Chapter 1
Parallel Computation of Steiner Minimal Trees

Frederick C. Harris, Jr.*

Abstract

Given a set of N cities, construct a connected network which has minimum length. The problem is simple enough, but the catch is that you are allowed to add junctions in your network. Therefore the problem becomes how many extra junctions should be added, and where should they be placed so as to minimize the overall network length.

This intriguing optimization problem is known as the Steiner Minimal Tree Problem (SMT), where the junctions that are added to the network are called Steiner Points.

The focus of this paper is the parallel computation for the generation of what Pawel Winter termed T_list and its implementation. This generation of T_list is followed by the extraction of the proper answer. When Winter developed his algorithm, the time for extraction dominated the overall computation time. After Cockayne and Hewgill's work, the time to generate T_list dominated the overall computation time. The ideas we present were implemented in a program called PARSTEINER94, and the results show that the time to generate T_list has now been cut by an order of magnitude. So now the extraction time once again dominates the overall computation time.

1 Introduction

Minimizing a network's length is one of the oldest optimization problems in mathematics and, consequently, it has been worked on by many of the leading mathematicians in history. In the mid-seventeenth century a simple problem was posed: Find the point P that minimizes the sum of the distances from P to each of three given points in the plane. Solutions to this problem were derived independently by Fermat, Torricelli and Cavalieri. They all deduced that either P is inside the triangle formed by the given points and that the angles at P formed by the lines joining P to the three points are all 120°, or P is one of the three vertices and the angle at P formed by the lines joining P to the other two points is greater than or equal to 120°.

In the nineteenth century a mathematician at the University of Berlin, named Jakob Steiner, studied this problem and generalized it to include an arbitrarily large set of points in the plane. This generalization created a star when P was connected to all the given points in the plane, and is a geometric approach to the 2-dimensional center of mass problem.

In 1934 Kössler and Jarník generalized the network minimization problem even further [9]: Given n points in the plane find the shortest possible connected network containing these points. This generalized problem, however, did not become popular until the book, *What is Mathematics*, by Courant and Robbins [6], appeared in 1941. Courant and Robbins linked the name Steiner with this form of the problem proposed by Kössler and Jarník, and it became known as the Steiner Minimal Tree problem. The general solution to this problem allows multiple points to be added, each of which is called a Steiner Point.

*Department of Computer Science, University of Nevada, Reno, NV 89557-0148, fredh@cs.unr.edu

Much is known about the exact solution to the Steiner Minimal Tree problem. Those who wish to learn about some of the spin-off problems are invited to read the introductory article by Bern and Graham [1], the excellent survey paper on this problem by Hwang and Richards [7], or the recent volume in The Annals of Discrete Mathematics devoted completely to Steiner Tree problems [8]. Some of the basic pieces of information about the Steiner Minimal Tree problem that can be gleaned from these articles are: (i) Melzak developed the first algorithm for Steiner Minimal Trees in 1961 [10], (ii) all of the original n points will be of degree 1, 2, or 3, (iii) the Steiner Points are all of degree 3, (iv) any two edges meet at an angle of at least 120° in the Steiner Minimal Tree, and (v) at most $n-2$ Steiner Points will be added to the network.

This paper concentrates on the Steiner Minimal Tree problem, henceforth referred to as the SMT problem. In Section 2 we review the known algorithms for calculating Steiner Minimal Trees. In Section 3 we present the first parallel algorithm for calculating SMT's. Some new results are presented in Section 4, with conclusions and future work in Section 5.

2 Current Implementations

The first computational results were presented by Cockayne and Schiller [5]. Their program, called STEINER, was written in FORTRAN on an IBM 360/50 at the University of Victoria. STEINER could calculate the SMT for any 7 point problem in less than 5 minutes of cpu time. When the problem size was increased to 8 the SMT could be found if 7 of the vertices were on the Steiner Hull. When this condition held it could calculate the SMT in under 10 minutes, but if this condition did not hold it would take an unreasonable amount of time.

Cockayne called STEINER a prototype for calculating SMT's and allowed Boyce and Seery of Bell Labs to obtain a copy of his code to improve the work. They improved the code, renamed it STEINER72, and were able to calculate the SMT for all 9 point problems and most 10 point problems in a reasonable amount of time [2]. Boyce and Seery continued their work and developed another version of the code that they thought could solve problems of size up to 12 points, but they reported no computation times.

Melzak had identified invalid tree structures in his work [10] but these were not characterized until Pawel Winter's work in 1981. Winter's characterizations were based upon several geometric constructions which enable many of the possible combinations previously generated to be eliminated. He implemented these improvements in a program called GEOSTEINER [12] which he wrote in SIMULA 67 on a UNIVAC-1100.

In his work, Winter split the computation process into two phases. The first phase, called T_list Generation, generates a set of Full Steiner Trees (FST's) which have n vertices and $n-2$ Steiner Points. The second phase is called Extraction and is a branch and bound process, primed with the length of the MST, looking for the shortest subset that covers all the original vertices.

GEOSTEINER could calculate in under 20 seconds SMT's for all randomly generated sets with 15 points. This improvement was put into focus when he mentioned that all previous implementations took more than an hour for non-degenerate problems of size 10 or more. Winter tried randomly-generated 20-point problems but did not give results since some of them did not finish in his cpu time limit of 30 seconds. He made only two comments for problems bigger than size 15: (i) the extraction time was dominating the overall computation time and (ii) "with further improvements, it is reasonable to assert that point sets up to 30 V-points could be solved in less than an hour [12]."

The next major program, EDSTEINER86, was developed in FORTRAN on an IBM 4381 by Cockayne and Hewgill [3]. This implementation was based upon Winter's results but had enhancements in the extraction process. EDSTEINER86 was able to calculate the FST for 79 out of 100 randomly-generated 32-point problems. For these 79 problems the cpu time for T_list varied from 1 to 5 minutes, while the extraction time never exceeded 70 seconds.

Cockayne and Hewgill subsequently improved their SMT program and renamed it EDSTEINER89 [4]. This improvement was completely focused on the extraction process. EDSTEINER89 was still written in FORTRAN but was run on a SUN 3/60 workstation. They randomly generated 200 32-point problems to solve and found that the generation of T_list dominated the computation time for problems of this size. The average time for T_list generation was 438 seconds while the average time for forest management and extraction averaged only 43 seconds. They then focused on 100 point problems and set a cpu limit of 12 hours. The average cpu time to generate T_list was 209 minutes for these problems, but only 77 finished the extraction in the cpu time limit. These programs and their results are summarized in Table 1.

Program	Author(s)	Institution	Points
STEINER	Cockayne & Schiller	Univ of Victoria	7
STEINER72	Boyce & Seery	ATT Bell Labs	10
STEINER73	Boyce & Seery	ATT Bell Labs	12
GEOSTEINER	Winter	Univ of Copenhagen	15
EDSTEINER86	Cockayne & Hewgill	Univ of Victoria	30
EDSTEINER89	Cockayne & Hewgill	Univ of Victoria	100

TABLE 1

SMT Programs, authors, and results.

3 Parallel Algorithm

Since the sequential version of the code does not lend itself to easy parallelization, one needs to back up and develop an understanding for how the algorithm works. The first thing that is obvious from the code is that you select a left subtree and then try to mate it with possible right subtrees. These subtrees, and the resulting children, are kept on Q_list which was primed with all the original vertices. Upon further examination we come to the conclusion that a left tree will mate with all trees that are shorter than it, and all trees of the same height that appear after it on Q_list, but it will never mate with any tree that is taller.

This analysis yields the need for the first function in our parallel algorithm which we will call `a_before_b()`. This piece of code will determine if node a would have come before node b in Winter's Q_list. This is necessary since in a parallel implementation the calculations and additions to Q_list will not happen in the same order as they would have in the sequential version. With this function we are able to finish the function `mate()`. This is a boolean function which determines if node a (left) can mate with node b (right).

Now that the foundational building blocks are completed, design attention can be turned to the main algorithm. The description of this is in a master–slave perspective. This perspective was taken due to the structure of most parallel architectures, as well as the fact that all nodes on the Q_list need a sequencing number assigned to them. The master

will therefore be responsible for numbering the nodes and maintaining the main Q_list and T_list.

The idea for the master–slave interaction came from a paper by Quinn and Deo [11], on parallel algorithms for Branch-and-Bound problems. Their idea was to let the master have a list of work that needs to be done. Each slave is assigned to a processor. Each slave requests work, is given some, and during its processing creates more work to be done. This new work is placed in the master's work list, which is sorted in some fashion. When a slave runs out of work to do, it requests more from the master. They noted that this leaves some processors idle at times (particularly when the problem was starting and stopping), but this approach provides the best utilization if all branches are independent.

This description almost perfectly matches the problem at hand. First, we will probably have a fixed number of processors which can be determined at runtime. Second we have a list of work that needs to be done. The hard part is implementing a sorted work list in order to obtain a better utilization. This is implemented in what we term the Proc_list, which is a list of the processes that either are currently running or have not yet started. This list is primed with the information about the initial members of Q_list, and for every new node put on the Q_list, a node which contains information about the Q_list node, is placed on the Proc_list in the a_before_b() sorted order.

4 Results

From the literature it is obvious that the current standard for calculating SMT's has been established by Cockayne and Hewgill. Their work on SMT's has pushed the boundary of computation out from the 15-point problems of Winter to a large percentage of 100-point problems.

Cockayne and Hewgill, in their investigation of the effectiveness of EDSTEINER89, randomly generated 100 problems with 100 points inside the unit square. They set up a CPU limit of 12 hours, and 77 of 100 problems finished within that limit. They described the average execution times as follows: T_list construction averaged 209 minutes, Forest Management averaged 27 minutes, and Extraction averaged 10.8 minutes.

The parallel algorithms we have outlined here have been implemented in a program called PARSTEINER94. This implementation is only the second SMT program since Winter's GEOSTEINER in 1981 and is the first parallel code. The major reason that the number of SMT programs is so small is that any implementation is necessarily complex.

While preparing the code for this work, Cockayne and Hewgill were kind enough to supply us with 40 of the problems generated for [4] along with their execution times. These data sets were given as input to PARSTEINER94, and the calculation timed. For all 40 cases, the average time to generate T_list was less than 20 minutes. This is exciting because we have been able to generate T_list properly, while cutting an order of magnitude off the time. A sample of what T_list for a 100 point problem looks like is presented in Figure 1, while the extracted answer is in Figure 2. Table 2 contains a comparison of the times of the first 10 cases just described.

5 Conclusions and Future Work

These results are quite promising for various reasons. First, the parallel implementation presented in this work is scalable, and therefore could be run with many more processors, thereby enhancing the speedup provided. Second, with the PVM platform used, we can in the future port this work to a real parallel MIMD machine, which will have much less

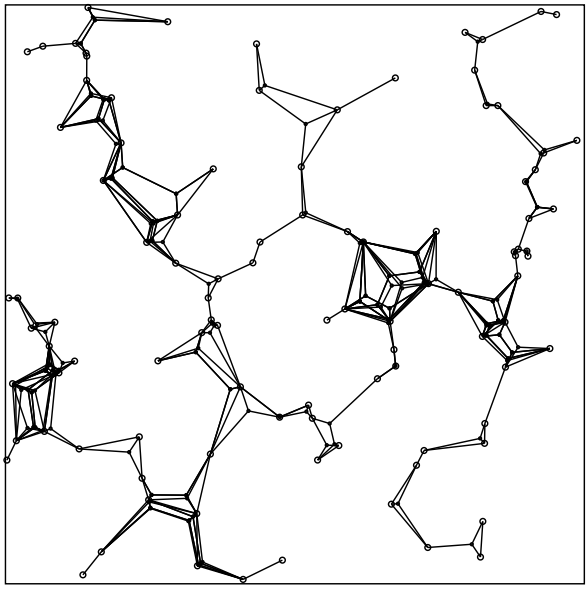

FIG. 1. *T_list for a random set of points.*

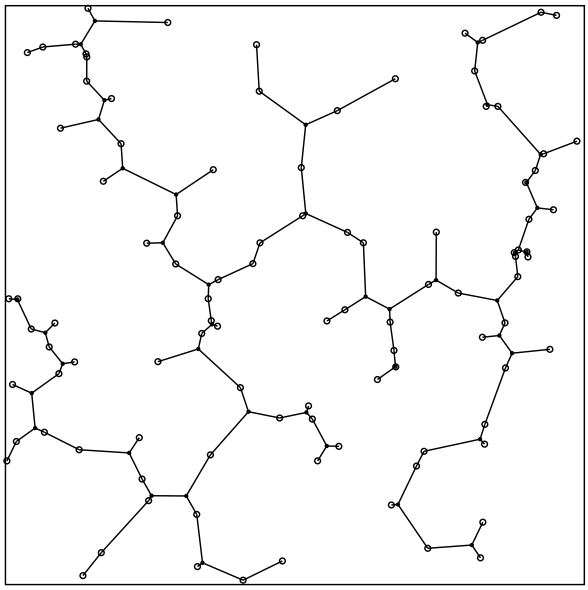

FIG. 2. *SMT extracted from T_list for a random set of points.*

Test Case	PARSTEINER94	EDSTEINER89
1	650	8597
2	1031	13466
3	1047	15872
4	1687	17061
5	874	13258
6	1033	15226
7	1164	12976
8	1109	16697
9	975	15354
10	554	8650

TABLE 2

Comparison of T_list computation times (in seconds).

communication overhead, or to a shared memory machine, where the communication could all but be eliminated. In either case we would expect the speedup to improve much more.

Enhancements are already under way to parallelize the Extraction process and thereby speed up the calculation even further. A new method of parallelizing the T_list generation is also currently being looked into which at first glance would appear to decrease the communication and would thereby allow the calculation to proceed with less delays.

References

[1] M. Bern and R. Graham, *The shortest-network problem*, Sci. Am., 260 (1989), pp. 84–89.

[2] W. Boyce and J. Seery, *STEINER 72 – an improved version of Cockayne and Schiller's program STEINER for the minimal network problem*, Tech. Rep. 35, Bell Labs., Dept. of Computer Science, 1975.

[3] E. Cockayne and D. Hewgill, *Exact computation of Steiner minimal trees in the plane*, Info. Process. Lett., 22 (1986), pp. 151–156.

[4] ——, *Improved computation of plane Steiner minimal trees*, Algorithmica, 7 (1992), pp. 219–229.

[5] E. Cockayne and D. Schiller, *Computation of Steiner minimal trees*, in Combinatorics, D. Welsh and D. Woodall, eds., Maitland House, Warrior Square, Southend-on-Sea, Essex SS1 2J4, 1972, Mathematical Institute, Oxford, Inst. Math. Appl., pp. 52–71.

[6] R. Courant and H. Robbins, *What is Mathematics? an elementary approach to ideas and methods*, Oxford University Press, London, 1941.

[7] F. Hwang and D. Richards, *Steiner tree problems*, Networks, 22 (1992), pp. 55–89.

[8] F. Hwang, D. Richards, and P. Winter, *The Steiner Tree Problem*, vol. 53 of Ann. Discrete Math., North-Holland, Amsterdam, 1992.

[9] V. Jarník and O. Kössler, *O minimálnich gratech obsahujicich n daných bodu [in Czech]*, Casopis Pesk. Mat. Fyr., 63 (1934), pp. 223–235.

[10] Z. Melzak, *On the problem of Steiner*, Canad. Math. Bull., 4 (1961), pp. 143–150.

[11] M. Quinn and N. Deo, *An upper bound for the speedup of parallel best-bound branch-and-bound algorithms*, BIT, 26 (1986), pp. 35–43.

[12] P. Winter, *An algorithm for the Steiner problem in the Euclidian plane*, Networks, 15 (1985), pp. 323–345.