Multilayer Obstacle-Avoiding Rectilinear Steiner Tree Construction Based on Spanning Graphs

Chung-Wei Lin, Shih-Lun Huang, Kai-Chi Hsu, Meng-Xiang Lee, and Yao-Wen Chang, *Member, IEEE*

***Abstract*—Given a set of pins and a set of obstacles on routing layers, a multilayer obstacle-avoiding rectilinear Steiner minimal tree (ML-OARSMT) connects these pins by rectilinear edges within layers and vias between layers and avoids running through any obstacle to construct a Steiner tree with a minimal total cost. The ML-OARSMT problem is very important for many very large scale integration designs with pins being located in multiple routing layers that contain numerous routing obstacles incurred from IP blocks, power networks, prerouted nets, etc. As a fundamental problem with extensive practical applications to routing and wirelength/congestion/timing estimations in early design stages, it is desired to develop an effective algorithm for the ML-OARSMT problem to facilitate the design flow. However, there is no existing work on this ML-OARSMT problem. In this paper, we first formulate the ML-OARSMT problem with rectangular obstacles and then identify key different properties of this problem from its single-layer counterpart. Based on the multilayer obstacle-avoiding spanning graph, we present the first algorithm to solve the ML-OARSMT problem. Our algorithm can guarantee an optimal solution for any two-pin net and many multiple-pin nets. Experiments show that our algorithm results in 33% smaller total costs on average than a construction-by-correction heuristic which is widely used for Steiner-tree construction in the recent literature.**

***Index Terms*—Layout, physical design, routing, Steiner tree.**

I. Introduction

GIVEN A SET of pins and a set of obstacles on routing layers, a multilayer obstacle-avoiding rectilinear Steiner minimal tree (ML-OARSMT) connects these pins by rectilinear edges within layers and vias between layers and avoids running through any obstacle to construct a Steiner tree with a minimal total cost.

Manuscript received January 20, 2008; revised May 17, 2008. Current version published October 22, 2008. This work was supported in part by the National Science Council of Taiwan under Grants NSC 96-2628-E-002-248-MY3, NSC 96-2628-E-002-249-MY3, NSC 96-2221-E-002-245, and NSC 96-2752-E-002-008-PAE, by Etron, by Incentia, by SpringSoft, and by TSMC. An earlier version of this paper was presented at the 2007 IEEE/ACM International Conference on Computer Aided Design [7]. This paper was recommended by Associate Editor L. Scheffer.

C.-W. Lin, S.-L. Huang, and M.-X. Lee were with the Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 106, Taiwan (e-mail: enorm@eda.ee.ntu.edu.tw; aaron@eda.ee.ntu.edu.tw; bo27@eda.ee.ntu.edu.tw).

K.-C. Hsu was with the Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan. She is now with Raydium Semiconductor Corporation, Hsinchu 300, Taiwan (e-mail: kethy@eda.ee.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 (e-mail: ywchang@cc.ee.ntu.edu.tw).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2008.2006095

The ML-OARSMT problem is of particular importance for modern very large scale integration (VLSI) designs. First of all, a port may have two output pins on adjacent layers at the same location; therefore, it can be used to facilitate the cell placement in any orientation. A router should select a preferred layer to connect the port so that the routing resource can be utilized efficiently, including many routing constraints such as the total wirelength, timing delay, number of vias, and design rules. Moreover, most pins of standard cells are located in lower layers, while many pins of macro cells are located in higher layers. Therefore, the router should be able to connect all the pins of a net, no matter on which layers these pins are. Furthermore, a router should also consider routing obstacles incurred from IP blocks, power networks, prerouted nets, feature patterns for manufacturability improvement, etc. As a fundamental problem with extensive practical applications to routing and wirelength/congestion/timing estimations, it is desired to develop an effective algorithm for the ML-OARSMT problem.

However, the rectilinear Steiner minimal tree problem on a plane (on a layer), even without the obstacle and multilayer considerations, is a well-known NP-complete problem [3], and the presences of obstacles and multilayers further increase the complexity. There is no existing work on this ML-OARSMT problem in the literature. Without considering obstacles, Yildiz and Madden [13] first addressed the Steiner tree construction on multiple layers. They extended an existing single-layer (SL) rectilinear Steiner tree heuristic and considered preferred routing restrictions and layer-specific routing and via costs.

On the other hand, the SL-OARSMT problem, which only considers an SL, has received dramatically increasing attention in recent years [2], [4]–[6], [8]–[12]. These works can be classified into two major categories including 1) *the construction-by-correction approach* and 2) *the connection graph-based approach*. The construction-by-correction approach constructs a Steiner or a spanning tree for a net first and then replaces the edges overlapping obstacles with edges around the obstacles. However, it may lose the global view of the obstacles and remove the overlaps locally around the obstacles. As a result, the solution quality may be limited. In contrast, the connection graph-based approach is first to construct a connection graph by pins and obstacle boundaries, which guarantees at least a desired SL-OARSMT to be embedded in the graph. Then, some search techniques are applied to find the desired SL-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.

Additionally, the switchbox rectilinear Steiner tree (SRST) problem is a special case of the SL-OARSMT problem. The SRST problem is to construct an optimal rectilinear Steiner tree interconnecting terminals on the perimeter of a switchbox without crossing any obstacles inside the switchbox. Chiang *et al.* [1] presented an algorithm to find an optimal solution for the SRST problem with an exponential-time complexity.

Nevertheless, none of these existing works considers the presences of obstacles and multilayers at the same time. There are several differences between the SL-OARSMT problem and the ML-OARSMT one.

1) For the SL-OARSMT problem, the shortest path between two pins can be constructed by considering only the pins and corners of obstacles. However, for the ML-OARSMT problem, considering only pins and corners of obstacles cannot always find the shortest path.
2) The SL-OARSMT problem only needs to consider the wirelength on a layer, but via costs should be included for the ML-OARSMT problem.

As a result, direct extensions of these existing methods to the ML-OARSMT problem may have limited solution quality or even generate some infeasible solutions. In this paper, we first formulate the ML-OARSMT problem with rectangular obstacles and then develop an effective and efficient algorithm to deal with the problem. Our algorithm consists of four steps: It first constructs a multilayer obstacle-avoiding spanning graph (ML-OASG), then constructs a corresponding spanning tree, converts the spanning tree into a rectilinear one, and finally turns the rectilinear tree into a rectilinear Steiner tree. Our work has the following distinguished features (and theoretical findings).

1) This is the first work in the literature for the ML-OARSMT problem.
2) The ML-OASG is significantly different from the SL counterpart presented in [6]. To guarantee an optimal SL-OARSMT for any 2-pin net and many multiple-pin nets, it suffices to consider the vertices induced from the pins and corners. However, this property does not hold for ML-OASG. Therefore, we need to develop a more sophisticated method with a more global view to guarantee this optimality.
3) For the SL-OARSMT problem, the completeness of a spanning graph helps the wirelength reduction significantly [6]. For the ML-OARSMT one, however, the completeness decreases the flexibility of transforming a slant edge into rectilinear edges. With this finding, we can significantly prune the solution space to speed up the processing. Our empirical results show a 3.5 times speedup due to this finding; furthermore, the redundant solution pruning does not affect the solution quality because it induces a much smaller *essential* solution space.

Experiments show that our algorithm results in significantly smaller total costs (including wirelength and via costs) than a heuristic of the construction-by-correction approach by 33% on average.

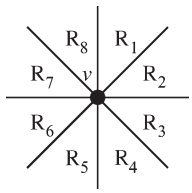


Fig. 1. For each vertex v being the center, the plane is divided into eight regions in [14], and it suffices to connect v to at most one vertex in each region for the Steiner minimal tree construction.

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

II. Background

Zhou [14] first introduced the spanning graph for VLSI Steiner minimal tree construction. Given a set of vertices on the plane, a spanning graph is a graph over the vertices that contains a minimal spanning tree. Since the spanning graph is sparse, it incurs lower time complexity for the tree construction. As shown in Fig. 1, for each vertex v being the center, the plane is divided into eight regions; for the Steiner minimal tree construction, it suffices to connect v to at most one vertex in each region because the distance between any two vertices in a region must be smaller than the maximum distance from v to either of the two vertices. Based on the spanning graph, Zhou [14] developed an efficient and effective Steiner tree algorithm with the time complexity $O(n \lg n)$.

III. Problem Formulation

We define an *obstacle*, a *pin vertex*, and a *via* as follows.

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

See layer 2 in Fig. 2(a) for two overlapped obstacles and layer 1 for point- and line-touched obstacles.

Definition 2: A *pin vertex* is a vertex on an arbitrary layer. A pin vertex must not locate inside any obstacle, but it could be at the corner or on the boundary of an obstacle.

See layer 2 in Fig. 2(b) for an illegal instance with two pin vertices inside an obstacle and layer 1 for a legal instance with a pin vertex at the corner and others on the boundary of an obstacle.

Definition 3: A *via* on layer z is an edge between (x, y, z) and $(x, y, z+1)$. (x, y, z) and $(x, y, z+1)$ must not locate inside any obstacle, but they could be at the corner or on the boundary of an obstacle.

See illegal vias inside an obstacle between layers 2 and 3 in Fig. 2(c) and legal ones between layers 1 and 2. Moreover, no edge of the ML-OARSMT can intersect with any obstacle as shown on layer 2 in Fig. 2(d). However, in practice, to maintain the space between an obstacle and an edge, an obstacle may be modeled as a larger one, so an edge could be point touched at the corner or line touched on the boundary of an obstacle as shown on layer 1 in Fig. 2(d).

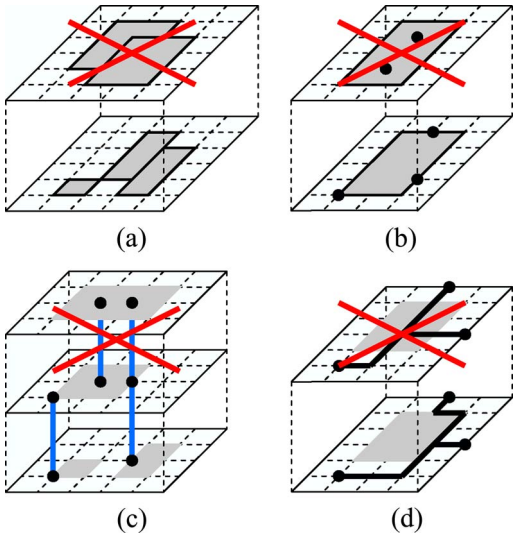


Fig. 2. (a) Illegal obstacles (layer 2) and legal obstacles (layer 1). (b) Illegal pin vertices (layer 2) and legal pin vertices (layer 1). (c) Illegal vias (between layers 2 and 3) and legal vias (between layers 1 and 2). (d) Illegal edges (layer 2) and legal edges (layer 1).

Let C_v be the cost of a via, N_l be the number of layers, $P = \{p_1, p_2, \ldots, p_m\}$ 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 $P \cup \{\text{corners in } O\}$ for the problem; we have $n \leq m + 4k$ since each obstacle has four corners. Furthermore, without obstacles, the connection cost between v_i and v_j can be computed by $|x_i - x_j| + |y_i - y_j| + |z_i - z_j| \times C_v$.

Considering rectilinear routes which use rectilinear edges within layers and vias between layers, we define the ML-OARSMT problem as follows.

- **Problem: ML-OARSMT:** Given constants C_v and N_l, a set P of pins, and a set O of obstacles, construct a multilayer rectilinear Steiner tree to connect the pins in P such that no tree edge or via intersects an obstacle in O and the total cost of the tree is minimized.

Note that an edge is allowed to pass through the boundary between two line-touched obstacles, according to this formulation. To deal with rectilinear obstacles, the formulation can be modified so that an edge is not allowed to pass through the boundary between two line-touched obstacles. If the formulation is modified, edges of this type should be removed from the spanning graph, and our algorithm can still work well.

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 $(x_{i,\min}, y_{i,\min}, z_i)$, $(x_{i,\min}, y_{i,\max}, z_i)$, $(x_{i,\max}, y_{i,\max}, z_i)$, and $(x_{i,\max}, y_{i,\min}, z_i)$, respectively. Moreover, $C = \bigcup_{i=1}^{k} \{c_{i,j}\}$, $j = 1, 2, 3,$ and 4.

IV. Algorithm

Our algorithm consists of four steps which are presented as follows.

1) An ML-OASG is constructed to connect all vertices in $P \cup C$. The construction is much different from the SL counterpart in [6], even on a single layer. See Fig. 3(b) for an example of the ML-OASG.
2) A multilayer obstacle-avoiding spanning tree (ML-OAST) is constructed to connect all pin vertices by selecting edges from the ML-OASG. See Fig. 3(c) for an example of the ML-OAST.
3) A multilayer obstacle-avoiding rectilinear spanning tree (ML-OARST) is constructed by transforming each slant edge of the ML-OAST into rectilinear edges. See Fig. 3(d) for an example of the ML-OARST.
4) An ML-OARSMT is finally constructed by removing redundant vertices, merging overlapping edges, and introducing Steiner points. See Fig. 3(e) for an example of the ML-OARSMT.

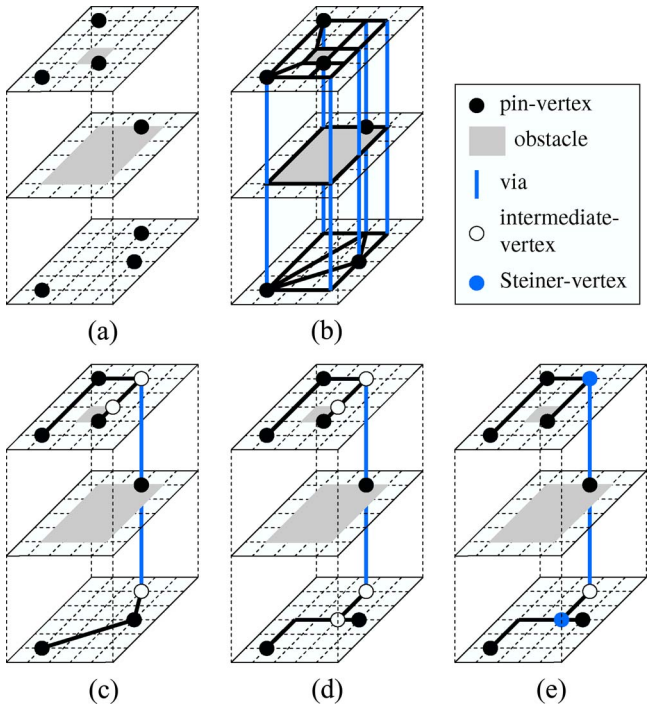


Fig. 3. Four steps for ML-OARSMT construction.

The following sections detail the four steps.

A. ML-OASG Construction

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

Definition 4: An ML-OASG is an undirected graph connecting all vertices in $P \cup C$, and no edge intersects with an obstacle in O.

1) Difficulties: It is not feasible to directly extend the single-layer obstacle-avoiding spanning graph (SL-OASG) construction in [6]. The SL-OASG can guarantee a rectilinear shortest path between two vertices, resulting in an optimal SL-OARSMT solution in some cases. Its SL-OASG is constructed by the following rules.

1) SL-OASG Connection Rule 1: The SL-OASG is constructed on all pin and corner vertices.
2) SL-OASG Connection Rule 2: Two vertices are connected if 1) there is no other vertex inside or on the

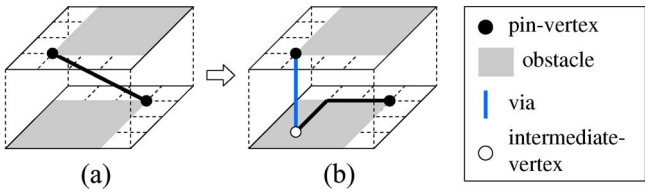


Fig. 4. (a) A trivial extension from the SL-OASG construction will construct an edge between the two vertices, but (b) it may result in an infeasible solution.

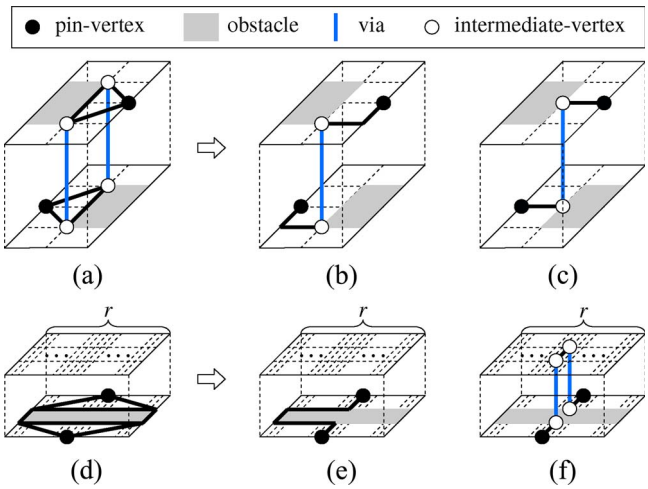


Fig. 5. (a) If an ML-OASG is constructed only on pin and corner vertices, (b) the resulting total cost is $4 + C_v$, (c) while the optimal total cost is $2 + C_v$. (d) For another instance, (e) the resulting total cost is $3 + r$, (f) while there is a solution with the total cost $3 + 2C_v$. If $r \gg C_v$, the difference will be very significant.

boundary of the bounding box of the two vertices and 2) there is no obstacle inside the bounding box of the two vertices.

Therefore, a trivial extension to the ML-OARSMT problem is to construct an ML-OASG based on the SL-OASG Connection Rule 2. As shown in Fig. 4(a), because the two vertices satisfy the SL-OASG Connection Rule 2, there will be an edge between them. However, different from the spanning graph in [6] which can transform slant edges into arbitrary L-shaped edges, the extended method may result in an infeasible solution which has an edge overlapping an obstacle as shown in Fig. 4(b). Thus, we have the following finding from this case.

Theorem 1: The SL-OASG Connection Rule 2 may result in infeasible solutions for the ML-OARSMT problem.

On the other hand, it is not sufficient to use the SL-OASG construction in [6] which is only applied on pin and corner vertices (SL-OASG Connection Rule 1). If an ML-OASG is constructed only on pin and corner vertices as shown in Fig. 5(a), it will result in an ML-OARSMT as shown in Fig. 5(b), which has a larger total cost $4 + C_v$ than the optimal total cost $2 + C_v$, as shown in Fig. 5(c). See Fig. 5(d) for another example where all pin vertices and obstacles are on the same layer, and the obstacle's length and width are r unit and 1 unit, respectively. If an ML-OASG is constructed only on pin and corner vertices, it will result in an ML-OARSMT as shown in Fig. 5(e) with the total cost of $3 + r$ which is an optimal solution on a single layer.

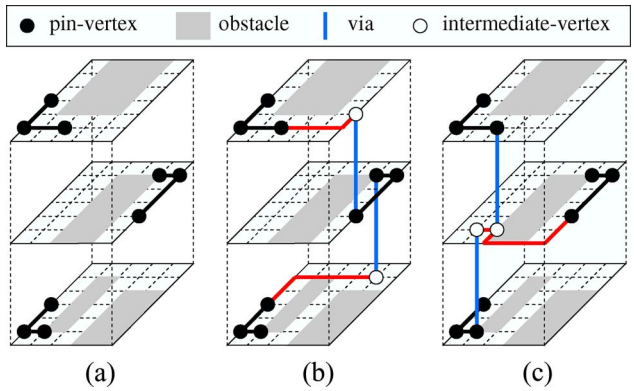


Fig. 6. (a) Given an instance, if an extension deals with layers one by one and connects between layers by the shortest paths, (b) the cost for the connections between layers will be up to $10 + 2C_v$ (vias and red edges). (c) Considering multiple layers at the same time, the cost for the connections between layers is $7 + 2C_v$.

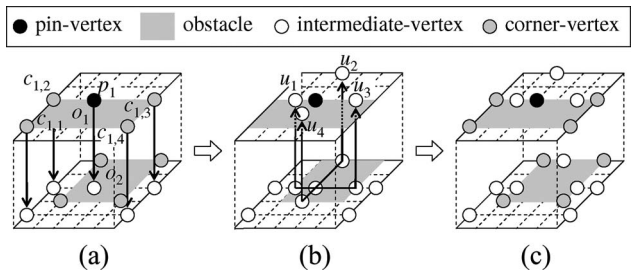


Fig. 7. Example vertex projection between layers.

However, there is an ML-OARSMT with the total cost $3 + 2C_v$ as shown in Fig. 5(f). If $r \gg C_v$, the difference between the two solutions will be very significant. From these two examples, we have the following finding.

Theorem 2: SL-OASG Connection Rule 1 cannot guarantee a rectilinear shortest path between two vertices.

Another extension to the ML-OARSMT problem is to deal with layers one by one. The SL-OARSMT construction in [6] can be applied on each layer, and some heuristics can be used to connect those SL-OARSMTs. However, given an instance where the SL-OARSMTs are far from each other layer by layer as shown in Fig. 6(a), if the connections between layers are decided by shortest paths, the cost for connections between layers will be up to $10 + 2C_v$ (vias and red edges) as shown in Fig. 6(b), which is not what we desire during the ML-OARSMT construction on each layer. Considering multiple layers at the same time, there is a smaller cost for connections between layers is $7 + 2C_v$ (vias and red edges) as shown in Fig. 6(c).

To avoid these problems, the SL-OASG on a layer should consider more *essential* vertices which may be on the same layer or on the other layers. Thus, we propose an algorithm so that some vertices are projected between layers and within a layer without overlapping with any obstacles. As a result, we can construct better ML-OARSMTs and find a rectilinear shortest path of any two vertices in our ML-OASG.

2) Vertex Projection Between Layers: Fig. 7 shows an example vertex projection between layers from layer 2 to 1. At the beginning, $O = \{o_1, o_2\}$, $s = 2$, $t = 1$, $V_s =$

Algorithm: Vertex-Projection-between-Layers(O, s, t, V_s, V_t, G)
Input: O /* the set of obstacles */
s /* the source-layer of projection */
t /* the target-layer of projection */
V_s /* the set of vertices on layer s */
V_t /* the set of vertices on layer t */
$G = (V, E)$ /* the ML-OASG */
1 **for** each vertex $v \in V_s$
2 $(x, y) =$ the x- and y-coordinates of v
3 **if** (x, y, t) is inside or on the boundary of an obstacle o_i
4 $u_1 = (x_{i,\min}, y, s)$
5 $u_2 = (x, y_{i,\max}, s)$
6 $u_3 = (x_{i,\max}, y, s)$
7 $u_4 = (x, y_{i,\min}, s)$
8 **for** $j = 1$ **to** 4
9 **if** u_j is not inside any obstacle
10 $(x', y') =$ the x- and y-coordinates of u_j
11 $u_t = (x', y', t)$
12 $V_s = V_s \cup \{u_j\}$
13 $V_t = V_t \cup \{u_t\}$
14 $V = V \cup \{u_j, u_t\}$
15 $E = E \cup \{(u_j, u_t)\}$
16 **else**
17 $u_t = (x, y, t)$
18 $V_t = V_t \cup \{u_t\}$
19 $V = V \cup \{u_t\}$
20 $E = E \cup \{(u_t, v)\}$

$\{c_{1,1}, c_{1,2}, c_{1,3}, c_{1,4}, p_1\}$, and $V_t = \{c_{2,1}, c_{2,2}, c_{2,3}, c_{2,4}\}$. As shown in Fig. 7(a), because the projected vertices of $c_{1,1}$, $c_{1,2}$, $c_{1,3}$, and $c_{1,4}$ are not inside any obstacle, the four projected vertices are inserted into V_1 (the vertex set of layer 1) and V (the vertex set of the spanning graph) (lines 16–19). Moreover, the edges between $c_{1,1}$, $c_{1,2}$, $c_{1,3}$, and $c_{1,4}$ and their projected vertices on layer 1 are inserted into E (the edge set of the spanning graph), respectively (line 20). On the other hand, the projected vertex of p_1 is inside o_2 (line 3); therefore, the algorithm sets u_1, u_2, u_3, and u_4 (lines 4–7) and checks if each of them is inside any obstacle or not as shown in Fig. 7(b). Because u_1, u_2, and u_3 are not inside any obstacle (line 9), they are inserted into V_2 (the vertex set of layer 2) and V, while their projected vertices on layer 1 are inserted into V_1 and V (lines 9–14). Moreover, the edges between u_1, u_2, and u_3 and their projected vertices on layer 1 are inserted into E, respectively (line 15). Finally, after the vertex projection between layers, the vertices on layers 1 and 2 are shown in Fig. 7(c).

Algorithm: Vertex-Projection-within-a-Layer(O, s, V_s)
Input: O /* set of obstacles */
s /* the layer of projection */
V_s /* set of vertices on layer s */
1 **for** each vertex $v \in V_s$
2 $u_1 =$ the closest non-blocked vertex at the left of v
3 $u_2 =$ the closest non-blocked vertex at the back of v
4 $u_3 =$ the closest non-blocked vertex at the right of v
5 $u_4 =$ the closest non-blocked vertex at the front of v
6 **for** $j = 1$ **to** 4
7 **if** u_j exists
8 $(x, y) =$ the x- and y-coordinates of u_j
9 **if** ($s \neq 1$ and $(x, y, s-1)$ is not inside an obstacle) or ($s \neq N_l$ and $(x, y, s+1)$ is not inside an obstacle)
10 $V_s = V_s \cup \{u_j\}$

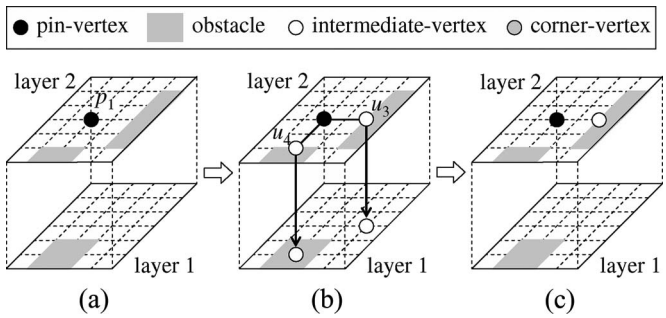


Fig. 8. Example vertex projection within a layer. We focus on the projection of p_1.

3) Vertex Projection Within a Layer: Fig. 8(a) shows an example vertex projection within layer 2. We focus on the projection of p_1, and similar operations are performed for each vertex. As shown in Fig. 8(b), u_3 and u_4 are set because they are the projected vertices of p_1 within layer 2 (lines 2–5). For u_3, because the projected vertex on layer 1 is not inside an obstacle, u_3 is inserted into V_2 (the vertex set of layer 2) (lines 7–10). On the other hand, u_4 is not inserted into V_2 because its projected vertex on layer 1 is inside an obstacle (lines 7–9). Finally, after the vertex projection within layer 2, only u_3 is inserted into V_2 for p_1 as shown in Fig. 8(c).

Algorithm: ML-OASG-Construction(O, P, N_l, n, T_n, G)
Input: O /* the set of obstacles */
P /* the set of pin-vertices */
N_l /* the number of layers */
n /* the size of $P \cup C$ */
T_n /* the threshold for n */
Output: $G = (V, E)$ /* the ML-OASG */
1 $V = \emptyset$
2 $E = \emptyset$
3 **for** $i = 1$ **to** N_l
4 $V_i =$ the set of all pin-vertices and corner-vertices on layer i
5 **do until** G is not changed in the iteration
6 **for** $i = 1$ **to** $N_l - 1$ /* from bottom to top */
7 **if** $n \leq T_n$
8 Vertex-Projection-within-a-Layer(O, i, V_i)
9 Vertex-Projection-between-Layers($O, i, i+1, V_i, V_{i+1}, G$)
10 **for** $i = N_l$ **to** 2 /* from top to bottom */
11 **if** $n \leq T_n$
12 Vertex-Projection-within-a-Layer(O, i, V_i)
13 Vertex-Projection-between-Layers($O, i, i-1, V_i, V_{i-1}, G$)
14 **if** $n > T_n$
15 Go to line 16
16 **for** $i = 1$ **to** N_l
17 $G = G \cup$ SL-OASG-Construction(V_i)
18 Return G

4) ML-OASG Construction: Because there is a tradeoff between the completeness of the ML-OASG and the computational efficiency, we use a parameter T_n as the threshold for the total number of pin and corner vertices n. If $n > T_n$, the iteration at line 5 is performed only once (lines 14–15), and the vertex projection within a layer (lines 8 and 12) is not performed. For each iteration at line 5, there are two passes for the vertex projections—one is a bottom–up pass (lines 6–9), and the other is a top–down pass (lines 10–13). After the vertex projections, we apply the SL-OASG construction in [6] for each layer (lines 16–17).

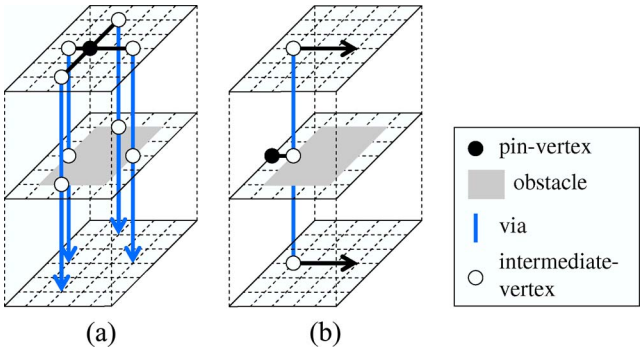


Fig. 9. Two vertex projections provide paths for vertices to escape from an obstacle (a) on the different layer and (b) on the same layer.

We assume V as the vertex set of the ML-OASG, i.e., V includes all vertices in $P \cup C$ and all added vertices by the vertex projections. The two vertex projections provide paths for vertices to "escape" from obstacles as shown in Fig. 9. Thus, by our ML-OASG construction, we can claim that, if $n \leq T_n$, the ML-OASG implies a rectilinear shortest path of any two vertices in V, i.e., a rectilinear shortest path of any two vertices can be obtained by transforming slant edges in the ML-OASG to rectilinear edges. To prove this property, we define the *neighbors* and the *territory* of a vertex as follows.

Definition 5: A vertex $f \in V$ is a *neighbor* of a vertex $v \in V$ if 1) v and f are on the same layer and no other vertex in V or obstacle is inside or on the boundary of the bounding box of v and f, or 2) v and f are on adjacent layers and they have the same x- and y-coordinates.

Note that, since only vias can be used for connections between layers, we use the second condition to strengthen the definition of a neighbor in [6].

Definition 6: A vertex g is in the *territory* of a vertex $v \in V$ if no other vertex in V or obstacle is inside the bounding box of v and g.

Note that g is not necessarily in V, and the territory of a vertex is not necessarily a closed region.

Lemma 1: If $n \leq T_n$, given a source $s \in V$, a target $t \in V$ $(s \neq t)$, 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 a territory in Definition 6, 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 boundary of the territory. Assuming that the intersecting vertex is u, we prove this lemma by the following three cases.

1) s and u are on the same layer, and the following edge from u to t of RSP(s,t) is also on the same layer. Consequently, this case degenerates into the SL one, and thus, this lemma is proved in [6].
2) s and u are on the same layer, and the following edge from u to t of RSP(s,t) is a via. We assume that the following edge from u to t is (u,u'). Without loss of generality, we also assume that u' is on the upper layer of u. If there is no obstacle inside or on the boundary of the bounding box of s and u', the vertex projections must make a vertex $s' \in V$ where s' is on the upper layer of s, and s' is connected to s by a via. On the other hand, if there is an obstacle inside or on the boundary of the bounding box of s and u' (s and the obstacle are on different layers because there is an edge from s to u), the vertex projections must make a vertex $s' \in V$ where s and s' are on the same layer, and there is no obstacle inside or on the boundary of the bounding box of s' and u'. Once s' is decided, $\delta_r(s,u') = \delta_r(s,s') + \delta_r(s',u')$ is trivial, resulting in $\delta_r(s,t) = \delta_r(s,u') + \delta_r(u',t) = \delta_r(s,s') + \delta_r(s',t)$. Finally, s' is trivially a neighbor of s; therefore, this lemma thus follows.
3) s and u are on different layers. Because s and u must have the same x- and y-coordinates, by Definition 6, s and u are on adjacent layers, and u must be in V after the vertex projections. Then, by Definition 5, u is a neighbor of s, i.e., $f = u$. ■

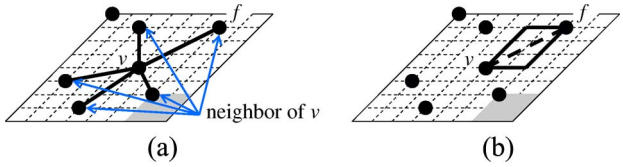


Fig. 10. (a) Because f is a neighbor of v, there must exist an edge between them in the ML-OASG, and (b) the edge can directly be transformed to rectilinear edges with the rectilinear shortest length between v and f.

Lemma 2: If $n \leq T_n$, given a vertex $v \in V$, for any neighbor f of v, there must exist an edge between v and f in the ML-OASG, i.e., a rectilinear shortest path between v and f is implied by the ML-OASG.

Proof: We prove this lemma by the following two cases.

1) v and f are on the same layer. Because the SL-OASG construction in Section IV-A4 exactly constructs an edge between a vertex and its neighbor on the same layer, there must exist an edge between v and f as shown in Fig. 10(a). By the definition of neighbors in Definition 5, since no obstacle is inside the bounding box of v and f, the edge between v and f can directly be transformed to rectilinear edges with the rectilinear shortest length as shown in Fig. 10(b).
2) v and f are on adjacent layers. By the definition of neighbors in Definition 5, they have the same x- and y-coordinates. Therefore, there is an edge constructed by the vertex projection between the two layers in Section IV-A2, and the edge is the rectilinear shortest length between v and f. ■

Theorem 3: If $n \leq T_n$, the ML-OASG implies a rectilinear shortest path of any two vertices in V.

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 $\delta_r(s,t) = \delta_r(s,f) + \delta_r(f,t)$ as shown in Fig. 11(a) and (b). By Lemma 2, a rectilinear shortest path of s and f is implied by the ML-OASG as shown in Fig. 11(c).

As shown in Fig. 11(d), we still need to prove that a rectilinear shortest path of f and t is implied by the ML-OASG to complete the proof. However, because f and t are both in

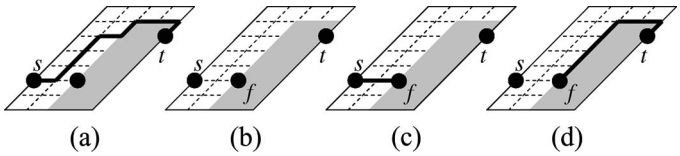


Fig. 11. (a) and (b) For any pair of vertices s and t and any of their rectilinear shortest paths, by Lemma 1, there must exist a neighbor f of s such that the rectilinear shortest length $\delta_r(s,t) = \delta_r(s,f) + \delta_r(f,t)$. (c) By Lemma 2, a rectilinear shortest path of s and f is implied in the ML-OASG. (d) By repeating the proof, a rectilinear shortest path of f and t is implied in the ML-OASG.

V, 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 T_n is a parameter set by the user. Moreover, we also claim that the number of vertices in the ML-OASG is $O(n)$.

Theorem 4: The number of vertices in the ML-OASG is $O(n)$.

Proof: The number of added vertices by each vertex projection is $O(n)$. If $n > T_n$, the vertex projections are performed $O(1)$ times; if $n \leq T_n$, the number of vertices is bounded by T_n. Therefore, the number of vertices in the ML-OASG is $O(n)$. ■

We consider the two cases for the ML-OASG construction for the following reasons.

1) If the number of vertices is small, a rectilinear shortest path between any two vertices is very important. For example, when the number of vertices is 2, a rectilinear shortest path between these two vertices is an optimal solution of the ML-OARSMT problem. Therefore, to make these vertices projected completely, we need both of the projections and perform them until the ML-OASG is not changed.
2) If the number of vertices is large, the computational efficiency should be maintained by reducing the size of the ML-OASG. Therefore, we perform the iteration only once and remove the vertex projection within a layer in this case. When the size of the ML-OASG is reduced, we can improve the efficiency of finding a desired tree topology from the ML-OASG. There is another reason why we remove the vertex projection within a layer. When we transform slant edges into vertical and horizontal edges in Section IV-C, we can use some heuristics to determine the positions of Steiner vertices and maximize the edge overlaps. However, the vertex projection within a layer may result in many rectilinear edges, which significantly decreases the flexibility of transforming slant edges into vertical and horizontal edges. This phenomenon does not happen for the SL-OARSMT problem because the completeness of SL-OASG mainly relies on the slant edges that do not decrease the flexibility of transforming slant edges into rectilinear edges.

From the aforementioned analysis, we have the following observation.

Observation 1: If the number of vertices is large, the computational efficiency can be much improved by removing the vertex projection within a layer, and the effectiveness is almost not affected.

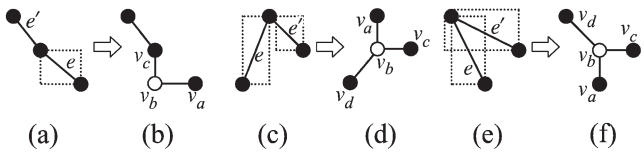


Fig. 12. Three cases in the SL-OARST construction for a slant edge and its neighboring edge. The graphs in (a), (c), and (e) are transformed into those in (b), (d), and (f), respectively.

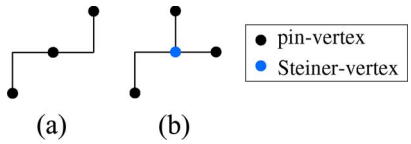


Fig. 13. When $m = 3$, a rectilinear Steiner tree is one of the two topologies: (a) Two simple paths between pin vertices or (b) three pin vertices connected to a single Steiner vertex.

Note that, to deal with rectilinear obstacles, the formulation can be modified so that an edge is not allowed to pass through the boundary between two line-touched obstacles. If the formulation is modified, edges of this type should be removed from the spanning graph, and those properties are still held.

B. ML-OAST Construction

We define an ML-OAST as follows.

Definition 7: An ML-OAST is an undirected tree connecting all pin vertices without intersecting with any obstacle.

We extend the SL-OAST construction in [6] and modify the cost of an edge between v_i and v_j from $|x_i - x_j| + |y_i - y_j|$ to $|x_i - x_j| + |y_i - y_j| + |z_i - z_j| \times C_v$.

C. ML-OARST Construction

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

Definition 8: An ML-OARST is an undirected graph connecting all pin vertices by rectilinear edges within layers and vias between layers without intersecting with any obstacle.

Because we use vias between layers in our ML-OAST, we only need to consider edges within layers. We extend the SL-OARST construction in [6] and modify some cases, trying to maximize edge overlaps so that the total cost can be reduced. Three cases for a slant edge e and its neighboring edge e' need to be considered as shown in Fig. 12. There are two possible transformations for (v_b, v_d) in Fig. 12(d) and (f). Different from [6] which randomly chooses one of them, we decide it in another iteration to further reduce the total cost.

D. ML-OARSMT Construction

In this step, we construct an ML-OARSMT. The definition of a *redundant vertex* in [6] is modified because only vias can be used for connections between layers.

Definition 9: A *redundant vertex* is a two-degree nonpin vertex, and the two edges connecting to it are parallel on a layer.

TABLE I
COMPARISONS ON THE TOTAL COST (WIRELENGTH AND VIA COSTS) AND CPU TIME (SECONDS) BETWEEN THE CC ALGORITHM AND OURS. "Imp." IS THE IMPROVEMENT ON THE TOTAL COST

Test Cases	m / k / N_l	$C_v = 3$					$C_v = 5$				
		CC		Ours		Imp.	CC		Ours		Imp.
		Total Cost (#via)	Time	Total Cost (#via)	Time	(%)	Total Cost (#via)	Time	Total Cost (#via)	Time	(%)
ind1	50 / 6 / 5	82,556 (59)	0.02	55,537 (49)	0.07	32.73	82,674 (59)	0.01	55,635 (49)	0.07	32.71
ind2	200 / 85 / 6	17,568 (293)	0.21	12,512 (224)	3.05	28.78	17,389 (283)	0.22	12,899 (208)	3.01	25.82
ind3	250 / 13 / 10	17,837 (529)	0.32	10,973 (359)	3.32	38.48	17,599 (480)	0.35	11,698 (343)	3.35	33.53
ind4	500 / 100 / 5	273,235 (0)	0.97	77,033 (0)	8.12	71.81	273,235 (0)	0.98	77,033 (0)	8.21	71.81
ind5	1,000 / 20 / 5	23,314,944 (0)	3.28	14,515,511 (0)	45.63	37.74	23,314,944 (0)	3.60	14,515,511 (0)	45.71	37.74
rt1	25 / 10 / 10	5,095 (91)	0.01	4,334 (76)	0.06	14.94	5,277 (91)	0.01	4,486 (76)	0.06	14.99
rt2	100 / 20 / 10	12,885 (290)	0.06	9,434 (215)	0.88	26.78	13,874 (263)	0.06	9,812 (200)	0.90	29.28
rt3	250 / 50 / 10	23,233 (705)	0.41	15,569 (490)	6.86	32.99	22,727 (633)	0.38	16,384 (429)	6.84	27.91
rt4	500 / 50 / 10	29,464 (1,282)	1.45	22,034 (918)	17.52	25.22	30,542 (1,144)	1.61	23,883 (879)	17.56	21.80
rt5	1,000 / 100 / 5	38,702 (1,102)	5.25	27,890 (869)	55.61	27.94	43,775 (1,025)	5.29	29,598 (814)	56.45	40.62
Avg.	—	—	—	—	—	33.74	—	—	—	—	33.62

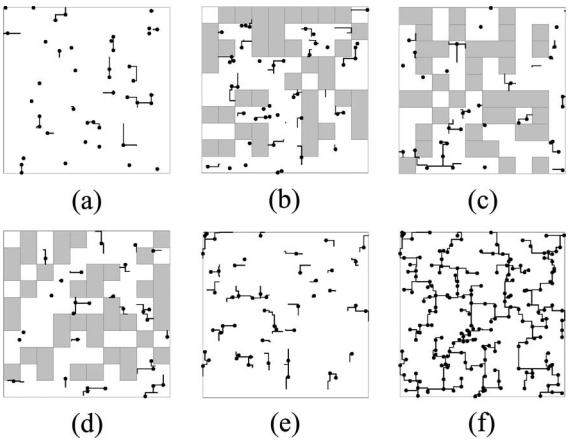


Fig. 14. Final routing result of ind2 under $C_v = 3$, where a pin vertex is represented by a solid circle. (a)–(e) show the results of layers 2–6, respectively. Layer 1 is not shown because there is not any route on it. (f) All pin vertices are projected onto a plane, without showing the obstacles.

For a redundant vertex, we merge the two edges connecting to it. Moreover, we also remove overlapping edges and mark Steiner vertices.

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. Note that the optimality is not guaranteed by using any extended method mentioned in Section IV-A1.

Theorem 5: If $m = 2$ and $n \leq T_n$, our constructed ML-OARSMT is an optimal solution.

Proof: By Theorem 3, the ML-OASG implies a rectilinear shortest path of any two vertices in V. Hence, its corresponding path in the ML-OASG is constructed in Section IV-B, and this rectilinear shortest path is trivially constructed in Sections IV-C and IV-D. ■

Note that T_n is a parameter set by the user. When $m = 3$, a rectilinear Steiner tree is one of the two topologies: two simple paths between pin vertices as shown in Fig. 13(a) or three pin vertices connected to a single Steiner vertex as shown in Fig. 13(b). We can construct an optimal OARSMT for the first topology.

Theorem 6: If $m \geq 3$, $n \leq T_n$, 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 vertices in the ML-OASG. These rectilinear shortest paths are generated for the same reasons in Theorem 5. ■

Most nets in a real case are two- or three-pin nets, which makes the aforementioned properties more important for practical applications. Moreover, when m is very large, the probability that the topology of an optimal solution contains only simple paths between pin vertices is small. Thus, in such a case, T_n does not need to be set too large.

Finally, since the number of vertices after the vertex projection is $O(n)$ from Theorem 4, the time complexity of each step is the same as those in [6]. Therefore, the overall time complexity of our algorithm is $O(n^3)$ in the worst case and $O(n^2 \lg n)$ for practical applications [the expected number of edges in the spanning graph is $O(n \lg n)$], where n is the total number of pin and corner vertices.

V. EXPERIMENTAL RESULTS

We implemented our algorithm in the C/C++ language on a 2.8-GHz AMD-64 machine with 8-GB memory under the Ubuntu 6.06 operating system. There are totally ten benchmark circuits, five test cases (ind1–ind5) from Synopsys and five random test cases (rt1–rt5) generated by us. Their parameters are listed in Table I. Both ind4 and ind5 have two obstacles covering an entire layer, and there is only a layer between them. Therefore, the routing is only allowed on one layer. Considering the ratios of m and k in industrial test cases, we set the ratios of m and k to 2.5, 5, and 10 to generate the five 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.

We set the parameter T_n of our algorithm to 10. Because there is no previous work targeting on this problem, we compared our algorithm with another algorithm, called CC, based on the construction-by-correction approach. It first constructs a 3-D minimum spanning tree for all pin vertices and then transforms slant edges into rectilinear edges to form an initial Steiner tree. Finally, it replaces the edges overlapping obstacles with edges or vias around the obstacles with a smaller cost. We also verified the generated ML-OARSMTs by a verification program to ensure that all pin vertices were connected without intersecting any obstacle.

TABLE II
COMPARISONS ON THE TOTAL COST AND CPU TIME (SECONDS) BETWEEN THE VPWAL ALGORITHM AND OURS

Test Cases	m	k	N_l	VPWAL		Ours		Imp. (%)	Speed-up (X)
				Total Cost	Time	Total Cost	Time		
ind1	50	6	5	55,635	0.10	55,635	0.07	0.00	1.43
ind2	200	85	6	12,976	7.92	12,899	3.01	0.59	2.63
ind3	250	13	10	11,668	7.00	11,698	3.35	-0.26	2.09
ind4	500	100	5	77,090	9.98	77,033	8.21	0.07	1.22
ind5	1,000	20	5	14,512,987	56.10	14,515,511	45.71	-0.02	1.23
rt1	25	10	10	4,491	0.15	4,486	0.06	0.11	2.50
rt2	100	20	10	9,827	2.44	9,812	0.90	0.15	2.71
rt3	250	50	10	16,426	44.52	16,384	6.84	0.26	6.51
rt4	500	50	10	24,074	141.51	23,883	17.56	0.79	8.06
rt5	1,000	100	5	29,740	422.17	29,598	56.45	0.48	7.48
Avg.	—	—	—	—	—	—	—	0.22	3.58

Table I lists the total costs of these algorithms under the conditions $C_v = 3$ and $C_v = 5$. Fig. 14 shows the resulting layout for the test case ind2 under $C_v = 3$. Considering the total cost, the average improvements over CC on the total cost are 33.74% and 33.62% under $C_v = 3$ and $C_v = 5$, respectively. Furthermore, the improvement over the CC algorithm can be up to 71.81% (for ind4). Since an optimal solution gives a lower bound for any solution for this ML-OARSMT problem, these improvements are highly significant. Our algorithm achieves such significant improvements because it is constructed not only on pin and corner vertices but also some essential vertices that lead to more desired solutions. Moreover, the vertex projections provide our ML-OASG with a more global view of obstacles and multilayers.

We also observed that the CC algorithm has a particularly larger total cost when its initial Steiner tree overlaps large obstacles, indicating that the CC algorithm lacks the global view of obstacles. Another observation is that the CC algorithm may have a larger total cost under $C_v = 3$ than that under $C_v = 5$ (for ind2, ind3, and rt3). It is because the value of C_v may change the topology of the initial Steiner tree, resulting in different overlapping conditions with obstacles. Thus, after the correction stage, the results become unstable. This is another evidence to point out that the CC algorithm indeed lacks the global view of obstacles. On the other hand, our algorithm always has a smaller total cost under $C_v = 3$ than that under $C_v = 5$, indicating that it is indeed stable to consider obstacles. Furthermore, there is another reasonable result that our algorithm uses fewer vias when $C_v = 5$, showing that the via cost is also well handled by our algorithm.

Table I also compares the CPU times of these algorithms. Although the CC algorithm is faster, its solution quality is relatively poor. Our algorithm is sufficiently efficient. For example, when the respective numbers of pin vertices and obstacles reach 500 and 100 (ind4), our algorithm takes less than 9 s and achieves 71.81% improvement over the CC algorithm. By the least squares fitting on the log–log axes where the CPU time is plotted as a function of the input size n, the slope of the fitting line is 2.17, implying that the empirical time complexity of our algorithm is close to $O(n^{2.1})$. It is far under the theoretical worst case complexity of $O(n^3)$ and closer to the practical analysis $O(n^2 \lg n)$ in Section IV-D.

There is another experiment where the same test cases are used and C_v is set to 5 (the results are similar if $C_v = 3$ although not presented here). Because we set the parameter T_n of our algorithm to 10, the vertex projection within a layer is not performed for these test cases. On the other hand, the comparative algorithm, called VPWAL, performs the vertex projection within a layer for these test cases. Table II shows the comparisons on the total cost and CPU time. Our algorithm has an average of 3.58 times speedup to achieve a smaller total cost in seven test cases, while the VPWAL algorithm takes much more time but has smaller total cost only in two test cases. This experiment justifies Observation 1 in Section IV-A4 that, if the number of vertices is large, the computational efficiency can be much improved by removing the vertex projection within a layer, and the effectiveness is almost not affected. This is because reducing the size of the ML-OASG is beneficial to the efficiency of finding a good tree topology from the ML-OASG, and the vertex projection within a layer results in many vertical and horizontal edges, which significantly decreases the flexibility of the ML-OARST construction. Note that this phenomenon does not happen for the SL-OARSMT problem because the completeness of SL-OASG mainly relies on the slant edges that do not decrease the flexibility.

Note also that a near-optimal Steiner tree algorithm might perform well if nets overlap with very few (or even no) obstacles. However, there is a tradeoff. Using a near-optimal Steiner tree algorithm may increase the risk of losing the global view of the obstacles. This phenomenon becomes more significant when there are more obstacles on the plane or the sizes of obstacles are large.

VI. CONCLUSION

In this paper, we have formulated the ML-OARSMT problem. We have also identified some special properties of the ML-OARSMT problem and shown the infeasibility of directly extending a previous SL-OARSMT to the ML-OARSMT problem with optimality guarantee. With these properties in mind, we have developed a spanning-graph-based algorithm which can guarantee an optimal solution for any two-pin net and many multiple-pin nets. Experimental results have shown that it is effective and efficient. Future work includes the consideration of preferred directions, the restriction of via locations, and the construction of a timing-driven routing tree.

ACKNOWLEDGMENT

The authors would like to thank Ms. C.-F. Shen and Mr. S.-L. Wang of Synopsys, Taiwan for their valuable help with this paper.

REFERENCES

[1] C. Chiang, M. Sarrafzadeh, and C. K. Wong, "An algorithm for exact rectilinear Steiner trees for switchbox with obstacles," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 39, no. 6, pp. 446–455, Jun. 1992.

[2] 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 λ-geometry plane," in *Proc. ISPD*, 2006, pp. 48–55.

[3] 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.

[4] 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.

[5] 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.

[6] C.-W. Lin, S.-Y. Chen, C.-F. Li, Y.-W. Chang, and C.-L. Yang, "Obstacle-avoiding rectilinear Steiner tree construction based on spanning graphs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 4, pp. 643–653, Apr. 2008.

[7] C.-W. Lin, S.-L. Huang, K.-C. Hsu, M.-X. Lee, and Y.-W. Chang, "Efficient multi-layer obstacle-avoiding rectilinear Steiner tree construction," in *Proc. ICCAD*, 2007, pp. 380–385.

[8] C.-H. Liu, Y.-H. Chou, S.-Y. Yuan, and S.-Y. Kuo, "Efficient multilayer routing based on obstacle-avoiding preferred direction Steiner tree," in *Proc. ISPD*, 2008, pp. 118–125.

[9] J. Long, H. Zhou, and S. O. Memik, "An $O(n \log n)$ edge-based algorithm for obstacle-avoiding rectilinear Steiner tree construction," in *Proc. ISPD*, 2008, pp. 126–133.

[10] 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.

[11] Y. Shi, P. Mesa, H. Yu, and L. He, "Circuit simulation based obstacle-aware Steiner routing," in *Proc. DAC*, 2006, pp. 385–388.

[12] P. C. Wu, J. R. Gao, and T. C. Wang, "A fast and stable algorithm for obstacle-avoiding rectilinear Steiner minimal tree construction," in *Proc. ASP-DAC*, 2007, pp. 262–267.

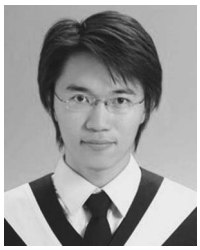
[13] M. C. Yildiz and P. H. Madden, "Preferred direction Steiner trees," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 21, no. 11, pp. 1368–1372, Nov. 2002.

[14] H. Zhou, "Efficient Steiner tree construction based on spanning graphs," in *Proc. ISPD*, 2003, pp. 152–157.

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, in 2005 and 2007, respectively.

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.



Shih-Lun Huang received the double B.S. degree in computer science and electrical engineering from National Tsing Hua University, Hsinchu, Taiwan, in 2006 and the M.S. degree in electronics engineering from National Taiwan University, Taipei, Taiwan, in 2008.

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

Mr. Huang is an honorary member of the Phi Tau Phi Scholastic Honor Society of Taiwan.

Kai-Chi Hsu received the B.S. degree in computer science from National Tsing Hua University, Hsinchu, Taiwan, in 2006 and the M.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, in 2008.

She is currently with the Raydium Semiconductor Corporation, Hsinchu. Her research interests include routing for analog integrated circuits.

Meng-Xiang Lee received the B.S. degree in electrical engineering from the National Central University, Taoyuan, Taiwan, in 2006 and the M.S. degree in electronics engineering from National Taiwan University, Taipei, Taiwan, in 2008.

He is currently in the army for his compulsory military service. His research interests include very large scale integration electronic design automation, large-scale routing, and floorplanning for multiple-supply-voltage designs.

Yao-Wen Chang (S'94–A'96–M'99) 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, Austin, in 1993 and 1996, respectively, all in computer science.

He is currently a Professor with the Department of Electrical Engineering and the Graduate Institute of Electronics Engineering, National Taiwan University. He is also currently a Visiting Professor with Waseda University, Kitakyushu, Japan. He was with the IBM T. J. Watson Research Center, Yorktown Heights, NY, in the summer of 1994, and the faculty of National Chiao Tung University, Hsinchu, Taiwan, from 1996 to 2001. His current research interests include VLSI physical design, design for manufacturability/reliability, and design automation for biochips. He has coauthored one book on routing and more than 140 ACM/IEEE conference proceeding/journal papers and has been working closely with the industry on projects in these areas.

Dr. Chang is a member of the IEEE Circuits and Systems Society, ACM, and ACM/SIGDA. He is an Editor for the *Journal of Information Science and Engineering*. He is currently an Associate Editor for the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS. He currently serves on the ICCAD Executive Committee, ACM/SIGDA Physical Design Technical Committee, and the organizing committees of ISPD and FPT. He 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, VLSI-DAT (Topic Co-Chair), etc. He is currently an independent board director of Genesys Logic, Inc. and a member of the board of governors of Taiwan IC Design Society. He was a winner of the 2006 ACM ISPD Placement Contest and the 2008 ACM ISPD Global Routing Contest; Best Paper Awards at ICCD-95 and the 2007 and 2008 VLSI Design/CAD Symposia; and 12 Best Paper Award Nominations from DAC (four times), ICCAD (twice), ISPD (three times), ACM TODAES, ASP-DAC, and ICCD. He has also been the recipient of many awards for research performance, such as the 2007 Distinguished Research Award, the inaugural 2005 First-Class Principal Investigator Award, the 2004 Dr. Wu Ta You Memorial Award from the National Science Council of Taiwan, and the 2004 MXIC Young Chair Professorship from the MXIC Corp.; and for excellent teaching from National Taiwan University (four times) and National Chiao Tung University.