Appeared in Proceedings of the Sixth Canadian Conference on Computational Geometry, pp. 308–313, 1994.

Optimal Rectilinear Steiner Minimal Trees in $O(n^2 2.62^n)$ Time

Joseph L. Ganley and James P. Cohoon
Department of Computer Science
University of Virginia
Charlottesville, Virginia 22903

Abstract

This paper presents an algorithm that computes an optimal rectilinear Steiner minimal tree of n points in at most $O(n^2 2.62^n)$ time. For instances small enough to solve in practice, this time bound is provably faster than any previous algorithm, and improves the previous best bound for practically solvable instances, which is $O(n3^n)$. Experimental evidence is also presented that demonstrates that the algorithm is fast in practice as well—much faster, in fact, than its worst-case time complexity suggests. As part of the analysis, it is proven that the number of full sets on a set of n terminals is at most $O(n1.62^n)$.

1 Introduction

The *rectilinear Steiner minimal tree (RSMT)* problem is stated as follows: given a set T of points called *terminals* in the plane, find a set S of additional points called *Steiner points* such that the length of a rectilinear minimum spanning tree of $T \cup S$ is minimized. For example, Figure 1 illustrates an optimal RSMT for a set of 27 terminals. Garey and Johnson [5] prove that the RSMT problem is NP-complete, indicating that a polynomial-time algorithm to compute an optimal RSMT is unlikely to exist.

A number of algorithms to compute optimal RSMTs have appeared in the literature; we will briefly discuss those for which good bounds on worst-case time complexity are known or those that perform well in practice.

One may compute an optimal RSMT by reducing an instance thereof to an instance of the Steiner problem in graphs. From a set of n terminals, one can use a theorem of Hanan [6] to construct a graph G with $O(n^2)$ vertices such that solving the Steiner problem in G solves the original geometric RSMT instance. The most efficient algorithm for solving the Steiner problem on G is the dynamic programming algorithm of Dreyfus and Wagner [3], which has time complexity $O(n^2 3^n)$ when applied to G. Thomborson, Alpern, and Carter [12] present some improvements to the Dreyfus-Wagner algorithm that do not change the algorithm's time complexity, but do improve its efficiency in practice. Ganley and Cohoon [4] present a more direct, geometric algorithm that has time complexity $O(n3^n)$ and is faster in practice than the Dreyfus-Wagner and Thomborson, Alpern, and Carter algorithms (see Section 7).

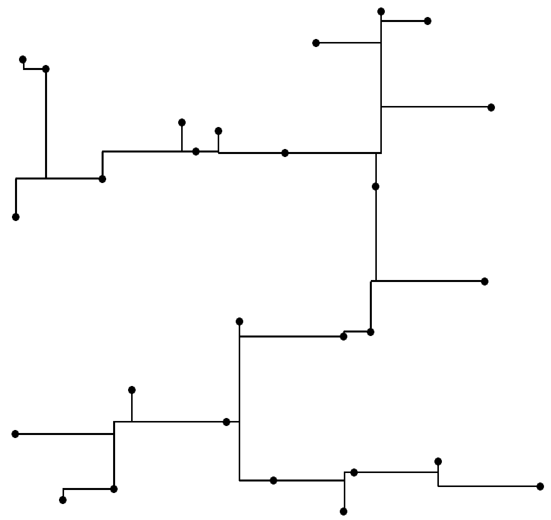
Figure 1: An optimal RSMT for a set of 27 terminals.

Smith ([11], page 162) presents an algorithm with worst-case asymptotic time complexity $n^{O(\sqrt{n})}$, which is asymptotically faster than any exponential algorithm, but his algorithm is not practically applicable due to tremendous constant factors in its time complexity. Salowe and Warme [10] present an algorithm that works very well in practice, but the only known bound on its worst-case time complexity is $O(2^{2^n})$ (our

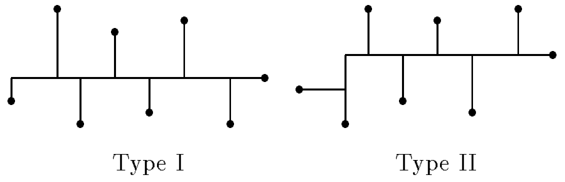

Figure 2: Possible full tree topologies according to Hwang's theorem.

results in Section 5 improve this slightly to $O(2^{n1.62^n})$.

Here we strike a balance between theoretical and practical efficiency by presenting a practical algorithm whose time complexity is at most $O(n^2(1+\phi)^n)$, where $\phi = (1+\sqrt{5})/2 \approx 1.62$. This is the best proven time complexity of any practically applicable algorithm, improving the previous $O(n3^n)$ bound achieved by the algorithm of Ganley and Cohoon [4].

As mentioned previously, the $n^{O(\sqrt{n})}$ algorithm of Smith [11] is asymptotically faster than any exponential algorithm, but by analyzing the constants in the time complexity of Smith's algorithm, we can prove that our algorithm is faster than Smith's for instances containing fewer than about 100 terminals (note that the best algorithms can only solve 35-terminal problems in practice [10]). Indeed, our analysis is conservative; Smith states that one would want to use a different algorithm for instances containing fewer than 300 terminals [11]. Our algorithm is also faster in practice than all other known algorithms except that of Salowe and Warme.

2 Background

Before describing the algorithm we define several terms. A set T of terminals is a *full set* if in every optimal RSMT for T, every terminal in T is a leaf. An RSMT of a full set is called a *full tree*. Hwang [8] proved that a full tree can have only one of two simple topologies. A *type I* topology is a *backbone* segment adjacent to one of the extreme terminals, with segments connecting the other terminals to the backbone. From left to right, these terminals must appear on alternating sides of the backbone. A *type II* topology is similar to a type I topology, but with the leftmost (or rightmost) terminal connected to the segment that connects the second terminal from the left (or right) to the backbone. The two topologies are illustrated in Figure 2. Using Hwang's theorem, an optimal RSMT of a full set can be computed in linear time by checking these two topologies. Finally, a well-known decomposition theorem regarding RSMTs is that every RSMT consists of a number of full trees that intersect at terminals of degree two or greater (see, e.g., Hwang, Richards, and Winter [9]).

```
(1) For $m = 2$ to $|T|$
(2)   For all $S \subseteq T$ such that $|S| = m$
(3)     $\ell[S] = \textbf{FullTree}(S)$
(4)     For all $A, B$ such that $A \bowtie B = S$
(5)       $\ell[S] = \min\{\ell[S], \ell[A] + \ell[B]\}$
```

Figure 3: The FDP algorithm. T is the set of input terminals, and the routine **FullTree** computes the length of an optimal full tree using Hwang's theorem.

3 Full set dynamic programming

Our algorithm builds upon the *Full set Dynamic Programming (FDP)* algorithm of Ganley and Cohoon [4]. Since that algorithm is central to the current one, we briefly describe it.

From the decomposition theorem mentioned above, it is clear that an optimal RSMT for any set of terminals is either a full tree satisfying Hwang's theorem, or can be divided into two smaller trees joined at a terminal. This observation leads to the FDP algorithm. Every subset of the input set of terminals is enumerated in order of increasing cardinality. For each subset S, the algorithm checks the length of the full tree produced by applying Hwang's theorem, and the lengths of the trees produced by joining the optimal RSMTs of every pair of subsets A and B such that $A \cup B = S$ and $|A \cap B| = 1$. (Henceforth we write $A \bowtie B = S$ if $A \cup B = S$ and $|A \cap B| = 1$.) The decomposition with minimum length is an optimal RSMT for the set S of terminals. Since the subsets are enumerated in order of cardinality, at each step the optimal RSMTs for the smaller subsets A and B have already been computed and stored.

Figure 3 describes the FDP algorithm in detail. As shown, the algorithm computes only the length of the optimal tree; a similar top-down pass computes the actual tree given the lengths computed by the first pass. The time complexity of the FDP algorithm is

$$n2^n + \sum_{m=2}^{n} \binom{n}{m} m 2^{m-1} = O(n3^n)$$

(the first term is incurred by line (3) of Figure 3, and the summation mirrors the remaining code).

4 Full set screening

The key concept in the Euclidean Steiner minimal tree algorithms of Cockayne and Hewgill [1, 2] and Winter [13] and the RSMT algorithm of Salowe and Warme [10] is that of *full set screening*. The idea is that relatively few subsets of the set of terminals can be full sets. Thus, one can apply a number of tests to each subset to potentially eliminate it from candidacy as a possible full set. For example, the subset must be connectable according to one of the topologies specified by Hwang's theorem.

Full set screening can also be used to improve the running time of the FDP algorithm. In the FDP algorithm, the innermost loop enumerates all pairs of subsets A and B such that $A \bowtie B = S$. Having identified a number of candidate full sets, one can also require that A be a candidate full set while still satisfying the decomposition theorem. Thus, if the number of candidate full sets that are subsets of S is asymptotically smaller than the total number of subsets of S, and if the subsets of S that are candidate full sets can be efficiently enumerated, then the time complexity of the FDP algorithm is improved.

Before proceeding further, we define some additional notation. For any set S of terminals, let $F(S)$ denote the set of candidate full sets that are subsets of S. Note that if S is itself a candidate full set, then $F(S)$ includes S. The set of input terminals is T, so the complete set of candidate full sets is $F(T)$. Let $f(n)$ denote the maximum number of candidate full sets over all sets of terminals of cardinality n.

5 Bounding the number of full sets

An important component of our analysis is proving an upper bound on $f(n)$ that is asymptotically smaller than $O(2^n)$. By using Hwang's theorem, we prove that $f(n) \leq O(n\phi^n)$, where $\phi = (1+\sqrt{5})/2$.

We first prove a pair of results regarding binary strings, and then use these results to derive the bound. These strings are described by regular expressions; readers not familiar with regular expression notation may refer to Hopcroft and Ullman [7]. Define a *string* S to be a sequence $s_1 s_2 \cdots s_n$, where $s_i \in \{\mathsf{a}, \mathsf{b}\}$, i.e. S is a string in the language $(\mathsf{a}+\mathsf{b})^*$. Define a *substring* of a string S to be $s_{i_1} s_{i_2} \cdots s_{i_k}$ where $1 \leq i_1 < i_2 < \cdots < i_k \leq n$. Each s_i in a string S is considered distinct; e.g., given a string $S = \mathsf{aaa}$, the substrings $s_1 s_2$ and $s_2 s_3$ are considered distinct even though they both have value aa. An *alternating string* is one in the language $\mathsf{b}(\mathsf{ab})^*(\mathsf{a}+\epsilon)$, and an *anti-alternating string* is one in the language $\mathsf{a}(\mathsf{ba})^*(\mathsf{b}+\epsilon)$.

Lemma 1 *There are at most ϕ^n alternating (or anti-alternating) substrings of an alternating string of length n, where $\phi = (1+\sqrt{5})/2$.*

Proof: Let $A(n)$ denote the number of alternating substrings of an alternating string of length n. Similarly let $B(n)$ be the number of anti-alternating substrings of an anti-alternating string of length n. An alternating substring of an alternating string S of length n can either be composed of its first element concatenated with an anti-alternating substring of an anti-alternating string of length $n-1$, or neither of its first two elements but an alternating substring of an alternating string of length $n-2$. This observation yields the following recurrence:

$$\begin{aligned} A(n) &= B(n-1) + A(n-2) \\ B(n) &= A(n-1) + B(n-2) \\ A(1) &= 1 \\ B(1) &= 1. \end{aligned}$$

This recurrence solves to $A(n) = F_n < \phi^n$ (F_n is the n^{th} Fibonacci number). The number of anti-alternating substrings in a string S is equal to number of alternating substrings in the complement of S, so the same applies for anti-alternating substrings. □

We now show that the number of alternating substrings of a string S is maximized if S is itself alternating.

Lemma 2 *The number $A(n)$ of alternating substrings of an alternating string of length n is at least as large as the number of alternating substrings of any string of length n.*

Proof: The proof is by induction on n. The first nontrivial base case is $n = 2$. The alternating string ba contains two alternating substrings, ab contains one, aa contains none, and bb contains two. For the inductive hypothesis, assume that an alternating string of length less than n contains at least as many alternating substrings as any string of equal length.

Let S be a string of length n, and assume that S is neither an alternating nor an anti-alternating string, so that it must contain a contiguous sequence C of the form aaa^* or bbb^* (without loss of generality assume that C is in bbb^*). Let W be the portion of S to the left of C, and let E be the portion of S to the right of C, like so:

$$S = \underbrace{\mathsf{babab}\cdots\mathsf{aba}}_{W}\underbrace{\mathsf{bbb}\cdots\mathsf{bbb}}_{C}\underbrace{\mathsf{abab}\cdots\mathsf{ababa}}_{E}.$$

Let $m = |W|$ and $c = |C|$, and assert that C is maximal in the sense that $w_m = e_1 = \mathsf{a}$. Assume that the number $A(S)$ of alternating substrings in S is greater than $A(n)$; we will show that this assumption leads to a contradiction. In all cases, at most one of the elements of C can be present in an alternating substring. If W is empty, then an alternating substring can be formed by any element of C along with an anti-alternating substring in W, or by no element of C nor w_1, but an alternating substring in $w_2 w_3 \cdots w_m$. By the inductive hypothesis and Lemma 1, the number of alternating substrings in any string of length m for $m < n$ is at most ϕ^m. Thus,

$$A(S) \leq c\phi^{n-c} + \phi^{n-c-2}.$$

We have assumed that $A(S) > A(n)$, implying

$$\begin{aligned} \phi^{n-c} + \phi^{n-c-2} &> \phi^n, \text{ i.e.,} \\ c + \frac{1}{\phi^2} &> \phi^c, \end{aligned}$$

which has no solution, contradicting our assumption. A similar argument can be applied if instead E is empty.

If W and E are both nonempty, then an alternating substring can be formed by concatenating an alternating substring of W, an element of C, and an anti-alternating substring of E, or by an alternating substring of W concatenated with an alternating substring of E. In the latter case, the number of strings in which w_m and e_1 both appear must be subtracted. Again using the inductive hypothesis and Lemma 1,

$$A(S) \leq c\phi^{n-c} + \phi^{n-c} - \phi^{n-c-2}.$$

Invoking the assumption that $A(S) > A(n)$ yields

$$c + 1 - \frac{1}{\phi^2} > \phi^c,$$

which again has no solution, providing the desired contradiction. □

We now use Lemmas 1 and 2 to bound the number of candidate full sets for a given backbone. Assume that no two terminals have the same x or y value; if necessary, this can be ensured by perturbation. Label the terminals $t_1, t_2, \ldots, t_n$, such that $x_{t_i} < x_{t_{i+1}}$ for all $1 \leq i < n$. We now show that if t_i and t_j define a backbone ℓ, then the number of full sets inducing a type I topology with ℓ as a backbone is at most $O(\phi^{j-i})$.

Theorem 1 *The number of full sets inducing a type I topology with a backbone defined by terminals t_i and t_j is at most $O(\phi^{j-i})$.*

Proof: Assume without loss of generality that the backbone ℓ is horizontal, its right endpoint is t_j, its left endpoint is (x_{t_i}, y_{t_j}), and that $y_{t_i} < y_{t_j}$. Let $\ell_y = y_{t_j}$.

A type I topology with ℓ as a backbone can contain only terminals t_k such that $i \leq k \leq j$. Build a string S of length $j - i - 1$ based on the positions of the terminals relative to ℓ; specifically, s_{k-i} is an a if $y_{t_k} < \ell_y$, and s_{k-i} is a b if $y_{t_k} > \ell_y$. Since $y_{t_i} < \ell_y$, terminal t_i corresponds to an a, and there is a one-to-one correspondence between alternating substrings of S and sets of terminals that form a valid type I topology with ℓ as their backbone. From Lemmas 1 and 2, the maximum number of alternating substrings of a string of length n is ϕ^n, so the maximum number of candidate full sets with ℓ as a backbone is ϕ^{j-i-1}. This amount must be doubled to account for the case where ℓ is vertical rather than horizontal, and doubled again for the case when ℓ is adjacent to t_i rather than t_j, giving us $4\phi^{j-i-1} = O(\phi^{j-i})$, as desired. □

We now use Theorem 1 to show that the maximum total number $f(n)$ of candidate full sets on n terminals is at most $O(n\phi^n)$.

Theorem 2 *The number $f(n)$ of candidate full sets on n points is at most $O(n\phi^n)$.*

Proof: A candidate full set must satisfy Hwang's theorem, so the quantity $f(n)$ is bounded by the total maximum numbers of candidate full sets with type I or type II topology for every possible backbone. Theorem 1 gives a bound on the number of candidate full sets with type I topology for a given backbone. Given a backbone defined by terminals t_i and t_j, a type II topology can be formed by joining a terminal to the left of t_i or to the right of t_j to the full tree. In the worst case, for every type I topology, any of these terminals can form a valid type II topology in this manner. Thus, from Theorem 1, the maximum number of candidate full sets with type II topology for a backbone defined by t_i and $t_j = t_i + m$ is $O((i+n-j)\phi^{j-i})$. Summing over all choices of t_i and t_j yields

$$f(n) \leq \sum_{i=1}^{n} \sum_{m=0}^{n-i} (n - m + 1)\phi^m = O(n\phi^n),$$

as desired. □

6 The $\mathbf{O(n^2 2.62^n)}$ algorithm

Given the results above, it remains only to show that for each subset S, the set of candidate full sets that are subsets of S, i.e. the set $F(S) = \{S' \in F(T) : S' \subseteq S\}$,

```
(1) For $m = 2$ to $|T|$
(2)   For all $S \subseteq T$ such that $|S| = m$
(3)     if $(S \notin F)$ then $\ell[S] = \infty$
(4)     Compute $F(S)$
(5)     For all $A, B$: $A \in F(S)$ and $A \bowtie B = S$
(6)       $\ell[S] = \min\{\ell[S], \ell[A] + \ell[B]\}$
```

Figure 4: The SFDP algorithm. T is the set of input terminals, and F is the set of candidate full sets.

can be efficiently computed. We now show that this can be accomplished in $O(|F(S)|)$ time.

The set $F(S)$ consists simply of S itself if $S \in F(T)$, plus $F(S')$ for each $S' = S - \{s_i\}$ such that $s_i \in S$. This observation dovetails nicely with the dynamic programming algorithm: as the algorithm enumerates the sets S of cardinality m, it retains $F(S)$ for each. Each new $F(S)$ is then computed as follows:

$$F(S) = (\{S\} \cap F(T)) \cup \bigcup_{S' = S - \{s_i\}} F(S').$$

Since each $F(S')$ has been stored from the previous iteration, this computation can be performed in $O(|F(S)|)$ time.

We are now ready to describe the algorithm, which we call *Screened Full set Dynamic Programming (SFDP)*. The subsets of the input set T of terminals are enumerated in order of increasing cardinality, starting with subsets of cardinality three. For each subset S, compute $F(S)$ as described in the previous paragraph, and examine each subset A in $F(S)$. For every such A, consider each set B such that $A \bowtie B = S$ (recall that $A \bowtie B = S$ means that $A \cup B = S$ and $|A \cap B| = 1$). The length of an optimal RSMT for S is the minimum over all such A and B of their combined lengths, and the minimum of this value and the length of a full tree on S if S is in $F(T)$. Since the subsets are enumerated in order of increasing cardinality, each required B has already been computed and stored. The SFDP algorithm is shown in pseudocode form in Figure 4.

As described above, $F(S)$ can be computed for any set S in $O(|F(S)|)$ time. Thus, the time complexity of the SFDP algorithm is

$$\sum_{m=3}^{n} \binom{n}{m} m f(m).$$

From Theorem 2, $f(m) \leq m\phi^m$, so the time complexity of the SFDP algorithm is at most

$$\sum_{m=3}^{n} \binom{n}{m} m^2 \phi^m = O(n^2(1+\phi)^n) \approx O(n^2 2.62^n).$$

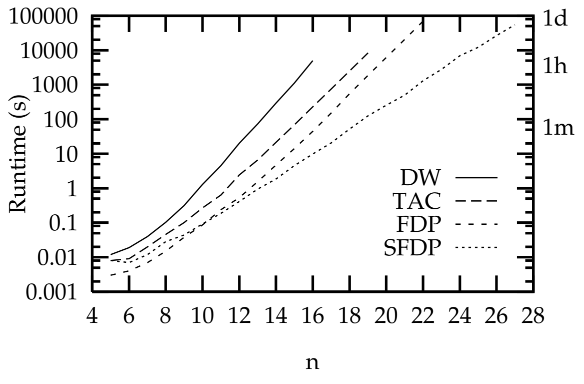

Figure 5: Running times of Dreyfus-Wagner (DW), Thomborson, Alpern, and Carter (TAC), FDP, and SFDP algorithms.

The time required to compute the set $F(T)$ of candidate full sets is $O(f)$, where f is the number of subsets that satisfy Hwang's theorem; by Theorem 2, this time complexity does not exceed $O(n\phi^n)$. Additional tests require polynomial time for each candidate full set satisfying Hwang's theorem [10], so this term is dominated by the decomposition summation.

7 Empirical results

We have implemented the SFDP algorithm in order to compare it empirically with the FDP algorithm [4], the Dreyfus-Wagner algorithm [3], and the algorithm of Thomborson, Alpern, and Carter [12].

Figure 5 plots the running time of each algorithm as a function of the number n of input terminals. As can be seen, the FDP algorithm is faster than both the Dreyfus-Wagner algorithm and the Thomborson, Alpern, and Carter algorithm, but still has the same slope on the logarithmic scale of the plot, since the base of the exponential in its time complexity is still 3.

While the SFDP algorithm is a bit slower than the FDP algorithm for $n < 10$, after this point it is faster than all three algorithms, and becomes more so as n grows, due to the asymptotic improvement in the exponential. Note that $O(n^2 2.62^n)$ is an extremely pessimistic bound. In practice, we apply a number of other tests for full set candidacy [10], and $f(n)$ seems to be *much* smaller than $O(n\phi^n)$; in fact, we conjecture that there is a polynomial upper bound on $f(n)$. Indeed, the slope of the running time curve for the SFDP algorithm suggests that its time complexity in practice is $O(p(n)2^n)$, where $p(n)$ is a polynomial function of n. This suggests that the number of full sets, at least in practice, is indeed polynomial, since the 2^n term is incurred by the outer two loops of the algorithm alone.

We note that the Salowe and Warme algorithm [10] is still significantly faster than the SFDP algorithm in practice. However, since they use a branch-and-bound search to examine the various full set decompositions, it is difficult to derive a nontrivial bound on the time complexity of their algorithm. Using the bound on the number of full sets from Theorem 2, one obtains an upper bound of $O(2^{n\phi^n})$ on the time complexity of the Salowe-Warme algorithm, though based on its performance in practice, this bound is clearly pessimistic.

The main source of inefficiency in the SFDP algorithm is examining every subset of the set of terminals. In practice, it appears that a very small portion of these subsets admits *any* valid decomposition into candidate full sets—in our tests, the ratio of the number of subsets with valid decompositions to the total number of subsets was roughly 0.04 for 20-terminal instances, and this ratio decreases as n grows. If one can efficiently determine *a priori* which subsets admit valid decompositions, and examine only those, we believe the time complexity of the algorithm can be improved to $O(p(n)c^n)$ for $c < 2$. Note that this modification requires a different way to handle enumerating candidate full sets, as the current technique is dependent on enumerating every subset of the set of terminals.

Acknowledgments

The authors gratefully acknowledge the support of National Science Foundation grant MIP–9107717 and Virginia CIT award 5–30971. Thanks are also due to Salowe and Warme and to Thomborson, Alpern, and Carter for providing the code for their respective algorithms [10, 12], and to Jeff Salowe for helpful discussions on several aspects of this paper.

References

[1] E. J. Cockayne and D. E. Hewgill. Exact computation of Steiner minimal trees in the plane. *Information Processing Letters*, 22:151–156, 1986.

[2] E. J. Cockayne and D. E. Hewgill. Improved computation of plane Steiner minimal trees. *Algorithmica*, 7:219–229, 1992.

[3] S. E. Dreyfus and R. A. Wagner. The Steiner problem in graphs. *Networks*, 1:195–207, 1972.

[4] J. L. Ganley and J. P. Cohoon. A faster dynamic programming algorithm for exact rectilinear Steiner minimal trees. In *Proceedings of the Fourth Great Lakes Symposium on VLSI*, pages 238–241, 1994.

[5] M. R. Garey and D. S. Johnson. The rectilinear Steiner tree problem is NP-complete. *SIAM Journal of Applied Mathematics*, 32:826–834, 1977.

[6] M. Hanan. On Steiner's problem with rectilinear distance. *SIAM Journal of Applied Mathematics*, 14:255–265, 1966.

[7] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA, 1979.

[8] F. K. Hwang. On Steiner minimal trees with rectilinear distance. *SIAM Journal of Applied Mathematics*, 30:104–114, 1976.

[9] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner Tree Problem*. North-Holland, Amsterdam, Netherlands, 1992.

[10] J. S. Salowe and D. M. Warme. An exact rectilinear Steiner tree algorithm. In *Proceedings of the International Conference on Computer Design*, pages 472–475, 1993.

[11] W. D. Smith. How to find Steiner minimal trees in Euclidean d-space. *Algorithmica*, 7:137–177, 1992.

[12] C. D. Thomborson, B. Alpern, and L. Carter. Rectilinear Steiner tree minimization on a workstation. In *Proceedings of the DIMACS Workshop on Computational Support for Discrete Mathematics*, 1992.

[13] P. Winter. An algorithm for the Steiner problem in the Euclidean plane. *Networks*, 15:323–345, 1985.