A Data Structure for Fast Near-Optimal Rectilinear Steiner Tree Generation

Nancy Van Cleave
Eastern Illinois University

Abstract. *We present a simple and efficient data structure to represent a useful class of Rectilinear Steiner Trees, as well as the near-optimal results obtained when this structure is used with a slightly modified version of Saab and Rao's adaptive heuristic algorithm, Stochastic Evolution. Empirical results for up to 100 points give an average improvement of more than 11% over Rectilinear Minimal Spanning Trees -- an improvement unequaled by other algorithms.*

1. Introduction

Rectilinear Steiner trees have a variety of applications which make them attractive for study. Especially of interest is their use in the global routing phase of VLSI chip design, as well as in routing utilities along streets and in buildings. Many heuristics, and a few optimal algorithms for small sets of about 20 points, have been suggested. Many of these algorithms are based on minimal spanning trees or geometric methods. [See for example HVW90, Hw79, KaRo90, KoSh85, LPV92, Ri89, and YaWi72].

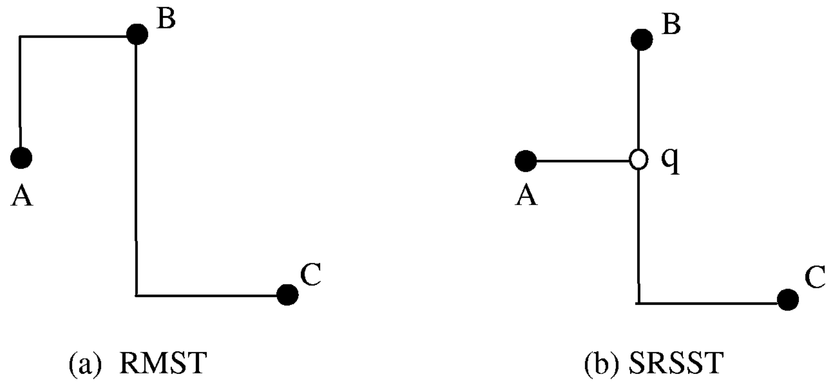


Figure 1. Rectilinear Spanning Trees

Shortest rectilinear Steiner spanning trees (SRSST) are similar to rectilinear minimal spanning trees (RMST), except we may add extra vertices to shorten the Steiner tree. These extra points, called Steiner points, are used to eliminate overlaps which can occur in minimal spanning trees. See figure 1. Let us formally define the problem as follows:

> **Definition**. *The rectilinear Steiner problem is that of connecting a set of points in the plane with a connected collection of vertical and horizontal lines with minimal overall length.*

F. K. Hwang [Hw76] proved the following relation between SRSSTs and RMSTs. It is used often to guarantee the performance of algorithms based on minimal spanning trees.

Theorem 1. (Hwang) *A minimal spanning tree over a set of points is no longer than 1.5 times the length of the shortest rectilinear Steiner spanning tree over the same set of points.*

We wish next to consider the class of Steiner trees which have at most one line per point and introduce a simple representation based on the intersection points of the lines in these trees. Then we shall describe how this representation may be used in an implementation of the Stochastic Evolution algorithm [SaRa91] for the Steiner problem, and finally, discuss our results.

2. A Special Class of Steiner Trees

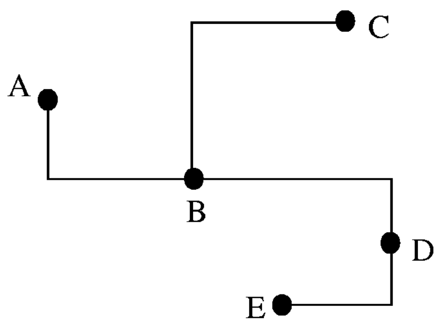
We can use the following theorem [JLV96] to prune our search space of a multitude of possible Steiner trees without deleting every optimal tree.

Theorem 2. *There exists a rectilinear shortest Steiner spanning tree over a set of n points composed of at most n lines.*

By imposing the restriction the *n* points must have unique coordinates (i.e., no two points in the set may lie on the same horizontal or vertical line), we can concentrate solely on trees which have exactly n lines, one associated with each point and connecting it to the rest of the Steiner tree. We are able to force any arbitrary set of points to meet this restriction by perturbing the coordinates by some epsilon in order to obtain *n* unique coordinates.

3. A Steiner Tree Representation

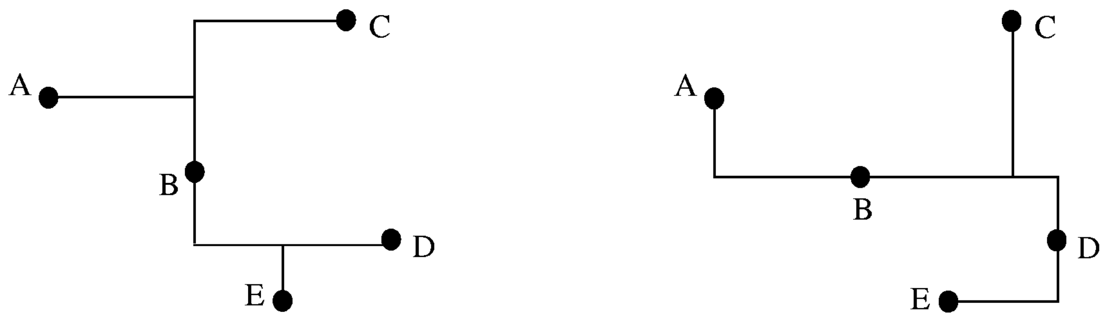
Often, minimal spanning trees are represented by a list of point pairs giving the direct connections between points. An example of this is shown in figure 2. These pairs merely indicate that *some* connection is to be made between the points, but do not tell us *how* the connection is to be made. Since shortest paths in the rectilinear metric are not unique, these pairs do not specify a particular tree, but a set of many trees over the points.



< A, B >
< B, C >
< B, D >
< D, E >

Figure 2. Graphical and Connection List Representations of an MST

We note if we process such a tree in a depth-first manner, assigning a direction to the single line through each point, we find these connection pairs may also be used to denote the intersections of the lines in the Steiner tree. See figure 3. By requiring the pairs to be ordered, arbitrarily choosing the horizontal component as the first in the ordered pair, we derive a representation which uniquely defines each Steiner spanning tree unambiguously.



< A, B >	< B, A >
< C, B >	< B, C >
< D, B >	< B, D >
< D, E >	< E. D >

Figure 3. Graphical and Intersection Pair Representations of Steiner trees over the same set of points.

Exactly two such representations are derivable (in linear time) from any RMST connection pairs list [Va92], and either of them may be used as the initial feasible solution required by the Stochastic Evolution algorithm.

4. The Stochastic Evolution Algorithm

Simulated Annealing (SA) and Stochastic Evolution (SE) algorithms are classified as combinatorial optimization techniques. Both are general purpose, but while SA is tunable to a particular problem, SE is an adaptive heuristic which rewards itself for finding better solutions. SA is modeled after the physical annealing process for solids where the material to be annealed is heated to a sufficiently high temperature (until the particles are arranged randomly), then slowly cooled to a ground state. The general SE algorithm is very similar to SA in that they both iteratively generate a new state from the current state. However, SA moves from one configuration to a neighboring configuration based on a simple move, eventually settling on a solution. On the other hand, the SE algorithm keeps track of the best solution found during the iterations, while moving from one configuration to the next which is derived by a compound move, but which is not necessarily an immediate neighbor of the previous configuration.

The SteinerSE algorithm is given in figure 4. The procedure requires an initial feasible configuration, C, as well as values for the control parameters InitialMaxOver and MaxCount.

During each iteration of the loop, if a configuration is found with a shorter length than Best, Count is decremented by MaxCount, thereby increasing the number of iterations. Hence, as long as improvements are made, the algorithm rewards itself with more time to search for even better configurations.

```
SteinerSe (Config, Best, InitialMaxOVer, MaxCount)
PRE:  C                      : an initial feasible configuration
                                    given as intersection pairs
      InitialMaxOver         : initial maximum negative gain
      MaxCount               : approximate number of iterations
                                    before improvement if found
POST: Best                   : configuration of least cost found
                                    by the algorithm

// Initializations
Best = C
Count = 0
MaxOver = InitialMaxOver

// Iteratively search for better solution
REPEAT
    // Create new solution
    PreCost = Length (C)
    Perturb (C, MaxOver)
    PostCost = Length (C)
    Update (MaxOver, PreCost, PostCost, InitialMaxOver)
    if PostCost < Length (Best) then
        // if improved, save best and reward counter
        Best = C
        Count = Count - MaxCount
    else
        // increment counter
        Count = Count + 1
UNTIL Count > MaxCount
```

Figure 4. General Stochastic Evolution Algorithm

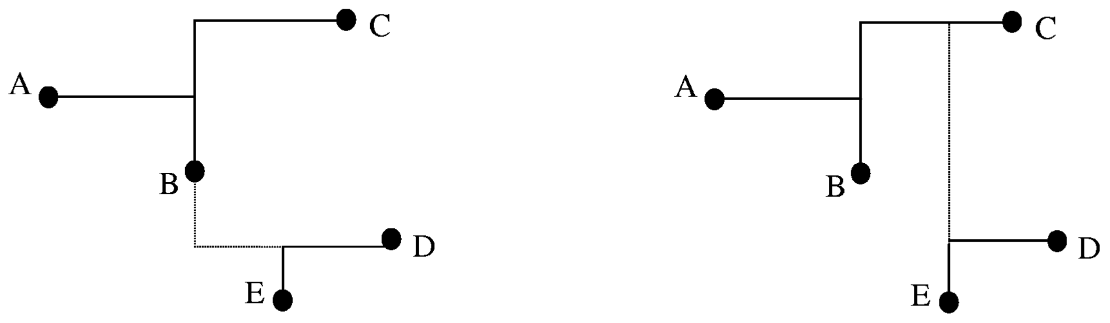
This continues until Count finally exceeds MaxCount. Thus MaxCount, the iteration bound, represents the fewest iterations needed before we encounter an improvement over the best configuration found so far. If MaxCount is chosen too small, the algorithm may not have time to find even a local minimum. If MaxCount is too large, time will be wasted looking for better configurations which do not exist.

InitialMaxOver is the initial maximum increase in length allowed while still accepting (moving to) a new configuration. As we iterate, if we do not find a better configuration we increase the value of MaxOver, allowing even greater increases in length until we finally locate a better solution or we exceed MaxCount. Once an improvement over Best is found, MaxOver is reset to InitialMaxOver in the procedure Update.

5. Perturbing Configurations

We defind a configuration to be the intersection pairs discussed in Section 3, and must consider how to generate a neighboring configuration. We could randomly choose an intersection pair to be replaced, but we must keep in mind removing such a pair from the Steiner tree causes the tree to be disconnected. This creates two subtrees, one of which may contain a single point in the case where we disconnect a leaf.

Hence, we may not replace pairs haphazardly, but must choose a replacement pair which reconnects the subtrees. To accomplish this, we must determine which points (and their associated lines) are in each of the subtrees. We may then randomly pick a horizontal line from one group and a vertical line from the other to reconnect the trees, using their intersection point to replace the pair chosen for deletion.



< A, B >
< C, B >
< D, B >*
< D, E >

< A, B >
< C, B >
< C, E >*
< D, E >

Figure 5. Steiner tree configuration and a random neighbor.

For example, suppose we have the tree and intersection pairs shown on the left in figure 5, and we randomly choose to replace the third intersection pair, < D, B >. This splits the tree into two subtrees, the first composed of the lines associated with A, B, and C; the second with D and E. Hence, the first group of points has two horizontal lines (one each associated with A and C), and one vertical line (with B). The second group has one horizontal (with D) and one vertical line (with E).

Now, suppose we randomly choose a horizontal line from the first group, say the line associated with C. We must then choose to connect C with E as it is the only vertical line in the second group. Thus we have the new configuration shown on the right in figure 5. We see this change in the configuration did not yield a very good tree, although it *is* feasible.

In Stochastic Evolution we use a compound move in an attempt to find an even better solution than our current best so far. Here we step through the ordered intersection pairs of the configuration and create replacement pairs for each one, randomly choosing horizontal and vertical lines from the two subtrees. At each step, the resulting configuration is evaluated and if the tree is shortened, or the change in tree length is less than a randomly generated number between one and the maximum allowable negative gain, that move is *accepted*. Otherwise, the resulting configuration is ignored and we continue to the next pair, still using the current configuration. Of course subsequent configurations will depend on earlier accepted moves since the configuration will change during the procedure and we wish to generate a feasible solution.

Since our algorithm keeps track of the best configuration found so far, it can do no worse than the original configuration: the minimal spanning tree itself. Hence we have the following theorem which is an extension of Theorem 1.

> **Theorem 3.** *The SteinerSE Algorithm produces a tree which is no longer than 1.5 times optimal.*

6. Empirical Results

Two variations of the SteinerSE algorithm were tested in Randhawa [Ra95] on data sets generated randomly from a uniform distribution on a fixed size grid. Kahng and Robins [KaRo90] determined such instances are statistically indistinguishable from the pin locations of actual VLSI layouts, and are in fact the standard testbed for Steiner tree heuristics [Ri89]. The results of these tests are shown in Table 1 below.

Table 1. SteinerSE Statistics
(Execution time is in seconds)

Set size	MST cost	Time	Ave. Percent Improvement over MST
10	50	1.5	10.8
15	70	6	9.9
20	125	21	11.1
25	125	30	10.9
30	120	39	12.3
35	120	57	10.9
50	70	91	11.7
100	40	550	11.1
		Overall Average:	11.1

In the case of 5, 7, and 10 point data sets [Va92], SteinerSE found all but one of the known optimal solutions and was off by less than 1% of the tree length in that case.

Table 2 gives a comparison of reported results from the original papers when available. Of course the other algorithms are presumed to have been executed on the general case, not point sets with unique coordinates.

Table 2. Comparison of Several Reported Results

Researchers	method	% improved
Lee et al	Prim	9
Hwang	Prim	9
Bern et al	Kruskal	9
Ricahrds (Hanan)	line-sweep	4
Lewis et al	line-sweep	8.4
Smith et al	geometric	8
Ho et al	edge-flip	9
Kahng et al	1-Steiner	10.9
SteinerSE	comb. opt.	11.1

We see SteinerSE obtains the best average improvement. Hence we have an algorithm which yields excellent results with the benefits of being simple to understand and implement.

7. References

BeCa85 Bern, M., and M. de Carvalho. “A Greedy Heuristic for the Rectilinear Steiner Tree Problem. *Technical Report*, UC Berkeley, CS Department (1985).

HVW90 Ho, J.-M., G. Vijayan, and C. K. Wong. “New Algorithms for the Rectilinear Steiner Tree Problem.” *IEEE Transactions on Compute Aided Design*, **9:2** (1990), 185-193.

Hw76 Hwang, F. K. “On Steiner Minimal Trees with Rectilinear Distance.” *SIAM Journal of Applied Math.*, **30** (1976), 104-114.

Hw79 Hwang, F. K. “An O(nlogn) Algorithm for Sub-optimal Rectilinear Steiner Trees.” *IEEE Transactions on Circuits and Systems*, **CAS-26:1** (1079), 75-77.

JLV96 Joyce, P., F. D. Lewis, and N. Van Cleave. “The Feasible Solution Space for Steiner Trees.” *Eighth SIAM Conference on Discrete Mathematics*, June 1996.

KaRo90 Kahng, A., and G. Robins. "On a New Class of Iterative Steiner Tree Heuristics with Good Performance." *Technical Report*, UCLA CS Dept., **#CSD-900014** (May 1990)

KoSh85 Komlos, J., and M. Shing. "Probabilistic Partitioning Algorithms for the Rectilinear Steiner Problem." *Networks*, **15** (1985), 413-423.

LBH76 Lee, J., N. Bose, and F. K. Hwang. "Use of Steiner's Problem in Suboptimal Routing in Rectilinear Metric." *IEEE Transactions on Circuits and Systems*, **CAS23:7** (July 1976), 470-476.

LPV91 Lewis, F. D., W. C. Pong, and N. Van Cleave. "A Linear-time Heuristic for Rectilinear Steiner Trees." *First Great Lakes Symposium on VLSI*, 1991.

LPV92 Lewis, F. D., W. C. Pong, and N. Van Cleave. "Optimum Steiner Tree Generation." *Second Great Lakes Symposium on VLSI*, 1992.

Ra95 Randhawa, R. S. "Random Methods for Rectilinear Steiner Tree Problem with Applications in VLSI Design," *Master's Thesis*, CS Dept., TTU, Lubbock (1995).

Ri89 Richards, D. "Fast Heuristic Algorithms for Rectilinear Steiner Trees." *Algorithmica* **4** (1989), 191-207.

SaRa91 Saab, Y. and V. Rao. "Combinatorial Optimization by Stochastic Evolution." *IEEE Transactions on Computer Aided Design*, **10:4** (1991), 525-535.

SLL80 Smith, M., D. Lee, and J. Lievman. "An O(*nlogn*) Heuristic Algorithm for the Rectilinear Steiner Minimal Tree Problem." *Eng. Optimization* **4** (1980), 179-192.

Va92 Van Cleave, N. "The Rectilinear Steiner Problem" *Ph.D. Dissertation*, CS Dept., University of Kentucky, Lexington (1992).

YaWi72 Yang, Y. Y., and O. Wing. "Optimal and Suboptimal Solution Algorithms for the Wiring Problem." *Proceedings of the IEEE International Symposium on Circuit Theory* (1972), 154-158.