Rectilinear Steiner Tree Construction with Obstacles

Hsin-Hsiung Huang[1], *De-Jing Huang*[2] *and Tsai-Ming Hsieh*[3]

[1]Institute of Electronic Engineering, Chung Yuan Christian University, Chung-Li, Taiwan, R.O.C.
[2]Department of Information and Computer Engineering, Chung Yuan Christian University, Chung-Li, Taiwan, R.O.C.
[3]Department of Computer Science and Information Engineering, National Taitung University, Taitung, Taiwan, R.O.C.
Email: bear@fpga.ice.cycu.edu.tw; johnson@fpga.ice.cycu.edu.tw; hsieh@nttu.edu.tw

ABSTRACT

We propose a three-stage spanning-graph based algorithm to connect all pins with obstacles. At the first stage, we construct the spanning graph according to the given pins and the four corner points of the obstacles. At the second stage, the searching algorithm is performed to find a sub-graph, the spanning tree, and transform the spanning tree into a rectilinear Steiner tree at the third stage. The objective of the algorithm is to minimize the total wire-length. Compared with the results without obstacles, it shows experimentally that our algorithm can obtain the routing solution with only 1.4% extra total wire-length and 9.86% additional via counts with consideration of obstacles.

1: INTRODUCTIONS

As the fabrication technology scaling, routing becomes an important stage in the physical design. It is fundamental to find the rectilinear Steiner minimal tree (RMST) for a set of pins of a multi-pin net. Ganley *et al.* [2] prove the problem to be NP-complete that there is no polynomial-time algorithm to solve the RMST. Obviously, the complexity of the problem will increase dynamically when we consider the obstacles, including the macro cells, IP and pre-routed nets.

For the RMST problem, many algorithms, including the sequential approach and spanning graph-based method, are proposed. Lee [5] proposes a maze-based router, but it takes large memory space and timing consumption. The line search technique can find the solution quickly on the continuous plane [7]. Kahng *et al.* [4] introduce the batched method to achieve a good solution with the complexity of $O(n \log^2 n)$, where n is the numbers of pins. Borah *et al.* [1] perform the $O(n^2)$ edge substitution method which iteratively connects a pin to the near edge and deletes the longest edge of the cyclic edges, where n is the numbers of the given pins. Zhou [9] provides an efficient $O(n \log n)$ algorithm which combines the concept of edge substitution with the spanning graph, where n is the numbers of the pins. However, some methods don't take obstacles into consideration and some algorithms achieve good routing quality with in-acceptable runtime.

There are more and more macro cells and IP in the design, and the routing paths are not allowed to go through the area, i.e. obstacles. Ganley *et al.* [2] give an algorithm which achieves the optimal solution for three or four terminals with obstacles. Yang *et al*. [8] propose a sequential approach that first constructs directly a RMST and then remove the edges inside the obstacles. Hu *et al.* [3] develop the an-OARSMan which is based-on the track graph and takes an efficient method to reduce the searching memory space. In addition, the approach can handle complex obstacles such as convex and concave polygon obstacles. Shen *et al*. [6] give a $O(n' \log n')$ method which first constructs a connection graph according to the pins and blockage boundaries and finds a sub-graph from the connection graph, where n' is the sum of pins and obstacle boundaries. Although the method can handle obstacles, the total wire-length is still too long for large size circuits. Besides, the via count of the graph-based approach are large.

We develop a spanning-graph based algorithm which handles the RMST with obstacles in the paper. The first contribution of the paper is presented a method with complexity of $O((8m + 2n) \times s \log s)$, where m is the numbers of obstacles, n is the numbers of the given pins and s is the sum of the numbers of given pins plus the corner points of the obstacles. The second contribution is to connect the edge with the idea of global view of pins and the obstacle boundaries. The proposed method, which can explore the more edges globally, improve the total wirelength of the greedy method that searches the nearest local points. With the global view of the pins and boundaries of obstacles, our method can obtain the shorter total wire-length. In addition, our method finds the solution with less numbers of via than old method.

We organize the paper as follows: section 2 and 3 illustrates our motivation and problem definition, respectively. In the section 4, the spanning graph-based method is explained in detailed. Experimental results are shown in section 5, and we make some conclusions and further work in section 6.

2: MOTIVATION

The RSMT problem with obstacles is usually solved by the graph-based and the sequential approach. The former methods usually first construct the spanning graph and then perform the searching algorithm to obtain a sub-graph. The latter approaches first build the routing tree without obstacles and then remove the edges inside obstacles. We know that the result of the sequential method will be limited to the initial routing tree. However, the spanning graph method can obtain better results due to the global view of the obstacle boundaries and pins. Therefore, we choose the spanning graph based approach in the paper.

Figure 1 shows the importance of considering the issues of total wire-length and via counts at the global routing. Assume there are five obstacles and four pins in

This work was supported by the National Science Council of Taiwan, R.O.C., under Grant NSC 95-2221-E-033-085.

a chip. If we search the nearest point by the concept of greedy, the result may contains 7 via counts and longer total wire-length (47 unit length), see Figure 1(a). With the global view, our method can find the routing solution with 2 via counts and the shorter wire-length (41 unit length) in Figure 1(b).

By the observation of Figure 1, it motivates us to find a spanning graph-based approach which obtains the solution with minimization of total wire-length and the via counts.

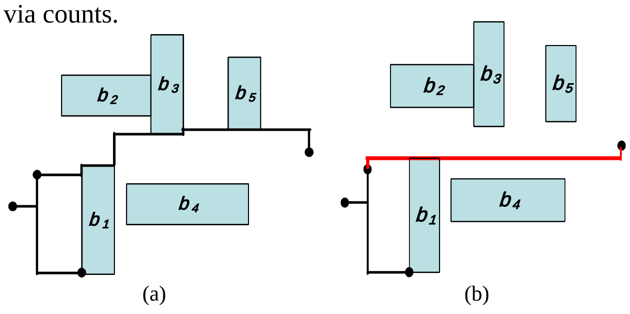

Figure 1 The previous method iteratively connects to nearest point greedily and results in a little of bending. (b) We record properly the useful information to reduce the total wire-length and via count.

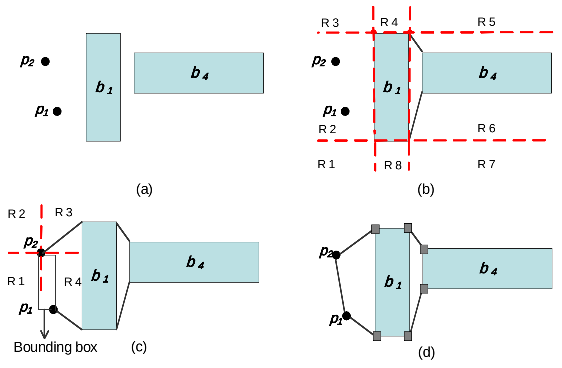

Figure 2: (a) Original circuit. Perform edge connection for (b) R6 of *b1* and (c) R4 of *p2*. (d) A spanning-graph result.

3: PROBLEM FORMULATION

In the section, we first describe the terminology used in the paper and then formulate the problem that we will solve in this paper.

3.1: TERMINOLOGY

For a set of *n* pin of a multi-pin net and a set of rectilinear obstacles, a spanning graph *G* is an undirected graph to connect all pins. *G* is regarded as an intermediate step in the minimal spanning tree construction.

In Figure 2, we define the searching regions which are used to construct the spanning graph. For each obstacle b_i, the plane is partitioned into eight regions by the four boundaries of the obstacle, see Figure 2(a). For each pin p_i, the full chip in Figure 2(b) is divided into four regions by the horizontal, vertical line passing through p_i.

For the spanning graph *G*, we iteratively use the searching techniques, including the traversal and tracking back, to find a spanning tree *T*. For each v_i in *G*, we find the neighbor nodes connected to v_i and add the nodes u_{ij} into *T* if u_{ij} is also the pins.

3.2: THE PROBLEM

Let $P=\{p_1, p_2,\ldots, p_n\}$ to be a set of pins of the *n*-pin net and a set of *m* rectangular obstacles, $B=\{ b_1, b_2,\ldots, b_m\}$ with the corner points $C=\{ c_1, c_2,\ldots,c_{4m} \}$. We can find a graph $G=(V, E)$, where $V =\{ v_1, v_2,\ldots, v_s \}=P\cup C$ and $E=\{ e_1, e_2,\ldots,e_t \}$. For each $v_i \in V$ has coordinate (x_i, y_i) and the distance between v_i and v_j is $|x_i-x_j| + |y_i-y_j|$ .Our objective is to find a rectilinear Steiner minimal tree, RMST, which connects all pins with some additional Steiner points to minimize the total wire-length with obstacles.

4. OUR ALGORITHM

In this section, we first give the pseudo code of our algorithm and then describe the operation by a simple example. Finally, we analyze the time complexity of the algorithm. The following is the pseudo code:

4.1: THE PROPOSED ALGORITHM

```
Algorithm: A spanning-graph-based Rectilinear Steriner Tree with obstacles
Input:  A set of point, $V = P \cup C$ with the sef of obstacles.
Output:A rectilinear steiner tree.
Method:
  //Stage 1: Constuct the spanning graph $G$ with obstacles
   sort $V$ by increasing order of $x$ coordinate;
   for each block $b_j \in B$
       Perform edge connection for points located at R2 and R6 of block $b_j$;
   endfor
   sort $V$ by increasing order of $y$ coordinate;
   for each block $b_j \in B$
       Perform edge connection for points located at R4 and R8 of block $b_j$;
  endfor
  sort $P$ by increasing order of $x+y$;
  for each pin $p_i \in P$
       if (there is no obstacles within the bounding box forming by $p_i$ and corner points )
           Perfrom edge connection for the points located at R3 and R1 of pin $p_i$;
  endfor
  sort $P$ by increasing order of $x-y$;
  for each pin $p_i \in P$
       if (there is no obstacles within the bounding box forming by $p_i$ and corner points )
           Perform edge connection for the points located at R2 and R4 of pin $p_i$;
  endfor

 //Stage 2: Search the $G$ to obtain a spanning tree $T$
 for each $v_i \in G$
     $U_i = neighbors(v_i)$;
     for $u_{ij} \in U_i$
         if( $u_{ij}$ is a given pin )
             $V = V - \{u_{ij}\}$; $T = T \cup \{u_{ij}\}$;
     endfor
  endfor
//Stage 3: Transform Spanning tree $T$ into Rectilinear steiner tree $T'$
```

Figure 3: The pseudo code of our algorithm

The algorithm which build a RMST with obstacles contains three stages, including constructing a spanning graph, searching to get a spanning tree and transforming it into a RMST. Inspired by the sweep line algorithm, we construct the graph by performing the edge connection. Unlike the previous method [6], we record all 2-pin edges which are not inside obstacles. The algorithm will be explained completely as follows.

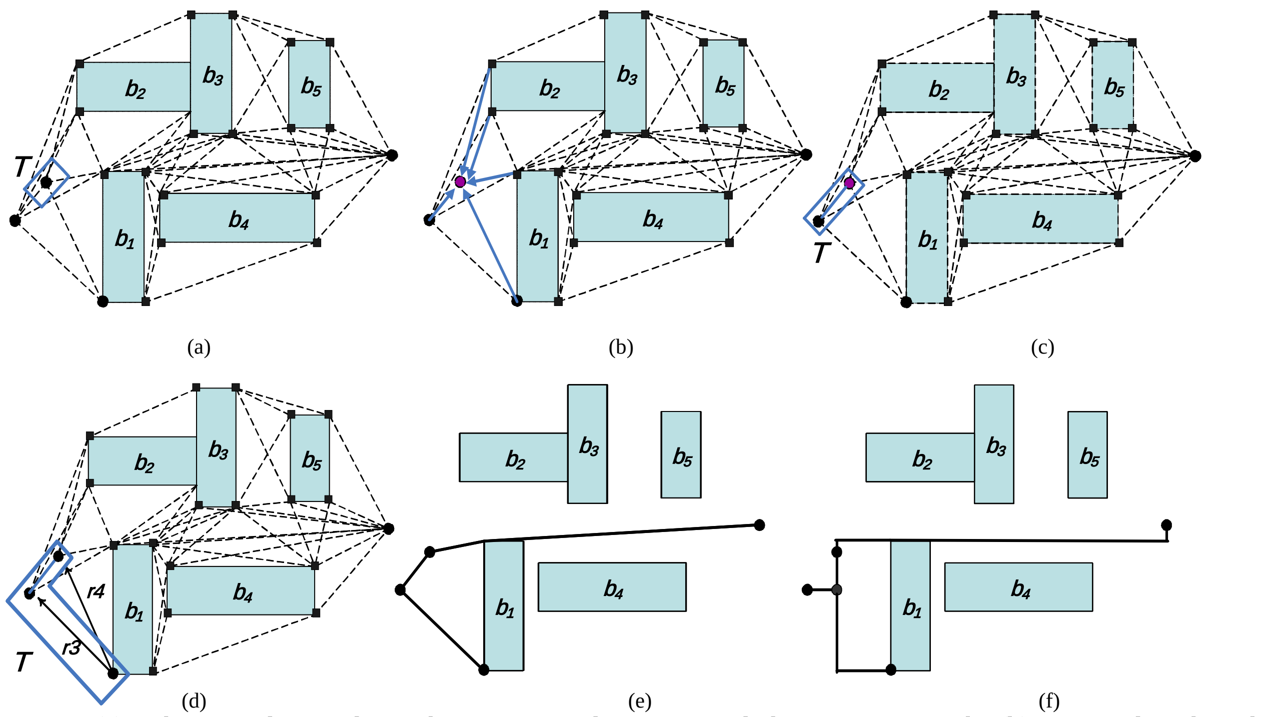

Figure 4: (a) Take a random node as the starting node to traversal the spanning graph. (b) Traversal each path connecting the starting node. (c) Trace back and add the pin into the spanning tree *T* when the neighbor node is also a pin. (d) The BFS-like traversal and add the shorter path *r3* into the spanning tree if the length of *r4* is greater than *r3*. (e) A minimal spanning tree is obtained by iteratively touring and tracing back. (f) Transform the tree into a RMST.

At the first stage, the modified edge connection (see Figure 2) is applied to construct the spanning tree. To simplify the edge connections, we only connected nodes located at the searching regions that are discussed in section 3.1. Firstly, all nodes in *V* are sorted by their *x*-coordinates with the increasing order. For each block $b_j \in B$, we perform edge connection for R2 and R6 (refer Figure 2(b)) of all corners with the information of the sorted nodes. Secondly, we sort the nodes by their *y*-coordinates with the increasing order and perform edge connection for R4 and R8 of all corners in each block. Thirdly, we connect to pins that are located at R1 and R3 (refer Figure 2(b)) by the increasing order of *x*+*y*. In similar way, we perform edge connection for the pins that are located at R2 and R4 by the ordering of *x*-*y*.

At the second stage, the searching algorithm is performed to find a feasible sub-graph. For the spanning graph of Figure 4(a), which contains the squared corner points and circular pins, we start from a random node v_i and the spanning tree contains only a root (v_i). For each node, which is either a corner point or a pin of the spanning graph in Figure 4(b), we traverse each path connected to v_i and trace back the node information, see Figure 4(c). When the path starts at one pin of the spanning tree and ends at another pin, the pin and the shorter path are added into the spanning tree in Figure 4(d). It terminates until all the pins are visited, *i.e.* the spanning tree contains all pins in Figure 4(e).

At the third stage, we transform the spanning tree into a rectilinear Steiner tree which considers the shared edge between the Steiner points, see Figure 4(f).

4.2: TIME COMPLEXITY ANALYSIS

For each stage, we first present the time complexity and then explain the operation. At stage 1, the spanning graph is constructed by an O ((8m + 2n) × s *log* s) algorithm, where *m* is the numbers of obstacles, *n* is the numbers of the given pins and *s* is the sum of the numbers of given pins plus the corner points of the obstacles. The nodes of *V* are first sorted by the increasing order of *x* (*y*)-coordinate and connect the corner points to the nearest points of R2 and R6 (R4 and R8), respectively. In addition, we sort all nodes of *V* in the increasing order of *x* + *y* (*x* - *y*) and perform edge connection for pins. Hence, the complexity is $O((8m + 2n) \times s \log s)$.

At stage 2, we first take a random point of the spanning graph as the starting point. We then traversal the neighbor nodes v_i connected to the starting point and add u_{ij} to the spanning tree *T* if the u_{ij} is also a given pin. Finally, the algorithm will terminate if all the given pins are visited and the complexity of stage 2 is $O(s \times w)$, where *s* is the sum of numbers of pins plus the numbers of corner points of the obstacles and *w* is the numbers of neighbor nodes obtained at each traversal iteration.

At stage 3, we transform the edges of the spanning tree into a rectilinear Steiner tree that contains a set of horizontal and vertical segments. The complexity is $O(t)$, where *t* is the numbers of edges in spanning tree.

In other words, the complexity of algorithms is $O((8m + 2n)\times s \log s)$, where *n* is the number of pins, *m* is the number of obstacles and *s* is the number of corner points of obstacles and the number of pins, see Table 1.

Table 1: Time complexity of each operation

Operation	Complexity
Construct a spanning graph	*O((8m +2n) ×s log s)*
Sort the vertices	*O(s log s)*
Search the spanning graph	*O(s × w)*
Transform the spanning tree into the rectilinear tree	*O(t)*

5: EXPERIMENTAL RESULTS

The experiments were implemented by using C++ language in Intel 2.4G machine with 256MB memory. The objective is to obtain a RMST with minimal wire-length and less via count under the obstacles.

Table 2 shows the statistics of ten benchmarks which we randomly generate under the 151000×151000 chip size. The second and third columns denote the numbers of pins and obstacles, respectively. Total wire-length and numbers of Steiner points are listed in fourth and fifth column, respectively. The final column is the run time. Compared with the method without obstacles (named A), the average additional wire-length and the extra numbers of Steiner points of our algorithm with obstacles (named B) are 1.4% and 9.86%, respectively. From the table, it shows that our algorithm can effectively construct the Steiner tree with 1.4% extra wire-length with obstacles.

6: CONCLUSION AND FUTURE WORK

In this paper, we implement the spanning graph-based algorithm which constructs the rectilinear Steiner tree with obstacles. The spanning graph-based method first constructs a spanning graph by the pins and the obstacles boundaries. A graph searching technique is then applied to explore a sub-graph, a spanning tree, among the spanning graph. Finally, we transform the spanning tree into a rectilinear Steiner tree. Compared with the approach without obstacles, our method can obtain a routing solution with a little extra wire-length (1.4%) and the numbers of Steiner point (9.86%). Our future work is to speedup the algorithm by modifying the procedure of constructing the spanning graph.

REFERENCES

[1] M. Borah, R.M. Owens and M.J. Irwin, " An Edge-Based Heuristic for Steiner Routing," *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 13, pp. 1563-1568, 1994.

[2] J. Ganley and J.P. Cohoon, "Routing A Multi-Terminal Critical Net: Steiner Tree Construction in the Presence of Obstacles," In *Proc. of IEEE International Symposium on Circuits and Systems*, Vol. 1, pp. 113-116, 1994.

[3] Y. Hu, T. Jing, X. Hong, Z. Feng, X. Hu and G. Yan, "An-OARSMan: Obstacle-Avoiding Routing Tree Construction with Good Length Performance, " In *Proc. of IEEE/ACM Asia and South Pacific Design Automation Conference*, pp. 7-12, 2005.

[4] A. B. Kahng and G. Robins, "A New Class of Iterative Steiner Tree Heuristics with Good Performance," *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 11, pp. 893-902, 1992.

[5] C. Y. Lee, "An Algorithm for Connection and Its Application," *IRE Transaction on Electronic Computer*, pp. 346-365, 1961.

[6] Z. Shen, C.N. Chu and Y.M. Li, "Efficient Rectilinear Steiner Tree Construction with Rectilinear Blockages," In *Proc. of IEEE International Conference on Computer Design*, pp. 38-44, 2005.

[7] J. Soukup, "Fast Maze Router," in *Proc. of ACM/IEEE Design Automation Conference*, pp.100-102, 1978.

[8] Y. Yang, Q. Zhu, T. Jing, and X.L Hong, "Rectilinear Steiner Minimal Tree among obstacles," In *Proc. of International Conference on ASIC*, pp. 348-351, 2003.

[9] H. Zhou, "Efficient Steiner Tree Construction Based on Spanning Graphs," *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 24, No. 7, pp. 704-710, 2005.

Table 2: Comparison of two algorithms which construct rectilinear steiner tree without and with obstacles.

Testcase	Pins	Obstacles	Total Wire-length			No. of Steiner Points			Run Time (sec.)	
			A	B	C	A	B	C	A	B
T1	5	500	13348	14017	5.0%	1	2	100.0%	0	218
T2	10	500	23888	25051	4.8%	4	6	50.0%	0	219
T3	15	500	28545	30148	5.6%	5	6	20.0%	0	237
T4	30	500	45691	46965	2.7%	12	13	8.33%	0	230
T5	50	500	53797	55915	3.9%	20	24	20.0%	0	262
T6	100	500	77103	80407	4.3%	45	46	2.22%	0	310
T7	150	500	93825	97590	4.0%	72	82	13.89%	1	334
T8	200	500	110133	113,569	3.1%	96	117	21.88%	1	473
T9	200	1000	110133	116,361	5.6%	96	120	25.0%	1	1653
T10	1000	1000	3566745	3,601,493	0.9%	479	495	3.34%	22	14796
Avg	-	-	412320.8	418151.6	1.4%	83	91.1	9.86%	2.5	1873.2

A= Construct the Steiner tree without obstacles,
B= Construct the Steiner tree with obstacles,
C= The results computed by the formula 100×(B-A)/A