Thumbnail Rectilinear Steiner Trees*

Joseph L. Ganley and James P. Cohoon

Department of Computer Science, University of Virginia, Charlottesville, Virginia 22903

Abstract

The rectilinear Steiner tree problem is to find a minimum-length set of horizontal and vertical line segments that interconnect a given set of points in the plane. Here we study the thumbnail rectilinear Steiner tree problem, where the input points are drawn from a small integer grid. Specifically, we devise a full-set decomposition algorithm for computing optimal thumbnail rectilinear Steiner trees. We then present experimental results comparing this algorithm with two existing algorithms for computing optimal rectilinear Steiner trees. The thumbnail rectilinear Steiner tree problem has applications in VLSI placement algorithms based on geometric partitioning and in global routing of field-programmable gate arrays.

1 Introduction

The *rectilinear Steiner tree (RST)* problem is defined as follows: given a set P of k points in the plane, find a set S of additional points called *Steiner points* such that the length of a minimum spanning tree of $P \cup S$ is minimized. The RST problem is NP-complete, suggesting that no polynomial-time algorithm can solve it exactly. However, many exponential-time algorithms have appeared in the literature that efficiently solve small problem instances [7, 9, 10, 12, 17].

Here we consider the RST problem when the point set P is drawn from an $m \times m$ integer grid, where m is small. We call this the *thumbnail rectilinear Steiner tree (TRST)* problem. The TRST problem has several applications in VLSI physical design automation. In particular, several researchers [2, 16, 18] have considered VLSI placement heuristics based on $m \times m$ geometric partitions of the circuit area. In these algorithms, TRSTs are heavily used; typically they are precomputed and stored for use by the placement algorithm. Such an approach limits m to about 4 or less due to space requirements, so efficient computation of TRSTs would allow investigation of this placement technique for larger m. Another application is in global routing of field-programmable gate arrays, which are typically composed of logic blocks arranged in a small grid, with rectilinear wires connecting them.

Researchers have previously investigated Euclidean Steiner trees of point sets from small grids [3, 4, 5] and triangular and hexagonal grids [14], but this is the first known study of the rectilinear problem. Note that the Euclidean problem is very different from the rectilinear one. For the Euclidean case, most previous work [3, 4] focuses on simple, mechanical constructions that are conjectured but not proven to be optimal. In the rectilinear case there seem to be no such simple constructions for which optimality can be reasonably conjectured. In particular, the Euclidean constructions largely focus on the complete $m \times m$ point set, which is uninteresting in the rectilinear case since a minimum spanning tree is an optimal TRST for such an instance. Cockayne and Hewgill [5] use their exact algorithm for the general Euclidean Steiner tree problem to compute optimal Euclidean Steiner trees for a number of point sets on a small integer grid.

Our goal in this research is to exploit the special nature of the inputs to devise algorithms that are more efficient than simply applying existing algorithms to these point sets.

This paper presents some preliminary results on the TRST problem. We devise a full-set decomposition algorithm for computing optimal TRSTs, which uses the special nature of the inputs to improve the screening of candidate full sets. We then present experimental results comparing this algorithm with two existing algorithms for computing optimal rectilinear Steiner trees. The reader should note that all results presented here are easily generalized to rectangular, rather than square, grids.

2 Terminology

A problem instance consists of a set P of k points called *terminals*, drawn from an $m \times m$ integer grid.

A set T of terminals is a *full set* if, in every optimal RST of T, every terminal is a leaf. An RST of a full set is called a *full tree*. Hwang [13] proved that a full set must have one of two simple topologies. These topologies are illustrated in Figure 1. Hwang's theorem allows computation of an optimal RST of a full set in linear time.

*The authors gratefully acknowledge the support of National Science Foundation grants MIP-9107717 and CCR-9224789. JLG also gratefully acknowledges a Virginia Space Grant Fellowship.

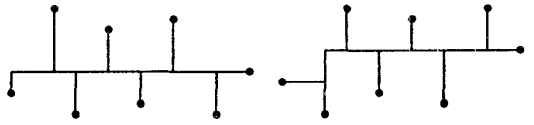

Figure 1: Possible full tree topologies according to Hwang's theorem.

3 Simple Approaches

As mentioned previously, VLSI placement algorithms that utilize TRSTs do so by precomputing the optimal TRST for every possible point set and simply looking them up at runtime. This is obviously quite time-efficient, but requires $O(2^{m^2})$ space, which is prohibitive for m larger than 4. By exploiting the fact that the grid is symmetrical under a 90-degree rotation, one can save a factor of four in the space requirements by sacrificing the $O(1)$ table lookup for an $O(m^2)$ binary search. Even with this reduction, however, storing the TRSTs for all point sets for $m = 5$ requires approximately 40 megabytes of storage. While this might be feasible on some machines, for $m = 6$, storage would require roughly 120 gigabytes of space, which is currently impractical.

Another simple approach is to enumerate all TRSTs for a given point set, and choose the one with minimum length. We would thus like a bound on the total number of possible TRSTs. This question is also interesting since many of the placement algorithms that use TRSTs enumerate all optimal TRSTs.

The number of TRSTs is clearly bounded by the number t of spanning trees in the complete $m \times m$ grid graph. There is no closed form for t [15], but one can easily show that it grows faster than $O(m^2 2^{m^2})$, and therefore enumerating all spanning trees is slower than the spanning tree enumeration algorithm described in the next section.

4 Spanning Tree Enumeration

Another approach is to apply an existing algorithm for the RST problem to thumbnail-sized inputs. One such algorithm is Hakimi's spanning tree enumeration algorithm [12]. Hakimi's algorithm has time complexity $O(n2^n)$, where n is the number of candidate Steiner points. Applied to terminals from an $m \times m$ grid, the time complexity of Hakimi's algorithm is at most $O(m^2 2^{m^2})$.

For the standard RST problem, Hakimi's algorithm is less efficient than many other algorithms [7, 9, 10, 17]. However, when applied to thumbnail-sized inputs, Hakimi's algorithm is faster. The practical algorithm with the best known time complexity for the general RST problem is due to Ganley and Cohoon [10]. Its time complexity is $O(k^2 2.62^k)$ for k terminals. When applied to points from an m by m grid, this time complexity can be as large as $O(m^4 2.62^{m^2})$, which is worse than Hakimi's algorithm.

5 A Dynamic Programming Algorithm

Another natural approach is to apply the Aho, Garey, and Hwang (AGH) algorithm for terminals on a small number of parallel lines [1]. For TRSTs, the terminals lie on a set of at most m parallel lines.

The AGH algorithm proceeds from left to right in the grid, building an optimal RST of the terminals in an x by m grid from optimal RSTs in an $x - 1$ by m grid, which have been stored from the previous iteration.

The AGH algorithm has time complexity $O(m16^m)$. This is clearly asymptotically superior to the other two algorithms described above. However, as will be shown empirically in Section 7, for the small values of m considered here, the AGH algorithm is slower than the others described earlier. In addition, note that the AGH algorithm requires $O(m8^m)$ space, and thus it is inapplicable to the TRST problem with $m > 7$.

6 Full-Set Decomposition

The key concept in the Steiner tree algorithms of Cockayne and Hewgill [5, 6], Salowe and Warme [17], and Winter [19] is that of *full-set decomposition*. A well-known theorem on Steiner trees is that every optimal Steiner tree can be decomposed into a number of full trees joined at terminals of degree greater than 1. Since Hwang's theorem enables linear-time computation of optimal full trees, one can compute an optimal RST by finding the full-set decomposition with minimum total length. The algorithms mentioned above examine every subset of the set of terminals, producing a set of *candidate full sets*. They then use a branch and bound algorithm to find a set of candidate full sets of minimum total length, whose union spans all the terminals.

Here we devise such an algorithm for the TRST problem. The fact that inputs are drawn from a small grid allows us some algorithmic improvements beyond those for the standard RST problem.

Let $MST(T)$ denote the length of a minimum spanning tree of a set T of terminals. A set T of terminals is *compact* if $MST(T) = |T| - 1$ (i.e., if a minimum spanning tree of T contains only unit-length edges).

Theorem 1 *There exists an optimal TRST τ in which every compact subset T of the set of terminals induces a connected component in τ.*

Proof: See Ganley and Cohoon [11]. □

Let $||a-b||$ denote the rectilinear distance between two points a and b.

Corollary 1.1 *There can be no two terminals a and b in a candidate full set such that $||a-b|| = 1$.* □

These results allow us to eliminate many full sets from candidacy. For every compact subset S of the terminals, every full set containing more than two terminals in S is eliminated from candidacy. The subset S is then added to the set of candidate full sets. While it is not technically a full set, it behaves as one in the decomposition since no other candidate full set can intersect it at more than one terminal.

We can also eliminate candidate full sets of cardinality exceeding m.

Theorem 2 *For $m \geq 4$, every candidate full set on an $m \times m$ grid contains at most m terminals.*

Proof: See Ganley and Cohoon [11]. □

In our implementation, we precompute a TRST for every possible full set on an $m \times m$ grid. The algorithm then reads in those full sets that are subsets of the input set T of terminals. Since no two terminals in a full set can share an x or y coordinate, the total number of full sets is at most $O(m^m)$, and consequently the time complexity of the algorithm is at most $O(2^{m^m})$. Note that $O(m^m) = O(c^{m \log m})$ for constant c, which is an improvement on the best known bound of $O(n1.62^n) = O(m^2 1.62^{m^2})$ on the number of full sets in an unrestricted point set [10]. Table 1 shows the total number of possible full sets of cardinality 3 or greater on an $m \times m$ grid, along with the value of the upper bound m^m, for small values of m. Note that since not every set of m or fewer terminals is a full set, the bound of $O(m^m)$ is not tight.

m	3	4	5	6	7
Full sets	17	196	1258	5368	20157
m^m	27	256	3125	46656	823543

Table 1: Number of full sets in an $m \times m$ grid.

If an edge in a full tree of a full set S intersects a terminal in $T - S$, then that terminal would have degree greater than 1, and thus the full set is eliminated from candidacy. Thus, the number of full sets that must be considered for a given instance is typically far smaller than the total number of possible full sets. Empirical results on the total number of candidate full sets are given in Section 7.

7 Experimental Results

We have implemented the spanning tree enumeration, dynamic programming, and full set decomposition algorithms in order to compare them experimentally. The results of these experiments are shown in Table 2. For $m \leq 4$, the values shown result from one run on each of the $\binom{m^2}{k}$ possible point sets. For $m \geq 5$, the values shown result from one run on each of 1000 different randomly generated point sets. There are a number of noteworthy features of these results.

m	k	STE	DP	FSD	%MST	f
3	3	0.001	0.627	0.011	81.0	3
3	6	0.001	0.629	0.013	95.2	4
4	4	0.002	0.682	0.013	66.0	8
4	8	0.009	0.823	0.020	72.2	37
4	12	0.002	0.695	0.014	98.2	5
5	5	0.020	2.020	0.020	47.7	14
5	10	0.737	2.010	0.055	51.3	52
5	15	0.120	2.007	0.051	72.0	41
5	20	0.004	2.030	0.023	99.0	8
6	6	0.450	25.33	0.030	34.1	28
6	12	152.5	25.97	1.639	31.9	91
6	18	23.01	26.19	9.616	44.8	78
6	24	1.048	25.91	0.213	77.7	35
6	30	0.007	26.67	0.040	99.2	7
7	7	13.55	830.7	0.079	21.0	35
7	14	18632	889.8	55.69	17.3	173
7	21	15867	862.6	205.2	29.1	144
7	28	455.0	901.0	74.69	46.6	74
7	35	6.317	907.8	0.223	84.1	32
7	42	0.014	906.6	0.109	99.5	8

Table 2: Average runtime (in seconds) for the spanning tree enumeration (STE), dynamic programming (DP), and full-set decomposition (FSD) algorithms. %MST indicates the percentage of the instances for which the TRST has the same length as the MST. The value of f is the maximum number of candidate full sets.

First, note that in spite of its asymptotic superiority, the dynamic programming algorithm is too slow to be used on instances with $m \leq 7$. (Also, recall that the space requirements of the algorithm prohibit its use on instances with $m > 7$.)

Second, note that the spanning tree enumeration and full-set decomposition algorithms are quite fast on very sparse or very dense instances. This is not surprising: for sparse instances, there are few terminals, and for dense instances, there are few candidate Steiner points (for Hakimi's algorithm) or candidate full sets (for the FSD algorithm).

For all but the smallest instances, the FSD algorithm is faster than both the STE and DP algorithms. In addition, the current branch and bound algorithm

for finding the optimal full-set decomposition it quite simple. We believe that its efficiency can be considerably improved by using more elaborate strategies to prune the search space [17].

Lastly, note that often a minimum spanning tree is an optimal TRST. Clearly a TRST cannot have length less than $k-1$, so the algorithms first compute an MST, and if it has length $k-1$, it is returned as the optimal TRST. However, in many cases the MST has length exceeding $k-1$, but is nonetheless an optimal TRST.

8 Conclusions

This is the first known study of the thumbnail rectilinear Steiner tree problem. We are continuing this work in a number of directions.

For VLSI placement applications, there is another important aspect of the TRST problem that we have not yet addressed: how to enumerate different optimal TRSTs. However, we believe that once a single TRST is computed, we can use an algorithm similar to that of Eppstein, Galil, Italiano, and Nissenzweig [8] to enumerate all optimal TRSTs.

As mentioned in Section 7, often a minimum spanning tree of a thumbnail-sized instance is an optimal TRST. Thus, one possible approach is to precompute and store only those TRSTs that are shorter than a minimum spanning tree, and look them up at runtime.

The algorithms examined here are not radically different from algorithms for the general RST problem; they simply use the restricted nature of the inputs to improve such algorithms. We believe it is possible to exploit the nature of the TRST problem more completely, and thus to devise much faster algorithms for it. Our ultimate goal is an algorithm that can compute an optimal TRST for any point set on an $m \times m$ grid for small $m \leq 10$ in less than one second.

References

[1] A. V. Aho, M. R. Garey, and F. K. Hwang. Rectilinear Steiner trees: Efficient special-case algorithms. *Networks*, 7:37–58, 1977.

[2] S. Bapat and J. P. Cohoon. A parallel VLSI circuit layout methodology. In *Proceedings of the Sixth IEEE International Conference on VLSI Design*, January 1993.

[3] F. R. K. Chung, M. Gardner, and R. L. Graham. Steiner trees on a checkerboard. *Mathematics Magazine*, 62:83–96, 1989.

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

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

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

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

[8] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig. Sparsification: A technique for speeding up dynamic graph algorithms. In *Proceedings of the Thirty-third Symposium on Foundations of Computer Science*, pages 60–69, 1992.

[9] 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.

[10] J. L. Ganley and J. P. Cohoon. Optimal rectilinear Steiner minimal trees in $O(n^2 2.62^n)$ time. In *Proceedings of the Sixth Canadian Conference on Computational Geometry*, pages 308–313, 1994.

[11] J. L. Ganley and J. P. Cohoon. Thumbnail rectilinear Steiner trees. Technical Report CS–95–01, Department of Computer Science, University of Virginia, Charlottesville, Virginia, 1995.

[12] S. L. Hakimi. Steiner's problem in graphs and its implications. *Networks*, 1:113–133, 1971.

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

[14] F. K. Hwang and D. Z. Du. Steiner minimal trees on the Chinese checkerboard. *Mathematics Magazine*, 64:332–339, 1991.

[15] G. Kreweras. Complexité et circuits Eulériens dan les sommes tensorielles de graphes. *Journal of Combinatorial Theory, Series B*, 24:202–212, 1978. (in French).

[16] S. Mayrhofer and U. Lauther. Congestion-driven placement using a new multi-partitioning heuristic. In *Proceedings of the International Conference on Computer-Aided Design*, pages 332–335, 1990.

[17] 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.

[18] P. R. Suaris and G. Kedem. A quadrisection-based place and route scheme for standard cells. *IEEE Transactions on Computer-Aided Design*, 8:234–244, 1989.

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