Obstacle-Avoiding Rectilinear Steiner Tree Construction Based on Spanning Graphs

Chung-Wei Lin, Szu-Yu Chen, Chi-Feng Li, Yao-Wen Chang, *Member, IEEE*, and Chia-Lin Yang, *Member, IEEE*

***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 the 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, 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 two-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.**

***Index Terms*—Physical design, routing, spanning tree, Steiner tree.**

I. Introduction

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 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 [5], [8], [9], [12], [13].

The rectilinear Steiner minimal tree problem, even without obstacle consideration, is a well-known NP-complete problem [7]. 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. The OARSMT problem is a fundamental problem with extensive practical applications to routing and wirelength/congestion/timing estimations in early IC design stages, such as floor planning and the placement. In particular, the application to congestion-driven routing has attracted much attention recently, where over congested regions can be treated as obstacles during routing. Therefore, 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 [12]); and 4) the connection-graph-based approach. Maze routing, which was first proposed in [10], can optimally route two-pin nets. However, its time complexity and memory usage grow prohibitively huge as the routing area becomes larger; therefore, Clow [2] presented the A$*$ maze routing to reduce the time complexity and memory usage. Due to its simplicity, many global routers are based on Clow's maze routing.

Based on ant-colony optimization, Hu *et al.* [9] 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 multipin net first and then replaces the edges overlapping obstacles with edges around the obstacles. This approach is popular in industries 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 [12]. In particular, when there are more/larger obstacles, this approach is less favorable because more/larger corrections need to be fixed. Example works in the category include those in [14] and [5]. Yang *et al.* [14] presented a heuristic to remove the overlaps. Very recently, Feng *et al.* [5] constructed an obstacle-avoiding Steiner tree for an arbitrary λ-geometry by Delaunay triangulation.

Manuscript received May 28, 2007; revised August 29, 2007. This work was supported in part by the National Science Council of Taiwan under Grants NSC 95-2221-E-002-372, NSC 95-2221-E-002-374, and NSC 95-2752-E-002-008-PAE. This paper was recommended by Associate Editor P. H. Madden.

C.-W. Lin is with the Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 106, Taiwan, R.O.C. (e-mail: enorm@eda.ee.ntu.edu.tw).

S.-Y. Chen was with the Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan, R.O.C. He is now with the Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 106, Taiwan, R.O.C. (e-mail: aknow@eda.ee.ntu.edu.tw).

C.-F. Li and C.-L. Yang are with the Department of Computer Science and Information Engineering, National Taiwan University, Taipei 106, Taiwan, R.O.C. (e-mail: akilae@eda.ee.ntu.edu.tw; yangc@csie.ntu.edu.tw).

Y.-W. Chang is with the Department of Electrical Engineering and Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 106, Taiwan, R.O.C. (e-mail: ywchang@cc.ee.ntu.edu.tw).

Digital Object Identifier 10.1109/TCAD.2008.917583

The last category is based on the connection graph. This approach first constructs a connection graph by pins and obstacle boundaries, which guarantees that 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 a much better solution quality. Nevertheless, there exists a tradeoff 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 is the running time.

Clarkson *et al.* [1] considered only two-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, Zheng *et al.* [15] introduced an implicit connection graph containing rectilinear edges and found obstacle-avoiding shortest paths. It can be used to solve the one-to-one shortest path problem, the one-to-many shortest paths problem, and the minimum spanning tree problem in the presence of obstacles, where its time complexity is $O(n^2 \lg n)$. On the other hand, Ganley and Cohoon [6] presented an algorithm to find an optimal OARSMT with three or four pins, but its time complexity is $O(n^4)$. Hu *et al.* [8] 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 graphlike approach. Based on the spanning graph [16] that does not consider obstacles, Shen *et al.* [12] 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 $\Omega(n^2 \lg n)$ although not analyzed or explicitly stated in [12]. This work [12] 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. Furthermore, 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 Shen *et al.* [12]. With this property, our algorithm guarantees to find an optimal OARSMT for any two-pin net and many higher pin nets. Moreover, we prove that the expected number of edges in our OASG is only $O(n \lg n)$, ensuring an efficient searching or processing on it. 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 squares analysis show that our algorithm run in about $O(n^{1.46})$ time, whereas the theoretical time complexity is $O(n^3)$ in the worst case and $O(n^2 \lg n)$ in a random case.

Extensive experiments based on 22 test cases (five industrial designs, 12 test cases from [5], and five 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 the algorithm in [12] 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%, 5.79%, and 0.93%, compared with the recent works [5], [12], [13], and an extension of Clow's A$*$ maze-routing algorithm. The empirical time complexity of the extension of Clow's A$*$ maze-routing algorithm is about $O(n^{1.59})$, whereas ours is about $O(n^{1.46})$. With the completeness of the OASG construction, in particular, our algorithm also provides key insights into the search for more desirable OARSMT solutions.

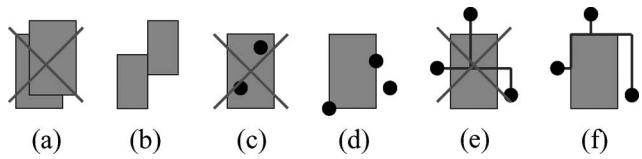


Fig. 1. (a) Any two obstacles cannot overlap each other, but (b) two obstacles could be point-touched at the corner or line-touched at the boundary. (c) A pin vertex may not locate inside any obstacle, but (d) it could be at the corner or on the boundary of an obstacle. (e) Any edge of the OARSMT cannot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

The rest of this paper is organized as follows. Section II formulates the OARSMT problem. Section III presents our OARSMT algorithm and its time complexity. Section IV reports the experimental results. Finally, we conclude this paper in Section V.

II. 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 [see Fig. 1(a) for two overlapped obstacles and Fig. 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 Fig. 1(c) for an illegal instance with two pin vertices inside an obstacle and Fig. 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 n be the size of $P \cup \{\text{corners in } O\}$. 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 OARSMT problem as follows.

Problem: OARSMT

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 the Steiner points), such that no tree edge intersects an obstacle in O and that the total wirelength of the tree is minimized.

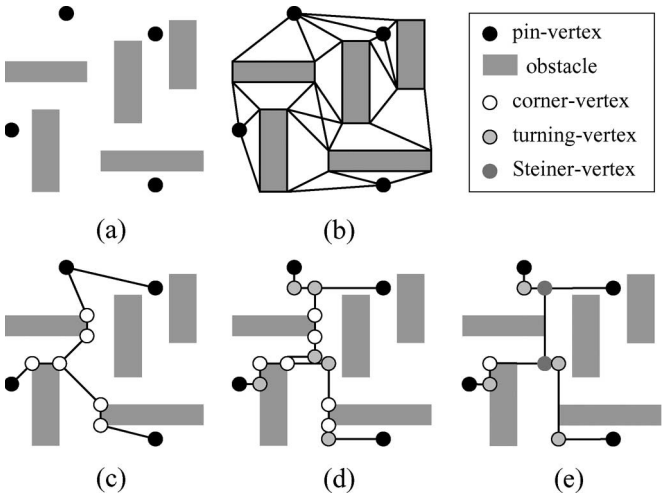


Fig. 2. (b)–(e) Four steps for OARSMT construction.

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 Fig. 1(e) for a rectilinear Steiner tree intersecting an obstacle and Fig. 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. Moreover, $C = \bigcup_{i=1}^{k}\{c_{i,j}\}$, where $j = 1, 2, 3$, and 4.

III. Algorithm

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

1) 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 III-D3, can ignore the obstacles without violating the obstacle-avoiding property [see Fig. 2(b) for an example of 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 Fig. 2(c) for an example of OAST construction].
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 Fig. 2(d) for an example of 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 Fig. 2(e) for an example of OARSMT construction].

The following sections detail the four steps.

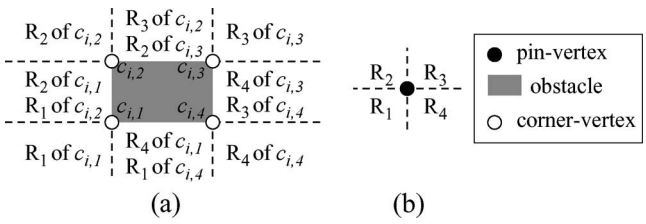


Fig. 3. Divided regions for (a) each corner vertex of an obstacle and (b) a pin vertex.

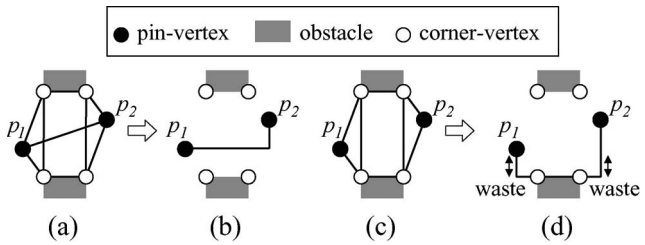


Fig. 4. Comparison between our OASG and that of Shen *et al.* (a) Our OASG has the edge (p_1, p_2) and (b) results in an optimal rectilinear connection. (c) The OASG of Shen *et al.* does not contain the edge and (d) results in two wasted segments.

A. OASG Construction

In this step, we construct an OASG which is defined as follows.

Definition 3: An 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 [16] 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 Fig. 3(a) and (b). The division is similar to that in [12], but we construct an OASG with more "essential" edges to improve the solution quality. As an example shown in Fig. 4, our OASG contains the edge (p_1, p_2) [see Fig. 4(a)], whereas that in [12] does not [see Fig. 4(c)]. After transforming them to rectilinear connections, we can obtain an optimal connection as shown in Fig. 4(b), whereas the work in [12] results in a suboptimal solution, as shown in Fig. 4(d).

In the example shown in Fig. 5 with $r + 1$ pin vertices, each obstacle is of two-unit high, and the edge (p_i, p_{i+1}), $1 \le i \le r$, is of four-unit long. For this case, we can reduce the total wirelength by about 33% over the algorithm in [12] and obtain an optimal solution. In Fig. 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 Fig. 5(b). However, the OASG constructed by Shen *et al.* [12] is shown in Fig. 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 Fig. 5(d).

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.

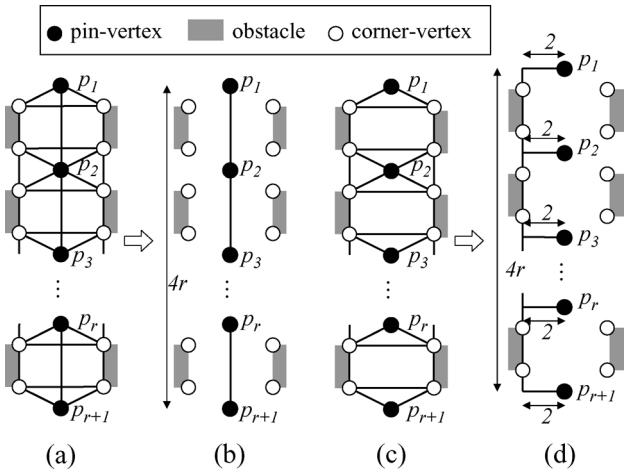


Fig. 5. Another comparison between our OASG and that of Shen *et al.* (a) Our OASG has the edges (p_i, p_{i+1}), $1 \leq i \leq r$, and (b) results in an optimal rectilinear connection with the total wirelength of $4r$. (c) The OASG of Shen *et al.* does not contain these edges and (d) results in the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

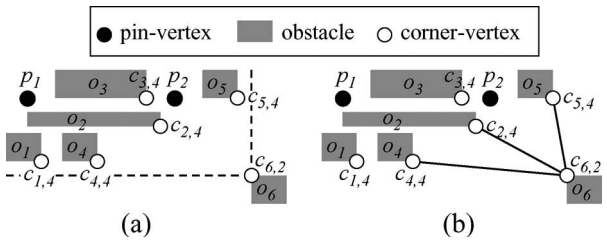


Fig. 6. (a) An example instance and (b) the OASG construction for the vertex $c_{6,2}$ in R_2 of a vertex.

As shown in Fig. 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, whereas 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 shown in Fig. 7. Fig. 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 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 18–20). 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 8–9), and $c_{1,4}$ is inserted into the candidate set A (lines 13–15). Similarly, $c_{4,4}$ and $c_{2,4}$ are inserted into the candidate set A (lines 13–15), whereas $c_{3,4}$ is not due to the intersection of the blocking information (lines 13–14). Then, when the sweeping line meets the pin vertex p_2 and the right boundary of

Algorithm: $OASG\text{-}R_2(O, P, v, E)$
Input: O /* the set of obstacles */
P /* the set of pin-vertices */
$v = (x, y)$ /* 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 **do until** the sweeping line (from left to right) meets v
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 $A = A \setminus \{v' | v' \in A$ and v' is blocked by $I\}$
8 **if** it meets r right boundaries of obstacles, $o_{\beta_1}, o_{\beta_2}, \ldots, o_{\beta_r}$
9 $I = I \setminus \{[y_{\beta_1,\min}, y_{\beta_1,\max}], \ldots, [y_{\beta_r,\min}, y_{\beta_r,\max}]\}$
10 **for** $j = 1$ **to** l
11 **if** $c_{\alpha_j,1} \in R_2$ of v and $[y, y_{\alpha_j,\min}]$ is not blocked by I
12 $A = A \cup \{c_{\alpha_j,1}\}$
13 **for** $j = 1$ **to** r
14 **if** $c_{\beta_j,4} \in R_2$ of v and $[y, y_{\beta_j,\min}]$ is not blocked by I
15 $A = A \cup \{c_{\beta_j,4}\}$
16 **else if** $c_{\beta_j,3} \in R_2$ of v and $[y, y_{\beta_j,\max}]$ is not blocked by I
17 $A = A \cup \{c_{\beta_j,3}\}$
18 **if** it meets i pin-vertices, $p_{\gamma_1}, p_{\gamma_2}, \ldots, p_{\gamma_i}$
19 **for** $j = 1$ **to** i
20 **if** $p_{\gamma_j} \in R_2$ of v and $[y, y_{\gamma_j}]$ is not blocked by I
21 $A = A \cup \{p_{\gamma_j}\}$
22 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.)
23 **for** each vertex $v' \in A$
24 **if** the vertex v' is a neighbor of v
25 $E = E \cup \{(v, v')\}$
26 Return E

Fig. 7. Algorithm of the OASG construction for the R_2 of a vertex.

o_5, p_2 and $c_{5,4}$ are inserted into the candidate set A (lines 18–21 and lines 13–15). Therefore, when the sweeping line meets the left boundary of o_6, the sweeping line halts, and the candidate set A is $\{c_{1,4}, c_{4,4}, c_{2,4}, p_2, c_{5,4}\}$. After the sorting (line 22), 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 23–24). Finally, $(c_{4,4}, c_{6,2})$, $(c_{2,4}, c_{6,2})$, and $(c_{5,4}, c_{6,2})$ are inserted into the set E (line 25), and the OASG within the R_2 of $c_{6,2}$ is constructed as shown in Fig. 6(b).

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. Moreover, 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. An example territory of the vertex s is shown in Fig. 8(a).

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 $\delta_r(s, t) = \delta_r(s, f) + \delta_r(f, t)$.

Proof: By the definition of the territory in Definition 5, t is outside or on the boundary of the territory of s; therefore, any of their rectilinear shortest paths RSP(s, t) must intersect the

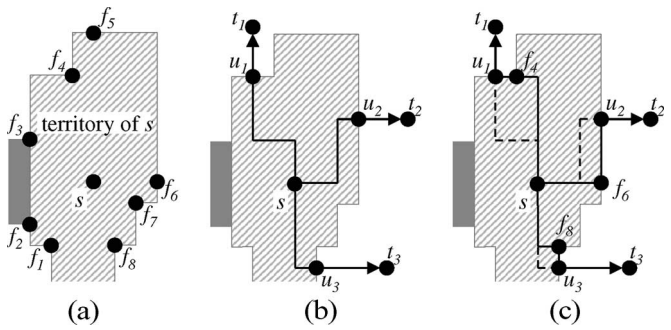


Fig. 8. (a) Example neighbors and territory of a vertex s. (b) A shortest path from s must intersect the boundary of the territory of s. (c) The path can be replaced by another path through a neighbor of s.

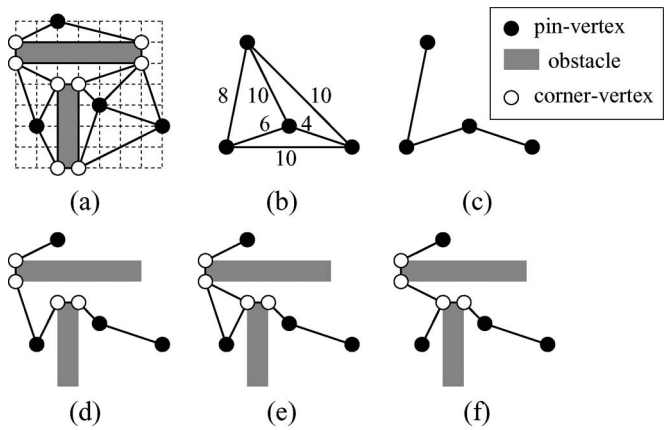


Fig. 9. Example of OAST construction.

boundary of the territory (for example, in Fig. 8(b), RSP(s, t_1) intersects the boundary at u_1). Assume that the intersecting vertex is u. Since no other vertex in $P \cup C$ or obstacle is inside the territory of s, there must exist a neighbor f of s such that RSP(s, u) can be replaced by RSP(s, f) and RSP(f, u) without increasing the total wirelength, i.e., $\delta_r(s,t) \geq \delta_r(s,f) + \delta_r(f,t)$ [for example, in Fig. 8(c), RSP(s, u_1) is replaced by RSP(s, f_4) and RSP(f_4, u_1)]. With the property of the rectilinear shortest path, $\delta_r(s,t) \leq \delta_r(s,f) + \delta_r(f,t)$. Therefore, there must exist a neighbor f of s such that $\delta_r(s,t) = \delta_r(s,f) + \delta_r(f,t)$. ■

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.

Proof: Since the OASG construction in Section III-A1 exactly constructs the edge between a vertex and its neighbor, for any neighbor f within any region of a vertex v, there exists an edge between v and f. By the definition of neighbors in Definition 4, since no obstacle is inside the bounding box of v and its neighbor f, the slant edge between v and f can directly be transformed to only one vertical edge and one horizontal edge with the rectilinear shortest length. If the edge between v and f is originally a vertical edge or a horizontal edge, the claim immediately follows. ■

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

Proof: For any pair of vertices s and t and any of their rectilinear shortest paths RSP(s, t), by Lemma 1, there must exist a neighbor f of s such that $\delta_r(s,t) = \delta_r(s,f) + \delta_r(f,t)$. By Lemma 2, a rectilinear shortest path of s and f is implied by the OASG.

At this moment, we still need to prove that a rectilinear shortest path of f and t is implied by the OASG to complete the proof. However, because f and t are both in $P \cup C$, after similar proofs (the number of proofs is finite because $\delta_r(s,t) \neq \infty$ and $\delta_r(s,f) > 0$), it is reduced to prove that a rectilinear shortest path between the vertex t and itself (t) is implied by the OASG. It is trivial, and the theorem thus follows. ■

Note that this property is very important. It can be used to guarantee that we can construct optimal solutions in many cases. Moreover, although the quality and efficiency may depend on different characteristics and distributions of obstacles, this property can also make our algorithm more stable.

B. OAST Construction

We first define an OAST as follows.

Definition 6: An 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 shown in Fig. 9, the OAST construction consists of three steps: 1) pin-vertices shortest path computation; 2) initial OAST construction; and 3) local refinement.

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 [3] for each pin-vertex pair to compute their distance, as shown in Fig. 9(b).

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 III-B1. We then apply Prim's algorithm [3] on the complete graph to obtain a minimum spanning tree [see Fig. 9(c)]. By the shortest paths computed in Section III-B1, we can map each edge in the minimum spanning tree to a shortest path in the spanning graph; therefore, the initial spanning tree on the spanning graph is constructed [see Fig. 9(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) 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 Fig. 9(e)] and compute the minimum spanning tree on it to remove unwanted cycles [see Fig. 9(f)]. This local refinement may lead to a new OAST with a smaller total wirelength.

C. OARST Construction

In this step, we transform each slant edge of the given OAST into vertical and horizontal edges to obtain an OARST.

Definition 7: An 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.

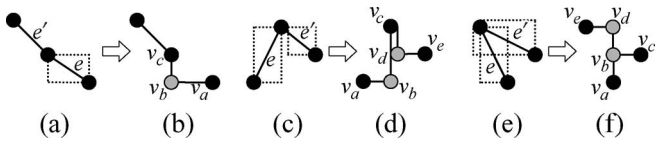


Fig. 10. Three cases in the OARST construction for a slant edge and its neighboring edge. The graphs in (a), (c), and (e) are transformed into those in (b), (d), and (f), respectively.

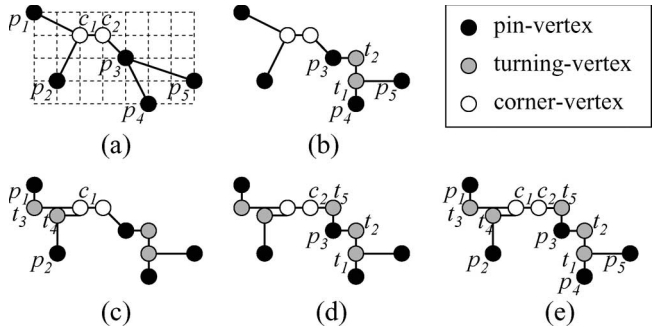


Fig. 11. Example of OARST construction.

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 Fig. 10(a)]. In this case, e is transformed into a vertical edge and a horizontal edge [see Fig. 10(b)]. There are two possible transformations; therefore, we randomly choose one.

Case 2) The two edges are in neighboring regions [see Fig. 10(c)]. In this case, both e and e' are transformed into a vertical edge and a horizontal edge. There are several possible transformations; therefore, we choose the one with edge overlap [see Fig. 10(d)].

Case 3) The two edges are in the same region [see Fig. 10(e)]. In this case, by using Fig. 10(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.

We use the example shown in Fig. 11(a) to explain the process. Initially, 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 Fig. 11(a), and the output set E is $\emptyset$. In the first iteration, the longest edge in A, (p_3, p_5), is selected as e, and its longest neighboring edge, (p_3, p_4), is selected as e'. Then, Case 3) [see Fig. 10(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 Fig. 11(b). 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 output set E is $\{(t_1, p_4), (t_1, p_5), (t_1, t_2), (t_2, p_3)\}$. Repeating similar operations, Fig. 11(b) is transformed to Fig. 11(c) and then Fig. 11(d). Finally, the OARST is constructed as shown in Fig. 11(e).

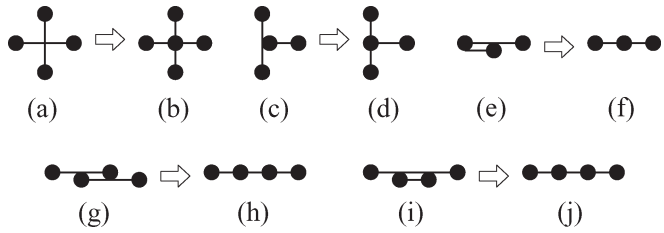


Fig. 12. Five cases of the overlapping edge removal. The graphs in (a), (c), (e), (g), and (i) are transformed into those in (b), (d), (f), (h), and (j), respectively.

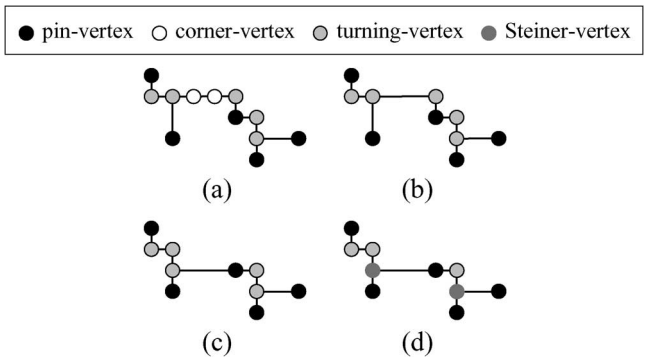


Fig. 13. OARSMT construction of Fig. 11(e).

D. OARSMT Construction

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

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

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 two, and the two edges connecting to it are parallel.

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

3) U-Shaped Pattern Refinement: The total wirelength can be further improved by some local refinements. Considering the tradeoff between solution quality and efficiency, we particularly refine U-shaped patterns because they are relatively easier to be detected. We could also consider other more complex patterns or generate some cycles and then remove the longest edges in those cycles; however, they may involve too many edges and, thus, significantly lower the efficiency. Accordingly, this refinement is much more efficient than the maze router's wire-straightening technique [4]. 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 two.

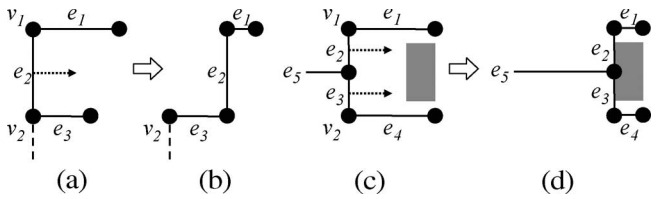


Fig. 14. Two cases of the U-shaped pattern refinement. The graphs in (a) and (c) are transformed into those in (b) and (d), respectively.

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

Case 1) Several edges form the shape as shown in Fig. 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, whereas edges e_1 and e_3 are still connected by it. Edges connected to a vertex satisfying the refinement rule [e_1 in Fig. 14(a)] are shortened. The resulting refinement is shown in Fig. 14(b).

Case 2) Several edges form the shape as shown in Fig. 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, whereas 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 Fig. 14(c)] are shortened. The resulting refinement is shown in Fig. 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. By using Fig. 13(b) as an example, the resulting removal is shown in Fig. 13(c).

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

E. 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 sufficient but not necessary conditions for an optimal solution, i.e., more optimal solutions may still be generated in other cases. Moreover, the U-shaped pattern refinement is not necessary for these theorems, implying that our OASG is indeed complete to generate these optimal solutions.

Theorem 2: If $m = 2$, our constructed OARSMT is an optimal solution.

Proof: By Theorem 1, the OASG implies a rectilinear shortest path of any two vertices in P. Hence, its corresponding path in the OASG is constructed in Section III-B, and this rectilinear shortest path is trivially constructed by the operations in Sections III-C and III-D. ■

When $m = 3$, a rectilinear Steiner tree is one of the two topologies: two simple paths between pin vertices, as shown in Fig. 15(a), or three pin vertices connected to a single Steiner vertex, as shown in Fig. 15(b). We can construct an optimal OARSMT for the first topology.

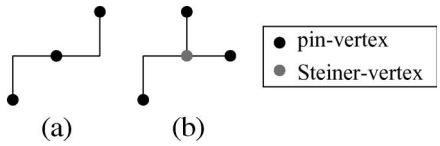


Fig. 15. When $m = 3$, a rectilinear Steiner tree is one of the two topologies: (a) Two simple paths between pin vertices or (b) three pin vertices connected to a single Steiner vertex.

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

Proof: The two simple paths are two rectilinear shortest paths between pin vertices. These rectilinear shortest paths are generated for the same reasons in Theorem 2. ■

Note that none of the aforementioned properties is guaranteed by the algorithm in [12] due to the missing "essential" edges; therefore, the algorithm in [12] cannot guarantee optimal solutions even for $m = 2$, as shown in Fig. 4. In addition, most nets in a real case are two- or three-pin nets, which makes the aforementioned properties more important for practical applications. Furthermore, regardless of the topology, we can construct an optimal OARSMT for a three-pin net if there is no obstacle.

Theorem 4: If $m = 3$ and there is no obstacle, our constructed OARSMT is an optimal solution.

Proof: If $m = 3$ and there is no obstacle, after the OAST construction, there are three cases for these three pin vertices. These cases are exactly the cases shown in Fig. 10(a), (c), and (e). Therefore, by our transformations, it is trivial that an optimal solution is generated. ■

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

Theorem 5: 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.

Proof: These simple paths are rectilinear shortest paths between pin vertices. These rectilinear shortest paths are generated for the same reasons in Theorem 2. ■

Similarly, this property is not guaranteed by the algorithm in [12].

F. Complexity Analysis

1) Number of Edges in the OASG: In the following, we present two theorems for the number of edges in the OASG for the worst and average cases. Let $n = |P| + |C|$. Theorem 6 states that the number of edges in the OASG is $O(n^2)$ in the worst case, whereas Theorem 7 shows that the expected number of edges in the OASG is $O(n \lg n)$.

Theorem 6: The number of edges in the OASG is $O(n^2)$.

Proof: There are at most n different vertices (including pin vertices and corner vertices); therefore, the number of edge is $O(n^2)$. ■

To compute the expected number of edges in the OASG, we first give several notations and lemmas.

Definition 12: Given an instance, $G_1 : (V_1, E_1)$ is defined as the OASG.

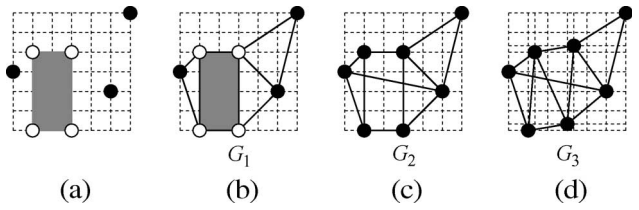


Fig. 16. Example of G_1, G_2, and G_3. (a) Given an instance, (b) G_1 has fewer edges than (c) G_2 which has fewer edges than (d) G_3.

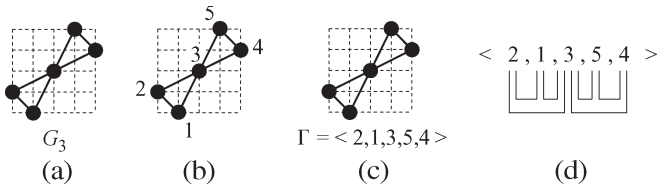


Fig. 17. (a) Given the G_3, Γ is defined by (b) labeling from 1 to n for vertices in G_3 by the order of their y-coordinates and (c) permuting these labeled numbers by the order of their x-coordinates, resulting in $\Gamma = \langle 2, 1, 3, 5, 4 \rangle$. (d) The six pairs make $N_4 = 6$ which is equal to the number of edges in G_3.

Definition 13: Given an instance, $G_2 : (V_2, E_2)$ is defined as the OASG constructed after regarding all corner vertices as pin vertices and removing all obstacles from the plane, i.e., there are totally n pin vertices and no obstacle on the plane.

Definition 14: Given an instance, $G_3 : (V_3, E_3)$ is defined as the OASG constructed after regarding all corner vertices as pin vertices, removing all obstacles from the plane, and moving a small enough distance ε for vertices with the same x- or y-coordinate so that all vertices have different x- and y-coordinates (see Fig. 16 for an example of G_1, G_2, and G_3).

Definition 15: Given an instance, a permutation Γ is defined by labeling from 1 to n for the vertices in G_3 by the order of their y-coordinates and by permuting these labeled numbers by the order of their x-coordinates.

Definition 16: Given a permutation Γ, N_4 is defined as the number of pairs (i, j), where $1 \leq i,\ j \leq n$, and there is no integer whose value is between i and j and whose position in the permutation Γ is between the positions of i and j (see Fig. 17 for an example of the permutation Γ and N_4; we have the following lemmas).

Lemma 3: $|E_1| \leq |E_2|$.

Proof: For any edge in G_1, it is always in G_2, but there are some edges in G_2 whose corresponding edges in G_1 are blocked by obstacles. As a result, $|E_1| \leq |E_2|$. ■

Lemma 4: $|E_2| \leq |E_3|$.

Proof: For any edge (v_1, v_2) in G_2, there is no other vertex inside or on the boundary of the bounding box of v_1 and v_2. Because the moving of each vertex is small enough, there is still no other vertex inside or on the boundary of the bounding box of v_1 and v_2. As a result, (v_1, v_2) is still in G_3, and $|E_2| \leq |E_3|$. ■

Lemma 5: $|E_3| = N_4$.

Proof: For vertices v_1, v_2, and v_3 in G_3, the y-coordinate of v_3 is between those of v_1 and v_2 if and only if its labeled number is between those of v_1 and v_2; the x-coordinate of v_3 is between those of v_1 and v_2 if and only if its position in the permutation Γ is between those of v_1 and v_2. For any edge (v_1, v_2) in G_3, there is no other vertex inside or on the boundary of the bounding box of v_1 and v_2, resulting in a pair of (i, j) in Γ where there is no integer whose value is between i and j and whose position in Γ is between those of i and j. On the other hand, a pair of (i, j) in Γ, where there is no integer whose value is between i and j and whose position in Γ is between those of i and j, means that there is no other vertex inside or on the boundary of the bounding box of the two corresponding vertices. As a result, $|E_3| = N_4$ due to the one-to-one mapping. ■

Lemma 6: The expected value of N_4 is $O(n \lg n)$.

Proof: For any pair $(i, i+j)$, where $i \geq 1$, $j \geq 1$, and $i + j \leq n$, there are $(j+1)!$ permutations from i to $i+j$. Among these $(j+1)!$ permutations, $(i, i+j)$ is counted if and only if i and $i+j$ are permuted successively; otherwise, there is at least an integer between i and $i+j$ whose position in Γ is between positions of i and j. Because there are $2j!$ permutations from i to $i+j$, where i and $i+j$ are permuted successively, and there are $n!$ permutation from 1 to n, the pair $(i, i+j)$ is counted $n!(2j!/(j+1)!)$ times among all permutations from 1 to n.

Because there are $(n-j)$ types of pairs $(i, i+j)$, the total count is $\sum_{j=1}^{n-1}((n-j)n!(2j!/(j+1)!))$ among all permutations from 1 to n. Therefore, the expected value of N_4 is

$$\begin{aligned}
&\frac{1}{n!}\sum_{j=1}^{n-1}\left((n-j)n!\frac{2j!}{(j+1)!}\right)\\
&=\sum_{j=1}^{n-1}\left((n-j)\frac{2}{j+1}\right)\\
&=2n\sum_{j=1}^{n-1}\frac{1}{j+1}-2\sum_{j=1}^{n-1}\left(1-\frac{1}{j+1}\right)\\
&=(2n+2)\sum_{j=1}^{n-1}\frac{1}{j+1}-2(n-1)\\
&<(2n+2)\int_{1}^{n-1}\left(\frac{1}{x}\right)dx-2(n-1)\\
&=(2n+2)\ln(n-1)-2(n-1).
\end{aligned}$$

As a result, the expected value of N_4 is $O(n \lg n)$. ■

Theorem 7: The expected number of edges in the OASG is $O(n \lg n)$.

Proof: Given an instance, by Lemmas 3, 4, and 5, the number of edges in the OASG is less than its corresponding N_4. By Lemma 6, the expected value of N_4 is $O(n \lg n)$. As a result, the expected number of edges in the OASG is $O(n \lg n)$ since the probability for each kind of the permutation Γ is the same. ■

2) Time Complexity: For the OASG construction in Section III-A, sorting is applied on the pin vertices and obstacles to perform the line sweeping algorithms. Assuming $n = |P| + |C|$, the time complexity of the sorting is $O(n \lg n)$. By using a tree structure to maintain the pin vertices and obstacles met by the sweeping line, the insertion, deletion, and searching can be done in $O(\lg n)$. Moreover, the OASG construction within a region of a vertex is $O(n \lg n)$ because the blocking information is also maintained by a tree structure. Therefore, the total time complexity for the OASG construction is $O(n^2 \lg n)$ in the worst case. However, in practical cases, the number of blocking information only depends on the local

TABLE I
COMPARISON ON THE TOTAL WIRELENGTH, WHERE "REF." IS THE U-SHAPED PATTERN REFINEMENT, "H" IS THE HALF-PERIMETER OF THE BOUNDING BOX OF ALL PIN VERTICES, AND "—" MEANS THAT THE RESULT IS NOT AVAILABLE. THE IMPROVEMENTS BEFORE "/" ARE ON THE TOTAL WIRELENGTH, WHEREAS THOSE AFTER "/" ARE ON THE DIFFERENCE FROM THE HALF-PERIMETER OF THE BOUNDING BOX OF ALL PIN VERTICES

Test Cases	m / k	Total Wirelength					Improvement (%) ($\frac{X-E}{X}$ / $\frac{X-E}{X-H}$)			
		[13] (A)	[5] (B)	MZ (C)	[12] (D)	Ours (E) / w/o REF.	[13] ($X=A$)	[5] ($X=B$)	MZ ($X=C$)	[12] ($X=D$)
ind1	10 / 32	—	—	623	646	632 / 632	—	—	−1.44 / −7.38	2.17 / 9.66
ind2	10 / 43	—	—	9,600	10,100	9,600 / 9,700	—	—	0.00 / 0.00	4.95 / 26.32
ind3	10 / 50	—	—	600	623	613 / 623	—	—	−2.17 / −12.75	1.61 / 8.00
ind4	25 / 79	—	—	1,114	1,121	1,121 / 1,121	—	—	−0.63 / −1.71	0.00 / 0.00
ind5	33 / 71	—	—	1,384	1,392	1,364 / 1,392	—	—	1.45 / 3.07	2.01 / 4.24
rc1	10 / 10	26,970	30,410	27,390	27,730	26,900 / 27,790	0.26 / 0.77	11.54 / 28.04	1.79 / 5.16	2.99 / 8.43
rc2	30 / 10	41,700	45,640	44,470	42,840	42,210 / 42,240	−1.22 / −2.29	7.52 / 13.11	5.08 / 9.04	1.47 / 2.70
rc3	50 / 10	62,380	58,570	57,740	56,440	55,750 / 56,140	10.63 / 15.42	4.81 / 7.20	3.45 / 5.19	1.22 / 1.86
rc4	70 / 10	66,560	63,340	61,320	60,840	60,350 / 60,800	9.33 / 13.29	4.72 / 6.88	1.58 / 2.34	0.81 / 1.20
rc5	100 / 10	80,010	83,150	77,930	76,970	76,330 / 76,760	4.60 / 6.09	8.20 / 10.73	2.05 / 2.74	0.83 / 1.12
rc6	100 / 500	—	149,750	82,999	86,403	83,365 / 84,139	—	44.33 / 51.00	−0.44 / −0.58	3.52 / 4.55
rc7	200 / 500	—	181,470	114,146	117,427	113,260 / 114,173	—	37.59 / 42.21	0.78 / 0.94	3.55 / 4.27
rc8	200 / 800	—	202,741	120,698	123,366	118,747 / 120,436	—	41.43 / 45.91	1.62 / 1.93	3.74 / 4.46
rc9	200 / 1,000	—	214,850	116,390	119,744	116,168 / 117,647	—	45.93 / 50.64	0.19 / 0.23	2.99 / 3.58
rc10	500 / 100	—	198,010	174,110	171,450	170,690 / 171,520	—	13.80 / 15.34	1.96 / 2.22	0.44 / 0.50
rc11	1,000 / 100	—	250,570	243,074	238,111	236,615 / 238,095	—	5.57 / 6.05	2.66 / 2.90	0.63 / 0.69
rc12	1,000 / 10,000	—	1,723,990	785,904	843,529	789,097 / 803,094	—	54.23 / 56.37	−0.41 / −0.44	6.45 / 7.00
rt1	10 / 500	—	—	2,277	2,438	2,267 / 2,289	—	—	0.44 / 1.09	7.01 / 15.91
rt2	50 / 500	—	—	48,242	51,981	48,441 / 48,833	—	—	−0.41 / −0.62	6.82 / 9.92
rt3	100 / 500	—	—	8,378	8,783	8,368 / 8,508	—	—	0.12 / 0.16	4.73 / 6.11
rt4	100 / 1,000	—	—	10,676	10,619	10,306 / 10,457	—	—	3.47 / 4.26	2.95 / 3.63
rt5	200 / 2,000	—	—	55,232	55,557	53,993 / 54,672	—	—	2.24 / 2.63	2.82 / 3.30
Avg.	—	—	—	—	—	—	4.72 / 6.66	23.31 / 27.79	1.06 / 0.93	2.89 / 5.79

topology and can be regarded as a constant, reducing the time complexity of the OASG construction to $O(n \lg n)$.

For the OAST construction in Section III-B, heaps are applied. Because the number of edges in the OASG is $O(n^2)$ in the worst case, the time complexity of the pin-vertices shortest path computation in Section III-B1 is $O(n^3)$. The time complexity of the initial OAST construction in Section III-B2 and the local optimization in Section III-B3 are both $O(n^2)$. Totally, the time complexity for the OAST construction is $O(n^3)$ in the worst case. However, in practical cases, the expected number of edges in the OASG is $O(n \lg n)$ [by the least squares fitting to be presented in Section IV, it is about $O(n^{1.03})$], reducing the time complexity of the pin-vertices shortest path computation and the OAST construction to $O(n^2 \lg n)$.

For the OARST construction in Section III-C, sorting is applied on the edges in the OAST. Because the number of the edges in the OAST is $O(n)$, the time complexity for the OARST construction is $O(n \lg n)$. For the OARSMT construction in Section III-D, the time complexity is $O(n^2)$ for checking each pair of edges and each vertex in the OARST. In addition, the time complexity of the U-shaped pattern refinement is also $O(n^2)$ in the worst case.

Finally, the overall time complexity of our algorithm is $O(n^3)$ in the worst case and $O(n^2 \lg n)$ for practical applications (see Section IV for the empirical performance).

IV. 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, 12 test cases used in [5] (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; therefore, 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. Moreover, the positions of pin vertices were also randomly generated without locating inside any obstacle.

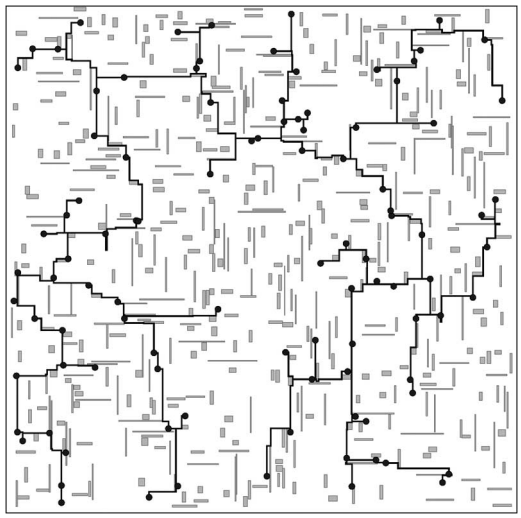


Fig. 18. Final routing result of rt3, where a pin vertex is represented by a solid circle.

We compared our algorithm with those presented in [5], [12], and [13]. We also compared with a maze-routing-based algorithm, namely, MZ, which constructs a minimum spanning tree first, starts from one pin as a constructed OARSMT, and iteratively finds the shortest path from an unconnected pin to the constructed OARSMT by Clow's A$*$ maze routing in [2]. The results in [13] are provided by the authors and were generated from a Unix workstation with 2.66-GHz CPU and 1-GB memory. The results in [5] are directly quoted from the paper, where the algorithm was performed on a Sun V880

TABLE II
COMPARISON ON THE CPU TIME, WHERE "—" MEANS THAT THE RESULT IS NOT AVAILABLE

Test Cases	CPU Time (second)					# Edges in our OASG
	[13]	[5]	MZ	[12]	Ours	
ind1	—	—	0.01	< 0.01	< 0.01	437
ind2	—	—	0.01	< 0.01	< 0.01	798
ind3	—	—	0.01	< 0.01	< 0.01	903
ind4	—	—	0.02	< 0.01	< 0.01	488
ind5	—	—	0.02	< 0.01	0.01	346
rc1	0.49	< 0.01	< 0.01	< 0.01	< 0.01	125
rc2	1.03	< 0.01	0.01	< 0.01	< 0.01	205
rc3	8.79	< 0.01	0.02	< 0.01	< 0.01	314
rc4	67.62	< 0.01	0.02	< 0.01	< 0.01	427
rc5	595.10	< 0.01	0.04	0.01	0.01	574
rc6	—	0.06	0.31	0.17	0.24	6,582
rc7	—	0.06	0.57	0.30	0.43	7,299
rc8	—	0.10	0.70	0.48	0.83	10,332
rc9	—	0.13	0.87	0.64	0.91	12,505
rc10	—	0.03	1.24	0.27	0.62	4,445
rc11	—	0.04	15.64	0.95	3.15	10,546
rc12	—	2.82	632.84	65.73	118.52	204,091
rt1	—	—	0.16	0.06	0.06	5,358
rt2	—	—	0.31	0.11	0.11	6,244
rt3	—	—	0.32	0.16	0.47	6,447
rt4	—	—	0.57	0.34	0.95	10,832
rt5	—	—	1.95	1.42	2.06	21,938

fire workstation with 755-MHz CPU and 4-GB memory. We implemented the algorithm in [12] and MZ. Different from our OASG graph construction, the algorithm in [12] only constructs an edge within each region. In addition, it operates without the U-shaped pattern refinement, as described in Section III-D3. We also verified the generated OARSMTs by another program to ensure that all pin vertices were connected without intersecting any obstacle.

Table I lists the total wirelengths of these algorithms without any scaling. Fig. 18 shows the resulting layout for the test case rt3. Considering the differences from the half-perimeter of the bounding box of all the pin vertices, the respective average improvements are 6.66%, 27.70%, 5.79%, and 0.93%, as compared with the algorithms in [5], [12], and [13], and MZ. The average improvement over the algorithm in [12] with the OASG alone is about 3.69%, whereas the overall improvement is about 5.79%. Note that our algorithm is more efficient than MZ for most cases and can have both shorter wirelengths and less running times in many cases. In contrast, MZ never obtains shorter wirelengths and less running times at the same time. 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.

Table II gives the comparison on the CPU times of these algorithms. Our algorithm is also sufficiently efficient. For example, when the numbers of pin vertices and obstacles reach 200 and 800, respectively (rc8), our algorithm takes only 0.83 s and achieves 4.46% and 1.93% improvements over the algorithm in [12] and MZ, respectively. By the least squares fitting on the log-log axes, the respective slopes of the fitting lines for our algorithm, that in [12], and MZ are 1.46, 1.40, and 1.59, implying that the empirical time complexity of our algorithm is close to $O(n^{1.46})$, whereas those in [12] and the MZ are about $O(n^{1.40})$ and $O(n^{1.59})$, respectively. Note that this is reasonable since we add more edges into our OASGs to guarantee the optimality described in Section III-E, whereas the work in [12] does not. Furthermore, the empirical time complexity is far under the theoretical worst case complexity of $O(n^3)$ in Section III-F2. 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 II. 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.

V. CONCLUSION

We have proposed an algorithm to construct an OARSMT. We can achieve an optimal solution for any two-pin net and nets with more pins in many cases. The 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.

ACKNOWLEDGMENT

The authors would like to thank J.-R. Gao and Prof. T.-C. Wang of the National Tsing Hua University for providing the test data.

REFERENCES

[1] K. L. Clarkson, S. Kapoor, and P. M. Vaidya, "Rectilinear shortest paths through polygonal obstacles in $O(n(\log n)^2)$ time," in *Proc. Symp. Comput. Geom.*, 1987, pp. 251–257.

[2] G. W. Clow, "A global routing algorithm for general cells," in *Proc. DAC*, 1984, pp. 45–51.

[3] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge, MA: MIT Press, 2001.

[4] J. Dion and L. M. Monier, "Contour: A tile-based gridless router," WRL, Palo Alto, CA, Res. Rep. 95/3, 1995.

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

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

[7] M. R. Garey and D. S. Johnson, "The rectilinear Steiner tree problem is NP-complete," *SIAM J. Appl. Math.*, vol. 32, no. 4, pp. 826–834, Jun. 1977.

[8] 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," *J. Inf. Comput. Sci.*, vol. 1, no. 3, pp. 107–116, 2004.

[9] 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. ASP-DAC*, 2005, pp. 7–12.

[10] C. Y. Lee, "An algorithm for path connections and its application," *IRE Trans. Electron. Comput.*, vol. 10, no. 2, pp. 346–365, Sep. 1961.

[11] C.-W. Lin, S.-Y. Chen, C.-F. Li, Y.-W. Chang, and C.-L. Yang, "Efficient obstacle-avoiding rectilinear Steiner tree construction," in *Proc. ISPD*, 2007, pp. 127–134.

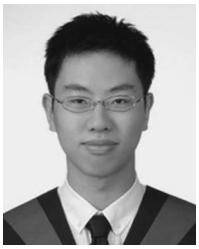
[12] Z. C. Shen, C. C. N. Chu, and Y.-M. Li, "Efficient rectilinear Steiner tree construction with rectilinear blockages," in *Proc. ICCD*, 2005, pp. 38–44.

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

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

[15] S. Q. Zheng, J. S. Lim, and S. S. Iyengar, "Finding obstacle-avoiding shortest paths using implicit connection graphs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 15, no. 1, pp. 103–110, Jan. 1996.

[16] H. Zhou, "Efficient Steiner tree construction based on spanning graphs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 5, pp. 704–710, May 2004.



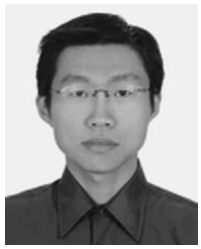
Chung-Wei Lin received the B.S. degree in computer science and information engineering and the M.S. degree in electronics engineering from the National Taiwan University, Taipei, Taiwan, R.O.C., in 2005 and 2007, respectively.

He is currently in the military service. His research interests include routing-related topics and design for manufacturability/reliability.

Mr. Lin was the recipient of the Presidential Award from the National Taiwan University for six semesters during his college years. He is an honorary member of the Phi Tau Phi Scholastic Honor Society of Taiwan.

Szu-Yu Chen received the B.S. degree in electrical engineering from the National Taiwan University, Taipei, Taiwan, R.O.C., in 2007. He is currently working toward the M.S. degree at the Graduate Institute of Electronics Engineering, National Taiwan University, Taipei.

His current research interests include VLSI electronic design automation, large-scale routing, and design for manufacturability/reliability.



Chi-Feng Li received the B.S. degree in computer science from the National Tsing Hua University, Hsinchu, Taiwan, R.O.C., in 2005, and the M.S. degree in computer science and information engineering from the National Taiwan University, Taipei, Taiwan, in 2007.

He is currently with the Department of Computer Science and Information Engineering, National Taiwan University. His research interests include design automation and low-power designs for the embedded computing systems.

Yao-Wen Chang (S'94–M'96) received the B.S. degree from National Taiwan University, Taipei, Taiwan, in 1988, and the M.S. and Ph.D. degrees from the University of Texas at Austin in 1993 and 1996, respectively, all in computer science.

He is a Professor in the Department of Electrical Engineering and the Graduate Institute of Electronics Engineering, National Taiwan University. He is currently also a Visiting Professor at Waseda University, Kitakyushu, Japan. He was with the IBM T. J. Watson Research Center, Yorktown Heights, NY, in the summer of 1994. From 1996 to 2001, he was on the faculty of National Chiao Tung University, Taiwan. His current research interests include VLSI physical design, design for manufacturability and reliability, design automation for biochips, and FPGA. He has been working closely with industry on projects in these areas. He has coauthored one book on routing and over 130 ACM/IEEE conference/journal papers in these areas.

Dr. Chang received an award at the 2006 ACM ISPD Placement Contest, Best Paper Awards at ICCD-95 and the 2007 VLSI Design/CAD Symposium, and 11 Best Paper Award Nominations from DAC (2000, 2005, 2007, 2008), ICCAD (2002, 2007), ISPD (two in 2007), ACM TODAES (2003), ASP-DAC (2004), and ICCD (2001). He has received many awards for research, such as the 2007 Distinguished Research Award, the inaugural 2005 First-Class Principal Investigator Award, and the 2004 Dr. Wu Ta You Memorial Award from the National Science Council of Taiwan, the 2004 MXIC Young Chair Professorship from the MXIC Corporation, and for excellent teaching from National Taiwan University (2004, 2006, 2007) and National Chiao Tung University (2000). He is currently an Associate Editor of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS (TCAD) and an editor of the *Journal of Information Science and Engineering* (JISE). He currently serves on the ICCAD Executive Committee, the ACM/SIGDA Physical Design Technical Committee, and the ISPD Organizing Committee and has served on the technical program committees of ASP-DAC (topic chair), DAC, DATE, FPL, FPT (program co-chair), GLSVLSI, ICCAD, ICCD, IECON (topic chair), ISPD, SOCC (topic chair), TENCON, and VLSI-DAT (topic chair). He is currently an independent board director of Genesys Logic, Inc., a member of the board of governors of the Taiwan IC Design Society, and a member of the IEEE Circuits and Systems Society, ACM, and ACM/SIGDA.

Chia-Lin Yang (M'02) received the B.S. degree from the National Taiwan Normal University, Taipei, Taiwan, R.O.C., in 1989, the M.S. degree from the University of Texas at Austin, in 1992, and the Ph.D. degree from the Department of Computer Science, Duke University, Durham, NC, in 2001.

In 1993, she was with VLSI Technology Inc. (now Philips Semiconductors) as a Software Engineer. She is currently an Associate Professor with the Department of Computer Science and Information Engineering, National Taiwan University, Taipei. Her research interests include energy-efficient microarchitectures, memory hierarchy design, and multimedia workload characterization.

Dr. Yang was the recipient of a 2000 to 2001 Intel Foundation Graduate Fellowship Award and the 2005 IBM Faculty Award.