AN EFFICIENT RECTILINEAR STEINER TREE ALGORITHM FOR VLSI GLOBAL ROUTING

Shawki Areibi

School of Engineering
University of Guelph
CANADA N1G 2W1
E-mail: sareibi@uoguelph.ca

Min Xie and Anthony Vannelli

Electrical Engineering Dept
University of Waterloo
CANADA N2L 2G1
E-mail: vannelli@cheetah.vlsi.uwaterloo.ca

ABSTRACT

As we move to deep sub-micron designs below 0.18 microns, the delay of a circuit, as well as power dissipation and area, is dominated by interconnections between logical elements (i.e. transistors)[1]. The focus of this paper is on the global routing problem. Both global and channel routing are NP-hard[2]; therefore, all existing solution methodologies are heuristics. The main aim is to develop an efficient K Rectilinear Steiner Trees (K-RST) algorithm. A k-RST routine is developed to generate a set of rectilinear Steiner trees for each net. The K-RST uses local tree segment transformations to ensure that there is no duplication of routing trees for a net. The shortest tree for a net is in general 11% shorter than that of the minimal spanning tree, which leads to area savings.

Keywords — **Global Routing, Steiner Trees, VLSI Circuit Layout.**

1. INTRODUCTION

Physical design of VLSI circuits is the process of mapping structural representations of circuits into layout representation of circuits. Structural representations describe the system in terms of logic components and their interconnects. Layout representations define circuits in terms of a set of geometric objects which specify the dimensions and locations of transistors and wires on a silicon surface. After the placement phase, we have to connect the pins among the modules according to the specified netlist. The routing area of the chip can be represented by a grid graph. The vertices of the grid graph represent the potential pins and vias, and the edges represent the capacity of a channel or a switch box. A channel is the routing space between two blocks and a switch box is the routing space enclosed by the blocks. The routing stage is divided into **global routing** and **detailed routing**.

Both global[3] and channel routing are NP-hard[2]; therefore, all existing solution methodologies are heuristics. The focus of this paper is on the global routing problem[3]. The main aim of this paper is to develop an efficient K-RST algorithm. A k-RST routine is developed to generate a set of rectilinear steiner trees for each net.

Section 2 formally introduces the global routing problem. In Section 3, the minimum steiner tree algorithm is presented. Section 4 introduces some of the results we obtained for randomly generated test cases. Finally, Section 5 makes conclusions and proposes future work.

2. BACKGROUND

The goal of global routing is to determine the connection pattern for each net and to minimize chip area. The connection pattern is defined by positions for feedthroughs, the pins to be connected, and channels in which the net segments that connect the pins lie. The connection pattern for each net is represented by a rectilinear steiner tree. There exist two solution methodologies to global routing. One routes the nets one at time (sequential routing), and the other solves the global routing problem as an Integer Linear Program (ILP). Sequential global routing methods[4] depend heavily on the ordering of the nets and are not capable of predicting congested areas in channels when net segments are added. Global routing methods that formulate the problem as ILPs usually solve the relaxed form of the ILP using linear programming. These methods can solve large global routing problems using improved stat-of-the-art optimization techniques.

The primary subproblem in global routing is to find a minimal rectilinear steiner tree (MRST) for a net. Hanan [5] showed that all possible steiner points must lie within the bounding box of the outer most nodes of a net, and that any minimal rectilinear steiner tree can be found over the sub-

The research of the first author is partially supported by a Natural Sciences and Engineering Research Council of Canada (NSERC) operating grant (OGP 43417).

grid which is formed by the intersection of the vertical and horizontal lines drawn at each node in the net. Let us call the point of intersection of the vertical and horizontal line a Hanan point. Kahng and Robins [6] sub_divided rectilinear MST base rectilinear steiner trees into MST-overlap heuristics and Kruskal-Steiner heuristics.

2.1. Standard Cell Design

In standard cell design[7], a circuit is composed of a set of logic cells. Each logic cell has the same height and variable width, and each cell contains a number of connection points, called pins. There may be equivalent pins which allow connection at multiple locations on a cell. A net is a set of pins which must be interconnected. The cells are arranged into rows. The space between two rows is called a channel; channels are open-ended regions with pins on the top and bottom boundaries. Interconnection takes place in the channels. Assuming a two layer design[8], one layer is used for horizontal channel segments, while the second layer is used for vertical connections between the cell and the channel segments in the same channel. Inter-channel connection is accomplished by means of vertical tracks called feedthroughs; the feedthroughs are either inserted between two cells or it is provided by the cells. A net that span multiple rows must

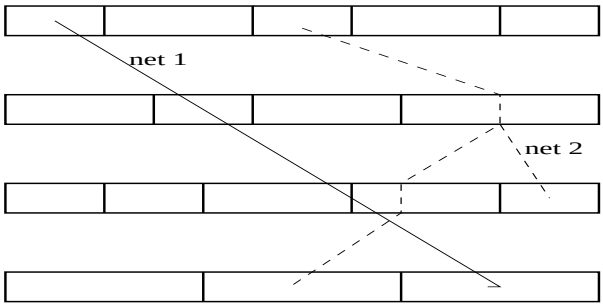

Fig. 1. Example of two nets that span multiple rows

use feedthroughs to complete its connections. If a net span r rows with no pin in the $r-2$ rows between its end points, then it needs $r-2$ feedthroughs to complete its connection. For a net with pins in between its extreme points, then the number of feedthroughs needed is less than $r-2$ since in standard cell, there are equivalent pins. These equivalent pins are used to connect the top segment of a net, with respect to the row, to its bottom segment. The two scenarios are shown in Figure 1.

The standard cell design style guarantees 100% routability, because the height of the channel is not fixed. When needed, feedthroughs can be inserted into the cell row to provide vertical inter-channel connection.
Inserting feedthroughs to cell rows except the longest row does not change the width of the chip.

The global routing problem is usually studied as a graph problem. The total area of the chip can be represented as a grid-graph where vertices are potential position of pins and vias. For standard cell layout style, each module of the circuit is represented by a vertex of the graph. If module c_i and c_j are placed adjacent to each other after placement phase, then there is an edge connecting the vertices that represent c_i and c_j. Figure 2 (b) show a grid-graph with 15 vertices. The number of vertices is determine by R_x and the number of cell rows in the chip. R_x is the row with maximum number of modules in the chip. If a row does not have the same number of cells as R_x, we put empty vertices in that row to make the grid-graph have the same number of vertices per row.

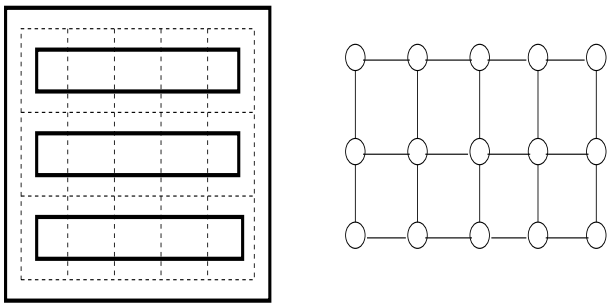

Fig. 2. Segmentation of the chip and the coarse grid graph.

3. SOLUTION METHODOLOGY

This section presents a efficient minimum steiner tree approximation method. Its solution is to the minimum steiner tree problem is at most $\frac{3}{2}$ time the optimal steiner tree.

3.1. Steiner Tree Generation

The K-RST problem is an extension of the minimum steiner tree problem, and it doesn't make it any more harder to solve. The Minimum Rectilinear Steiner Tree (MRST) problem can be stated as: Given a set of n points, P = [p_1, p_2, ..., p_{n-1}], find a set S of steiner points such that the minimum rectilinear spanning tree over P $\cup$ S has minimum length. The rectilinear distance/length in the Manhattan plan between two points $p_i(x_i, y_i)$ and $p_j(x_j, y_j)$ is equal to $|x_j - x_i| + |y_j - y_i|$

The steiner tree problem is a well known problem, and has been proven to be NP-complete [5, 6]. However, the optimal steiner tree can be approximated by a spanning tree using either Prim's or Kruskal's polynomial time MST algorithm. Steiner tree heuristics based on Prim's or Kruskal's spanning tree algorithm are called MST-overlap methods. Let the length of a minimum spanning tree represented by

cost(MST) and the length of the minimum steiner tree represented by cost(MRST). It can be shown that the worst case performance ratio of MST-overlap base MRST heuristics is bounded by $\frac{cost(MST)}{cost(MRST)} \leq \frac{3}{2}$. Thus the MST-overlap base MRST heuristics is at most a factor of $\frac{3}{2}$ times longer than the optimum steiner tree.

```
Steiner Tree Algorithm
1. ( P ∪ S, E) steiner( P)
2. E ← prim(P);
3. (T, F, gain) ← 1–steiner( P, E);
4. While ( gain > 0)
5.       P ← T;
6.       E ← F;
7.       (T, F, gain) ← 1–steiner( P, E);
8. End While
9. return  ( P, E);
```

Fig. 3. Iterated 1-steiner tree heuristic.

The heuristic used to generate one rectilinear steiner tree is illustrated in Figure 3. It is based on Prim's minimal spanning tree algorithm and the 1-steiner tree method [6]. The worst case performance of the 1-steiner method is shown in [6] to be strictly less than $\frac{3}{2}$ times the optimal steiner tree.

The function *prim* computes the minimal spanning tree using Prim's algorithm and it returns the edge set that represents the spanning tree. The edge set is a graph theoretical representation of a tree; i.e. a list of vertices and their parents. The vertices of the points are number from 0 to *n* - 1 where *n* is the number in P. Thus the parent of the root is vertex -1. The 1-steiner function takes the vertex and edge sets of a spanning tree as input and returns the vertex and edge set of the constructed 1-steiner tree, and the change in tree length that was the result of adding a steiner point to the spanning tree. The iterated 1-steiner tree method terminates when no more steiner points can be added to the spanning tree which reduce its length.

The 1-steiner function solves the 1-steiner tree problem. The 1-steiner tree problem is a special case of the general minimal steiner tree problem with the restriction that $|S| = 1$. Thus the 1-steiner tree problem is to find a spanning tree in the point set $P \cup \{s\}$ where {s} is chosen such that the gain of MST($P \cup \{s\}$) over MST(P) is greatest. MST($\bullet$) is the minimum spanning tree over a point set, and the gain is the reduction in tree length.

The principle used in the 1-steiner function is: visit all candidate points em s, and the spanning tree for $P \cup \{s\}$ is computed each time. The point that has the greatest gain is selected as the 1-steiner point. The set of candidate points is the set of Hanan points over P.

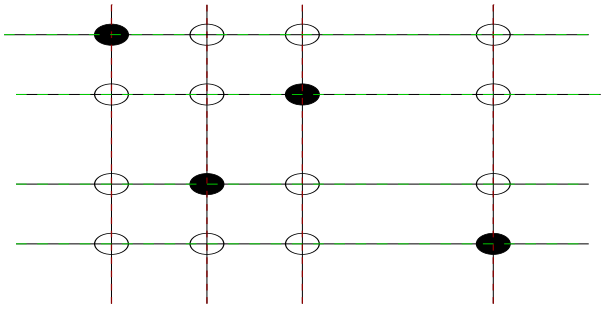

Fig. 4. The Hanan grid and steiner points for set P in black.

Hanan [5] proved that an optimal rectilinear steiner tree can be embedded in the grid composed of horizontal and vertical lines drawn through the points in P. The set of Hanan points are the intersecting points of all the vertical and horizontal lines. Figure 4 shows a point set in solid dots and its Hanan grid and points. For a point-set with n points, there are O(n^2) possible Hanan points.

Since there are O(n^2) possible Hanan points, constructing a minimum spanning tree for each $P \cup \{s\}$ point set just to find the point that has the highest gain is computationally expensive. A linear-time dynamic spanning tree maintenance method is developed by [6]. It involves computing a spanning tree over $P \cup \{s\}$ given the minimum spanning over P. This linear time spanning tree update scheme is based on the uniqueness property.

The uniqueness property says that all vertices of a spanning tree have at most degree of four in the 2-dimension Manhattan plan. The proof of this property is in [6]. Immediate from uniqueness property is that given a point *s*, two lines that crosses *s* at +45 and -45 degrees partition the plane into four regions, each region contains at most one point closest to *s*. Thus every point of set P in the 2-dimensional Manhattan plane has an MST with maximum degree ≤ 4. To add a point *s* to a existing tree, it is sufficient to connect it to its four closest neighbor in the spanning tree.

The 1-steiner algorithm is shown is Figure 5. It takes a set of vertices V and edges E as input and returns a set of vertices W, a set of edges F, and the gain of vertex n+1. Lines 3 to 8 is the body of the function. It steps through all Hanan points for the set of vertices V and keeps track of which Hanan point *s* results in the maximum gain. The function *spanning_update* finds the spanning tree over V $\cup$ {s} using the dynamic spanning tree maintenance.

The spanning tree maintenance is performed by the function *spanning_update* and it is shown in Figure 6. It takes as input a set of vertices V, a set of edges E, and a point *s*, and returns the vertex set V $\cup$ {s}, a set of edges F, and the gain induced by *s*. The *for loop* steps through the four partitions for point *s*. Each partition is labeled as a quadrant. The

```
1–Steiner Tree Algorithm
1. ( P ∪ s, E, gain) 1–steiner( V, E)

2. maxgain ← 0;
3. for each s ∈ H
4.       ( W, F, gain) ← spanning_update( V, E, s);
5.       if( gain > maxgain) then
6.            maxgain ← gain;
7.            maxpoint ← s;
8. end for
9. if (maxgain > 0) then
10.      (W, F, gain) ← spanning_update( V, E, maxpoint);
11.      return (W, F, maxgain);
12.  else return (W, E, 0);
```

Fig. 5. Pseudo-code for 1-steiner algorithm.

```
Spanning Tree Update Algorithm
1. ( V ∪ s, F, gain) spanning_update( V, E, s)

2. delta ← 0;
3. V ← V ∪ s;
4. F ← E;
5. for each d ∈ { quadrant 1, 2, 3, 4 }
6.       u ← closest_point( V, s, d);
7.       delta ← delta — distance( s, u);
8.       F ← F ∪ {(s, u)};
9.       if (cycle( V,F))
10.           ( v, w) ← largest_cycle_segment( V,F);
11.           F ← F \ {(v, w)};
12.           delta ← delta + distance( v, w);
13. end for
14.  return (V, F, delta);
```

Fig. 6. Dynamic Update of Spanning Tree.

quadrants are labeled in a counter wise order with right partition being quadrant 1. For each partition, the closest point *u* to *s* is found in line 6 using the *closest_point* function. The change in tree length is calculated by adding edge *s, u* to the tree. The function *cycle* in line 9 traverses the spanning tree looking for a cycle in the tree that is the result of adding edge *s, u* to the it. If there is a cycle, the longest segment of the cycle is found by the function *largest_cycle_segment*, and it is removed from the tree. The change in tree length is calculated by the removal of the longest segment.

Finding the closest pin for each region is done by visiting all points in that quadrant and saving the closest one. Cycle detection is done by a depth-first search and signaling the presence of a cycle when we visit a vertex twice. Once a cycle is detected, finding the longest segment in a cycle is the simple task of going through the cycle again.

Figure 7 shows the progress of the *spanning_update* when a Hanan point *s* is added to the spanning tree. The connection is shown as arcs connecting the vertices. This emphasis that the actual L-orientation has not been chosen to connect the points in the Manhattan plane. The number beside each

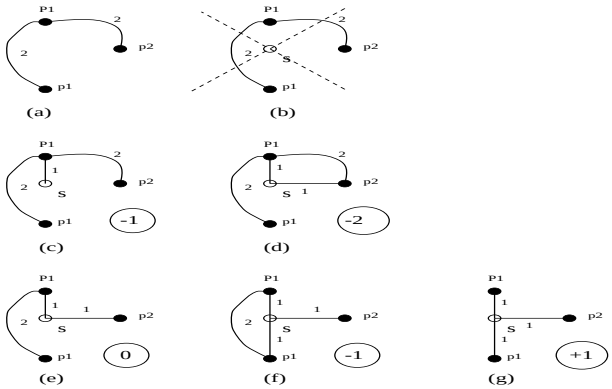

Fig. 7. Steps in the dynamic spanning tree update function.

arc is its length in the Manhattan plane. The circled number on the bottom right is the gain by adding/removing an edge.

3.1.1. K-Minimum Rectilinear Spanning Tree

The K-rectilinear spanning tree problem is to generate k-rectilinear minimum spanning tree (K-RST) for a given point set P. Formerly the K-RST problem is stated as: Given a point-set P = $\{p_0, p_1, \ldots, p_{n-1}\}$ find a set S = $\{S_0, S_2 \ldots S_k\}$ such that $P \cup S_i$ is a minimum spanning tree, and $S_i \neq S_j$ where each $S_i \subseteq$ H and H is the set of Hanan points. Thus generating K rectilinear spanning trees is equivalent to generating K different sets of steiner points.

```
Batch 1–Steiner Tree Algorithm
1. ( S) B1–steiner( V, E)

2. gain ← 0;
3. S ← ∅;
4. for each h ∈ H
5.       ( W, F, gain) ← spanning_update( V, E, h);
6.       if( gain > 0) then
7.            S ← S ∪ h;
8. end for
9.  return (S);
```

Fig. 8. Batch 1–Steiner heuristic.

A variant of the Iterated 1-steiner method is used to solve the K-RST problem. This variant depends on the batch 1-steiner algorithm shown in Figure 8. The batch 1-steiner algorithm finds a batch of steiner points in the set of Hanan points that induce a gain to the *original MST*. The *B1-steiner* functions finds a set S of "good" Hanan points. It takes as input the original set of vertices V, and its set of edges E which represents the spanning tree of V. Lines 2 and

3 initialize the gain and set S to be 0 and null respectively. Lines 4 to 8 are the body of the *B1-steiner* function. It steps through all points *h* in H, and finds the gain of $\{V \cup h\}$ for *h*. If *h* induces a positive gain, a reduction in tree length, then we save *h* in S. *B1-steiner* returns the set S.

After identifying the set S of steiner points that induce a gain in the *original MST*, all possible combination $S_i \subset S$ are generated. A spanning tree is generated for each $\{P \cup S_i\}$ using the 1-steiner algorithm. But instead of stepping through points in H, we step through points in S_i. The gain induced by each S_i on the original MST is noted, and the K highest gain steiner sets S_i is choosen to provide a result to the K-RST problem. As proving the performance ratio of a approximation method is very difficult, the performance of the described k-RST method is not analyzed.

3.1.2. L-edge Generation and Local Tree Modification

Once steiner trees for all nets have been generated independently, we have to map the tree arcs to the underlying edges in the grid graph G. Each arc in a tree is mapped to an L–edge, using a specified L–orientation. An L–edge consists of a horizontal segment and/or a vertical segment depending on the location of a vertex and its parent. If a vertex and its parent are located on the same horizontal line then the arc is only mapped to a horizontal segment; the same applies for the pair of points that lie on the vertical line. If a pair of points is non-colinear, then both horizontal and vertical segments are needed to connect the points. Both segments are located on the border of the boundary box for the pair of points.

L–orientation has two values, BOTTOM_ORIEN or TOP_ORIEN; it is choosen randomly for each tree. If L–orientation is equal to TOP_ORIEN then we connect a pair of points using its top boundary box border and then complete the connection using either its left or right border which depends on whether bottom vertex to the left or right of the top vertex. The same is true for using BOTTOM_ORIEN to connect a pair of points except we use its bottom border, and then choose the left or right border depending on the location of the top vertex with respect to the bottom vertex. Figures 9(a) and (b) show the mapping of edge segments using BOTTOM_ORIEN and TOP_ORIEN L–orientation respectively for a pair of non-colinear vertices.

When we finish mapping the arcs of a tree to the edges of the grid graph, we check the edge usage to make sure that each edge is used once for a tree. If an edge is used more than once, then the extras are removed from the edge set of the tree. This is to make sure that we don't over estimate the edge usage, and force it closer to the edge's capacity which would make the solution to global routing problem inferior.

After the above steps, we have to make sure that the sets of edges used to map the set of trees for a net is different.

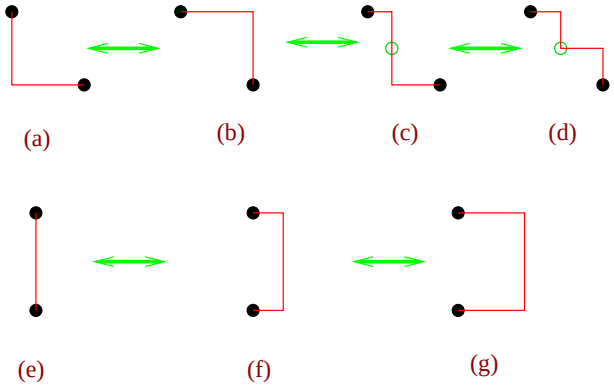

Fig. 9. L-edge and local transformation for steiner trees.

Otherwise, we have the same tree twice for a net, which is undesireable. If two trees for a net use the same edges, we employ a local tree modification technique to change the edges used by one of the trees. We modify a tree by using local transformation of the L–edge used to map an arc to the edges. Example of such transformations are shown in Figure 9. Let us suppose that Figure 9(a) is the original L–shape connection for the pair of vertices. Figure 9(b) to (d) shows some of the possible transformation for this arc. Figure 9(b) is simply the remap of the arc using the other L_orientation. In Figure 9(c) and (d) we introduce a virtual point shown as the empty circle and map the L–edge between the top point and the virtual point using TOP_ORIEN. The L–edge connection between the virtual point and the bottom point is mapped using BOTTOM_ORIEN for Figure 9(c) and TOP_ORIEN for Figure 9(d). All the above discussed L–edge transformation are within the bounding box of the pair of vertices. These type of edge transformations do not increase the length of the tree. However when we have a pair of colinear points we have to use the U-shape connection shown in Figure 9(f) and (g) for the pair of points shown in Figure 9(e).

4. EXPERIMENTAL RESULTS

In this section we present some of the results obtained using K-RST algorithm developed in Section 3. Table 1 shows the results of the K-RST program for nets with 3 to 10 modules on a 15x15 grid. The nets are randomly generated. Five trees are generated for each net. The length of the RSTs are the lengths returned after each 1-steiner iteration. Since a steiner point of degree two is not included in the steiner set, then there are a limited number of valid steiner points that could reduce the rectilinear spanning tree (as in the case when the number of modules is small). Therefore, a way to generate more steiner trees is to include two degree steiner points in the steiner set. When the number of modules in a

net is large, as with the 10 module net, the number of steiner sets that give rise to near minimal rectilinear steiner trees is also increased.

Table 2 shows random nets in diferent grid sizes. It also shows the percent improvement of rectilinear steiner trees over minimal spanning trees. As the table shows, the average improvement is around 12%, which is close to what is reported in [6].

# of points	MST	RST #1	2	3	4	5
3	15	13	15	15	15	15
4	22	20	20	21	22	22
5	20	18	19	18	19	19
6	31	27	28	28	28	29
7	30	24	25	26	25	26
8	42	37	38	38	39	39
9	34	32	33	32	33	33
10	85	74	74	74	75	74

Table 1. Lenght of MST and 5 RSTs for nets with different modules.

# of Points	Grid Size	MST	RST	% Improved
15	30	42	36	14.1
15	60	193	169	12.4
30	60	259	233	10
30	120	584	516	11.6

Table 2. Random points with different grid sizes.

The K-RST routine generates K trees for each net. Small terminal nets have less steiner sets due to the limited number of Hanan points. This is shown in Table 1; 3 and 4 terminal nets have lengths close to the minimal spanning tree. While nets with 5 or more terminal nets have many steiner sets that generate near minimal steiner trees. Therefore, the number of Hanan points greatly influence the number of different steiner sets.

5. CONCLUSION

The solution of the global routing problem requires a set of rectilinear steiner trees for each net. In this paper a K-RST routine was developed to generate the set of trees for each net. The trees for a net are guaranteed to be different by local edge transformation. The shortest tree for a net is in general 11% shorter than that of the minimal spanning tree, which leads to area savings.

As feature sizes are scaled down, devices are much faster. Which results in an increase of clock rate – i.e. smaller cycle time. The interconnect delay can consume from 50% to 70% of the clock cycle [9, 10]. Therefore, to stay competitive, the routing trees for each net have to statisfy delay requirements, which require high performance interconnect topologies. Future implementation should consider the generation of high performance trees for each net.

6. REFERENCES

[1] N. Sherwani, *Algorithms for VLSI Physical Design Automation*, Kluwer Academic Publishers, Boston, 1993.

[2] G. Hachtel and C. Morrison, "Linear Complexity Algorithms for Hierarchical Routing", *IEEE Transactions on Computer Aided Design*, vol. 8, n. 1, pp. 64–80, 1989.

[3] A. Vannelli, "An Adaptation of An Interior Point Method for Solving The Global Routing Problem", *IEEE Transaction on CAD/ICAS*, vol. 10, n. 2, pp. 193–203, 1991.

[4] C. Sechen and A. Sangiovanni, "The TimberWolf 3.2: A New Standard Cell Placement and Global Routing Package", in *Proceedings of The* $23rd$ *DAC*, pp. 432–439, Las Vegas, Nevada, June 1986, IEEE/ACM.

[5] M. Hanan, "On Steiner's Problem with Rectilinear Distance", in *VLSI Circuit Layout:Theory and Design*, pp. 133–138, New York, 1985, IEEE PRESS.

[6] A. Kahng and G. Robins, *On Optimal Interconnects for VLSI*, Kluwer Academic, Boston, MA, 1995.

[7] J.T. Mowchenko and C.S. Ma, "A New Global Routing Algorithm for Standard Cell ICs", in *IEEE International Symposium on Circuits and Systems*, pp. 27–30, Philadelphia,Pa, 1987, IEEE.

[8] R.M. Karp and F.T. Leighton, "Global Wire Routing in Two-dimensional Arrays", *Algorithmica*, vol. 2, pp. 113–129, 1987.

[9] J. Lillis, T.Y. Lin and C.Y Ho, "New Performance Driven Routing Techniques with Explicit Area/Delay Tradeoff and Simultaneous Wire Sizing", in *Proceedings of* $33rd$ *DAC*, Las Vegas, Nevada, June 1996, ACM/IEEE.

[10] J. Cong and P. Madden, "Performance Driven Global Routing for Standard Cell Design", in *International Symposium on Physical Design*, Apr 1997.