Efficient Obstacle-Avoiding Rectilinear Steiner Tree Construction *

Chung-Wei Lin[†], Szu-Yu Chen[‡], Chi-Feng Li[§], Yao-Wen Chang[†‡], and Chia-Lin Yang[§]
Graduate Institute of Electronics Engineering[†]
Department of Electrical Engineering[‡]
Department of Computer Science and Information Engineering[§]
National Taiwan University, Taipei 106, Taiwan
enorm@eda.ee.ntu.edu.tw, ywchang@cc.ee.ntu.edu.tw, yangc@csie.ntu.edu.tw

ABSTRACT

Given a set of pins and a set of obstacles on a plane, an obstacle-avoiding rectilinear Steiner minimal tree (OARSMT) connects these pins, possibly through some additional points (called Steiner points), and avoids running through any obstacle to construct a tree with a minimal total wirelength. The OARSMT problem becomes more important than ever for modern nanometer IC designs which need to consider numerous routing obstacles incurred from power networks, prerouted nets, IP blocks, feature patterns for manufacturability improvement, antenna jumpers for reliability enhancement, etc. Consequently, the OARSMT problem has received dramatically increasing attention recently. Nevertheless, considering obstacles significantly increases the problem complexity, and thus most previous works suffer from either poor quality or expensive running time. Based on the obstacle-avoiding spanning graph (OASG), this paper presents an efficient algorithm with some theoretical optimality guarantees for the OARSMT construction. Unlike previous heuristics, our algorithm guarantees to find an optimal OARSMT for any 2-pin net and many higher-pin nets. Extensive experiments show that our algorithm results in significantly shorter wirelengths than all state-of-the-art works.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids [Placement and Routing]

General Terms

Algorithms, Performance, Design

Keywords

Physical design, routing, Steiner tree, spanning tree

*This work was partially supported by National Science Council of Taiwan under Grant No's NSC 95-2221-E-002-372, NSC 95-2221-E-002-374, and NSC 95-2752-E-002-008-PAE.

ISPD'07, March 18–21, 2007, Austin, Texas, USA.

1. INTRODUCTION

Given a set of n pins and a set of obstacles on a plane, an obstacle-avoiding rectilinear Steiner minimal tree (OARSMT) connects these pins, possibly through some additional points (called Steiner points), and avoids running through any obstacle to construct a tree with a minimal total wirelength. The OARSMT problem becomes more important than ever for modern nanometer IC designs which need to consider numerous routing obstacles incurred from large-scale power networks, prerouted nets, IP blocks, feature patterns for manufacturability improvement, antenna jumpers for reliability enhancement, etc. Consequently, the OARSMT problem has received dramatically increasing attention recently [3, 7, 8, 11, 12].

The rectilinear Steiner minimal tree problem, even without obstacle consideration, is a well-known NP-complete problem [5]. The presence of obstacles further increases the complexity, and thus most previous works on the OARSMT problem suffer from either poor quality or expensive running time. As a fundamental problem with extensive practical applications to routing and wirelength/congestion/timing estimations in early IC design stages, such as floorplanning and the placement, it is desired to develop an effective and efficient algorithm for the OARSMT problem to facilitate the IC design flow.

Previous methods for the OARSMT problem can be classified into four major categories: (1) the maze-routing based approach, (2) the nondeterministic approach, (3) the construction-by-correction approach (called the sequential approach in [11]), and (4) the connection graph based approach. Maze routing, first proposed in [9], can optimally route 2-pin nets. However, its time complexity and memory usage grow prohibitively huge as the routing area becomes larger. Further, its multi-pin variants [6, 10] incur unsatisfiable solution quality since they are initially designed for 2-pin nets. As a result, the above drawbacks make the maze-routing based approach less popular for modern applications.

Based on ant colony optimization, Hu et al. [8] presented a nondeterministic local search heuristic to handle small-scale OARSMT problems with complex obstacles of both concave and convex polygons. Although this nondeterministic approach is flexible in handling complex obstacles, it incurs prohibitively expensive running time for large-scale designs.

The construction-by-correction approach constructs a Steiner or a spanning tree for a multi-pin net first and then replaces the edges overlapping obstacles with edges around the obstacles. This approach is popular in industry due to its simplicity and efficiency. However, the first step for the tree construction may not have the global view of the obstacles, and thus the second step might only remove the overlaps locally around the obstacles. As a result, the solution quality may be limited, as pointed out in [11]. Example works in the category include [13] and [3]. Yang *et al.* [13] presented a heuristic to remove the overlaps. Very

recently, Feng *et al.* [3] constructed an obstacle-avoiding Steiner tree for an arbitrary λ-geometry by Delaunay triangulation.

The last category is based on the connection graph. This approach is to first construct a connection graph by pins and obstacle boundaries, which guarantees at least a desired OARSMT is embedded in the graph. Then, some search techniques are applied to find the desired OARSMT from the connection graph. Unlike the construction-by-correction approach, this approach has a more global view of both pins and obstacles. Consequently, this approach can often obtain much better solution quality. Nevertheless, there exists a trade-off between effectiveness and efficiency in this approach; the larger size of the connection graph, the higher probability that a better OARSMT is embedded in the connection graph, but the more expensive running time.

Clarkson *et al.* [1] considered only 2-pin nets and presented an $O(n(\lg n)^2)$-time algorithm to compute a rectilinear shortest path between two pins through polygonal obstacles, where n is the number of pins and obstacle boundaries. Later, Ganley and Cohoon [4] presented an algorithm to find an optimal OARSMT with three or four pins, but its time complexity is $O(n^4)$. Hu *et al.* [7] developed an efficient hierarchical heuristic to partition all pins into subsets, then connect pins in each subset, and finally construct an OARSMT using a connection graph-like approach. Based on the spanning graph [14] that does not consider obstacles, Shen *et al.* [11] recently proposed a clever heuristic to construct an OARSMT. In this heuristic, an obstacle-avoiding spanning graph (OASG) was first constructed and then transformed into an OARSMT. The time complexity of the OASG construction is $O(n \lg n)$, and that of the OARSMT transformation is $\Omega(n^2 \lg n)$ though not analyzed or explicitly stated in [11]. This work [11] is effective in general, but we observe that it misses many "essential" edges which can lead to more desired solutions in the construction of the OASG, resulting in significant degradation in the solution quality for many practical cases. Further, its OARSMT transformation procedure could also be significantly improved.

In this paper, we construct an OASG with "essential" edges and prove the existence of a rectilinear shortest path between any two pins, which is not guaranteed in the OASG constructed by [11]. With this property, our algorithm guarantees to find an optimal OARSMT for any 2-pin net and many higher-pin nets. After constructing an initial OARSMT, we develop an effective refinement scheme for the U-shaped connection in the OARSMT to further reduce the total wirelength. Empirical results based on the least-square analysis show that our algorithm run in about $O(n^{1.46})$ time while the theoretical time complexity is $O(n^3)$.

Extensive experiments based on 22 test cases (5 industrial designs, 12 test cases from [3], and 5 larger random designs) show that our algorithm significantly outperforms all state-of-the-art works in the total wirelength and requires comparable running time to [11] for practical-sized problems. Considering the differences from the half-perimeter of the bounding box of all pins (which is a lower bound of the optimal OARSMT solution), the respective average improvements are 27.79%, 6.66%, and 5.79%, compared with the recent works [3], [12], and [11]. With the completeness of the OASG construction, in particular, our algorithm also provides key insights into the search for more desirable OARSMT solutions.

The rest of this paper is organized as follows. Section 2 formulates the OARSMT problem. Section 3 presents our OARSMT algorithm and its time complexity. Section 4 reports the experimental results. Finally, we conclude our work in Section 5.

2. PROBLEM FORMULATION

We define an *obstacle* and a *pin-vertex* as follows:

DEFINITION 1. *An* obstacle *is a rectangle on the xy-plane. No two obstacles overlap with each other, but two obstacles could be point-touched at the corner or line-touched at the boundary.*

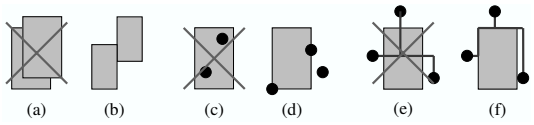

Figure 1: (a) Any two obstacles cannot overlap each other, but (b) two obstacles could be point-touched at the corner or line-touched at the boundary. (c) A pin-vertex may not locate inside any obstacle, but (d) it could be at the corner or on the boundary of an obstacle. (e) Any edge of the OARSMT cannot intersect any obstacle, but (f) it could be point-touched at the corner or line-touched on the boundary of an obstacle.

See Figure 1 (a) for two overlapped obstacles, and Figure 1 (b) for point-touched and line-touched obstacles.

DEFINITION 2. *A* pin-vertex *is a vertex on the xy-plane. A pin-vertex must not locate inside any obstacle, but it could be at the corner or on the boundary of an obstacle.*

See Figure 1 (c) for an illegal instance with two pin-vertices inside an obstacle, and Figure 1 (d) for a legal instance with a pin-vertex at the corner and another on the boundary of an obstacle.

Let $P = \{p_1, p_2, \ldots, p_m\}$ be a set of pin-vertices for an m-pin net, $O = \{o_1, o_2, \ldots, o_k\}$ be a set of k obstacles, and $V = \{v_1, v_2, \ldots, v_n\} = P \cup \{\text{corners in } O\}$ be the set of n vertices for the problem, where v_i has the coordinate (x_i, y_i). We have $n \leq m + 4k$ since each obstacle has four corners. The rectilinear (Manhattan) distance between v_i and v_j can be computed by $|x_i - x_j| + |y_i - y_j|$.

We consider rectilinear (vertical and horizontal) routes and define the *obstacle-avoiding rectilinear Steiner minimal tree* (OARSMT) problem as follows:

- **Problem: Obstacle-Avoiding Rectilinear Steiner Minimal Tree:** Given a set P of pins and a set O of obstacles on a plane, construct a rectilinear Steiner tree to connect the pins in P, possibly through some additional points (called Steiner points), such that no tree edge intersects an obstacle in O and the total wirelength of the tree is minimized.

Note that no edge of the OARSMT can intersect with any obstacle, but an edge could be point-touched at the corner or line-touched on the boundary of an obstacle. See Figure 1 (e) for a rectilinear Steiner tree intersecting an obstacle, and Figure 1 (f) for tree edges being line-touched on the boundary of an obstacle.

Throughout this paper, we represent the bottom-left, top-left, top-right, and bottom-right *corner-vertices* of an obstacle o_i by $c_{i,1}$, $c_{i,2}$, $c_{i,3}$, and $c_{i,4}$ with their coordinates being $(x_{i,\min}, y_{i,\min})$, $(x_{i,\min}, y_{i,\max})$, $(x_{i,\max}, y_{i,\max})$, and $(x_{i,\max}, y_{i,\min})$, respectively. Besides, $C = \bigcup_{i=1}^{k} \{c_{i,j}\}, j = 1, 2, 3, \text{ and } 4$.

3. ALGORITHM

We now present our algorithm. Our algorithm consists of the following four steps:

1. Obstacle-Avoiding Spanning Graph (OASG) Construction: In this step, an OASG connecting all vertices in $P \cup C$ is constructed. This step ensures that the following steps, except the operations in Section 3.4.3, can ignore the obstacles without violating the obstacle-avoiding property. See Figure 2(b) for an example OASG construction.
2. Obstacle-Avoiding Spanning Tree (OAST) Construction: An OAST connecting all pin-vertices is constructed by selecting edges from the OASG constructed in Step 1. See Figure 2(c) for an example OAST construction.

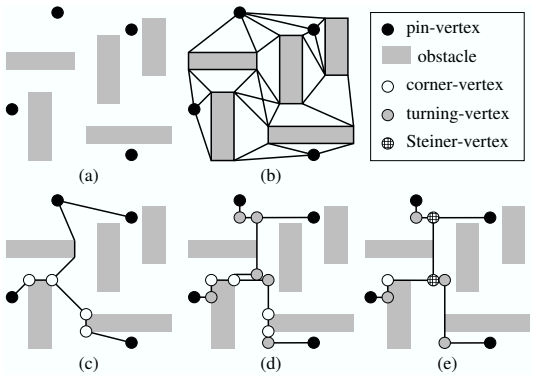

Figure 2: The four steps ((b)–(e)) for OARSMT construction.

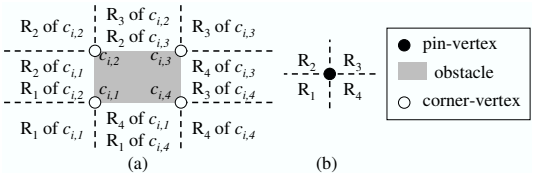

Figure 3: The divided regions for (a) each corner-vertex of an obstacle and (b) a pin-vertex.

3. Obstacle-Avoiding Rectilinear Spanning Tree (OARST) Construction: An OARST is constructed by transforming each slant edge of the OAST in Step 2 to rectilinear (vertical and horizontal) edges. See Figure 2(d) for an example OARST construction.
4. Obstacle-Avoiding Rectilinear Steiner Tree (OARSMT) Construction: Finally, an initial OARSMT is constructed by introducing Steiner points and removing overlapping edges of the OARST in Step 3. Then, a refinement scheme for some particular routing shapes is applied to find an OARSMT with a smaller total wirelength. See Figure 2(e) for an example OARSMT construction.

The following subsections detail the four steps.

3.1 OASG Construction

In this step, we construct an obstacle-avoiding spanning graph (OASG) which is defined as follows:

DEFINITION 3. *An* obstacle-avoiding spanning graph (OASG) *is an undirected connected graph on the vertex set $P \cup C$, where no edge intersects with an obstacle in O.*

We extend the spanning graph proposed by Zhou [14] to consider obstacles for the OASG construction. For each vertex in $P \cup C$, we divide the plane into four regions, R_1, R_2, R_3, and R_4, as shown in Figures 3 (a) and (b). The division is similar to that in [11], but we construct an OASG with more "essential" edges to improve the solution quality. As an example shown in Figure 4, our OASG contains the edge (p_1, p_2) (see Figure 4 (a)) while that in [11] does not (see Figure 4 (c)). After transforming them to rectilinear connections, we can obtain an optimal connection as shown in Figure 4 (b), while the work [11] results in a suboptimal solution as shown in Figure 4 (d).

As the example shown in Figure 5 with $r+1$ pin-vertices, each obstacle is of 2-unit high, and the edge (p_i, p_{i+1}), $1 \le i \le r$, is of 4-unit long. For this case, we can reduce the total wirelength by about 33% over the algorithm in [11] and obtain an optimal

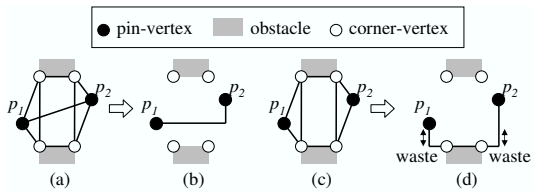

Figure 4: A comparison between our OASG and that of Shen *et al.* (a) Our OASG has the edge (p_1, p_2) and (b) results in an optimal rectilinear connection. (c) The OASG of Shen *et al.* does not contain the edge and (d) results in two wasted segments.

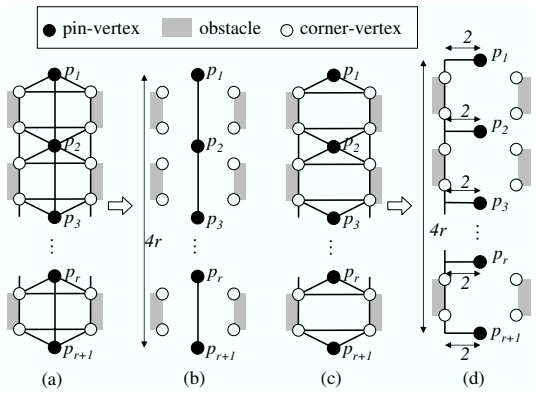

Figure 5: Another comparison between our OASG and that of Shen *et al.* (a) Our OASG has the edges (p_i, p_{i+1}), $1 \le i \le r$, and (b) results in an optimal rectilinear connection with the total wirelength of $4r$. (c) The OASG of Shen *et al.* does not contain these edges and (d) results in the connection with the total wirelength of $6r + 2$.

solution. In Figure 5 (a), our OASG contains the edges (p_i, p_{i+1}), $1 \le i \le r$, resulting in an optimal rectilinear connection with the total wirelength of $4r$, as shown in Figure 5 (b). However, the OASG constructed by [11] is illustrated in Figure 5 (c), which does not contain the edges (p_i, p_{i+1}), $1 \le i \le r$, resulting in the connection with the total wirelength of $6r + 2$, as shown in Figure 5 (d).

3.1.1 OASG Construction within a Region

For the OASG construction within a region, the *neighbors* of a vertex are defined as follows:

DEFINITION 4. *A vertex $f \in P \cup C$ is a* neighbor *of a vertex $v \in P \cup C$ if no other vertex in $P \cup C$ or obstacle is inside or on the boundary of the bounding box of v and f.*

As shown in Figure 6 (b), $c_{4,4}$, $c_{2,4}$, and $c_{5,4}$ are the neighbors of $c_{6,2}$, but p_2 is not because $c_{5,4}$ is on the boundary of the bounding box of $c_{6,2}$ and p_2. Our OASG construction is to construct edges between a vertex $v \in P \cup C$ and each of its neighbors. We will focus on R_2 of a vertex in $P \cup C$ for the discussion, while the other regions are similarly handled. Note that if the vertex is at the corner or on the boundary of an obstacle, it is clear that no edge will be constructed within the regions blocked by the obstacle.

The algorithm of the OASG construction for R_2 of a vertex is summarized in Figure 7. Figure 6 (a) shows an example to construct the OASG within the R_2 of $c_{6,2}$. After the initialization steps (lines 1–3), line sweeping is performed from left to

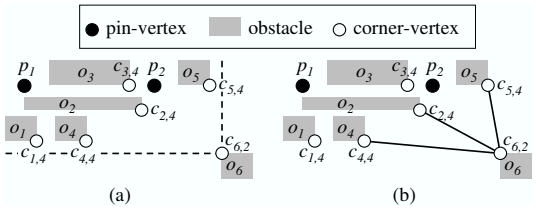

Figure 6: (a) An example instance and (b) the OASG construction for the vertex $c_{6,2}$ in R_2 of a vertex.

Algorithm: $OASG\text{-}R_2(O, P, v, E)$
Input: O /* the set of obstacles */
P /* the set of pin-vertices */
$v = (\overline{x}, \overline{y})$ /* OASG is for the R_2 of v */
Output: E /* edges added to OASG */
1 $E = \emptyset$
2 $A = \emptyset$ /* candidate set */
3 $I = \emptyset$ /* interval set as the blocking information */
4 Perform line sweeping from left to right
5 **if** it meets l left boundaries of obstacles, $o_{\alpha_1}, o_{\alpha_2}, \ldots, o_{\alpha_l}$
6 $I = I \cup \{[y_{\alpha_1,\min}, y_{\alpha_1,\max}], \ldots, [y_{\alpha_l,\min}, y_{\alpha_l,\max}]\}$
7 **if** it meets r right boundaries of obstacles, $o_{\beta_1}, o_{\beta_2}, \ldots, o_{\beta_r}$
8 $I = I \setminus \{[y_{\beta_1,\min}, y_{\beta_1,\max}], \ldots, [y_{\beta_r,\min}, y_{\beta_r,\max}]\}$
9 **for** $j = 1$ **to** l
10 **if** $c_{\alpha_j,1} \in R_2$ of v and $[\overline{y}, y_{\alpha_j,\min}]$ is not blocked by I
11 $A = A \cup \{c_{\alpha_j,1}\}$
12 **for** $j = 1$ **to** r
13 **if** $c_{\beta_j,4} \in R_2$ of v and $[\overline{y}, y_{\beta_j,\min}]$ is not blocked by I
14 $A = A \cup \{c_{\beta_j,4}\}$
15 **else if** $c_{\beta_j,3} \in R_2$ of v and $[\overline{y}, y_{\beta_j,\max}]$ is not blocked by I
16 $A = A \cup \{c_{\beta_j,3}\}$
17 **if** it meets i pin-vertices, $p_{\gamma_1}, p_{\gamma_2}, \ldots, p_{\gamma_i}$
18 **for** $j = 1$ **to** i
19 **if** $p_{\gamma_j} \in R_2$ of v and $[\overline{y}, y_{\gamma_j}]$ is not blocked by I
20 $A = A \cup \{p_{\gamma_j}\}$
21 **if** the sweeping line meets v
22 Go to line 23
23 Sort vertices in A in the non-decreasing y-coordinate order
(For vertices with the same y-coordinate,
sort them with the non-decreasing x-coordinate order.)
24 **for** each vertex $v' \in A$
25 **if** the vertex v' is a neighbor of v
26 $E = E \cup \{(v, v')\}$
27 Return E

Figure 7: The algorithm of the OASG construction for the R_2 of a vertex.

right. When the line meets the left boundary of o_1, the interval $[y_{1,\min}, y_{1,\max}]$ is inserted into the interval set I as the "blocking information" (lines 5–6). When the line meets the left boundary of o_2, the interval $[y_{2,\min}, y_{2,\max}]$ is also inserted into the interval set I as the "blocking information" (lines 5–6). At the same time, the sweeping line meets the pin-vertex p_1, but p_1 is not inserted into the candidate set A due to the intersection of the blocking information (lines 17–19). When the sweeping line meets the right boundary of o_1, $[y_{1,\min}, y_{1,\max}]$ is deleted from the interval set I (lines 7–8), and $c_{1,4}$ is inserted into the candidate set A (lines 12–14). Similarly, $c_{4,4}$, $c_{2,4}$ are inserted into the candidate set A (lines 12–14), while $c_{3,4}$ is not due to the intersection of the blocking information (lines 12–13). Then, when the sweeping line meets the pin-vertex p_2 and the right boundary of o_5, p_2 and $c_{5,4}$ are inserted into the candidate set A (lines 17–20 and lines 12–14). Therefore, when the sweeping line meets the left boundary of o_6, the sweeping line halts (lines 21–22), and the candidate set A is $\{c_{1,4}, c_{4,4}, c_{2,4}, p_2, c_{5,4}\}$. After the sorting (line 23), the candidate set A becomes $\{c_{1,4}, c_{4,4}, c_{2,4}, p_2, c_{5,4}\}$. Therefore, $c_{4,4}$, $c_{2,4}$, and $c_{5,4}$ can easily be detected as the neighbors of $c_{6,2}$ (lines 24–25). Finally, $(c_{4,4}, c_{6,2})$, $(c_{2,4}, c_{6,2})$, and

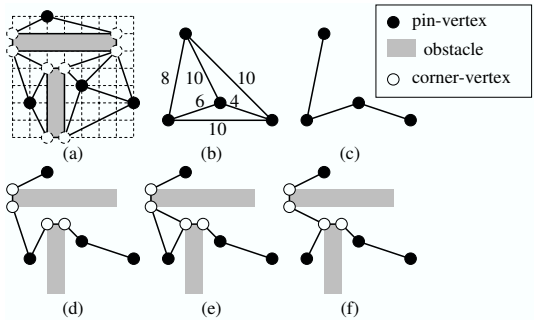

Figure 8: An example OAST construction.

$(c_{5,4}, c_{6,2})$ are inserted into the set E (line 26), and the OASG within the R_2 of $c_{6,2}$ is constructed as shown in Figure 6 (b).

3.1.2 Properties of Pin-Vertex Shortest Paths

We claim that the OASG implies a rectilinear shortest path of any two vertices in $P \cup C$, *i.e.*, a rectilinear shortest path of any two vertices can be obtained by transforming some edges in the OASG to rectilinear (vertical and horizontal) edges. Besides, each slant edge is transformed into only one vertical edge and one horizontal edge. We first define the *territory* of a vertex in $P \cup C$ as follows:

DEFINITION 5. *A vertex g on the xy-plane is in the* territory *of a vertex $v \in P \cup C$ if no other vertex in $P \cup C$ or obstacle is inside the bounding box of v and g.*

Note that the territory of a vertex is not necessarily a close region.

LEMMA 1. *Given a source $s \in P \cup C$, a target $t \in P \cup C$ ($s \neq t$), and any of their rectilinear shortest paths, $RSP(s,t)$, there must exist a neighbor f of s such that the rectilinear shortest length $\delta_r(s,t) = \delta_r(s,f) + \delta_r(f,t)$.*

LEMMA 2. *Given a vertex $v \in P \cup C$, for any neighbor f of v, there must exist an edge between v and f in the OASG,* i.e., *a rectilinear shortest path of v and f is implied by the OASG.*

Due to the limitation of space, we omit the proofs of Lemma 1, Lemma 2, and other theorems throughout this paper.

THEOREM 1. *The OASG implies a rectilinear shortest path of any two vertices in $P \cup C$.*

3.2 OAST Construction

We first define an *obstacle-avoiding spanning tree* (OAST) as follows:

DEFINITION 6. *An* obstacle-avoiding spanning tree (OAST) *is an undirected tree connecting all pin-vertices without intersecting with any obstacle.*

We construct an OAST by selecting some edges from the given OASG. As illustrated in Figure 8, the OAST construction consists of three steps: (1) pin-vertices shortest path computation, (2) initial OAST construction, and (3) local refinement.

3.2.1 Pin-Vertices Shortest Path Computation

For each edge in the given OASG, its length is defined as the Manhattan distance of its two end vertices. We apply Dijkstra's shortest-path algorithm [2] for each pin-vertex pair to compute their distance, as illustrated in Figure 8 (b).

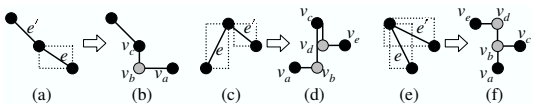

Figure 9: Three cases in the OARST construction for a slant edge and its neighboring edge. The graphs in (a), (c), and (e) are transformed into those in (b), (d), and (f), respectively.

Algorithm: $OARST(E_i, V_o, E_o)$
Input: E_i /* the edge set of OAST */
Output: V_o /* the vertex set of OARST */
E_o /* the edge set of OARST */
1 $V_o = \emptyset$
2 $E_o = \emptyset$
3 $A = E_i$ /* unprocessed edge set */
4 **while** $A \neq \emptyset$
5 Select the longest edge e in A /* $e = (v, v')$ */
6 **if** e is a vertical edge or a horizontal edge
7 $V_o = V_o \cup \{v, v'\}$
8 $E_o = E_o \cup \{e\}$
9 **else**
10 Select a neighboring edge e' of e with longest sharing length
11 **if** $e' = NULL$ or the relation of e and e' is Case 1
/* $e' = NULL$ if e has no neighboring edge */
/* Case 1, Figures 9 (a) and (b) */
12 $A = A \setminus \{e\}$
13 Decide v_a, v_b, and v_c
14 $V_o = V_o \cup \{v_a, v_b, v_c\}$
15 $E_o = E_o \cup \{(v_a, v_b), (v_b, v_c)\}$
16 **else if** the relation of e and e' is Case 2
/* Case 2, Figure 9 (c) and (d) */
17 $A = A \setminus \{e, e'\}$
18 Decide v_a, v_b, v_c, v_d, and v_e
19 $V_o = V_o \cup \{v_a, v_b, v_c, v_d, v_e\}$
20 $E_o = E_o \cup \{(v_a, v_b), (v_b, v_c), (v_c, v_d), (v_d, v_e)\}$
21 **else** /* Case 3, Figures 9 (e) and (f) */
22 $A = A \setminus \{e, e'\}$
23 Decide v_a, v_b, v_c, v_d, and v_e
24 $V_o = V_o \cup \{v_a, v_b, v_c, v_d, v_e\}$
25 $E_o = E_o \cup \{(v_a, v_b), (v_b, v_c), (v_c, v_d), (v_d, v_e)\}$
26 Return (V_o, E_o)

Figure 10: The algorithm for the OARST construction.

3.2.2 Initial OAST Construction

We then construct a complete graph for the $|P|$ pin-vertices. The edge weight is defined as the distance of its two end vertices computed in Section 3.2.1. We then apply Prim's algorithm [2] on the complete graph to obtain a minimum spanning tree (see Figure 8 (c)). By the shortest paths computed in Section 3.2.1, we can map each edge in the minimum spanning tree to a shortest path in the spanning graph, so the initial spanning tree on the spanning graph is constructed (see Figure 8 (d)). It should be noted that shortest paths may share a common edge. In such a case, the initial spanning tree on the spanning graph will count it only once.

3.2.3 Local Refinement

In the initial OAST, there could be some pairs of vertices whose corresponding edges are in the OASG, but not in the initial OAST. We add such edges into the OAST (see Figure 8 (e)) and compute the minimum spanning tree on it to remove unwanted cycles (see Figure 8 (f)). This local refinement may lead to a new OAST with a smaller total wire length.

3.3 OARST Construction

In this step, we transform each slant edge of the given OAST into vertical and horizontal edges to obtain an *obstacle-avoiding rectilinear spanning tree* (OARST).

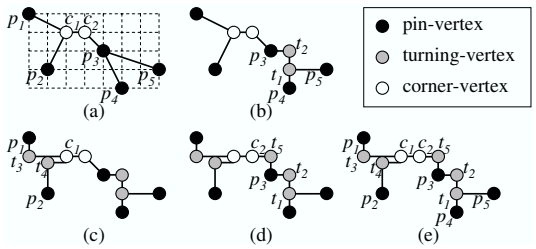

Figure 11: An example OARST construction.

DEFINITION 7. *An* obstacle-avoiding rectilinear spanning tree (OARST) *is an undirected graph connecting all pin-vertices with vertical and horizontal edges.*

We then define a *neighboring* edge and its *sharing length* in an OAST as follows:

DEFINITION 8. *A* neighboring edge *of an edge e is an edge which has a common end vertex with e.*

DEFINITION 9. *The* sharing length *of two edges e_1 and e_2 is the summation of the overlapping lengths when e_1 and e_2 are projected to the x- and the y-axes.*

Three cases in the OARST construction for a slant edge e and its neighboring edge e' need to be considered, in which we take the common vertex as the origin on the xy-plane:

Case 1. The two edges are in opposite regions (see Figure 9 (a)). In this case, e is transformed into a vertical edge and a horizontal edge (see Figure 9 (b)). There are two possible transformations, so we randomly choose one.

Case 2. The two edges are in neighboring regions (see Figure 9 (c)). In this case, both e and e' are transformed into a vertical edge and a horizontal edge. There are several possible transformations, so we choose the one with edge overlap (see Figure 9 (d)).

Case 3. The two edges are in the same region (see Figure 9 (e)). In this case, using Figure 9 (f) as an example, e and e' are transformed into (v_a, v_b) and (v_b, v_c), respectively . There are two possible transformations for (v_c, v_e), and we randomly choose one.

Figure 10 summarizes the algorithm for the OARST construction. We use the example shown in Figure 11 (a) to explain the process. After the initialization steps (lines 1–3), the unprocessed edge set A is $\{(p_1, c_1), (p_2, c_1), (c_1, c_2), (c_2, p_3), (p_3, p_4), (p_3, p_5)\}$ as shown in Figure 11 (a), and the set E_o is $\emptyset$. In the first iteration, (p_3, p_5) is selected as e (line 5), and (p_3, p_4) is selected as e' (line 10). Then, Case 3 (see Figure 9 (e)) is applied, and they are transformed into (t_1, p_4), (t_1, p_5), (t_1, t_2), and (t_2, p_3) as shown in Figure 11 (b) (lines 21–25). After the first iteration, the unprocessed edge set A is $\{(p_1, c_1), (p_2, c_1), (c_1, c_2), (c_2, p_3)\}$, and the set E_o is $\{(t_1, p_4), (t_1, p_5), (t_1, t_2), (t_2, p_3)\}$. In the second iteration, (p_1, c_1) is selected as e (line 5), and (p_2, c_1) is selected as e' (line 10). Then, Case 2 is applied (see Figure 9 (c)); (p_1, c_1) is transformed into (p_1, t_3) and (t_3, c_1), and (p_2, c_1) is transformed into (p_2, t_4) and (t_4, c_1) (see Figure 11 (c)) (lines 16–20). After the second iteration, the unprocessed edge set A is $\{(c_1, c_2), (c_2, p_3)\}$, and the set E_o is $\{(t_1, p_4), (t_1, p_5), (t_1, t_2), (t_2, p_3), (p_1, t_3), (t_3, c_1), (p_2, t_4), (t_4, c_1)\}$. In the third iteration, (c_2, p_3) is selected as e (line 5), and (c_1, c_2) is selected as e' (line 10). Then, Case 1 is applied (see Figure 9 (a)), and (c_2, p_3) is transformed into (c_2, t_5) and (t_5, p_3) (see Figure 11 (d)) (lines 11–15). After the third iteration, the unprocessed edge set A is $\{(c_1, c_2)\}$, and the set E_o is $\{(t_1, p_4), (t_1, p_5), (t_1, t_2), (t_2, p_3), (p_1, t_3),$

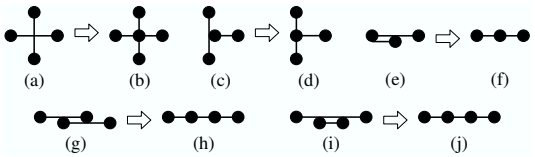

Figure 12: Five cases of the overlapping edge removal. The graphs in (a), (c), (e), (g), and (i) are transformed into those in (b), (d), (f), (h), and (j), respectively.

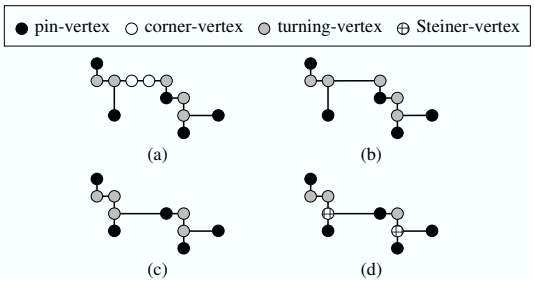

Figure 13: The OARSMT construction of Figure 11 (e).

$(t_3, c_1), (p_2, t_4), (t_4, c_1), (c_2, t_5), (t_5, p_3)\}$. In the fourth iteration, (c_1, c_2) is selected as e (line 5). Since (c_1, c_2) is a horizontal edge, it is transformed into (c_1, c_2) directly (lines 6–8). After the fourth iteration, the unprocessed edge set A is $\emptyset$, and the set E_o is $\{(t_1, p_4), (t_1, p_5), (t_1, t_2), (t_2, p_3), (p_1, t_3), (t_3, c_1), (p_2, t_4), (t_4, c_1), (c_2, t_5), (t_5, p_3), (c_1, c_2)\}$. Finally, the OARST is constructed as shown in Figure 11 (e).

3.4 OARSMT Construction

In this step, we construct an obstacle-avoiding rectilinear Steiner tree (OARSMT). The construction consists of three steps: (1) overlapping edge removal, (2) redundant vertex removal, and (3) U-shaped pattern refinement.

3.4.1 Overlapping Edge Removal

For each pair of edges in the OARST, we classify their relation into five cases as shown in Figure 12 (a), (c), (e), (g), and (i), and then transform them into those in Figure 12 (b), (d), (f), (h), and (j), respectively. Using Figure 11 (e) as an example, the result after overlapping edge removal is shown in Figure 13 (a).

3.4.2 Redundant Vertex Removal

A *redundant-vertex* is defined as follows:

DEFINITION 10. *A* redundant-vertex *is a non-pin-vertex with the degree of 2, and the two edges connecting to it are parallel.*

For a redundant-vertex, we merge the two edges connecting to it. Using Figure 13 (a) as an example, two vertices are removed as shown in the Figure 13 (b).

3.4.3 U-Shaped Pattern Refinement

The total wirelength can be further improved by some local refinements. Considering the trade-off between solution quality and efficiency, we especially refine U-shaped patterns. The *U-shaped pattern refinement rules* are defined as follows:

DEFINITION 11. *A vertex satisfies the* U-shaped pattern refinement rules *if it is not a pin-vertex, and its degree is 2.*

We need to consider two cases for the U-shaped pattern refinement:

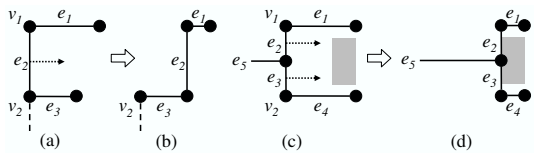

Figure 14: Two cases of the U-shaped pattern refinement. The graphs in (a) and (c) are transformed into those in (b) and (d) respectively.

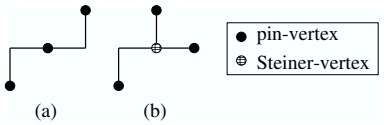

Figure 15: When $m = 3$, a rectilinear Steiner tree is one of the two topologies: (a) two simple paths between pin-vertices, or (b) three pin-vertices connected to a single Steiner-vertex.

Case 1. Several edges form the shape as shown in Figure 14 (a). One of the vertices v_1 and v_2 must satisfy the refinement rule. In this case, without intersecting any obstacle, the edge e_2 is moved as right as possible, while edges e_1 and e_3 are still connected by it. Edges connected to a vertex satisfying the refinement rule (e_1 in Figure 14 (a)) are shortened. The resulting refinement is shown in Figure 14 (b).

Case 2. Several edges form the shape as shown in Figure 14 (c). Both vertices v_1 and v_2 must satisfy the refinement rules. In this case, without intersecting any obstacle, the edges e_2 and e_3 are moved as right as possible, while edges e_1 and e_4 are still connected by them. The edge e_5 is stretched, but the two edges connected to a vertex satisfying the refinement rule (e_1 and e_4 in Figure 14 (c)) are shortened. The resulting refinement is shown in Figure 14 (d).

After the U-shaped pattern refinement, the redundant vertex removal is applied to ensure that there is no redundant-vertex in the OARSMT. Using Figure 13 (b) as an example, the resulting removal is shown in Figure 13 (c).

A *Steiner-vertex* is a vertex which is not a pin-vertex, and its degree is more than 2. We also mark Steiner-vertices. As an example shown in Figure 13 (c), two Steiner-vertices are marked (see Figure 13 (d)).

THEOREM 2. *The overall time complexity of our algorithm is $O(n^3)$ in the worst case and $O(n^2 \lg n)$ for practical applications.*

Note that n is the total number of pin-vertices and corner-vertices.

3.4.4 Optimality

We can construct an optimal OARSMT when the pin number $m = 2$. Even for nets with $m \geq 3$, our algorithm can still achieve optimal solutions in many cases. In the following, we give theorems for the optimality of our algorithm. Note that these theorems give the sufficient but not necessary conditions for an optimal solution, *i.e.*, more optimal solutions may still be generated in other cases. Besides, the U-shaped pattern refinement is not necessary for these theorems, implying that our OASG is indeed complete to generate these optimal solutions.

THEOREM 3. *If $m = 2$, our constructed OARSMT is an optimal solution.*

When $m = 3$, a rectilinear Steiner tree is one of the two topologies: two simple paths between pin-vertices as shown in Figure 15

Test Cases	m	k	HPBB (A)	Total Edge-Length				Improvement (%) ($\frac{X-E}{X}$ / $\frac{X-E}{X-A}$)		
				[12] (B)	[3] (C)	[11] (D)	Ours (E)	[12] ($X = B$)	[3] ($X = C$)	[11] ($X = D$)
ind1	10	32	501	—	—	646	632	—	—	2.17 / 9.66
ind2	10	43	8,200	—	—	10,100	9,600	—	—	4.95 / 26.32
ind3	10	50	498	—	—	623	613	—	—	1.61 / 8.00
ind4	25	79	705	—	—	1,121	1,121	—	—	0.00 / 0.00
ind5	33	71	732	—	—	1,392	1,364	—	—	2.01 / 4.24
rc1	10	10	17,890	26,970	30,410	27,730	26,900	0.26 / 0.77	11.54 / 28.04	2.99 / 8.43
rc2	30	10	19,470	41,700	45,640	42,840	42,210	-1.22 / -2.29	7.52 / 13.11	1.47 / 2.70
rc3	50	10	19,380	62,380	58,570	56,440	55,750	10.63 / 15.42	4.81 / 7.20	1.22 / 1.86
rc4	70	10	19,850	66,560	63,340	60,840	60,350	9.33 / 13.29	4.72 / 6.88	0.81 / 1.20
rc5	100	10	19,600	80,010	83,150	76,970	76,330	4.60 / 6.09	8.20 / 10.73	0.83 / 1.12
rc6	100	500	19,593	—	149,750	86,403	83,365	—	44.33 / 51.00	3.52 / 4.55
rc7	200	500	19,882	—	181,470	117,427	113,260	—	37.59 / 42.21	3.55 / 4.27
rc8	200	800	19,803	—	202,741	123,366	118,747	—	41.43 / 45.91	3.74 / 4.46
rc9	200	1,000	19,964	—	214,850	119,744	116,168	—	45.93 / 50.64	2.99 / 3.58
rc10	500	100	19,900	—	198,010	171,450	170,690	—	13.80 / 15.34	0.44 / 0.50
rc11	1,000	100	19,984	—	250,570	238,111	236,615	—	5.57 / 6.05	0.63 / 0.69
rc12	1,000	10,000	65,422	—	1,723,990	843,529	789,097	—	54.23 / 56.37	6.45 / 7.00
rt1	10	500	1,363	—	—	2,438	2,267	—	—	7.01 / 15.91
rt2	50	500	16,280	—	—	51,981	48,441	—	—	6.82 / 9.92
rt3	100	500	1,996	—	—	8,783	8,368	—	—	4.73 / 6.11
rt4	100	1,000	1,985	—	—	10,619	10,306	—	—	2.95 / 3.63
rt5	200	2,000	8,097	—	—	55,557	53,993	—	—	2.82 / 3.30
Average	—	—	—	—	—	—	—	4.72 / 6.66	23.31 / 27.79	2.89 / 5.79

Table 1: The comparison on the total edge-length, where "HPBB" is the half-perimeter of the bounding box of all pin-vertices, and "—" means that the result is not available. The results before "/" are the improvements on the total edge-length, while those after "/" are the improvements on the difference from the half-perimeter of the bounding box of all pin-vertices.

(a), or three pin-vertices connected to a single Steiner-vertex as shown in Figure 15 (b). We can construct an optimal OARSMT for the first topology.

THEOREM 4. *If $m = 3$ and the topology of an optimal solution contains two simple paths between pin-vertices, our constructed OARSMT is an optimal solution.*

Note that none of the aforementioned properties is guaranteed by the algorithm in [11] due to the missing "essential" edges, so [11] cannot guarantee optimal solutions even for $m = 2$, as illustrated in Figure 4. Also, most nets in a real case are 2-pin nets or 3-pin nets, which makes the above properties more important for practical applications. Furthermore, regardless of the topology, we can construct an optimal OARSMT for a 3-pin net if there is no obstacle.

THEOREM 5. *If $m = 3$ and there is no obstacle, our constructed OARSMT is an optimal solution.*

When $m \geq 4$, we can also construct an optimal OARSMT which contains only simple paths between pin-vertices.

THEOREM 6. *If $m \geq 4$ and the topology of an optimal solution contains only simple paths between pin-vertices, our constructed OARSMT is an optimal solution.*

Similarly, this property is not guaranteed by the algorithm in [11].

4. EXPERIMENTAL RESULTS

We implemented our algorithm in the C/C++ language on a 2 Ghz AMD-64 machine with 8 GB memory under Ubuntu 6.06 operating system. There are totally 22 benchmark circuits, five industrial test cases (ind1–ind5) from Synopsys, twelve test cases used in [3] (rc1–rc12), and five random test cases (rt1–rt5) generated by us. We removed an overlap of two obstacles in rc12 because it is invalid. On the other hand, the number of obstacles is usually much larger than that of pin-vertices in a real design, so we set the ratios of k and m to 5, 10, and 50 to generate the five large random cases. Given the constraints on the areas and the aspect ratios of obstacles, their positions, lengths, and widths were randomly generated without overlapping each other. Besides, the positions of pin-vertices were also randomly generated without locating inside any obstacle.

We compared our algorithm with those presented in [12], [3], and [11]. The results of [12] are provided by the authors, and were generated from a Unix workstation with 2.66 GHz CPU and 1 GB memory. The results of [3] are directly quoted from the paper, where the algorithm was performed on a Sun V880 fire workstation with 755 MHz CPU and 4 GB memory. We also implemented the algorithm in [11]. Different from our OASG graph construction, it only constructs an edge within each region. In addition, it operates without the U-shaped pattern refinement as described in Section 3.4.3. We also verified the generated OARSMTs by another program to ensure that all pin-vertices were connected without intersecting any obstacle.

Table 1 lists the total wirelengths of these algorithms without any scaling. Considering the differences from the half-perimeter of the bounding box of all pin-vertices, the respective average improvements on the total wirelength are 6.66%, 27.70%, and 5.79%, compared with the algorithms in [12], [3], and [11]. Furthermore, the improvement over [11] can be up to 26.32% (for ind2). Since the half-perimeter of the bounding box of all pin-vertices is a lower bound for an optimal solution for this OARSMT problem, these improvements are very significant. (If we consider the differences from an optimal solution, the improvement is even larger.) In larger test cases, since the half-perimeters of these cases are far from their optimal solutions, the improvements seem to be less than those of small cases. In fact, considering the percentages of the reduced length, the algorithm is still very effective, independent of the sizes of test cases. Figure 16 shows the resulting layout for the test case rt3.

Table 2 compares the CPU times of these algorithms. Our algorithm is sufficiently efficient. For example, when the numbers of pin-vertices and obstacles reach 200 and 1,000 respectively (rc9), our algorithm takes only 0.91 seconds and achieves 3.58% improvement over the algorithm in [11]. As shown in Figure 17,

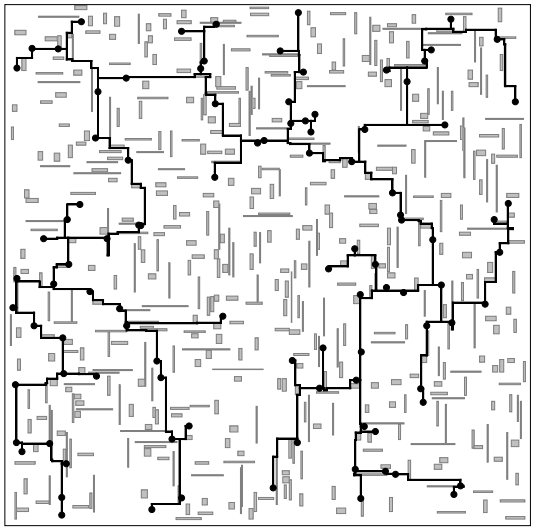

Figure 16: The final routing result of rt3, where a pin-vertex is represented by a solid circle.

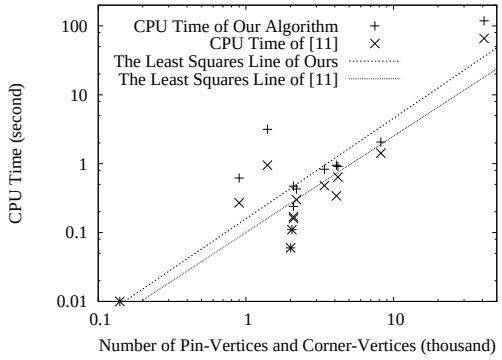

Figure 17: The CPU time is plotted as a function of n.

the CPU times of [11] and ours are plotted as functions of the input size n. By the least squares fitting on the log-log-axes, the respective slopes of the fitting lines are 1.40 and 1.46, implying that the empirical time complexity of our algorithm is close to $O(n^{1.46})$ while that of [11] is about $O(n^{1.40})$. Note that this is reasonable since we add more edges into our OASGs to guarantee the optimality described in Section 3.4.4, while the work [11] does not. Further, the empirical time complexity is far under the theoretical worst-case complexity of $O(n^3)$ in Theorem 2. The much lower empirical time complexity can be explained by the sizes of our OASGs. The numbers of edges in our OASGs are listed in the last column of Table 2. By the least squares fitting on the log-log function of the number of edges to the circuit size, the number of edges in our OASG grows only about $O(n^{1.03})$ empirically in the input size n, which is far under the theoretical worst-case complexity of $O(n^2)$. The experimental results show that our algorithm is very effective and efficient.

5. CONCLUSIONS

We have proposed an algorithm to construct an obstacle-avoiding rectilinear Steiner tree (OARSMT). We can achieve an optimal solution for any 2-pin net and nets with more pins in many cases. Experimental results have shown that our algorithm is very effective and efficient. With the completeness of the OASG construction, in particular, our algorithm also provides key insights into the search for more desirable OARSMT solutions.

Test Cases	CPU Time (second) [12]	[3]	[11]	Ours	# Edges in our OASG
ind1	—	—	< 0.01	< 0.01	437
ind2	—	—	< 0.01	< 0.01	798
ind3	—	—	< 0.01	< 0.01	903
ind4	—	—	< 0.01	< 0.01	488
ind5	—	—	< 0.01	0.01	346
rc1	0.49	< 0.01	< 0.01	< 0.01	125
rc2	1.03	< 0.01	< 0.01	< 0.01	205
rc3	8.79	< 0.01	< 0.01	< 0.01	314
rc4	67.62	< 0.01	< 0.01	< 0.01	427
rc5	595.10	< 0.01	0.01	0.01	574
rc6	—	0.06	0.17	0.24	6,582
rc7	—	0.06	0.30	0.43	7,299
rc8	—	0.10	0.48	0.83	10,332
rc9	—	0.13	0.64	0.91	12,505
rc10	—	0.03	0.27	0.62	4,445
rc11	—	0.04	0.95	3.15	10,546
rc12	—	2.82	65.73	118.52	204,091
rt1	—	—	0.06	0.06	5,358
rt2	—	—	0.11	0.11	6,244
rt3	—	—	0.16	0.47	6,447
rt4	—	—	0.34	0.95	10,832
rt5	—	—	1.42	2.06	21,938

Table 2: The comparison on the CPU time, where "—" means that the result is not available.

6. REFERENCES

[1] K. L. Clarkson, S. Kapoor and P. M. Vaidya, "Rectilinear shortest paths through polygonal obstacles in $O(nlog^{3/2}n)$ time," *Proc. SCG*, pp. 251–257, 1987.

[2] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, 2^{nd} edition, The MIT Press, 2001.

[3] Z. Feng, Y. Hu, T. Jing, X. Hong, X. Hu, and G. Yan, "An $O(nlogn)$ algorithm for obstacle-avoiding routing tree construction in the lambda-geometry plane," *Proc. ISPD*, pp. 48–55, 2006.

[4] J. Ganley and J. P. Cohoon, "Routing a multi-terminal critical net: Steiner tree construction in the presence of obstacles," *Proc. ISCAS*, vol. 1, pp. 113–116, 1994.

[5] M. Garey and D. Johnson, "The rectilinear Steiner tree problem in NP-Complete," *SIAM Journal of Applied Mathematics*, pp. 826–834, 1977.

[6] D. W. Hightower, "A solution to the line routing problem on the continous plane," *Proc. of the 6th Design Automation Workshop*, pp. 1–24, 1969.

[7] Y. Hu, Z. Feng, T. Jing, X. Hong, Y. Yang, G. Yu, X. Hu, and G. Yan, "FORst: a 3-step heuristic for obstacle-avoiding rectilinear Steiner minimal tree construction," *Journal of Information and Computational Science*, pp. 107–116, 2004.

[8] Y. Hu, T. Jing, X. Hong, Z. Feng, X. Hu, and G. Yan, "An-OARSMan: obstacle-avoiding routing tree Construction with good length performance," *Proc. ASP-DAC*, pp. 7–12, 2005.

[9] C. Y. Lee, "An algorithm for connections and its application," *IRE Trans. on Electronic Computer*, pp. 346–365, 1961.

[10] K. Mikami and K. Tabuchi, "A computer program for optimal routing of printed circuit connectors," *Proc. IFIPS*, pp. 1475–1478, 1968, H47.

[11] Z. Shen, C. Chu, and Y. Li, "Efficient rectilinear Steiner tree construction with rectilinear blockages," *Proc. ICCD*, pp. 38–44, 2005.

[12] Y. Shi, T. Jing, L. He, and Z. Feng, "CDCTree: novel obstacle-avoiding routing tree construction based on current driven circuit model," *Proc. ASP-DAC*, pp. 630–635, 2006.

[13] Y. Yang, Q. Zhu, T. Jing, X. Hong, and Y. Wang, "Rectilinear Steiner minimal tree among obstacles," *Proc. ASIC*, pp. 348–351, 2003.

[14] H. Zhou, "Efficient Steiner tree construction based on spanning graphs," *IEEE Trans. Computer-Aided Design*, Vol. 23, No. 5, pp. 704-710, May 2004.