A New Heuristic for Rectilinear Steiner Trees

Ion I. Măndoiu* Vijay V. Vazirani* Joseph L. Ganley†

1 Introduction

The Steiner tree problem is that of finding a minimum-length interconnection of a set of points in the plane, and has long been one of the fundamental problems in the field of electronic design automation. Although recent advances of integrated circuit technology into the deep-submicron realm have introduced additional routing objective functions, the Steiner tree problem retains its importance: For non-critical nets, or in physically small instances, minimum length is still frequently a good objective function, since a minimum-length interconnection has minimum overall capacitance and occupies a minimum amount of area. Furthermore, the development of good algorithms for the Steiner tree problem often lays a foundation for expanding these algorithms to accommodate objective functions other than purely minimizing length.

The rectilinear Steiner tree (RST) problem—in which the terminals are points in the plane and distances between them are measured in the L_1 metric—has been the most-examined variant in electronic design automation, since IC fabrication technology typically mandates the use of only horizontal and vertical interconnect. The RST problem is **NP**-hard [8], and much effort has been devoted to designing heuristic and approximation algorithms [1, 2, 5, 11, 13, 15, 16, 18, 24, 25, 26]. In an extensive survey of RST heuristics up to 1992 [14], the Batched Iterated 1-Steiner (BI1S) heuristic of Kahng and Robins [15] emerged as the clear winner with an average improvement over the MST on terminals of almost 11%. Subsequently, two other heuristics [2, 16] have been reported to match the same performance.

After a steady, but relatively slow progress [4, 6, 21], exact RST algorithms have recently witnessed spectacular progress [22]. The new release [23] of the GeoSteiner code by Warme, Winter, and Zachariasen has average running time comparable to the fast BI1S implementation of Robins [19] on random instances. We are thus faced with the paradoxical situation that an exact algorithm for an **NP**-hard problem has the same average running time as a state-of-the art heuristic for the problem.

We try to remedy this situation by proposing a new RST heuristic. Our experiments show that the new heuristic has better average running time than both Robins' implementation of BI1S and the GeoSteiner code. Moreover, the new heuristic gives higher-quality solutions than BI1S on the average; of course, it cannot beat GeoSteiner in solution quality.

Our results are obtained by exploiting a number of recent algorithmic and implementation ideas. On the algorithmic side, we build on the recent 3/2 approximation algorithm of Rajagopalan and Vazirani [17] for the metric Steiner tree problem on quasi-bipartite graphs; these are graphs that do not contain edges connecting pairs of Steiner vertices. This algorithm is based on the linear programming relaxation of a sophisticated integer formulation of the metric Steiner tree problem, called the bidirected cut formulation.

It is well known that the RST problem can be reduced to the metric Steiner tree problem on graphs [10]; however, the graphs obtained from the reduction are not quasi-bipartite. We give an RV-based heuristic for finding Steiner trees in arbitrary (non quasi-bipartite) metric graphs. The heuristic, called *Iterated RV* (IRV), computes a Steiner tree of a quasi-bipartite subgraph of the original graph using the RV algorithm, in order to select a set of candidate Steiner vertices. The process is repeated with the selected Steiner vertices treated as terminals—thereby allowing the algorithm to pick larger quasi-bipartite subgraphs, and seek additional Steiner vertices for inclusion in the tree—until no further improvement is possible.

The speed of our heuristic depends critically on the size of the quasi-bipartite subgraphs considered in each iteration. We reduce the size of the graphs that correspond to RST instances by applying *reductions*—deletion of edges and vertices that do not affect the quality of the result. Our key edge reduction is based on Robins and Salowe's result that bounds the maximum degree of a rectilinear MST [20], and allows us to retain in the graph at most 4 edges incident to each Steiner vertex. Notably, this observation also formed the basis of a significant speed-up in the running time of BI1S [9], and is currently used in the implementation [19]. Our vertex reduction is based on the *empty rectangle* test that has its roots in the work of Berman and Ramaiyer [1] (see also [5, 25]).

We ran experiments to compare our implementation of IRV against Robins' implementation of BI1S [19] and against the GeoSteiner code of Warme, Winter, and Zachariasen [23]. The results reported in Section 4 show that, on both random and real VLSI instances, our new heuristic produces on the average higher-quality solutions than BI1S. The quality improvement is not spectacular, around 0.03% from the cost of the MST on the average, but we should note that solutions produced by BI1S are themselves less than 0.5% away from optimum on the average—this leaves little space for improvement.

More importantly, IRV's improvement in solution quality is achieved with an excellent running time. Our IRV code runs 4–8 times faster than GeoSteiner, and 2–8 times faster than Robins' implementation of BI1S on random instances

*College of Computing, Georgia Institute of Technology, {mandoiu,vazirani}@cc.gatech.edu

†Cadence Design Systems, ganley@cadence.com

with up to 200 terminals—the speed-up increases with the number of terminals. On random instances, GeoSteiner has almost the same average running time as Robins' BI1S code, with a factor 2 advantage for BI1S on small instances.

After noticing that BI1S can also benefit from vertex reductions, we incorporated the empty rectangle test into Robins' code. The enhanced BI1S code becomes 60% faster than our IRV code on random instances. However, this does not necessarily mean that BI1S is the best heuristic in practice—results on real VLSI instances indicate a different hierarchy. On these instances IRV is faster than the enhanced BI1S, and GeoSteiner is also substantially faster than Robins' BI1S. It is often claimed [15] that random RST instances are statistically indistinguishable from real VLSI instances. Our results show that this claim is only partly true: While the relative solution quality does not change between experiments ran on random instances as compared to those ran on VLSI instances, the relative running time may change.

The two main contributions of this paper, the IRV algorithm and the speed up in the implementation of BI1S, produce two heuristics with significant speed advantage over the exact GeoSteiner code. Such advantage may be valuable in all VLSI stages in which the RST problem occurs—it is clearly valuable in applications where exact accuracy is not essential, e.g. in wirelength estimation during placement. Moreover, the two heuristics hold more promise than the GeoSteiner algorithm for giving efficient extensions to objective functions other than length minimization.

2 Steiner trees in graphs

The *metric Steiner tree in graphs* (GST) problem is: Given a graph $G = (V, E)$ whose vertices are partitioned in two sets, T and S, the *terminal* and *Steiner* vertices respectively, and non-negative edge costs satisfying triangle inequality, find a minimum cost tree spanning all terminals and any subset of the Steiner vertices. Recently, Rajagopalan and Vazirani [17] presented a 3/2 approximation algorithm for GST when restricted to quasi-bipartite graphs, i.e., graphs that have no edge connecting a pair of Steiner vertices. In this section we review the RV algorithm, discuss its implementation, and present an RV-based heuristic for the GST problem on arbitrary graphs.

2.1 The bidirected cut relaxation

The RV algorithm is based on a sophisticated integer program (IP) formulation of the GST problem. A related, but simpler formulation is given by the following observation: A set of edges $E' \subseteq E$ connects T if and only if every cut of G separating two terminals crosses at least one edge of E'. The IP formulation resulting from this observation is called the undirected cut formulation. The IP formulation on which the RV algorithm is based, called the *bidirected cut formulation*, is obtained by considering a directed version of the above cut condition.

Let $\vec{E}$ be the set of arcs obtained by replacing each undirected edge $(u, v) \in E$ by two directed arcs $u \to v$ and $v \to u$. For a set C of vertices, let $\delta(C)$ be the set of arcs $u \to v$ with $u \in C$ and $v \in V \setminus C$. Finally, if t_o is a fixed terminal, let $\mathcal{C}$ contain all sets $C \subseteq V$ that contain at least one terminal but do not contain t_o. The bidirected cut formulation is trying to pick a minimum cost collection of arcs from $\vec{E}$ in such a way that each valid set has at least one outgoing arc:

$$
\begin{aligned}
(1) \qquad \text{minimize} \quad & \sum_{e \in \vec{E}} \text{cost}(e) x_e \\
\text{subject to} \quad & \sum_{e:\, e \in \delta(C)} x_e \geq 1, \qquad C \in \mathcal{C} \\
& x_e \in \{0, 1\}, \qquad e \in E
\end{aligned}
$$

By allowing x_e's to assume non-negative fractional values we obtain a linear program (LP) called the bidirected cut *relaxation* of GST:

$$
\begin{aligned}
(2) \qquad \text{minimize} \quad & \sum_{e \in \vec{E}} \text{cost}(e) x_e \\
\text{subject to} \quad & \sum_{e:\, e \in \delta(C)} x_e \geq 1, \qquad C \in \mathcal{C} \\
& x_e \geq 0, \qquad e \in E
\end{aligned}
$$

The *dual* of the covering LP (2) is the packing LP:

$$
\begin{aligned}
(3) \qquad \text{maximize} \quad & \sum_{C \in \mathcal{C}} y_C \\
\text{subject to} \quad & \sum_{C:\, e \in \delta(C)} y_C \leq \text{cost}(e), \qquad e \in \vec{E} \\
& y_C \geq 0, \qquad C \subseteq V
\end{aligned}
$$

From LP-duality theory, the cost of a feasible solution to (3) is always less than or equal to the cost of any feasible solution to (2), and hence, less than or equal to the cost of any feasible solution to (1). The RV algorithm uses this observation to guarantee the quality of the solution produced: the algorithm constructs feasible solutions to both IP (1) and LP (3), in such a way that the the costs of the two solutions differ by at most a factor of 3/2.

2.2 The RV algorithm

The RV algorithm works on quasi-bipartite graphs G. At a coarse level, the RV algorithm is similar to the Batched Iterated 1-Steiner algorithm of Kahng and Robins [15]: both algorithms work in phases, and in each phase some Steiner vertices are iteratively added to the set of terminals. While BI1S adds Steiner vertices to T greedily—based on the decrease in the cost of the MST—the RV algorithm uses the bidirected cut relaxation to guide the addition.

In each phase, the RV algorithm constructs feasible solutions to both IP (1) and LP (3). The bidirected cut formulation and its relaxation are inherently asymmetric, since they require a terminal t_o to be singled out. However, the RV-Phase algorithm works in a symmetric manner—the information it computes can be used to derive feasible solutions for any choice of t_o.

A set $C \subseteq V$ is called *proper* if both C and $V \setminus C$ contain terminals; with respect to the original set of terminals only sets in $\mathcal{C}$ and their complements are proper. During its execution the RV-Phase algorithm tentatively converts some Steiner vertices into terminals—note that the only proper sets created by these conversions are the singleton sets containing the new terminals. The algorithm maintains a variable y_C, called *dual*, for every proper set, including the newly created ones. The *amount of dual felt* by arc e is $\sum_{C\,:\,e\in\delta(C)} y_C$; we say that e is *tight* when $\sum_{C\,:\,e\in\delta(C)} y_C = \text{cost}(e)$. A set C of vertices is *unsatisfied* if it is proper and $\delta(C)$ does not contain any tight arc.

The RV-Phase algorithm starts with y_C set to 0 for every proper set C, and an empty list $\vec{L}$ of tight arcs. It then proceeds in a *primal-dual* manner, by alternatively raising dual variables as long as this does not violate the packing constraints of (3), and picking tight edges into $\vec{L}$, thus satisfying more and more proper sets. When the algorithm stops, all proper sets are satisfied by tight arcs in $\vec{L}$:

The RV-Phase algorithm:

1. $\vec{L} \leftarrow \emptyset$; For each proper set C, $y_C \leftarrow 0$.
2. While there exist unsatisfied sets do:

   Uniformly rise the y values of minimally unsatisfied sets until an arc $u \rightarrow v$ goes tight.

   If $u \notin T$, then $T \leftarrow T \cup \{u\}$; go to Step 1.

   Else, $\vec{L} \leftarrow \vec{L} \cup \{u \rightarrow v\}$.

Theorem 1 *[17] (a) If arc $u \rightarrow v$, $u \notin T$, goes tight then* $\text{cost}(\text{MST}(T \cup \{u\})) < \text{cost}(\text{MST}(T))$.

(b) At the end of the RV-Phase *algorithm,* $\text{MST}(T)$ *cannot be improved by adding Steiner vertices.*

The RV algorithm (whose pseudocode we omit) repeats the RV-Phase algorithm followed by removal of unnecessary Steiner vertices, until no further improvement is made in the cost of $\text{MST}(T)$. At the end of the algorithm, the duals raised around proper sets are converted into a solution to (1) by picking t_o and discarding y_S's with $S \notin \mathcal{C}$. The 3/2 approximation guarantee follows by relating the cost of this solution to the cost of $\text{MST}(T)$.

2.3 Efficient implementation of the RV-Phase algorithm

Since our heuristic on general graphs uses RV-Phase as a subroutine, we describe here an efficient implementation of it. Several implementation ideas are derived from the following key property maintained throughout the RV-Phase algorithm:

Lemma 2 *[17] Let u and v be two terminals. If all arcs along some path $u \rightarrow x_1 \rightarrow \cdots \rightarrow x_k \rightarrow v$ are tight, then so are the arcs on the reverse path, $v \rightarrow x_k \rightarrow \cdots \rightarrow x_1 \rightarrow u$.*

For implementation purposes we do not need to keep track of the duals raised—all that matters is the order in which arcs get tight. The tightening time of an arc can be determined by monitoring the number of minimally unsatisfied sets (henceforth called *active* sets) that are felt by that arc. It is easy to see that the set of vertices reachable via tight arcs from a terminal u always form an active set; Lemma 2 implies that no other active set can contain u. Thus, we get:

Corollary 3 *For any terminal u, there is exactly one active set containing u at any time during the algorithm. Hence, the tightening time of any arc $u \rightarrow v$ is exactly* $\text{cost}(u, v)$.

Unlike terminals, Steiner vertices may be contained in multiple active sets. Hence, arcs out of Steiner vertices will feel dual at varying rates during the algorithm.

Lemma 4 *Let u be a Steiner vertex. If arc $u \rightarrow v$ is the first arc out of u to go tight, then arc $v \rightarrow u$ goes tight at the same time or before $u \rightarrow v$ is. Moreover, each arc $u \rightarrow w$ for which $w \rightarrow u$ is already tight will go tight when $u \rightarrow v$ goes tight.*

Proof: In order to get tight, $u \rightarrow v$ must feel some active set, i.e., there must exist a tight path from a terminal $v' \neq v$ to u. After $u \rightarrow v$ gets tight, there is a tight path from v' to v, and by Lemma 2 the reverse path (hence the arc $v \rightarrow u$) must also be tight. The second claim follows similarly. □

Since several arcs out of a Steiner vertex get tight simultaneously, we say that a Steiner vertex *crystallizes*. Note that crystallization is precisely the moment when the vertex begins to be treated as terminal. Lemma 4 implies that, in order to detect when a Steiner vertex crystallizes, it suffices to monitor the amount of dual felt for the shortest arc out that Steiner vertex.

Our implementation maintains a list of active sets; initially containing a singleton set for each terminal. We also maintain the amount of dual felt by the shortest arc out of each Steiner vertex, initially 0. Arcs out of terminals are sorted in non-decreasing order, then marked as tightened in this order. As new arcs are tightened, we update the list of active sets and the amount of dual felt by the shortest arcs out of Steiner vertices, crystallizing Steiner vertices as needed. The maintenance of the list of active sets gives a worst case running time is $O(k \cdot |T| \cdot |S|)$, where k is the number of crystallized Steiner vertices—all other operations can be easily implemented in $O(k \cdot |E| \cdot \log |V|)$.

2.4 The heuristic for general graphs

A simple way of dealing with non-quasi-bipartite graphs is to remove all Steiner-Steiner edges and then run the RV algorithm. To allow Steiner-Steiner edges to come into play, we iterate this process. If a Steiner vertex is added to T by some run of the RV algorithm, for subsequent runs we

extend the graph by adding *all* edges incident to it, not just those leading to terminals.

Our experiments have shown that it is better—in both running time and solution quality—to extend the graph after running just one RV-Phase, not the full RV algorithm, on the quasi-bipartite graph. This gives the following algorithm:

The IRV Algorithm:

1. $T_1 \leftarrow T_o \leftarrow T$
2. Remove from G all edges (u, v) with $u \notin T$, $v \notin T$, and run the RV-Phase algorithm on the resulting graph; this will add some Steiner vertices to T.
3. Construct an MST on T, then prune from $T \setminus T_o$ all vertices with tree degree ≤ 2.
4. If $\text{cost}(\text{MST}(T)) < \text{cost}(\text{MST}(T_1))$, then $T_1 \leftarrow T$; go to Step 2.
5. Return $\text{MST}(T_1)$.

3 Rectilinear Steiner trees

The *rectilinear Steiner tree* (RST) problem is defined as follows: Given a set T of *terminals* in the Cartesian plane, find a shortest interconnection of the terminals using only horizontal and vertical lines. Lines are allowed to meet at points other than the terminals; non-terminal meeting points are called *Steiner points*.

By a classical result of Hanan [10], there exists an optimum rectilinear Steiner tree that uses only Steiner points located at intersections of vertical and horizontal lines passing through terminals. Thus, finding a minimum rectilinear Steiner tree on a set of terminals reduces to finding a minimum Steiner tree in the Hanan grid, with edge costs given by the L_1 (or Manhattan) metric, $d(u, v) = |x_u - x_v| + |y_u - y_v|$.

The IRV algorithm yields good results when applied to a graph for which the cost and structure of the minimum Steiner tree does not change much after the removal of Steiner-Steiner edges. For the RST problem, the best choice w.r.t. solution quality is to run IRV on the *complete* graph induced by the Hanan grid. We obtain a practical running time by applying a few simple, yet very effective reductions to this graph.

3.1 Edge reduction

By a result of Robins and Salowe [20], for any set of points there exists a rectilinear MST in which each point p has at most one neighbor in each of the four diagonal quadrants $-x \leq y < x$, $-y < x \leq y$, $x < y \leq -x$, and $y \leq x < -y$ translated at p. Hence, the optimum Steiner tree in the quasi-bipartite graph is not affected if we remove all edges incident to a Steiner vertex except those connecting it to the closest terminals from each quadrant. We can also discard all edges connecting pairs of terminals except for the $|T| - 1$ edges in $\text{MST}(T)$—this merely amounts to a particular choice of breaking ties between edges during RV-Phase. Combined, these two edge reductions leave a quasi-bipartite graph with $O(|T|+|S|)$ edges, as opposed to $O(|T|\cdot(|T|+|S|))$ without edge reductions.

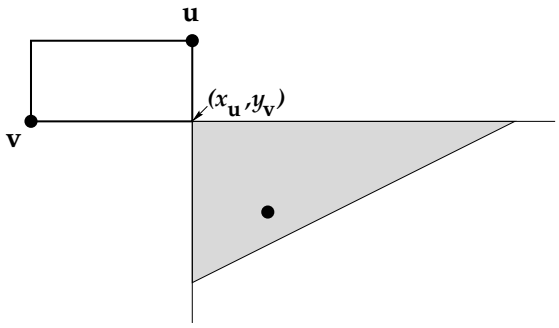

Figure 1: The empty rectangle test.

3.2 Vertex reduction

As noted by Zachariasen [25], the Full Steiner Tree reductions, which play a crucial role for exact algorithms such as [4, 22], can also be used to remove from the Hanan grid a large number of Steiner vertices without affecting the optimum Steiner tree. Simpler versions of the tests suffice in our case, since we only want to leave unaffected the optimum Steiner tree in the graph that results after the removal of Steiner-Steiner edges.

We incorporated in our code the *empty rectangle* test [25], originally due to Berman and Ramaiyer [1]. For the configuration in Figure 1, the test says that the grid point (x_u, y_v) can be safely omitted unless the rectangle determined by terminals u and v is empty (i.e., contains no terminals in its interior) and the shaded quadrant contains at least one terminal. We used a simple $O(|T|^2)$ implementation of this test; an $O(n \log n + k)$ implementation, where k is the number of empty rectangles, is also possible [7].

In fact, the above test can be strengthened [5, 25] so that it removes all but a set of $O(|T|)$ Steiner points, still with no increase in the cost of the optimum RST with no Steiner-Steiner edges. Using this stronger test, the overall running time of IRV as applied to RST can be reduced to $O(k \cdot |T|^2)$, where k is the number of crystallized Steiner vertices (usually a small fraction of $|T|$).

4 Experimental results

We compared our algorithms against Robins' implementation [19] of BI1S [9, 15], and against the recent release [23] of the GeoSteiner algorithm of Warme, Winter, and Zachariasen [22].

All tests were conducted on a SGI Power Challenge machine with 16 195 MHz IP27 processors (only one of which is used in our sequential implementation) and 4 G-Bytes of internal memory, running under IRIX Release 6.4 IP27. We coded our heuristics in C, and used Robins' publicly available BI1S C code. We compiled both programs using the gcc compiler (version egcs-2.90.27, with `-O4 -inline-functions` optimization flags). The timing was

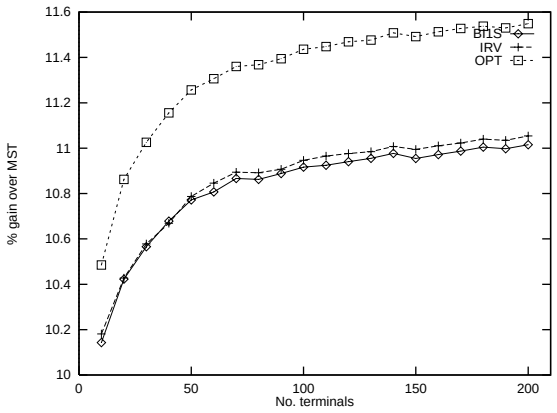

Figure 2: Average improvement over MST.

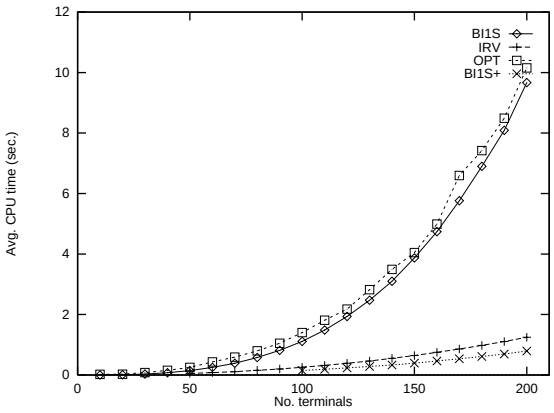

Figure 3: Average CPU time.

performed using low level Unix interval timers, under similar load conditions.

The test bed for our experiments consisted of two categories of instances:

Random instances: For each instance size between 10 and 200, in increments of 10, we generated uniformly at random 1000 instances consisting of points in general position[1] drawn from a 10000×10000 grid.

Real VLSI instances: To further validate our results, we ran all heuristics on a set of 9 instances extracted from two different VLSI designs.

Following the standard practice [14], we use the *percent improvement over the MST on terminals*,

$$\frac{\text{cost(MST)} - \text{cost(Heuristic)}}{\text{cost(MST)}} \times 100,$$

to compare the relative performance of the three algorithms.

Figure 2 shows the average improvement over MST for BI1S, IRV, and GeoSteiner on random instances. The average running times on random instances are plotted in Figure 3, we include in this comparison the version of BI1S enhanced by the inclusion of the empty rectangle test (sf BI1S+). Statistics on the 9 VLSI instances are presented in Table 1.

5 Conclusions

The experimental data presented in the previous section shows that IRV produces high-quality rectilinear Steiner trees, typically better than those produced by the Batched Iterated 1-Steiner heuristic. The same data shows that BI1S is significantly sped up by the addition of the empty rectangle test. With this enhancement, BI1S runs faster than IRV on random instances, but not on large real VLSI instances as those considered in our tests. It should be interesting to perform more extensive tests on real VLSI instances to see how the relative running time of the two heuristics is affected by the large fraction of small nets present in VLSI designs.

Our experimental data also confirms the excelent performance of the exact algorithm of Warme, Winter, and Zachariasen [22]. When exact algorithms achieve practical running times, one is immediately prompted to ask if any interest remains for sub-optimal heuristics. We think that this interest will not dissapear, definitely not in RST applications where speed is more important than exact solutions, e.g., in wirelength estimation during placement [3]. Moreover, heuristics such as IRV and BI1S hold more promise than the GeoSteiner algorithm for giving efficient extensions to objective functions other than length minimization.

[1] A set of points is in general position if no two points share a common x- or y-coordinate.

References

[1] Piotr Berman and Viswanathan Ramaiyer. Improved approximations for the Steiner tree problem, *Journal of Algorithms*, 17 (1994), pp. 381–408.

[2] M. Borah, R.M. Owens, and M.J. Irwin. An edge-based heuristic for Steiner routing, *IEEE Trans. on CAD* 13 (1994), pp. 1563–1568.

[3] A.E. Caldwell, A.B. Kahng, S. Mantik, I.L. Markov, and A. Zelikovsky. On wirelength estimations for row-based placement, ISPD 1998, pp. 4–11.

[4] U. Fössmeier and M. Kaufmann. Solving rectilinear Steiner tree problems exactly in theory and practice, Proc. 5th European Symposium on Algorithms (1997), Springer-Verlag LNCS 1284, pp. 171–185.

[5] U. Fössmeier, M. Kaufmann, and A. Zelikovsky. Faster approximation algorithms for the rectilinear Steiner tree problem. *Discrete and Computational Geometry* 18 (1997), pp. 93–109.

[6] J.L. Ganley and J.P. Cohoon. A faster dynamic programming algorithm for exact rectilinear Steiner minimal trees, *Fourth Great Lakes Symp. on VLSI* (1994), pp. 238–241.

Design/Net	No. term.	Average improvement			CPU seconds			
		BI1S	IRV	GeoSteiner	BI1S	BI1S+	IRV	GeoSteiner
16BSHREG.CLK	185	1.757	1.757	1.757	5.17	1.31	0.25	2.80
16BSHREG.RESET	406	3.666	3.666	3.810	52.23	10.07	1.65	4.37
16BSHREG.VDD	573	8.079	8.079	8.118	165.47	30.29	2.94	1.73
16BSHREG.VSS	556	7.854	8.131	8.192	155.15	36.71	3.29	7.90
MAR.BRANCH	188	9.007	9.158	9.221	7.73	1.26	0.62	5.21
MAR.CLK	264	7.637	7.748	7.957	16.53	2.34	1.57	13.16
MAR.GND	245	6.300	6.321	6.476	13.22	1.96	1.26	1.03
MAR.RESET	109	11.206	11.246	11.246	1.22	0.24	0.16	0.65
MAR.VDD	340	6.038	6.003	6.181	46.75	7.69	1.59	8.19

Table 1: Gain over MST and running time for VLSI instances.

[7] R.-H, Güting, O. Nurmi, and T. Ottmann. Fast algorithms for direct enclosures and direct dominances, *Journal of Algorithms* 10 (1989), pp. 170–186.

[8] M.R. Garey and D.S. Johnson, The rectilinear Steiner tree problem is NP-complete, *SIAM J. Appl. Math.*, 32 (1977), pp. 826–834.

[9] J. Griffith, G. Robins, J.S. Salowe, and T. Zhang. Closing the gap: near-optimal Steiner trees in polynomial time, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13 (1994), pp. 1351–1365.

[10] M. Hanan. On Steiner's problem with rectilinear distance, *SIAM J. Appl. Math.*, 14 (1966), pp. 255–265.

[11] J.-M. Ho, G. Vijayan, and C.K. Wong. New algorithms for the rectilinear Steiner tree problem, *IEEE Trans. on CAD*, 9 (1990), pp. 185–193.

[12] F.K. Hwang. On Steiner minimal trees with rectilinear distance, *SIAM J. Appl. Math.*, 30 (1976), pp. 104–114.

[13] F.K. Hwang. An $O(n \log n)$ algorithm for suboptimal rectilinear Steiner trees, *IEEE Trans. on Circuits and Systems*, 26 (1979), pp. 75-77.

[14] F.K. Hwang, D.S. Richards, and P. Winter. *The Steiner tree problem*, Annals of Discrete Mathematics 53, North-Holland, Amsterdam, 1992.

[15] A.B. Kahng and G. Robins. A new class of iterative Steiner tree heuristics with good performance, *IEEE Trans. on CAD*, 11 (1992), pp. 1462–1465.

[16] F.D. Lewis, W.C.-C. Pong, and N. Van Cleave. Local improvement in Steiner trees, *Proceedings of the Third Great Lakes Symposium on VLSI*, 1993, pp. 105–106.

[17] S. Rajagopalan and V.V. Vazirani. On the bidirected cut relaxation for the metric Steiner tree problem, *10th ACM-SIAM Symposium on Discrete Algorithms*, 1999, pp. 742–751.

[18] D.S. Richards. Fast heuristic algorithms for rectilinear Steiner trees, *Algorithmica*, 4 (1989), pp. 191–207.

[19] Gabriel Robins. Steiner code available at `www.cs.virginia.edu/ robins/steiner.tar`

[20] G. Robins and J.S. Salowe. On the maximum degree of minimum spanning trees. *Proc. of the ACM Symp. on Computational Geometry*, 1994, pp. 250-258.

[21] J.S. Salowe and M.D. Warme. Thirty-five-point rectilinear Steiner minimal trees in a day, *Networks* 25 (1995), pp. 69–87.

[22] D.M. Warme, P. Winter, and M. Zacharisen. Exact Algorithms for Plane Steiner Tree Problems: A Computational Study. Technical Report DIKU-TR-98/11, Dept. of Computer Science, University of Copenhagen, 1998.

[23] D.M. Warme, P. Winter, and M. Zacharisen. The GeoSteiner 3.0 package, available via ftp from `ftp.diku.dk/diku/users/martinz/geosteiner-3.0.tar.gz`

[24] Y.F. Wu, P. Widmayer, and C.K. Wong. A faster approximation algorithm for the Steiner problem in graphs, *Algorithmica* 23 (1986), pp. 223–229.

[25] M. Zachariasen. Rectilinear Full Steiner Tree Generation. Networks 33 (1999), pp. 125–143.

[26] A. Zelikovsky. An 11/6-approximation algorithm for the network Steiner problem, *Algorithmica*, 9 (1993), pp 463–470.