Steiner Trees in Industries
D.-Z. Du and X. Cheng (Eds.) pp. – - –

The Rectilinear Steiner Tree Problem: A Tutorial

Martin Zachariasen
Department of Computer Science
University of Copenhagen
E-mail: `martinz@diku.dk`

Abstract

We give a tutorial on the rectilinear Steiner tree problem in the plane. First, fundamental structural results are given with full proofs. Then, recent exact algorithms allowing the solution of problem instances with several thousand terminals are presented, and finally we review some of the many heuristics proposed for the problem.

Contents

1 Introduction

The rectilinear Steiner tree problem (RSTP) in the plane has received substantial attention over the last four decades due to its evident applications in VLSI design. Given a finite set of points (also called *terminals*) in the plane, construct a tree of minimal length that interconnects the terminals and uses only horizontal and vertical line segments. In VLSI design, the points correspond to electrical terminals that should be interconnected; minimizing the length therefore minimizes the amount of wire needed. The constraints on the orientation of the line segments come from current fabrication technology requirements.

This paper is a tutorial on the rectilinear Steiner tree problem in the plane. Fundamental results for the problem and important algorithmic developments during the last five years are presented. For a thorough survey on RSTP (covering the developments up to 1992), we refer to the excellent book by Hwang, Richards and Winter [12]. This book also covers polynomial-time solvable cases and generalizations that are not discussed in this tutorial.

The reader is expected to be a graduate student in mathematics, computer science or engineering with a moderate background in operations research. Also, researchers in related fields or engineers in VLSI design should find this tutorial useful. The text includes small exercises mainly intended for self-study. Some of these cover special cases or the basis for induction proofs given in the text.

Before embarking on the structural and algorithmic results known for RSTP, it should be noted that the problem is indeed NP-hard. This fact was established by Garey and Johnson [5]; the proof is rather involved — and since this text is devoted to geometric properties and algorithms for RSTP, we omit the NP-hardness proof here.

The tutorial is organized as follows: Firstly, we give fundamental structural properties of optimal solutions for RSTP (Section 2). We give full proofs for the theorems stated. Secondly, we show how these structural properties can be used to design practical exact algorithms for the problem (Section 3). The final part is devoted to classical heuristics and recent developments in approximation algorithms for the problem (Section 4). In particular, we give a detailed description of Arora's polynomial time approximation scheme for RSTP.

2 Structural Properties

Given a finite set Z of n points in the plane, we would like to construct a *rectilinear Steiner minimum tree (SMT)*. This is a tree that interconnects Z, consists of horizontal and vertical line segments, and has minimum total length. Equivalently, the task is to construct a Steiner minimum tree for Z under the L_1 metric: For two points $u = (u_x, u_y)$ and $v = (v_x, v_y)$, their L_1 distance is $|uv| = |u_x - v_x| + |u_y - v_y|$, that is, the sum of distances in each of the two dimensions.

In the following we will mainly use the former definition, since it gives us a direct geometric realization of SMTs. We first give some notation and definitions, and then we present three classical results, one by Hanan [8] and the other two by Hwang [11]. The notation, definitions and proofs in this section are based on [12, 16]. The proofs are somewhat simpler than those originally given by Hanan and Hwang.

2.1 Basic Notation and Definitions

An SMT consists of horizontal and vertical line *segments* that only intersect at their endpoints. The intersection points are called *nodes*. The nodes are either *terminals* (from the set Z) or *non-terminals*. We distinguish between three types of non-terminals: *corner points* (having degree two or exactly two incident perpendicular segments), *T-points* (having degree three) and *cross-points* (having degree four). T-points and cross-points are also called *Steiner points*.

A *line of segments* is a sequence of one or more adjacent, collinear segments with no terminal nodes sharing two adjacent segments (however, the endpoints of the line may be terminals). A *complete line* is a line of segments of maximal length; it is not properly contained in any other line of segments.

A corner point c is an endpoint of two complete lines, one in each of the two perpendicular directions given by the incident segments. Let u and v be the other endpoints of the incident complete lines. The pair of complete lines (cu, cv) is called a *complete corner* located at c; cu and cv are the *legs* of the complete corner.

We illustrate these definitions in Figure 1; they will form the basic elements in the proofs given below.

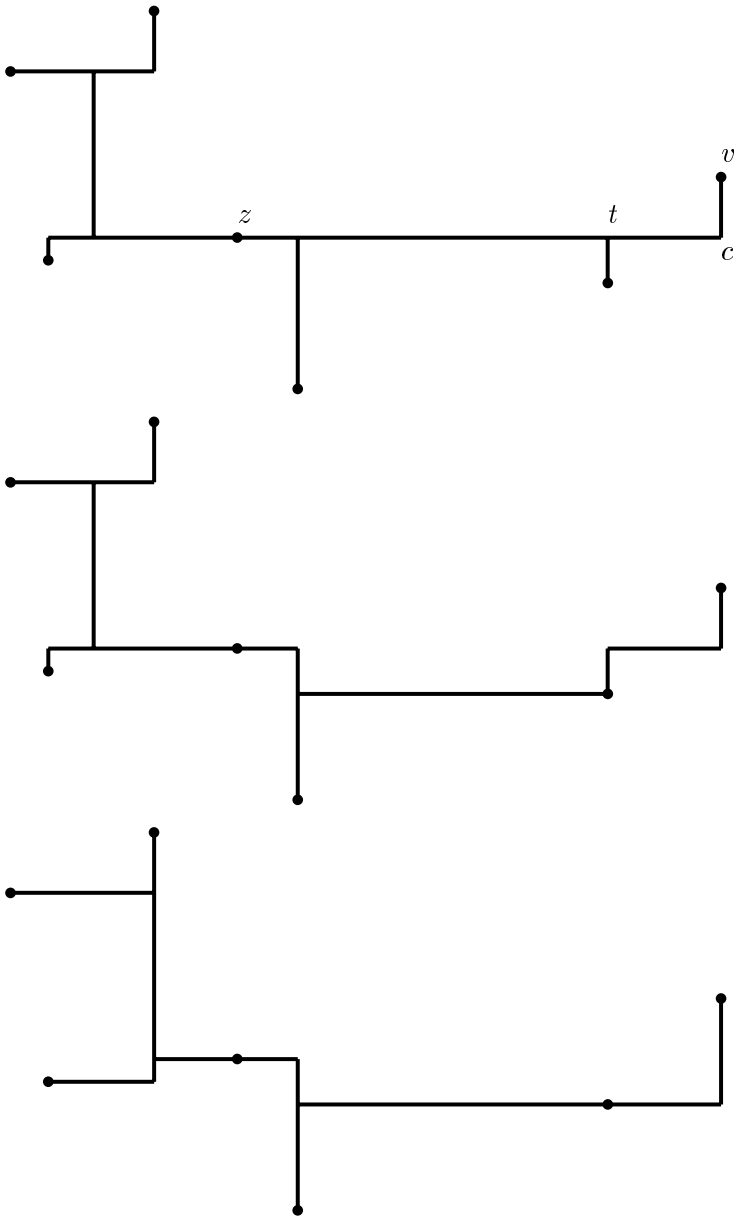


Figure 1: Three SMTs for the same terminal set. Nodes v and z are terminals, c is a corner point, t is a T-point. cz is a complete line and the pair (cz, cv) is a complete corner. The topmost SMT is neither fulsome nor canonical (as defined in Section 2.2); the middle SMT is fulsome but not canonical, while the bottommost SMT is both fulsome and canonical.

2.2 Canonical Full Steiner Trees

One of the major difficulties when constructing algorithms for RSTP is that there in general exists an infinite number of SMTs for a given terminal set Z. One SMT may be transformed into another SMT by performing so-called *sliding* and *flipping* operations that do not change the length of the tree (Figure 2). In order to limit the number of SMTs to be considered we will give a particular characterization of SMTs that turns out to be very strong. Thus all SMTs that do not fulfill the properties of this characterization will be ignored.

Figure 2: Sliding and flipping operations.

A rectilinear Steiner tree in which every terminal is a leaf is denoted a *full Steiner tree (FST)*. Every SMT is a union of FSTs (see Figure 1). A *fulsome* SMT is an SMT in which the number of FSTs is maximized. In particular, no FST in a fulsome SMT can be split into two FSTs of the same total length. Alternatively, we may say that we maximize $\sum_{z \in Z} deg(z)$ where $deg(z)$ is the degree of terminal $z \in Z$ — this holds since the number of FSTs is $1 + \sum_{z \in Z}(deg(z) - 1)$; see Exercise 3. We shall use both views in the following.

Now consider an FST F in a fulsome SMT. The FST F is said to be *canonical* if no vertical segment s can be moved to the right using sliding and/or flipping operations (without increasing the length of F and without moving any other vertical segments of F; horizontal segments may be moved freely). If every FST in a fulsome SMT is canonical, then the SMT is canonical. It is clear that there exists a fulsome and canonical SMT: For every FST F, as long as a vertical segment can be moved to the right, then do so. Since every transformation moves some vertical segment further to the right, this process must stop; the final FST is therefore canonical.

One particular consequence of this definition is that for any corner point in a canonical FST, the incident vertical segment is completely to the right of the the incident horizontal segment (Figure 1).

2.3 Hwang-topology FSTs

Let F be an FST in a fulsome and canonical SMT. In this section we show that F has a very particular shape, denoted a *Hwang-topology*. The precise statement is given below in Theorem 2.3, but before we prove this theorem we give a crucial lemma that forms the cornerstone of the theorem.

Lemma 2.1 *Let uv be a segment in F where u and v are non-terminals. Then u and v cannot be incident to two segments perpendicular to uv and on the same side of uv.*

Proof. Suppose two such segments exist (Figure 3). Assuming that F is fulsome, we will prove that F cannot be canonical.

Let a be the endpoint of the complete line that contains the perpendicular segment incident to u (in the direction of the segment as seen from u); let a' be the other endpoint. Define b and b' analogously as the endpoints of the complete line that contains the segment incident to v (see Figure 3). Assume w.l.o.g. that $|ua| \leq |vb|$. Clearly a cannot be a terminal, since otherwise we could slide uv until hitting a, contradicting the fact that F is fulsome. Furthermore, since F is an SMT, a must be a corner point (again by sliding uv a third segment incident at a would overlap with uv). This means that there are no other nodes than u and a on the line from u to a: flipping the corner point a would prove that F could not be optimal if this was the case. In particular, this means that the segment s_u cannot exist. This again implies the the segment s_a *must* exist, since u cannot be a corner point.

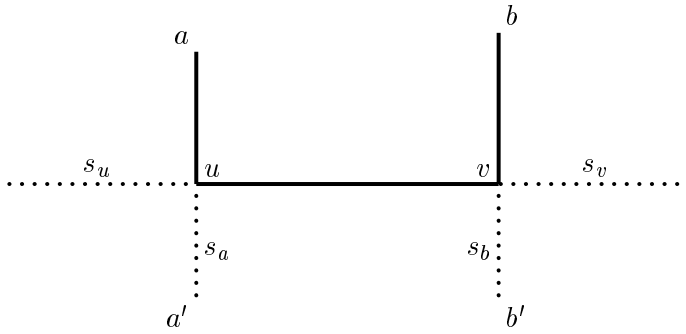


Figure 3: Lemma 2.1, proof illustration.

Now, v cannot be a corner point either, so s_v or s_b must exist. Assume that s_b exists. Then we can use the same arguments as above to prove that

either a' or b' must be a corner point. In fact, since a' cannot be a corner point (flipping this corner point and the corner point a would contradict the optimality of F), b' must be a corner point and thus s_v does not exist. We arrive at the situation depicted in Figure 4a.

On the other hand, if s_b does not exist, then s_v must exist and we arrive at the situation depicted in Figure 4b.

So far we have made no assumptions on the actual orientation of the segment uv. If uv in fact is horizontal, F is clearly not canonical, independent on the actual location of the first corner point a. In the first case (Figure 4a), either of the two corners will not be canonical. In the latter case (Figure 4b) we may slide uv vertically and again obtain two opposite corner points either of which is not canonical.

If uv is vertical, the only difficult case is illustrated in Figure 4c. Here we cannot slide uv horizontally to the right, but the corner point is nevertheless not canonical. □

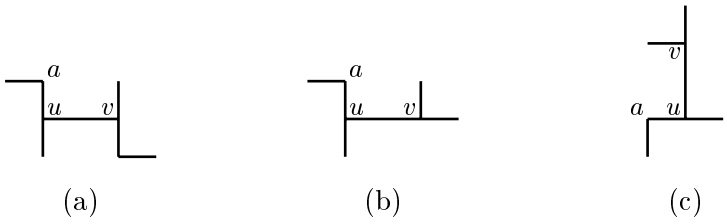


Figure 4: Lemma 2.1, different configurations. (a) segment s_b exists; (b) segment s_b does not exist; (c) segment uv is vertical.

Using this lemma as a workhorse, we can begin to give a more detailed characterization of F. First we assume that F has at least one corner point c. Consider the complete corner (defined in Section 2.1) located at c. Let cv be one of the legs of the complete corner and let $s_1, s_2, \ldots, s_l$ denote the sequence of (interior) Steiner points on cv in increasing distance from c. Lemma 2.1 now implies the following sequence of corollaries:

- The Steiner points $s_1, s_2, \ldots, s_l$ must be T-nodes; let v_i be the third node adjacent to Steiner point s_i, $i = 1, \ldots, l$ (i.e., not on the leg cv).

- Segment $s_i v_i$ is on the opposite side of cv as $s_{i+1} v_{i+1}$ for all $i = 1, \ldots, l-1$, that is, the incident segments *alternate* along the leg of the complete corner.

- Segment s_1v_1 is on the opposite side of cv as the second leg of the complete corner.
- All nodes v_i, $i = 1, \ldots, l$ must be *terminals*.
- The endpoint v of the leg cv must be a terminal. If v was a corner point, F would either not be optimal or not canonical, depending on the orientation of the supposed corner point. If v was a T-point, F would not be canonical since it would violate Lemma 2.1.

An analogous result is obviously obtained for the second leg; thus there are *no other* complete corners in F, and in particular no other corner point. We will now show that at most one leg of the complete corner can have more than one incident segment.

Lemma 2.2 *At most one of the legs of a complete corner in F has more than one incident segment.*

Proof. Assume that both legs have at least two incident segments, as shown in Figure 5. Consider the rectangle R given by the Steiner points s_2 and s_2'. At least one of the two terminals v_2 and v_2' is on the boundary of R; assume w.l.o.g. that v_2 is. Now we flip the corner point c and slide the segment $s_1's_2'$ as far as possible towards v_2' (see Figure 5). If we hit v_2' we have shown that F is not fulsome. Otherwise we subsequently slide s_1s_2 towards v_2 until we hit v_2, again contradicting the fact that F is fulsome. □

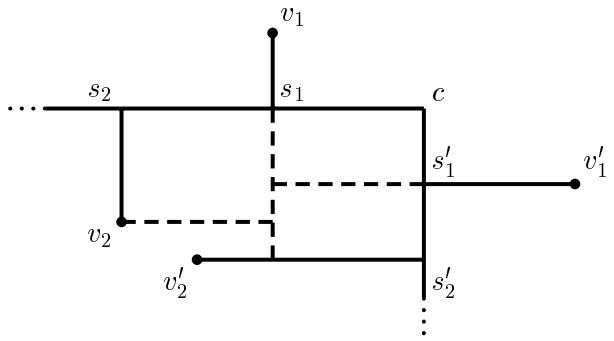


Figure 5: Lemma 2.2, proof illustration.

So far we assumed that F has at least one corner point. If F has no corner point, the situation is even more simple. Then F consists of a single complete line connecting two terminals; all other terminals are connected

to this line via alternating incident segments. The arguments are similar to those given for the corner point case, since the corner-free case is essentially a special case of the former in which one of the legs of the corner has zero length. However, one peculiar case arises: If F spans exactly four terminals, the complete line could have single Steiner point being a cross-point, that is, the FST consists of a cross-point to which the four terminals are connected. Note that this case happens since the complete line is not a leg of a complete corner; in the corner point case all Steiner points had to be T-points in order not to violate the conditions of Lemma 2.1. We arrive at the following important theorem:

Theorem 2.3 *[11] An FST in a fulsome and canonical SMT spanning k terminals consists of a complete corner (also denoted the* backbone*) given by a* root z_0 *and a* tip z_{k-1}*. The root is incident to the* long *leg and the tip incident to the* short *leg of the complete corner. There are two main types (i) and (ii) and two degenerate cases of type (i):*

- *Type (i) has $k-2$ alternating segments incident to the long leg and no segment incident to the short leg. The first degenerate case (i′) has a zero-length short leg, i.e., the complete corner is degenerated into a complete line. The second degenerate case (i″) is a cross-point interconnecting exactly four terminals.*
- *Type (ii) has $k-3$ alternating segments incident to the long leg and one segment incident to the short leg.*

Note that the terminology *short* leg and *long* leg is not meant to connote geometric length — rather, the long leg can have more incident segments than the short leg. The two types are illustrated in Figure 6, and the two degenerate type (i) cases are depicted in Figure 7.

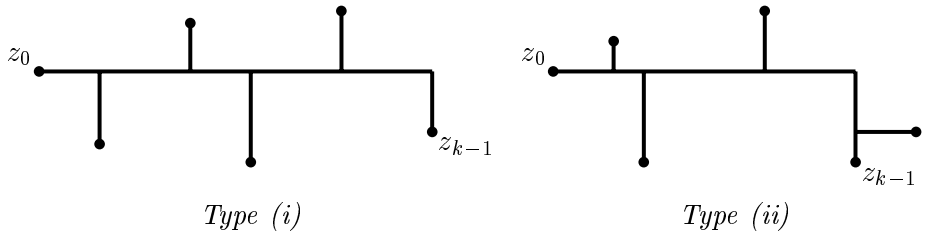


Figure 6: Hwang-topology FSTs.

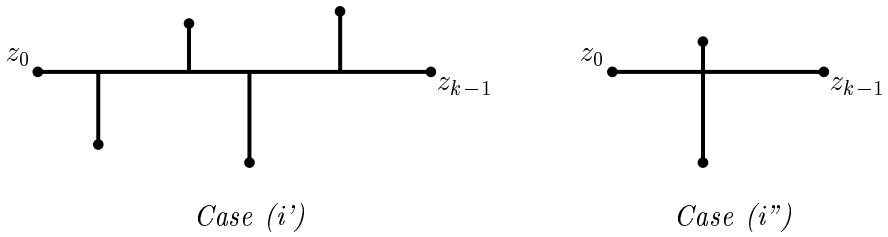


Figure 7: Degenerate cases of type (i) Hwang-topology FSTs.

As we shall see in the following two sections, this theorem has some nice theoretical consequences. However, it is also used in the design of practical algorithms for RSTP, as will be shown in Section 3 on exact algorithms for the problem.

Before moving on, we give yet another property of fulsome and canonical FSTs that will be used in Section 2.5.

Lemma 2.4 *Let F be a fulsome and canonical FST. If F is a type (i) FST, we let d_s denote the length of the short leg; otherwise, if F is a type (ii) FST, we let d_s denote the distance from the corner point to the Steiner point on the short leg. Let s be any segment incident to the long leg of F and on the same side of the long leg as the short leg. Then, $|s| > d_s$.*

Proof. Assume that there exists a segment s such that $|s| \leq d_s$. Then we may perform a sequence of flipping and sliding operations as shown in Figure 8 that split F into two FSTs, contradicting the fact that F is fulsome. □

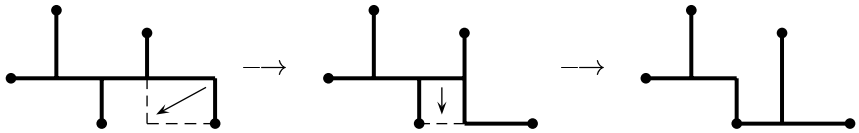


Figure 8: Lemma 2.4; sequence of flipping and sliding operations showing that a type (i) FST is not fulsome.

2.4 The Hanan Grid

The first paper solely devoted to RSTP was written by Hanan [8] in 1966. In addition to characterizing optimal solutions for small instances of the problem, Hanan gave the following fundamental structural result. Draw horizontal and vertical lines through every terminal in Z. Let $H(Z)$ denote the grid that is obtained, also called the *Hanan grid* for Z. Let $I_{H(Z)}$ be the set of $O(n^2)$ intersections in $H(Z)$, where $n = |Z|$ is the number of terminals (Figure 9); note that $Z \subseteq I_{H(Z)}$.

Theorem 2.5 *[8] There exists an SMT for Z such that every Steiner point belongs to $I_{H(Z)}$.*

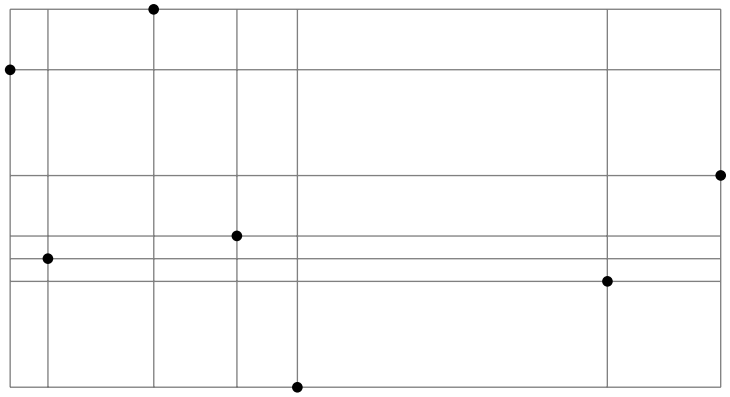


Figure 9: Hanan grid for the terminal set from Figure 1.

Alternatively, we may say that there exists an SMT for Z that is (geometrically) contained in the Hanan grid. The proof of Theorem 2.5 is a direct corollary of Theorem 2.3; see Exercise 4. One obvious consequence of this theorem is that we only need to consider a polynomial number of Steiner points candidates — namely the $O(n^2)$ intersection points in the Hanan grid. This means that there exist short certificates of optimal solutions, since we only need to consider Steiner point coordinates that are among the coordinates of the given terminals. Thus we have proven that RSTP is in NP; this is in major contrast with the Euclidean Steiner tree problem for which this question is still unsettled [12].

2.5 The Steiner Ratio

Consider interconnecting Z under the L_1 metric without being allowed to use Steiner points. This corresponds to computing a *rectilinear minimum spanning tree (MST)* for Z: Construct a minimum-length tree interconnecting Z in which only direct connections between terminals are allowed. Note that in the geometric embedding of such a tree, line segments may overlap.

Minimum spanning trees in edge-weighted graphs can be computed in polynomial time (essentially in linear time in the number of edges), but for the rectilinear problem an MST can be computed in $O(n \log n)$ time, even though the complete graph on the terminals has $O(n^2)$ edges (see Section 4.1). For a given terminal set Z, we let $|SMT(Z)|$ and $|MST(Z)|$ denote the length of an SMT and an MST for Z, respectively. Clearly $|SMT(Z)| \leq |MST(Z)|$ since an SMT is a shortest possible interconnection of Z, but the question is: How much shorter can an SMT be relative to an MST for the same set of terminals? Define

$$\rho_1(Z) = \frac{|SMT(Z)|}{|MST(Z)|}$$

to be the ratio between the length of an SMT and an MST for Z. The *Steiner ratio* ρ_1 for the L_1 metric in the plane is defined as

$$\rho_1 = \inf_Z \rho_1(Z)$$

That is, the Steiner ratio is the smallest possible ratio between SMT and MST length for any set of terminals. In the remaining part of this section we will prove the following theorem:

Theorem 2.6 *[11] The Steiner ratio for the rectilinear plane is*

$$\rho_1 = \frac{2}{3}$$

This may at first seem to be a purely theoretical exercise, but as will be shown in Section 4 on approximation algorithms for RSTP, this theorem gives us a firm bound on the quality of heuristics that are based on computing MSTs.

Before we start giving the proof, consider the set of terminals $Z_4 = \{(-1,0),(0,-1),(1,0),(0,1)\}$. $SMT(Z_4)$ is a cross of length 4. Since the length of $MST(Z_4)$ is 6, we have $\rho_1(Z_4) = 2/3$. Thus there does actually exist a terminal set for which the minimum ratio is achieved.

The proof of Theorem 2.6 only needs to be established for every possible FST, in particular only for Hwang-topology FSTs (that by definition are fulsome and canonical). To see why, consider an $SMT(Z)$ that is a union of FSTs $F_1, \ldots, F_m$. Assume that the Steiner ratio theorem holds for every FST F_i; consequently there exists an MST, denoted by MST_i, for the terminal set spanned by F_i such that $|MST_i| \leq 3/2|F_i|$. The union of all MSTs, denoted by T, is clearly a spanning tree for Z. Since

$$|MST(Z)| \leq |T| = \sum_{i=1}^{m} |MST_i| \leq \sum_{i=1}^{m} 3/2|F_i| = 3/2|SMT(Z)|$$

the theorem also holds for any — not necessarily full — SMT.

We will therefore focus our attention on an arbitrary Hwang-topology FST F spanning a set of terminals Z_F, and show that $|MST(Z_F)| \leq 3/2|F|$. Suppose F spans $k = |Z_F|$ terminals. Our proof will be by induction on k. The basis, $k \leq 4$, is left as Exercise 5.

First we assume that F is a type (i) FST. The root is denoted by z_0 and the alternating incident segments, in the direction from the root to the corner point, are denoted by $z_1 s_1, \ldots, z_{k-1} s_{k-1}$, where s_{k-1} is the corner point of F. It turns out to be useful also to consider the root as being connected to the long leg via Steiner point $s_0 = z_0$. Let $d_i = |z_i s_i|$ be the length of segment $z_i s_i$, $i = 0, \ldots, k-1$.

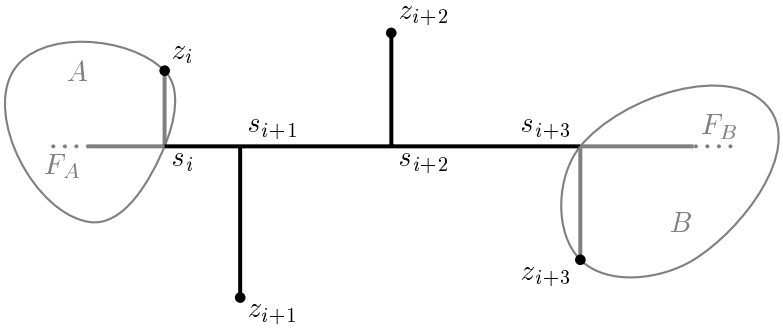


Figure 10: Theorem 2.6, proof illustration.

Below we will prove that there always exists an $i \in \{0, \ldots, k-4\}$ such that $d_i \leq d_{i+2}$ and $d_{i+1} \geq d_{i+3}$ (Figure 10). Let $A = \{z_0, \ldots, z_i\}$ and $B = \{z_{i+3}, \ldots, z_{k-1}\}$. Let F_A and F_B be the parts of F that interconnect A and B, respectively, and let F_C be the remaining part of F.

By the inductive hypothesis, $|MST(A)| \leq 3/2|SMT(A)| \leq 3/2|F_A|$ and $|MST(B)| \leq 3/2|SMT(B)| \leq 3/2|F_B|$.

Let $C = \{z_i, z_{i+1}, z_{i+2}, z_{i+3}\}$. Consider the boundary of the smallest axis-aligned rectangle that contains C. This boundary has length $2(|s_i s_{i+3}| + d_{i+1} + d_{i+2})$, and contains all terminals in C. Therefore, we can construct a tree interconnecting C that consists of terminal-terminal connections by deleting the longest connection between two terminals on the boundary. Thus we have

$$|MST(C)| \leq 3/2(|s_i s_{i+3}| + d_{i+1} + d_{i+2}) = 3/2|F_C|$$

In conclusion,

$$|MST(Z_F) \leq |MST(A)| + |MST(B)| + |MST(C)| \leq 3/2|F|$$

What remains to be shown is that there always exists an $i \in \{0, \ldots, k-4\}$ such that $d_i \leq d_{i+2}$ and $d_{i+1} \geq d_{i+3}$. Assume that this condition is not true for $i = 0$ (otherwise we are done). Consider d_4; the condition is fulfilled for $i = 1$ unless $d_4 > d_2$. Repeating this argument for all i, the only way the condition cannot be fulfilled is if the length of the incident segments on each side of the long leg are strictly increasing along the long leg. But this is in contradiction with Lemma 2.4 that says that the length of the short leg is *shorter* than all incident segments on the same side. This proves that there must exist a sequence of four terminals fulfilling the condition for a type (i) FST.

For a type (ii) FST all the arguments above can be repeated; the single terminal attached to the short leg will never be part of the set C. The only problem is that we have no bound on the length of the short leg. That is, we may arrive in the situation shown in Figure 11a, in which the above condition is not fulfilled for any i. However, in this case we may consider the corner-flipped FST instead (Figure 11b). In this FST the first four terminals on the long leg, corresponding to $i = 0$, will always fulfill the condition. This finishes our proof of the Steiner ratio theorem for the rectilinear plane.

3 Exact Algorithms

The NP-hardness of the rectilinear Steiner tree problem leaves little hope that any polynomial time exact algorithm exists for the problem. However, it turns out that fast and practical exact algorithms can be constructed for the problem. These algorithms are fast in the sense that realistic problem instances — in particular instances from VLSI design — can be solved

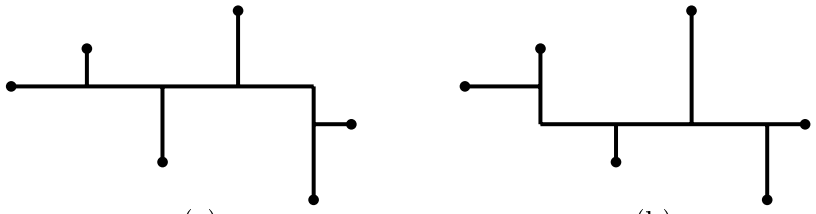


Figure 11: Type (ii) remaining case.

quickly in practice. The history of exact algorithms for RSTP is not long. In fact, it is fair to say that no substantial progress was made before 1990. In 1993 Salowe and Warme [17] submitted a paper describing an algorithm that could solve 30-terminal problems in less than one hour; in 1995 Hetzel [9] could solve 50-terminal problems within the same amount of time. The real breakthrough occurred a few years later when Warme [19] computed SMTs for problem instances with more than 1000 terminals.

In this section we first give some necessary optimality conditions for SMTs (Section 3.1). Then we describe the currently fastest exact algorithm for RSTP (Section 3.2) [19, 20, 24]. This algorithm uses Hwang's powerful characterization of fulsome and canonical FSTs (Theorem 2.3) as a starting point. Finally, in Section 3.3 we discuss solution methods that use the property that an SMT exists in Hanan grid for the set of terminals.

3.1 Necessary Optimality Conditions

An *edge* $e = (u, v)$ in an SMT is a direct connection between a pair of nodes u and v (which are either terminals or Steiner points). In a fulsome and canonical SMT an edge is either a single segment or a pair of perpendicular segments adjacent at a corner point. The length of an edge $e = (u, v)$, denoted by $|e|$, is the L_1 distance between u and v.

In this section we give some bounds on the length of edges in SMTs; also, we present some properties that particular configurations of edges must fulfill. Furthermore, note that any subtree of an SMT clearly must be an SMT for the nodes spanned; in particular this holds for FSTs. Tests based on this condition are usually denoted *upper bound tests*, and can be applied by computing heuristic trees that span the set of nodes in question.

In order to simplify the exposition, we consider $SMT(Z)$ and $MST(Z)$ as being unique. It is easy to see that all optimality conditions given will be

valid for any $SMT(Z)$ and $MST(Z)$.

Bottleneck Steiner Distances

Assume $z_i, z_j \in Z$ is a pair of distinct terminals and let $P_T(z_i, z_j)$ denote the unique path between z_i and z_j in a tree T. The path consists of one or more edges connecting the nodes.

Consider the paths $P_{SMT(Z)}(z_i, z_j)$ and $P_{MST(Z)}(z_i, z_j)$. Note that the latter can easily be computed. Pick an edge $e \in P_{SMT(Z)}(z_i, z_j)$ and remove it from $SMT(Z)$. This breaks the tree into two connected components that contain each of the terminals z_i and z_j, respectively. Now follow the path $P_{MST(Z)}(z_i, z_j)$ which only consists of edges connecting terminals. One of the edges on this path, say $f = (z_k, z_l)$, will reconnect the two components of the broken SMT. Clearly, we must have that $|e| \leq |f|$ since otherwise we would have shown that $SMT(Z)$ was not a shortest tree.

This observation leads to the following definition. The *bottleneck Steiner distance*, $b_{z_i z_j}$, between a pair of terminals z_i and z_j is equal to the length of the longest edge on $P_{MST(Z)}(z_i, z_j)$. Note that there exists no terminal-path between z_i and z_j for which the longest edge is smaller than $b_{z_i z_j}$ (see Exercise 6).

Lemma 3.1 *For any edge $e \in P_{SMT(Z)}(z_i, z_j)$, we have $|e| \leq b_{z_i z_j}$.*

Bottleneck Steiner distances between every pair of terminals can be determined in $O(n^2)$ time by computing $MST(Z)$ and doing a depth-first traversal in this tree from every terminal. The optimality condition provided by Lemma 3.1 turns out to be very powerful in practice, and can be supplemented by a generalization given in Exercise 7.

Empty Regions

In the previous section we gave an upper bound on the length of edges connecting a pair of terminals. In this section we give some conditions that depend on how *close* other terminals are to an edge or a pair of edges. Let (u, v) be an edge in $SMT(Z)$. Consider the region

$$\mathcal{L}(u, v) = \{p \in \mathbb{R}^2 \ : \ |pu| < |uv| \ \wedge \ |pv| < |uv|\}$$

also denoted the "lune" given by (u, v) (Figure 12a). The lune is the intersection between the interior of two L_1 circles with radius $|uv|$ centered at u and v, respectively.

Lemma 3.2 *If (u, v) is an edge in $SMT(Z)$, then $\mathcal{L}(u, v)$ contains* no *other point (terminal, Steiner point, or interior segment point) from $SMT(Z)$.*

Proof. Assume on the contrary that there exists a point $p \in \mathcal{L}(u, v)$. Remove edge (u, v) from $SMT(Z)$, splitting the tree into two connected components. The point p belongs to one of the two components, say the one that contains u. By adding the edge (p, v) we have constructed a shorter tree interconnecting the terminals. If p belongs to the other component we would also be able to construct a shorter tree, a contradiction. □

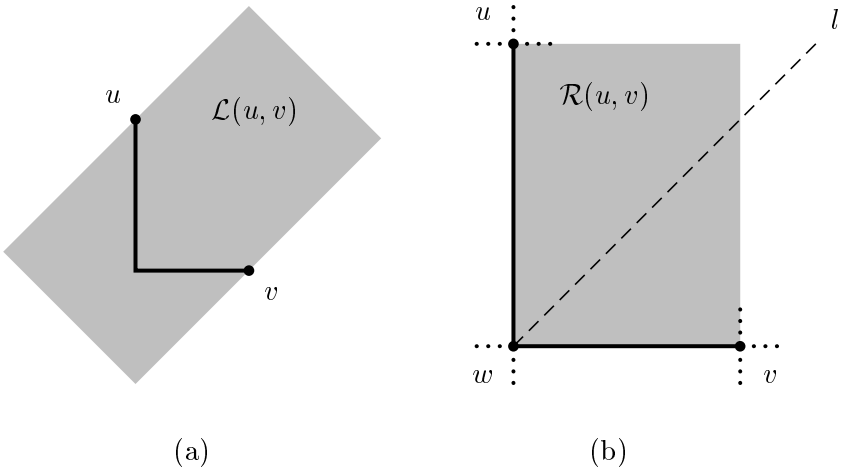


Figure 12: Empty regions. (a) empty lune; (b) empty corner rectangle. Gray-shaded areas cannot contain a point of $SMT(Z)$.

Now, assume that the nodes u and v are not connected directly via an edge, but through a third node w such that the segments uw and wv are perpendicular (Figure 12b). Let $\mathcal{R}(u, v)$ be the interior of the axis-aligned rectangle with sides uw and wv; note that $\mathcal{R}(u, v) \subset \mathcal{L}(u, v)$.

Lemma 3.3 *If uw and wv are perpendicular segments in $SMT(Z)$, then $\mathcal{R}(u, v)$ contains no other point of $SMT(Z)$.*

Proof. Assume on the contrary that $SMT(Z)$ contains a point $p \in \mathcal{R}(u, v)$. Let l be the line through w which bisects the perpendicular angle, and assume that p is above l (in Figure 12b). Remove uw from $SMT(Z)$. If p

belongs to the same component as u then add a vertical segment from p down to segment wv, otherwise reconnect by connecting u and p. In both cases the tree is shortened, a contradiction. If p is below l a similar arguments shows that the tree can be shortened is this case, too. Finally, assume that p is exactly on the line l. Since $SMT(Z)$ consists of vertical and horizontal segments, there must exist another point $p' \in \mathcal{R}(u, v)$ that is either above or below l, again allowing us to shorten the tree. □

The optimality condition given in Lemma 3.3, denoted the *empty corner rectangle property*, has been used with great success in the design of both exact and heuristic methods for RSTP [2, 15, 24].

3.2 FST Based Exact Algorithms

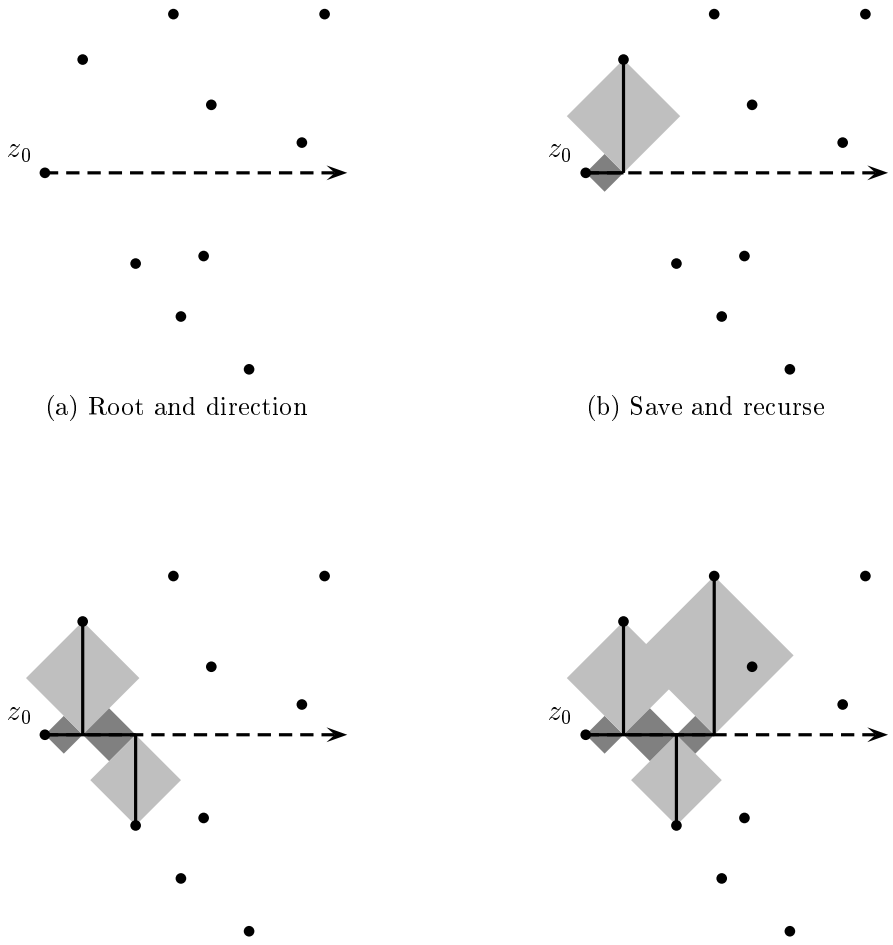
In this section we give a description of the currently most efficient method for solving RSTP to optimality. This algorithm uses an overall approach that was suggested by Winter [22] for the Euclidean Steiner tree problem in the plane. We will use the fact that there exists an SMT which is a union of FSTs having Hwang-topology (Theorem 2.3).

The idea is simply to generate *all* Hwang-topology FSTs that fulfill certain necessary optimality conditions, in particular those given in the previous section. This may a first seem to be an hopelessly inefficient approach since we (in principle) have to consider all $O(2^n)$ subsets of terminals; however, most subsets are only considered implicitly and very few (i.e., approximately linear in the number of terminals) FSTs survive all the conditions in practice. After this first *FST generation* phase we need to select a subset of the generated FSTs that interconnect all terminals and have minimum total length. This second phase is called *FST concatenation*, and it turns out to be the computationally hardest task of the two phases.

FST Generation

Assume that some terminal $z_0 \in Z$ is the root of a Hwang-topology FST (see Theorem 2.3). The long leg has one of four possible directions: North, East, South or West. Let us consider a specific direction, say East. This situation is shown in Figure 13a. Let us (informally) describe a procedure for generating all FSTs having root z_0 and direction East.

Sort all terminals to the right of the vertical line through z_0 by their x-coordinate. Let Z_a be the list of sorted terminals that are above the horizontal line through z_0 and let Z_b be the corresponding list of terminals



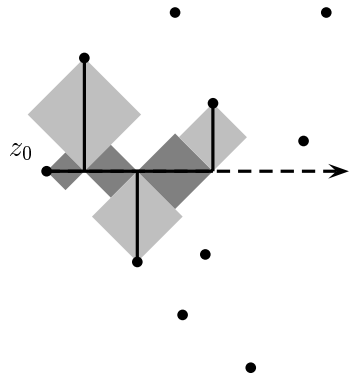
(a) Root and direction

(b) Save and recurse

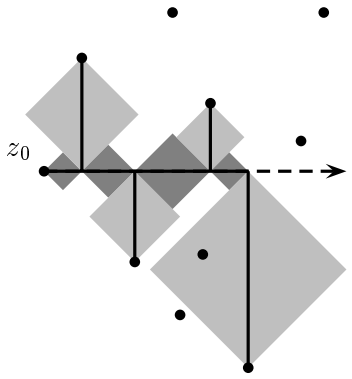
(c) Save and recurse

(d) Skip and continue

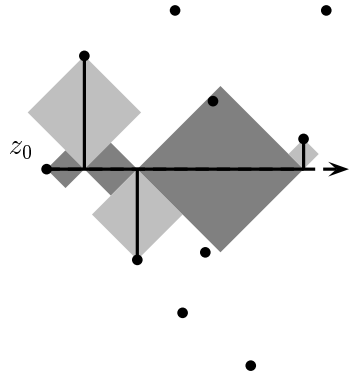
Figure 13: FST generation algorithm example.



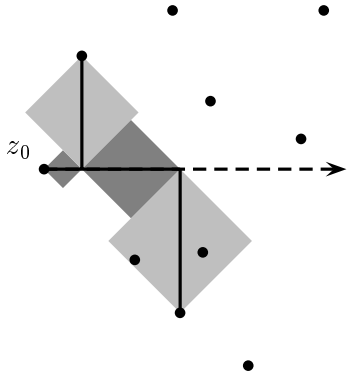
(e) Save and recurse



(f) Backtrack



(g) Backtrack



(h) Skip and continue (etc.)

Figure 13: (cont.)

below this line. Consider the first terminal in Z_a and connect it to the root as shown in Figure 13b, that is, create a segment along the long leg and another one connecting the terminal to the long leg. Now we may test whether this partial FST can be a subtree in some possibly larger FST. This is done by applying several necessary optimality conditions, including those given in Section 3.1.

In the example in Figure 13 we only show the effect of applying the empty lune condition (Lemma 3.2). Since both lunes in Figure 13b are empty we save this partial FST and continue growing this FST. This is done by choosing the next terminal from Z_b (Figure 13c); recall that the terminals must alternate along the long leg. Again all necessary optimality conditions are fulfilled and we recurse (Figure 13d). In this case a non-empty lune appears; this means that this partial FST cannot be a subtree in some larger FST. Therefore, we skip this terminal and choose the next candidate from Z_a (Figure 13e). In Figure 13f we again get a non-empty lune and since there are no more candidates in Z_b, we backtrack, i.e., choose another candidate for the previous terminal (Figure 13g). Here we again need to backtrack — and the FST generation algorithm continues until all FSTs having z_0 as root and long leg direction East are generated (note that we also need to consider the case where the first terminal is chosen from Z_b). Finally, this algorithm is repeated for all combinations of terminals and directions.

As described, this procedure only generates type (i) FSTs, but type (ii) FSTs can be generated simultaneously: Here we need to try all possibilities of attaching a single terminal to the last vertical segment.

An FST-independent *preprocessing* phase which runs in $O(n^2)$ time can be used to speed up this FST growing algorithm significantly in practice [24]. In fact, for most problem instances the preprocessing dominates the total running time even if the second part is the one that requires exponential time in the worst-case. A well-tuned implementation of this algorithm [21] generates the FSTs for a randomly generated 1000 terminal instance in less than one second; the number of FSTs surviving all tests is approximately $4n$. This set of FSTs includes $n-1$ edges from an MST for Z, which may be considered as the 2-terminal FSTs (Exercise 8 discusses why an arbitrary MST for Z can be used).

FST Concatenation — Spanning Trees in Hypergraphs

Let $H = (V, E)$ be a *hypergraph* with the set of terminals as its vertices and the set of generated FSTs as its hyperedges. Each hyperedge $e \in E$ is

a set of vertices of cardinality $|e| \geq 2$, which corresponds to the terminals spanned by the FST. An hyperedge that spans k vertices is denoted a k-edge. Hyperedge $e \in E$ has a weight c_e that is equal to the geometric length of the corresponding FST.

A *chain* in H from $v_0 \in V$ to $v_k \in V$ is an alternating sequence of vertices and hyperedges, $v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k$, such that all vertices and hyperedges are distinct and $v_{i-1}, v_i \in e_i$ for $i = 1, 2, \ldots, k$. A spanning tree in H is is a subset of hyperedges $E' \subseteq E$ such that there is a *unique* chain between every pair of vertices $v_i, v_j \in V$ in the induced subgraph (Figure 14). The uniqueness implies that there can be no pair of distinct hyperedges $e_i, e_j \in E'$ that share two or more vertices, i.e., we have $|e_i \cap e_j| \leq 1$ for all $e_i, e_j \in E'$. The problem of finding a minimum spanning tree (MST) in H — where each hyperedge $e \in E$ has weight c_e — is equivalent to solving the FST concatenation problem.

The MST in hypergraph problem (MSTHG) is NP-hard when the hypegraph contains edges of cardinality four or more [19]. Actually, deciding the existence of a spanning tree in such a hypergraph is NP-complete. A number of methods for solving this problem have been suggested (see [20] for a survey). Warme [19] gave an integer programming formulation that was solved using branch-and-cut. This is currently the fastest solution method in practice and we will therefore give a description of the main components of the algorithm here.

We solve MSTHG by setting up an integer programming (IP) formulation. Denote by x an $|E|$-dimensional *binary* vector; each element x_e has value 1 if the edge $e \in E$ is chosen to be part of the MST and 0 otherwise. The IP formulation is then

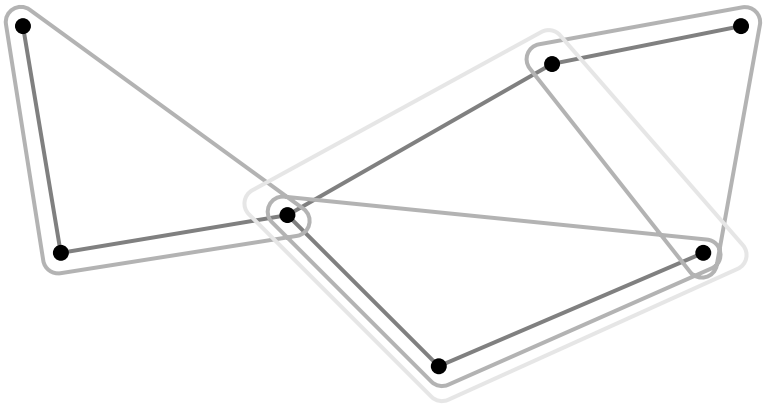
$$\min cx \tag{1}$$

$$\text{s.t.} \quad \sum_{e \in E} (|e| - 1) x_e = |V| - 1 \tag{2}$$

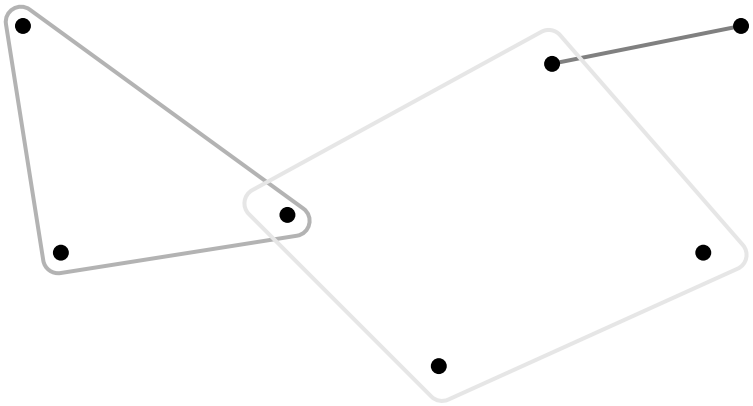
$$\sum_{e \in E : |e \cap S| \geq 1} (|e \cap S| - 1) x_e \leq |S| - 1, \quad \forall S \subset V,\ |S| \geq 2 \tag{3}$$

This formulation requires some explanation. We only give informal arguments showing that this formulation solves MSTHG; a formal proof can be found in [19].

The objective (1) is to minimize the total length of the chosen hyperedges subject to two types of constraints: Firstly, equation (2) enforces the correct number and cardinality of hyperedges to construct a spanning tree. The intuition behind this equation is that the number of 2-edges in a spanning



(a)



(b)

Figure 14: Spanning tree in hypergraph example. (a) hypergraph; (b) spanning tree.

tree of an ordinary graph with $|V|$ vertices is exactly $|V| - 1$, otherwise the tree is either not connected or contains a cycle. Since every hyperedge $e \in E$ can be seen as consisting of $|e|-1$ 2-edges (that is, a local tree interconnecting the vertices spanned by the hyperedge), the equation follows.

Secondly, constraints (3) eliminate cycles by extending the standard notion of subtour elimination constraints. For a given subset $S \subset V$, the total number of 2-edges contained in the subset (again viewing hyperedges as a set of 2-edges), is at most $|S| - 1$, otherwise a cycle is created (Figure 15). Every edge $e \in E$ which intersects S contributes with $|e \cap S| - 1$ 2-edges. The intersection property is equivalent to $|e \cap S| \geq 1$, but in fact only edges for which $|e \cap S| \geq 2$ contribute to the sum. Using the first condition, however, has some advantages as will be seen in the following.

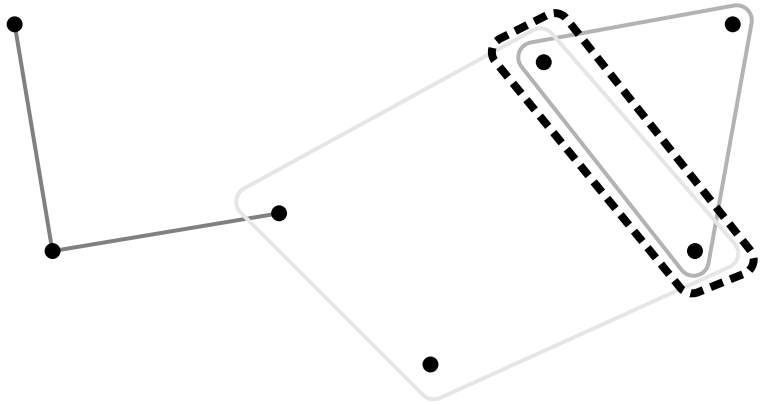


Figure 15: Violation of a cycle elimination constraint. The dotted closed curve defines a set S for which the number of internal edges is greater than $|S| - 1$.

This integer program is solved via branch-and-cut. We give the main details here, but in order to solve this problem quickly in practice a number of additional techniques must be applied [19]. The implemented branch-and-cut algorithm [21] solves a typical 100 terminal problem in a few seconds.

Lower bounds for the IP formulation are provided by linear programming (LP) relaxation, i.e., by relaxing integrality of every component x_e of x to $x_e \geq 0$. The major obstacle in solving this LP is the exponential number of constrants given by (3). Therefore, the LP is solved using an iterative method. First a subset of the constraints (3), more precisely those for which $|S| = 2$, are included. Then the LP is solved, returning a (fractional)

solution x. Now we would like to see if x fulfills all the cycle elimination constraints; if not, we should add at least one of these constraints to the LP and iterate. The process of identifying violated constraints is denoted *separation*. It turns out that separation can be done in polynomial time by solving a series of max-flow problems in an appropriately defined auxiliary graph.

Before we give the separation algorithm, a few definitions are needed. The *congestion level* of a vertex $v \in V$ for an LP-solution x is:

$$b_v = \sum_{e \in E : v \in e} x_e$$

This is the total (fractional) amount of edges that have v as one of their vertices. Clearly, in an MST we must have $b_v \geq 1$, since otherwise the vertex is not included in the solution. For a subset $S \subseteq V$ we have:

$$\sum_{v \in S} b_v = \sum_{e \in E : |e \cap S| \geq 1} |e \cap S| x_e$$

Furthermore, define the function

$$\begin{aligned} f(S) &= |S| - \sum_{e \in E : |e \cap S| \geq 1} (|e \cap S| - 1) x_e \\ &= |S| - (\sum_{v \in S} b_v - \sum_{e \in E : |e \cap S| \geq 1} x_e) \end{aligned}$$

The separation problem is equivalent to finding an $S \subseteq V$ such that $S \neq \emptyset$ and $f(S) < 1$ or proving that no such S exists. This can be achieved by minimizing this function over all $S \neq \emptyset$.

Define a flow network $G_x = (N, A)$ for an LP-solution x as follows. The vertex set is $N = \{s\} \cup V \cup E \cup \{t\}$, where V and E are the vertices and edges in H, respectively. The vertex s is the source and the vertex t the sink. The arc set is defined as $A = A_s \cup A_\infty \cup A_t$, where

$$\begin{aligned} A_s &= \{(s, v) : v \in V\} \\ A_\infty &= \{(v, e) : e \in E, v \in e\} \\ A_t &= \{(e, t) : e \in E\} \end{aligned}$$

Arc $(s, v) \in A_s$ has capacity $b_v - 1$, arc $(e, t) \in A_t$ capacity x_e while all arcs in A_∞ have infinite capacity (Figure 16). Let $W = \{s\} \cup S \cup F$ — where $S \subseteq V$ and $F \subseteq E$ — be a minimum weight $s - t$ cut in G_x. That

is, $t \notin W$ and the total weight of the edges crossing from W to $N \setminus W$ is minimized. Note that no edges in A_∞ can be part of a minimum weight cut, since there exists a cut of finite weight: simply let $S = F = \emptyset$. Now we have the following lemma:

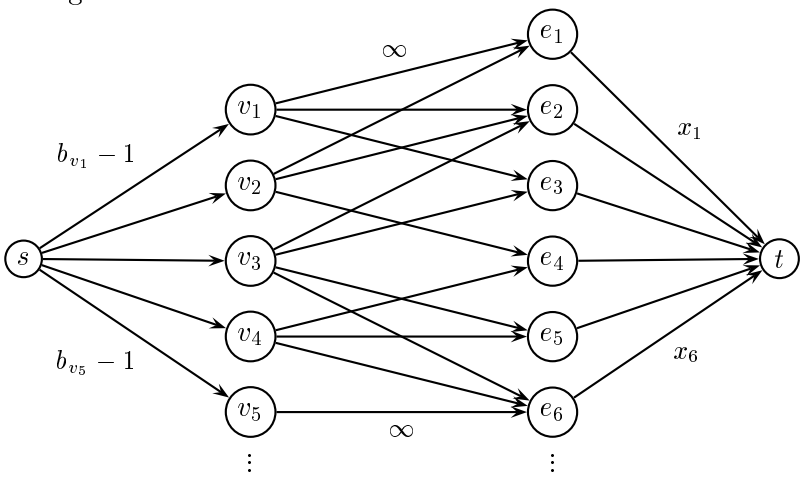


Figure 16: Flow network for solving the separation problem.

Lemma 3.4 *Let $W = \{s\} \cup S \cup F$ be a minimum weight $s - t$ cut in G_x. Then S is a minimum of $f(S)$.*

Proof. If $v \in S$ then all $e \in E$ for which $v \in e$ must be in F, since otherwise the cut has infinite weight. Now assume that $e \in F$ but $e \cap S = \emptyset$. Then moving e to the other side of the cut can only decrease the weight of the cut. Therefore, we may assume that F is completely determined by S. The value of the cut is therefore:

$$\begin{aligned} & \sum_{e:|e\cap S|\geq 1} x_e \quad + \quad \sum_{v\in V\setminus S} (b_v - 1) \\ = & \sum_{e:|e\cap S|\geq 1} x_e \quad + \quad \sum_{v\in V\setminus S} b_v - (|V| - |S|) \end{aligned}$$

By adding the constant $|V|$, subtracting the constant $\sum_{v\in V} b_v$ and rearranging, we obtain:

$$\begin{aligned} |S| \quad - \quad & (\sum_{v\in V} b_v - \sum_{v\in V\setminus S} b_v - \sum_{e:|e\cap S|\geq 1} x_e) \\ = \; |S| \quad - \quad & (\sum_{v\in S} b_v - \sum_{e:|e\cap S|\geq 1} x_e) \end{aligned}$$

Therefore, there is a constant difference between the value of the cut and $f(S)$; this proves the lemma. □

Minimizing $f(S)$ under the additional condition $S \neq \emptyset$ is achieved as follows. First we pick a terminal $v_1 \in V$. Then we set up a reduced flow network in which v_1 and the hyperedges that contain v_1 have been deleted. The reduced flow problem is solved, thus obtaining a set of vertices S' (possibly empty). The set $S' \cup \{v_1\}$ is clearly a minimum over $f(S)$ under the condition that $v_1 \in S$. Then we remove a second terminal $v_2 \in V$ (and the hyperedges that contain it from the flow network) and iterate.

3.3 Hanan Grid Based Exact Algorithms

The first exact algorithms for RSTP were based on the classical result that there exists an SMT in the Hanan grid $H(Z)$ for the set of terminals (Section 2.4). Consider the ordinary graph $G = (V, E)$ representing the Hanan grid. The intersections in $H(Z)$ are the vertices in G while the segments interconnecting neighbouring intersections form the edges of G. Let c_e denote the geometric length of the horizontal or vertical segment that edge $e \in E$ represents. G is a planar graph in which every vertex has maximum degree four.

Hanan's theorem shows that RSTP reduces to the *Steiner tree problem in graphs (STPG)*: Given an edge-weighted graph $G = (V, E)$ and a set of terminals $Z \subseteq V$, find a tree in G that interconnects Z and has minimum total length. Numerous exact and heuristic algorithms have be suggested for this well-studied problem [12], all of which (in principle) can be used to solve the corresponding rectilinear Steiner tree problem.

In this section we give a short introduction to the best exact algorithm for STPG [14]. This algorithm uses the so-called directed IP formulation for STPG; other formulations and their relations to each other are described in [6]. But before presenting the directed IP formulation, we discuss some algorithms that can be used for reducing the Hanan grid before applying algorithms for STPG.

Hanan Grid Graph Reductions

A straightforward and fast method for reducing the Hanan grid is to generate Hwang-topology FSTs (Section 3.2). That is, take the set of generated FSTs and place them on the Hanan grid. Note that every Hwang-topology FST is contained in the Hanan grid. Edges and Steiner points in the Hanan grid

which are not used by any FST can clearly be deleted. This is in practice the most efficient way to reduce the Hanan grid, e.g., for $n = 1000$ there are in the worst-case $n^2 = 1000000$ vertices in the complete Hanan grid but only around 0.4% of these are retained after FST generation [24].

The disadvantage of using FST generation is that there is no guarantee of polynomial running time. On the other hand, general graph reduction techniques for the STPG are known to perform very poorly on the Hanan grid graph [14]. Therefore, Winter [23] proposed several reduction techniques that take advantage of the special structure of the Hanan grid graph, in particular that vertices have low degree and that many edges have the same length. Uchoa, Poggi de Aragão and Ribeiro [18] extended the ideas of Winter to reducing Hanan grid graphs with holes; these graphs occur frequently in VLSI routing [9].

Directed IP Formulation for STPG

We solve STPG for a graph $G = (V, E)$ with non-negative edge weights c_e, $e \in E$, and terminal set $Z \subseteq V$ by setting up the following IP formulation. First we create a *directed* graph $\bar{G} = (V, \bar{E})$ having the same set of vertices as G. For every edge $(u, v) \in E$ there are two directed edges $[u, v] \in \bar{E}$ and $[v, u] \in \bar{E}$, both having the same cost as $(u, v) \in E$: $c_{[u,v]} = c_{[v,u]} = c_{(u,v)}$. Let an arbitrary terminal $r \in Z$ be designated as the *root*. Solving STPG is now equivalent to finding a rooted tree of minimum total length in $\bar{G}$ (with r as root) that contains all terminals in Z. Such a tree is called a *Steiner arborescence*.

Denote by x an $|\bar{E}|$-dimensional binary vector; each element $x_{\bar{e}}$ has value 1 if the edge $\bar{e} \in \bar{E}$ is chosen to be part of the Steiner arborescence and 0 otherwise. For any non-empty set $S \subset V$ let

$$\delta(S) = \{[u, v] \in \bar{E} \ : \ u \in S \ \wedge \ v \in V \setminus S\}$$

be the set of edges leaving from S and ending in $V \setminus S$. The IP formulation is then

$$\min cx \tag{4}$$

$$\text{s.t.} \quad \sum_{\bar{e} \in \delta S} x_{\bar{e}} \geq 1, \quad \forall S \subset V, \ r \in S, \ (V \setminus S) \cap Z \neq \emptyset \tag{5}$$

The constraints (5) ensure that there is a path from the root to every terminal: Any cut separating the root and a terminal must have at least one edge crossing the cut.

Koch and Martin [14] gave a branch-and-cut algorithm for solving STPG using this IP formulation. Since the number of constraints is exponential a separation method is used to iteratively add constraints to the linear program (see also Section 3.2). The separation problem is solved by finding a maximum flow from the root r to every terminal $t \in Z \setminus \{r\}$ with capacities identical to the fractional solution values of the edges. If the value of the flow is less than 1, the corresponding minimum cut gives us a violated constraint that can be added to the formulation. Although the results presented in [14] are good for several classes of graphs, the algorithm has serious problems solving Hanan grid graph problems; RSTP instances with 40 or more terminals are very hard to solve using the complete Hanan grid graph.

4 Approximation Algorithms

The need for solving RSTP in the VLSI design domain — and the lack of fast exact algorithms until a few years ago — has spawned a constant flow of heuristic algorithms for the problem. Almost all of these use the rectilinear minimum spanning tree (MST) as a starting point. Since the Steiner ratio theorem (Section 2.5) tells us that

$$\frac{|MST(Z)|}{|SMT(Z)|} \leq \frac{3}{2}$$

for any set of terminals Z, we know that an MST is at most 50% longer than an SMT for the same set of terminals. We say that an MST algorithm has an *approximation ratio* of 3/2. Polynomial time algorithms that have a (known) approximation ratio are usually denoted *approximation algorithms*.

A comprehensive survey of heuristics for RSTP was given in [12]. In this tutorial we only describe a few of these (and more recent) approaches without going into the same level of detail as in the preceding sections. Firstly, simple and fast algorithms that take an MST and compute short embeddings of the MST in the plane are presented in Section 4.1. These algorithms find good ways to draw the MST in the plane such that the length of all segments (only counting overlapping segments once) is as short as possible. Secondly, we give a short introduction to the 1-Steiner heuristic which is currently among the best performing heuristics for RSTP (Section 4.2).

None of these heuristics provide an approximation ratio that is strictly less than 3/2. For a long time it remained an open problem to design a heuristic with a strictly better approximation ratio. Zelikovsky [25] first

broke the 3/2 barrier by giving an algorithm with a 11/8 approximation ratio. However, Arora [1] finished the search for algorithms with better constant approximation ratios when he gave a polynomial time approximation scheme (PTAS) for RSTP: For any fixed $\epsilon > 0$ there exists a polynomial time algorithm that has an approximation ratio of $1 + \epsilon$. In Section 4.3 we present the main algorithmic ideas leading to this amazing result.

4.1 MST Embeddings

A rectilinear MST can be computed in $O(n \log n)$ time by using, e.g., rectilinear Voronoi diagrams [12], but this is rather involved both in theory and practice. Here we sketch a simpler algorithm for computing an MST in $O(n \log n)$ time.

Consider a terminal $z \in Z$. Draw the two $\pm 45^\circ$ lines passing through z. This divides the plane into 8 regions (not including the point z): The four half-lines extending from z and the interior of the four regions formed by the lines. Now we have the following lemma whose proof is left as Exercise 9:

Lemma 4.1 *[10] In $MST(Z)$ any terminal $z \in Z$ has at most one neighbour in each of the 8 regions; furthermore, this terminal will be a closest neighbour to z in its region.*

Using this lemma, we compute an MST as follows. For each terminal $z \in Z$, find a nearest neighbour in each of the 8 regions. This can be done for *all* terminals in $O(n \log n)$ time [7]. Construct a graph (having Z as it vertices) in which every terminal $z \in Z$ is connected to the (at most) 8 nearest regional neighbours. The resulting graph has $O(n)$ vertices and edges; therefore, an MST can be computed in $O(n \log n)$ time using, e.g., Kruskal's algorithm.

Consider the MST shown in Figure 17. Each MST edge is a shortest path under the L_1-metric between its endpoints. Assume that we choose a particular drawing (or embedding) of the edges. Line segments from different edges may overlap, but we only need to keep one copy of overlapping line segments in order to get a tree that interconnects Z. An *optimal embedding* is a drawing for which the resulting heuristic tree, obtained by merging overlapping line segments, is as short as possible.

In order to speed up the construction of good MST embeddings, it turns out to be beneficial to start with a *separable* MST. This is an MST for which the bounding boxes of the edges only overlap if the corresponding edges share a terminal (this is the case for the MST shown in Figure 17).

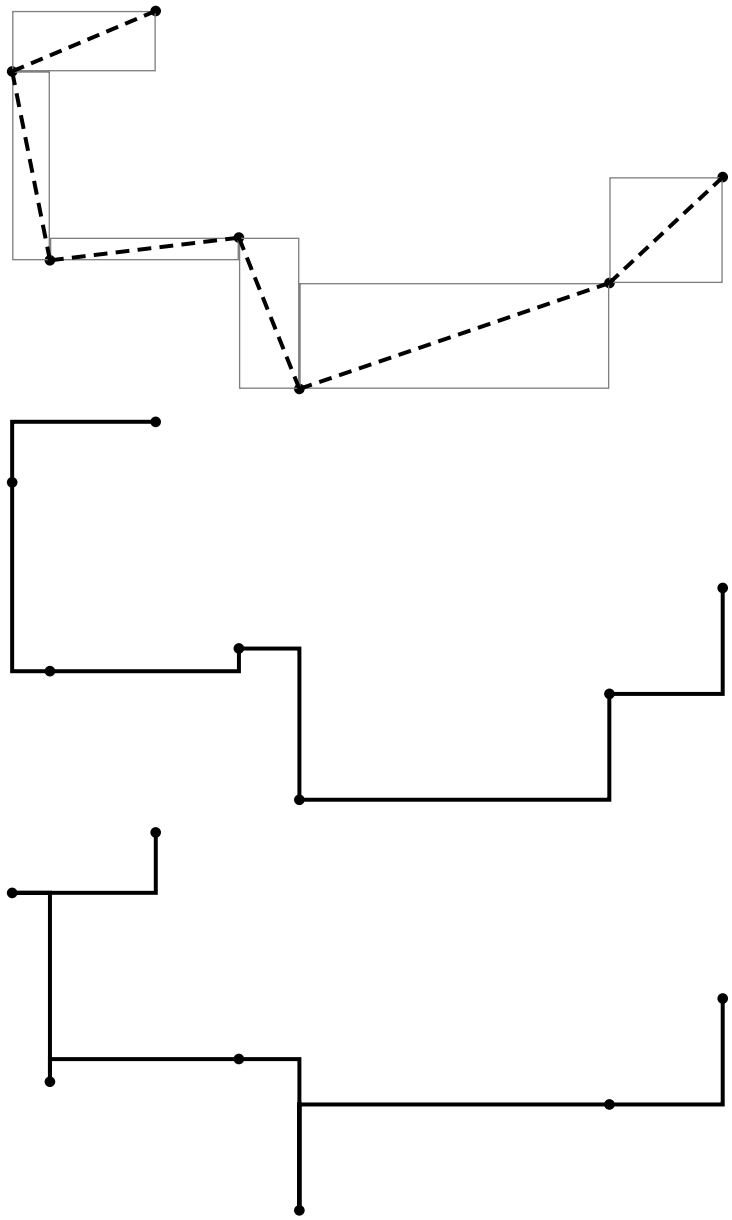


Figure 17: Rectilinear MST for the terminal set from Figure 1. In the topmost figure each MST edge is drawn using dashed lines. Any staircase connection between the terminals can be chosen; overlapping bounding boxes indicate that the corresponding embeddings may overlap. In the embedding given by the middle figure no line segments overlap; the bottommost figure is an optimal embedding (in this case also an SMT for the terminal set).

Ho, Vijayan and Wong [10] proved that a separable MST always exists and that it can be constructed in $O(n \log n)$ time.

The simplest possible drawing of an MST is an *L-shaped* embedding: Every edge has at most one corner point (it bends at most once). Clearly, there are at most two ways to draw an edge in an L-shaped embedding. Note that the embeddings shown in Figure 17 are L-shaped. Starting with a separable MST, an optimal L-shaped embedding can be constructed in $O(n)$ time [10]. The algorithm uses dynamic programming by rooting the MST and constructing an optimal embedding bottom up in the tree. The running time follows from the fact that the maximum degree in an MST is bounded by a constant (Lemma 4.1).

Let us now consider the case where at most two bends or so-called *Z-shaped* drawings are allowed. Interestingly, it turns out that an optimal Z-shaped embedding also is optimal among *all* possible embeddings. Furthermore, an optimal L-shaped embedding can be constructed in polynomial time [10]. When the running time is taken into account, optimal MST embeddings — and in particular optimal L-shaped embeddings — produce fairly good heuristic solutions in practice.

4.2 1-Steiner Heuristics

The problem of computing an SMT for a terminal set Z can be formulated as follows. Find a set of Steiner points S such that $MST(Z \cup S)$ has minimum length. There are two facts which make this a good starting point for designing a heuristic for RSTP. Firstly, the number of Steiner points is at most $n - 2$ (see Exercise 1), and secondly, we clearly only need to consider the Steiner points in the Hanan grid, $H(Z)$, as candidates to be included in the set S.

A greedy approach is therefore to start with $S = \emptyset$ and iteratively add Steiner points to S such that the length of the MST is minimized in every iteration. This is the basic *iterated 1-Steiner* heuristic for RSTP:

1. $S = \emptyset$
2. Find a point $s \in H(Z)$ such that $|MST(Z \cup S \cup \{s\})|$ is minimized
3. if $|MST(Z \cup S \cup \{s\})| \geq |MST(Z \cup S)|$ then **stop**
4. $S = S \cup \{s\}$; remove points in S having degree ≤ 2 in $MST(Z \cup S)$
5. Goto 2

In step 2 we find a Steiner point that gives a maximum decrease in MST length. There are $O(n^2)$ Steiner points in the Hanan grid and each MST computation takes $O(n \log n)$ time, so a trivial implementation of step 2 takes $O(n^3 \log n)$ time. By using a more sophisticated algorithm, step 2 can be performed in $O(n^2)$ time [13].

The algorithm stops when no improving Steiner point can be found (step 3). In step 4 we remove Steiner points that have degree two or less in the MST since the inclusion of these Steiner points does not decrease the length of the MST. The number of iterations can be greater than $n-2$ (see Figure 18), but is finite when the Steiner points are chosen from the Hanan grid. In practical implementations at most n iterations are made, giving a total running time of $O(n^3)$ using the improved search for a best Steiner point in step 2.

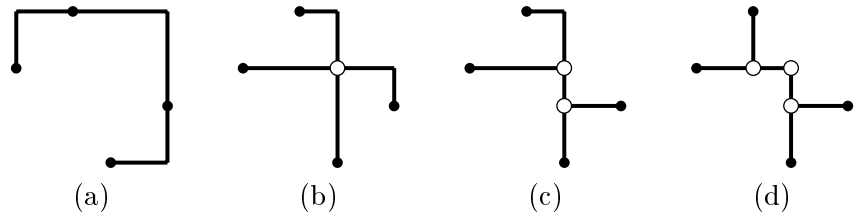


Figure 18: 1-Steiner heuristic example. (a) MST of terminals. (b)-(d) Sequential insertion of Steiner points. The degree-2 Steiner point in the final tree is removed by the algorithm.

A number of variants of the iterated 1-Steiner heuristic have been proposed. One variant is to add several Steiner points in each iteration, hereby avoiding expensive MST computations [13]. Until recently this variant was recognized as the champion heuristic for RSTP with respect to solution quality, producing solutions within 0.5% from optimum on average. A new heuristic proposed by Mandoui, Vazirani and Ganley [15] challenges the leading position of iterated 1-Steiner. On the top-level this new heuristic is similar to iterated 1-Steiner; it also adds one or more Steiner points to the terminal set until the MST does not improve. However, Steiner points are identified using a much more sophisticated algorithm. This algorithm is based on the directed IP formulation for STPG (Section 3.3). By applying a primal-dual approach using this formulation, a heuristic solution is constructed. The Steiner points used by this heuristic solution are added to the terminal set and the process iterates.

Although these variants of the iterated 1-Steiner approach produce ex-

cellent solutions, their running time requirement is not significantly lower than what is needed to construct SMTs using the exact algorithms described in Section 3.2 — except in very large, rare or contrived instances.

4.3 Arora's PTAS

In this section we describe the "ultimate" heuristic for RSTP, at least from a theoretical point of view. Given any fixed $\epsilon > 0$, the heuristic produces (in polynomial time) a rectilinear tree whose length is within a factor of $1 + \epsilon$ from optimum. This polynomial time approximation scheme (PTAS) was given by Arora [1], and it was a major breakthrough with its main focus on approximating the Euclidean travelling salesman problem (TSP). The algorithm and proof of approximation for RSTP were essentially corollaries to the results for TSP.

This presentation focuses on the algorithmic ideas leading to the PTAS. Therefore, we state the approximation theorem without giving its proof. The algorithm consists of three steps: 1) Perturbation 2) Shifted quadtree construction and 3) Dynamic Programming. In the following we describe each of these steps.

Let $OPT = |SMT(Z)|$ be the length of the SMT for the n terminals in Z. We assume w.l.o.g. that $\epsilon = 2^{-C}$ and $n = 2^N$ where C and N are positive integers. Note that ϵ is considered to be a *constant* in all subsequent asymptotic expressions.

Perturbation

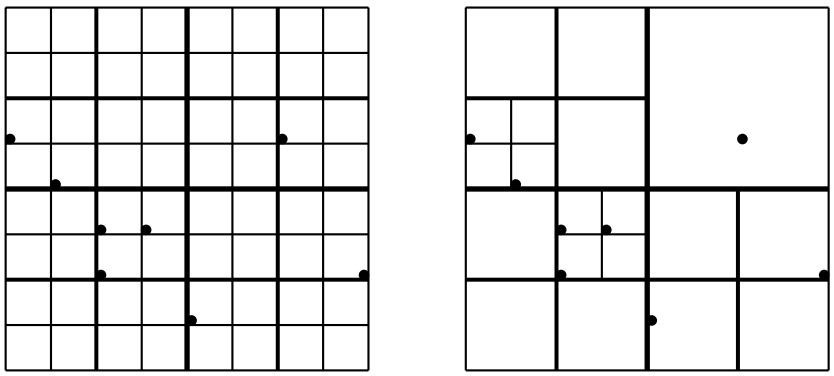
First we perturb and scale the terminal coordinates as follows. The *bounding square* of Z is the smallest axis-aligned square that contains Z. Scale the coordinates of the terminals such that the side length of the bounding square becomes $2n/\epsilon = 2^{N+C+1} = O(n)$. Note that $OPT \geq 2n/\epsilon$ in the scaled instance. Place a unit grid over the bounding square and move the terminals to the nearest grid point. Since every terminal is moved at most distance 1, the relative length difference between an SMT for the original (scaled) problem instance and an SMT in the perturbed instance is at most $n/OPT \leq n/(2n/\epsilon) = \epsilon/2$. Therefore, in order to find a $1+\epsilon$ approximation in the original instance, all we need is to find a $1+\epsilon/2$ approximation in the perturbed instance.

Shifted quadtree construction

Consider the perturbed instance: All terminals in Z are grid points on a $L \times L$ unit grid U, where $L = 2^{N+C+1} = O(n)$. Since the Hanan grid for Z is a subset of U, there exists an SMT in U. In particular note that all Steiner points will be grid points.

In order to decide to which sub-square a terminal belongs, move every terminal symbolically up and to the right, such that it belongs to exactly one unit square of U; special care is obviously needed for terminals on the right or upper boundary of U.

A *dissection* is a recursive partitioning of U into smaller squares. The dissection is represented by a 4-ary tree (every internal node has four children) with the root representing U. Divide U into four equal squares, represented by the children of the root. For each of the four children again divide the sub-square into four equal squares etc. The process stops when a node represents a unit square in U (Figure 19a). The depth of this tree is clearly $\log(2n/\epsilon) = O(\log n)$, and it has as many leaves as there are unit squares in U, i.e., $O(n^2)$ leaves.



(a) (b)

Figure 19: Subdivision of bounding square. (a) Dissection; (b) Quadtree.

Let us now bring the terminals into play. A *quadtree* is similar to a dissection, except that we only divide a sub-square if it contains more than one terminal (Figure 19b). How many nodes are there in a quadtree? The tree has at most $4n$ leaves and since the depth of the tree is $O(\log n)$, there are at most $4n \times O(\log n) = O(n \log n)$ nodes in the quadtree.

Let a and b be two integers in $\{0, \ldots, L-1\}$. A *quadtree with shift (a,b)* is a quadtree in which the first vertical division of U occurs at x-coordinate a and the horizontal division at y-coordinate b, where the usual quadtree has $a = b = L/2$. The bounding square is still divided into four sub-squares, but these may be "wrapped-around" as shown in Figure 20. Subsequent divisions are made relative to the first division and may therefore also wrap-around.

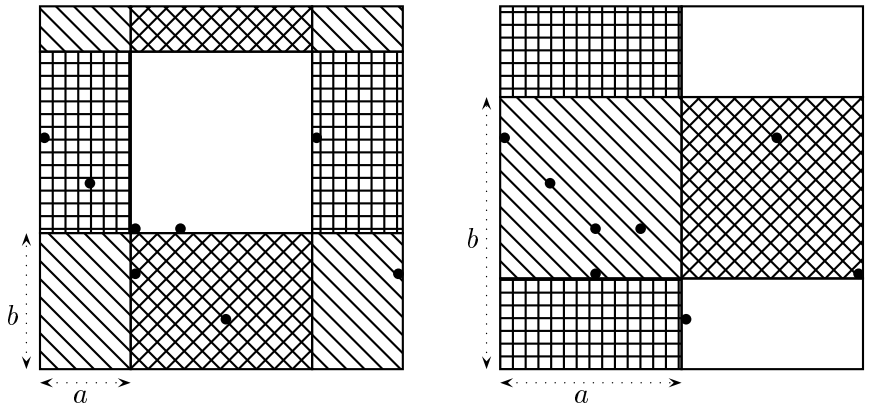


Figure 20: Shifted quadtree. Two examples with different a and b shifts are shown. Note that each region consists of up to four rectangles, as indicated with the fill styles used.

Given a shift (a, b), an (m, r)-light rectilinear Steiner tree for Z is a tree that intersects each side of every sub-square in the shifted quadtree at most r times; furthermore, these intersections occur at m prespecified *portals* on each sub-square side. Now we have the following theorem:

Theorem 4.2 *[1] Let shifts $0 \leq a, b < L$ be picked randomly. Then with probability at least $1/2$ there exists a rectilinear Steiner tree with total length at most $(1+\epsilon)OPT$ that is (m, r)-light with respect to the shifted quadtree where $m = O(\log L/\epsilon) = O(\log n)$ and $r = O(1/\epsilon) = O(1)$.*

Therefore, if we find an *optimal* (m, r)-light rectilinear Steiner tree for a given shift (a, b) (with m and r defined as in Theorem 4.2), we have constructed a $(1+\epsilon)$ approximation with probability at least $1/2$. By trying all $L \times L = O(n^2)$ shifts we are guaranteed to find the required approximation.

In the next section we show that an optimal tree for a given shift indeed can be found in polynomial time.

Dynamic programming

We use a dynamic programming algorithm to compute an optimal (m, r)-light rectilinear Steiner tree for a given (a, b)-shift. The algorithm will be described for $a = b = L/2$, i.e., ignoring squares that wrap-around (the general case is similar but a bit involved). First, note that a quadtree can be computed in $O(n \log^2 n)$ time by using a sorting-based algorithm.

The optimal (m, r)-light tree is constructed bottom-up in the quadtree; optimal solutions to subproblems are stored and used to construct optimal solutions to parent problems.

Consider a leaf node in the quadtree and a corresponding sub-square Q. The intersection of the optimal global tree with Q is a forest. However, this forest has at most $4r = O(1)$ intersections with the boundary of Q, and these intersections are to be chosen among $4m = O(\log n)$ prespecified portals. Thus there are $O((\log n)^{O(1)})$ possible ways in which the optimal tree can intersect the boundary of Q. For each of these, we construct a minimum length forest that spans the intersections and the single terminal in Q (if any). Each forest can clearly be constructed in $O(1)$ time — thus each leaf in the quadtree can be processed in $O((\log n)^{O(1)})$ time.

For each internal node there are again $O((\log n)^{O(1)})$ ways in which the global optimal tree can intersect the corresponding square. For a given intersection specification, the optimal forest is constructed by combining optimal solutions stored at the children. Again the processing of each internal node takes $O((\log n)^{O(1)})$ time. When the dynamic programming finishes, the optimal tree can be read from the root node of the quadtree.

Since there are $O(n \log n)$ nodes in the quadtree, the total running time to compute an optimal (m, r)-light tree for $m = O(\log n)$ and $r = O(1)$ is $O(n(\log n)^{O(1)})$.

5 Conclusion

The practical and theoretical developments with regard to RSTP over the last five years are amazing. On the practical side, we can today solve problem instances with several thousand terminals to optimality. On the theoretical side, Arora constructed a polynomial-time approximation scheme for the problem, showing that it can be approximated arbitrarily well in polynomial time. However, the FST based exact algorithms leave several theoreti-

cal problems open, e.g., whether it is possible to prove strong average-case bounds on the number of surviving FSTs for randomly generated instances. Conversely, Arora's breakthrough poses the question of whether a practical implementation of his sophisticated algorithm can be devised. In addition, it is still an open question whether the suggested exact and heuristic algorithms can be applied to variants and generalisations of RSTP.

Further Reading

Heuristics and VLSI oriented variants of RSTP are described in the book by Kahng and Robins [13]. For more information on the Steiner tree problem under other metrics and in higher dimensions, consult the book by Hwang, Richards and Winter [12], and the books by Cieslik [3, 4].

Exercises

Exercise 1 Prove that the number of Steiner points in an SMT spanning n terminals is at most $n-2$. Hint: Assume that the tree has k Steiner points; use the fact that the tree has $n+k-1$ edges and that the degree of every Steiner point is at least 3.

Exercise 2 Give a sequence of sliding and flipping operations that transform the fulsome SMT in Figure 1 (middle) to the fulsome and canonical SMT in Figure 1 (bottom).

Exercise 3 Prove that the number of FSTs in a tree is $1 + \sum_{z \in Z}(deg(z) - 1)$, where $deg(z)$ is the degree (number of incident FSTs) of terminal $z \in Z$.

Exercise 4 Prove Theorem 2.5, the Hanan grid theorem, using Hwang's characterization of FSTs (Theorem 2.3).

Exercise 5 Given a Hwang-topology FST F spanning at most four terminals, show that there always exists a rectilinear minimum spanning tree of length at most $3/2|F|$ that spans the same set of terminals.

Exercise 6 Prove that the use of $MST(Z)$ to define bottleneck Steiner distances in Section 3.1 is the best possible in the sense that there exists no terminal-path $P(z_i, z_j)$ connecting $z_i, z_j \in Z$ for which the longest edge is shorter than $b_{z_i z_j}$. Hint: Assume such a path exists and prove that $MST(Z)$ would not be minimal.

Exercise 7 Consider removing a set $e_1, e_2, \ldots, e_k$ of distinct edges from an SMT. For each of the remaining components containing at least one terminal, choose one (arbitrary) terminal from this component. Let Z_R be the set of chosen terminals.

Construct a minimum spanning tree (MST) over Z_R using $b_{z_i z_j}$ (defined in Section 3.1) as the distance between a pair of terminals $z_i, z_j \in Z_R$. Let l be the length of this MST. Show that $l \geq \sum_{l=1}^{k} |e_l|$.

Exercise 8 The FST generation algorithm described in Section 3.2 outputs the $n-1$ edges of an *arbitrary* MST for Z as candidates for 2-terminal FSTs in an SMT for Z. Prove that this is indeed sufficient, i.e., that there exists an SMT in which all 2-terminal FSTs edges come from one particular MST. Hint: Consider an SMT containing a *minimum* number of 2-terminal FSTs *not* belonging to the particular MST and arrive at a contradiction.

Exercise 9 Prove Lemma 4.1. Hint: Assume that there are two neighbours $u, v \in Z$ within a single region. Prove that $|uv| < |uz|$ or $|uv| < |vz|$, contradicting the fact that the tree is an MST. Use a similar argument for proving that the single neighbour must be a *closest* neighbour.

Acknowledgments

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

References

[1] S. Arora. Polynomial Time Approximation Schemes for Euclidean Traveling Salesman and Other Geometric Problems. *Journal of the ACM*, 45(5):753–782, 1998.

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

[3] D. Cieslik. *Steiner Minimal Trees.* Kluwer Academic Publishers, Boston, 1998.

[4] D. Cieslik. *The Steiner Ratio.* Kluwer Academic Publishers, Boston, (to appear).

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

[6] M. X. Goemans and Y. S. Myung. A Catalog of Steiner Tree Formulations. *Networks*, 23:19–28, 1993.

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

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

[9] A. Hetzel. *Verdrahtung im VLSI-Design: Spezielle Teilprobleme und ein sequentielles Lösungsverfahren*. PhD thesis, Institute for Discrete Mathematics, University of Bonn, 1995.

[10] J.-M. Ho, G. Vijayan, and C. K. Wong. New Algorithms for the Rectilinear Steiner Tree Problem. *IEEE Transactions on Computer-Aided Design*, 9(2):185–193, 1990.

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

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

[13] A. B. Kahng and G. Robins. *On Optimal Interconnections for VLSI*. Kluwer Academic Publishers, Boston, 1995.

[14] T. Koch and A. Martin. Solving Steiner Tree Problems in Graphs to Optimality. *Networks*, 33:207–232, 1998.

[15] I. Mandoiu, V. V. Vazirani, and J. L. Ganley. A New Heuristic for Rectilinear Steiner Trees. *IEEE Transactions on CAD*, 19:1129–1139, 2000.

[16] D. S. Richards and J. S. Salowe. A Simple Proof of Hwang's Theorem for Rectilinear Steiner Minimal Trees. *Annals of Operations Research*, 33:549–556, 1991.

[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] E. Uchoa, M. Poggi de Aragão, and C. Ribeiro. Preprocessing Steiner Problems from VLSI Layout. Technical Report MCC. 32/99, PUC-Rio, Brazil, 1999.

[19] D. M. Warme. *Spanning Trees in Hypergraphs with Applications to Steiner Trees*. PhD thesis, Computer Science Dept., The University of Virginia, 1998.

[20] D. M. Warme, P. Winter, and M. Zachariasen. Exact Algorithms for Plane Steiner Tree Problems: A Computational Study. In D.-Z. Du, J. M. Smith, and J. H. Rubinstein, editors, *Advances in Steiner Trees*, pages 81–116. Kluwer Academic Publishers, Boston, 2000.

[21] D. M. Warme, P. Winter, and M. Zachariasen. GeoSteiner 3.1. Department of Computer Science, University of Copenhagen (DIKU), `http://www.diku.dk/geosteiner/`, 2001.

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

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

[24] M. Zachariasen. Rectilinear Full Steiner Tree Generation. *Networks*, 33:125–143, 1999.

[25] A. Z. Zelikovsky. An 11/8-approximation Algorithm for the Steiner Problem on Networks with Rectilinear Distance. In *Janos Bolyai Mathematica Societatis Conf.: Sets, Graphs, and Numbers*, pages 733–745, 1991.