TR CS-93-12

Experimental Evaluation of a Partitioning Algorithm for the Steiner Tree Problem in R^2 and R^3

Sivakumar Ravada and Alan T. Sherman
Computer Science Department
University of Maryland Baltimore County
Baltimore, Maryland 21228-5398

June 23, 1993

Abstract

We experimentally evaluate sequential and distributed implementations of an approximation partitioning algorithm by Kalpakis and Sherman for the *Geometric Steiner Minimum Tree Problem (GSMT)* in R^d for $d = 2, 3$. Our implementations incorporate an improved method for combining the subproblems, and single and double edge-reduction techniques to eliminate unnecessary Steiner points created by the partitioning process. Results show that these refinements are crucial in practice to produce high-quality results. Moreover, with these refinements, the partitioning algorithm is an effective practical heuristic, which in less time finds solutions roughly comparable with those computed by Smith's Algorithm.

The partitioning algorithm depends on a parameter t through which the user can trade off time for solution quality. To solve each of the subproblems of size at most t, we apply Smith's Algorithm, the only other Steiner tree algorithm known for R^3 under the Euclidean norm. Using uniformly generated point sets in the d-cube $[0, 100]^d$, we compare the running time and solution quality for the partitioning algorithm for various values of t against that of Smith's Algorithm applied to the entire input. We also estimate the constant factor in the $1 + O(t^{-1/d(d-1)})$ asymptotic performance bound proven by Kalpakis and Sherman and find that it is less than one.

With $d = 2$ on input sets of 25 points, our sequential implementation with $t = 7$ typically found within 6 seconds a Steiner tree within 1% of the cost (342) of that computed by Smith's Algorithm in 20 seconds. At $t = 13$ and using approximately 15 seconds, our sequential implementation typically found Steiner trees within 0.1% of this cost. Similarly, with $d = 3$ on input sets of 60 points, our distributed implementation with $t = 10$ typically found within 22 seconds a Steiner tree within 1% of the cost (1059) of that computed by Smith's Algorithm in 200 seconds. At $t = 31$ and using approximately 108 seconds, our distributed implementation usually found slightly better Steiner trees than did Smith's Algorithm.

Keywords. Approximation algorithms, combinatorial optimization, distributed algorithms, Euclidean Steiner minimum tree problem (ESMT), experimental evaluation of algorithms, geometric Steiner minimum tree problem (GSMT), graph algorithms, partitioning algorithms, post-processing heuristics.

1 Introduction

Kalpakis and Sherman [6, 7] present and probabilistically analyze a deterministic partitioning approximation algorithm for the Steiner minimum tree problem in R^d for any dimension $d \geq 2$. Although they prove a theoretical upper bound on the asymptotic performance of their algorithm, they neither determine the constant factor in this bound nor present any empirical evidence to demonstrate its effectiveness. To assess the practical viability of their algorithm, we programmed refined sequential and distributed versions of it on a network of workstations for $d = 2, 3$. This report describes our experimental evaluation of these two implementations.

For brevity, we assume the reader is familiar with the paper by Kalpakis and Sherman, including all of its notation and terminology. In particular, we assume the reader is familiar with the geometric Steiner minimum tree problem and with the basic partitioning algorithm analyzed by Kalpakis and Sherman, *Algorithm 1.* We are especially interested in GSMT in R^2 and R^3 under the Euclidean norm because of its applications to single- and multi-layer VLSI routing with non-rectilinear wires.

The *Geometric Steiner Minimum Tree problem (GSMT)* is to connect at minimun cost n given terminals in d-dimensional Euclidean space. No exact polynomial-time algorithm is known for this NP-hard problem. Algorithm 1 computes an approximate solution by partitioning the terminals, by computing an exact or approximate Steiner tree for each subset of the partition, and by combining the solutions to the subproblems.

Kalpakis and Sherman recommend that Algorithm 1 be refined by incorporating post-processing heuristics described by Balakrishnan and Patel [1] and by using an improved method for combining the subproblems. We implement both of these refinements, including single and double edge reduction post-processing heuristics. We call the resulting algorithm *Algorithm 2.* But whereas Kalpakis and Sherman work with arbitrary norms on R^d, we experiment exclusively with the Euclidean norm $\| \|_2$ on R^2 and R^3.

Algorithm 1 depends on a parameter t through which the user can trade time for solution quality. This parameter limits the number of terminals in each subproblem of the partition. We experimentally measure how the choice of t affects the running time and performance of Algorithms 1 and 2. In addition, we approximate the constant factor in the $1 + O(t^{-1/(d(d-1))})$ asymptotic performance bound proven by Kalpakis and Sherman and find that it is less than 1. Throughout, we select the input terminals at random, independently with a uniform distribution over the d-cube $[0, 100]^d$.

The only other experimental evaluation of any GSMT algorithm in R^3 of which we are aware is by Smith [10], who in 1992 presented an exact iterative-improvement algorithm. We shall call this algorithm *Smith's Algorithm.* Smith's Algorithm works for all $d \geq 2$ and can be used as an approximation algorithm by terminating it whenever desired. To solve the subproblems created by Algorithms 1 and 2, we used a C language implementation of Smith's Algorithm, given to us by Smith.

All other implementations of GSMT algorithms for the Euclidean norm of which we are aware provide solutions only to the *ESMT* problem—GSMT restricted to R^2 under the Euclidean norm. For example, in 1972 Cockayne and Schiller [4] reported experiments using an exact ESMT algorithm on up to $n \leq 10$ terminals. Experiments with other exact ESMT algorithms have been reported in 1985 by Winter [12] with $n \leq 15$, and in 1992 by Cockayne and Hewgill [3] with $n \leq 30$. All three of these algorithms stem from an idea by Melzak [8]. Additional exact ESMT algorithms have been proposed in 1977 by Boyce [2], and in 1990 by Trietsch and Hwang [11]. In addition, in 1981, Smith, Lee, and Liebman [9] presented experimental work with their $O(n \lg n)$ approximation ESMT algorithm.[1] We did not experiment with these implementations because they were not available to us and because they do not work for $d = 3$.

It is traditional to evaluate the performance of any approximation GSMT algorithm in terms of its *performance ratio*—the cost of the approximate Steiner tree computed by the algorithm divided by the cost of any Steiner minimum tree on the same input. But it is difficult to compute this ratio because it is difficult to compute Steiner minimum trees. Therefore, we simply compare the results of Algorithms 1 and 2 with those of Smith's Algorithm, which is the only available competing algorithm for $d = 3$. Experiments show that our refinements of Algorithm 1 are very important in practice. Moreover, our implementations of Algorithm 2 typically use much less time to find Steiner trees of comparable cost to those computed by Smith's Algorithm.

The rest of this report is organized as follows. Section 2 briefly explains our refinements of Algorithm 1. Section 3 reports and discusses our experimental results. Section 4 summarizes our conclusions.

2 A Refined Partitioning Algorithm

We implement a refinement, Algorithm 2, of the basic partitioning algorithm, Algorithm 1. Algorithm 2 includes single and double edge-reduction post-processing heuristics and uses an alternate method to combine the subproblems. This section briefly explains these refinements.

Algorithm 2

Given any set $\hat{x}$ of n terminals in R^2 or R^3, and given any parameters $2 \leq t \leq n$ and τ, Algorithm 2 computes an approximate Steiner tree for $\hat{x}$ in four steps:

1. Partition the n terminals into $\Theta(n/t)$ sets, each of size at most t, as done in Algorithm 1.

2. Solve each subproblem exactly or approximately: using Smith's Algorithm, for each set of terminals in the partition, compute a Steiner tree for the set. For each subproblem, terminate Smith's Algorithm after τ seconds.

3. Combine the Steiner trees computed in Step 2 as follows: compute T_{APR} to be a minimum spanning tree of the complete graph on the union of all terminals and Steiner points from Step 2.

4. Locally modify T_{APR} using single and double edge reductions.

Algorithm 2 differs from Algorithm 1 in two respects. First, Algorithm 2 adds Step 4. Second, in Step 3, Algorithm 1 simply computes a minimum spanning tree of the union of the Steiner trees computed in Step 2. In both Algorithms 1 and 2, this union is always connected because the median terminal for each cut of the partition is included in both sides of the partition.

To control the running time of Algorithms 1 and 2, for each subproblem in Step 2, we terminate Smith's algorithm after a specified amount of time, as given by a control parameter τ. To compute minimum spanning trees, for simplicity we use Prim's algorithm [5], implementing the underlying priority queue as a binary heap in Algorithm 1 and as a linear array in Algorithm 2.

[1] Throughout let $\lg = \log_2$.

Combining the Subproblems

By combining the subproblems in Step 3 in a more flexible way, Algorithm 2 offers a practical improvement over Algorithm 1. Let $G = (V, E)$ be the connected graph that is the union of all Steiner trees computed for all subproblems in Step 2, and let $G_c = (V, V \times V)$ be the associated complete graph on V. By computing a minimum spanning tree on G_c rather than on G, Algorithm 2 often finds lower-cost solutions than does Algorithm 1. For example, consider a situation in which there are nearby terminals x and y for which x and y are in separate adjacent subsets of the partition, and edge (x, y) is not in the Steiner tree computed for either subset. Let T_{APR} be the Steiner tree computed by Algorithm 1, and let (u, w) be the longest edge in the simple path from from x to y in T_{APR}. If $\|x - y\|_2 < \|u - w\|_2$, then T_{APR} can be improved by removing edge (u, w) and adding edge (x, y).

The additional time required by this refinement is typically minor in comparison to the time required to solve the subproblems. Let $T_{MST}(v, e)$ be the time to compute a minimum spanning tree on any graph with v vertices and e edges. With this refinement, the time to compute a minimum spanning tree in Step 3 of Algorithm 2 increases from $T_{MST}(V, E)$ to $T_{MST}(V, V^2)$, where $E \in O(V)$ and $|V| \leq (2^{d+2} + 1)n$, as shown in Corollary 1 by Kalpakis and Sherman [6]. Thus, using Prim's algorithm, the time to carry out Step 3 in Algorithm 2 increases from $O(n \lg n)$ to $O(n^2)$.

Edge-Reduction Heuristics

In Step 4 we implement the following two edge-reduction heuristics described by Balakrishnan and Patel [1]. Each of these heuristics performs a local optimization of T_{APR} in linear time in the number of edges in T_{APR}.

1. *Single edge reduction.* For each Steiner point s of degree 1 in T_{APR}, remove s and its incident edge from T_{APR}.

2. *Double edge reduction.* For each Steiner point s of degree 2 in T_{APR}, remove s and replace its two incident edges (x, s) and (s, y) with the single edge (x, y).

These heuristics remove unnecessary Steiner points, which can result from Steps 2 and 3. By the triangle inequality, no double edge reduction can increase the cost of the resulting Steiner tree.

3 Experimental Analysis

To evaluate our implementations of Algorithms 1 and 2, we ran them on randomly generated point sets in R^2 and R^3 under the Euclidean norm for a variety of values of n and t. For each run, we recorded the cost of the approximate Steiner tree returned by our implementation and compared it with that computed by Smith's Algorithm. In addition, we measured the time of each run.

3.1 Methods

We implemented Algorithms 1 and 2 in the C language on Silicon Graphics IRIS4D (Crimson and Indigo) workstations. For Smith's Algorithm, we used a C language implementation given to us by Smith. Since we were primarily interested in measuring performance and not running time, we made little attempt to optimize our implementations for time.

For each value of n, we tried all values for t in the range 2 to $n/2$ that produced distinct partitions. For each value of n, we generated 10 data sets in the cube $[0, 100]^d$. Using the Unix random number generator 'random', we selected each terminal independently with uniform distribution. For each value of t, we computed the observed mean and unbiased standard deviation of the performance and running time of each algorithm on the 10 data sets.

Within Algorithms 1 and 2, we used Smith's Algorithm to solve each of the subproblems approximately, limiting the time spent by Smith's Algorithm by the parameter τ. Smith's Algorithm tends quickly to reach a state of "approximate convergence" for which further reductions in cost are relatively slow and minor. We chose values of τ sufficiently large to enable Smith's Algorithm to attain this approximate convergence.

In our distributed implementation of Algorithm 2, each of the $\Theta(n/t)$ subproblems is assigned to a separate workstation. One workstation serves as a central controller to read the input, to partition the terminals, to distribute the subproblems, and to compute the minimum spanning tree. After solving its subproblem, each workstation writes its solution to a designated file in a shared file system.

3.2 Results

Our experimental results are summarized in Tables 1, 2, and 3. Table 1 shows the performance of our implementations of Algorithms 1 and 2 in comparison with that of Smith's Algorithm for $n = 25$ and $d = 2, 3$. Table 2 lists the running times for these computations. Table 3 gives similar performance and timing data for our distributed implementation of Algorithm 2 for $n = 60$ and $d = 3$. We arbitrarily chose the parameters $n = 25$ and $n = 60$ so that Smith's Algorithm approximately converges within 30 and 200 seconds, respectively.

In Table 1, we also give *relative performance ratios* $\hat{R}_1 = c_1/c_S$ and $\hat{R}_2 = c_2/c_S$ of Algorithms 1 and 2 relative to Smith's Algorithm, where c_1, c_2, and c_S are the costs of the (approximate) Steiner trees computed by Algorithm 1, Algorithm 2, and Smith's Algorithm, respectively. Figure 1 shows sample Steiner trees computed by Algorithm 2 and by Smith's Algorithm for $n = 25$, $t = 7$, and $d = 2$.

In addition, we estimate the constant factors λ_1 and λ_2 in the $1 + O(t^{-1/d(d-1)})$ asymptotic performance bound proven by Kalpakis and Sherman for the (true) performance ratios R_1 and R_2 for Algorithms 1 and 2. We estimate these constant factors by solving the equations $\hat{R}_1 \approx 1 + \hat{\lambda}_1 t^{-1/d(d-1)}$ and $\hat{R}_2 \approx 1 + \hat{\lambda}_2 t^{-1/d(d-1)}$ for $\hat{\lambda}_1$ and $\hat{\lambda}_2$. These approximate constant factors should be viewed as very coarse estimates since our values of n and t may be too small to project asymptotic trends. Moreover, the big-Oh asymptotic bound yields only lower bounds on λ_1 and λ_2 and $\hat{R}_1$ and $\hat{R}_1$ are relative and not exact performance ratios.

In Table 3 we also compute the relative performance ratio of Algorithm 2 with respect to Smith's Algorithm. Since the sequential and distributed implementations of Algorithm 2 compute the same Steiner trees, the sole advantage of our distributed implementation is that it runs faster. To measure this improvement in running time, we compute *Speedup* $= T_{seq}/T_{dis}$, where T_{seq} and T_{dis} are the running times for our sequential and distributed implementations of Algorithm 2, respectively.

3.3 Discussion

Our experiments show that Algorithm 2 is a useful practical heuristic for computing geometric Steiner trees. In this section we discuss our results and raise some open questions.

Table 1: Performance results for our implementations of Algorithms 1 and 2 for $n = 25$ and $d = 2, 3$. Costs c_1, c_2, and c_S are costs of Steiner trees computed, respectively, by Algorithms 1 and 2 for various values of t and by Smith's Algorithm. These costs are given as mean $\pm$ standard deviation, computed from 10 sets of 25 randomly chosen points in the cube $[0, 100]^d$ and measured by the Euclidean norm. Relative performance ratios $\hat{R}_1 = c_1/c_S$ and $\hat{R}_2 = c_2/c_S$ are computed with respect to Smith's Algorithm. Values $\hat{\lambda}_1$ and $\hat{\lambda}_2$ are approximate constant factors for the big-Oh expressions for $\hat{R}_1$ and $\hat{R}_2$, respectively. For comparison, the costs of minimum-cost spanning trees on the same point sets are 350.07 ± 24.80 and 618.21 ± 48.34 for $d = 2, 3$, respectively.

		Algorithm 1			Algorithm 2			Smith's Algorithm
d	t	c_1	$\hat{R}_1$	$\hat{\lambda}_1$	c_2	$\hat{R}_2$	$\hat{\lambda}_2$	c_S
2	5	397.73 ± 40.54	1.1637	0.3662	348.95 ± 26.82	1.0210	0.0470	341.76 ± 24.61
2	7	364.88 ± 32.95	1.0676	0.1789	344.45 ± 25.59	1.0078	0.0208	
2	13	355.57 ± 29.11	1.0404	0.1456	341.87 ± 25.50	1.0003	0.0011	
3	5	688.81 ± 58.44	1.1696	0.2224	612.20 ± 58.43	1.0395	0.0517	588.90 ± 44.71
3	7	656.81 ± 60.31	1.1153	0.1594	606.68 ± 51.54	1.0301	0.0417	
3	13	626.84 ± 50.64	1.0644	0.0987	597.84 ± 47.82	1.0151	0.0232	

Table 2: Timing data for Algorithms 1 and 2 for runs described in Table 1 for $n = 25$ and $d = 2, 3$. Times T_1, T_2, and T_S are running times measured, respectively, from Algorithms 1 and 2 for various values of t and from Smith's Algorithm. These times are given in mean seconds $\pm$ standard deviation, as measured on a Silicon Graphics Indigo workstation from 10 sets of 25 randomly chosen points in the cube $[0, 100]^d$. The parameter τ indicates the maximum time allowed for Smith's Algorithm on any subproblem within Algorithms 1 and 2.

			Algorithm 1		Algorithm 2		Smith's Algorithm
d	t	τ (sec)	T_1 (sec)	T_S/T_1	T_2 (sec)	T_S/T_2	T_S (sec)
2	5	1.00	0.13 ± 0.05	153.84	0.14 ± 0.07	142.85	20
2	7	3.00	5.05 ± 1.40	3.96	5.71 ± 1.50	3.50	
2	13	8.00	14.66 ± 0.86	1.36	15.03 ± 0.48	1.33	
3	5	1.00	0.34 ± 0.05	73.52	0.34 ± 0.06	73.52	25
3	7	4.00	10.10 ± 1.28	2.47	10.45 ± 1.38	2.39	
3	13	9.00	18.01 ± 0.32	1.38	18.47 ± 0.43	1.35	

Table 3: Performance and timing results of our distributed implementation of Algorithm 2 for $n = 60$ and $d = 3$. Cost c_2 is the cost of the Steiner tree computed by Algorithm 2 for various values of t. Times T_{seq} and T_{dis} are, respectively, the running times for our sequential and distributed implementations of Algorithm 2, as measured in seconds on a Silicon Graphics Indigo workstation. Speedup $= T_{seq}/T_{dis}$ indicates the relative speedup in time of our distributed implementation over our sequential implementation. Costs and times are given as mean $\pm$ standard deviation, computed from 10 sets of 60 randomly chosen points in the cube $[0, 100]^3$ and measured by the Euclidean norm. Relative performance ratio $\hat{R}_2 = c_2/c_S$ is computed with respect to the cost $c_S = 1059 \pm 37.17$ computed in 200 seconds by Smith's Algorithm. Value $\hat{\lambda}_2$ is an approximate constant factor for the big-Oh expression for $\hat{R}_2$. The parameter τ indicates the maximum time allowed for Smith's Algorithm on any subproblem within Algorithm 2. For comparison, the cost of a minimum-cost spanning tree on the same point sets is 1087.16 ± 33.56.

			Algorithm 2 Performance			Algorithm 2 Time		
d	t	τ (sec)	c_2	$\hat{R}_2$	$\hat{\lambda}_2$	T_{seq} (sec)	T_{dis} (sec)	Speedup
3	10	9.00	1069.74 ± 38.26	1.0101	0.0148	69.93 ± 4.36	21.99 ± 3.12	3.18
3	17	20.00	1059.00 ± 32.24	1.0000	*	83.08 ± 0.35	29.32 ± 2.57	2.83
3	31	95.00	1050.90 ± 32.01	0.9923	*	196.63 ± 3.83	107.50 ± 3.37	1.82

* Less than 0.0001.

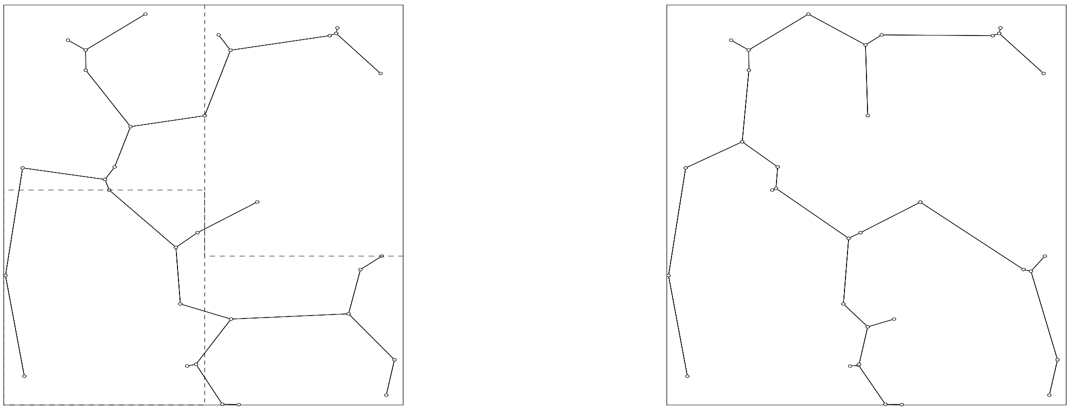
Interpretation of Results

From Table 1 we make the following observations for $n = 25$ and $d = 2, 3$. First, Algorithm 2 consistently outperformed Algorithm 1. In particular, Algorithm 1 usually computed a poor Steiner tree, with cost exceeding that of a minimum spanning tree. By contrast, Algorithm 2 usually computed a Steiner tree with cost less than that of a minimum spanning tree. Thus, the refinements in Algorithm 2 are very important in practice. Second, Algorithm 2 produced reasonably good Steiner trees in the sense that their costs were comparable with those computed by Smith's Algorithm running in more time. For example, with $d = 2$ and $t = 13$, the cost of the Steiner tree computed by Algorithm 2 within 15 seconds was virtually the same as that computed by Smith's Algorithm in 20 seconds. Third, as predicted by the asymptotic performance bound, the performance of Algorithms 1 and 2 improved with increasing values of t, since for larger t the subproblems encompass more of the original problem. However, this improvement took place to an extent smaller than that suggested by the $1 + O(t^{-1/d(d-1)})$ bound, as indicated by the fact that λ_1 and λ_2 decreased with increasing t.

Table 2 shows a significant influence of the parameter t on the running times of Algorithms 1 and 2, underscoring the practical utility of allowing the user to adjust this parameter.

Table 3 shows, that for $n = 60$ and $d = 3$, our distributed implementation of Algorithm 2 offers a substantial speedup over our sequential implementation. For example, with $t = 10$, our distributed implementation ran over 3 times faster than our sequential implementation, and within 22 seconds found a solution within 1% of the cost of that computed by Smith's Algorithm in 200 seconds. For $t < 10$, however, the overhead to distribute the subproblems in our distributed implementation exceeded the time to solve each subproblem. As t increases, the speedup diminishes because larger t produce fewer subproblems.

In Table 3, the result $\hat{R}_2 < 1$ for $t = 31$ means that our distributed implementations of Algo-



(a) (b)

Figure 1: Sample Steiner trees computed by Algorithm 2 and by Smith's Algorithm under the Euclidean norm on 25 terminals randomly chosen from the square $[0, 100]^2$. Tree (a) was computed by Algorithm 2 with $t = 7$ within 6 seconds. Tree (b) was computed by Smith's Algorithm in 20 seconds. Dashed lines in (a) show the partition created by Algorithm 2.

rithm 2 found lower-cost Steiner trees than did Smith's Algorithm running for 200 seconds. This result is striking because our distributed implementation used under 108 seconds and our sequential implementation used under 197 seconds. In fact, on these data sets, it took Smith's Algorithm over 30 minutes to find any Steiner tree at least as good as the ones computed by Algorithm 2 within the aforementioned times. This finding is consistent with the asymptotic theory proven by Kalpakis and Sherman that, for any fixed d, the performance ratio of Algorithms 1 can be made arbitrarily close to 1 by choosing t sufficiently large.

It is particularly interesting that for $t = 31$ Algorithm 2 outperformed Smith's Algorithm given that Algorithm 2 applies Smith's Algorithm to solve each of its subproblems. This paradox is explained by different input sizes: when run alone, Smith's Algorithm worked on $n = 60$ terminals; but when applied to solve the subproblems from Algorithm 2, Smith's Algorithm worked on at most $t = 31$ terminals at a time. For Smith's Algorithm, the quality of its ouput depends on its allowed computation time, and time exponential in the number of terminals is required for it to find a Steiner minimum tree. Smith's Algorithm will eventually converge on a Steiner minimum tree, but when run for limited amounts of time on 60 terminals, lower-cost Steiner trees are found by allocating the computation time among the partitioned subproblems of Algorithm 2 than by running Smith's Algorithm on the entire input. This phenomenon illustrates why partitioning is an effective design strategy for computationally difficult problems.

The choice of parameter τ affects the time and performance of Algorithm 2 and thus complicates the evaluation of Algorithm 2. For this reason, our results should be interpreted as conservative estimates of what is possible with Algorithm 2: it may be possible to achieve even better results with other choices of τ.

Open Questions

Our work raises a number of interesting open questions. In their theoretical analysis, Kalpakis and Sherman assumed that the terminals are independent identically distributed random variables, and in our experiments, we worked exclusively with the uniform distribution. But these assumptions are not always valid in practice. How does Algorithm 2 perform under more general distributions? And for such distributions, are there more effective partitioning techniques than the straightforward one in Algorithms 1 and 2?

Although we made little attempt to optimize our implementations for time, a practitioner who wants to compute Steiner trees on large terminal sets would need to do so. For example, in addition to exploiting parallelism to solve the subproblems simultaneously, one could exploit parallelism within the solution of each subproblem. One could also exploit parallelism to partition the input, to distribute the subproblems, and to compute minimum spanning trees. How can these issues be carefully worked out? Using the best available parallel and distributed machines, what is the largest number of terminals on which Steiner minimum trees can be computed within a reasonable amount of time?

4 Conclusion

We have implemented and experimentally evaluated in R^2 and R^3 a partitioning algorithm analyzed by Kalpakis and Sherman for the geometric Steiner tree problem. Results show that using edge reduction techniques and using an improved method for combining the subproblems significantly reduces the cost of Steiner trees found, in comparison with those found by the basic algorithm. Moreover, with these refinements, the partitioning algorithm is a viable practical algorithm which in less time typically produces results roughly comparable with those computed by Smith's Algorithm. Our distributed implementation provides substantial additional savings in time.

As with all experimental analyses of algorithms, it is important to keep in mind that the experiments compare implementations of algorithms; they do not compare algorithms. Nevertheless, since we made little attempt to optimize our implementation for time, we view our results as conservative estimates of what is possible in practice with the refined partitioning algorithm.

Acknowledgments

We are grateful to Warren D. Smith for providing us with a C language implementation of his GSMT Algorithm. In addition, we appreciate helpful comments from Konstantinos Kalpakis on our implementations and presentation. We also thank Thomas Cain and James Mayfield for editorial comments. All computer work was carried out on a network of Silicon Graphics workstations in the Computer Science Department at the University of Maryland Baltimore County.

References

[1] A. Balakrishnan and N. R. Patel. Problem reduction methods and a tree generation algorithm for the Steiner network problem. *Networks*, 17:65–85, 1987.

[2] W. M. Boyce. An improved program for the full Steiner tree problem. *ACM Transactions on Mathematical Software*, 3(4):359–385, December 1977.

[3] E. J. Cockayne and D. E. Hewgill. Improved computation of plane Steiner minimal trees. *Algorithmica*, 7(2/3):219–229, 1992.

[4] E. J. Cockayne and D. G. Schiller. Computation of Steiner minimal trees. In Welsh and Woddall, editors, *Combinatorics*, pages 53–71. Inst. Math. Appl., 1972.

[5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press and McGraw Hill, Cambridge, Massachusetts, 1990.

[6] K. Kalpakis and A. T. Sherman. Probabilistic analysis of an enhanced partitioning algorithm for the Steiner tree problem in R^d, July 1992 (revised May 1993). Submitted to *Networks*. [Preliminary versions appear as Technical Report CS-TR-2936/UMIACS-TR-92-83, University of Maryland College Park, July 1992; and as Technical Report TR CS-92-10, Computer Science Department, University of Maryland Baltimore County, July 1992.]

[7] K. Kalpakis and A. T. Sherman. Probabilistic analysis of an enhanced partitioning algorithm for the Steiner tree problem in R^d. In P. V. Dooren and M. Spong, editors, *Proceedings of the 13th Annual Allerton Conference on Communication, Control, and Computing*, pages 553–564. University of Illinois at Urbana-Champaign, 1992.

[8] Z. A. Melzak. On the problem of Steiner. *Canadian Mathematical Bulletin*, 4(2):143–148, May 1961.

[9] J. M. Smith, D. T. Lee, and J. S. Liebman. An $O(N \lg N)$ heuristic for Steiner minimal tree problems on the Euclidean metric. *Networks*, 11:23–29, 1981.

[10] W. D. Smith. How to find Steiner minimal trees in Euclidean d-space. *Algorithmica*, 7(2/3):137–177, 1992.

[11] D. Triesch and F. Hwang. An improved algorithm for Steiner trees. *SIAM Journal on Applied Mathematics*, 50(1):244–263, February 1990.

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