A Linear-Time Algorithm to Construct a Rectilinear Steiner Minimal Tree for k-Extremal Point Sets

D.S. Richards and J. S. Salowe

Computer Science Report No. TR-90-11
May 22, 1990

Algorithmica, final

A Linear-Time Algorithm to Construct a Rectilinear Steiner Minimal Tree for k-Extremal Point Sets

by

D.S. Richards, J.S. Salowe

Department of Computer Science
University of Virginia
Charlottesville, Virginia 22903

ABSTRACT

A k-extremal point set is a point set on the boundary of a k-sided rectilinear convex hull. Given a k-extremal point set of size n, we present an algorithm that computes a rectilinear Steiner minimal tree in time $O(k^4 n)$. For constant k, this algorithm runs in $O(n)$ time and is asymptotically optimal, and for arbitrary k, the algorithm is the fastest known for this problem.

Keywords: Dynamic Programming, Polynomial-time Algorithm, Rectilinear Convex Hull, Rectilinear Metric, Steiner Minimal Tree.

1. Introduction

Given a set S of n terminals in the plane, the rectilinear Steiner minimal tree problem is to find a minimum-length interconnection for S using only vertical and horizontal edges. In finding this tree, new points called *Steiner points* can be created. The decision problem associated with the rectilinear Steiner minimal tree problem is NP-complete [6]. Nevertheless, the optimization problem is important enough that substantial research activity has focused on heuristic and special-case algorithms, as well as discovering new properties of Steiner trees [9].

In this paper, we devise an algorithm to construct a rectilinear Steiner minimal tree when S lies on the boundary of its *rectilinear convex hull*. A rectilinear convex hull $Rconv(S)$ is a smallest-area simply-connected figure containing a shortest path between every pair of terminals in S, where paths consist of horizontal and vertical edges. Equivalently, if one places a coordinate axis at any point on the boundary of $Rconv(S)$, at least one of the quadrants is empty [3]. The boundary $B(S)$ of $Rconv(S)$ consists of vertical and horizontal edges.

Let $B = B(S)$. Call a point set which lies on B *extremal*. Our contribution is a Steiner tree algorithm for k-extremal point sets; a k-extremal point set is an extremal point set where B has exactly k sides. We give an algorithm that finds a rectilinear Steiner minimal tree for k-extremal point sets of size n which runs in $O(k^4 n)$ time. If k is a constant, our algorithm runs in time linear in the number of input points. If k is near n, it is as efficient as the fastest known algorithm for this problem.

Algorithms for an extremal point sets appear in Bern [3] and Provan [12]. Given an extremal point set of size n, Bern's algorithm runs in $O(n^5)$ time and Provan's in $O(n^6)$ time. Neither algorithm depends on k. In addition to the work of Bern and Provan, some relevant work on special-case rectilinear Steiner minimal tree algorithms appears in the literature. Aho, Garey, and Hwang [2] give a linear-time algorithm when S lies on two parallel lines, and they give a cubic-time algorithm when S lies on the boundary of a rectangle. The latter result was subsequently improved to linear time by Agarwal and Shing [1] and Cohoon, Richards, and Salowe [4]. Richards and Salowe [13] simplify results of Aho, Garey, and Hwang [2] and Hwang [8] and study L- and T-shaped rectilinear convex hulls. All of the algorithmic results above are subsumed or improved on by this paper.

The remainder of the paper is divided up into five sections. In Section 2, the theoretical underpinnings of the algorithm are presented. The main result is Theorem 1, which describes a canonical form for rectilinear Steiner minimal trees. With Theorem 1, we are able to restrict how connected components of a ''canonical'' tree appear in the interior of $Rconv(S)$. One such case is the simple Steiner tree: a simple Steiner tree is a Steiner tree whose intersection with the interior of $Rconv(S)$ is a set of nonintersecting line segments, each of which only has endpoints on B. In Section 3, an algorithm for simple Steiner trees is given; simple Steiner trees form the basis for a dynamic programming algorithm for the remainder of the forms in Theorem 1; this algorithm is given in Section 4. The chief result of Section 4 is Theorem 2 which shows that the connected components can be made to lie on a limited number of grid lines called ''blue'' grid lines. We show that there are $O(k)$ blue grid lines, and each subproblem is bounded by at most four blue grid lines, giving a total of $O(k^4)$ subproblems to solve. Each subproblem takes $O(k^4+n)$ time, giving an $O(k^8+k^4 n)$ time algorithm. In Section 5, we reduce the time complexity to $O(k^4 n)$ by combining subsolutions more carefully. Section 6 summarizes the results and gives directions for further research.

2. A Canonical Form for Steiner Minimal Trees

In this section, we prove several results to restrict our search for rectilinear Steiner minimal trees. This section is divided up into two subsections. The first subsection presents terminology that will be used throughout the paper, and the second subsection gives the theoretical results describing the canonical form.

2.1. Terminology

Let the set S of input points be called *terminals.* Define the *grid graph* for S in the following manner. Draw vertical and horizontal lines through each terminal; the vertices of the grid graph are the intersection points of these lines, and the edges are line segments connecting two adjacent grid points. The grid points will sometimes be called *Hanan grid points*. Hanan [7] proved that there is a rectilinear Steiner minimal tree which is a subset of the grid graph. Yang and Wing [14] further proved that there is a rectilinear Steiner minimal tree which is a subset of the grid graph within $Rconv(S)$.

We further restrict the search for rectilinear Steiner minimal trees by developing and applying tie-breaking rules. Among the set τ of rectilinear Steiner minimal trees for S lying on the grid graph within $Rconv(S)$, we prefer certain trees and prove that in such trees, forms and positions of Steiner tree edges are limited. Given τ, let $\tau_1 \subseteq \tau$

be the trees which maximize length along the boundary B. Let $\tau_2 \subseteq \tau_1$ be the trees where in addition the vertex degree of the terminals and the ''inner boundary corners'' of B is maximized. The terminals and the inner boundary corners of B will be called *nodes*. (Inner boundary corners of B are defined below; this tie-breaking rule is called *node degree*.) Finally, let $\tau_3 \subseteq \tau_2$ be trees which are *leftmost* in the sense described below. Trees in τ_3 are called *optimal Steiner trees*. For future reference, we highlight and abbreviate the tie-breaking rules in the order they are applied:

1. Boundary Length.
2. Node degree.
3. Leftmost trees.

There are three fundamental operations that will be performed on Steiner trees in this paper: sliding, flipping, and transplanting. The sliding operation is performed on H-shaped subtrees as indicated in Figure 2.1. The crossbar edge can be *slid*, back and forth, without increasing length. The *flipping* operation is performed on interior *corners* (a pair of lines incident at a degree two vertex which is not a terminal). The two edges forming the corner can be *flipped* as depicted in Figure 2.1. Finally, in the *transplanting* operation, an edge e is added to a tree to form a single cycle, and an edge e' on the cycle is deleted, where the length of e is less than or equal to the length of e'. None of these operations increases the length of the Steiner tree.

We need to be more specific in defining slides and flips to say when trees are *leftmost*. There are four directions to slide in and four directions to flip in. The directions and two examples are shown in Figure 2.1. For a tree to be leftmost, it is not possible to make a sequence of N- and S-slides ending in a NW-flip, a SW-flip, or a W-slide in the interior of B.

Let a corner whose horizontal edge is to the right of its vertical edge be called a *right-bending* corner and all others *left-bending* corners. One consequence of the definition of leftmost trees is the following fact about corners in the interior of B: All corners in the interior of B must bend towards the right since a NW- or SW-flip can be

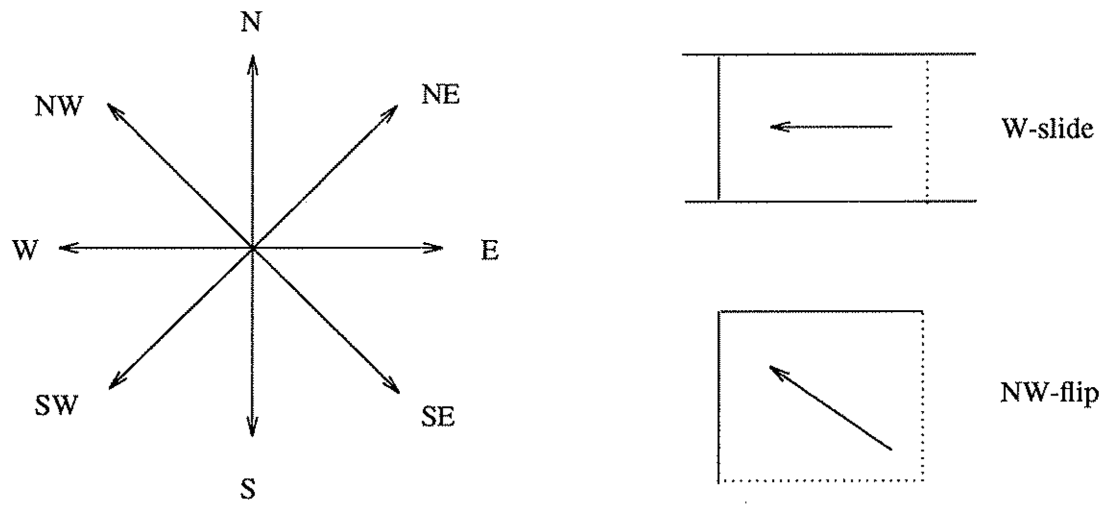


Figure 2.1 — Directions to Slide and Flip

applied to a left-bending corner.

An *interior edge* is an edge inside B that does not properly contain any terminals or Steiner points. An *interior line* consists of one or more adjacent, collinear interior edges. A *complete interior line* (sometimes called a complete line when the context is clear) is an interior line which stretches completely across *Rconv* (S). The *relative interior* of a line does not include the endpoints, but the *closure* does. Similarly, the relative interior of a figure does not contain the boundary, but the closure does. Edges are said to be *properly incident* to a line if they intersect the relative interior.

A "non-terminal" of degree two is called a *corner-vertex*, a Steiner point of degree three is called a *T-vertex*, and a Steiner point of degree four is called a *cross-vertex*. Steiner points which are connected by a single edge are said to be *adjacent*. Two interior lines l_1 and l_2 are said to *alternate* off a third line l if l_1 and l_2 intersect the relative interior of l at two adjacent T-vertices, and l_1 and l_2 are on opposite sides of l. Edges are said to *alternate off of l* if every adjacent pair of lines incident to the relative interior of l alternates.

As mentioned, a corner consists of two lines incident to a corner-vertex. These lines are called the *legs* of the corner. A corner is a *complete corner* if the corner is inside B, but both legs intersect B. A *T* consists of three lines intersecting at a T-vertex. The two collinear lines are called the *head* of the T, and the third line is called the *body*.

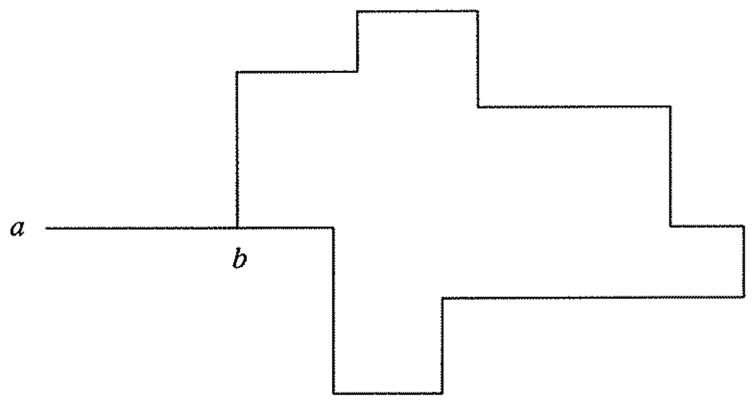
The boundary contains a series of *boundary edges*. Boundary edges can meet at terminals or *boundary corners*. With respect to the interior of *Rconv* (S), an *inner boundary corner* is a reflex angle, and an *outer boundary corner* is a convex angle. Note that all outer boundary corners are terminals. There are at most four special *boundary lines*, or series of collinear, adjacent boundary edges, called *tabs*. Tabs connect two outer boundary corners and are at the positions as far to the north, south, east, and west as possible on B.

2.2. The Canonical Form

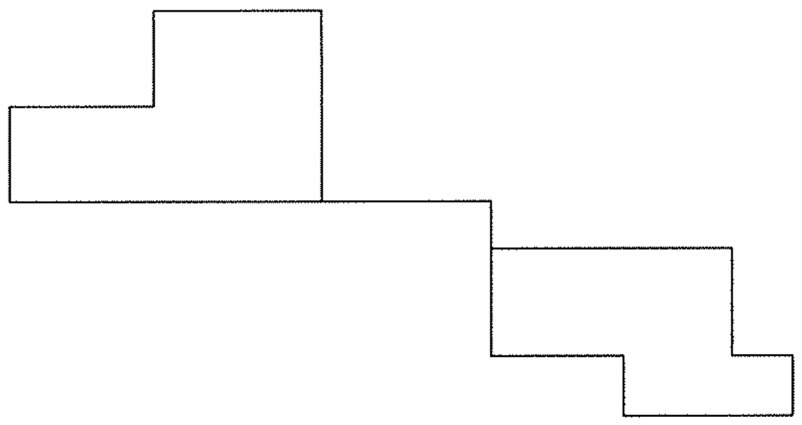
In this subsection, we present results to restrict the forms of optimal Steiner trees.

In general, there are only four extreme points, so tabs may be degenerate and contain only one point. There is another type of degeneracy where regions in *Rconv* (S) of nonzero area are connected by rectilinear lines. These two degeneracies are depicted in Figure 2.2. In order to remove these degeneracies, we preprocess the rectilinear convex hull. Suppose the left tab is degenerate, and let a be the single leftmost point in *Rconv* (S). By the result of Yang and Wing, we can add a new terminal b as depicted in Figure 2.2 and remove the edge to a to get a new problem with a nondegenerate tab. All tabs can be processed in this manner. Similarly, if regions of nonzero area are separated by rectilinear lines, Yang and Wing's result implies that the rectilinear convex hull can be separated into two or more nondegenerate subproblems. Note that the nonempty regions can be identified with a simple linear-time scan[11]. We will therefore assume in the text below that no degeneracies are present.

Lemma 1: Let p_1 and p_2 be two adjacent Steiner points lying on interior line l. In an optimal Steiner tree, the lines incident to p_1 and p_2 perpendicular to l must be on opposite sides of l.



Degenerate Tab



Two Regions With Nonzero Area

Figure 2.2 — Two Degeneracies

Proof: Let e_1 and e_2 be perpendicular to l and incident to p_1 and p_2, respectively. Assume to the contrary e_1 and e_2 lie on the same side of l; if either line crosses l we ignore the other portion. One of the two lines, say e_1, ends first; that is, e_1 is no longer than e_2. Note e_1 cannot end at either a corner-vertex bending towards e_2 or a T-vertex; otherwise, the portion u of l between e_1 and e_2 can be slid to decrease total length. (Note that no portion of the Steiner tree can be between e_1 and e_2 and reachable by sliding u. This observation will be used implicitly in several proofs.) If e_1 ends at a corner-vertex with the corner pointing away from e_2, the corner can be flipped to decrease total length. Therefore, e_1 must hit the boundary at point a.

There are two boundary edges incident to point a. If one of the edges is perpendicular to e_1 and between e_1 and e_2, sliding u to a increases boundary length; if not, e_1 ends at a boundary corner, and sliding u to a increases node degree. □

Lemma 2: In an optimal Steiner tree, edges properly incident to a leg of a corner must alternate. The edge closest to the corner-vertex must point in the direction opposite of the other leg of the corner.

Proof: Let C be a corner with legs S_1 and S_2, and let edge e be the edge closest to the corner-vertex c intersecting S_1. If e and S_2 do not point in opposite directions, either e points in the same direction as S_2 or e intersects S_1 at a cross-vertex. In either case, flipping C decreases the length of the Steiner tree. (See Figure 2.3.) If there are exactly two edges incident to one leg of a corner, the argument above and Lemma 1 imply that they must alternate. If there are three or more edges properly incident to a leg, Lemma 1 implies that the edges must alternate. □

Lemma 3: In an optimal Steiner tree, the body of a T must hit the boundary. Furthermore, no interior edges are properly incident to the body of a T.

Proof: Let b be the body of a T, and let h be the head. The first vertex that line b intersects is either an interior vertex or a point on the boundary. If it is an interior vertex, it has either one or two edges perpendicular to b. If there is only one perpendicular edge, the vertex is either a corner-vertex or a T-vertex. Lemma 2 implies that the edges incident to the legs of a corner must alternate, so the first point that b intersects cannot be a corner-vertex.

We now consider the case when b intersects one perpendicular edge at a T-vertex. Let e be the edge perpendicular to b at the T-vertex, let f be the portion of h pointing in the same direction as e, and let t be the portion of b between e and f. One of e and f must end first (i.e. closest to b). Using the same argument as in Lemma 1, it follows that the only possibility is for f to end before e at a corner bending away from e. There are four cases where the corner bends towards the right and the configuration does not violate Lemma 2; they are depicted in Figure 2.4. In each case, the trees are not leftmost: In cases 2 and 3, we can perform a W-slide on t. In case 1, one can do a N-slide on t followed by a NW-flip, and in case 4, we do a S-slide on t followed by a SW-flip. We conclude that f cannot end first, so b cannot first intersect a T-vertex.

Finally, if b intersects two perpendicular edges at vertex c, let t be the portion of b between h and c (t may be all of b). If t is vertical, the tree is not leftmost as we can perform a W-slide. If t is horizontal, let e be the edge incident to c above b, and let f be the portion of h above b. One of e and f must end first, though not at a T-vertex or cross vertex (N-slide on t decreases total length) and not at the boundary (N-slide on t either increases boundary length or increases node degree). Therefore, the first line ends at a corner, pointing towards the right. It follows that this line l is the rightmost of h and e. By symmetry, the lower endpoint of l also ends at a corner bending towards the right, so shifting l towards the right decreases total length. □

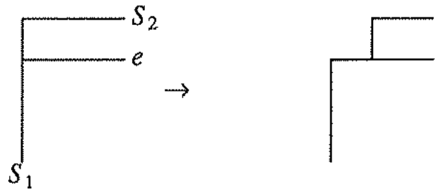


Figure 2.3

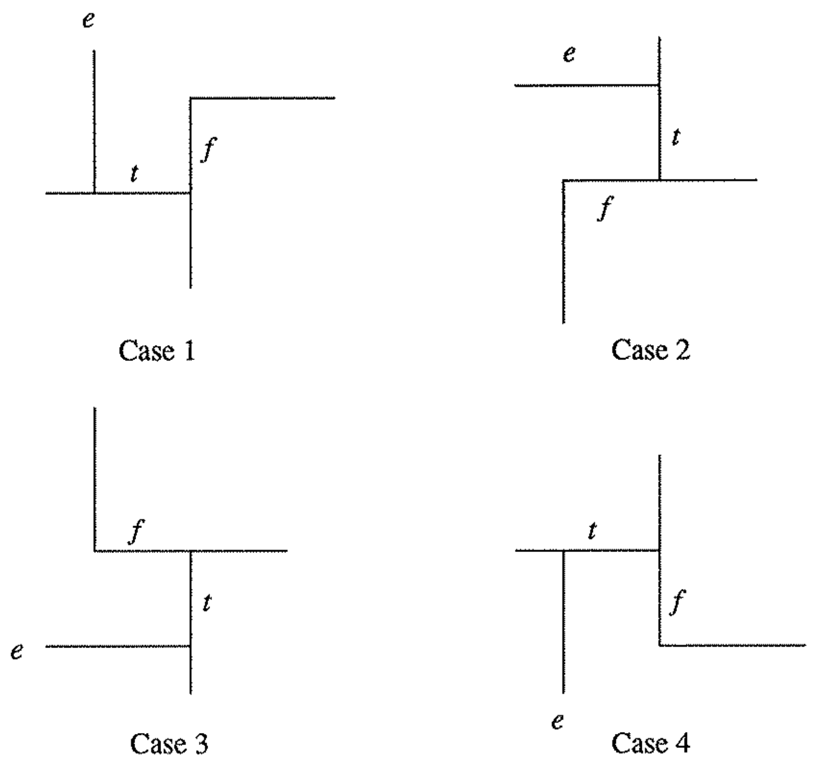


Figure 2.4

Lemma 4: In an optimal Steiner tree, the body of a T intersects a node.

Proof: Let b be the body of a T, and let h be the head. Lemma 3 shows that b hits the boundary and does not contain any interior Steiner points. Assume to the contrary that b does not end at a boundary corner or at a terminal, so b must intersect the interior of a boundary edge E. There must be Steiner tree edges on both sides of b along E, otherwise the corner at $b \cap E$ can be flipped to decrease total length.

Line b can be slid in one direction until it is incident to a terminal (on either E or h), a boundary corner (on either E or h, possibly increasing the use of boundary length), or a corner on h bending away from E. The first two possibilities contradict optimality, since node degree increases. A symmetric argument can be applied when b is slid in the other direction. Therefore, h must end in two corners, each bending away from E, so h can be slid away from E to decrease total length. □

Lemma 5: In an optimal Steiner tree, the legs of a corner must hit the boundary, that is, the corner is a complete corner. At most one of the legs can have more than one properly incident line.

Proof: Lemma 3 implies that a leg of a corner cannot be the body of a T. Since trees are leftmost, corners must bend towards the right, so the horizontal leg of the corner cannot end at a second corner as this corner would bend towards the left. The horizontal leg, therefore, ends at the boundary.

If the vertical leg l of a corner ends at a corner-vertex, this must bend towards the right. Let these two corners be called c and c'. By Lemma 2, edges properly incident to l must alternate, and the edges closest to c and c' must point towards the left. Line l can be shifted towards the right to decrease total length. (See Figure 2.5.)

To show that only one leg can have more than one incident edge, suppose to the contrary that each leg has at least two incident edges. Let l_1 be the vertical leg, and let l_2 be the horizontal leg. Also let e_1 and e_2 intersect l_1, and let f_1 and f_2 intersect l_2. (These edges are indexed according to increasing distance from the corner-vertex.) Lemma 4 implies that e_2 and f_2 end at nodes and do not contain any interior Steiner points. Let e_2 end at c, and let f_2 end at point d. There are four cases, depending on the orientation of the corner and whether e_1 is above or below d. For one of the two possible corner orientations, two subcases are depicted in Figure 2.6. Note that c must be to the left of d by convexity, and that for the orientation in the figure, c must be below d by convexity. In either case, a series of slides and flips increases the node degree by one, since c and d are either boundary corners or terminals. □

Lemma 6: In an optimal Steiner tree, the head of a T is either a leg of a complete corner or a complete line.

Proof: If one side of the head ends at a corner-vertex, then Lemma 5 implies that the head is one leg of a complete corner. Lemma 3 states that it is not possible for the head of a T to be the body of another T, so the only other possibility is that both sides of the head intersect the boundary. □

Let a *topology* be a connected portion of a Steiner tree in the relative interior of B. Since it is in the relative interior, a topology does not contain any terminals or Steiner points on the boundary. The main theorem limits the class of Steiner tree topologies for optimal Steiner trees.

Theorem 1: A topology in an optimal Steiner tree satisfying the tie-breaking rules must be one of the following types:

1. Interior lines with no Steiner points.
2. Two interior lines forming a cross with no other incident edges.
3. Interior lines with alternating edges.
4. Complete corners with no incident edges.
5. Complete corners with alternating edges incident to one leg and no edges incident to the other leg.

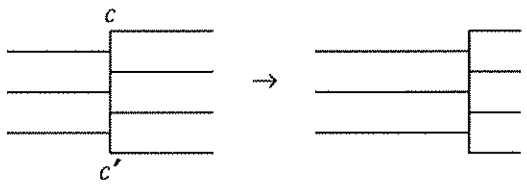
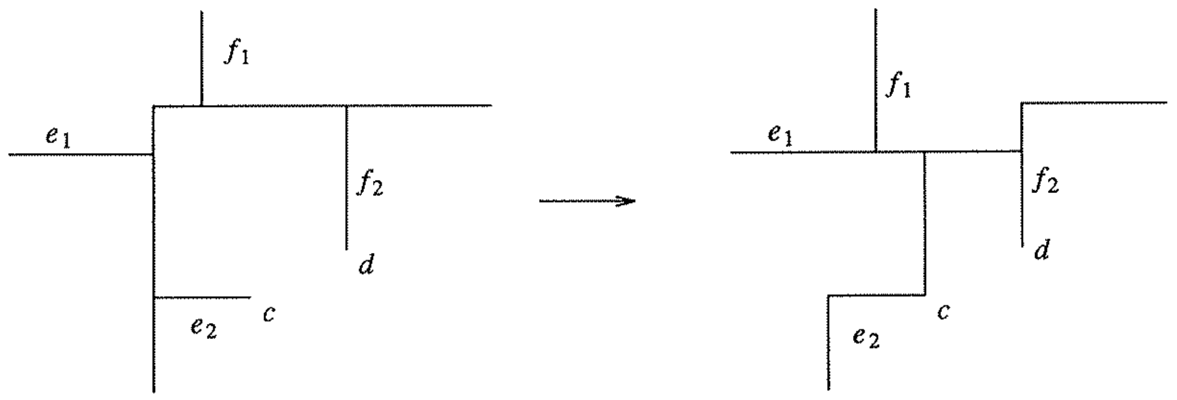
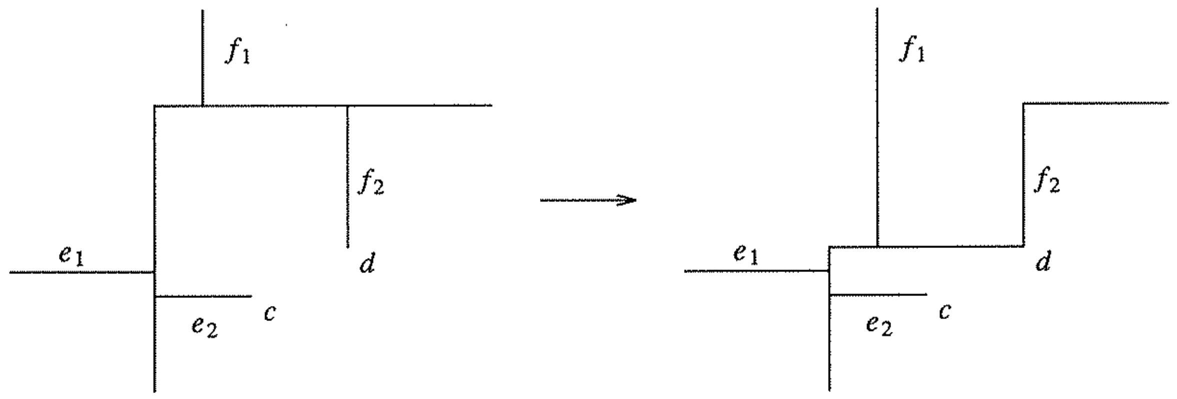


Figure 2.5



Case 1



Case 2

Figure 2.6

6. Complete corners with one edge incident to one leg and alternating edges incident to the other leg.

These topologies may occur several times, but they are disjoint in the relative interior of the convex hull. Furthermore, in each corner case, the edges closest to the corner-vertex must be directed opposite of the other leg of the corner.

Proof: Suppose that an optimal Steiner tree has edges in the relative interior of B. If a topology contains a vertical line l incident to one boundary, l either ends at the other boundary, bends at a corner, or ends at a T. If l hits the other boundary, it may or may not have Steiner points. If it does not, it is of type 1, if it has exactly one Steiner point, Lemma 3 implies that it is of type 2 or 3, and if it has two or more Steiner points, Lemmas 1 and 3 imply that the topology consists of l and incident, alternating edges and is of type 3.

If l bends at a corner, Lemma 5 implies that this corner must be a complete corner. Lemma 2 shows that edges incident to a leg must alternate, and Lemma 3 proves that these edges contain no interior Steiner points and must hit the boundary. Lemma 5 also proves that at most one leg can have more than one Steiner point, justifying types 4, 5 and 6.

If l ends at a T-vertex, Lemma 5 implies that the head of the T must be either a complete corner or a complete line. An analysis similar to the one above shows that one of the types 3, 5, and 6 must occur. A symmetric argument can be used if l is horizontal. □

3. Minimal Simple Steiner Trees

In a *simple* Steiner tree, only type 1 topologies from Theorem 1 can appear. Each interior line is an entire grid line, and there are no interior Steiner points. Note that the organization of interior lines in a simple Steiner tree can be complicated, as indicated in Figure 3.1.

We give a dynamic programming algorithm to find minimal simple Steiner trees. The basic step is of the following form. Consider any grid line ab between boundary points a and b, where a or b is a terminal or a boundary corner. This defines a subproblem whose boundary B_{ab} is composed of the boundary clockwise from a to b combined with the edge ab itself. For the terminal set on B_{ab} we find several constrained simple Steiner trees such that *no interior line is properly incident to ab*; such a simple Steiner tree is an *ab-tree*. In particular we compute:

T_a — A minimum ab-tree with a regarded as a terminal (whether or not it actually is a terminal).

T_b — Analogous to T_a.

T_{ab} — A minimum ab-tree with both a and b regarded as terminals.

$T_{\overline{ab}}$ — A minimum ab-tree constrained to contain the edge ab.

A $T_{\overline{ab}}$ tree with the edge ab removed can be regarded as a "Steiner forest" with two components, separating a and b.

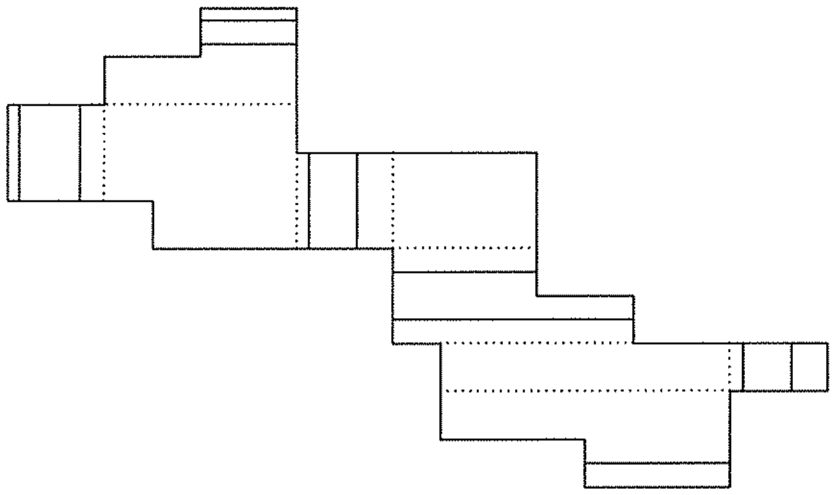


Figure 3.1 - A simple Steiner tree

3.1. Overview

The algorithm generates each B_{ab} with four line sweeps beginning at each of the tabs of B and progressing across the interior to the opposite tab. The processing of all four sweeps is interleaved; we always work next on the B_{ab} that has the smallest area (the process is similar to merging four lists), and the four degenerate zero-area cases are processed first.

It can be shown that at least one of the tabs in a simple Steiner tree has no interior lines properly incident to it. This is because a simple case analysis shows that at most two tabs can have incident lines in any simple Steiner tree.

Lemma 7: Let a and b be two endpoints connected by tab l_{ab} counterclockwise, and assume that there is a minimal simple Steiner tree where no interior line is incident to l_{ab}. Then a minimal simple Steiner tree is the shorter of $T_{ab} \cup k_{ab}$ and $T_{\overline{ab}}$, where k_{ab} is l_{ab} with its longest boundary edge deleted.

Proof: Let T be a minimal simple Steiner tree with no lines incident to l_{ab}. If T contains l_{ab}, then it is the same length as $T_{\overline{ab}}$. If T does not contain l_{ab}, there must be a gap between a and b, so a and b must be connected by T_{ab}, and the remainder of the terminals must be connected by k_{ab}. □

The algorithm actually begins with a preprocessing step discussed below. Lemma 7 guarantees we can get a minimal simple Steiner tree if we correctly compute T_a, T_b, T_{ab}, and $T_{\overline{ab}}$. The remainder of this section describes how to construct these trees.

Note that the m interior edges of a simple Steiner tree divides the interior of B into $m + 1$ subregions, called *open regions* since they are free of internal lines. Some open regions are simple rectangles while others are more complex. Each open region is rectilinearly convex and so has four tabs.

Our algorithm uses the fact that for any ab-tree all of ab is on the boundary of one open region. Conceptually, we solve the ab case by exhaustively trying all possible open regions abutting ab. We recursively solve the subproblems "behind" the region's other three tabs, not including ab, and combine these to calculate T_{ab}, etc., for each open region.

A *canonical region* for ab is defined without reference to a tree. Formally, it is a rectilinearly convex region containing ab whose other three tabs contain complete grid lines; the remainder of the boundary is from B_{ab}. An analysis of a canonical region below will be over all ab-trees with no internal lines properly within the canonical region. Note that given an ab-tree the open region that includes ab is itself a canonical region. Therefore if we exhaustively analyze all canonical regions then we will consider the open region of the optimal ab-tree.

We will see below that we can confine our search to a subset of all canonical regions. In particular we define a *minimum canonical region* to be a nonempty canonical region not contained in another canonical region. In Figure 3.2 there are several minimal canonical regions; two are bounded by the tabs e_1, e_3 e_5 and e_2, e_4 e_6. Note that there are many other canonical regions, such as the one bounded by e_1, e_4 e_5, that are not minimal.

We give some definitions that allow us to collapse many cases into one analysis. Let d_{xy} be the (rectilinear) distance from x to y. Let b_{xy} be the length of the boundary B_{ab} (the portion not including ab) from x to y. Let g_{xy} be the length of the boundary from x to y with the longest inter-terminal gap removed (where x and y are regarded as terminals in this definition, even if they are not). Let f_{xy} be the length of the boundary from x to y with the gap from

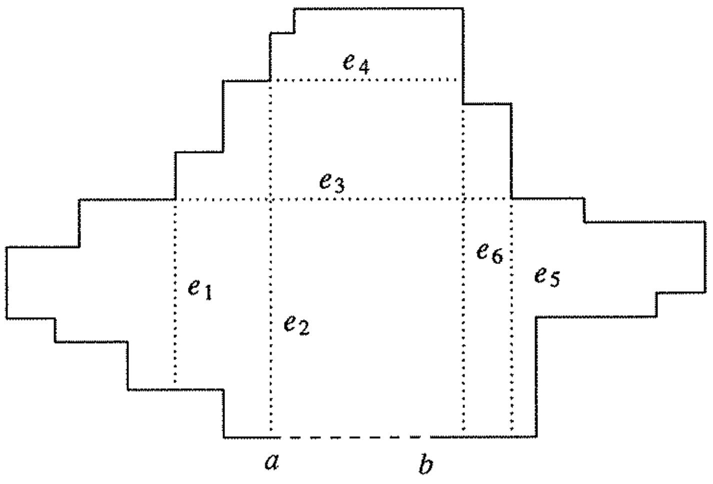


Figure 3.2 - Minimal canonical regions

y to the terminal nearest y removed; it is zero if there are no terminals between x and y. Note that f_{xy} is not necessarily f_{yx}. Let L_{ab} be the length of T_{ab}, and so on.

We refer to L_{ab} and similar quantities as "analyses" below. While L_{ab} is just an integer we assume that associated with it is a record of how it was calculated. In particular, there are several ways T_{ab} could have been constructed (which are explicitly listed below) and we keep a record of which way was used. As is typical in dynamic programming algorithms, we do not store T_{ab} but record how it was built from previous subanalyses with back-pointers.

3.2. The General Step

We begin by giving the details of how to compute the best possible T_{ab} etc. for a particular canonical region. Since for every open region there are one or more corresponding canonical regions, computing the best possible trees for each canonical region and minimizing over all such regions must give the correct answer. In a subsection below we show how to speed up this procedure considerably. We will assume hereafter, without loss of generality, that ab is horizontal and B_{ab} surrounds a region above it.

To delimit a canonical region we need to specify its upper horizontal tab and the vertical tabs on the left and right. A generic instance is shown in Figure 3.3. Some of the regions behind the tabs of the canonical region may be empty, thus creating simpler degenerate cases.

Note that we have already calculated $T_{\overline{cd}}$, T_{cd}, T_c, T_d, $T_{\overline{ef}}$, and so on, since these correspond to smaller-area subproblems. For any $T_{\overline{ab}}$ if we remove ab there is an inter-terminal gap on B_{ab}, excluding ab, between the (clockwise) last terminal connected to a and the first terminal connected to b; call this gap the *cut* for that $T_{\overline{ab}}$. To calculate $T_{\overline{ab}}$ we try all possible places where the cut can occur. There are several possibilities. If the cut occurs properly between h and e then

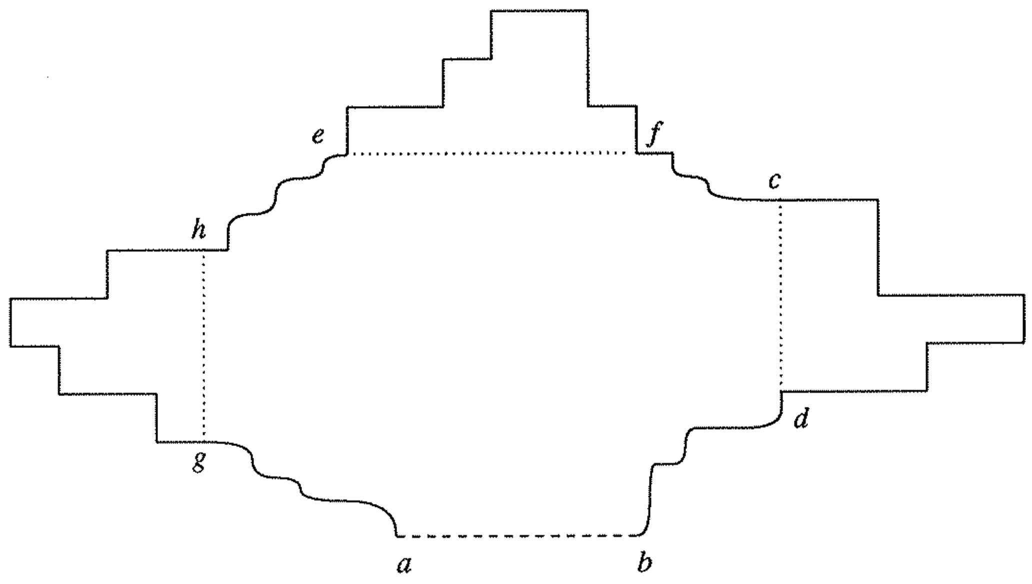


Figure 3.3 - A general canonical region above line *ab*.

$$L_{\overline{ab}} = d_{ab} + b_{ag} + L_{gh} + g_{he} + L_{ef} + b_{fc} + L_{cd} + b_{db}$$

If the cut is between *e* and *f* then

$$L_{\overline{ab}} = d_{ab} + b_{ag} + L_{gh} + b_{he} + L_{\overline{ef}} - d_{ef} + b_{fc} + L_{cd} + b_{db}$$

If the cut spans across *c* then

$$L_{\overline{ab}} = d_{ab} + b_{ag} + L_{gh} + b_{he} + L_{ef} + f_{fc} + L_{d} + d_{db}$$

This last case is only applied when *c* is not a terminal. The other cases are analogous. $L_{\overline{ab}}$ for a particular canonical region is the minimum over all these choices. The various boundary length quantities, such as b_{ag} and f_{db}, are computed in a preprocessing step.

To compute T_{ab} for this particular open region we get this simple relation,

$$L_{ab} = \min \begin{cases} L_{\overline{ab}} \\ b_{ag} + L_{gh} + b_{he} + L_{ef} + b_{fc} + L_{cd} + b_{db} \end{cases}$$

These cases are based on the presence or absence of the line between *a* and *b* in a minimal simple Steiner tree. The calculation of L_a depends on whether *b* is a terminal or not. If so we use L_{ab}, otherwise there may be terminals between *d* and *b* so we get

$$L_{a} = \min \begin{cases} b_{ag} + L_{gh} + b_{he} + L_{ef} + b_{fc} + L_{cd} + f_{db} \\ L_{ab} \end{cases}$$

The calculation of L_b is symmetrical.

The details for each of the cases are often simplified due to degeneracies; for example *c* may directly connect to *b* (so *d* is *b* and the region behind *cd* is empty). Of course, in some of these cases it is possible for more than one tab to be "behind," say, gh; in that way Figure 3.3 is not completely general. However the algorithm is general enough to include all these cases, since degenerate cases make a zero contribution. After trying each canonical region for this a and b, we can combine the trees to report T_{ab}, etc., breaking ties arbitrarily. (If it is desired to abide by the boundary or leftness rules then ties should be broken accordingly; we do not make use of these rules though for simple Steiner trees.) This completes the sketch of the general step of the algorithm.

3.3. Using Minimal Canonical Regions

What is the time complexity of this approach, not counting the preprocessing step? Each sweep does $O(n)$ iterations of the general step. Each such step exhausts all $O(n^3)$ choices of tabs (defining a canonical region), leading to $O(n^4)$ total time. We now show how to implement this in $O(n+k^2)$ time.

This lemma states that we can restrict our attention to *minimal canonical regions.*

Lemma 8: If canonical region F is contained in canonical region E then the ab-trees constructed for E will be no shorter than those for F.

Proof: This follows immediatedly since each ab-tree consistent with E is also consistent with F. □.

To characterize a minimal canonical region we first note that its top tab f_2 cannot be slid down. It follows that f_2 intersects an inner boundary corner. Further its left tab f_1 cannot be slid to the right so either it intersects an inner boundary corner or it intersects a itself. Similarly its right tab f_3 cannot be slid left and intersects an inner boundary corner or b. Of course if f_2 is the next grid line above ab then it is not slid down, even if possible, since that creates an empty region.

We can make further observations about these tabs. Given f_2, let α be its left endpoint and let β be its right endpoint. If the region is a minimal canonical region, f_1 must be incident to the leftmost of a and α, and f_3 must be incident to the rightmost of b and β. The key observation is that we need only try $O(k)$ placements of f_2 to exhaust all the minimal canonical regions. In particular we successively place f_2 so that it intersects each inner boundary corner above ab, and, finally, we place it on the next grid line above ab. The latter case is the *covering line case.* Once f_2 has been placed then f_1 and f_3 are forced into positions that can be determined in constant time.

This approach seems to take $O(kn)$ time. In most cases, however, only one placement of f_2 needs to be tried. Suppose that neither a nor b is an inner boundary corner. In that case the boundary B_{ab} has vertical edges intersecting both a and b. It follows that there is only one minimal canonical region, the covering line case. Since this case can be handled in constant time and there are only $O(k)$ cases where ab intersects an inner boundary corner, the algorithm takes $O(n+k^2)$ total time. As we will see the preprocessing can be done within the same time bound leading to an overall $O(n+k^2)$ time algorithm.

3.4. The Three-sided Case

We can make an improvement for certain boundaries B that will occur in the next section. Let the portion of the boundary properly between two tabs be called a *staircase*. While every rectilinearly convex region has four tabs, some of its four staircases may be empty. We say a boundary is *three-sided* if at least one staircase is empty.

For the three-sided case, we can prove a restriction on the placement of f_2 by a simple case analysis, when f_2 is not the covering line. Let e_a be the boundary line along B between a and the inner boundary corner to its left, and let e_b be a similar line to the right of b; either could be of zero length. Then either f_2 intersects f_1 above e_a or it intersects f_3 above e_b.

We can effectively exhaust all placements of f_2 by trying each placement of f_1 or f_3 that intersects e_a or e_b, respectively. Recall that if, say, f_1 intersects f_2, it must happen at an inner boundary corner. Therefore the total number of placements of f_2 is the number of inner boundary corners above both e_a and e_b, plus the covering line case. It follows that the total time for three-sided boundaries is $O(n)$ plus the cost of preprocessing.

3.5. Preprocessing

We begin by regarding the terminal points on the boundary as being vertices on a simple cycle; call these *terminal vertices*. Further the $O(k)$ inner boundary corners and the points orthogonally opposite these corners are also vertices on this cycle, placed in their correct relative positions relative to the terminal vertices; call these *new vertices*. Note that a terminal vertex and a new vertex can coincide. The length of an edge in the cycle is just the boundary length between them.

To compute b_{xy}, for any two vertices x and y, we begin by computing the cumulative length function around the cycle, starting at any point. In constant time each b_{xy} can be computed by simple subtraction.

To compute f_{xy} and f_{yx}, for any two new vertices x and y, we first find for each new vertex the nearest terminal vertex, in both directions. Also for each terminal vertex we record its nearest new vertex, in both directions. This is done by scanning the cycle. Since for any new x and y the gaps nearest to x or y can be found using the b's above, f_{xy} and f_{yx} can now be computed in constant time.

The computation of g_{xy}, for any new x and y, is more complicated, but can be mapped to a solved problem. In particular, consider an array of n numbers and a set of m intervals (contiguous subarrays). The problem is to report the maximum array element in every interval. There is an $O(n+m)$ time algorithm known [5]; it constructs a ''Cartesian tree'' and performs constant-time nearest common ancestor queries on the tree for each interval.

To map our problem to the above problem we can form an array of the inter-terminal lengths. (Actually it is simplest to unroll the cycle twice to form an array twice as long, to avoid any wraparound.) Then to compute g_{xy} we find the nearest terminals to x and y, between x and y, and use these as the endpoints of an interval. This interval query returns the largest unbroken gap between x and y. Hence each of the $O(k^2)$ quantities g_{xy} can now be computed in constant time given the b's above, leading to $O(n+k^2)$ time for preprocessing.

4. The Basic Algorithm

In this section, we present an algorithm to find a rectilinear Steiner minimal tree for k-extremal point set S. Theorem 1 implies that, given S, the topologies of rectilinear Steiner minimal trees can be restricted to six cases, possibly occurring many times in B. Simple Steiner trees were considered in the previous section. In this section we give an algorithm for discovering any other trees. In particular, we find trees containing at least one complete line with incident lines or at least one complete corner; these are called nontrivial topologies.

We use a dynamic programming approach: given a particular nontrivial topology and the sides it hits, we restrict its intersection points and then decompose the convex figure into at most four convex subproblems. Each subproblem involves a smaller area, but possibly more points, than the original problem. Subproblems will be divided along special grid lines called *blue lines*. We will show that there are $O(k)$ blue lines, and since each subproblem is bounded by at most four blue lines, the number of subproblems is $O(k^4)$. To combine subproblem solutions, a method introduced by Aho, Garey, and Hwang [2] is described in Lemmas 9 and 10. As mentioned in the previous section, back-pointers can be maintained to rebuild the actual tree. We indicate how to reconstruct the tree at the end of Section 5. The cost of the algorithm will be seen to be $O(k^8 + k^4 n)$ time, which is linear in n for constant k. This algorithm is improved to $O(k^4 n)$ in the next section.

4.1. Preliminaries

Let l be a complete vertical interior line. Let $S_1 \subseteq S$ be the terminal set to the left of and including l, and let $S_2 \subseteq S$ be the terminal set to the right of and including l. Let B_1 be the boundary defined by l and the portion of B to the left of l, and let B_2 be the analogous boundary to the right.

Lemma 9: Suppose some Steiner minimal tree for S contains a vertical line l. Then the union of the following two Steiner trees is a Steiner minimal tree T for S containing l: a Steiner tree T_1 for S_1 of least weight constrained to contain l, and a similar tree T_2 for S_2.

Proof: Choosing a tree of greater weight in place of T_1 or T_2 contradicts the minimality of T. □

Note that T_1 and T_2 may not be Steiner minimal trees for S_1 and S_2, respectively.

Lemma 10: A Steiner minimal tree for the terminal set consisting of S_1 and all Hanan grid points along l can be transformed by transplants to a Steiner tree T_1 for S_1 of least weight constrained to contain l.

Proof: Consider adjacent Hanan grid points x and y in B_1 along l. Any Steiner minimal tree T_1 either contains the edge between x and y, or there is some other path from x to y. This path must use an edge e between two grid points x' and y', where the distance from x' to y' equals the distance from x to y. If e is transplanted to the boundary edge between x and y, total length remains the same. Repeating this gives a minimum length rectilinear Steiner tree for S_1 containing l. □

A similar procedure can be developed to find a constrained Steiner minimal tree where two interior lines l and s are constrained to occur, provided l and s are orthogonal. Note that the number of terminals on the boundary is at most $2n$, since a rectilinearly convex boundary can intersect at most $2n$ grid points.

Our algorithm subdivides B by attempting to find the location of nontrivial topological *backbones*. For a complete line l with incident edges, the backbone is l, and for a complete corner, the backbone consists of the legs of the corner and the lines on each leg closest to the corner-vertex (if these lines exist). The backbones will be used to define the four or fewer recursive subproblems, each bounded by blue lines and portions of the original boundary.

Blue grid lines are colored by examining inner boundary corners. Consider an inner boundary corner b, as depicted in Figure 4.1. For each choice of b, at most ten Hanan grid lines are marked blue; these lines are the dotted lines in Figure 4.1. Reorient B by rotation so that it appears as in Figure 4.1. Point b_1 is the first terminal below b, b_2 is the first terminal to the right of b, d_1 is the first terminal on A below b, a_1 is the first terminal on A above b, e_1 is the second terminal on A above b, and so on. (In this specification, if there is a terminal at b, $b = b_1 = b_2$, if there is a terminal a on A across from b, $a = a_1 = d_1$, and the same for C.) Note that $O(k)$ blue lines are colored for

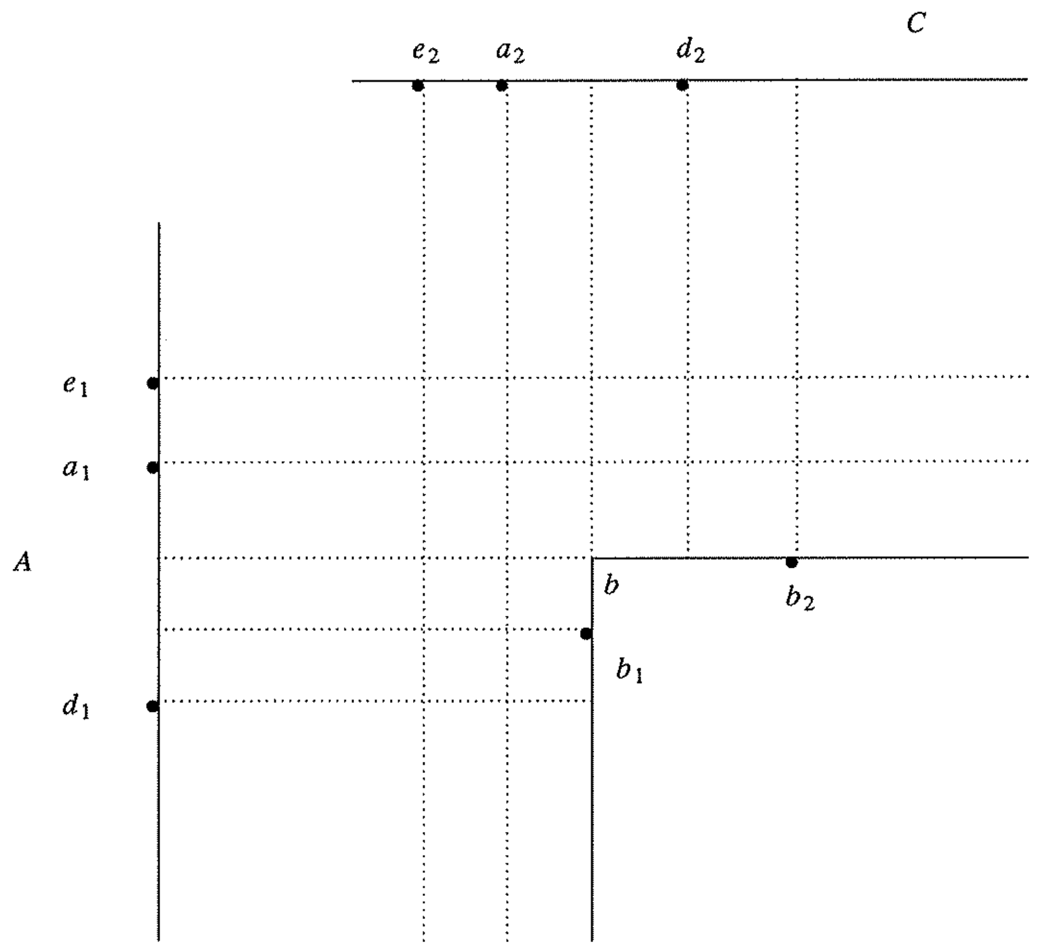


Figure 4.1 — Blue Lines

boundary B.

Theorem 2: There is a Steiner minimal tree where all nontrivial topological backbones appear on blue lines.

Proof: Consider an optimal Steiner minimal tree containing a nontrivial topology; Theorem 1 implies that nontrivial topological instances are of types 2-6, each of which is considered below. We show that any nontrivial topological instance can be slid so that its backbone lies only on blue lines. The resulting tree may not be optimal but does have minimum length.

The five nontrivial topological instances of Theorem 1 can be regrouped as four complete corners and two complete lines. This grouping is based on the parity of the number of incident edges, and the cases are depicted in Figure 4.2. If a complete corner is present, there is an even-even case, an even-odd case, an odd-even case, and an odd-odd case. Because of Lemma 5, we rename these cases to the more intuitive zero-even, zero-odd, one-even, and one-odd cases, respectively. If a complete line l is present, the ~~number of edges incident to l may be even or~~ odd (the cross is a special instance of the even case). Not all orientations are depicted in Figure 4.2. However, we assume the depicted orientations are the only ones in our exposition and appeal to symmetric arguments for the undepicted orientations.

Let T be an optimal Steiner tree containing a nontrivial topological instance. We show that each nontrivial topological instance can be placed on blue lines by shifting.

Case 1: A complete line with an odd number of incident edges.

Suppose that a complete line l intersects boundary edges U and D. Let the edge incident to l and closest to U be e, and suppose that e intersects boundary edge L. Assume that l is slid as far to the left as possible without increasing boundary length so that it intersects U at point a and D at point b. Let c be the left endpoint of e. We first claim that d is an inner boundary corner. If b is not d, then there are no Steiner tree edges incident to b on the left, or l would not be as far to the left as possible. If d is an outer boundary corner, there must be a terminal at d. This terminal must be connected to l by a path using the lowest edge f incident to l. Convexity implies that the left endpoint of f cannot be to the right of d, so a portion of f can be transplanted to D, contradicting the optimality of T.

We also claim that either a is the leftmost terminal on U to the right of d, b is the leftmost terminal on D to the right of d, or b is d. To see this, suppose b is not d. There can be no leftward tree edge out of a or b, or else l can be slid further to the left without increasing length or decreasing boundary length. Either a or b is a terminal; otherwise there are rightward edges incident to a and b, and l can be slid right to decrease length. Therefore, suppose a is a terminal and there is another terminal a' on U to the left of a and to the right of d. Since a' is not connected to l from a it must be connected to l by a path including edge e. Convexity implies the left endpoint c of e cannot be to the right of d, so a new edge connecting a and a' can be added, e can be deleted, and the resulting tree either has less total length or greater boundary length, contradicting the optimality of T. If a is not a terminal but b is, a similar argument using the bottommost incident edge on l can be given. Note that the grid lines containing the three possible locations for l were colored blue since d is an inner boundary corner.

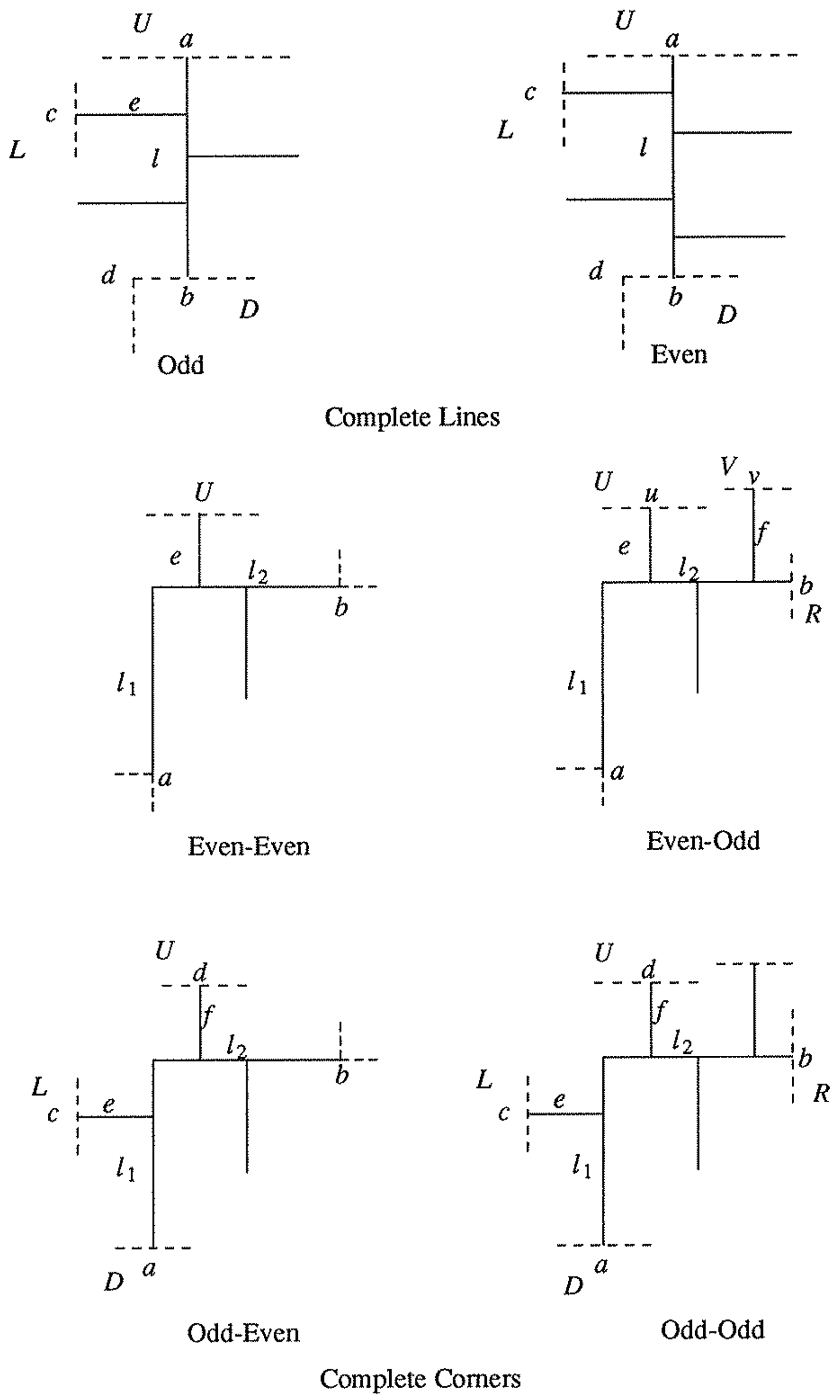


Figure 4.2 — Nontrivial Topologies

Case 2: A complete line with an even number of incident edges.

Recall that in this case, we assume a nonzero number of incident edges. Suppose that there is a complete line l intersecting boundary edges U and D at a and b, respectively. Reorient the figure so that the left endpoint of U is not to the right of the left endpoint d of D. As with case 1 above there are a limited number of ways for l to connect to U and D. In fact, they are the same three connections as in case 1 and a similar proof can be used here.

Case 3: Zero-Even Corners.

Consider an zero-even corner with legs l_1 and l_2. Since each leg is incident to an even number of edges, Lemma 5 implies one of the legs has no incident edges. We can reorient B as depicted in Figure 4.2, where only l_2 may have incident edges. Let l_1 intersect B at point a and l_2 at point b. Let the edge incident to l_2 closest to the corner-vertex be called e and suppose it intersects B along boundary edge U (e may not exist). We claim that points a and b are inner boundary corners of B, and if e exists, it can be slid so that $e \cap U$ is either the inner boundary corner of U or the leftmost terminal on U strictly to the right of a.

To prove this, note the boundary length constraint implies that a and b must be corners of B. If b was not a corner, for instance, l_2 could be slid downwards to increase boundary length or decrease total length. For edge e, prior to reorientation, Lemma 4 implies that e is either incident to an inner boundary corner or a terminal. If e is incident to an inner boundary corner, we are done, so suppose that e is incident to a terminal u which is not an inner boundary corner. In the orientation of Figure 4.2, u must be to the right of a. If u is not the leftmost terminal on U strictly to the right of a, there is some other terminal u_l that is. We claim that Steiner tree edges along U connect u to u_l, so e can be slid to the left until it intersects u_l. To prove this claim, suppose that u and u_l are not connected along U. Then u is connected to u_l by a path including l_2, l_1, and a, so a portion of l_2 can be transplanted to U to increase boundary length.

We now show that the grid lines containing edges l_1, l_2, and e are blue. Both l_1 and l_2 lie on blue grid lines since they are incident to inner corners. For e, if the left endpoint c of U is to the left of a, then the grid line incident to u_l was colored blue in the first phase. On the other hand, if c is to the right of a, c must be an inner boundary corner or u_l would lie at an outer boundary corner, and since there are Steiner tree edges from u to u_l, e could be slid to increase boundary length, contradicting the selection of T. Since c is an inner boundary corner, u_l was colored blue.

Case 4: Zero-Odd Corners.

Consider an Zero-odd corner with legs l_1 and l_2; recall that in this case one of the legs has no incident edges. Reorient B so that it appears as depicted in Figure 4.2, let the leftmost line incident to l_2 be called e, and let the rightmost line incident to l_2 be called f. By Lemma 2, both e and f point upwards.

Let the intersection point of l_1 and B be a, and let the intersection point of l_2 and B be point b lying on boundary edge R. Without loss of generality, suppose that l_2 is slid upwards as far as possible without increasing total length or decreasing boundary length. Let e intersect boundary edge U at u, and let f intersect boundary edge V at v.

By sliding edges, we can show that point a is an inner boundary corner, and point b is either the upper endpoint of R (which must be an inner boundary corner) or the highest terminal on R below u. Further, u can be taken to be either the inner boundary corner of U or the leftmost point on U strictly to the right of a.

This claim is proved in a similar fashion as the previous case. The boundary length constraint implies that a must be a corner in B. For b, since l_2 is slid upwards as far as possible, there is no Steiner tree edge incident to b from above. If b is not the upper endpoint of R then b must be a terminal. Otherwise a downward tree edge incident to b would permit l_2 to be shifted down to decrease length. Suppose further that b' is a terminal on R above b and below u. Note b' must be connected to l_2 by a path through edge f. Consequently, f can be transplanted to the boundary along R, increasing the use of boundary length. (By convexity b' cannot be above v and below u.)

Using the same argument as in case 3, e can be slid so that u is either at the inner boundary corner of U or the leftmost terminal on U strictly to the right of a, and one can show the grid lines containing lines l_1, l_2, and e are blue by the method used in case 4.

Case 5: One-Even Corners.

Suppose we have an one-even corner with legs l_1 and l_2. Notice that if we flip the corner so that the vertical leg hits b, we get an zero-odd corner which was considered in case 4.

Case 6: One-Odd Corners.

The last case to consider is the one-odd corner. Let l_1 and l_2 be the legs of the corner. Lemma 5 implies that one of the legs has only one incident edge; we reorient B as in Figure 4.2, so that the leg with one incident edge is vertical. Denote the edge incident to l_1 by e, and let the leftmost edge incident to l_2 be f. Let l_1, l_2, e, and f intersect boundary edges D, R, L, and U, respectively, where $a = l_1 \cap D$, $b = l_2 \cap R$, $c = e \cap L$, and $d = f \cap U$. As in the previous cases a, b, c and d can be restricted to a small number of positions after the selection of D, R, L, and U.

Without loss of generality, assume that l_1 is slid as far left as possible and that l_2 is slid as far upwards as possible without decreasing boundary length. Point a is either the inner boundary corner of D or the leftmost terminal on D to the right of L. Point b is either the inner boundary corner of R or the highest terminal on R below U. Further, we can take c to be either the inner boundary corner of L or the highest terminal on L below b, and we can take d to be either the inner boundary corner of U or the leftmost terminal on U strictly to the right of a. As before, the grid lines containing lines l_1, l_2, e, and f can be shown to be blue (note that the grid lines containing e and f are colored blue since, at worst, each is incident to the second terminal "opposite" an inner boundary corner).
□

In cases 1 and 2, it can also be shown that the edge e closest to U can be made to lie on a blue line. The proof is left to the reader.

4.2. Description of Algorithm

We use dynamic programming to devise an efficient algorithm for k-extremal point sets. For a rectilinear convex hull with boundary B, a Steiner minimal tree is either simple or contains one or more nontrivial topological instances. Theorem 2 implies that if there is an optimal Steiner tree containing nontrivial topological instances, there is a Steiner minimal tree whose nontrivial topological backbones lie on blue lines. We describe the algorithm recursively; in the dynamic programming algorithm, subsolutions are computed in order of increasing area and a table look-up is used. There are seven cases, case 0 being the simple Steiner tree, and cases 1-6 being the cases enumerated in Theorem 2. In the recursive cases 1-6, subproblems are split at blue grid lines and are combined by the algorithm in the proof of Lemma 10; details appear at the end of Section 5. There are two important properties of the subproblems: blue lines only appear on tabs in recursive subproblems, and no new inner boundary corners are created, so the original set of $O(k)$ blue lines is sufficient.

Case 0

A Steiner minimal tree may be a simple Steiner tree for which an algorithm was given in Section 3.

Cases 1 and 2.

In the proof of Theorem 2, it is shown that the nontrivial topological backbone, i. e. the complete line, must lie on a blue grid line. There are a total of $O(k)$ candidate blue grid lines, each of which splits the problem involving boundary B into two convex subproblems of smaller area.

Case 3.

The placement of an zero-even corner depends on the choice of a, b, and $e \cap B$. (Note that $e \cap B$ need not lie on the boundary edge above a.) For each choice, there are three subproblems to resolve: Problem I is bounded by e and l_2, problem II is bounded by l_1 and l_2, and problem III is bounded by l_1, part of l_2, and e. For problems I and II, the blue lines are tabs. For problem III, we first flip the corner-vertex $l_1 \cap l_2$; in this subproblem, the blue lines are tabs and no new inner boundary corners are induced. This flip is justified in Lemma 12 below.

Cases 4 and 5.

Similar to Case 3.

Case 6.

Unlike cases 3, 4 and 5, the placement of one-odd corners depends on the choice of four parameters, namely a, b, c, and d, and there are four subproblems. Problem I is bounded by lines f and l_2, problem II is bounded by l_1 and l_2, problem III is bounded by l_1 and e, and problem IV is bounded by e, l_1, l_2, and f. In problem IV, the corner is flipped before the subproblem is solved.

4.3. Analysis of Algorithm

We now prove that the basic algorithm runs in $O(k^8 + k^4 n)$ time and is correct.

Lemma 11: No subproblem is bounded by more than four blue grid lines.

Proof: Blue lies only appear as portions of tabs in recursive subproblems. □

Corollary 1: There are a total of $O(k^4)$ subproblems.

In order to prove correctness, we must justify the corner flips in cases 3-6. Let e_1 and e_2 be two boundary edges intersecting at inner boundary corner b. Assume that no original terminals lie on b or in the relative interior of e_1 or e_2. Let R be the rectangle bounded by e_1 and e_2, and let the other edges of R be e'_1 and e'_2, appearing clockwise as e_1, e_2, e'_2, and e'_1. Let S_3 be the set of terminals consisting of S and all the grid points along e_1 and e_2, and let S'_3 be the set of terminals consisting of S and all the grid points along e'_1 and e'_2.

Lemma 12: Suppose there exists a Steiner minimal tree for S_3 containing e_1 and e_2 and having no interior line incident to b or properly incident to either e_1 or e_2. Then the length of a Steiner minimal tree for S_3 is the same as the length of a Steiner minimal tree for S'_3.

Proof: Let T be the asserted tree for S_3. Let T' be a Steiner minimal tree for S'_3. Note that by Lemma 10, we can assume that T' contains all of e'_1 and e'_2. Then *length* $(T') \geq$ *length* (T), since the lines e'_1 and e'_2 can be flipped to e_1 and e_2, giving a feasible Steiner tree for S_3 (it may be that some cycles may be induced as well, so certain edges would be removed). Second, as there is a Steiner minimal tree T where e_1 and e_2 appear with no properly incident lines, e_1 and e_2 can be flipped, giving a feasible solution for S'_3 containing e'_1 and e'_2, so *length* $(T) \geq$ *length* (T'), and the lengths of T and T' must be equal.

Further note that except at the intersection points of e_1 and e'_1 and e_2 and e'_2, no edges in T' are incident to the closed rectangle R. This is because such an edge implies *length* $(T') >$ *length* (T), a contradiction. □

In the corner cases described in Theorem 2, the problem solved when the corner is flipped is a S'_3 problem; Lemma 12 states that the resulting Steiner minimal tree can be trivially transformed into a Steiner minimal tree for the S_3 problem which is needed to apply Lemma 10.

Theorem 3: A Steiner minimal tree for a k-extremal point set can be found in $O(k^8 + k^4 n)$ time.

Proof: Theorem 1 implies that optimal Steiner trees contain only certain topologies. Corollary 2 implies that only $O(k^4)$ subproblems must be considered, each of which has at most $2n$ terminals. In each subproblem, there is a Steiner minimal tree computation. For each of the $O(k^4)$ subproblems considered in increasing area, a minimal simple Steiner tree is found in $O(k^2 + n)$ time, and the $O(k^4)$ decompositions described in cases 1-6 are considered. For each decomposition, the subproblems have smaller area, so a minimal tree is already stored. Lemmas 9, 10, and

12 imply that the subproblem solutions can be combined to give a Steiner minimal tree. The cost of this algorithm is therefore $O(k^8 + k^4 n)$. □

If k is a constant, the algorithm is linear in n and asymptotically optimal, but if k is near n, the algorithm uses $O(n^8)$ time, which is slower than the algorithms of Bern and Provan.

5. The Improved Algorithm

The algorithm from the preceding section can be improved to $O(k^5 + k^4 n) = O(k^4 n)$ time. The most time-consuming cases are the corner cases 3-6. In the zero-even, zero-odd, and one-even corners, as many as three boundary intersections need to be selected, and in the one-odd corners, as many as four boundary intersections must be chosen. It also appears that the algorithm to compute minimal simple Steiner trees is more efficient, after preprocessing, when restricted to the three-sided case. To get a more efficient algorithm, we define several classes of subproblems.

5.1. Preliminaries

Let e_1 and e_2 be two perpendicular edges intersecting at point c and intersecting B at points a and b, respectively. Let R be the closed rectangle determined by e_1 and e_2; that is, three of R's vertices are a, b, and c.

Lemma 13: R cannot contain an outer corner of B.

Proof: Suppose R contains an outer corner of B. There must be a terminal d at this outer corner which is either connected to c by a path through a or a path through b. If the path is through a, e_1 can be transplanted to connect d to e_2, increasing boundary length or total length, and a symmetric argument can be used if c is connected to d using a path through b. □

A simple corollary is used several times below: The boundary from a to b contains at least two edges, including a tab, in addition to the edges containing a and b.

Now consider a corner C with legs l_1 and l_2 and edges e and f incident to l_1 and l_2, respectively. Assume that e and f are the edges closest to the corner-vertex c. Define C to be a *solid corner* if e or f cannot be slid so that c becomes a T-vertex. Let R be the closed, 6-gon bounded by l_1, l_2, e, f, a line segment perpendicular to e on the same side of e as c and intersecting e at its endpoint opposite l_1, and a line segment perpendicular to f on the same side of f as c and intersecting f at its endpoint opposite l_2.

Lemma 14: Let C be a solid corner as described above. Some portion of the boundary between e and f (i. e., on the same side of e as c and on the same side of f as c) must lie outside R.

Proof: Let the portion of the boundary be X and suppose to the contrary that all of X lies inside R. Recall X must contain an outer boundary corner with a terminal x. As x must be connected to the Steiner tree, it is either connected to c by a path through e or through f, say e. As the entire subtree connecting c to x lies inside R, e can be slid without increasing length so that one of its endpoints is c, contradicting the assumption it is a solid corner. □

In the analysis in Section 4, solutions corresponding to non-solid corners would be discovered in the subproblems for cases 1 and 2. Therefore, only solid corners need be explicitly considered in cases 3-6. In the remainder of the section, corners are assumed to be solid.

Lemma 13 and Lemma 14 can be used restrict the locations of tabs in the corner cases 3-6. The locations of tabs are marked with T's in Figure 4.3, and the proofs are left to the reader. Notice that wherever a T is marked, all of the tab must appear in the region.

Define a *paired tab* to be a structure where two tabs intersect at a single point, and define a *triple tab* to be a structure where three tabs intersect at two points.

Lemma 15: If a boundary has a paired tab, the one-odd corner cannot occur.

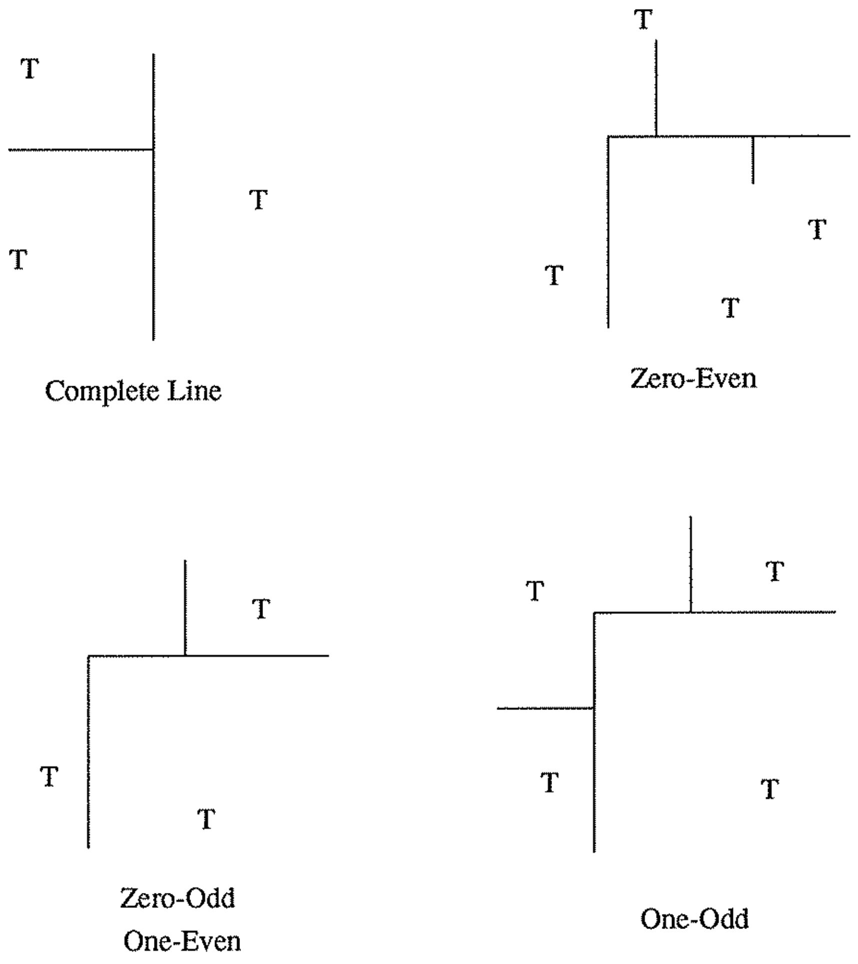


Figure 4.3 — Locations of Tabs

Proof: Referring to Figure 4.3, exactly one tab must appear in each of the four regions depicted. □

Lemma 16: If a boundary has two paired tabs the one-even and zero-odd corners cannot occur.

Proof: Each instance requires at least three disjoint tabs. □

Lemma 17: If a boundary has a triple tab, no corner can occur.

Proof: Lemma 15 implies that one-odd corners cannot occur. The one-even and zero-odd cases require at least three disjoint tabs, only two of which occur when a triple tab is present. For an zero-even corner, exactly two tabs must be on either side of the corner. □

Recall that when a corner is present, at most four subproblems are defined. Each subproblem is bounded by a paired tab, implying the following result:

Lemma 18: At most one one-odd corner can appear, and if one does appear, no other corners appear. Further, at most two one-even or zero-odd corners can appear.

Proof: Suppose a tree contains an one-odd corner. Then each of the resulting subproblems has a paired tab, and Lemma 15 implies the first assertion. Further, since only one original tab is in each of the four subproblems, each of the subproblems is bounded by a triple tab, implying that no other corners are present by Lemma 17.

Now suppose a tree contains an one-even or zero-odd corner. It is possible that a subproblem contains two disjoint tabs and a single paired tab. (All other regions contain triple tabs.) If this subproblem contains an one-even or zero-odd corner, its induced subproblems are either bounded by triple tabs or two paired tabs, implying by Lemmas 16 and 17 that such an instance cannot occur in any of the induced subproblems. □

5.2. Description of the Improved Algorithm

In order to state the improved, $O(k^4 n)$ time algorithm, we define four new special-case algorithms:

A. Three-sided with no corners.

B. Arbitrary with no corners.

C. Zero-edge corners. That is, the solution contains one or more zero-odd, one-even, or zero-even corners and no one-odd corners.

D. One-odd corners. That is, the solution contains exactly one one-odd corner.

The special-case algorithms B, C, and D partition the set of possible Steiner minimal trees: Either the Steiner tree contains an one-odd corner or not. If not, it either contains a zero-edge corner or no corner at all. In the next few paragraphs, we describe and analyze each of the four algorithms. The overall structure uses dynamic programming to avoid recomputing subsolutions, where we assume that solutions are computed in order of increasing number of interior Hanan gridpoints.

A: If the figure is three-sided and contains no interior corners, a Steiner minimal tree can appear in one of two forms. First, it may be a minimal simple Steiner tree. Aside from preprocessing, this tree can be computed in $O(n)$ time. Otherwise, the Steiner minimal tree contains a complete line. There are $O(k)$ possible blue lines for the topological backbone position, and each subproblem is also of type A. Aside from preprocessing, the total time per subproblem is therefore $O(n+k)$, and since there may be as many as $O(k^4)$ subproblems, algorithm A takes $O(k^4 n)$ time.

Regarding preprocessing for b_{xy}, f_{xy}, and g_{xy} queries, it can be shown that only $O(n)$ extra processing time is needed after an initial expenditure of $O(k^2+n)$ time to support constant time queries. The most difficult queries are the g_{xy}'s. Since no inner boundary corners are created, the original set of $O(k)$ "new" vertices suffice for the subproblems, though the boundary has changed. Specifically, each subproblem consists of the original boundary edges and at most four "blue" grid lines, appearing as parts of tabs. If one projects the original terminal set S onto the y-axis, the projection of a blue grid line is an interval along the y-axis. Two Cartesian trees can be built, one for the x-coordinates of S and one for the y-coordinates; these trees can be used to support the g_{xy} queries in the subproblems. The details are left to the reader.

B: Consider a figure with exactly four staircases; if it has fewer algorithm A can be used. Either the Steiner tree contains a complete line with incident edges or not. If not, we run the minimal simple Steiner tree algorithm, expending $O(k^2+n)$ time. If it contains a complete line l, choose one of $O(k)$ locations for placement, and assuming without loss of generality l is vertical, choose one of $O(k)$ locations for the horizontal edge closest to the upper endpoint of l. This gives three subproblems, each of which contains a distinct tab and therefore at most three staircases (see Figure 4.3). Consequently, each of the subproblems is amenable to algorithm A. Starting with the original convex hull, algorithm B is called once and takes $O(k^2)$ time.

C: A zero-edge corner is one with at least one leg with no incident edges. Though one-even corners can only appear at most twice, zero-even corners can appear many times. By a tab analysis, one can show that the zero-edge corners are arranged so that if corners are visited in a left-to-right sweep, some number of zero-even corners are bracketed with at most one one-even or zero-odd corner on each end.

In algorithm C, all subproblems save the original are bounded by exactly two blue lines forming a paired tab. Therefore, consider subproblems bounded by a paired tab in the upper left end; the other cases are symmetric. There are $O(k^3)$ locations for zero-edge corners in this subproblem, as there are as many as three edges to anchor on blue lines: two legs and the edge nearest the corner-vertex.

For one of these placements, no other corner appears between the paired tab at the upper left end and the two legs of the corner. If the corner is zero-even, then it can be flipped so that it bends in the same way as the upper left paired tab; three subproblems are defined, two of which are of type A and one of which is a smaller instance of type C. If the corner is zero-odd, then it also defines three subproblems except that by Lemma 16, all of the subproblems are instances of A. In summary, there are $O(k^2)$ subproblems, each solved in $O(k^3)$ time.

To process the original convex hull, $O(k^3)$ placements for a zero-edge corner are tried. Suppose the corner bends to the right with the vertical leg below the horizontal. Then two subproblems are of type A and the other is as described above. Algorithm C takes $O(k^5)$ time.

D: If an one-odd corner is present, Lemma 18 implies that it is the only corner appearing in the Steiner tree. Therefore, starting with the original convex hull, the $O(k^4)$ placements of the one-odd corner are checked. Each subproblem is of type A, so algorithm D takes $O(k^4)$ time.

Figure 4.4 summarizes the flow of control and interdependence of the algorithms above. There are seven algorithms in total, (1) the algorithm to find the convex hull of a k-extremal point set, EXTREMAL, (2) an algorithm for 3-sided simple Steiner trees, 3-SIDED SIMPLE, (3) an algorithm for general simple Steiner trees, SIMPLE, and algorithms (4) A, (5) B, (6) C, and (7) D. (In the figure, C is split into two parts, C and C'. In algorithm C', a sequence of zero or more zero-even corners are placed, followed by at most one one-even or zero-odd corner.) A Steiner minimal tree for extremal point set S is found by calling EXTREMAL(S).

5.3. Reconstruction of the Steiner Tree

We now show how to combine subproblems to construct a Steiner minimal tree. There are several ways to do this; we outline one. The algorithm described above gives a template to rebuild the Steiner tree. There is some number of subsolutions which must be combined along certain dividing lines, where the Hanan gridpoints along each dividing line are terminals. Note that the dividing lines contain at most two orthogonal blue lines to which (the extended) Lemma 10 can be applied. We now sketch an efficient implementation of the construction used to prove Lemma 10.

The basic procedure is to take a Steiner minimal tree T and transform it into another Steiner minimal tree T', where T' contains a line l, all of whose Hanan gridpoints are terminals. If T already contains l, we are done. Otherwise, we must search for edges to transplant to l. Assume that T is represented as an adjacency list. It is well known that the maximum number of Steiner points is $n-2$, where n is the initial number of terminals [10]. (Note that edges connect two endpoints, which are either Steiner points or terminals. They need not be on the same horizontal or vertical line.) The graph therefore has complexity $O(n)$.

We perform a scan of the terminals of l, marking an adjacent pair if there is a Steiner edge between them. We then perform a depth-first search on the Steiner tree to discover the path between the two extreme terminals of l. When we visit an edge e in the path which is parallel to l and spans the grid lines between an unmarked adjacent pair x and y, we transplant e to the edge between x and y. Note that length constraints imply that the length of e and the distance from x to y must be equal. The adjacent pair x and y is then marked.

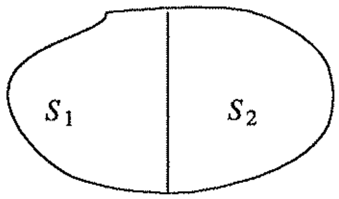
After edges are transplanted so that l is included in the tree, l is coalesced by removing terminals of degree 2. (Note that length constraints imply that l is the only edge that needs to be coalesced, i.e., no edges are fragmented in this procedure.) This algorithm implements Lemma 10 in $O(n)$ time. There are a total of $O(k)$ subproblems that need to be combined in this way, as Theorem 2 implies that at most $O(k)$ nontrivial topological instances can appear in an optimal Steiner minimal tree. The algorithm above reconstructs a Steiner minimal tree in $O(kn)$ time.

This proves the main result of the paper:

Theorem 4: A Steiner minimal tree for a k-extremal point set can be found in $O(k^4 n)$ time.

$$\begin{tikzcd}
 & \text{Extremal }(S) \arrow[dl] \arrow[d] \arrow[dr] & \\
B(S) & C(S) & D(S)
\end{tikzcd}$$

$$\begin{tikzcd}
 & A(S) \arrow[dl, "\text{simple}"'] \arrow[dr, "\text{nonsimple}"] & & \\
\text{3-sided simple}(S) & & \text{Select line} \arrow[dl] \arrow[dr] & \\
 & A(S_1) & & A(S_2)
\end{tikzcd}$$



$$\begin{tikzcd}
 & B(S) \arrow[dl, "\text{simple}"'] \arrow[dr, "\text{nonsimple}"] & & \\
\text{simple}(S) & & \text{Select line} \arrow[dl] \arrow[dr] & \\
 & A(S_1) & & A(S_2)
\end{tikzcd}$$

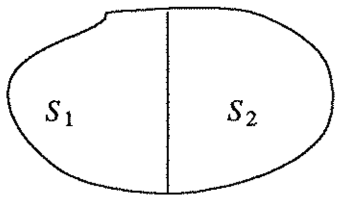
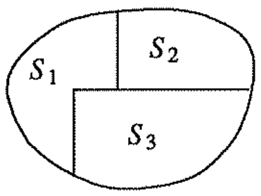
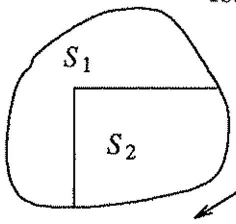


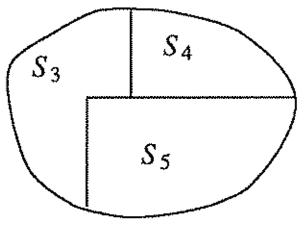
Figure 4.4a — Summary of Improved Algorithm

$$\begin{tikzcd}
 & C(S) \arrow[dl, "\text{1st corner is zero-even}"'] \arrow[dr, "\text{1st corner is one-even or zero-odd}"] & & \\
C'(S) & & \text{place corner} \arrow[dl] \arrow[d] \arrow[dr] & \\
 & A(S_1) & A(S_2) & C'(S_3)
\end{tikzcd}$$



$$\begin{tikzcd}
 & & & C'(S) \arrow[dll, "\text{1st corner is zero-even}"'] \arrow[d] \arrow[drr, "\text{1st corner is one-even or zero-odd}"] & & & \\
 & \text{place corner} \arrow[dl] \arrow[dr] & & A(S) & & \text{place corner} \arrow[dl] \arrow[d] \arrow[dr] & \\
A(S_1) & & C'(S_2) & & A(S_3) & A(S_4) & A(S_5)
\end{tikzcd}$$





$$\begin{tikzcd}
 & & D(S) \arrow[d] & \\
 & & \text{place corner} \arrow[dll] \arrow[dl] \arrow[d] \arrow[dr] & \\
A(S_1) & A(S_2) & A(S_3) & A(S_4)
\end{tikzcd}$$

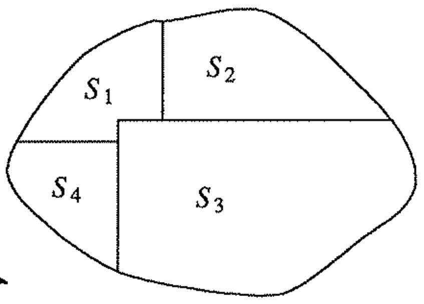


Figure 4.4b — Summary of Improved Algorithm (continued)

6. Final Remarks

In this paper, we presented an algorithm to find a rectilinear Steiner minimal tree for a k-extremal point set of size n which uses $O(k^4\ n)$ time. For constant k, this is an asymptotically optimal linear-time algorithm, and it is the fastest known algorithm for any value of k. We note that in many practical applications such as VLSI design, k is a constant.

There are several open problems to mention. First, can the complexity be reduced? One promising approach is to exploit the coherence of subproblems; some subproblems are very similar to others. Another question is whether the complexity of computing simple Steiner trees for the four staircase problem (after preprocessing) can be reduced. Finally, one may want to prove more powerful combining results than Lemmas 9, 10, and 12. Such results could prove useful to other Steiner tree problems.

7. References

1. P. K. Agarwal and M. T. Shing, Algorithms for the Special Cases of Rectilinear Steiner Trees: I. Points on the Boundary of a Rectilinear Rectangle, Tech. Report TRCS86-17, Univ. of California at Santa Barbara, 1986.

2. A. V. Aho, M. R. Garey and F. K. Hwang, Rectilinear Steiner Trees: Efficient Special-Case Algorithms, *Networks,* **7**, 1977, pp. 37-58.

3. M. W. Bern, *Network Design Problems: Steiner Trees and Spanning k-Trees*, PhD Thesis, Univerisity of California at Berkeley, 1987.

4. J. P. Cohoon, D. S. Richards and J. S. Salowe, A Linear-Time Steiner Tree Routing Algorithm for Terminals on the Boundary of a Rectangle, *Internation Conference on Computer-Aided Design*, 1988, pp. 402-405.

5. H. Gabow, J. L. Bentley and R. E. Tarjan, Scaling and Related Techniques for Geometry Problems, *16th Symposium on the Theory of Computing*, 1984, pp. 135-143.

6. M. R. Garey and D. S. Johnson, *Computers and Intractability*, Freeman, 1979.

7. M. Hanan, On Steiner's Problem with Rectilinear Distance, *SIAM J. of Appl. Math,* **14**, 1966, pp. 255-265.

8. F. K. Hwang, On Steiner Minimal Trees with Rectilinear Distance, *SIAM Journal Applied Math.*, **30**, 1976, pp. 104-114.

9. F. K. Hwang and D. S. Richards, *Steiner Tree Problems*, University of Virginia, 1988.

10. E. L. Lawler, Combinatorial Optimization: Networks and Matroids, Holt, Rinehart & Winston, 1976.

11. T. M. Nicholl, D. T. Lee, Y. Z. Liao and C. K. Wong, On the X-Y Convex Hull of a Set of X-Y Points, *BIT*, **23**, 1983, pp. 456-471.

12. J. S. Provan, Convexity and the Steiner Tree Problem, *Networks,* **18**, 1988.

13. D. S. Richards and J. S. Salowe, Special Convex Cases for Rectilinear Steiner Minimal Trees, *NATO Workshop on Topological Network Design*, 1989.

14. Y. Y. Yang and O. Wing, Optimal and Suboptimal Solution Algorithms for the Wiring Problem, *Proc. IEEE International Symposium Circuit Theory*, 1972, pp. 154-158.