# Efficient Multilayer Obstacle-Avoiding Rectilinear Steiner Tree Construction Based on Geometric Reduction

Chih-Hung Liu, Chun-Xun Lin, I-Che Chen, D. T. Lee, *Fellow, IEEE,* and Ting-Chi Wang

***Abstract*—Given a set of pin-vertices, an obstacle-avoiding rectilinear Steiner minimal tree (OARSMT) connects all the pin-vertices possibly through Steiner points using vertical and horizontal segments with the minimal wirelength and without intersecting any obstacle. To deal with multiple routing layers and preferred routing orientations, we consider the multilayer OARSMT (ML-OARSMT) problem and the obstacle-avoiding preferred direction Steiner tree (OAPD-ST) problem. First, we prove that the multilayer case is theoretically different from the 2D one, and propose a reduction to transform a multilayer instance into a 3D instance. Based on the reduction, we apply computational geometry techniques to develop an efficient algorithm, utilizing existing OARSMT heuristics, for the ML-OARSMT problem and the OAPD-ST problem. Furthermore, we develop an advanced Steiner point selection to avoid inferior Steiner points and to improve the solution quality. Experimental results show that our algorithm provides a solution with excellent quality and has a significant speed-up compared to previously known results.**

***Index Terms*—Obstacle-avoidance, physical design, rectilinear Steiner minimal tree (RSMT), routing.**

## I. Introduction

THE RECTILINEAR Steiner minimal tree (RSMT) problem is a fundamental research topic in very large scale integration (VLSI) layout design since the routes for signal nets are usually represented by rectilinear Steiner trees. However, as the manufacturing technology significantly increases the density of a chip, a modern IC design contains more and more hard IP cores, macro blocks, and prerouted nets, which are referred to as obstacles in the routing process. Therefore, the obstacle-avoiding RSMT (OARSMT) problem has become very important and received lots of attention recently [2]–[10].

In addition, the advance of IC technology also offers an abundance of routing layers to provide more routing resources. Therefore, Lin *et al.* [11] first formulated the multilayer OARSMT (ML-OARSMT) problem and then extended the spanning graph in [2] to construct a multilayer spanning graph and developed an effective algorithm.

Moreover, considering signal integrity and IC manufacturing, the orientation of routing in a single layer tends to be the same (either horizontal or vertical), and thus *preferred directions* are assigned to each routing layer. Liu *et al.* [12] formulated the obstacle-avoiding preferred direction Steiner tree (OAPD-ST) problem, and developed an $O(n^2 \log n)$-time algorithm based on an $O(n^2)$-space routing graph which contains at least one optimal solution. Recently, Chuang and Lin [13] proposed an effective algorithm which does not construct a routing graph.

According to the experimental results in [11]–[13], those algorithms still behave like a quadratic-time algorithm in practice. In our opinion, the inefficiency of the algorithms in [11] and [12] results from the quadratic size of a routing graph, and the inefficiency of the algorithm in [13] results from their backtracking procedure of finding Steiner points. Since the Steiner tree construction will be invoked many times during the physical design stage, it is necessary to develop a more efficient algorithm for the ML-OARSMT and OAPD-ST problems for practical applications.

The obstacle-avoidance shortest path problems were of major interest in computational geometry in the 1980s and the 1990s, and have been well-studied in both the 2D plane and the 3D space. In this situation, it is probably beneficial to apply those computational geometry techniques to the VLSI routing (e.g., Liu *et al.*'s [6] OARSMT algorithm).

We, therefore, employ computational geometry techniques to develop an efficient algorithm for the ML-OARSMT problem and the OAPD-ST problem, which not only gives excellent solutions, but also behaves like a subquadratic-time algorithm for practical applications.

We first prove that the size of an ML-OARMST is $\Omega(n^2)$, indicating that the multilayer model is very different from its 2D counterpart. Hence, we propose a reduction which

Manuscript received December 30, 2013; revised April 8, 2014 and August 19, 2014; accepted September 22, 2014. This work was supported by the Ministry of Science and Technology under Grant MOST 103-2220-E-007-001, MOST 103-2221-E-005-042, and MOST 103-2221-E-005-043. A preliminary version of this paper was presented at the 2012 ACM/EDAC/IEEE Design Automation Conference (DAC) [1]. This paper was recommended by Associate Editor E. Young.

C.-H. Liu is with the Department of Computer Science, University of Bonn, Bonn 53175, Germany (e-mail: chliu@uni-bonn.de).

C.-X. Lin and I.-C. Chen are with the Research Center for Information Technology Innovation, Academia Sinica 115, Taipei, Taiwan.

D. T. Lee is with the Department of Computer Science and Engineering, National Chung Hsing University, Taichung 402, Taiwan, and also with the Institute of Information Science and Research Center for Information Technology Innovation, Academia Sinica 115, Taipei, Taiwan (e-mail: dtlee@dragon.nchu.edu.tw).

T.-C. Wang is with the Department of Computer Science, National Tsing Hua University, Hsinchu 300, Taiwan.

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.2014.2363390

transforms a multilayer instance into a 3D instance and thus enables us to employ computational geometry techniques.

Based on the reduction, we compute the visibility graph as defined in [14] and utilize the Steiner-point based framework in [6] to develop a four-step algorithm for the ML-OARSMT problem. Specifically, we adopt the algorithm in [14] to construct multilayer visibility graph (ML-VG). We then combine the concept of Kruskal algorithm in [15] to select good Steiner points from the ML-VG. After that, we take those selected Steiner points to generate a Steiner tree, and finally, we refine the generated tree to reduce the total cost.

Furthermore, we make use of the geometric transformation in [16] to extend our ML-OARSMT algorithm and make it amenable for the OAPD-ST problem.

Last but not least, we also propose an advanced Steiner point selection which potentially selects more proper Steiner points but does not increase the time complexity. Precisely, an update operation in the advanced selection adjusts the current connected components after selecting a Steiner point, and thus avoids selecting inferior Steiner points. More importantly, our algorithm with the advanced selection guarantees an optimal solution of four pin-vertices for the RSMT problem, the OARSMT problem, and the OAPD-ST problem through Hanan Grid [22], Escape graph [23], and preferred direction evading graph (PDEG) [12], respectively. Besides, any four-leaf subtree in a generated solution on those graphs is optimal, which supports the stability of the solution quality.

Compared with the Steiner point selection in [4] and [6], our advanced Steiner point selection adopts the concept of Kruskal algorithm [15] instead of Prim algorithm [15], and performs a novel update operation to avoid selecting an inferior Steiner point, which guarantees the optimal solution for four pin-vertices on certain routing graphs.

Experimental results show that our algorithm provides a solution with excellent quality and has a significant speed-up compared to previously known results. Compared with [11], our algorithm for the ML-OARSMT problem improves the solution quality in terms of total cost by 2.95% on the average, and also achieves 47.89 times speed-up on the average at the same time. Compared with [12], our algorithm for the OAPD-ST problem improves the solution quality in terms of total cost by 6.76% on the average, and also achieves 17.20 times speed-up on the average at the same time. By least squares fitting analysis, the empirical time complexity of our algorithm is $\Theta(n^{1.18})$ and $\Theta(n^{1.15})$ for the ML-OARSMT problem and the OAPD-ST problem, respectively, indicating our algorithm behaves as a loglinear-time algorithm.

Experimental results also show that our advanced Steiner point selection only takes little overhead in run time to achieve reasonable improvement in wirelength. For the PDEG in [12], it reduces the wirelength by 0.44% on the average with 13% more run time, and for our proposed preferred direction visibility graph (PD-VG) in Section IV-B, it reduces the wirelength by 0.18% on the average with 7% more run time.

We summarize our main contributions as follows.

1) We propose a 3D reduction to transform a multilayer instance into a 3D instance such that 3D computational geometry techniques can be employed.
2) Based on the 3D reduction, we apply computational geometry techniques in [14] and [16] to construct ML-VG and PD-VG, which leads to the high efficiency and high effectiveness of our algorithm.
3) We adopt the four-step Steiner-point-based framework in [6] to develop an efficient algorithm for the ML-OARSMT problem and the OAPD-ST problem. Except step 3, the content of the other three steps is new.
4) We develop an advanced Steiner point selection, which avoids selecting inferior Steiner points and guarantees certain optimality on several routing graphs, leading to more stable solution quality.

The rest of this paper is organized as follows. Section II formulates the two problems. Section III presents the theoretical results; Section IV describes the algorithm, and Section V gives the advanced Steiner point selection. Section VI shows the experimental results, and Section VII makes the conclusion.

## II. Problem Formulation

An obstacle is a rectangle in a layer. Any two obstacles cannot overlap with each other except the boundary. A pin-vertex is a vertex in any layer which will be connected in a signal net. No pin-vertex can be inside any obstacle except on the boundary. A via is an edge connecting two points $(x, y, z)$ and $(x, y, z+1)$. Each endpoint of a via must not be inside an obstacle except on the boundary.

We assume that the costs of vias are the same, and the unit cost of a wire is identical in all the layers. Let $P = \{p_1, p_2, \ldots, p_m\}$ be the set of pin-vertices in an $m$-pin net, $O = \{o_1, o_2, \ldots, o_k\}$ be a set of $k$ obstacles, $n$ be the size of $P \cup \{corners\ in\ O\}$ ($n \leq m + 4 * k$), $N_l$ be the number of routing layers, and $C_v$ be the cost of a via. Since $N_l$ is a small constant in practice, $n$ is the input size.

### A. ML-OARSMT Problem

Given a via cost $C_v$, a set $P$ of pin-vertices, a set $O$ of obstacles, and $N_l$ routing layers, the ML-OARSMT problem is to construct a rectilinear routing tree $T$ connecting all the pin-vertices in $P$ possibly through some additional points (called Steiner points) such that no tree edge intersects the interior of any obstacle in $O$ and the total cost is minimized, where the cost of $T$, cost($T$), is defined to be the total wirelength + $C_v$ $*$ the number of vias.

### B. OAPD-ST Problem

The OAPD-ST problem is a more restricted version of the ML-OARSMT problem in which the orientation of a tree edge must follow the preferred direction (PD) constraints: only vertical edges are allowed in odd number layers, and only horizontal edges are allowed in even number layers. Hereafter, all the mentioned trees/distances/edges/paths are obstacle-avoiding and rectilinear.

## III. Theoretical Results

In Section III-A, we prove that the complexity of the ML-OARMST (Definition 1) is $\Omega(n^2)$, which indicates that a multilayer model is very different from its 2D counterpart and

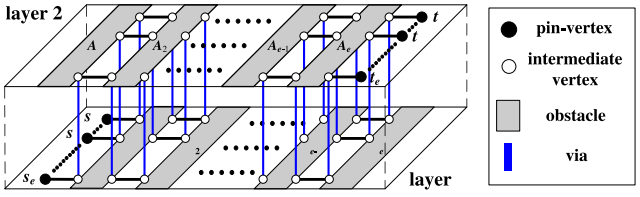



Fig. 1. A worst-case ML-OARMST instance. $P$ is $\{s_1, \ldots, s_e, t_1, \ldots, t_e\}$, and the shortest path between $s_i$ and $t_i$ represents a necessary "edge" of the ML-OARMST of $P$.

motivates us to consider 3D techniques for the ML-OARSMT problem. In Section III-B, we propose a 3D reduction technique to transform a multilayer instance into a 3D one (Theorem 2) such that 3D computational geometry techniques can be employed for the ML-OARSMT problem.

### A. Lower Bound of an ML-OARMST

*Definition 1:* Given an ML-OARSMT problem instance, a ML-OARMST is a tree connecting all the pin-vertices in $P$ using a set of shortest paths between pairs of those pin-vertices such that the sum of costs of those paths is the minimum.

In Fig. 1, there are $m = 2e$ pin-vertices ($s_i = p_{2i-1}$ and $t_i = p_{2i}$ for $1 \le i \le e$) and $k = 2e$ obstacles ($A_i = o_{2i-1}$ and $B_i = o_{2i}$ for $1 \le i \le e$). Thus, $n = m + 4k = 10e$. For $1 \le i \le e$, there exists only one shortest path between $s_i$ and $t_i$, and this path consists of at least $2e$ segments. For two points $p$ and $q$, let $d(p, q)$ be the length of the obstacle-avoiding rectilinear shortest path between them. We assume $d(s_i, t_i)$ for $1 \le i \le e$ is smaller than $d(s_j, s_{j+1})$ and $d(t_j, t_{j+1})$ for $1 \le j \le e - 1$. In other words, if a shortest path between two pin-vertices is viewed as an edge, the $e$ shortest paths between $s_i$ and $t_i$ for $1 \le i \le e$ are the $e$ edges of least cost.

According to the cut property of minimum spanning trees [15], an edge crossing a cut with minimum cost must belong to a minimum spanning tree. Therefore, the $e$ shortest paths between $s_i$ and $t_i$ for $1 \le i \le e$ must be included in the ML-OARMST. Since each of those paths has at least $2e$ segments and $e = n/10$, we conclude the following theorem.

*Theorem 1:* The size of an ML-OARMST is $\Omega(n^2)$.

Theorem 1 indicates that the multilayer model is very different from its 2D counterpart for the reason that the size of a 2D OARMST is linear. Moreover, Theorem 1 also shows that it is impossible to develop a worst-case subquadratic-time algorithm for the ML-OARSMT problem which first constructs an ML-OARMST and then improves it into ML-OARSMT.

### B. 3D Reduction

We first observe that a rectangle in the multilayer model can be viewed as a specific rectilinear box in the 3D space. For an obstacle $o$ in the multilayer model, no path can go through the region between $o$ and the projections of $o$ in adjacent layers. For instance, in Fig. 2(a), there is an obstacle $o_1$, rectangle $ABCD$, in layer 2, and the projection of $o_1$ in layer 1 is rectangle $A_1B_1C_1D_1$. No path can go through the interior of box $ABCDD_1A_1B_1C_1$, but a path could go through each face of this box except rectangle $ABCD$. Similarly, we have box

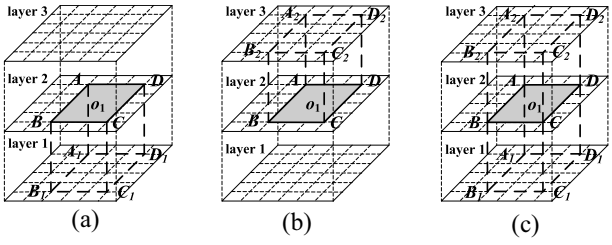



Fig. 2. Examples for the properties of an obstacle. Projection in (a) projection in layer 1. and (b) projection in layer 3. (c) Union of (a) and (b).

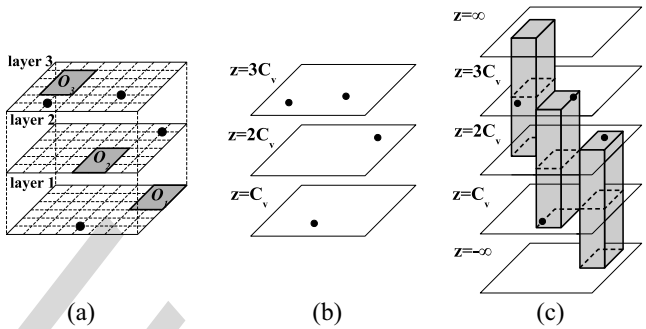



Fig. 3. Reduction from a three-layer instance $I$ to a 3D instance $R(I)$. (a) Three-layer instance $I$. (b) Transformed pin-vertices. (c) Transformed obstacles from layer 1 and layer $N_l$.

$ABCDD_2A_2B_2C_2$ as shown in Fig. 2(b). By merging the two boxes, we have box $A_2B_2C_2D_2D_1A_1B_1C_1$ in Fig. 2(c) such that no path can go through the interior of this box, but a path can go through each face of this box.

Based on the observation, we propose a reduction $R(\cdot)$ to transform a multilayer instance $I$ into a 3D instance $R(I)$.

1) For each pin-vertex $p = (x, y, l)$ in $P$, $R(p)$ is $(x, y, l * C_v)$ as shown in Fig. 3(b).
2) For each obstacle $o$ in $O$, let $l$ be the layer number of $o$. If $2 \le l \le N_l - 1$, $R(o)$ represents a rectilinear box constructed by projecting $o$ to $z = (l + 1) * C_v$ and $z = (l-1) * C_v$, and then connecting the vertices of those two projections using line segments parallel to the $z$-axis. For instance, $o_2$ in layer 2 in Fig. 3(a) is transformed into a box between $z = C_v$ and $z = 3 * C_v$ in Fig. 3(c). If $l = 1$, $R(o)$ represents a rectilinear box constructed by applying the above method on $z = -\infty$ and $z = 2 * C_v$; if $l = N_l$, $R(o)$ represents a rectilinear box by applying the above method on $z = (N_l - 1) * C_v$ and $z = \infty$. For instance, $o_1$ and $o_3$ in Fig. 3(a) are transformed into two boxes in Fig. 3(c). The purpose of projections on $z = -\infty$ and $z = \infty$ is to prevent shortest paths from passing above layer $N_l$ or below layer 1.

Let $R(P)$ be $\{R(p)|p \in P\}$ and $R(O)$ be $\{R(o)|o \in O\}$. If several boxes in $R(O)$ intersect, we combine them into a 3D rectilinear obstacle. For $1 \le i \le N_l$, let $Z_i$ denote plane $z = i * C_v$, and let $Z_{-\infty}$ and $Z_{\infty}$ denote planes $z = -\infty$ and $z = \infty$ respectively. The following theorem can be simply proved from the reduction $R(\cdot)$.

*Theorem 2:* There exists an ML-OARSMT $T$ in a multilayer instance $I$ if and only if there exists a 3D-OARSMT $T'$ in $R(I)$ such that $\text{cost}(T) = \text{cost}(T')$.

*Proof:*

*Necessity:* For each edge $e = (u_1, u_2)$ in $I$, let $R(e)$ be $(R(u_1), R(u_2))$. It is clear that for each edge $e$ of $T$, $R(e)$ is valid in $R(I)$ and $\text{cost}(e) = \text{cost}(R(e))$. Let $R(T)$ be $\{R(e)|$ an edge $e \in T\}$. Hence, there exists a 3D-OARSMT $T' = R(T)$ in $R(I)$ such that $\text{cost}(T') = \text{cost}(T)$.

*Sufficiency:* Let $R^{-1}(\cdot)$ be the inverse function of $R(\cdot)$ such that for each point $p$ in $I$, $R^{-1}(R(p)) = p$. If $R^{-1}(T')$ is valid in $I$, $R^{-1}(T')$ is directly $T$. Otherwise, we move parts of $T'$ to construct another 3D-OARSMT $T''$ without increasing the wirelength such that $\text{cost}(T') = \text{cost}(T'')$ and $T = R^{-1}(T'')$. Since each pin-vertex of $R(I)$ is located on $Z_i$ where $1 \le i \le N_l$, $T'$ has no edge higher than $Z_{N_l}$ or lower than $Z_1$; otherwise, $T'$ can be refined to reduce the wirelength, contradicting that $T'$ is minimized. Hence, if $T'$ has some "invalid" edges which make $R^{-1}(T')$ invalid, each of them must be between $Z_i$ and $Z_{i+1}$ where $1 \le i \le N_l - 1$.

Let a $z$-component be a maximal connected component whose vertices share the same $z$-coordinate. Those "invalid" edges of $T'$ can be grouped into several $z$-components. For each invalid $z$-component $C$ of $T'$, the number of incident edges higher than $C$ must be equal to the number of incident edges lower than $C$; otherwise, moving $C$ along the direction in which the number of incident edges is larger will reduce the wirelength, contradicting that $T'$ is minimized.

Therefore, we can move each $z$-component of $T'$ located between $Z_i$ and $Z_{i+1}$ to $Z_{i+1}$ without increasing the wirelength and finally transform $T'$ into $T''$ such that $R^{-1}(T'')$ is valid and $\text{cost}(T'')$ is equal to $\text{cost}(T)$. Those movements will not be denied by any obstacle since if an obstacle in $R(I)$ will deny the movement of a $z$-component located between $Z_i$ and $Z_{i+1}$, the bottom face of this obstacle must be located between $Z_i$ and $Z_{i+1}$, which violates the reduction $R(\cdot)$. As a result, there exists an ML-OARSMT $T = R^{-1}(T'')$ in $I$ such that $\text{cost}(T) = \text{cost}(T')$. ■

## IV. Algorithm

### A. ML-OARSMT Problem

Liu *et al.* [6] adopted a fact that an optimal OARSMT is an OARMST connecting all pin-vertices and appropriately selected Steiner points to propose a Steiner-point-based framework for the OARSMT problem: 1) graph construction; 2) Steiner point selection; 3) minimum terminal spanning tree (MTST) construction (Definition 3); and 4) refinement. We follow their framework to develop a four-step ML-OARSMT algorithm (summarized in Fig. 4). Except the MTST construction, the content of the other three steps is new.

*1) ML-VG Construction:* To deal with shortest paths among rectilinear obstacles in the 3D space, Clarkson *et al.* [14] proposed a 3D visibility graph (3D-VG) which embeds a shortest path for each pair of vertices in the set of pin-vertices and obstacle corners. Based on our reduction, we use the algorithm in [14] to construct the ML-VG $G(V, E)$ for a multilayer instance $I$. In detail, we first compute the 3D-VG $G'(V', E')$ for $R(I)$, and then delete vertices in $Z_\infty$ and $Z_{-\infty}$ from $G'$ to construct $G$. The 3D-VG algorithm can handle rectilinear obstacles each of which is a union of intersecting boxes, and thus is feasible for our reduction.

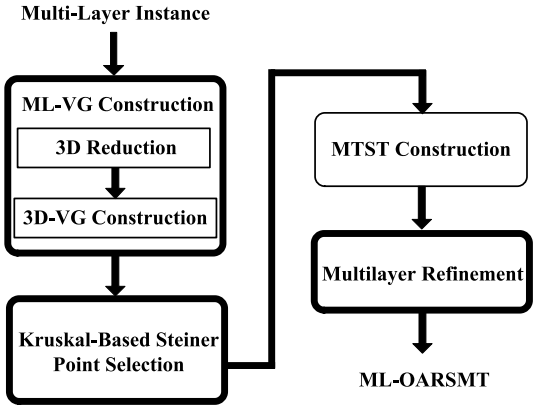



Fig. 4. Flow of our algorithm.

The 3D-VG construction is summarized as follows.

a) Let $U$ be the union of $R(P)$ and the obstacle corners in $R(O)$. Let $(x_1, x_2, \ldots, x_{|U|})$, $(y_1, y_2, \ldots, y_{|U|})$, and $(z_1, z_2, \ldots, z_{|U|})$ be the $x$, $y$, and $z$-coordinates of the vertices in $U$. Let $V_I$ be the set of vertices obtained by intersecting the edges of obstacles in $R(O)$ with planes $x = x_1, \ldots, x = x_{|U|}$, $y = y_1, \ldots, y = y_{|U|}$, and $z = z_1, \ldots, z = z_{|U|}$. Let $V_S$ be $U \cup V_I$.

b) Let $z_m$ be the median of the $z$-coordinates of vertices in $V_S$, and $P_{zm}$ be the plane $z = z_m$. For each vertex $v = (x, y, z)$ in $V_S$, create a vertex $u = (x, y, z_m)$ if the line segment $\overline{vu}$ does not intersect any obstacle. Apply the 2D-VG algorithm in [14] on $P_{zm}$ to create essential vertices.

c) Let $V_H$ and $V_L$ be the set of vertices in $V_S$ higher than $P_{zm}$ and the set of vertices in $V_S$ lower than $P_{zm}$, respectively.

d) Repeat Steps 2 and 3 on $V_H$ and $V_L$ separately until $V_H$ and $V_L$ are empty.

e) Let $V'$ be the union of $V_S$ and the vertices created during Step 2–4. For each pair of two vertices $v, u \in V'$, if: 1) $\overline{vu}$ is rectilinear and 2) $\overline{vu}$ does not intersect any obstacle in $R(O)$ or any vertex in $V'$, add $\overline{vu}$ to $E'$.

Fig. 5 illustrates the ML-VG construction. First Fig. 5(a) is an ML-OARSMT instance, and Fig. 5(b) transforms Fig. 5(a) into a 3D instance. Then Fig. 5(c) generates the set $V_S$, and Fig. 5(d) projects $V_S$ on $P_{zm}$. Finally, Fig. 5(e) constructs the 2D-VG on each layer, and Fig. 5(f) connects all the edges.

*Theorem 3:* The ML-VG has $O(|V_S| \log n)$ vertices and edges, and can be constructed in $O(|V_S| \log^2 n)$ time, where $|V_S|$ is $O(n^2)$ in the worst case.

*Proof:* Clarkson *et al.* [14] had shown that the 3D-VG has $O(|V_S| \log^2 |V_S|)$ vertices and edges, and can be constructed in $O(|V_S| \log^3 |V_S|)$ time. One $O(\log |V_S|)$ factor of the number of vertices, the number of edges, and the time complexity results from the depth of recursion for Steps 2–4 of the 3D-VG construction. Since the number of distinct $z$-coordinates in $R(P) \cup \{\text{obstacle corners in } R(O)\}$ is at most $N_l + 2$ according to the reduction, the depth of recursion is also at most $N_l + 2$. As a result, we can remove one $O(\log |V_S|)$ factor

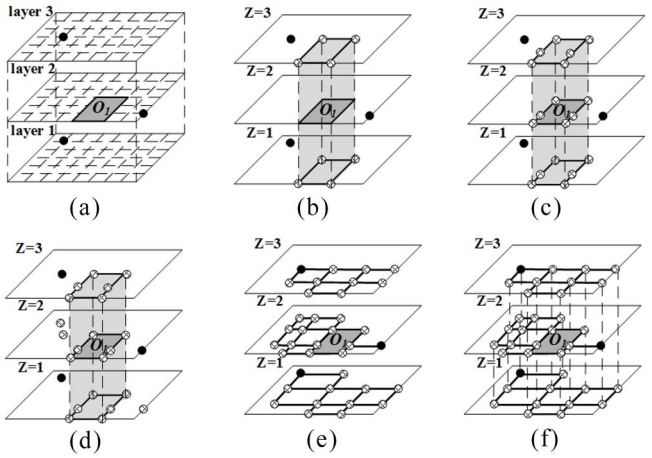



Fig. 5. Example for the ML-VG construction, where $C_v = 1$. (a) A multilayer instance. (b) 3D Reduction. (c) $V_S$ generation. (d) Recursive projection on $P_{zm}$. (e) 2D-VG construction on each layer. (f) Edge connection (ML-VG).

from the number of vertices, the number of edges, and the time complexity since $N_l$ is a small constant under the problem formulation in Section II.

Since there are $O(n)$ vertices in $R(I)$, the $O(n)$ coordinates of those vertices will make $O(n)$ axis-parallel planes. Therefore, since there are $O(n)$ boxes in $R(I)$, there are $O(n^2)$ intersections between those axis-parallel planes and the edges of those boxes, implying that $|V_S|$ is $O(n^2)$ in the worst case. ■

*2) Steiner Point Selection:* We will select Steiner points from the ML-VG. Liu *et al.* [6] used the concept of Prim algorithm [15] to select Steiner points, i.e., they started from one pin-vertex to find Steiner points until all pin-vertices are reached. However, the choice of the starting pin-vertex will affect the quality of selected Steiner points. Therefore, in order to obtain a stable result, we use the concept of Kruskal algorithm [15] to simultaneously start from all pin-vertices and to select Steiner points.

*Definition 2:* For two vertices $v$ and $u$ in a graph $G(V, E)$, their shortest path component $\mathrm{SPC}(v, u)$ is the minimal subgraph of $G$ such that $\mathrm{SPC}(v, u)$ contains all the shortest paths between $v$ and $u$ in $G$ [6].

For the ML-VG $G(V, E)$ with a set $P \subset V$ of pin-vertices, our Kruskal-based Steiner point selection is stated as follows.

1) Initialize each $p_i \in P$ as a connected component $C_i$.
2) Find the nearest pair of connected components $C_i$ and $C_j$. Let the shortest path between $C_i$ and $C_j$ be from a vertex $s_i \in C_i$ and a vertex $s_j \in C_j$.
3) Construct $\mathrm{SPC}(s_i, s_j)$ and combine $C_i$, $\mathrm{SPC}(s_i, s_j)$, and $C_j$ into one connected component. If $s_i \notin P$ ($s_j \notin P$), select $s_i$ ($s_j$) as a Steiner point.
4) Repeat Steps 2 and 3 until there exists only one connected component.

We use the Dijkstra shortest path algorithm [15] to implement the selection. We view the vertices of all the connected components as sources, i.e., their weights are zero, and then manipulate the priority queue to find the nearest source for each vertex in $V$. For each edge $(u, v) \in E$, let $s_1$ and $s_2$ be the nearest sources of $u$ and $v$, respectively. If $s_1$ and $s_2$ belong

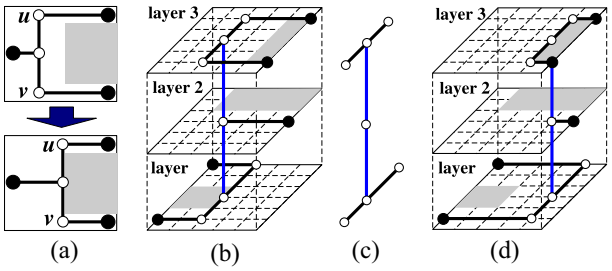



Fig. 6. (a) A U-shaped pattern. (b)–(d) A multilayer pattern.

to different components, $(u, v)$ corresponds to a path between $s_1$ and $s_2$ whose length is $d(s_1, u)+|(u, v)|+d(v, s_2)$. Finding the shortest one of such "paths" will determine the nearest pair of two connected components. In particular, when Step 2 is repeated, we directly set the vertices of $\mathrm{SPC}(s_i, s_j)$ as new sources without reinitializing the whole priority queue, which increases the efficiency.

In order to construct $\mathrm{SPC}(s_i, s_j)$, we find all the edges each of which corresponds to a shortest path between $s_i$ and $s_j$, and then backtrack from the endpoints of those edges to compute $\mathrm{SPC}(s_i, s_j)$, which is similar to the backtracking procedure for a shortest path region in [6]. Besides, we use the operations of disjoint sets in [15] to maintain the relationships between vertices and connected components.

*3) MTST Construction:*

*Definition 3:* For a graph $G(V, E)$ and a terminal set $S \subseteq V$, a MTST connects all the vertices in $S$ using a set of terminal paths among vertices in $S$ such that the sum of lengths of those terminal paths is minimized, where a terminal path is a path in $G$ between two vertices in $S$ passing through vertices in $V \setminus S$ [3].

We view all the pin-vertices and selected Steiner points as terminals and thus construct the MTST of the ML-VG as an initial solution. Mehlhorn [17] proposed an $O(|E| + |V| \log |V|)$-time MTST algorithm. By Theorem 3, this step takes $O(n^2 \log^2 n)$ time in the worst case.

*4) Refinement:* We refine the initial solution in Section IV-A3 to reduce more wirelength. For the OARSMT problem, the traditional way is to move the U-shaped patterns [2] as shown in Fig. 6(a). In general, a U-shaped pattern results from a line segment with different numbers of edges incident on its two sides, e.g., $\overline{uv}$ in Fig. 6(a).

However, in the multilayer model, line segments in different layers may affect each other. Therefore, we replace the line segment of a 2D U-shaped pattern with a collection of line segments.

*Definition 4:* For a routing tree $T$ in a multilayer instance, a preferred direction component (PDC) is a maximal connected component of $T$ whose elements either share the same $x$-coordinate or share the same $y$-coordinate. For example, Fig. 6(c) shows a PDC of Fig. 6(b). A PDC is also known as a consecutive line segment in [12].

Since the PDC in Fig. 6(c) is vertical, it is contained in a vertical plane, and its two sides are referred to as the two sides of the vertical plane. As shown in Fig. 6(b)–(d), if a PDC has a different number of edges incident on its two sides,

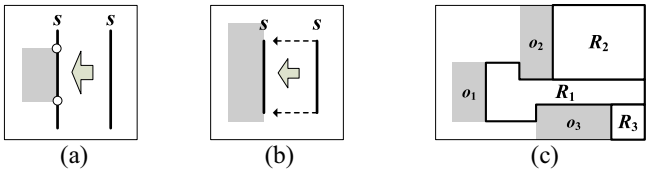



Fig. 7. Segment dragging. (a) $s$ touches at least a corner of an obstacle. (b) $s$ only touches the boundary of an obstacle. (c) A planar subdivision for the segment dragging query of (b).

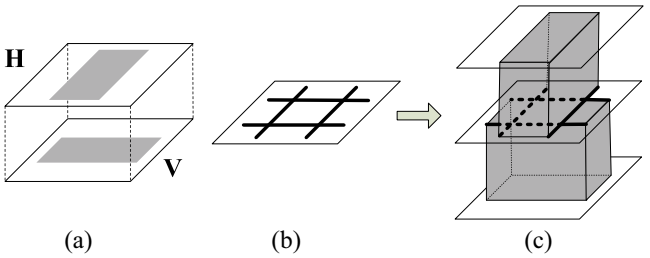



Fig. 8. (a) A two-layer OAPD-ST instance $I$. (b) The transformed one-layer from (a) in [16]. (c) $R'(I)$.

moving this PDC may possibly reduce the wirelength. Hence, we greedily move a PDC to reduce the wirelength until no PDC can be moved to reduce the wirelength.

In order to move a PDC, it is very important to decide the possible moving offset, and the possible moving offset depends on the closest obstacle in the moving direction. Determining the closest obstacle of a PDC is equivalent to determining the closest obstacles of its segments and vertices and selecting the closest one. We employ computational geometry techniques to determine the closest obstacle of a segment, and similarly of a vertex. For simplicity, we only describe how to compute the left closest obstacle of a vertical segment, and similarly for the other cases.

The left closest obstacle of a segment is exactly the first-touched obstacle by dragging this segment leftward. As shown in Fig. 7(a) and (b), there are two cases in dragging a segment $s$ to touch an obstacle: 1) $s$ touches at least a corner of an obstacle and 2) $s$ only touches the boundary of an obstacle. As a result, the left closest obstacle of a segment can be determined by the left closest obstacle corner and the left closest obstacle boundary.

For the first case, Chazelle [18] proposed a data structure which can answer the first-touched point for each orthogonal segment dragging query in $O(\log n)$ time after $O(n \log n)$-time preprocessing. Therefore, after preprocessing all the obstacle corners, we can solve the first case in $O(\log n)$ time. For the second case, it is clear that the closest obstacle boundary belongs to the closer one of the first-touched obstacles by shooting a ray from the two endpoints of the segment. As shown in Fig. 7(c), we can partition the plane into a subdivision such that each obstacle $o_i$ is associated with a region $R_i$ and $o_i$ is the first-touched obstacle if we shoot leftward a ray from an arbitrary point in $R_i$. Such a planar subdivision can be constructed by a simple plane-sweep algorithm within $O(n \log n)$ time. Moreover, since a point location query in a planar subdivision can be answered in $O(\log n)$ time after $O(n \log n)$-time preprocessing [19], the second case can also be solved in $O(\log n)$ time.

To conclude, we can find the closest obstacle of a segment in $O(\log n)$ time after $O(n \log n)$-time preprocessing. For a PDC consisting of $i$ elements, we can determine the possible moving offset in $O(i \log n)$ time.

*5) Time Complexity:*

*Theorem 4:* Our ML-OARSMT construction takes $O(m|V_S| \log^2 n)$ time, where $|V_S|$ is $O(n^2)$ in the worst case.

*Proof:* By Theorem 3 and the discussion in Section IV-A3, both the ML-VG construction and the MTST construction take $O(|V_S| \log^2 n)$ time. We only analyze the Steiner point selection and the refinement.

We use the Dijkstra shortest path algorithm in [15] to implement our Kruskal-based Steiner point selection. Since the Dijkstra algorithm extracts each vertex from the priority queue at most once, it takes $O(|V| \log |V| + |E|)$ time for a graph $G(V, E)$. However, unlike the Dijkstra algorithm, since we let several vertices be new sources during the selection, an extracted vertex may also be reinserted into the priority queue and extracted again later. In detail, each time an SPC has been constructed, we will set all the vertices of the SPC as sources, i.e., let their weights be zero, and reinsert them into the priority queue. There are $m$ pin-vertices, so $m - 1$ SPCs will be constructed, implying that a vertex can be extracted at most $m$ times in the worst case. Since there are $O(|V_S| \log n)$ vertices, the Steiner point selection takes $O(m|V_S| \log^2 n)$ time.

The refinement moves several PDCs of the initial solution to reduce the wirelength. Since the ML-VG has $O(|V_S| \log n)$ vertices and edges, by Definition 4, both the total size of all PDCs and the time needed to find them out are $O(|V_S| \log n)$ in the worst case. As discussed in Section IV-A4, if a PDC has $i$ elements, its possible moving offset can be determined in $O(i \log n)$ time after $O(n \log n)$-time preprocessing. Therefore, the refinement takes $O(|V_S| \log^2 n)$ time. Note that during the refinement, two PDCs may be merged into one PDC, and the possible moving offset of the new PDC can be determined from the original two PDCs.

To conclude, our ML-OARSMT algorithm takes $O(m|V_S| \log^2 n)$ time in the worst-case. ■

The run time depends on $|V_S|$, and $|V_S|$ varies with the input instance. If there are only $O(1)$ extremely large and extremely long-and-narrow obstacles, each of which generates $O(n)$ vertices to $V_S$, $|V_S|$ will be $O(n)$ instead of $O(n^2)$. Moreover, if the constructed SPCs in the Steiner point selection are distributed uniformly, a vertex will be extracted only $O(1)$ times on the average, instead of $O(m)$. Under these circumstances, the time complexity of our algorithm will be $O(n \log^2 n)$.

### B. OAPD-ST Problem

We extend our four-step ML-OARSMT algorithm to deal with the OAPD-ST problem. Since the last three steps are directly applicable, we only discuss the graph construction. Without loss of generality, we assume that $N_l$ is even.

Lee and Yang [16] proposed an $O(n \log n)$-time algorithm to transform a two-layer instance with PD constraint into a one-layer instance without PD constraint. As shown in Fig. 8(a) and (b), a rectangle in the horizontal layer is

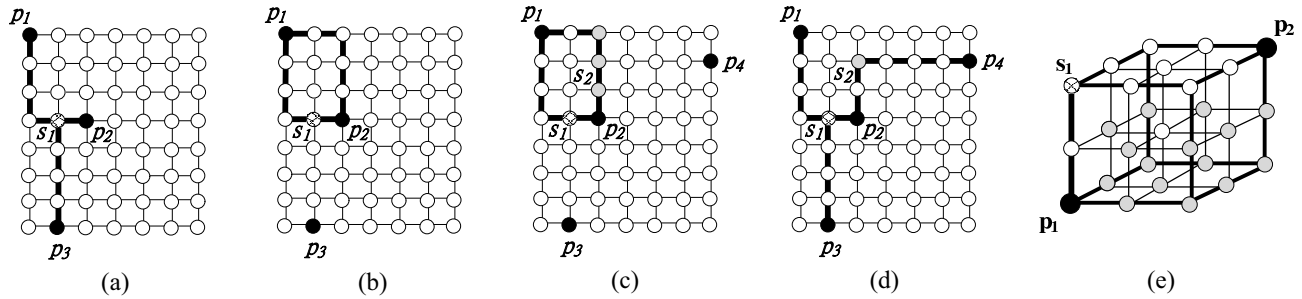



Fig. 9. (a) The optimal Steiner tree connecting $p_1$, $p_2$, and $p_3$. (b) $\partial$SPC($p_1, p_2$). (c) $s_2$ will be selected to connect $p_4$. (d) $s_2$ is not a Steiner point. (e) There are many gray vertices in 3D SPC($p_1, p_2$).

transformed into its two vertical boundary segments to prevent invalid horizontal routing, and a rectangle in vertical layer is transformed into its two horizontal boundary segments. They proved that a valid path in the transformed one-layer instance is directly a valid path in the original two-layer instance where a bend of the path in the transformed one-layer instance corresponds to a via in the original two-layer instance.

We combine the transformation in [16] and the concept of the 3D reduction in Section III-B to propose a new reduction $R'(\cdot)$. Let $z_0$ be $-\infty$, $z_{\lceil N_L/2\rceil+1}$ be $\infty$, and $z_i$ be $i \times C_v$ for $1 \le i \le \lceil N_L/2\rceil$. $R'(\cdot)$ transforms an OAPD-ST instance $I$ into a 3D instance $R'(I)$ as follows.

1) For $1 \le i \le \lceil N_L/2\rceil$, layers $(2i-1)$ and $2i$ are transformed into plane $z = z_i$ by the method in [16].
2) For each pin-vertex $p = (x, y, l)$ in $P$, $R'(p)$ is $(x, y, z_{\lceil l/2\rceil})$.
3) For each obstacle $o$ in $O$, let $l$ be the layer number of $o$. If $o$ is in a horizontal layer, $R'(o)$ represents a rectilinear box constructed by projecting $o$ to $z = z_{\lceil l/2\rceil}$ and $z = z_{(\lceil l/2\rceil+1)}$ and connecting the vertices of those two projections using line segments parallel to the $z$-axis. Otherwise, $R'(o)$ represents a rectilinear box constructed by applying the above method on $z = z_{\lceil l/2\rceil}$ and $z = z_{(\lceil l/2\rceil-1)}$.

Fig. 8(c) shows $R'(I)$ for a two-layer instance $I$ in Fig. 8(a).

The PD-VG for an OAPD-ST problem instance $I$ can be constructed as follows.

1) Use $R'(\cdot)$ to transform $I$ into $R'(I)$.
2) Apply the ML-VG construction in Section IV-A1 on $R'(I)$ to obtain $G'(V', E')$.
3) Transform the ML-VG $G'(V', E')$ back to the PD-VG $G(V, E)$. For a vertex $v = (x, y, z)$, let $v_1$ be $(x, y, 2z-1)$ and $v_2$ be $(x, y, 2z)$. For each $v \in V'$, insert $v_1$ and $v_2$ into $V$ and $(v_1, v_2)$ into $E$. For each edge $e = (u, v) \in E'$, if $e$ is vertical, insert $(u_1, v_1)$ into $E$; if $e$ is horizontal, insert $(u_2, v_2)$ into $E$; if $e$ is perpendicular to routing layers, insert either $(u_1, v_2)$ or $(u_2, v_1)$ into $E$ depending on whether $u$ is higher than $v$ or not. Remove all invalid vertices and edges from $G$, which intersect the interior of an obstacle.

Note that the ML-VG embeds all the shortest paths among pin-vertices and obstacle corners, but the PD-VG does not, since the visibility graph does not consider the bends, which represent vias in the transformed layers. Since the transformation takes $O(n \log n)$-time [16], the PD-VG construction takes the same time as does the ML-VG construction. Moreover, since the last three steps of the algorithm are the same, we conclude with the following theorem.

*Theorem 5:* Our OAPD-ST construction takes $O(m|V_S| \log^2 n)$ time, where $|V_S|$ is $O(n^2)$ in the worst case.

## V. Advanced Steiner Point Selection

We propose an advanced Steiner point selection to improve the solution quality. Compared with the original Steiner point selection in Section IV-A2, the advanced Steiner point selection potentially avoids selecting inferior Steiner points, and still has the same time complexity. Moreover, our graph Steiner tree algorithm consisting of the advanced Steiner point selection and the MTST construction guarantees an optimal solution with four terminals in certain graphs, and any sub-tree of four leaves in the generated solution is also optimal.

### A. Original Selection

*Definition 5:* For a graph $G(V, E)$ and two vertices, $u$ and $v$, in $V$, $\partial$SPC($u, v$) is the boundary of SPC($u, v$), which is the collection of points $u' \in$ SPC($u, v$) satisfying $\exists v' \notin$ SPC($u, v$), $(u', v') \in E$.

SPCs can assist the Steiner point selection. For example, Fig. 9(a) shows the optimal Steiner tree among $p_1$, $p_2$, and $p_3$, and Fig. 9(b) indicates that the optimal Steiner point $s_1$ is located on $\partial$SPC($p_1, p_2$). This is because the path between $p_1$ and $p_2$ and the path between $p_2$ and $p_3$ in Fig. 9(a) share the path between $s_1$ and $p_2$, and the path between $s_1$ and $p_2$ belongs to $\partial$SPC($p_1, p_2$) in Fig. 9(b).

The intuitive and original use of SPCs in Section IV-A2 potentially selects inferior Steiner points. For example, as shown in Fig. 9(c), after selecting $s_1$ as a Steiner point, since $s_2$ is the closest vertex of $p_4$ to $\partial$SPC($p_1, p_2$), $s_2$ will be selected to connect $p_4$. However, as shown in Fig. 9(d), $s_2$ is not a Steiner point in the resulting tree. This is because after $s_1$ has been selected as a Steiner point, the partial topology of the resulting tree has been fixed, and several vertices of $\partial$SPC($p_1, p_2$) are no longer good Steiner point candidates. For example, the gray vertices of $\partial$SPC($p_1, p_2$) in Fig. 9(c) are not good Steiner point candidates after selecting $s_1$.

Since there are at most $m-2$ Steiner points, if we select an inferior Steiner point, we will lose the possibility to select a

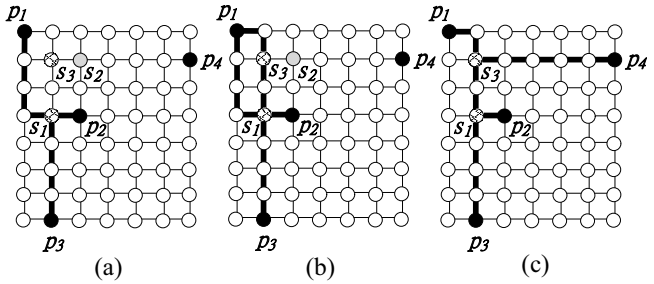



Fig. 10. (a) SPC$(p_1, p_2)$ is replaced with a shortest path between $p_1$ and $p_2$ passing through $s_1$. (b) SPC$(p_1, p_2)$ is replaced by SPC$(p_1, s_1)$ and SPC$(s_1, p_2)$, $s_3$ is included in SPC$(p_1, s_1)$. (c) The optimal Steiner tree for $p_1$, $p_2$, $p_3$, and $p_4$.

good Steiner point. For example, Fig. 10(c) shows the optimal Steiner tree connecting $p_1$, $p_2$, $p_3$, and $p_4$, and $s_3$ is a Steiner point in the optimal solution. However, if we selected $s_2$ in Fig. 9(c) as a Steiner point, we would not select $s_3$ in Fig. 10(a) as a Steiner point.

Moreover, this situation will become worse in a 3D rectilinear graph. Informally speaking, this is because $\partial$SPC$(p_1, p_2)$ in 2D resembles a rectilinear polygon consisting of only two rectilinear paths (if obstacles are all rectangles), while $\partial$SPC$(p_1, p_2)$ in 3D usually resembles a rectilinear polyhedron consisting of at least six faces, each of which is a rectilinear polygon. For instance, Fig. 9(e) shows an example of SPC$(p_1, p_2)$ in 3D, and if $s_1$ is selected as a Steiner point, 15 gray vertices on $\partial$SPC$(p_1, p_2)$ will not be good Steiner point candidates.

### B. Update Operation

An intuitive enhancement is to replace the SPC$(p_1, p_2)$ with a single shortest path between $p_1$ and $p_2$. However, a bad path will lose the possibility to select a good Steiner point. For example, as shown in Fig. 10(a), a single shortest path between $p_1$ and $p_2$ passing through $s_1$ is selected, but $s_3$ is not on this path. Unfortunately, the number of shortest paths could be exponential, thus it would be intractable to select the best path.

To overcome the potential drawback, we introduce an update operation to amend an SPC after one of its vertices has been selected as a Steiner point. Precisely, when a vertex $s_1$ of $\partial$SPC$(p_1, p_2)$ has been selected as a Steiner point, SPC$(p_1, p_2)$ will be replaced with SPC$(p_1, s_1)$ and SPC$(s_1, p_2)$. This update operation not only excludes a selection of an inferior Steiner point but also provides a possibility to select a real Steiner point. For example, in Fig. 10(b), since SPC$(p_1, s_1)$ contains $s_3$ but neither of SPC$(p_1, s_1)$ and SPC$(s_1, p_2)$ includes $s_2$, $s_3$ can be selected from SPC$(p_1, s_1)$ as a Steiner point rather than $s_2$.

The objective of the update operation is to select real Steiner points instead of inferior ones. However, as a polynomial-time heuristic for an NP-hard problem, our update operation still cannot guarantee an optimal solution.

### C. New Procedure of Steiner Point Selection

We use a five-terminal instance to illustrate the procedure of our advanced Steiner point selection in Fig. 11. Let $\mathcal{C}$

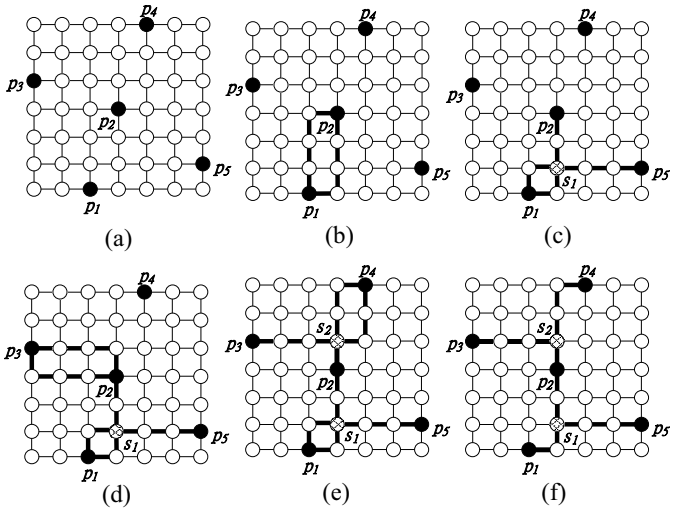



Fig. 11. (a) An instance of five terminals. (b) SPC$(p_1, p_2)$ is built. (c) $s_1$ is selected and SPC$(p_1, p_2)$ is replaced by SPC$(p_1, s_1)$ and SPC$(s_1, p_2)$. (d) $p_3$ is connected and SPC$(p_2, p_3)$ is constructed. (e) $s_2$ is selected and SPC$(p_2, p_3)$ is replaced by SPC$(p_2, s_2)$ and SPC$(s_2, p_3)$. (f) The resulting optimal SMT.

be the set of current components consisting of pin-vertices and SPCs. First, as shown in Fig. 11(b), since $p_1$ and $p_2$ are the nearest components, SPC$(p_1, p_2)$ is computed and inserted into $\mathcal{C}$. Then, as shown in Fig. 11(c), $p_5$ and SPC$(p_1, p_2)$ are the nearest components and $s_1$ is the closest vertex to $p_5$ in SPC$(p_1, p_2)$, so SPC$(s_1, p_5)$ is constructed and inserted into $\mathcal{C}$. Since $s_1$ is selected as Steiner point, SPC$(p_1, p_2)$ is replaced by SPC$(p_1, s_1)$ and SPC$(s_1, p_2)$, and SPC$(p_1, s_1)$, SPC$(s_1, p_2)$ and SPC$(s_1, p_5)$ are inserted into $\mathcal{C}$. Similarly, as shown in Fig. 11(d) and (e), $p_3$ is connected and SPC$(p_2, p_3)$ is constructed, and later $s_2$ is selected as Steiner point to connect $p_4$, in which SPC$(p_2, p_3)$ is replaced by SPC$(p_2, s_2)$ and SPC$(s_2, p_3)$. At last, as shown in Fig. 11(f), computing an MTST for the terminal set $P \cup S$ will yield a Steiner tree for $P$, which is optimal in this instance.

### D. Time Complexity Analysis

We give the time complexity of the advanced Steiner point selection in the following analysis. The analysis will show that the time complexity of the advanced Steiner point selection is identical to the original Steiner point selection.

*Theorem 6:* Our algorithm with the advanced Steiner point selection takes $O(m|V_S| \log^2 n)$ time, where $|V_S|$ is $O(n^2)$ in the worst case.

*Proof:* Selecting a Steiner point will remove at most one SPC and include at most three SPCs, so there are $O(m)$ SPCs during the Steiner point selection. Hence, each vertex in $V$ will be inserted into and extracted from the priority queue $O(m)$ times. According to Theorem 3, ML-VG has $O(|V_S| \log n)$ vertices, and thus there are $O(m|V_S| \log n)$ such events. Since the degree of a vertex is a constant in a rectilinear graph, an event to insert or extract a vertex takes $O(\log n)$ time, and all the events take $O(m|V_S| \log^2 n)$ time. ■

Compared with Theorem 4, our algorithm with the advanced Steiner point selection has the same time complexity.

### E. Optimality

We will show that our graph Steiner-tree algorithm consisting of the advanced Steiner point selection and the MTST construction can guarantee the optimal solution of four pin-vertices for the RSMT problem, the OARSMT problem, and the OAPD-ST problem through Hanan grid [22], Escape graph [23], and PDEG [12], respectively, and any sub-tree with four leaves in the generated solution is optimal. Note that the escape graph and PDEG are obstacle-avoiding routing graphs for the OARSMT problem and the OAPD-ST problem, respectively, and guarantee the existence of at least one optimal solution.

These routing graphs $G(V, E)$ share the following important properties.

1) It contains at least one optimal solution for the original problem.
2) It contains an obstacle-avoiding shortest path between any two vertices.
3) For any subset $V'$ of $V$, it contains the optimal Steiner tree for $V'$.

We first prove our claims in a uniform grid graph, which also has the above properties. Since our proof only uses those properties, the claims apply to Hanan grid, Escape graph, and PDEG.

*Theorem 7:* Given a uniform grid graph $G(V, E)$ and a terminal set $P \subseteq V$, if $|P| \leq 4$, our algorithm generates an optimal Steiner minimal tree (SMT).

*Proof:* Assume that the corresponding grid is $w \times l \times h$ such that $V = \{(x, y, z) \mid 1 \leq x \leq w, 1 \leq y \leq l, 1 \leq z \leq h\}$ and $E = \{(u, v) \mid |\overline{uv}| = 1, u, v \in V\}$. It is clear that the distance between two vertices in $G$ is the $L_1$ (Manhattan) distance between them. Let $T^*$ be an optimal SMT, and for any two vertices $u$ and $v$, let $d(u, v)$ be the length of the shortest path between them in $G$. If $|P| = 2$, an optimal solution is a shortest path between the two terminals, and the theorem can be trivially proved. Therefore, we only consider the cases for $|P| = 3$ and $|P| = 4$.

We first prove the case for $|P| = 3$. If $T^*$ contains no Steiner point, $T^*$ is merely an MTST for $P$, which must be found by our second step, the MTST construction, so we only consider the case in which $T^*$ contains exactly one Steiner point. Let $P$ be $\{p_1, p_2, p_3\}$ and $s$ be the Steiner point of $T^*$ such that $|T^*| = d(p_1, s) + d(p_2, s) + d(p_3, s)$. We prove that $s$ belongs to $\partial\text{SPC}(p_1, p_2)$, and the proof implies that $s$ also belongs to $\partial\text{SPC}(p_2, p_3)$ and $\partial\text{SPC}(p_1, p_3)$. Let $s'$ be the closest vertex from $\partial\text{SPC}(p_1, p_2)$ to $s$. If $s$ is outside $\text{SPC}(p_1, p_2)$ [Fig. 12(a)], since $d(p_1, s) = d(p_1, s') + d(s, s')$ and $d(p_2, s) = d(p_2, s') + d(s, s')$, $d(p_1, s') + d(p_2, s') + d(p_3, s') \leq d(p_1, s') + d(p_2, s') + d(s', s) + d(p_3, s) < d(p_1, s) + d(p_2, s) + d(p_3, s)$, contradicting $T^*$ is optimal. If $s$ is inside $\text{SPC}(p_1, p_2)$ [Fig. 12(b)], since $d(p_1, s') + d(p_2, s') = d(p_1, s) + d(p_2, s)$ and $d(p_3, s) = d(p_3, s') + d(s', s)$, $d(p_1, s') + d(p_2, s') + d(p_3, s') < d(p_1, s') + d(p_2, s') + d(p_3, s) = d(p_1, s) + d(p_2, s) + d(p_3, s)$, contradicting $T^*$ is optimal. Therefore, $s$ belongs to $\partial\text{SPC}(p_1, p_2)$, $\partial\text{SPC}(p_2, p_3)$ and $\partial\text{SPC}(p_1, p_3)$, and our algorithm can find $s$ no matter starting with $\text{SPC}(p_1, p_2)$, $\text{SPC}(p_2, p_3)$, or $\text{SPC}(p_1, p_3)$.

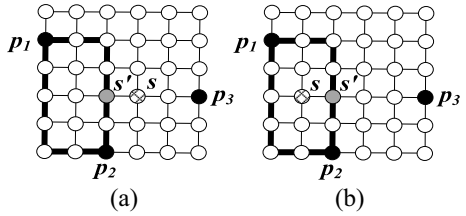



Fig. 12. Three-pin cases. (a) $s$ is outside $\text{SPC}(p_1, p_2)$. (b) $s$ is inside $\text{SPC}(p_1, p_2)$.

Now, we prove the case for $|P| = 4$. If $T^*$ contains exactly one Steiner point $s$, either: 1) three terminals are connected to $s$ and the other terminal is connected to one terminal or 2) the four terminals are all connected to $s$. For case 1), the above proof for three terminals is applicable, and for the case 2), $s$ can be viewed as two identical Steiner points. Thus, we only consider the case in which $T^*$ contains exactly two Steiner points.

Let $P = \{p_1, p_2, p_3, p_4\}$ and let $s$ and $t$ be the two Steiner points. Without loss of generality, we assume that $T^*$ consists of five paths, from $p_1$ to $s$, from $p_2$ to $s$, from $s$ to $t$, from $t$ to $p_3$ and from $t$ to $p_4$. Please see Fig. 13(a) for an example. It is clear that $s$ is the Steiner point of the optimal SMT connecting $p_1$, $p_2$, and $t$, and $t$ is the Steiner point of the optimal SMT connecting $p_3$, $p_4$, and $s$; otherwise, $T^*$ is not optimal. According to the proof for three terminals, $s$ belongs to $\partial\text{SPC}(p_1, p_2)$, $\partial\text{SPC}(p_1, t)$, and $\partial\text{SPC}(p_2, t)$, and $t$ belongs to $\partial\text{SPC}(p_3, p_4)$, $\partial\text{SPC}(p_3, s)$, and $\partial\text{SPC}(p_4, s)$. It is clear that $d(s, t)$ is the minimum distance between $\text{SPC}(p_1, p_2)$ and $\text{SPC}(p_3, p_4)$; otherwise, we can find a closer pair of $s' \in \text{SPC}(p_1, p_2)$ and $t' \in \text{SPC}(p_3, p_4)$ to generate a Steiner tree shorter than $T^*$. Note that there could be more than one pair of $s$ and $t$ leading to an optimal SMT.

We prove the case in which our advanced Steiner point selection started with $\text{SPC}(p_1, p_2)$, i.e., $d(p_1, p_2)$ is the minimum distance between any two pin-vertices, and it is similar for the other five cases. If $d(p_3, p_4)$ is smaller than $d(\text{SPC}(p_1, p_2), p_3)$ and $d(\text{SPC}(p_1, p_2), p_4)$, $\text{SPC}(p_3, p_4)$ will be constructed, and since $s$ belongs to $\partial\text{SPC}(p_1, p_2)$ and $t$ belongs to $\partial\text{SPC}(p_3, p_4)$, they will be selected. Therefore, we discuss the case in which $d(p_3, p_4)$ is larger than $d(\text{SPC}(p_1, p_2), p_3)$ or $d(\text{SPC}(p_1, p_2), p_4)$. Among all the possible pairs $(s, t)$ of optimal Steiner points, let $(s_3, t_3)$ be the pair $(s, t)$ with the minimum distance between $s$ and $p_3$, and let $(s_4, t_4)$ be the pair $(s, t)$ with the minimum distance between $s$ and $p_4$. Please see Fig. 13(b) for an example.

Under these circumstances, as shown in Fig. 13(c), $s_3$ is the optimal Steiner point connecting $p_1$, $p_2$, and $p_3$, and as shown in Fig. 13(d), $s_4$ is the optimal Steiner point connecting $p_1$, $p_2$, and $p_4$. Therefore, if $p_3$ is closer to $\text{SPC}(p_1, p_2)$ than $p_4$ is, due to the optimality for three terminals, our advanced Steiner point selection will select $s_3$ to connected $p_3$. After selecting $s_3$, $\text{SPC}(s_3, p_3)$ is generated, and $\text{SPC}(p_1, p_2)$ is replaced with $\text{SPC}(s_3, p_1)$ and $\text{SPC}(s_3, p_2)$. Since $t_3$ belongs to $\partial\text{SPC}(s_3, p_3)$ ($t_3$ is $t$ in the above paragraph now), our advanced Steiner point selection will successfully select $t_3$ to connect $p_4$; otherwise, there is a better Steiner point $v$ in

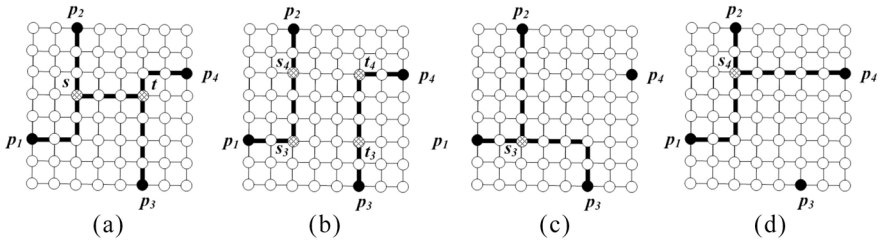



Fig. 13. A four-pin case. (a) An optimal SMT connects $p_1$, $p_2$, $p_3$, and $p_4$ using two Steiner points $s$ and $t$. (b) $(s_3, t_3)$ and $(s_4, t_4)$ minimize the distances between $s$ and $p_3$ and $s$ and $p_4$, respectively. (c) $s_3$ is the optimal Steiner point for $p_1$, $p_2$, and $p_3$. (d) $s_4$ is the optimal Steiner point for $p_1$, $p_2$, and $p_4$.

TABLE I
COMPARISON ON THE TOTAL COST, NUMBER OF VIAS, AND CPU TIME BETWEEN [11] AND OURS, WHERE $C_v$ IS 3

| Test Cases | $m$ / $k$ / $N_l$ | $n$ | Total Cost(# via) | | Imp. (%) | Time | | speed-up |
|---|---|---|---|---|---|---|---|---|
| | | | [11] (A) | ours (B) | $(\frac{A-B}{A})$ | [11] (C) | ours (D) | $(\frac{C}{D})$ |
| ind1 | 50 / 6 / 5 | 74 | 55537(49) | 54207(49) | 2.39 | 0.132 | 0.008 | 16.5x |
| ind2 | 200 / 85 / 6 | 540 | 12512(224) | 12008(206) | 4.03 | 1.866 | 0.079 | 23.62x |
| ind3 | 250 / 13 / 10 | 302 | 10973(359) | 10555(348) | 3.81 | 1.981 | 0.073 | 27.14x |
| ind4 | 500 / 100 / 5 | 900 | 77033(0) | 77292(0) | -0.34 | 4.648 | 0.048 | 96.83x |
| ind5 | 1000 / 20 / 5 | 1080 | 14515511(0) | 14599961(0) | -0.58 | 24.421 | 0.099 | 246.68x |
| rt1 | 25 / 10 / 10 | 65 | 4334(76) | 4169(70) | 3.81 | 0.053 | 0.018 | 2.94x |
| rt2 | 100 / 20 / 10 | 180 | 9434(215) | 9132(209) | 3.2 | 0.591 | 0.069 | 8.57x |
| rt3 | 250 / 50 / 10 | 450 | 15569(490) | 14750(478) | 5.26 | 3.776 | 0.42 | 8.99x |
| rt4 | 500 / 50 / 10 | 700 | 22034(918) | 21013(936) | 4.63 | 9.369 | 0.621 | 15.09x |
| rt5 | 1000 / 100 / 5 | 1400 | 27890(869) | 26970(814) | 3.3 | 27.858 | 0.856 | 32.54x |
| Avg. | – | – | – | – | 2.95 | – | – | 47.89x |

$\partial\text{SPC}(s_3, p_1)$ or $\partial\text{SPC}(s_3, p_2)$ closer to $p_3$ than $t_3$ is, contradicting that $T^*$ is optimal. It is symmetric for the case in which $p_4$ is closer to $\text{SPC}(p_1, p_2)$ than $p_3$ is. ■

*Corollary 1:* Given a Hanan grid [22], an Escape graph [23], or a PDEG [12] and a terminal set $P$, if $|P| \leq 4$, our algorithm generates an optimal SMT.

Theorem 7 and Corollary 1 are very significant since the graph Steiner tree algorithms in [4] and [6] do not guarantee to generate an optimal solution for a four-terminal instance. For the 2D RSMT problem, FLUTE [9] is a well-known software and guarantees an optimal solution when the number of terminals is at most 9. However, its extension to 3D or for those cases where obstacles exist may not guarantee an optimal solution. For example, FOARS [8], an extension of FLUTE to the OARSMT problem, does not guarantee any optimal solution.

*Theorem 8:* Given a uniform grid graph, a Hanan grid, an Escape graph, or a PDEG and a terminal set $P$, our algorithm with the advanced Steiner point selection generates a Steiner tree in which any four-leaf subtree whose leaves are terminals or Steiner points is optimal.

*Proof:* Our advanced Steiner point selection generates SPCs between terminals and Steiner points. In the generation of an SPC, a terminal and a Steiner point are equivalently processed. Moreover, the proof for Theorem 7 does not depend on the starting SPC. Therefore, for any four-leaf subtree of the generated solution whose leaves are terminals or Steiner points, the theorem follows from Theorem 7. ■

## VI. EXPERIMENTAL RESULTS

We have implemented our algorithm in C language, and conducted all the experiments on an IBM x3550 server with 12 3.3-GHz processors and 64-GB memory. All the results are generated using one processor instead of 12. We requested binaries from Lin *et al.* [11], Liu *et al.* [12], and Chuang and Lin [13]. However, Chuang and Lin [13] could not provide us with their binary executable, so we could not make a direct comparison with it. The experiments of our algorithm in Sections VI-A and VI-B (Tables I and II) use the original Steiner point selection method in Section IV-A2.

### A. ML-OARSMT Problem

There are totally ten benchmark circuits, five industry test cases (ind1–ind5) from Synopsys, and five random test cases (rt1–rt5) generated by Lin *et al.* [11]. The five random test cases are generated based on the five industry test cases and thus can be viewed as practical cases.

Table I lists the total cost, the number of vias, and the CPU time of [11] and our algorithm under the condition $C_v = 3$. Compared with [11], our algorithm improves the solution quality in terms of total cost by 2.95% on the average, and also achieves 47.89 times speed-up on the average at the same time as shown in Table I. For a large test case (ind5), the speed-up is up to about 246 times.

Moreover, in order to analyze the empirical time complexity, for each test case, we create a point whose $x$-coordinate is the input size $n$ and whose $y$-coordinate is the CPU time (of our algorithm or [11]). By least squares fitting on the log-log-axes of those ten points, the slopes of the fitting lines for our algorithm and [11] are 1.18 and 2.04, respectively, implying that the empirical time complexity of our algorithm is $\Theta(n^{1.18})$ and that of [11] is $\Theta(n^{2.04})$. As a result, our algorithm behaves like a subquadratic-time algorithm for practical applications, and is very close to a loglinear-time algorithm.

TABLE II
COMPARISON ON THE TOTAL COST, NUMBER OF VIAS, AND CPU TIME BETWEEN [12] AND OURS, WHERE $C_v = 3$ AND $N_l = 10$

| Test Cases | | | Total Cost(# via) | | Imp. (%) | Time | | speed-up |
|---|---|---|---|---|---|---|---|---|
| $m$ | $k$ | $n$ | [12] (A) | ours (B) | $(\frac{A-B}{A})$ | [12] (C) | ours (D) | $(\frac{C}{D})$ |
| 10 | 50 | 210 | 5209(31) | 4830(30) | 7.28 | 0.12 | 0.04 | 3.0x |
| 10 | 100 | 410 | 10370(29) | 9683(29) | 6.62 | 0.55 | 0.09 | 6.11x |
| 20 | 100 | 420 | 15097(63) | 14090(62) | 6.67 | 0.64 | 0.10 | 6.4x |
| 20 | 200 | 820 | 29528(63) | 27533(63) | 6.76 | 1.03 | 0.18 | 5.72x |
| 50 | 250 | 1,050 | 58589(165) | 54617(164) | 6.78 | 2.24 | 0.30 | 7.46x |
| 50 | 500 | 2,050 | 117713(166) | 109684(169) | 6.82 | 14.16 | 0.84 | 16.85x |
| 100 | 500 | 2,100 | 165430(335) | 154518(342) | 6.60 | 16.57 | 0.94 | 17.62x |
| 100 | 1,000 | 4,100 | 329025(336) | 306155(349) | 6.95 | 53.13 | 1.86 | 28.56x |
| 200 | 1,000 | 4,200 | 461616(682) | 431312(699) | 6.56 | 61.03 | 2.09 | 29.20x |
| 200 | 2,000 | 8,200 | 919814(681) | 859542(720) | 6.55 | 213.53 | 4.18 | 51.08x |
| Average | | | – | – | 6.76 | – | – | 17.20x |

Compared with [11], among the ten test cases, our algorithm generates slightly worse solutions for ind4 and ind5. Both ind4 and ind5 are viewed as 2D OARSMT instances because in each case, layer 2 and layer 4 are entirely covered by an extremely large obstacle, and all the pin-vertices are located in layer 3. For the 2D OARSMT problem, since the spanning graph in [11] has more information than the 2D visibility graph, the algorithm in [11] could generate better results for ind4 and ind5 than our algorithm. However, the spanning graph for the ML-OARSMT problem in [11] is extended from a single-layer version in [2] but not naturally defined for a multilayer instance. Therefore, for the ML-OARSMT problem, our algorithm could generate better results than that in [11].

After excluding ind4 and ind5, the empirical time complexity of our algorithm is $\Theta(n^{1.08})$, while that of [11] is $\Theta(n^{1.88})$. In other words, our algorithm is faster than [11] by about an order of $n$ over the eight test cases.

### B. OAPD-ST Problem

In order to address our motivation, we need to conduct experiments on test cases similar to practical conditions. However, unlike random test cases from Lin *et al.* [11] for the ML-OARSMT problem, the random test cases generated by Liu *et al.* [12] for the OAPD-ST problem are not very practical since the number of pin-vertices is too large and even larger than the number of obstacles, and the obstacles are too large and too long-and-narrow. As a result, we take the shapes and sizes of obstacles in the industrial test cases (ind1–ind5) for the ML-OARSMT problem to randomly generate a new set of test cases for the OAPD-ST problem. The detailed setting of those test cases is shown in Table II. For each kind of test cases, we generate 100 samples, and the reported results (total cost, number of vias, and CPU time) are average results.

Table II lists the total cost, the number of vias, and the CPU time of [12] and our algorithm under the conditions $C_v = 3$ and $N_l = 10$. Compared with [12], our algorithm improves the solution quality in terms of total cost by 6.76% on the average, and also achieves 17.20 times speed-up on the average at the same time as shown in Table II. For the largest test case ($m = 200$ and $k = 2000$), the speed-up is 51 times.

Similar to Section VI-A, we use the least squares fitting method to analyze the performance, and show that the slopes of the fitting lines of our algorithm and the algorithm in [12] are 1.15 and 1.93, respectively, implying that the empirical time complexity of our algorithm and the algorithm in [12] is $\Theta(n^{1.15})$ and $\Theta(n^{1.93})$, respectively. As a result, our algorithm behaves like a subquadratic-time algorithm for practical applications, and is very close to a loglinear-time algorithm.

In order to compare our algorithm with that in [13], we also run our algorithm on the test cases in [12]. Although the solution quality is similar, according to the least squares fitting analysis, the empirical time complexity of our algorithm on those test cases is $\Theta(n^{2.00})$, but that of [13] is $\Theta(n^{2.11})$, indicating that our algorithm could perform a bit faster. Note that our empirical time complexity is not loglinear because the test cases in [12] include too many pin-vertices, and too many extremely large and extremely long-and-narrow obstacles. Besides, since we do not have the binary code of [13], we cannot evaluate the performance of their algorithm on our test cases.

Furthermore, we also generate new test cases from the 22 commonly used benchmarks in [2]–[8] and [10] for the OARSMT problem by randomly assigning the layers of pin-vertices and obstacles, and create 100 samples for each original benchmark. As shown in Table III, compared with [12], our algorithm improves the solution quality in terms of total cost by 5.48% on the average. The least squares fitting analysis shows that the empirical complexities of our algorithm and the algorithm in [12] are $\Theta(n^{0.99})$ and $\Theta(n^{1.63})$, respectively. These time complexities are not fully reasonable because RT01–RT05 contain many extremely long-and-narrow obstacles, different from IND1–IND5 and RC01–RC12 such that the corresponding points are not close to the least squares fitting line. If we neglect RT01–RT05, the empirical complexity for ours and [12] are $\Theta(n^{1.04})$ and $\Theta(n^{2.03})$, respectively.

### C. Advanced Steiner Point Selection

We compare our algorithm with and without the advanced Steiner point selection in both PD-VG and PDEG using randomly generated test cases in Section VI-B. Since PDEG contains at least one optimal OAPD-ST, it is a more proper graph to evaluate the effectiveness of our advanced Steiner point selection. Moreover, Corollary 1 also indicates that our method guarantees an optimal solution of four terminals on

TABLE III
COMPARISON ON THE TOTAL COST, NUMBER OF VIAS, AND CPU TIME BETWEEN [12] AND OURS ON TEST CASES GENERATED FROM 2D OARSMT BENCHMARKS, WHERE $C_v$ IS 3

| Test Cases | $m$ / $k$ / $N_l$ | $n$ | Total Cost(# via) | | Imp. (%) | Time | |
|---|---|---|---|---|---|---|---|
| | | | ours (A) | [12] (B) | $(\frac{B-A}{B})$ | ours(C) | [12] (D) |
| IND1 | 10/32/6 | 138 | 673.6(17.41) | 699.18(18.37) | 3.66 | 0.0061 | 0.0069 |
| IND2 | 10/43/6 | 182 | 9271.06(17.02) | 9452.49(19.83) | 1.92 | 0.0070 | 0.0041 |
| IND3 | 10/50/6 | 210 | 642.95(16.49) | 677.67(18.73) | 5.12 | 0.0069 | 0.0052 |
| IND4 | 25/79/6 | 341 | 1211.79(39.02) | 1283(43.51) | 5.55 | 0.0138 | 0.0144 |
| IND5 | 33/71/6 | 317 | 1503.5(58.03) | 1586.49(61.99) | 5.23 | 0.0121 | 0.0128 |
| RC01 | 10/10/6 | 50 | 25523.96(20.62) | 28494.82(20.34) | 10.43 | 0.0031 | 0.0042 |
| RC02 | 30/10/6 | 70 | 41216.44(63.78) | 43158.22(63.24) | 4.50 | 0.0105 | 0.0126 |
| RC03 | 50/10/6 | 90 | 53502.15(115.55) | 55878.54(105.78) | 4.25 | 0.0147 | 0.0216 |
| RC04 | 70/10/6 | 110 | 57379.87(157.39) | 60402.29(147.53) | 5.00 | 0.0181 | 0.0329 |
| RC05 | 100/10/6 | 140 | 74515.02(228.84) | 79279.45(214.35) | 6.01 | 0.0259 | 0.0603 |
| RC06 | 100/500/6 | 2100 | 79055.63(262.86) | 84072.45(235.07) | 5.97 | 2.3502 | 10.0199 |
| RC07 | 200/500/6 | 2200 | 109128.34(511.73) | 114534.45(458.64) | 4.72 | 2.5789 | 12.3797 |
| RC08 | 200/800/10 | 3400 | 112053.72(658.44) | 119985.42(699.04) | 6.61 | 9.1623 | 50.7001 |
| RC09 | 200/1000/10 | 4200 | 110604.39(653.17) | 116933.19(697.2) | 5.41 | 13.3447 | 72.5772 |
| RC10 | 500/100/10 | 900 | 172356.51(1573.37) | 184558.5(1585.5) | 6.61 | 0.6738 | 5.5734 |
| RC11 | 1000/100/10 | 1400 | 247622.15(3207.39) | 261715.64(3361.06) | 5.39 | 1.0328 | 24.4056 |
| RC12 | 1000/10000/10 | 41000 | 776352.86(3697.03) | - | - | 103.2286 | - |
| RT01 | 10/500/6 | 2010 | 1913.96(24.28) | 2060.8(23.89) | 7.13 | 1.0830 | 2.8899 |
| RT02 | 50/500/6 | 2050 | 46198.94(129.73) | 48429.63(114.53) | 4.61 | 1.4107 | 9.0445 |
| RT03 | 100/500/6 | 2100 | 8545.12(227.09) | 8945.39(224.47) | 4.47 | 1.0421 | 3.4234 |
| RT04 | 100/1000/10 | 4100 | 8757.34(291.76) | 9320.22(325.38) | 6.04 | 5.2386 | 12.0890 |
| RT05 | 200/2000/10 | 8200 | 45713.1(633.45) | 48876.28(691.62) | 6.47 | 34.0381 | 132.8429 |
| Avg. | – | – | – | – | 5.48 | – | – |

TABLE IV
COMPARISON ON THE TOTAL COST AND CPU TIME BETWEEN OUR ALGORITHM WITH AND WITHOUT THE ADVANCED STEINER POINT SELECTION ON PD-VG AND PDEG, WHERE $C_v = 3$ AND $N_l = 10$

| Test Cases | | | Total Cost(# via) | | | | Imp. (%) | | Time | | | | Ratio | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | PD-VG | | PDEG | | | | PD-VG | | PDEG | | | |
| $m$ | $k$ | $n$ | w/o (A) | w/ (B) | w/o (C) | w/ (D) | $(\frac{A-B}{A})$ | $(\frac{C-D}{C})$ | w/o (A) | w/ (B) | w/o (C) | w/ (D) | $(\frac{B}{A})$ | $(\frac{D}{C})$ |
| 10 | 50 | 210 | 4830(30) | 4823(29) | 4817(25) | 4813(24) | 0.14 | 0.10 | 0.04 | 0.04 | 0.05 | 0.06 | 1.03x | 1.16x |
| 10 | 100 | 410 | 9683(29) | 9650(29) | 9708(24) | 9656(23) | 0.33 | 0.54 | 0.08 | 0.09 | 0.22 | 0.25 | 1.07x | 1.14x |
| 20 | 100 | 420 | 14090(62) | 14070(61) | 14058(52) | 14027(50) | 0.14 | 0.23 | 0.09 | 0.10 | 0.59 | 0.69 | 1.08x | 1.17x |
| 20 | 200 | 820 | 27533(63) | 27468(62) | 27507(52) | 27374(50) | 0.24 | 0.48 | 0.18 | 0.19 | 2.27 | 2.58 | 1.08x | 1.14x |
| 50 | 250 | 1,050 | 54617(164) | 54547(161) | 54441(134) | 54214(127) | 0.13 | 0.42 | 0.29 | 0.31 | 4.89 | 5.35 | 1.07x | 1.09x |
| 50 | 500 | 2,050 | 109684(169) | 109412(166) | 109521(133) | 108906(127) | 0.25 | 0.56 | 0.84 | 0.90 | 21.70 | 24.03 | 1.07x | 1.11x |
| 100 | 500 | 2,100 | 154518(342) | 154202(335) | 153946(269) | 153135(255) | 0.20 | 0.53 | 0.93 | 1.0 | 26.28 | 28.57 | 1.07x | 1.09x |
| 100 | 1,000 | 4,100 | 306155(349) | 305861(342) | 306150(270) | 304579(256) | 0.10 | 0.51 | 1.86 | 1.99 | 71.40 | 81.61 | 1.07x | 1.14x |
| 200 | 1,000 | 4,200 | 431312(699) | 430839(686) | 430270(543) | 428136(517) | 0.11 | 0.50 | 2.09 | 2.23 | 80.97 | 87.15 | 1.07x | 1.13 x |
| 200 | 2,000 | 8,200 | 859542(720) | 858353(705) | 857504(543) | 853090(515) | 0.14 | 0.51 | 4.17 | 4.45 | 209.05 | 237.24 | 1.07x | 1.13x |
| Average | | | – | – | – | – | 0.18 | 0.44 | – | – | – | – | 1.07x | 1.13x |

PDEG, and any four-leaf sub-tree in a generated solution is optimal.

Table IV lists the total cost, the number of vias, and the CPU times. As shown in Table IV, the advanced Steiner point selection improves the solution quality in terms of total cost by 0.18% in PD-VG and 0.44% in PDEG. Moreover, it also lowers the number of vias on the average.

Furthermore, as shown in Table IV, the advanced Steiner point selection on the average increases only 7% and 13% run time in PD-VG and PDEG, respectively. Moreover, the increase does not grow with the input size, indicating that the advanced Steiner point selection does not change the time complexity (Theorem 6).

To conclude, the advanced Steiner point selection only uses a little more run time to reduce reasonable total cost. If users want to lower the total cost, they can perform our algorithm on PDEG to obtain better solution quality. On the other hand, if the efficiency is more important, they can adopt the version in PD-VG with the advanced Steiner point section since the corresponding speed is still fast compared to all the state-of-the-art works.

### D. Remarks

In order to extensively evaluate our algorithm, we also conduct experiments on OARSMT, RSMT, and multilayer RSMT test cases, respectively. For the OARSMT problem, we adopt

the 22 commonly used benchmarks in [2]–[8] and [10] and for the RSMT problem, we remove obstacles from those 22 OARSMT benchmarks. For the multilayer RSMT problem, we remove obstacles from the ten test cases in Section VI-A.

For the OARSMT algorithm, we compare our result with FOARS [8] and Liu *et al.* [6]. Over the 22 benchmarks, our algorithm generates 0.56% shorter wirelength than [8] but 0.38% longer wirelength than [6] on the average. Moreover, the run time of our algorithm is about three times and four times that of [8] and [6], respectively. The speed performance of our algorithm is worse than [6] and [8] because the size of the 2D visibility graph is $O(n \log n)$, while the size of the routing graphs in [8] and [6] are both linear.

For the RSMT problem, we compare our algorithm with FLUTE-3.0 [9], which is a well-known software for the RSMT construction and can guarantee an optimal solution for a signal net with at most nine pin-vertices. Over the 22 benchmarks, our algorithm generates 0.77% longer wirelength than [9] on the average, and the run time of our algorithm is about twice that of [9] on the average. However, the empirical time complexity for our algorithm and [9] is similar.

For the multilayer RSMT problem, compared with [11], our algorithm improves the cost by 2.29% on the average over the ten test cases, and the average speed up is about 86 times. Moreover, for the ten test cases, our algorithm regardless of obstacles improves the cost by 5.84% on the average than considering obstacles, and the average speed-up is about seven times. However, since both the multilayer RSMT problem and the ML-OARSMT problem are NP-hard, for a test case, our algorithm cannot guarantee to generate a better solution when all the obstacles are removed.

To sum up, although our algorithm naturally works for the RSMT and OARSMT problems, the efficiency and the effectiveness are worse than [9] for the RSMT problem and [6] for the OARSMT problem because our algorithm is designed for a multilayer model instead of the 2D plane, and does not take certain techniques only feasible for the 2D plane. On the other hand, the approaches in [6] and [9] could not be directly extended to a multilayer model. Moreover, the algorithms in [6] and [9] could be adopted to further improve the 2D sub-trees of the results generated by our algorithms for the multilayer RSMT and ML-OARSMT problems, respectively. Noticeably, aside from the input size $n$, the run time used for such refinement also depends on the number of 2D sub-trees.

## VII. Conclusion

For the ML-OARSMT problem and the OAPD-ST problem, we have employed computational geometry techniques to develop the first subquadratic-time algorithm for practical applications. In order to employ computational geometry techniques, we have proposed a reduction to transform a multilayer instance into a 3D instance, and we believe that this reduction will provide a way to employ more computational geometry techniques. Moreover, the computational geometry techniques employed may also be useful for other routing problems. Besides, we have developed an advanced Steiner point selection, which avoids selecting inferior Steiner points and works for many routing graphs. Experimental results show that our algorithm behaves like a subquadratic-time algorithm for practical applications, and when compared with existing approaches, our algorithm provides a solution with excellent quality and achieves a significant speed-up.

In the future, we plan to study the ML-OARSMT and OAPD-ST problems with different unit wire costs in different layers. Moreover, since the advanced Steiner point selection method guarantees the optimal solution of at most four pin-vertices on certain routing graphs, we also consider developing a better Steiner point selection method to improve the optimality guarantee.

## References


[1] C.-H. Liu, I.-C. Chen, and D. T. Lee, "An efficient algorithm for multilayer obstacle-avoiding rectilinear Steiner tree construction," in *Proc. 49th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, San Francisco, CA, USA, 2012, pp. 613–622.

[2] 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.

[3] J. Long, H. Zhou, and S. Memik, "EBOARST: An efficient edge-based obstacle-avoiding rectilinear Steiner tree construction algorithm," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 12, pp. 2169–2182, Dec. 2008.

[4] L. Li and E. F.-Y. Young, "Obstacle-avoiding rectilinear Steiner tree construction," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2008, pp. 523–528.

[5] C.-H. Liu, S.-Y. Yuan, S.-Y. Kuo, and Y.-H. Chou, "An $O(n \log n)$ path-based obstacle-avoiding algorithm for rectilinear Steiner tree construction," in *Proc. Design Autom. Conf. (DAC)*, San Francisco, CA, USA, 2009, pp. 314–319.

[6] C.-H. Liu *et al.*, "Obstacle-avoiding rectilinear Steiner tree construction: A Steiner-point-based algorithm," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 7, pp. 1050–1060, Jul. 2012.

[7] T. Huang and E. F.-Y. Young, "Obstacle-avoiding rectilinear Steiner minimum tree construction: An optimal approach," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2010, pp. 610–613.

[8] G. Ajwani, C. Chu, and W.-K. Mak, "FOARS: FLUTE based obstacle-avoiding rectilinear Steiner tree construction," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 2, pp. 194–203, Feb. 2011.

[9] C. Chu and Y.-C. Wong, "FLUTE: Fast lookup table based rectilinear Steiner minimal tree algorithm for VLSI design," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 1, pp. 70–83, Jan. 2008.

[10] T. Huang and E. F.-Y. Young, "An exact algorithm for the construction of rectilinear Steiner minimum trees among complex obstacles," in *Proc. 48th Design Autom. Conf. (DAC)*, San Diego, CA, USA, 2011, pp. 164–169.

[11] C.-W. Lin, S.-L. Hunag, K.-C. Hsu, M.-X. Lee, and Y.-W. Chang, "Multilayer obstacle-avoiding rectilinear Steiner tree construction based on spanning graph," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 11, pp. 2007–2016, Nov. 2008.

[12] 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. Int. Symp. Phys. Design (ISPD)*, Portland, OR, USA, 2008, pp. 118–125.

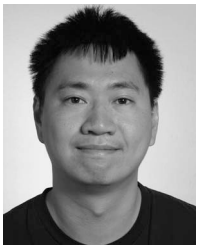
[13] J.-R. Chuang and J.-M. Lin, "Efficient multi-layer obstacle-avoiding preferred direction rectilinear Steienr tree construction," in *Proc. ASP-DAC*, Yokohama, Japan, 2011, pp. 527–532.

[14] K. L. Clarkson, S. Kapoor, and P. M. Vaidya, "Rectlinear shortest paths through polygonal obstacles in $O(n \log^2 n)$ time," in *Proc. 3rd Annu. ACM Symp. Comput. Geom. SCG*, New York, NY, USA, 1987, pp. 251–257.

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

[16] D. T. Lee and C. D. Yang, "Finding rectilinear paths among obstacles in a two-layer interconnection model," *Int. J. Comput. Geom. Appl.*, vol. 7, no. 6, pp. 581–598, 1997.

[17] K. Mehlhorn, "A faster approximation algorithm for the Steiner problem in Graphs," *Inf. Process Lett.*, vol. 27, no. 3, pp. 125–128, Mar. 1988.
[18] B. Chazelle, "An algorithm for segment-dragging and its implementation," *Algorithmica*, vol. 3, nos. 1–4, pp. 205–221, 1988.
[19] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed. Berlin, Germany: Springer, 2008.
[20] E. Wein and J. Benkoski, *Hard Macros Will Revolutionize SoC Design*. EEdesign, 2004.
[21] D. T. Lee, C. D. Yang, and C. K. Wong, "On bends and distances of paths among obstacles in two-layer interconnection model," *IEEE Trans. Comput.*, vol. 43, no. 6, pp. 711–724, Jun. 1994.
[22] M. Hanan, "On Steiner's problem with rectilinear distance," *SIAM J. Appl. Math.*, vol. 14, no. 2, pp. 255–265, Mar. 1966.
[23] J. L. Ganley and J. P. Cohoon, "Routing a multi-terminal critical net: Steiner tree construction in the presence of obstalce," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, London, U.K., 1994, pp. 113–116.



**Chih-Hung Liu** received the B.S. degree in computer science and the Ph.D. degree in electronics engineering, both from the National Taiwan University, Taipei, Taiwan, in 2005 and 2009, respectively. He received the Humboldt Research Fellowship for Post-Doctoral Researchers in 2012.

He is currently a Post-Doctoral Researcher with the Department of Computer Science, University of Bonn, Bonn, Germany. His current research interests include graph algorithms, computational geometry, combinatorial optimization, and their applications to very large scale integration designs, especially rectilinear Steiner tree construction, and Voronoi diagrams.

**Chun-Xun Lin** received the B.S. degree in electrical engineering from the National Cheng Kung University, Tainan, Taiwan, and the M.S. degree in electronics engineering from the Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, Taiwan, in 2009 and 2011, respectively. He is currently pursuing the Ph.D. degree in electrical and computer engineering from the University of Illinois at Urbana-Champaign, Urbana, IL, USA.

His current research interests include very large scale integration physical design, combinatorial optimization, and computational geometry.

**I-Che Chen** receievd the B.S. degree in electrical engineering from the National Taiwan University, Taipei, Taiwan, in 2010. He is currently pursuing the Ph.D. degree in computer science and engineering from the University of Michigan, Ann Arbor, MI, USA.

His current research interests include approximate computing, which achieves low-power and low-area design by producing small errors intentionally.

**D. T. Lee** (S'75–M'78–SM'84–F'92) received the B.S. degree in electrical engineering from the National Taiwan University, Taipei, Taiwan, in 1971, and the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana-Champaign, Urbana, IL, USA, in 1976 and 1978, respectively.

He is currently with the Department of Computer Science and Engineering, National Chung Hsing University, Taichung, Taiwan, where he has been a President and a Distinguished Chair Professor, since 2011. He is on leave from the Institute of Information Science, Academia Sinica, Taipei, Taiwan, where he has been a Distinguished Research Fellow, since 1998. From 1978, he was a Professor at the Department of Electrical and Computer Engineering, Northwestern University, Chicago, IL, USA. His current research interests include design and analysis of algorithms, computational geometry, very large scale integration layout, algorithm visualization, software security, bio-informatics, and digital libraries. He has published over 150 technical articles in scientific journals and conference proceedings, and is an Editor of *Algorithmica*, the *International Journal of Information and Computer Security*, Co-Editor-in-Chief of the *International Journal of Computational Geometry and Applications*, *LNCS Transactions on Computational Science*, and a Series Editor of *Lecture Notes Series on Computing* (World Scientific Publishing Company, Singapore).

Prof. Lee was a Research Awardee and an Ambassador Scientist, Alexander von Humboldt Foundation, Germany. He is a fellow of ACM, Academician of Academia Sinica, and a member of The World Academy of Sciences.

**Ting-Chi Wang** received the B.S. degree in computer science and information engineering from the National Taiwan University, Taipei, Taiwan, and the M.S. and Ph.D. degrees in computer science from the University of Texas at Austin, Austin, TX, USA. He is currently a Professor with the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan. His current research interests include very large scale integration design automation.

Dr. Wang was the recipient of the Best Paper Award from 2006 Asia and South Pacific Design Automation Conference (ASP-DAC) for his work on redundant via insertion. He supervised a team to win the first place at the 2008 International Symposium on Physical Design (ISPD) Global Routing Contest. He has served on the technical program committees of major conferences on electronic design automation, including ASP-DAC, Design Automation Conference, Design, Automation & Test in Europe, International Conference on Computer-Aided Design, and ISPD.