Optimum Steiner Tree Generation

F. D. Lewis, Wang Chia-Chi Pong, and N. Van Cleave

Department of Computer Science
University of Kentucky
Lexington, Kentucky 40506

Abstract

Several phases of the VLSI design process use rectilinear Steiner spanning trees in estimating wire length. Since the problem is $\mathcal{NP}$-complete heuristics form the major portion of the collection of algorithms for this problem. Exact solutions are rare and very few have even been implemented. Thus they seem not to be practical. We first reduce the feasible solution space so that exact solutions are possible. Then we develop two branch and bound algorithms which achieve exact solutions. Distributing the computation between processors and parallel computation methods are currently being tested in an attempt to extend the size of the problems which can be actually solved.

1. Introduction.

In circuit design, timing considerations often require finding wire routing paths of minimal length between groups of terminals while performing placement and global routing. Thus the problem of connecting a set of n points in the plane with a minimum length collection of vertical and horizontal wires becomes important. This is the *rectilinear Steiner problem.*

Since it is $\mathcal{NP}$-complete [GJ77], this problem is usually attacked with the aid of heuristic algorithms. (Two excellent surveys of such heuristics appear in the papers by Winter [Wi87] and Richards [Ri89].) And, due to its massive descriptive complexity, the Steiner problem seems almost immune to practical exact solution. In fact, the only attempts at this have been an interesting dynamic programming approach [Be90] and a heroic branch and bound attempt from long ago [YW72]. Only the latter was implemented, and only small problems were attempted in that study. Thus neither seems practical, even on the computing machinery of today.

In order to have a chance of producing exact solutions for even modest data sets something must be done about the size of the feasible solution space for the problem. We reduce it from $O(2^{2n(n-1)})$ to $O(2^{n(2\log n-1)}/n^4)$. Then we develop two branch and bound methods which take advantage of this to achieve optimum solutions to the rectilinear Steiner problem. This allows us to achieve solutions - especially when distributing the computational tasks over several processors.

2. Preliminaries.

A *shortest rectilinear Steiner spanning tree* over a set of points is a connected collection of vertical and horizontal lines of minimum total length which spans the points. In the interest of brevity we shall often refer to them as *Steiner trees* in the remainder of the paper.

These Steiner spanning trees are closely related to ordinary spanning trees. In fact, a result due to Hwang [HW76] places them in context with minimum spanning trees by showing that shortest Steiner spanning trees are no smaller in length than two-thirds of minimum spanning tree length. This means that for any set of points, any collection of lines shorter than this cannot span them.

An even earlier result (due to Hanan [Ha66]) literally provides a framework upon which to build rectilinear Steiner trees. By passing horizontal and vertical lines through each of the points we are able to form a grid. Figure 1 contains an example of such a grid over three points. Hanan's theorem states that a shortest rectilinear Steiner spanning tree can be formed from the edges of this grid. This provides our first estimate of the size of the feasible solution space for Steiner tree generation, namely the number of combinations of grid edges. This is $O(2^{2n(n-1)})$, a very large number.

0-8186-2610-0/92 $3.00 © 1992 IEEE

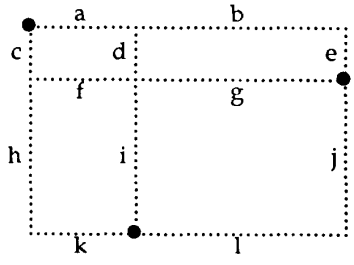

Figure 1 - An Induced Grid

3. The Search Space.

Branch and bound methods examine all feasible solutions for a problem in an orderly manner. First, the solution space is organized as a tree. Then the tree is searched, usually in a depth first manner. Thus providing a small search space is essential if we wish to actually perform branch and bound computations rather than merely discuss them.

Hanan's discovery that Steiner trees exist on the grid induced by the points provides our initial estimate of the search space size. Knowing that the search space consists of all combinations of grid edges establishes a size bound of $O(2^{2n(n-1)})$.

By limiting the types of combinations of grid edges needed to build Steiner trees, we may reduce the size of the solution space. A result from [JLV91] provides the first reduction.

> **Theorem 1.** *For any set of n points, there is a shortest Steiner spanning tree over the points which can be constructed by passing at most one line through each point.*

Consider the example found in figure 1. It contained a set of three points (which we shall name: top, bottom, and right) and their induced grid. All of the possible vertical and horizontal lines through each point are included below in table 1.

Restricting the Steiner trees examined to those containing at most n lines reduces the size of the feasible solution space to something close to $O(2^{n(2\log n-1)})$.

Lines Through Point on:		
top	bottom	right
a	k	fg
ab	kl	g
c	l	e
ch	di	ej
	i	j

Table 1 - Lines Through Points in Figure 1

The next two theorems (from [JLV91]) bring this down to $O(2^{n(2\log n-1)}/n^4)$. First we remove at least half of the lines from one of the points on the edge of the grid.

> **Theorem 2.** *There is a shortest Steiner spanning tree which does not contain a line along one of the four sides of the induced grid.*

This result allows us to select a side of the grid (we shall select the left one) and remove lines along it (in this case c and ch) from consideration as we build spanning trees. We also may remove all lines which intersect with these perimeter lines. (For our example this means k, kl, and fg.)

A related result removes even more perimeter lines.

> **Theorem 3.** *Lines need not span points on the perimeter of the grid in shortest Steiner spanning trees.*

Now the line ej can be removed from consideration. Summing it all up, we need not consider lines on one side of the grid or those connecting to them or perimeter lines which span points. Thus we only need examine combinations of the lines in the following table when we build shortest Steiner spanning trees.

Lines Through Point on:		
top	bottom	right
a	l	g
ab	di	e
	i	j

Table 2 - Essential Lines in Figure 1

We also employ a technique called *corner clipping* to even further reduce the size of our search space. This means omitting lines from corners which need not enter consideration when we build trees. This was used in both previous studies of exact Steiner tree algorithms.

4. A Pretaxial Algorithm.

Our branch and bound search tree now consists of all collections of lines which contain at most a line for each point. (We should note that if no two points share a vertical or horizontal coordinate, then we need exactly n lines.) And, in building the search tree we need only consider sets of lines selected from groups of lines for each point according to the rules put forth in the theorems of the last section.

A collection such as table 2 is exactly what is needed. The following algorithm constructs just such a table of lines.

```
construct(P, Line, size)
PRE:  P = {p1, ... , pn} = the set of points
POST: Line[,] = lines in search space
      size[i] = number of lines through pi

for all points pi
  place lines through pi in column i of Line[,]
remove lines on left grid boundary
remove lines intersecting left grid boundary
remove lines spanning points on perimeter
```

Figure 2 - Line Table Construction

This algorithm builds a table like that presented earlier for the example of figure 1. Our pretaxial search tree depends upon this table and is built from the table by selecting a line from the i-th column at the i-th level of the search tree.

Since the theorems of the previous section assure us that there is a shortest Steiner spanning tree composed of these lines, we may state the following and claim that the proof is a corollary to the theorems of the previous section.

> **Theorem 4.** *The pretaxial branch and bound search tree contains a shortest Steiner spanning tree.*

As an aid in pruning the search tree we order both the points and the lines which contain each point. Various orderings for the points are possible. Then depth first search is used to examine all of the nodes of the search tree. We merely travel down the tree to the left examining collections of lines each node until we reach a leaf. Then we backtrack and do this again and again until the tree has been searched.

As is usual with branch and bound algorithms we need not search the entire tree. We may discontinue our depth first search (and backtrack) whenever our collection of lines:

a) contains a line through each point,
b) contains a cycle,
c) is longer than the best spanning tree found so far, or
d) will never span the points.

The general pretaxial branch and bound algorithm is presented in figure 3. It is a depth first search of a search tree built by selecting lines from a table built by the construct algorithm and placed in Line[,]. Collections of lines (called tree) are made by selecting a line from each column of the table Line[,]. The variable place[point] keeps track of the line used last for that point. For the lines of table 2 we would examine all possible collections of lines (one from each column) in the following order.

{a,l,g}, {a,l,e}, {a,l,j},
{a,di,g}, {a,di,e}, {a,di,j},
{a,i,g}, {a,i,e}, {a,i,j},
{ab,l,g}, {ab,l,e}, {ab,l,j},
{ab,di,g}, {ab,di,e}, {ab,di,j},
{ab,i,g}, {ab,i,e}, {ab,i,j}

Note that all of the collections do span the points. But not all of the sets of lines are connected and thus are not *trees* much less spanning trees. So, we have a set of *spanning forests*. Collections such as {a,di,g} and {ab,di,e}) are spanning trees and of course {a,di,g} is the shortest Steiner spanning tree.

Checking to see if these collections are trees is done in two phases. First we compare tree.size to the size of the minimum spanning tree. If it is less than two-thirds of this length then by Hwang's theorem [Hw76] it cannot be a Steiner spanning tree. (And of course if besttree.size is smaller we ignore it.) In either case we lose interest in the collection.

```
pretaxial(P, Line, size, besttree)
PRE:  P = {p1, ... ,pn} the set of points
      Line[i,j] = i-th line passing through point pj
      size[i] = how many lines pass through pi
      besttree = minimum rectilinear
        spanning tree over P
POST: besttree = shortest rectilinear Steiner
        spanning tree over P

{initialize}
for i := 1 to n do place[i] := 0
point := 0
tree.size := 0
tree := ∅

repeat
  {construct collection of lines (the tree)}
  while tree.size < besttree.size and point < n do
    point := point + 1
    add Line[place[point], point] to tree
    adjust tree.size

  {check for best spanning tree}
  if tree.size<besttree.size
    and tree is indeed a tree
      besttree := tree
      bestree.size := tree.size

  {backtrack}
  for i := point + 1 to n do place[i] := 0
  repeat
    remove line[point, place[point]] from tree
    adjust tree.size
    place[point]:=(place[point]+1) mod size[point]
    point := point - 1
  until place[point + 1] ≠ 0 or point = 0
until point = 0 and place[1] = 0
```

Figure 3 - Pretaxial Branch and Bound Algorithm

If the tree we just constructed is in the correct size range then we must actually check to see if it spans the points. To do this we build a graph with the vertical and horizontal lines of the tree as vertices of the graph. The edges of the graph connect the vertices which represent intersecting lines.

Since vertical and horizontal lines (in the tree) must only intersect with each other, this construction forms a bipartite graph. (The two sets of nonconnected vertices represent the vertical and horizontal lines of the tree.) Figure 4 contains these graphs for the two collections of lines {a,di,e} and {a,di,g}.

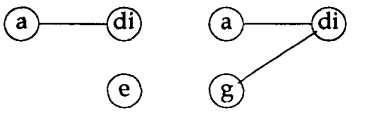

Figure 4 - Connection Graphs

Then, using depth-first search we can determine the number of components in the graph. If the graph is connected then the tree spanned the points. And, if there are backedges, then there is a cycle in the tree. This is a fairly fast computation since all of the information needed to construct any of these bipartite graphs can be tabulated prior to examining the branch and bound search tree.

5. An Epitaxial Algorithm.

Our second branch and bound technique is very much in the spirit of Yang and Wing [YW72]. We include one line per point as before, but instead of ordering the lines beforehand, from a seed (point), we grow a spanning tree in a manner much like Prim's algorithm for minimum spanning trees. This is designed to reduce the size of the search tree since lines not connected to the tree we are construction shall not be added to the tree.

Intuitively, the algorithm calls for us to select a point and note the lines which pass through it. Then for each of these lines we consider the lines which it intersects. We select one of these and add the lines which intersect that one to those we are considering. Then we keep selecting lines which intersected selected lines until we have selected one line through each point. This is the tree. Care must be exercised to select lines through points not in the tree so we get a spanning tree and of course no cycles.

Beginning at the top left point of the example in figure 1, the candidates for shortest Steiner spanning tree appear as paths in the search tree pictured in figure 5. Note that we use table 2 in building our epitaxial search tree. From the point on the left we must select either line a or ab. If we select a, then only line di visits another point. Thus we pick it. Now we must select line g to reach the point on the right. The cases using line ab are slightly more interesting since we do have choices.

Note that as expected we have a smaller branch and bound search tree than that of the pretaxial algorithm. This is due to the fact that each of the collections of lines in the search space is a Steiner tree

spanning the points.

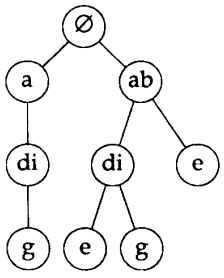

Figure 5 - Epitaxial Search Tree

The precise version of the recursive core of the algorithm appears as figure 6.

```
search(P, tree, pool, besttree)
PRE:  P = {p1, ...,pn} the set of points
      tree = a Steiner tree partially spanning P
      pool = lines connecting tree to pi not in tree
      besttree = a Steiner spanning tree
        over the set of points
POST: besttree ≤ shortest Steiner spanning tree
        below the node

for every line li in the pool
  remove li from pool
  newtree := tree + li
  adjust newtree.size
  if newtree.size < besttree.size then
    if all pi are in newtree then
      besttree := newtree
      besttree.size := newtree.size
    else
      newpool := pool-lines through same point as li
      place lines connecting li to the pi
        not in newtree in newpool
      if newpool ≠ empty then
        search(P,newtree,newpool,besttree)
```

Figure 6 - Epitaxial Branch & Bound Algorithm

It should be evident that we need not check to see if the tree built spans the points or is in fact a tree. Since we select lines from the tree to points not yet in the tree (just like Prim's minimum spanning tree algorithm) we end up with a tree spanning all of the points.

We must now show that the epitaxial algorithm does indeed find a shortest Steiner spanning tree over the points. Consider the following lemma.

> **Lemma.** *Any Steiner spanning tree can be built from any of its subtrees by adding lines through points not in the subtree to the subtree.*

The search space pruning theorems assure us that a line from the point on the left perimeter is in a shortest Steiner spanning tree composed of no more than n lines (one line for each point). If we begin by selecting the lines from the point on the left edge of the grid, the lemma maintains that we shall discover a shortest Steiner tree. This leads to the correctness result for this section.

> **Theorem 5.** *The epitaxial branch and bound search tree contains a shortest Steiner spanning tree over the set of points.*

6. Results and Conclusions.

Results from our empirical studies have been mixed. Both algorithms were programmed in C and run on a Sequent and an IBM 3090. As expected, the epitaxial algorithm outperformed the pretaxial algorithm handily. We feel that this was due to two reasons. First, no spanning or cycle checks needed to be done. And, even though connecting lines needed to be located, the smaller search space helped a great deal.

Processing up to seven point problems was quick. Times ran from 14 seconds on the Sequent to three seconds on the 3090. Since all of our data sets contained points which shared no coordinates, the grid sizes were maximum. This means that the search space for seven points was a little less than 2^{25}. Obviously pruning helped a lot.

Even 10 point problems were possible. Ours took about 1.5 hours on the 3090. (Note that the search space here is around 2^{60}.) Beyond 10 points sequential computation requires days on the 3090.

At present we have directed our efforts toward distributed and parallel processing since in effect we are searching a tree. We are adapting the DIB system [FM87] to the epitaxial algorithm and developing new parallel tree traversal techniques.

References

Be90 Bern, M. "Faster Exact Algorithms for Steiner Trees in Planar Networks." *Networks,* **20** (1990) 109 - 120.

FM87 Finkel, R. and U. Mamber. "DIB - a distributed implementation of backtracking." *ACM Trans. on Prog. Lang. and Sys.* **9** (1987), 235 - 256.

GJ77 Garey, M. R., and D. S. Johnson. "The Rectilinear Steiner Tree Problem is NP-Complete." *SIAM Journal of Applied Mathematics,* **32** (1977), 826 - 834.

Ha66 Hanan, M. "On Steiners Problem with Rectilinear Distance." *SIAM Journal of Applied Mathematics,* **14:2** (1966), 255 - 265.

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

JLV91 Joyce, P. M., F. D. Lewis, and N. Van Cleave. "The Feasible Solution Space for Steiner Trees." *Computer Science Technical Report 199-91* University of Kentucky, (1991).

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

Wi87 Winter, P. "Steiner Problem in Networks: A Survey." *Networks,* **17** (1987), 128 - 167.

YW72 Yang, Y. Y. and O. Wing. "Optimal and Suboptimal Solution Algorithms for the Wiring Problem." *Proc. IEEE Int. Symp Circuit Theory,* (1972), 154 - 158.