal satisfying

$$|za_i| < |p_ia_i| \text{ and } |zp_i| < |p_ia_i|$$

i.e., $z \in L(a_i, p_i)$. Let a'_i be a point on $\widehat{a_ic_i}$. Let $\beta = \angle a_io_{ac}a'_i$, and let p'_i denote the projection of a'_i onto $\widehat{p_iq_i}$. Then $\beta = \angle p_io_{pq}p'_i$, and

$$|p'_ia'_i|^2 = (|qp'_i| - |qa'_i|)^2 = (A\cos\frac{\beta}{2} - B\sin\frac{\beta}{2})^2 = \frac{A^2}{2} + \frac{B^2}{2} + (\frac{A^2}{2} - \frac{B^2}{2})\cos\beta - AB\sin\beta$$

where

$$A = \sqrt{3}r_{pq}\cos\frac{\alpha_{p_i}}{2} - r_{pq}\sin\frac{\alpha_{p_i}}{2} - \sqrt{3}r_{ac}\cos\frac{\alpha_{a_i}}{2} - r_{ac}\sin\frac{\alpha_{a_i}}{2} = |p_ia_i|$$

$$B = \sqrt{3}r_{pq}\sin\frac{\alpha_{p_i}}{2} + r_{pq}\cos\frac{\alpha_{p_i}}{2} - \sqrt{3}r_{ac}\sin\frac{\alpha_{a_i}}{2} + r_{ac}\cos\frac{\alpha_{a_i}}{2}$$

The square of the distance between z and a'_i is given by

$$|za'_i|^2 = C + D\cos\beta - E\sin\beta$$

where $z = (z^x, z^y)$, $o_{ac} = (o_{ac}^x, o_{ac}^y)$, and

$$C = |o_{ac}z|^2 + r_{ac}^2$$
$$D = 2((o_{ac}^x - z^x)(a_i^x - o_{ac}^x) + (o_{ac}^y - z^y)(a_i^y - o_{ac}^y))$$
$$E = 2((o_{ac}^x - z^x)(a_i^y - o_{ac}^y) - (o_{ac}^y - z^y)(a_i^x - o_{ac}^x))$$

It follows that a_i' and p_i' with $|p_i'a_i'| = |ra_i'|$ satisfy

$$\sin\beta = \frac{-FG \stackrel{+}{-} \sqrt{F^2G^2 - (G^2+H^2)(F^2-H^2)}}{G^2+H^2}$$

where $F = \frac{A^2}{2} + \frac{B^2}{2} - C$, $G = E - AB$, and $H = D - \frac{A^2}{2} + \frac{B^2}{2}$.
Similarly, the square of the distance between z and p_i' is given by

$$|zp_i'|^2 = \bar{C} + \bar{D}\cos\beta - \bar{E}\sin\beta$$

where $o_{pq} = (o_{pq}^x, o_{pq}^y)$ and $\bar{C}$, $\bar{D}$, $\bar{E}$ are determined in analogous way as C, D, E.

It follows that a_i' and p_i' with $|p_i'a_i'| = |zp_i'|$ satisfy

$$\sin\beta = \frac{-\bar{F}\bar{G} \stackrel{+}{-} \sqrt{\bar{F}^2\bar{G}^2 - (\bar{G}^2+\bar{H}^2)(\bar{F}^2-\bar{H}^2)}}{\bar{G}^2+\bar{H}^2}$$

where $\bar{F} = \frac{A^2}{2} + \frac{B^2}{2} - \bar{C}$, $\bar{G} = \bar{E} - AB$ and $\bar{H} = \bar{D} - \frac{A^2}{2} + \frac{B^2}{2}$.

The smallest β determines the new location of p_i. In particular, if $\beta > \angle p_i o_{pq} q_i$, then there is no feasible location of s_{pq} on the Steiner arc $\widehat{pq}$.

Suppose that there is a terminal z, $z \in L(c_i, q_i)$ (Figure 5b). Consequently, $\widehat{p_iq_i}$ can be reduced by moving q_i away from q. The determination of how far q_i can be moved is analogous to the previous case.

Similar feasibility tests can be applied by examining line segments pq_i and pp_i.

4.3 Bottleneck Property

Let $\widehat{p_iq_i}$ be a subarc of the Steiner arc $\widehat{pq}$. Let a_i be as defined in Subsection 4.2. If there are two terminals $z_p \in Z(p)$ and $z_q \in Z(q)$ such that

$$b_{z_pz_q} < |a_ip_i|$$

then p_i can be moved toward q until the equality is obtained (or p_i moves beyond q_i). The new location of p_i is obtained by rotating it around o_{pq} by the smallest angle β satisfying

$$b_{z_pz_q} = A\cos\frac{\beta}{2} - B\sin\frac{\beta}{2}$$

where A and B are as in Subsection 4.2. In particular, if $\beta > \angle p_i o_{pq} q_i$, then there is no feasible location of s_{pq} on $\widehat{p_iq_i}$. It follows that

$$\sin\frac{\beta}{2} = \frac{-b_{z_p z_q} B \stackrel{+}{-} \sqrt{b_{z_p z_q}^2 B^2 - (A^2+B^2)(b_{z_p z_q}^2 - A^2)}}{A^2+B^2}$$

The bottleneck-based reductions are very powerful and not very time-consuming if the bottleneck Steiner distances are determined in the preprocessing phase in $O(n^3)$ time using an algorithm which is very similar to the all-shortest paths algorithm (see [13]).

4.4 Upper Bounds

If one is willing to use the additional computational effort, there is space for improvement of the bottleneck test when examining the line segment $p_i a_i$. Let $LCT(p)$ denote an upper bound on the Steiner minimal tree for $Z(p)$ obtained by one of several available heuristics [13]. Suppose that

$$b_{z_p z_q} + LCT(p) < |p_i p|$$

Note that $|a_i p|$ is a valid upper bound; when inserted into the above inequality, the original inequality $b_{z_p z_q} < |a_i p_i|$ is obtained. However, if a smaller $LCT(p)$ can be determined, a stronger test will be obtained.

Suppose that q is an equilateral point of non-zero order, i.e., $q = e_{ac}$. Let $z_a \in Z(a)$ and $z_c \in Z(c)$ such that $b_{z_a z_c}$ is minimized. Then

$$LCT(a) + LCT(c) + b_{z_a z_c}$$

is an upper bound on the Steiner minimal tree for $Z(q)$. Note that $LCT(a) = 0$ if a is of zero order (similarly for $LCT(c)$). Otherwise, $LCT(a)$ (and/or $LCT(c)$) is determined recursively.

Sometimes it is possible to obtain even a better upper bound. Suppose that a is an equilateral point of non-zero order. If a full Steiner tree with ac as its Simpson line exists, then $|ac|$ is a valid upper bound.

Suppose that c is of non-zero order, i.e., $c = e_{uv}$. If a full Steiner tree with au as its Simpson line exists, then $|au| + b_{z_u z_v} + LCT(v)$ is a valid upper bound. Similarly, if a full Steiner tree with av as its Simpson line exists, then $|av| + b_{z_u z_v} + LCT(u)$ is a valid upper bound. If u is also of non-zero order, i.e., $u = e_{bd}$, and a full Steiner tree with ab (respectively ad) as its Simpson line exists, then $|ab| + b_{z_u z_v} + b_{z_b z_d} + LCT(d)$ (respectively $|ad| + b_{z_u z_v} + b_{z_b z_d} + LCT(b)$) is a valid upper bound. Continuing in this manner until zero order equilateral points are encountered, and repeating this process for a if it is of non-zero order, several valid upper bounds can be obtained. The smallest is then used in the bottleneck test.

4.5 Wedge Property

Suppose that $\widehat{p_i q_i}$ is the feasible subarc of the Steiner arc $\widehat{pq}$. Draw four half-lines (Figure 6): L_1: rooted at e_{pq} through p_i, L_2: rooted at e_{pq} through q_i, L_3: rooted at p through q_i, L_4: rooted at q through p_i. Consider the regions R_1: bounded by L_1, L_2 and $\widehat{p_i q_i}$, R_2: bounded by L_2 and L_3, R_3: bounded by L_1 and L_4.

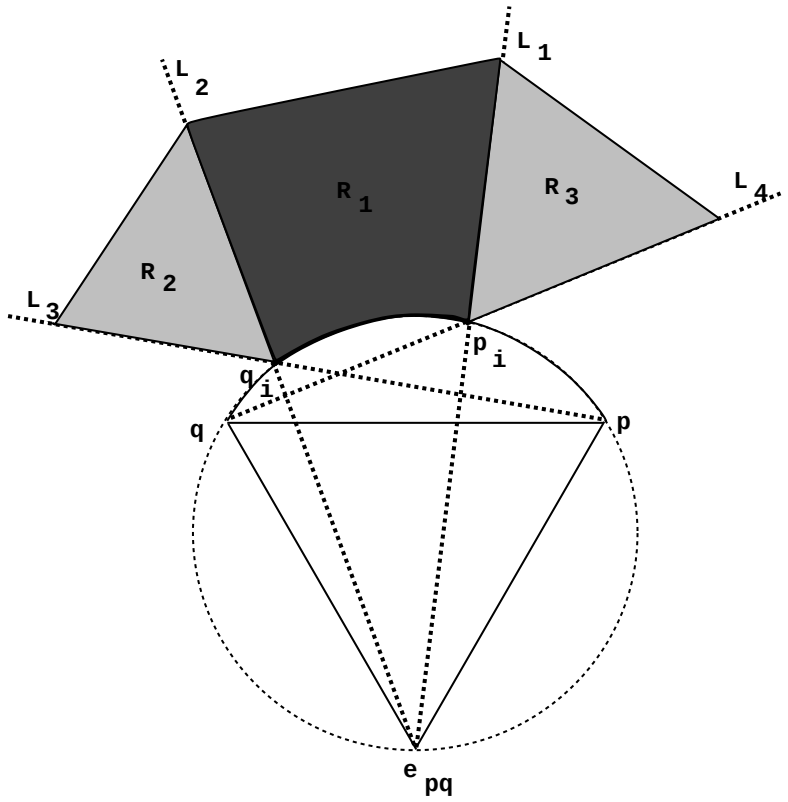

Figure 6: Wedge property

Suppose that the region R_1 contains no terminals. If R_2 or R_3 contains no terminals, then no full Steiner tree involving e_{pq} and with a Steiner point on $\widehat{p_i q_i}$ can exist.

Suppose that the region R_1 contains at least one terminal but R_2 is empty. Let z denote the terminal in R_1 so that $\angle p_i e_{pq} z$ is maximized. Let q_i' denote the intersection of $e_{pq}z$ with $\widehat{p_i q_i}$. Then no full Steiner tree involving e_{pq} has a Steiner point in the interior of $\widehat{q_i' q_i}$.

Similarly, suppose that the region R_1 contains at least one terminal but R_3 is empty. Let z denote the terminal in R_1 so that $\angle p_i e_{pq} z$ is minimized. Let p_i' denote the intersection of $e_{pq}z$ with $\widehat{p_i q_i}$. Then no full Steiner tree involving e_{pq} has a Steiner point in the interior of $\widehat{p_i p_i'}$.

5 Full Steiner Trees

5.1 Initial Pruning

Consider an equilateral point e_{pq} surviving the tests described in Section 4. Let z be a terminal, $z \notin Z(e_{pq})$. A necessary condition for the FST F with $e_{pq}z$ as its Simpson line to be a subtree of an SMT, is that z is located in the wedge region R_1 (see Subsection 4.5).

Assume that terminals are numbered from 1 to n. Let z_m denote the highest numbered terminal in $Z(e_{pq})$. Suppose that $z < z_m$. Then there exists an equilateral point e' based on terminals $Z(e_{pq}) \cup \{z\} \setminus \{z_m\}$ such that F and the FST with $e'z_m$ as its Simpson line are identical. Consequently, when generating FSTs based on e_{pq} and a terminal in R_1, only terminals with higher numbers than z_m need to be considered as z.

An FST F with $e_{pq}z$ as its Simpson line has length $|e_{pq}z|$. Hence, if a shorter tree spanning $Z(e_{pq}) \cup \{z\}$ exists, there is no need to retain F. There are several ways of obtaining such a tree. We used the following three methods in the *optimality test*:

- Minimum spanning tree for $Z(e_{pq}) \cup \{z\}$ using bottleneck Steiner distances between terminals.
- $O(n \log n)$ heuristic for the terminals $Z(e_{pq}) \cup \{z\}$ (based on the Delaunay triangulation) [18].
- Concatenation of FSTs with terminals in $Z(e_{pq}) \cup \{z\}$.

The locations of Steiner points of every surviving FST F can be determined in linear time and the test based on the lune property are applied to each of the edges of F.

5.2 Pairwise Compatibility

The set of surviving FSTs $\mathcal{F}$ can be reduced further by using the notion of compatibility. Two FSTs F_i, F_j are *compatible* if they may appear simultaneously in an SMT; they are either connected (share exactly one terminal) or disjoint (share no terminals). If two FSTs cannot appear simultaneously in an SMT, they are said to be *incompatible*. Note that two incompatible FSTs may share any number of terminals, including none. In Subsection 5.3 we generalize the notion of compatibility to any subset $\mathcal{F}' \subset \mathcal{F}$. [1]

Let F_i, F_j be two FSTs in $\mathcal{F}$. Let $Z(F_i)$ and $Z(F_j)$ denote the terminals spanned by F_i and F_j, respectively. If $|Z(F_i) \cap Z(F_j)| > 1$, or F_i and F_j intersect each other, they are incompatible.

If F_i and F_j share exactly one terminal z, the angle between the two edges incident with z must be at least 120°. If this is the case, the optimality test (see Subsection 5.1) is performed on $F_i \cup F_j$. Finally, the following *cut-and-link* test is performed. Let F_i^{-z} and F_j^{-z} denote trees spanning respectively $Z(F_i) \setminus \{z\}$ and $Z(F_j) \setminus \{z\}$. Let F_i^{+q} and

[1] Our compatibility definition is different from (and more general than) the one originally introduced in [10], in which two FSTs are compatible if they may appear simultaneously in an SMT *and* share exactly one terminal.

F_j^{+q} denote trees spanning respectively $Z(F_i) \cup \{q\}$ and $Z(F_j) \cup \{q\}$ for some terminal q, $q \notin Z(F_i) \cup Z(F_j)$.

Lemma 1 *If the edges of F_i and F_j meeting at z make an angle that exceeds* 120° *and*

$$|F_i^{-z}| + |F_j^{+q}| < |F_i \cup F_j| \quad \textit{and} \quad |F_i^{+q}| + |F_j^{-z}| < |F_i \cup F_j|$$

then $F_i \cup F_j$ cannot be a subtree of an SMT.

Proof. Assume that $F_i \cup F_j$ is a subtree of an SMT T^* (Figure 7a). Since the angle between the edges meeting at z is strictly more than 120°, no other edge of T^* can be incident with z.

If we remove one of the edges incident with z, T^* is split into two subtrees. The terminal q belongs to exactly one of them. Assume that it belongs to the same subtree as F_i does. Remove the edges in F_i and F_j from T^* and reconnect all terminals by inserting F_i^{-z} and F_j^{+q} (Figure 7b). Since $|F_i^{-z}| + |F_j^{+q}| < |F_i \cup F_j|$, this new tree is shorter than T^*, a contradiction.

If q belongs to the other subtree when the tree is split, we insert F_i^{+q} and F_j^{-z} instead (Figure 7c), obtaining a shorter tree once again. Consequently, T^* cannot be an SMT. ■

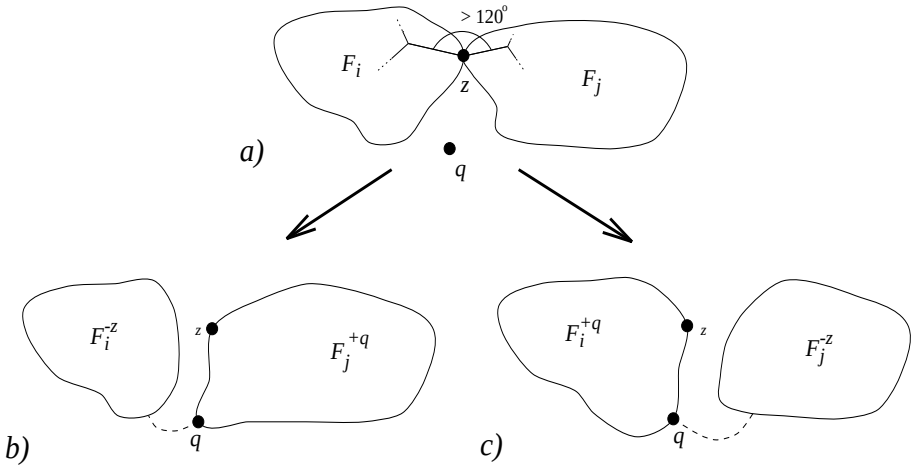

Figure 7: Cut-and-link test

The lengths of the trees F_i^{-z}, F_i^{+q}, F_j^{-z} and F_j^{+q} are computed as for the optimality test when pruning FSTs (Subsection 5.1). In addition, only terminals close to z are used as q. The length of the longest MST-edge was set to be the maximum distance allowed.

5.3 Subset Compatibility

The notion of pairwise compatibility is generalized to any subset $\mathcal{F}'$ of FSTs in $\mathcal{F}$ as follows. Let $\mathcal{F}^+ \subseteq \mathcal{F}$ be $\mathcal{F}'$ augmented with the those FSTs in $\mathcal{F} \setminus \mathcal{F}'$ which are pairwise

compatible to every FST in $\mathcal{F}'$; if $\mathcal{F}' = \emptyset$, then $\mathcal{F}^+ = \mathcal{F}$. Remove from $\mathcal{F}^+$ all FSTs $F_i \in \mathcal{F}^+ \setminus \mathcal{F}'$ for which there exists a connected, non-optimal triple F_i, F_j, F_k with $F_j, F_k \in \mathcal{F}'$. Define $G(\mathcal{F}^+)$ to be an undirected graph with the set of terminals Z as its vertices; two terminals z_i and z_j of $G(\mathcal{F}^+)$ are adjacent if and only if there exists an FST in $\mathcal{F}^+$ spanning z_i and z_j.

Let F_i be an FST, and let $z \in Z(F_i)$. If there exists a tree F_i^{-z} and a terminal q, $q \notin Z(F_i)$, such that

$$|F_i^{-z}| + |zq| < |F_i|$$

then z cannot be a leaf in any SMT containing F_i (remove F_i and reconnect by inserting F_i^{-z} and the edge zq). Such a terminal z, which is known to have degree at least 2 in every SMT containing F_i is called a *concatenation terminal of* F_i.

The subset $\mathcal{F}'$ is said to be *compatible*, if the following conditions are satisfied:

- All FSTs in $\mathcal{F}'$ are pairwise compatible.
- Every connected triple of FSTs in $\mathcal{F}'$ is optimal (Subsection 5.1). [2]
- $G(\mathcal{F}^+)$ is connected; otherwise it is not possible to construct an SMT which contains $\mathcal{F}'$ (this test is a generalization of [10, Test (i), (ii) and (iii)]).
- For every terminal $z \in Z$, *either* there exists an FST $F_i \in \mathcal{F}^+$ spanning z such that z is not a concatenation terminal of F_i *or* there is a pair of compatible FSTs $F_j, F_k \in \mathcal{F}^+$ spanning z.

5.4 Pruning using Compatibility

We are now ready to describe the pruning tests based on subset compatibility. During the pruning of FSTs we try to identify some FSTs which must be in every SMT. We denote this set of required FSTs by $\mathcal{F}^*$ and obviously we must have that $\mathcal{F}^*$ is compatible.

- If the removal of an FST $F_i \in \mathcal{F} \setminus \mathcal{F}^*$ from $\mathcal{F}$ makes $\mathcal{F}^*$ incompatible then F_i is required (append F_i to $\mathcal{F}^*$).
- If the inclusion of an FST $F_i \in \mathcal{F} \setminus \mathcal{F}^*$ to $\mathcal{F}^*$ makes $\mathcal{F}^*$ incompatible, then F_i can be deleted from $\mathcal{F}$.
- If the inclusion of two FSTs $F_i, F_j \in \mathcal{F} \setminus \mathcal{F}^*$ to $\mathcal{F}^*$ makes $\mathcal{F}^*$ incompatible then F_i and F_j are (pairwise) incompatible.

These tests are repeatedly carried out until none of them applies. Experience has shown that the computational overhead caused by these tests is small compared to the savings in the concatenation phase.

[2] In order to avoid repeated computation, information about connected and optimal triples is stored in a 2-dimensional array of lists.

6 Concatenation

6.1 Problem Decomposition

When the pruning of $\mathcal{F}$ has finished, the problem is decomposed by using the method proposed in [9]: Let $\mathcal{F}$ be the list of FSTs surviving all pruning tests. Define $G(\mathcal{F})$ as in Subsection 5.3. Then each biconnected component of $G(\mathcal{F})$ corresponds to a subproblem on which concatenation can be done separately.

Because of the effective pruning of FSTs, the graph $G(\mathcal{F})$ is sparse. Therefore this simple decomposition method is quite effective. Other decomposition methods which are based on more complicated connectivity properties of $G(\mathcal{F})$ exist [9, 17], and their inclusion is a possible future improvement to the current implementation.

6.2 Backtrack Search

Each subproblem corresponds to a subset $\mathcal{F}' \subseteq \mathcal{F}$ of the pruned set of FSTs. Let Z' denote the set of terminals spanned by the FSTs in $\mathcal{F}'$. For each FST $F_i \in \mathcal{F}'$ the compatibility matrix provides a list $\mathcal{F}_i^+ \subseteq \mathcal{F}'$ of FSTs compatible and connected to F_i, and a list $\mathcal{F}_i^- \subseteq \mathcal{F}'$ of FSTs that are incompatible to F_i (not including F_i itself). For each terminal $z \in Z'$ let $\mathcal{F}_z \subseteq \mathcal{F}'$ denote the set of FSTs spanning z.

Starting with the empty tree $T = \emptyset$, we try to append FSTs to T from a list $\mathcal{F}^+$; initially $\mathcal{F}^+ = \mathcal{F}_z$, where z is chosen such that $|\mathcal{F}_z|$ is minimum over all terminals. When an FST $F_i \in \mathcal{F}^+$ has been is added to T we set $\mathcal{F}^+ = (\mathcal{F}^+ \cup \mathcal{F}_i^+) \setminus \mathcal{F}_i^-$ (removing duplicates). This guarantees that $\mathcal{F}^+$ at any time contains exactly those FSTs that are connected to some FST in T, and are compatible to all FSTs in T. Any FST in $\mathcal{F}^+$ which during the pruning of FSTs has been identified as required to appear in an SMT is added to T first and obviously backtracking can be avoided until all other FSTs in $\mathcal{F}^+$ have been evaluated.

The subtree T is checked for compatibility as follows: A counter $c_z(T)$ is dynamically maintained for every terminal $z \in Z'$. At any point in time, $c_z(T)$ holds the number of FSTs spanning z which may be added to T at a later step, i.e., which are compatible to all FSTs in T. Note that $c_z(\emptyset) = |\mathcal{F}_z|$ for all $z \in Z'$.

Let $\bar{Z}(T) \subseteq Z'$ denote the set of terminals which are either not spanned by T, *or* are leaves in T *and* concatenation terminals of the FSTs in T. If, for a given tree T, there exists a terminal $z \in \bar{Z}(T)$ for which $c_z(T) = 0$, then T cannot be extended to an SMT; if $c_z(T) = 1$, the only FST F_i which may be added necessarily *must* be added (F_i becomes required during the backtrack search on T).

The information stored about FST triples (Subsection 5.3) is used during backtracking. Every time an FST F_i is added to T it is verified whether every connected triple including F_i in T is optimal.

Several other tests that could be performed during the backtracking process have been investigated, but none turned out to be beneficial. One possibility is to use the optimality

test on smaller subtrees, e.g., when T contains less than 20 terminals, but the computational effort needed to compute the upper bounds surpassed the savings obtained. Another possibility is to select the FSTs from $\mathcal{F}^+$ to be added to T in some strategic fashion (an example is given in [17]), but we found that even for very simple selection methods the savings could not match the extra computational effort required. The reason is apparently the strong dynamic compatibility test described above. The backtrack search tree is pruned so effectively that there is no reason to consider any particular ordering of the search.

It should also be noted that instead of growing a single tree T, one could grow a forest of FSTs. One apparent advantage is the following: By adding FSTs with 3 or more terminals first, it is possible to use a Kruskal-like algorithm when only MST edges are left to add. However, our experiments with this variant did not seem to make it preferable to the single tree alternative. Again the explanation seems to be that the dynamic compatibility test limits the search so strongly that the situation in which only MST edges are left is only seldomly reached.

7 Computational Experience

The algorithm has been implemented in C++ and the performance of the new program `geosteiner96` has been evaluated on a HP9000 workstation[3]. The class library LEDA-R-3.3 [15] has been used and the random number generator was the `random_source` class in LEDA. Experiments have been made on randomly generated instances, regular lattices and public library instances.

7.1 Randomly Generated Problem Instances

The overall performance of `geosteiner96` has been evaluated on problem instances for which terminals have been drawn uniformly from a square. One hundred instances were generated for each size $10, 20, ..., 150$. In the following we present CPU-time measurements (all in seconds), FST-counts and various properties related to the optimal solutions [4].

Table 1 shows the average CPU-time used for each of the three phases: equilateral point generation, full Steiner tree generation, and concatenation. Figures 8, 9, 10, 11 present the same numbers graphically and, in addition, give the maximum CPU-time used for each problem instance size. Note the logarithmic scaling in Figures 10 and 11.

The first two phases show average quadratic growth, $O(n^{2.07})$ and $O(n^{1.99})$ respectively, while the concatenation phase clearly is exponential. However, the concatenation time is negligible for problem instances with less than 100 terminals, but dominates for problem instances larger than 130 terminals.

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

[4] Detailed CPU-time measurements are only reported for problem sizes up to 140 terminals, since we were not able to solve one of the 150 terminal instances within a cut-off time of one week. However, other statistics for 150 terminal instances are presented, and for this problem size the averages are taken over the 99 solved instances.

The single most important factor that determines the running time of the concatenation phase is, as could be expected, the number of FSTs in the largest subproblem. In Figure 12 the concatenation CPU-time has been plotted as a function of the largest subproblem FST-count for each 150 terminal problem instance. Notice that the concatenation time for a majority of the problems is well below one hour - few instances require more than one CPU day.

It is hard to make direct CPU-time comparisons to `edsteiner89` by Cockayne and Hewgill [10], the best exact algorithm from the literature. The CPU-time reported in [10] for equilateral point and full Steiner tree generation also shows quadratic growth, although the absolute CPU-times are about 25 times ours. This may be explained mainly by the use of of less powerful hardware. As far as the concatenation phase is concerned, direct comparison is impossible, since not all instances generated have been solved in [10] (a cut-off time of 20 hours was used). In particular, Cockayne and Hewgill were not able to solve a 45 terminal problem within the 20 hour time bound. This may be compared to the 29 minutes used by `geosteiner96` to solve the hardest 100 terminal problem.

Another measure which clearly shows the superiority of `geosteiner96` is the number of FSTs that survive all pruning tests. In Table 2 we see that the FST pruning tests described in Subsection 5.4 more than halve the number of FST candidates. The average number of surviving FSTs for 100 terminal instances is 84.7 and this may be compared to the 200 FSTs surviving the tests used by `edsteiner89`. Another interesting detail is that 39.8 of these FSTs already have been identified as being a part of an SMT before the concatenation commences. This is more than two thirds of the 58.4 FSTs in the optimal solution.

n	Equilateral Points (average)	Full Steiner Trees (average)	Concate-nation (average)	Total (average)	Total (median)	Total (max)
10	2.0	1.1	0.0	3.1	2.8	8.7
20	12.2	5.0	0.0	17.3	14.1	95.6
30	29.3	10.2	0.0	39.6	32.5	132.0
40	54.0	15.9	0.0	70.0	66.8	228.9
50	83.7	25.9	0.1	109.6	98.7	302.7
60	123.1	34.0	0.3	157.5	137.7	511.6
70	162.9	48.0	0.7	211.5	196.6	579.9
80	217.0	63.4	1.5	282.0	265.1	633.9
90	274.5	81.2	4.5	360.2	341.8	795.3
100	328.3	103.3	29.5	461.2	419.2	1740.3
110	421.1	136.0	23.9	581.0	533.1	1545.7
120	501.1	174.2	54.9	730.2	648.5	2967.0
130	600.8	225.1	223.0	1048.9	824.9	9231.5
140	704.8	268.5	2530.3	3503.6	943.9	121180.8

Table 1: CPU-times for randomly generated problem instances (seconds).

It is well-known that the length of an SMT cannot be shorter than $\sqrt{3}/2$ times the length of the minimum spanning tree (MST), that is, the reduction over the MST is at most $1 - \sqrt{3}/2 \approx 13.4\%$ [13]. The average reduction is much smaller as can been seen from Table 3. It stabilizes around 3.2%, a number which may be compared to the $2.5\% - 3.0\%$

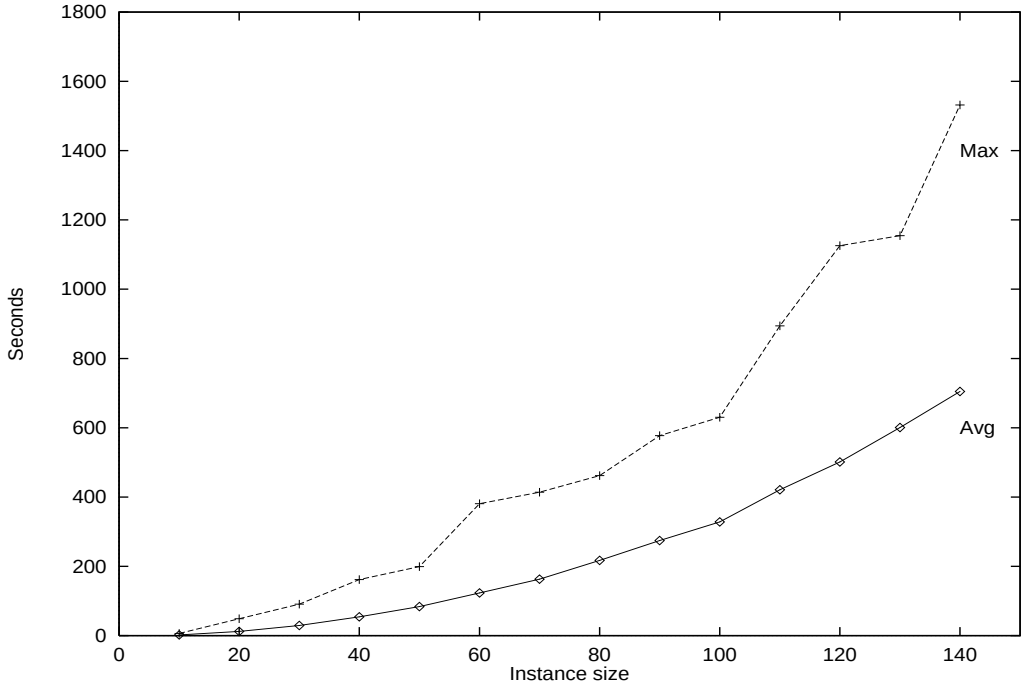

Figure 8: CPU-times for the generation of equilateral points (seconds).

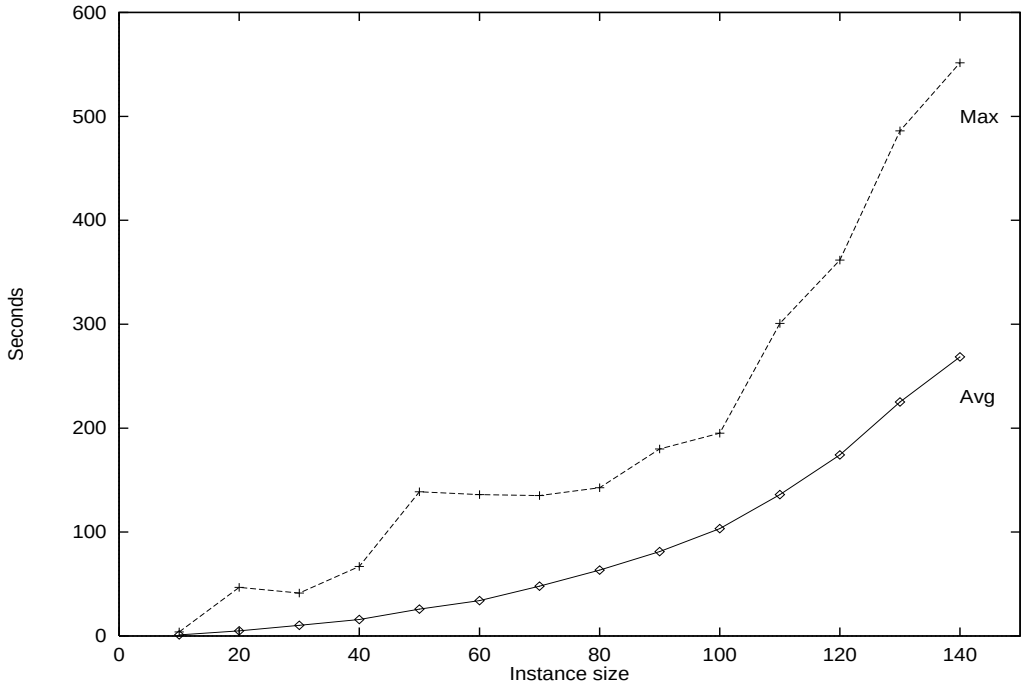

Figure 9: CPU-times for the generation of FSTs (seconds).

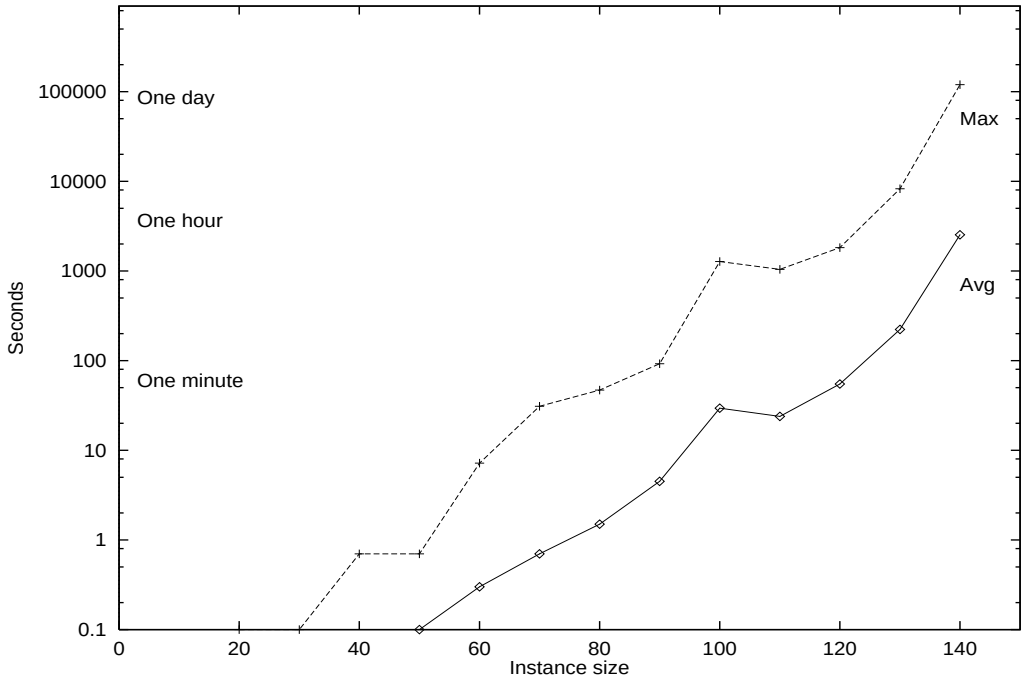

Figure 10: CPU-times for concatenation of FSTs (seconds).

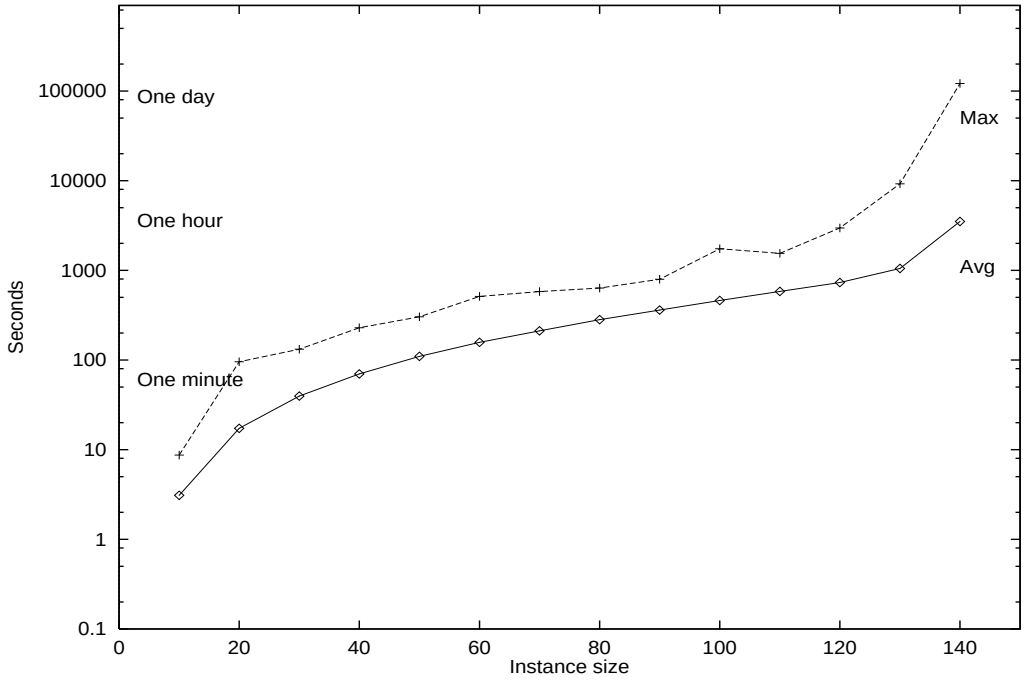

Figure 11: Total CPU-times (seconds).

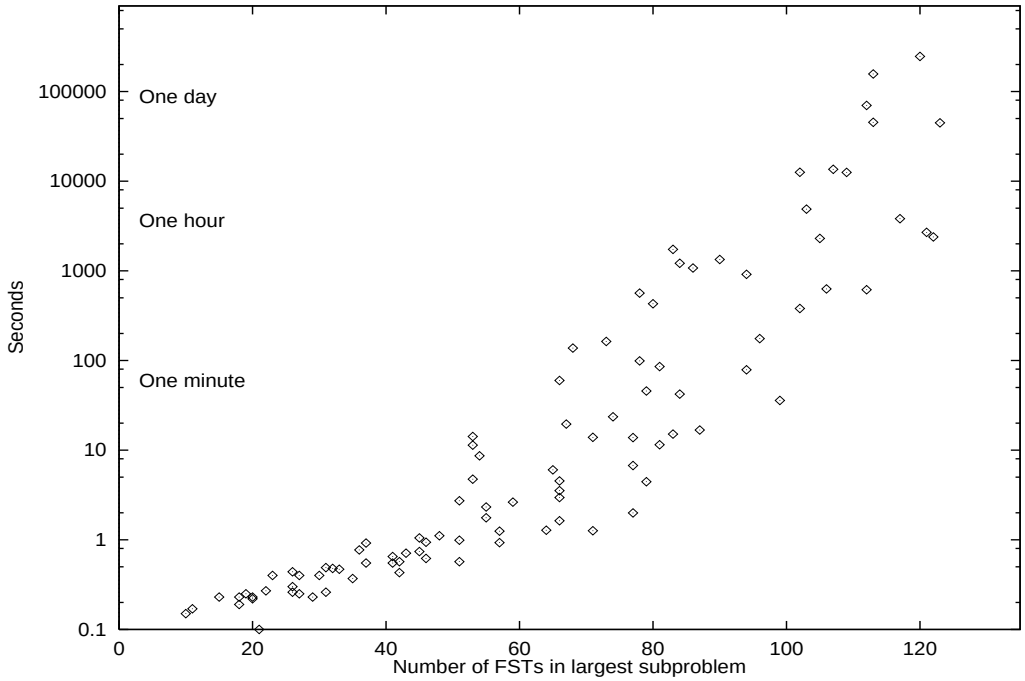

Figure 12: Concatenation CPU-time versus largest component size.

n	Before pruning	After pruning	Required	Optimal solution
10	17.8	5.7	4.9	5.3
20	39.2	13.7	9.8	11.4
30	61.6	22.4	13.7	17.3
40	83.9	31.5	16.8	23.1
50	107.1	39.7	21.2	28.8
60	129.0	49.3	24.7	34.8
70	151.5	58.9	27.8	40.8
80	174.5	66.3	32.5	46.3
90	195.4	76.4	34.9	52.1
100	216.0	84.7	39.8	58.4
110	240.0	93.9	43.9	64.6
120	263.8	104.0	46.5	70.1
130	287.4	114.5	49.3	75.9
140	309.1	125.2	52.1	81.8
150	331.5	135.0	55.3	87.7

Table 2: Average number of FSTs for randomly generated problem instances.

savings typically obtained by different types of heuristics. In addition, the variation of the reduction becomes smaller for larger instances, something which indicates that 3.2% actually may be close to the asymptotic average value for this point distribution.

Another issue, which is also relevant for the construction of heuristics, is the FST-size distribution (number of terminals in FSTs of the optimal solution). A common assumption made by heuristic algorithm designers is that the average FST-size is low and few FSTs are particularly large. Our experiments show that the average FST-size stabilizes very clearly around 2.7 (Table 3). On the other hand, the average size of the largest FST occurring in optimal solutions grows slowly and is 5.4 for problem instances with 150 terminals. This slow growth is due to a small number of relatively large FSTs - the largest among all 1500 instances has 10 terminals - but the distribution is still centered around 2-4 terminals without any observable shifting tendency: One half of the FSTs has two terminals (MST-edges), one third has three terminals, 1/8 four terminals and 1/40 five terminals. Less than 1% of the FSTs have six or more terminals. In an SMT spanning 150 terminals there are on average two FSTs with five terminals and less than one FST with six or more terminals.

n	Reduction over MST (percent)	Average FST-size (#terminals)	Maximum FST-size (#terminals)
10	3.02 ± 1.65	2.77 ± 0.39	3.87 ± 0.80
20	3.02 ± 0.97	2.69 ± 0.20	4.28 ± 0.64
30	3.15 ± 0.80	2.69 ± 0.19	4.55 ± 0.76
40	3.16 ± 0.61	2.70 ± 0.14	4.75 ± 0.77
50	3.23 ± 0.55	2.71 ± 0.14	4.81 ± 0.77
60	3.26 ± 0.58	2.71 ± 0.13	4.86 ± 0.78
70	3.18 ± 0.56	2.70 ± 0.11	5.15 ± 0.93
80	3.18 ± 0.45	2.71 ± 0.10	5.06 ± 0.72
90	3.18 ± 0.41	2.71 ± 0.09	5.11 ± 0.68
100	3.20 ± 0.44	2.70 ± 0.09	5.22 ± 0.76
110	3.23 ± 0.38	2.69 ± 0.09	5.29 ± 0.87
120	3.21 ± 0.39	2.70 ± 0.08	5.22 ± 0.95
130	3.22 ± 0.39	2.70 ± 0.08	5.24 ± 0.88
140	3.23 ± 0.37	2.70 ± 0.08	5.34 ± 0.86
150	3.23 ± 0.33	2.70 ± 0.07	5.44 ± 0.80

Table 3: Randomly generated instances. Reduction over MST, average FST-size and maximum FST-sizes in optimal solutions. Second numbers in each column are standard deviations for the respective averages.

7.2 Regular Lattices

Properties of SMTs for terminal sets with special configurations have been studied for a long time. Problem instances with the terminals arranged at corners of regular lattices of unit squares have received considerable attention, but for some time only the ladder case (2 x m terminals) had been solved completely [8]. Conjectures on the structure of SMTs for larger instances, in particular quadratic m x m instances were given [7]. Only very recently the general case was solved completely [4, 5, 6].

This means that there is no point in using a general program like `geosteiner96` to solve

these types of instances, since special polynomial-time algorithms can be constructed. Still, since these instances are known to present a major challenge to general algorithms [10], we present CPU-times needed to solve some small regular lattice instances (Table 4). It should be noted that we have made *no* algorithmic modifications to take advantage of the known structure of these problems, i.e., `geosteiner96` solves these problems as if they had been randomly generated problems.

The apparently bad performance may be explained by several factors:

- The bottleneck based pruning test in the equilateral point generation phase fails, since all bottleneck distances are 1. The result is a large number of equilateral points and full Steiner trees.
- The compatibility tests used when pruning FSTs also fail, since all terminals are "equally" close.
- The problems cannot be decomposed into subproblems by the theorem in [9] or for that matter by any other known decomposition theorem. This gives a very fast exponential growth for the running time of the concatenation phase.

In order to evaluate how sensitive `geosteiner96` is to changes in the extremely regular structure of these problems, we tried to solve a number of perturbed regular lattice instances. Each terminal is randomly repositioned within a square centered around its original location. The side of the square is 2ϵ where ϵ is a parameter which limits the maximal perturbation. Results for the 5 x 5 regular lattice are presented in Table 5. It can be seen that only a very small perturbation considerably reduces the number of FSTs surviving the pruning test, and herby the total running time. Also, the reduction over the MST decreases as the perturbation level increases.

7.3 Public Library Instances

Finally, we present optimal solution values for the 46 library instances by Soukup and Chow [19] in Table 6 (problem instances can be obtained from *OR-Library* [1]). All problem instances have been solved in less than 13 minutes by `geosteiner96`.

In Table 7 we give CPU times for another series of problem instances from *OR-Library*. These are randomly generated problem instances with 10 to 100 terminals, 15 instances for each size (see also [2, 3]). All problem instances have been solved in less than 14 minutes by `geosteiner96`. The average CPU-times are similar to the average values found for our own randomly generated problem instances (Table 1). It should be noted that none of Beasley's problem instances were previously solved to optimality.

Instance	FST-count after pruning	Opt. sol. value	Reduction over MST (%)	CPU-time (seconds)
2 x 2	1	2.7321	8.93	0.1
2 x 3	1	4.6252	7.50	1.9
2 x 4	5	6.4641	7.66	9.9
2 x 5	1	8.3451	7.28	28.6
2 x 6	14	10.1962	7.31	63.8
2 x 7	1	12.0725	7.13	126.3
3 x 3	11	7.4641	6.70	33.9
3 x 4	42	10.1962	7.31	120.9
3 x 5	82	12.9282	7.66	292.1
3 x 6	136	15.6603	7.88	575.2
3 x 7	193	18.3923	8.04	1106.3
4 x 4	106	13.6603	8.93	434.3
4 x 5	184	17.4465	8.18	1051.5
4 x 6	298	21.0562	8.45	2577.8
4 x 7	391	24.7495	8.34	5961.2
5 x 5	306	22.1244	7.82	3323.1
5 x 6	468	26.5885	8.32	19810.0
5 x 7	632	31.2136	8.19	445005.2

Table 4: Regular lattice problems. FST-count, optimal solution value and total CPU-time.

Maximal perturbation (ϵ)	FST-count after pruning	Opt. sol. value	Reduction over MST (%)	CPU-time (seconds)
0.40	27	19.85	4.32	105.8
0.20	48	20.95	5.11	301.2
0.10	75	21.57	5.99	602.9
0.05	124	21.86	6.76	1027.6
0.00	306	22.12	7.82	3323.1

Table 5: Perturbed 5 x 5 regular lattice problem. Averages over 10 instances for each perturbation level.

Problem number	n	Optimal solution	Total CPU-time	Problem number	n	Optimal solution	Total CPU-time
1	5	1.6644	0.29	24	4	0.2528	0.06
2	6	1.5005	1.10	25	3	0.1990	0.01
3	7	2.0777	0.96	26	3	0.1243	0.00
4	8	2.1388	0.79	27	4	1.1782	0.03
5	6	2.0441	0.86	28	4	0.2044	0.06
6	12	2.1842	5.67	29	3	1.4660	0.03
7	**12**	**2.2053**	**5.08**	**30**	**12**	**1.0198**	**210.52**
8	12	2.1778	5.00	**31**	**14**	**2.3322**	**2.14**
9	**7**	**1.5594**	**9.60**	**32**	**19**	**2.8142**	**49.62**
10	6	1.5988	1.58	**33**	**18**	**2.2258**	**22.27**
11	6	1.2741	0.12	**34**	**19**	**2.1381**	**21.37**
12	**9**	**1.6483**	**3.70**	**35**	**18**	**1.3554**	**17.48**
13	**9**	**1.2734**	**1.59**	36	4	0.8789	0.07
14	12	2.2049	2.38	**37**	**8**	**0.7660**	**1.40**
15	14	1.2304	1.04	38	14	1.4248	1.68
16	3	1.1668	0.02	39	14	1.4312	1.26
17	**10**	**1.6428**	**0.98**	**40**	**10**	**1.4180**	**7.77**
18	**62**	**3.8176**	**734.92**	**41**	**20**	**1.9767**	**7.24**
19	**14**	**1.7065**	**11.88**	**42**	**15**	**1.3153**	**2.74**
20	3	1.0396	0.01	**43**	**16**	**2.3308**	**37.86**
21	**5**	**1.8182**	**0.69**	**44**	**17**	**2.1869**	**10.06**
22	4	0.5033	0.07	**45**	**19**	**1.9310**	**27.25**
23	4	0.5130	0.06	**46**	**16**	**1.3660**	**352.69**

Table 6: Soukup and Chow's library problem instances. Numbers in bold indicate previously unsolved problem instances.

n	Equilateral Points (average)	Full Steiner Trees (average)	Concate-nation (average)	Total (average)	Total (median)	Total (max)
10	1.5	1.3	0.0	2.8	2.4	6.4
20	10.7	3.9	0.0	14.6	14.9	25.3
30	30.5	8.4	0.0	38.9	37.8	71.9
40	62.2	22.9	0.1	85.3	75.7	163.4
50	74.8	23.7	0.0	98.6	92.4	149.2
60	111.5	34.3	0.1	145.9	135.0	257.5
70	157.6	46.2	0.4	204.2	185.9	335.8
80	212.3	63.8	0.5	276.5	255.2	444.8
90	240.6	78.6	0.4	319.6	298.7	419.3
100	392.7	110.0	17.3	519.9	496.8	782.1

Table 7: CPU-times for Beasley's library problem instances (seconds).

8 Concluding Remarks

The overall strategy of `geosteiner96` is very similar to that of `geosteiner` [20] and `edsteiner89` [10]. A relatively small set of FSTs that can appear in an SMT is determined during the first phase. Generated FSTs are concatenated using backtrack search to obtain an overall SMT during the second phase.

The generation of FSTs is similar to that used in `geosteiner` and `edsteiner89`[5]. However, some of the pruning tests have been modified so that they are more powerful and/or faster. Furthermore, they have been presented in a more unified manner. Computational experience indicates that the generation of FSTs cannot be improved much more for randomly generated problem instances as the number of surviving non-optimal FSTs is indeed very small. FSTs are generated approximately 25 times faster by `geosteiner96` than by `edsteiner89`. Also, fewer FSTs survive the pruning tests of `geosteiner96`, although the difference (at least for small problem instances) is not spectacular. However, even a small decrease of the number of surviving FSTs has tremendous impact on the concatenation phase.

The concatenation of FSTs in `geosteiner96` is much more efficient than in `geosteiner` and in `edsteiner89`. The stronger notion of pairwise compatibility (originally introduced in [9], and the notion of subset compatibility (introduced in our paper), together with a very powerful preprocessing of surviving FSTs, makes it possible to solve randomly generated problem instances with up to 140 terminals within one hour. This is indeed a significant improvement compared with `edsteiner89`. In particular, `geosteiner96` was able to find optimal solutions to *all* public libraries instances. Apart from obtaining optimal solutions for relatively large problem instances, `geosteiner96` will also be useful to test approximation algorithms and heuristics using larger problem instances than previously possible.

The concatenation of FSTs remains the bottleneck of `geosteiner96` as it was the case for `geosteiner` and `edsteiner89`. We believe that in order to solve even larger problem instances, one of the following three approaches should be followed. The first one is to strengthen the notion of compatibility; one possibility is to compute lower bounds by using mathematical programming. The second is to apply stronger decomposition methods (see e.g., [17]). The third is to avoid the concatenation by using the luminary algorithm [14]. This approach requires however that the generation of Steiner trees for topologies with all terminals is made more efficient and comparable to our generation of FSTs for full topologies of all subsets of terminals.

The ideas presented in this paper can be applied to more general problems where an SMT is either required to be inside a simple polygon or must avoid a set of polygonal obstacles. Furthermore, the new notion of compatibility can be adapted to the rectilinear Steiner tree problem. This would most likely yield an exact algorithm capable of solving larger problem instances than those tackled by the best exact algorithm [17].

[5] Generation of FSTs in `edsteiner89` was basically the same as in `geosteiner`. `edsteiner89` primarily focused on better concatenation of FSTs.

References

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

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

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

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

[5] M. Brazil, J. H. Rubinstein, D. A. Thomas, J. F. Weng, and N. C. Wormald. Full Minimal Steiner Trees on Lattice Sets. Preprint.

[6] M. Brazil, J. H. Rubinstein, D. A. Thomas, J. F. Weng, and N. C. Wormald. Minimal Steiner Trees for Rectangular Arrays of Lattice Points. Technical Report 24, Department of Mathematics, University of Melbourne, 1995.

[7] F. Chung, M. Gardner, and R. L. Graham. Steiner Trees on a Checkerboard. *Mathematics Magazine*, 62(2):83–96, 1989.

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

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

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

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

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

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

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

[15] 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.

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

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

[18] 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.

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

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