Toward a Steiner Engine:
Enhanced Serial and Parallel Implementations
of the Iterated 1-Steiner MRST Heuristic

Tim Barrera, Jeff Griffith, Sally A. McKee,,
Gabriel Robins, Tongtong Zhang

Technical Report No. CS-92-40
December 01, 1992

Toward a Steiner Engine: Enhanced Serial and Parallel Implementations of the Iterated 1-Steiner MRST Heuristic

Tim Barrera, Jeff Griffith, Sally A. McKee,
Gabriel Robins and Tongtong Zhang

Computer Science Department, Thornton Hall,
University of Virginia, Charlottesville, VA 22903-2442

Abstract

The minimum rectilinear Steiner tree (MRST) problem arises in global routing and wiring estimation, as well as in many other areas. The MRST problem is known to be NP-hard, and the best performing MRST heuristic to date is the Iterated 1-Steiner (I1S) method recently proposed by Kahng and Robins [13]. I1S achieves significantly improved average-case performance while avoiding the worst-case examples from which other approaches suffer, yet the algorithm has heretofore lacked a practical implementation. In this paper we develop a straightforward, efficient implementation of I1S, achieving speedup factors of over 200 compared to previous implementations. We also propose a parallel implementation of I1S that achieves near-linear speedup on K processors. Extensive empirical testing confirms the viability of our approaches, which allow for the first time the benchmarking of I1S on nets containing several hundred pins.

1 Introduction

The minimum rectilinear Steiner tree problem is central to VLSI physical design phases such as global routing and wiring estimation, where we seek low-cost topologies to interconnect the pins of signal nets [17]:

The Minimum Rectilinear Steiner Tree (MRST) problem: Given a planar set P of n points, find a set S of *Steiner points* such that the minimum spanning tree (MST) over $P \cup S$ has minimum cost.

The cost of a tree edge is the Manhattan distance between its endpoints, and the cost of a tree is the sum of its edge costs. Figure 1 shows an MST and an MRST for the same four-pin net.

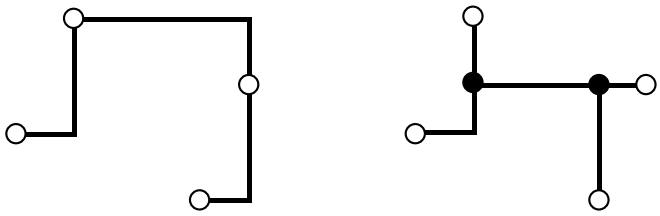

Figure 1: A minimum spanning tree (left) and MRST (right) for a set of 4 pins.

Several fundamental results have guided research on the MRST problem. First, Hanan [7] has shown that there always exists an MRST with Steiner points chosen from the intersection of all the horizontal and vertical lines passing through each point in P; indeed, this result generalizes to higher dimensions [22]. Unfortunately, a second major result established that despite this restriction on the solution space, the MRST problem remains NP-complete [4]. This has prompted a large number of heuristics for the MRST problem, as surveyed in [12].

In solving intractable problems, we often seek provably good heuristics having bounded worst-case error from optimal. Thus, a third important result established that the rectilinear MST is a fairly good approximation to the MRST, with a worst-case performance ratio of $\frac{cost(MST)}{cost(MRST)} \leq \frac{3}{2}$ [10], implying that any MST-based strategy which improves upon an initial MST topology will also enjoy a performance ratio of at most $\frac{3}{2}$. This encouraged the development of a number of Steiner tree heuristics which resemble classic MST construction methods [8] [9] [11] [15] [16], all of which produce trees having average cost 7% to 9% smaller than MST cost [19] [24]. However, Kahng and Robins [14] recently characterized the limitations of MST-based MRST heuristics by giving a class of examples on which the $\frac{3}{2}$ bound is tight, i.e. the MST can be unimprovable.

This negative result has motivated research into alternate schemes for MRST approximation, among which the one with best performance is the recently proposed Iterated 1-Steiner algorithm [13]. The Iterated 1-Steiner method (I1S) is the first known MRST heuristic which always performs better than $\frac{3}{2}$ times optimal [20], and for which a uniform performance ratio of $\frac{11}{8}$ has been shown [2] using the methods of [25]. I1S also performs very well in practice, achieving 11% average improvement over MST cost, which is within 0.5% of the average optimal cost [21].

This performance success was achieved at the expense of a high time complexity, and although a variant of I1S can be implemented to run within time $O(r \cdot n^2 \log n)$, the computational geometric methods employed to achieve this time bound hide large constants and are difficult to code. Thus, actual previous implementations of this variant use a naive approach, and require time $O(r \cdot n^4 \log n)$, where $r \leq n$ corresponds to the number of "rounds".

The primary contribution of this paper is a practical $O(r \cdot n^3)$ time implementation of I1S, establishing the practicality and viability of the iterated 1-Steiner method. For $n = 100$ our new implementation is about 247 times faster than the previous, naive implementation. This enables us for the first time to observe the behavior of I1S on several hundred points.

Since a typical CAD environment consists of a network of workstations, exploiting the available large-scale parallelism provides a natural and effective means of improving the performance of CAD algorithms. A second contribution of our work is a parallel version of I1S that achieves high parallel speedups. Since Steiner tree construction is a very computationally expensive part of global routing, our parallel implementation may be viewed as an important first step toward obtaining a "Steiner engine", i.e. an efficient tool for producing near-optimal Steiner trees. Finally, the 1-Steiner method generalizes to arbitrarily weighted graphs, and is thus suitable for a basis of a global router, where obstruction and congestion considerations affect routing.

In Section 2 we review the I1S method. Section 3 outlines our enhanced implementation of I1S. Section 4 discusses the parallel implementation, and Section 5 presents simulation results regarding performance, running times and parallel speedups. We conclude in Section 6 with directions for further research.

2 Overview of the Iterated 1-Steiner Approach

We begin with a review of the 1-Steiner method of [13]. For two pointsets P and S we define the MST savings $\Delta MST(P, S) = cost(MST(P)) - cost(MST(P \cup S))$. We use H(P) to denote the set of Hanan Steiner point candidates. For a pointset P, a 1-Steiner point $x \in H(P)$ maximizes $\Delta MST(P, \{x\}) > 0$. The I1S method repeatedly finds 1-Steiner points and includes them into S. The cost of the MST over $P \cup S$ will decrease with each added point, and the construction

terminates when there is no x with $\Delta MST(P \cup S, \{x\}) > 0$. Although a Steiner tree may contain at most $n-2$ Steiner points [6], I1S may add $n-1$ Steiner points or more; thus we eliminate any extraneous Steiner points having degree 2 or less in the MST. Figure 2 illustrates a sample execution of I1S, and Figure 3 describes the algorithm formally.

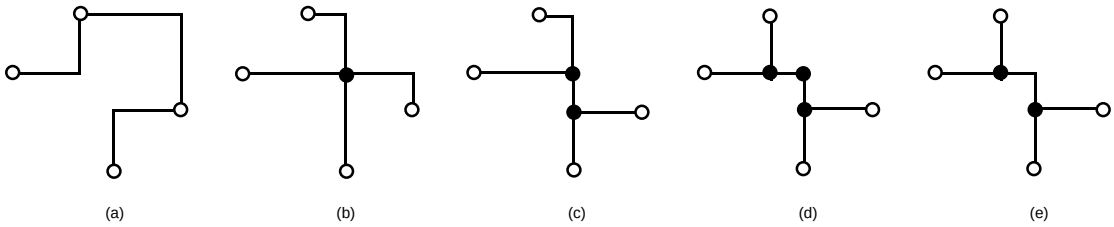

Figure 2: Execution of Iterated 1-Steiner (I1S) on a 4-point example. Note that in step (d) a degree-2 Steiner point is formed and is thus eliminated from the topology.

Algorithm Iterated 1-Steiner (I1S)
Input: A set P of n points **Output:** A rectilinear Steiner tree over P

$S = \emptyset$
While $T = \{x \in H(P) \mid \Delta MST(P, \{x\}) > 0\} \neq \emptyset$ **Do**
  Find $x \in T$ with maximum $\Delta MST(P, \{x\})$
  $S = S \cup \{x\}$
  Remove from S points with degree ≤ 2 in $MST(P \cup S)$
Output MST$(P \cup S)$

Figure 3: The Iterated 1-Steiner (I1S) algorithm.

Although a 1-Steiner point may be found in $O(n^2)$ time using complicated techniques from computational geometry [5] [13] [18], such methods suffer from large constants in their time complexities, and are notoriously difficult to implement. We therefore combine a "batched" version of I1S with an incremental MST update scheme (described in Section 3 below) to efficiently add an entire set of "independent" Steiner points in a single iteration, thereby obtaining a practical algorithm with a reduced time complexity.

Following Hanan's result, for each candidate Steiner point $x \in H(P)$ the Batched 1-Steiner (B1S) variant computes the induced MST savings $\Delta MST(P, \{x\})$. Next we select a maximal "independent" set of Steiner points, where the criterion for independence is that no candidate

Steiner point is allowed to reduce the potential MST cost savings of any other candidate. Formally, candidate Steiner points x and y are independent (and thus may be added in the same round) only if $\Delta MST(P, \{x\}) + \Delta MST(P, \{y\}) \leq \Delta MST(P, \{x, y\})$.

Once a maximal set S of independent Steiner points has been determined, it is inserted into P, and we iterate this process with $P := P \cup S$ until we reach a round that fails to induce a Steiner point. Clearly, the total time required by B1S is $O(n^3)$ per round. In practice we observed that the number of rounds is a small constant; for example, when $n = 300$, the average number of rounds is about 2.5. The B1S algorithm is summarized in Figure 4.

Algorithm Batched 1-Steiner (B1S)
Input: A set P of n points **Output:** A rectilinear Steiner tree over P

While $T = \{x \in H(P) \mid \Delta MST(P, \{x\}) > 0\} \neq \emptyset$ **Do**
$S = \emptyset$
For $x \in \{T$ in order of non-increasing $\Delta MST\}$ **Do**
If $\Delta MST(P \cup S, \{x\}) \geq \Delta MST(P, \{x\})$ **Then** $S = S \cup \{x\}$
$P = P \cup S$
Remove from P Steiner points with degree ≤ 2 in MST(P)
Output MST(P)

Figure 4: The Batched 1-Steiner (B1S) algorithm.

3 An Enhanced Serial Implementation

We now describe an efficient yet simple incremental MST update algorithm. In computing the MST savings of each of the n^2 Steiner candidates, a key observation is that once we have computed an MST over the pointset P, the addition of a single new point x into P can only induce a small constant number of changes between $MST(P)$ and $MST(P \cup \{x\})$. This follows from the observation that each point can have at most 8 neighbors in a rectilinear MST, i.e. at most one per octant.

This suggests the following linear-time algorithm for dynamic MST maintenance: connect the new point to each of its potential neighbors, then delete the longest edge on each resulting cycle. An example of this method is given in Figure 5, while Figure 6 describes it formally. Note

that dynamic MST maintenance may be achieved in sub-linear time [3], but such methods seem impractical due to their complexity and large hidden constants.

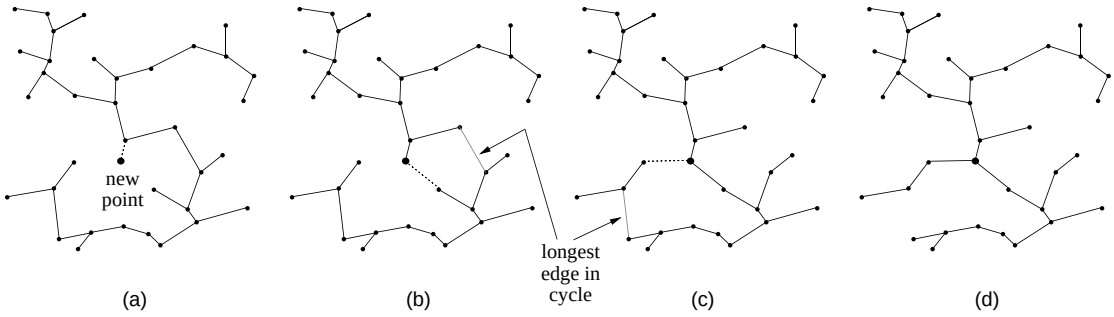

Figure 5: Dynamic MST maintenance: adding a point to an existing MST entails connecting the point to its closest neighbor in each octant, and deleting the longest edge on each resulting cycle (the Euclidean metric has been used for clarity in this example).

Dynamic MST Maintenance (MSTM)
Input: A set P of n points, $MST(P)$, a new point x **Output:** MST($P \cup \{x\}$)

$T = MST(P)$
For $i = 1$ to 8 **do**
  Find in octant i (with respect to x) the point $p \in P$ closest to x
  Add to T edge (p, x)
  If T contains a cycle **Then** remove from T the longest edge on the cycle
Output T

Figure 6: Linear-time dynamic MST maintenance.

4 A Parallel Implementation

The I1S algorithm is highly parallelizable since each one of p processors can compute the MST savings of $\frac{n^2}{p}$ of the Steiner candidates. In our implementation, all processors send their best candidate to a master processor, which selects the best of these candidates for inclusion into the pointset. This procedure is iterated until no improving candidates can be found. The B1S algorithm parallelizes similarly.

The Parallel Virtual Machine (PVM) system [1] [23] is used to control remote processors.

PVM is a software package that allows a heterogeneous network of parallel and serial computers to be used as a single computational resource. The PVM system consists of two parts, a daemon process and a user library, providing mechanisms for initiating processes on other machines and for controlling synchronization and communication among processes.

Each of the processors in our implementation is a SUN4 workstation, communicating over an ethernet. The master processor sends to the available processors equal sized subsets of of the Hanan candidate set H(P), and the computation/response time of each processor is tracked. If any individual processor is determined to be considerably slower than the rest, it is henceforth given smaller tasks to perform. If a processor does not complete a task within reasonable time, it is sent an abort message, and the task is reassigned to the fastest idle processor available. This prevents individual slow processors from seriously impeding the speed of the overall computation.

5 Experimental Results

We have implemented both serial and parallel versions of the I1S and the B1S algorithms, using C++ in the SUN workstation environment. The serial I1S and B1S heuristics have been benchmarked for various pointset cardinalities: up to 1000 instances of each cardinality were solved using both of the I1S and B1S algorithms. The instances were generated randomly by choosing x and y coordinates from a uniform distribution in the 1000×1000 grid; such instances are statistically similar to the pin locations of actual VLSI nets, and are the standard testbed for Steiner tree heuristics [13] [12]. Performance results are summarized in Tables 1 and 2 in the Appendix, and are illustrated in Figures 7 through 9. Both I1S and B1S yield Steiner trees with cost averaging about 11% smaller than the MST cost; this is within 0.5% of the average *optimal* Steiner tree cost[21].

We timed the execution of the serial and parallel versions of I1S and B1S, using both the naive $O(r \cdot n^4 \log n)$ implementation [13] and our new implementation which incorporates the efficient, dynamic MST maintenance as described in Section 3. The parallel implementation uses 9 SUN 4/40 (IPC) workstations, with a SUN 4/75 (Sparc2) as the master processor. The improvements in running time are dramatic: for $n = 100$ our serial implementation is 247 times

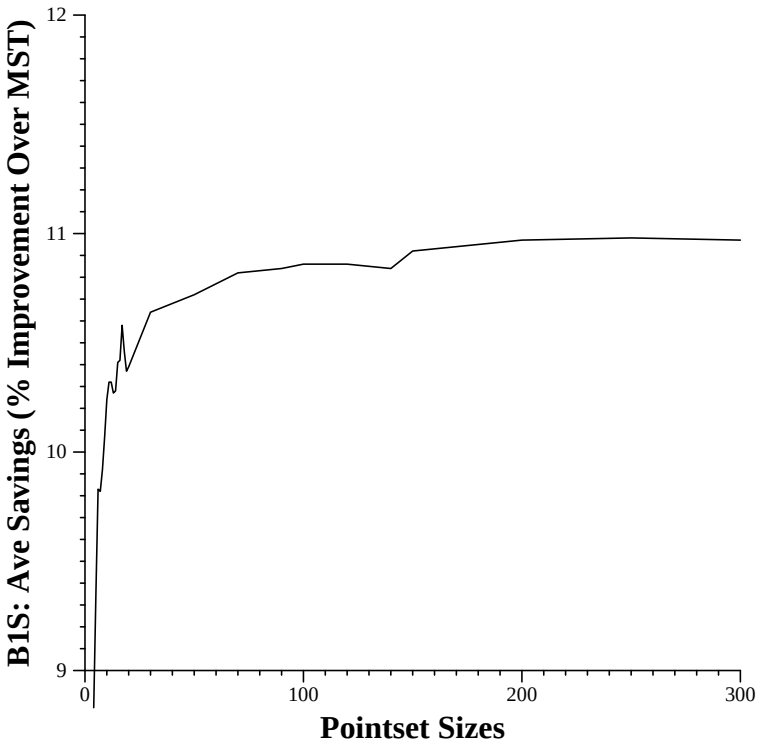

Figure 7: Average performance of the Batched 1-Steiner algorithm, shown as percent improvement over MST cost.

faster than the naive implementation, and our parallel implementation running on 10 processors is 1163 times faster. Detailed execution timings for various cardinalities are given in Tables 3 and 4 in the Appendix, and are illustrated in Figure 10. Even for small pointsets, our new implementations are considerably faster than the previous, naive ones: for example, for $n = 5$ the new serial B1S is on average 2 times faster than the naive implementation, while for $n = 10$ the serial speedup factor approaches 12. The speedup of course increases with the pointset cardinality. The parallel speedup we achieve also increases with the problem size, reaching about 7.2 for $n = 250$ on 10 processors.

6 Conclusion

We have proposed an enhanced implementation of the Iterated 1-Steiner and Batched 1-Steiner heuristics of [13], and have achieved execution speeds about 250 times faster than previous

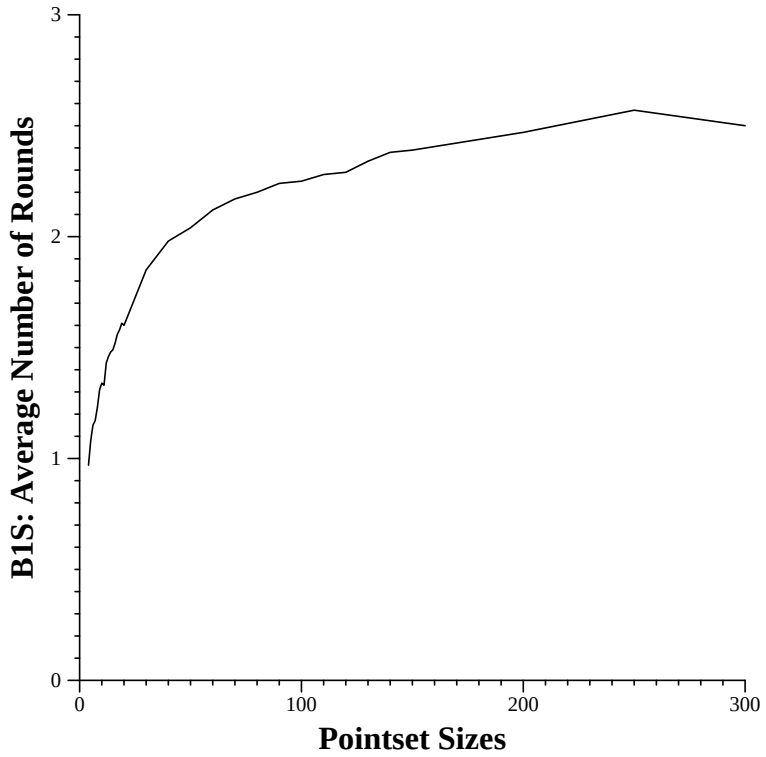

Figure 8: Average number of rounds for the Batched 1-Steiner algorithm.

implementations. This speedup has enabled the testing of I1S and B1S on several hundred points for the first time, and for these large pointsets the average performance of the 1-Steiner method consistently achieves around 11% improvement over MST cost.

We have also implemented parallel versions of both I1S and B1S that achieve high parallel speedups, confirming that the 1-Steiner approach is highly suitable for parallelization. The 1-Steiner approach generalizes to arbitrary weighted graphs, and is thus suitable to support a global router, where obstructions and congestion must also be considered while routing. Since Steiner tree construction is a very computationally expensive component of global routing, our implementations suggest the feasibility of a "Steiner engine" for efficiently computing near-optimal Steiner trees.

One obvious future research direction would be to reduce even further the time complexity of the 1-Steiner approach. Another extension is to investigate Iterated 1-Steiner performance

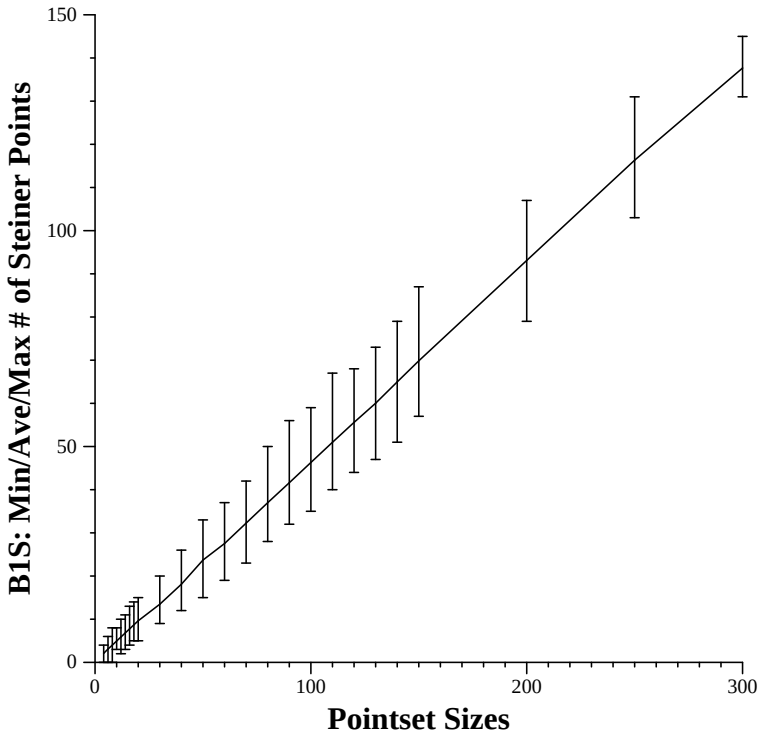

Figure 9: Average number of Steiner points induced by the Batched 1-Steiner algorithm; vertical bars indicate for each pointset cardinality the range of the minimum and maximum number of Steiner points.

in higher dimensions, since both I1S and B1S readily generalize to all dimensions. Finally, our new code can be used to examine the asymptotics of the MST/Steiner ratio for much larger pointsets than was previously possible (see Figure 11).

References

[1] A. Beguelin, J. J. Dongarra, G. A. Geist, R. Manchek, and V. S. Sunderam, *A User's Guide to PVM: Parallel Virtual Machine*, Tech. Rep. ORNL/TR-11826, Oak Ridge National Laboratory, 1991.

[2] P. Berman and V. Ramaiyer, *Improved Approximations for the Steiner Tree Problem*, in Proc. ACM/SIAM Simposium on Discrete Algorithms, San Francisco, CA, January 1992, pp. 325–334.

[3] G. N. Fredrickson, *Data Structures for On-Line Updating of Minimum Spanning Trees*, SIAM J. Comput., 14 (1985), pp. 781–798.

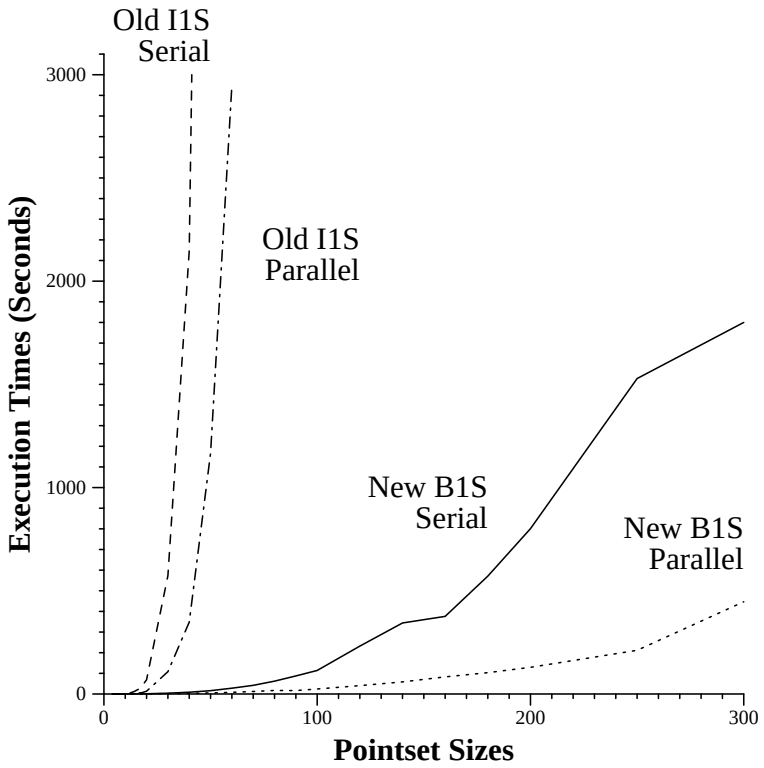

Figure 10: Average execution times (in seconds) for the serial and parallel B1S, using both the naive MST implementation and our new incremental MST maintenance scheme.

[4] M. Garey and D. S. Johnson, *The Rectilinear Steiner Problem is NP-Complete*, SIAM J. Applied Math., 32 (1977), pp. 826–834.

[5] G. Georgakopoulos and C. H. Papadimitriou, *The 1-Steiner Tree Problem*, J. Algorithms, 8 (1987), pp. 122–130.

[6] E. N. Gilbert and H. O. Pollak, *Steiner Minimal Trees*, SIAM J. Applied Math., 16 (1968), pp. 1–29.

[7] M. Hanan, *On Steiner's Problem With Rectilinear Distance*, SIAM J. Applied Math., 14 (1966), pp. 255–265.

[8] N. Hasan, G. Vijayan, and C. K. Wong, *A Neighborhood Improvement Algorithm for Rectilinear Steiner Trees*, in Proc. IEEE Intl. Symp. on Circuits and Systems, New Orleans, LA, 1990.

[9] J.-M. Ho, G. Vijayan, and C. K. Wong, *New Algorithms for the Rectilinear Steiner Tree Problem*, IEEE Trans. on Computer-Aided Design, 9 (1990), pp. 185–193.

[10] F. K. Hwang, *On Steiner Minimal Trees with Rectilinear Distance*, SIAM J. Applied Math., 30 (1976), pp. 104–114.

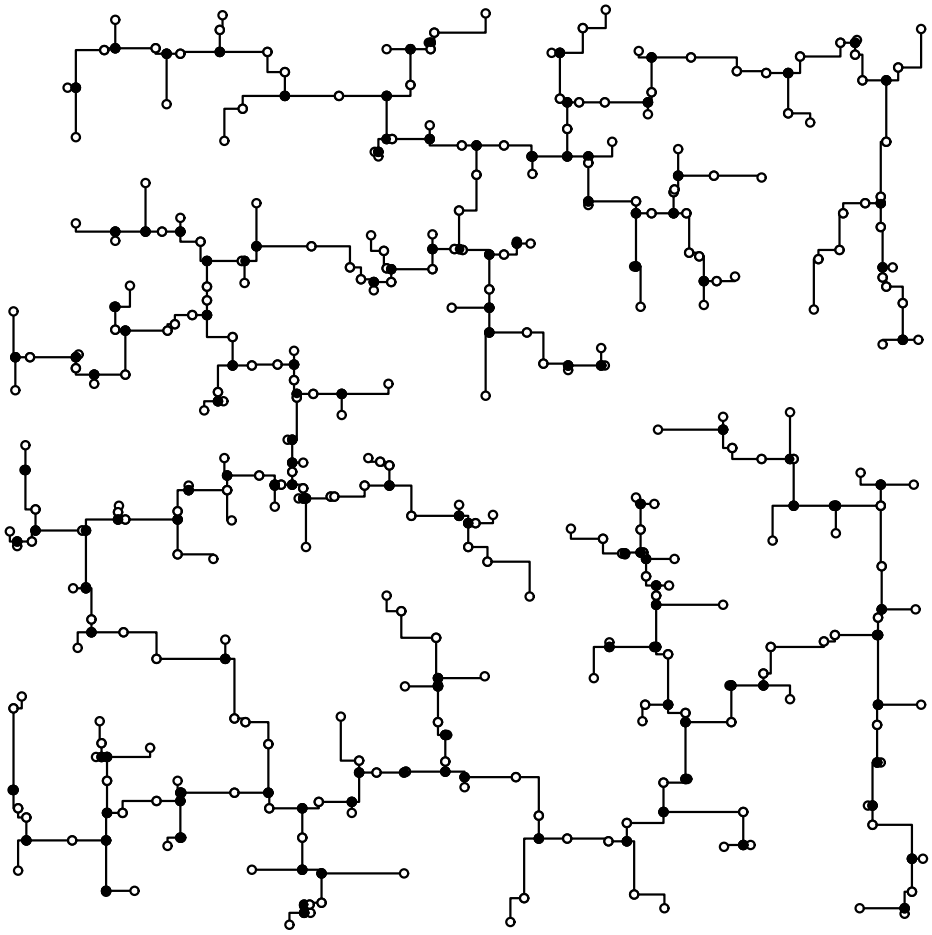

Figure 11: An example of the output of B1S on a random 300-point set (hollow dots). The Steiner points produced by our algorithm are denoted by dark solid dots.

[11] ——, *An O(n log n) Algorithm for Rectilinear Minimal Spanning Trees*, J. ACM, 26 (1979), pp. 177–182.

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

[13] A. B. Kahng and G. Robins, *A New Class of Iterative Steiner Tree Heuristics With Good Performance*, IEEE Trans. on Computer-Aided Design, 11 (1992), pp. 893–902.

[14] ——, *On Performance Bounds for a Class of Rectilinear Steiner Tree Heuristics in Arbitrary Dimension*, IEEE Trans. on Computer-Aided Design (to appear), (1992).

[15] J. H. Lee, N. K. Bose, and F. K. Hwang, *Use of Steiner's Problem in Sub-Optimal Routing in Rectilinear Metric*, IEEE Trans. on Circuits and Systems, 23 (1976), pp. 470–476.

[16] K. W. Lee and C. Sechen, *A New Global Router for Row-Based Layout*, in Proc. IEEE Intl. Conf. on Computer-Aided Design, Santa Clara, CA, November 1990, pp. 180–183.

[17] B. T. Preas and M. J. Lorenzetti, *Physical Design Automation of VLSI Systems*, Benjamin/Cummings, Menlo Park, CA, 1988.

[18] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.

[19] D. Richards, *Fast Heuristic Algorithms for Rectilinear Steiner Trees*, Algorithmica, 4 (1989), pp. 191–207.

[20] G. Robins, *On Optimal Interconnections*, Ph.D. dissertation, Department of Computer Science, UCLA, 1992.

[21] J. S. Salowe, *On Exact Algorithms for Rectilinear Steiner Minimal Trees.* manuscript, November 1992.

[22] T. L. Snyder, *On the Exact Location of Steiner Points in General Dimension*, SIAM J. Comput., 21 (1992), pp. 163–180.

[23] V. S. Sunderam, *PVM: A Framework for Parallel Distributed Computing*, Concurrency: Practice and Experience, 2 (1990), pp. 315–339.

[24] P. Winter, *Steiner Problem in Networks: A Survey*, Networks, 17 (1987), pp. 129–167.

[25] A. Z. Zelikovsky, *The 11/6 Approximation Algorithm for the Steiner Problem on Networks*, Information and Computation (to appear), (1992).

7 Appendix: Performance Data

	Iterated 1-Steiner					
	Performance			Number of SPs		
$\|P\|$	Min	Ave	Max	Min	Ave	Max
4	0.00	8.77	26.06	0	1.08	3
5	0.00	9.25	24.33	0	1.56	4
6	0.00	9.92	25.67	0	2.16	4
7	1.31	9.81	22.12	1	2.49	5
8	1.65	10.05	23.37	1	2.96	5
9	0.44	10.16	20.46	1	3.43	6
10	1.01	10.15	20.86	2	3.86	7
12	1.37	10.28	18.66	2	4.77	8
14	3.01	10.27	17.66	2	5.66	9
16	3.08	10.48	18.21	4	6.64	10
18	4.26	10.45	16.69	4	7.55	11
20	3.66	10.48	16.62	4	8.36	12
30	4.97	10.65	15.67	8	12.88	19
40	6.54	10.81	15.63	12	17.42	24
50	7.31	10.78	14.90	15	21.85	27
60	8.10	10.84	13.52	21	26.29	32
70	8.36	10.69	12.73	24	30.49	39
80	8.58	10.79	12.72	30	34.55	42
90	8.38	11.02	13.60	33	40.08	45
100	9.33	10.98	13.90	39	44.83	49

Table 1: Iterated 1-Steiner statistics: the performance figures denote percent improvement over MST cost. Also given are statistics regarding the number of Steiner points produced.

	Batched 1-Steiner												
	Performance			Number of SPs			Number of rounds			Ave #SPs per round			
$\|P\|$	Min	Ave	Max	Min	Ave	Max	Min	Ave	Max	1	2	3	4
4	0.00	8.83	27.19	0	2.10	4	0	0.97	2	1.14	0.37	0.00	0.00
5	0.00	9.35	23.59	0	2.59	5	0	1.08	3	1.51	0.46	0.06	0.00
6	0.00	9.83	25.99	0	3.09	6	0	1.15	3	1.91	1.42	0.03	0.00
7	0.41	9.82	23.44	2	3.52	7	1	1.17	3	2.32	0.59	0.02	0.00
8	0.00	9.92	20.93	0	3.99	8	0	1.23	3	2.73	0.96	0.17	0.00
9	1.04	10.07	20.62	2	4.48	7	1	1.31	3	3.12	1.16	0.19	0.00
10	1.26	10.24	20.03	3	4.95	8	1	1.34	4	3.56	1.11	0.10	0.01
11	2.01	10.32	20.81	3	5.42	10	1	1.33	3	4.01	1.01	0.03	0.00
12	1.85	10.32	18.83	2	5.88	10	1	1.43	4	4.31	1.28	0.09	0.00
13	2.06	10.27	19.74	3	6.39	11	1	1.46	4	4.80	1.23	0.09	0.04
14	3.01	10.28	19.56	3	6.82	11	1	1.48	4	5.18	1.25	0.14	0.07
15	2.78	10.41	18.10	4	7.27	12	1	1.49	4	5.59	1.33	0.12	0.00
16	2.47	10.42	18.45	4	7.76	13	1	1.52	4	6.01	1.33	0.13	0.00
17	3.00	10.58	17.94	5	8.33	14	1	1.56	3	6.50	1.38	0.27	0.00
18	3.62	10.46	18.19	5	8.70	14	1	1.58	4	6.85	1.52	0.26	0.01
19	3.71	10.37	18.05	5	9.14	14	1	1.61	5	7.21	1.58	0.10	0.01
20	4.35	10.39	17.90	5	9.63	15	1	1.60	4	7.68	1.58	1.00	0.00
30	6.49	10.64	14.73	9	13.47	20	1	1.85	4	11.61	1.75	0.21	0.01
40	6.28	10.68	14.55	12	18.10	26	1	1.98	5	15.95	2.47	0.23	0.00
50	7.25	10.72	15.28	15	23.70	33	1	2.04	4	19.97	2.78	0.56	0.00
60	8.67	10.77	14.29	19	27.53	37	1	2.12	5	23.88	3.81	0.58	0.00
70	7.80	10.82	13.55	23	32.26	42	1	2.17	4	28.02	3.52	0.62	0.00
80	7.34	10.83	12.88	28	36.99	50	1	2.20	4	32.07	4.66	0.64	0.01
90	7.99	10.84	13.52	32	41.61	56	1	2.24	4	36.18	5.14	1.17	0.04
100	8.78	10.86	13.60	35	46.30	59	1	2.25	4	40.21	5.55	1.31	0.01
110	8.04	10.86	12.86	40	50.98	67	1	2.28	4	44.96	5.77	0.53	0.08
120	7.83	10.86	12.62	44	55.59	68	1	2.29	5	48.64	7.15	1.02	0.00
130	8.25	10.85	12.63	47	60.03	73	1	2.34	4	52.87	7.22	0.82	0.00
140	8.45	10.84	12.57	51	64.97	79	1	2.38	5	56.94	8.12	1.07	0.22
150	9.07	10.92	12.63	57	69.85	87	1	2.39	4	59.92	8.28	1.13	0.00
200	9.39	10.97	12.32	79	93.10	107	2	2.47	4	80.04	11.68	1.48	0.50
250	10.26	10.98	11.68	103	116.32	131	2	2.57	4	100.00	14.53	0.82	0.00
300	9.76	10.97	12.18	131	137.67	145	2	2.50	4	121.85	16.85	1.33	0.00

Table 2: Batched 1-Steiner statistics: the performance figures denote percent improvement over MST cost. Also given are statistics regarding the number of Steiner points produced, the number of rounds, the the number of Steiner points induced per round.

	Iterated 1-Steiner Average Execution Speeds								
	Serial			Parallel			Speedup		overall
	old	new	ratio	old	new	ratio	old	new	gain
$\|P\|$	A	B	A/B	C	D	C/D	A/C	B/D	A/D
4	0.01	0.01	1.00	0.14	0.18	0.78	0.07	0.06	0.06
5	0.05	0.03	1.67	0.21	0.25	0.84	0.24	0.12	0.20
6	0.12	0.05	2.40	0.30	0.38	0.79	0.40	0.13	0.32
7	0.27	0.09	3.00	0.50	0.40	1.25	0.54	0.22	0.68
8	0.56	0.13	4.31	0.46	0.49	0.94	1.22	0.27	1.14
9	1.03	0.20	5.15	0.60	0.56	1.07	1.72	0.36	1.84
10	1.8	0.27	6.7	1.08	0.64	1.69	1.67	0.42	2.81
12	4.76	0.57	8.35	1.66	0.92	1.80	2.87	0.61	5.17
14	10.82	0.97	11.15	2.87	1.14	2.52	3.77	0.85	9.49
16	21.78	1.54	14.14	4.11	1.55	2.65	5.30	0.99	14.05
18	40.37	2.31	17.48	6.93	2.25	3.08	5.97	1.03	17.94
20	69.11	3.42	20.21	13.51	3.42	3.95	5.12	1.00	20.21
30	573	16.4	34.94	108	7.03	15.36	5.31	2.33	81.51
40	2147	47	45.68	348	19.5	17.85	6.17	2.41	110.10
50	9542	102	93.55	1166	42	27.76	8.18	2.43	227.19
60	27083	206	131.47	2949	59	49.98	9.18	3.49	459.03
70		405			94			4.31	
80		547			146			3.75	
90		886			221			4.01	
100		1283			277			4.63	

Table 3: Execution times for Iterated 1-Steiner: the serial execution times are given in CPU seconds, while the parallel execution times are elapsed wall-clock times. Both the serial and parallel versions were tested with the old MST routine (naive implementation) as well as the new MST routine (our implementation). The overall gain is the ratio of the old serial time to the new parallel time. The parallel implementation uses 9 SUN 4/40 (IPC) workstations, with a SUN 4/75 (Sparc2) as the master processor.

	Batched 1-Steiner Average Execution Speeds								
	Serial			Parallel			Speedup		overall
	old	new	ratio	old	new	ratio	old	new	gain
$\|P\|$	A	B	A/B	C	D	C/D	A/C	B/D	A/D
4	0.01	0.01	1.00	0.16	0.15	1.07	0.06	0.07	0.07
5	0.04	0.02	2.00	0.20	0.19	1.05	0.20	0.11	0.27
6	0.10	0.04	2.50	0.21	0.20	1.05	0.48	0.20	0.53
7	0.20	0.06	3.33	0.24	0.22	1.09	0.83	0.27	0.91
8	0.37	0.09	4.11	0.27	0.24	1.13	1.37	0.38	1.54
9	0.63	0.12	5.25	0.39	0.27	1.44	1.62	0.44	2.33
10	1.02	0.16	6.38	0.51	0.29	1.76	2.00	0.55	3.52
12	2.28	0.26	8.77	1.06	0.35	3.03	2.15	0.74	6.51
14	4.69	0.44	10.66	1.57	0.41	3.83	2.99	1.07	11.44
16	8.40	0.61	13.77	2.07	0.47	4.40	4.06	1.30	17.87
18	14.67	0.81	18.11	4.14	0.57	7.26	3.54	1.42	25.73
20	24.07	1.09	22.08	5.57	0.61	9.13	4.32	1.79	39.46
30	151	3.80	39.74	40.2	1.79	22.46	3.75	2.12	84.36
40	522	8.59	60.77	126	3.23	39.01	4.14	2.66	161.61
50	1130	16.1	70.19	200	5.03	39.76	5.65	3.20	224.65
60	2745	28.1	97.69	753	8.03	93.77	3.65	3.50	341.84
70	5520	41.9	131.74	1002	12.1	82.81	5.51	3.46	456.20
80	10350	62.2	166.40	2084	17.1	121.87	4.97	3.64	605.26
90	15450	87.5	176.57	2582	16.5	156.48	4.97	5.30	936.36
100	28140	114	246.84	4748	24.2	196.20	5.93	4.71	1162.81
120		232			40.5			5.73	
140		344			58.6			5.87	
160		376			79.67			4.72	
180		571			103			5.54	
200		801			129			6.21	
250		1528			212			7.21	
300		1800			447			4.03	

Table 4: Execution times for Batched 1-Steiner: the serial execution times are given in CPU seconds, while the parallel execution times are elapsed wall-clock times. Both the serial and parallel versions were tested with the old MST routine (naive implementation) as well as the new MST routine (our implementation). The overall gain is the ratio of the old serial time to the new parallel time. The parallel implementation uses 9 SUN 4/40 (IPC) workstations, with a SUN 4/75 (Sparc2) as the master processor.