國立交通大學

電子工程學系 電子研究所碩士班

碩 士 論 文

系統晶片及奈米技術下

直角史坦那樹之建構

Unification of Rectilinear Steiner Tree Construction for SoC and Nanometer Technologies

研 究 生：余彥廷

指導教授：江蕙如 博士

中 華 民 國 九 十 八 年 六 月

系統晶片及奈米技術下直角史坦那樹之建構

Unification of Rectilinear Steiner Tree Construction for SOC and Nanometer Technologies

研究生：余彥廷　　Student: Yen-Ting Yu

指導教授：江蕙如 博士　　Advisor: Dr. Iris Hui-Ru Jiang

國 立 交 通 大 學

電子工程學系電子研究所碩士班

碩 士 論 文

A Thesis
Submitted to Department of Electronics Engineering & Institute of Electronics
College of Electrical and Computer Engineering
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of
Master
in
Electronics Engineering

June 2009

Hsinchu, Taiwan, Republic of China

中華民國 九十八年六月

系統晶片及奈米技術下直角史坦那樹之建構

學生：余彥廷　　指導教授：江蕙如 博士

國立交通大學

電子工程學系　電子研究所碩士班

摘　　要

直角史坦那最小樹是實體設計上的一個必要問題，此外，在製程上有不同的限制，包括障礙物的避開、多層繞線、特定層的繞線方向，在系統晶片及奈米技術下的直角史坦那最小樹的建構上是不能被忽略的。這篇論文首先統合單層及多層避開障礙物的直角史坦那最小樹的建立，之後將其延伸到考慮特定繞線方向以及時間驅動的直角史坦那最小樹。這些延伸說明了我們的演算法可以很容易的適用到這些結構上，實驗結果也顯示我們的演算法超越文獻中的最佳結果。

UNIFICATION OF RECTILINEAR STEINER TREE CONSTRUCTION FOR SOC AND NANOMETER TECHNOLOGIES

Student: Yen-Ting Yu Advisor: Dr. Iris Hui-Ru Jiang

Department of Electronics Engineering

Institute of Electronics

National Chiao Tung University

Abstract

The rectilinear Steiner minimal tree (RSMT) problem is essential in physical design. Moreover, the variant constraints for fabrication issues, including obstacle avoidance, multiple routing layers, layer-specific routing directions, cannot be ignored during RSMT construction for modern SoC and nanometer technologies. This thesis unifies single- and multi-layer obstacle-avoiding RSMT construction first and then extends it to consider preferred routing directions and to target timing-driven RSMT. These extensions demonstrate that our algorithm can easily be adapted to configurations. Experimental results show that our algorithm is promising and outperforms the state-of-the-art works.

Acknowledgements

I would like to express my heartfelt gratitude to my advisor, Prof. Iris Hui-Ru Jiang. I've learned several ways to cope with a tough research problem from her guidance. She is the teacher who is willing to spend her precious time just to help you preparing your oral examination. Meanwhile, I appreciate my lab members, especially Wan-Yu Lee for her foods. Finally, I display my warmest appreciation to my parents for their love and support.

Yen-Ting Yu

National Chiao Tung University

June 2009

Table of Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

Rectilinear Steiner minimal tree (RSMT) construction has been extensively studied and considered as a fundamental problem in physical design. An RSMT is a tree of rectilinear edges connecting a given set of points possibly through some extra (i.e., Steiner) points with minimum total wire length; it is frequently performed for interconnect estimation during floor planning, placement, and routing stages. To make the estimation practical, we shall consider the fabrication issues for modern SoC and nanometer process technologies. Advanced nanometer technology offers an abundance of routing layers, e.g., 11 layers in 65 nm [1], and normally assigns a preferred routing direction to each layer; on the other hand, a large-scale SoC design often contains a tremendous number of obstacles. If timing is the main concern, the objective could be changed to minimize the path length from a designated source point to the rest.

However, even the simplest case, the RSMT problem without considering obstacle avoidance (OA), multiple layers (ML), preferred direction (PD) constraints, has been proven to be NP-complete [2]. Due to the high complexity and frequent usage, it is desired to construct an RSMT with these constraints of good quality in reasonable runtime.

A 2:1 performance bound of minimum spanning tree (MST) to RSMT for general graphs can be applied to these variations. Thus, existing approaches for RSMT typically contain three steps:

1) Connection graph generation (CG): Step 1 generates a connection graph to connect all pins. (Obstacle boundaries can also be included.) This graph contains

geometrical proximity information among pins (and obstacle boundaries, or not). It can be a complete graph, a spanning graph, an escape graph [3], or a Delaunay triangulation (DT) [4]. Fig. 1 illustrates these possibilities for a planar instance with 4 pins and 5 obstacles. Fig. 1(a) shows the corresponding complete graph where vertices represent pins and each edge reflects the wirelength between the related two pins with obstacle consideration. Fig. 1(b) shows the corresponding escape graph where lines are stretched horizontally and vertically along pins and the corners of obstacles, and the intersection of any two lines contributes a vertex. Fig. 1(c) shows the corresponding spanning graph where vertices represent pins and the corners of obstacles, and two vertices are connected if there is no other vertex inside or on the boundary of the bounding box of the two vertices, and there is no obstacle inside the bounding box of the two vertices [9], [11]. Fig. 1(d) shows the corresponding DT where vertices represent pins and the corners of obstacles, and the circumcircle of each triangle does not contain any other vertex. In addition, the corners of obstacles may be not included in DT to reduce the complexity of the connection graph, as shown in Fig. 1(e).

2) spanning tree construction (ST): Step 2 constructs a minimum spanning tree (MST) over all pins based on the connection graph. If the connection graph includes the corners of obstacles, the MST is obstacle-avoiding (all tree edges bypass obstacles). Otherwise, it is obstacle-weighted (tree edges may run through obstacles, but the impacts of obstacles are considered into edge weights), or mixed (tree edges are obstacle-weighted first and then obstacle-avoiding). If the goal is to construct a timing-driven RSMT, the spanning tree can be the shortest path tree (SPT) on the connection graph instead and is built up by Dijkstra's shortest path algorithm.

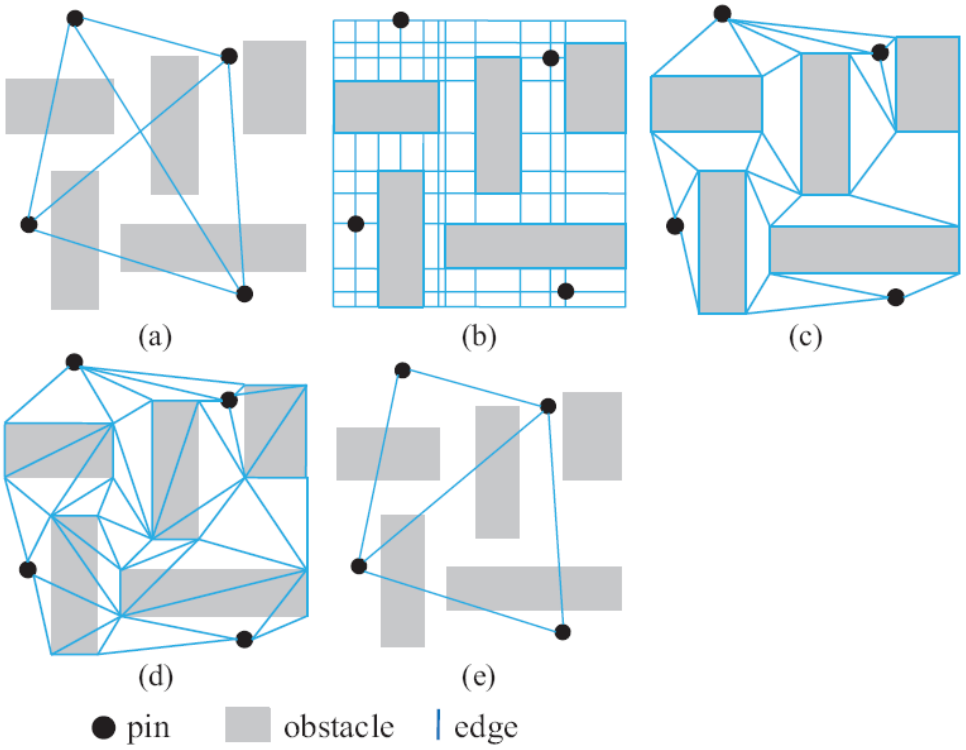


Fig. 1. Given an instance with 4 pins and 5 obstacles, the possible options for the connection graph can be (a) a complete graph, (b) an escape graph, (c) a spanning graph, (d) a Delaunay triangulation with obstacles, or (e) a Delaunay triangulation without obstacles.

3) Rectilinearization and refinement (RR): Step 3 transforms the spanning tree into a rectilinear Steiner tree and refines the total cost. If the spanning tree is obstacle-weighted, the edges intersecting obstacles are fixed during rectilinearization. The total cost includes wirelength and vias. Planar U-shaped pattern refinement is usually applied. In addition, if the connection graph is a Hanan grid or an escape graph, some works merge steps 2 and 3 into one.

As listed in TABLE I, we compare the configurations provided and the techniques used in each step for the state-of-the-art works and ours.

Recently, most of research endeavors have focused on single-layer obstacle-avoiding RSMT (SL-OARSMT) [6], [7], [8], [9], [10]. Among them, [9]

produced the best results; the breakthrough done in [9] was to include more “essential edges” into their spanning graph. The essential edge introduced by [9] can directly connect two pins without obstacles inside their bounding box and then lead to more desirable solutions. [11] then extended [9] to construct a 3D spanning graph and solved the multi-layer variation; so far, it has been the first one and only one work handling multi-layer obstacle-avoiding RSMT (ML-OARSMT). [11] projected vertices between layers and within layers to link the spanning graphs for adjacent layers together. The projection reflects the usage of vias. Even so, it seems somewhat indirect to include the information of preferred directions into their 3D spanning graph.

[12] first included preferred directions into RSMT but ignored obstacles. [13] first attempted to combine all of these issues into RSMT construction and formulated the obstacle-avoiding preferred direction Steiner tree problem (OAPDST). In addition, [13] directly constructed a rectilinear MST over a 3D improved escape graph. However, the MST was not further refined, so the solution quality may be limited. Each work listed here focused on only one specific configuration and cannot easily be adapted to other configurations.

As shown in Fig. 2, in this thesis, we generalize an approximation algorithm which is a preliminary version announced in [14], [15] to unify the tree construction. Steps 1 and 2 construct an obstacle-weighted MST/SPT on the DT of pins only. Step 3 rectilinearizes each tree edge on a 3D extended escape graph and then refines it.

Our innovative features include:

1) We develop a unified approach to Steiner tree construction for variant configurations.

2) The conventional construction-by-correction approach cannot extract the global

geometrical information among pins and obstacles in the connection graph. We overcome the drawback by introducing potentially essential edges during DT construction and adequately associating the impacts of obstacles into edge weights.

3) We construct the DT and the obstacle-weighted MST/SPT in an efficient way since the total edge weight of the tree is not required to be exact, just expected to be correlated to the final tree. It can effectively guide step 3 how to connect pairs of pins.

4) We generalize 3D U-shaped pattern refinement. Experimental results show that our algorithm outperforms the state-of-the-art works for SL-OARSMT, ML-OARSMT and OAPDST and can extend to handle timing-driven RSMT.

Moreover, our results reveal the following findings: The guidance of the obstacle-weighted MST leads to smaller total costs and shorter runtimes, and novel 3D U-shaped refinement works well not only on our algorithm but also for previous work.

The rest of the thesis is organized as follows. Chapter 2 presents problem formulations about ML-OARSMT and how our algorithm works. The extensions of ML-OARSMT are presented in Chapter 3. Experimental results are presented in Chapter 4. Conclusions are drawn in Chapter 5. And finally, APPENDIX shows an application of our procedure.

TABLE I

THE COMPARISON BETWEEN RECENT WORKS ON RSMT

Approach	Configuration				Procedure		
	ML[1]	OA[2]	PD[3]	TD[4]	Step 1: CG (obstacles included)	Step 2: ST	Step 3: RR
[6]	SL	Y	N	N	Delaunay triangulation (Y)	Obstacle-avoiding MST	-
[7]	SL	Y	N	N	Spanning graph (Y)	Obstacle-avoiding MST	No refinement
[8]	SL	Y	N	N	Complete graph (N)	Mixed obstacle-weighted & obstacle-avoiding MST	-
[9]	SL	Y	N	N	Improved spanning graph (Y)	Obstacle-avoiding MST	-
[10]	SL	Y	N	N	Sparse spanning graph (Y)	Obstacle-avoiding MST	-
[11]	ML	Y	N	N	3D improved spanning graph (Y)	Obstacle-avoiding MST	-
[12]	ML	N	Y	N	3D Hanan grid (N)	-	-
[13]	ML	Y	Y	N	3D improved escape graph (Y)	Rectilinear & obstacle-avoiding MST	N/A
Ours	ML	Y	Y	Y	Delaunay triangulation (N)	Obstacle-weighted MST/SPT[5]	3D refinement

[1]ML: Multi-layer; SL: Single-layer.
[2]OA: Obstacle-avoidance.
[3]PD: Preferred direction.
[4]TD: Timing-driven.
[5]SPT: Shortest path tree for timing-driven RSMT.

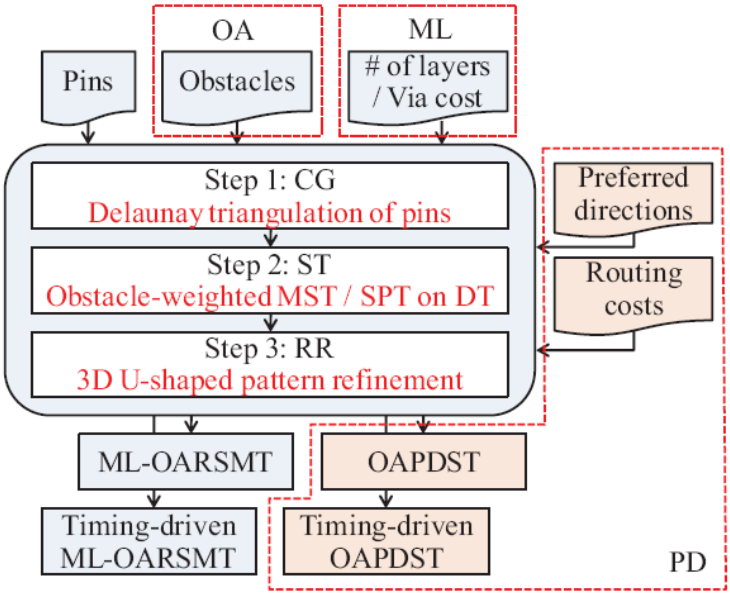


Fig. 2. We unify rectilinear Steiner tree construction. Our method can handle different configurations, e.g., obstacle avoidance, multiple routing layers, preferred direction, and timing-driven.

Chapter 2

PROBLEM FORMULATION AND ALGORITHM

We adopt the formulation of the multi-layer obstacle-avoiding rectilinear Steiner minimal tree (ML-OARSMT) problem in [11].

> ***Problem: Multi-Layer Obstacle-Avoiding Rectilinear Steiner Minimal Tree (ML-OARSMT):*** Given the equivalent wirelength cost C_v of a via, the number N_l of layers, a set $P = \{p_1, p_2, \ldots, p_m\}$ of pins, a set $O = \{o_1, o_2, \ldots, o_k\}$ of obstacles, construct a multi-layer rectilinear Steiner tree to connect all pins in P by only rectilinear edges, such that no tree edge or via intersects any obstacle in O and the total cost of the tree is minimized.

An obstacle is a rectangle on a layer, indicated by its four corner-vertices. A pin-vertex *pi* is a vertex (x_i, y_i, z_i) on layer z_i, while a via (x_j, y_j, z_j) on layer z_j is an edge between (x_j, y_j, z_j) and (x_j, y_j, z_j+1). No two obstacles can overlap with each other, but two obstacles can be point-touched or line-touched. Since an arbitrary rectilinear obstacle can be partitioned into a set of rectangles, without loss of generality, assume all obstacles are rectangular. All vertices of pins and vias must not locate inside any obstacle, but they can be at the corner or on obstacle boundaries. In addition, the single-layer obstacle-avoiding rectilinear Steiner minimal tree (SL-OARSMT) problem is a special case of ML-OARSMT as $N_l = 1$.

As outlined in Fig. 2, our algorithm is based on the construction-by-correction approach. Here, we use the example given in [11], depicted in Fig. 3(a), to demonstrate our algorithm. Assume $C_v = 3$, and each grid size is 20×20 (unit wirelength) ^{2}.

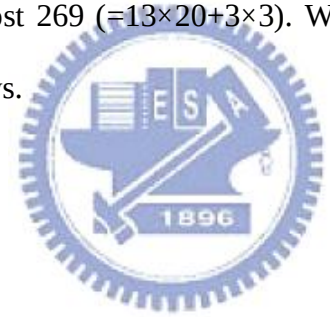
1) Step 1: All pins (actually only pins) are projected onto a pseudo plane, and their

DT is then constructed. During DT construction, some “illegal” edges (indicated by dotted lines) that may be essential are added. (The essential edges can lead to more desirable solutions [11].) (see Fig. 3(b) and Fig. 4(e)-(l))

2) Step 2*:* An obstacle-weighted minimum spanning tree is grown up over the DT. We bias the edge weights in DT to consider obstacle penalties. (see Fig. 3(b))

3) Step 3: Each tree edge is rectilinearized on a 3D extended escape graph (see Fig. 3(c)), and is then processed by novel 3D U-shaped pattern refinement. Our ML-OARSMT for this instance is of cost 195 (=9×20+5×3). (see Fig. 3(d))

In addition, the ML-OARSMT generated by [11], as shown in Fig. 7(a), is of cost 326 (=16×20+2×3). It can be improved by our refinement method; as shown in Fig. 7(c), the refined tree is of cost 269 (=13×20+3×3). We detail each step and analyze the time complexity as follows.



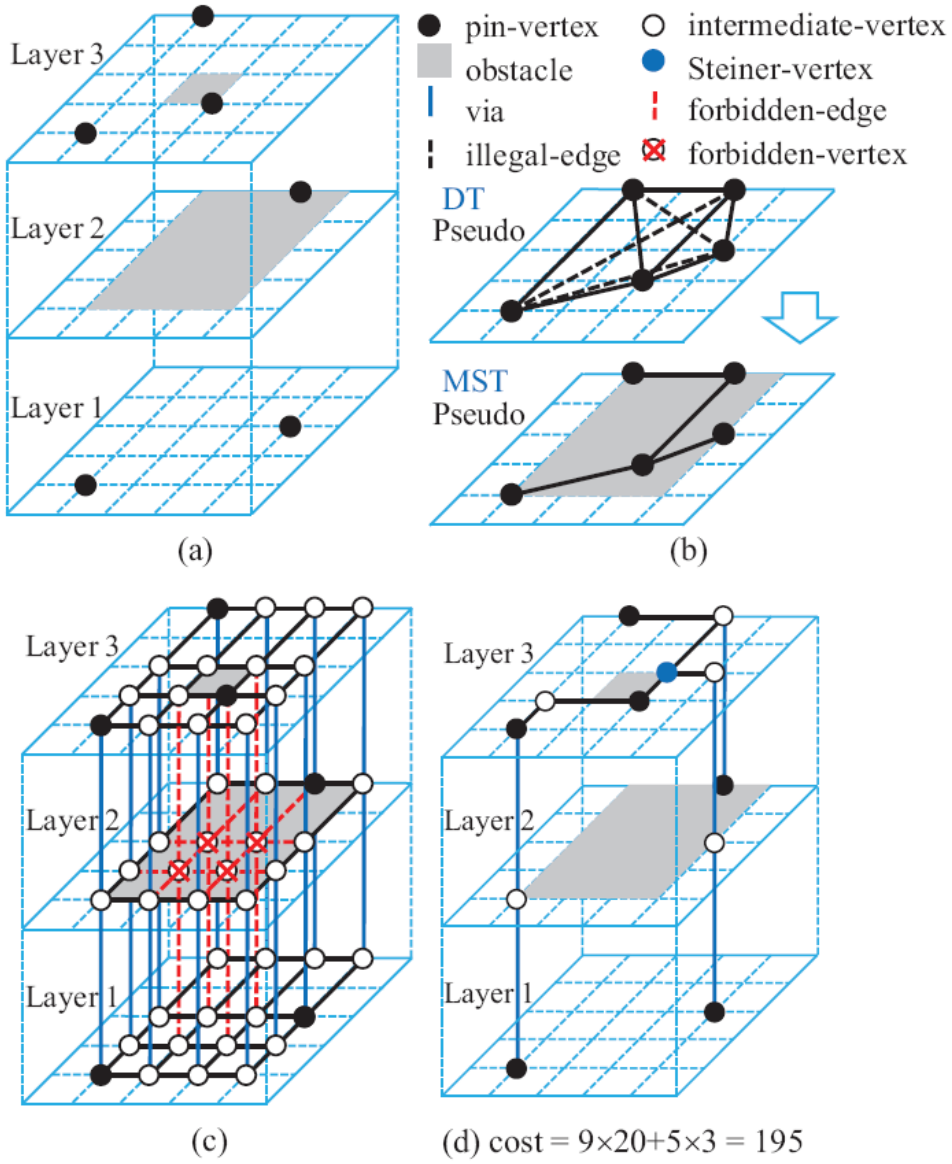


Fig. 3. (a) An instance of ML-OARSMT given in [11], where C_v = 3, each grid size is 20×20 (unit)^2. (b) Steps 1 and 2 of our algorithm for ML-OARSMT. (c) The corresponding 3D extended escape graph. (d) The resulting ML-OARSMT.

A. Delaunay Triangulation of Pins

Initially, all pins are projected onto a pseudo plane, i.e., each pin-vertex is indicated by its *x*- and *y*-coordinates. If two pin-vertices are projected to the same location, they are connected by an edge. Conceptually, this pseudo plane abstracts the geometrical proximity among pins, as well as views single-layer and multi-layer trees as one.

For a given set P of vertices in a plane, a Delaunay triangulation DT(P) is a triangulation such that the circumcircle of each triangle does not contain any other vertex of P. A DT(P) maximizes the minimum angle of all the angles of the triangles in it, thus avoiding sliver triangles, i.e., a DT(P) tends to connect neighboring vertices.

As depicted in Fig. 4(a), during DT construction, sometimes two triangles possibly violate the definition of DT, i.e., the circumcircle of one triangle contains another vertex. The common edge of these two triangles is an illegal edge, and it is then flipped to a legal edge [4], as shown in Fig. 4(b).

Assume x_{max} and y_{max} are the maximum x- and y-coordinates of the given set of vertices. DT construction begins with a large triangle with three vertices located at (0, $3y_{max}$), ($3x_{max}$, 0), and ($-3x_{max}$, $-3y_{max}$) on the pseudo plane. Then, one pin at a time is inserted. If it is located inside some triangle, it splits this triangle into three. (see Fig. 4(c)) Otherwise, it is on the common edge of two triangles, it then splits these into four. (see Fig. 4(d)) The dotted lines in Fig. 4(c)(d) are introduced by the inserted pin. If illegal edges are generated, they are then flipped into legal ones until no illegal edge remains. This process repeats until all pins are inserted. Finally, the initial large triangle and its induced edges are removed.

Normal DT construction discards these illegal edges; however, we preserve these illegal edges since they contain much more global information than legal ones and may lead to better solutions. Fig. 3(b) gives the corresponding DT of the instance in Fig. 3(a), where illegal edges are indicated by dotted lines, and legal edges are indicated by solid lines. Fig. 4(e)-(l) detail the DT construction step-by-step, where three illegal edges (indicated by dotted lines) are flipped in Fig. 4(h)(i), Fig. 4(j)(k), and Fig. 4(k)(l).

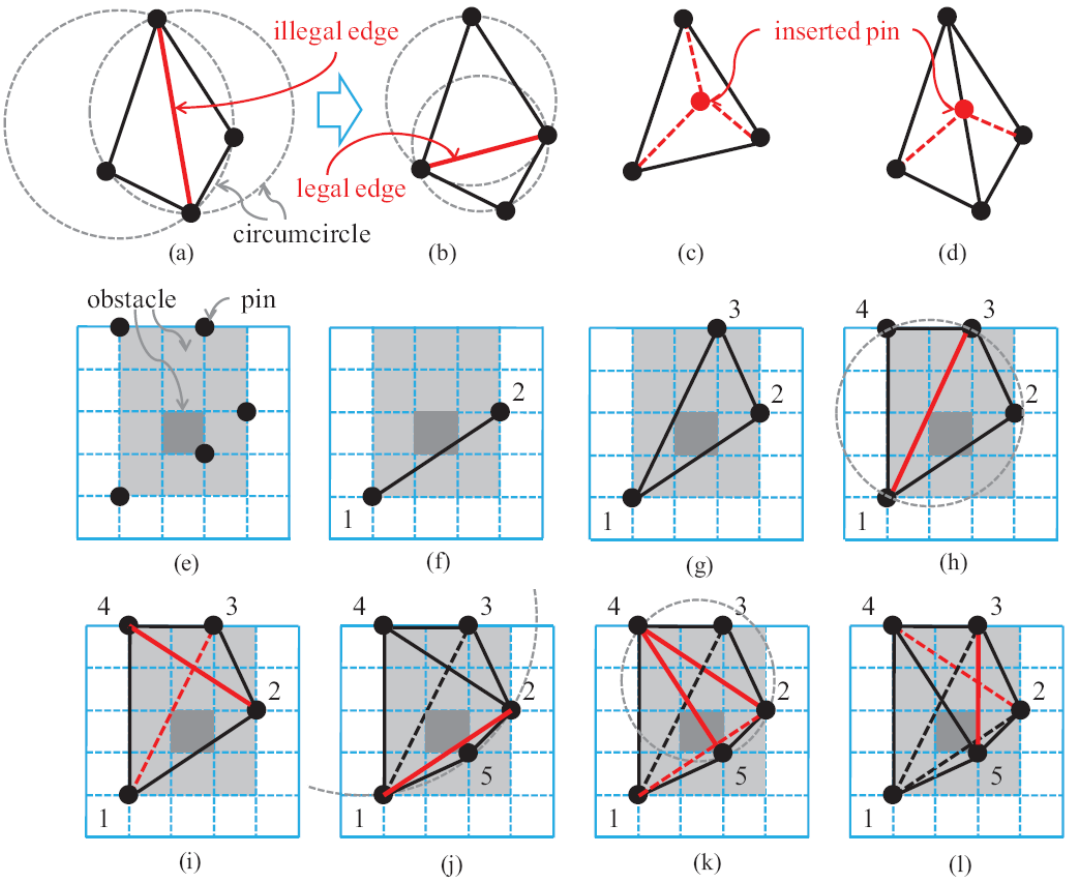


Fig. 4. (a)-(b) During DT construction, an illegal edge in is flipped into a legal one. (c)-(d) The inserted pin may be located inside one triangle or on the boundary of two triangles. (e) The pins and obstacles in the instance given in Fig. 3(a) are projected to the pseudo plane. (f)-(l) Step-by-step DT construction for the instance given in Fig. 3(a). The number attached beside each pin indicates its order to be included in DT. Please note that the edges between pins and the initial large triangle are not shown.

B. Obstacle-Weighted MST on DT

As shown in Fig. 3(b), after the DT is constructed for the projected pins on a pseudo plane, the obstacle-weighted minimum spanning tree is constructed based on Kruskal's algorithm [5]. (Another option is Prim's algorithm [5].)

The conventional construction-by-correction approach does not include the geometrical information of obstacles in the connection graph. To overcome this

drawback, we encode the obstacle penalties to edge weights of DT(P). Because DT(P) contains potentially essential edges and its edge weights include the obstacle information, DT(P) possesses the global geometrical information among pins and obstacles.

On the other hand, the MST is used to guide step 3 how to connect pins. The edge weight is not required to be exact, just expected to be correlated to the cost of the final RSMT. Hence, we use a simple and fast, yet effective, formula to estimate the impact of obstacles. The obstacle penalty $op(p_i, p_j)$ between two pins p_i and p_j located on the same layer is simplified from [8], where only the obstacles completely passing through the bounding box between p_i, p_j horizontally or vertically are counted. Fig. 5 shows two examples for obstacle penalty computation. The pair of pins in Fig. 5(a) has no obstacles completely passing through their bounding box. If Z-shaped routing is applied, there is no routing overhead. Hence, their obstacle penalty equals zero. On contrast, in Fig. 5(b), one obstacle crosses over the bounding box of the given pair of pins. The detour incurs either $2l_1$ or $2l_1$ penalty, so their obstacle penalty can be the smaller one.

In addition, we introduce a parameter α to further reflect the congestion of obstacles; in our experiments, α is computed by the density of obstacles. When two pins are located at the same layer with nonzero obstacle penalty or at different layers, the parameter α is used to magnify their distance. The edge weight $w(p_i, p_j)$ is computed as follows.

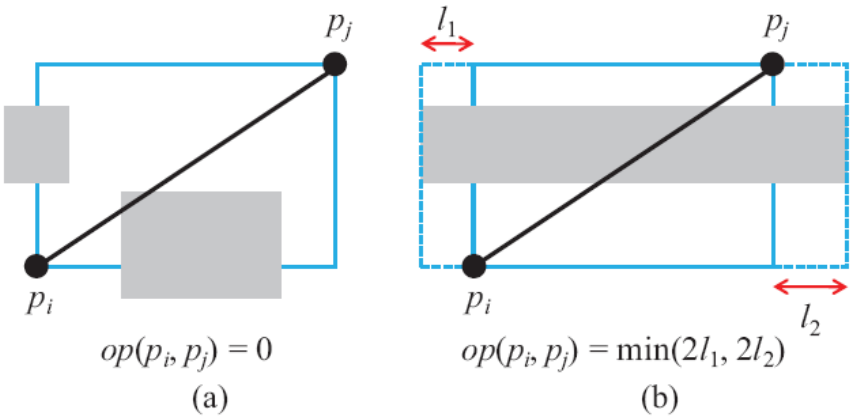


Fig. 5. The obstacle penalty is counted for two pins located on the same layer. (a) Z-shaped routing can avoid obstacle penalties. (b) The obstacle completely passes through the bounding box, and its obstacle penalty is computed as the smaller detour.

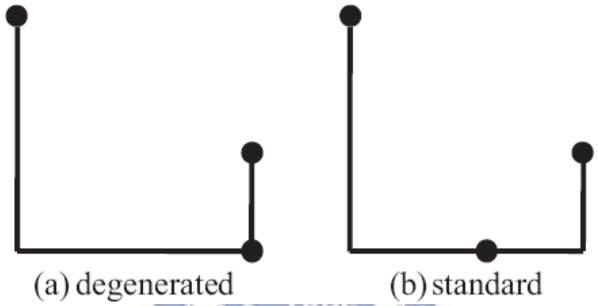


Fig. 6. All 3D U-shaped patterns are classified into (a) degenerated and (b) standard ones.

As p_i, p_j are on different layers, $z_i \neq z_j$,

$w(p_i, p_j) = \alpha \cdot (|x_j - x_i| + |y_j - y_i| + Cv \cdot |z_j - z_i|)$.

As p_i, p_j are on the same layer, $z_i = z_j$,

If $op(p_i, p_j) = 0$, $w(p_i, p_j) = |x_j - x_i| + |y_j - y_i|$;

otherwise, $w(p_i, p_j) = \alpha \cdot (|x_j - x_i| + |y_j - y_i| + op(p_i, p_j))$.

Although we estimate the obstacle penalties in a simple way, our results reveal that steps 1 and 2 are necessary, and they can give a good guidance for step 3.

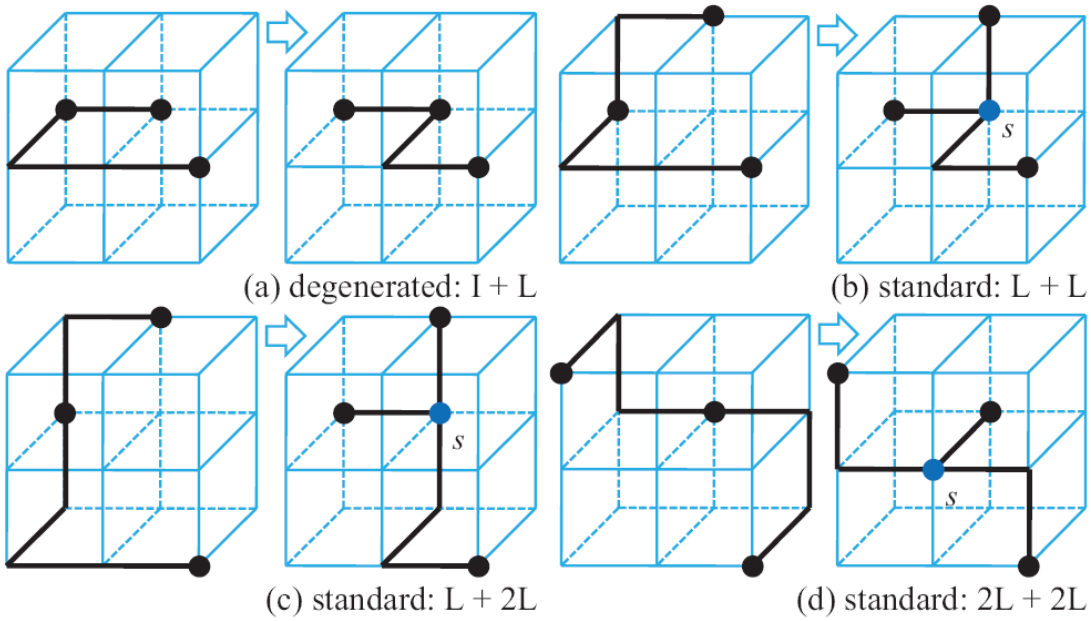


Fig. 7. Several cases for 3D U-shaped pattern refinement. (*s*: Steiner-vertex)

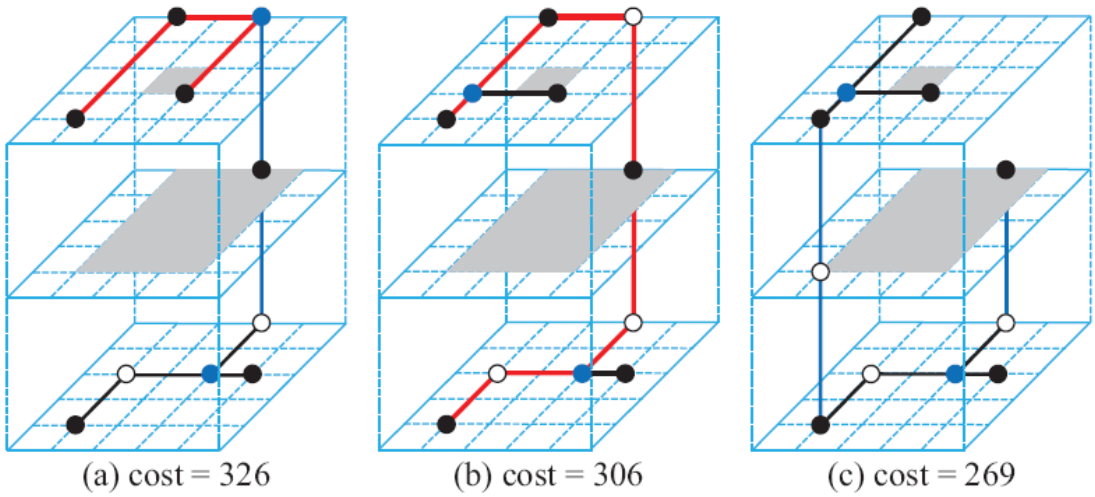


Fig. 8. (a) The ML-OARSMT for Fig. 3(a) generated by [11] is of cost 326. It has a degenerated pattern, marked by bold lines. (b) The refined tree has a complicated standard pattern and an obstacle around. (c) The resulting tree is of cost 269.

C. Rectilinearization and 3D U-Shaped Pattern Refinement

Based on the guidance of the obstacle-weighted MST, each MST edge is rectilinearized and then refined if a 3D U-shaped pattern is found. Rectilinearization

and 3D U-shaped pattern refinement are compounded into one operation and are iteratively applied edge-by-edge. By doing so, the refinement done for early edges can benefit consequent edges, thus our refinement does not always hurt runtimes. In addition, the MST edges are processed in a random order.

Rectilinearization is performed on a 3D extended escape graph based on Dijkstra's shortest path algorithm [5]. We extend the planar escape graph [3] to a 3D one as follows. A 3D extended escape graph is constructed by stretching lines from all pin-vertices and the corner-vertices of all obstacles along *x*-, *y*-, and *z*-axes, where the line segments intersecting or passing through obstacles are prohibited to be used. Fig. 3(c) shows the 3D extended graph for the instance in Fig. 3(a). To make the implementation flexible, we adequately associate forbidden flags to the vertices inside obstacles and at obstacle boundaries.

By extending the proof done in [3], we can prove by induction that at least one optimal solution of ML-OARSMT is embedded in the 3D extended escape graph. This fact holds even if the via cost and the number of layers vary. Hence, the 3D extended escape graph does not keep our solution away from optimality. On the other hand, the proof of the 2:1 performance bound of MST to SMT for general graphs can be applied here [5].

When a tree edge is rectilinearized and connected to the partially constructed rectilinear Steiner tree, a U-shaped pattern may be formed. We generalize U-shaped pattern refinement from 2D to 3D cases. Here, we only consider the U-shaped patterns that can potentially be optimized. We have the following theorem for U-shaped patterns without obstacles inside or around.

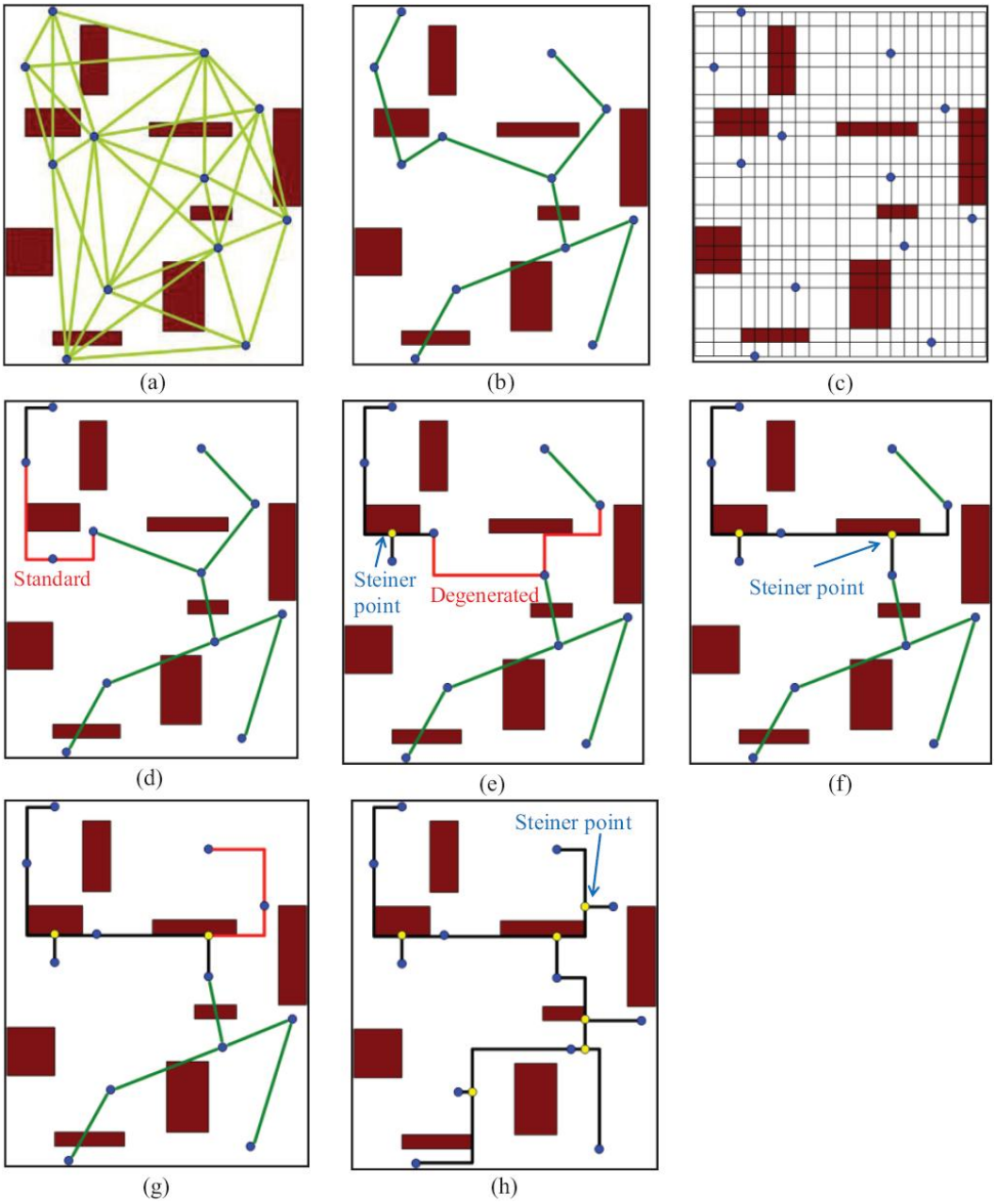


Fig. 9. An instance of SL-OARSMT. (a) Step 1: Delaunay triangulation for pins on a pseudo plane. (b) Step 2: The obstacle-weighted MST. (c)-(h) Step 3:Rectilinearization and 3D U-shape refinement, where (c) is the escape graph, (d)-(g) are intermediate trees, and (h) is the resulting SL-OARSMT.

Theorem: A 3D U-shaped pattern is formed by at least three vertices (pin-vertices and/or Steiner-vertices). It can be either degenerated if the middle vertex is located at one turning corner of the U or standard if the middle vertex is located within the

middle segment of the U.

Proof Sketch: If there is no obstacle inside or around a U-shaped pattern, a 2-vertex U-shape never occurs because rectilinearization is performed based on Dijkstra's shortest path algorithm. If there are obstacles inside, the 2-vertex U-shaped pattern results from detours. Thus, a U-shaped pattern has at least three vertices.

A U-shaped pattern with 4 or more vertices can be decomposed into several smaller ones. For a 3-vertex U-shaped pattern, only the location of the middle vertex may vary. Moreover, the middle vertex cannot be located within one of two I-shaped segments of the U. In this case, this 3-vertex U-shaped pattern is composed of one I-shaped segment plus a 2-vertex U-shaped pattern. As mentioned above, a 2-vertex U-shaped pattern never occurs. Hence, the middle vertex can be located either at one turning corner of the U or within the middle segment of the U.

1) Degenerated U-shape: The middle vertex is located in one turning corner of U. (see Fig. 6(a)) This type can be identified by one I-shaped segment plus one or more L-shaped segments. The refinement can be applied only when three vertices are located in the same plane, i.e., they have the same *x*-, *y*-, or *z*-coordinate. The L-shaped segments of the U can then be rerouted for cost reduction.

2) Standard U-shape: The middle vertex is located within the middle segment of the U. (see Fig. 6(b)) This type can be identified by several L-shaped segments plus several L-shaped segments. The L-shaped segments of a standard one can be ripped-up and then rerouted from these vertices to their Steiner-vertex. (The optimal location of the Steiner-vertex is at the median of the coordinates of these three vertices.)

More complicated cases can be decomposed into smaller ones. Fig. 7 lists several

examples for 3D U-shaped pattern refinement. Please note that our classification is complete. Fig. 8 shows the instance in Fig. 3(a) can be further improved by fixing one degenerated pattern plus one standard U-shaped one. The cost is reduced from 326 to 269. In addition, for this case, our algorithm can generate the tree in Fig. 3(d) of cost 195, even without refinement.

For easier visualization, Fig. 9 demonstrates a planar example with 12 pins and 8 obstacles; as mentioned in Section II, SL-OARSMT is a special case of ML-OARSMT. Fig. 9(a) depicts the corresponding DT, where illegal edges are also included. Then, based on the edge weight defined in Section III.B, Fig. 9(b) shows the corresponding MST. Fig. 9(c) shows the corresponding escape graph. Based on the MST in Fig. 9(b), rectilinearization and refinement starts from an edge randomly selected, say the edge at the up-left corner in this case. As shown in Fig. 9(d), after an edge is rectilinearized, a standard U-shaped pattern is found and refined. Fig. 9(e) shows a degenerated U-shaped pattern. It can be seen that if there are obstacles around a U-shaped pattern, e.g., Fig. 9(d)(g), the Steiner-vertex might have to be shifted accordingly; even so, we still can improve the total cost. (see Fig. 9(e)(h)) If the refined pattern has worse cost, the original one retains.

D. Time Complexity Analysis

Let $n=m+4k$ for an instance with m pins and k obstacles. Step 1 takes $O(m\lg m)$ time for DT construction [4]; step 2 takes $O(m(\lg m)^2)$ time for Kruskal's algorithm; step 3 takes $O(n^3)$ time for the 3D extended escape graph construction, Dijkstra's algorithm, 3D U-shaped pattern refinement. As mentioned in Section III.B, steps 1 and 2 can effectively guide step 3, and they have low time complexities, so they are worthwhile. Although 3D U-shaped pattern refinement in step 3 has a high time

complexity, it can be expected to produce good solutions.

Compared with [11], steps 1 and 2 of our algorithm have relatively low time complexities, and step 3 has the same order complexity. Since the time complexities are the same, it would be a good decision to take time on sophisticated refinement.

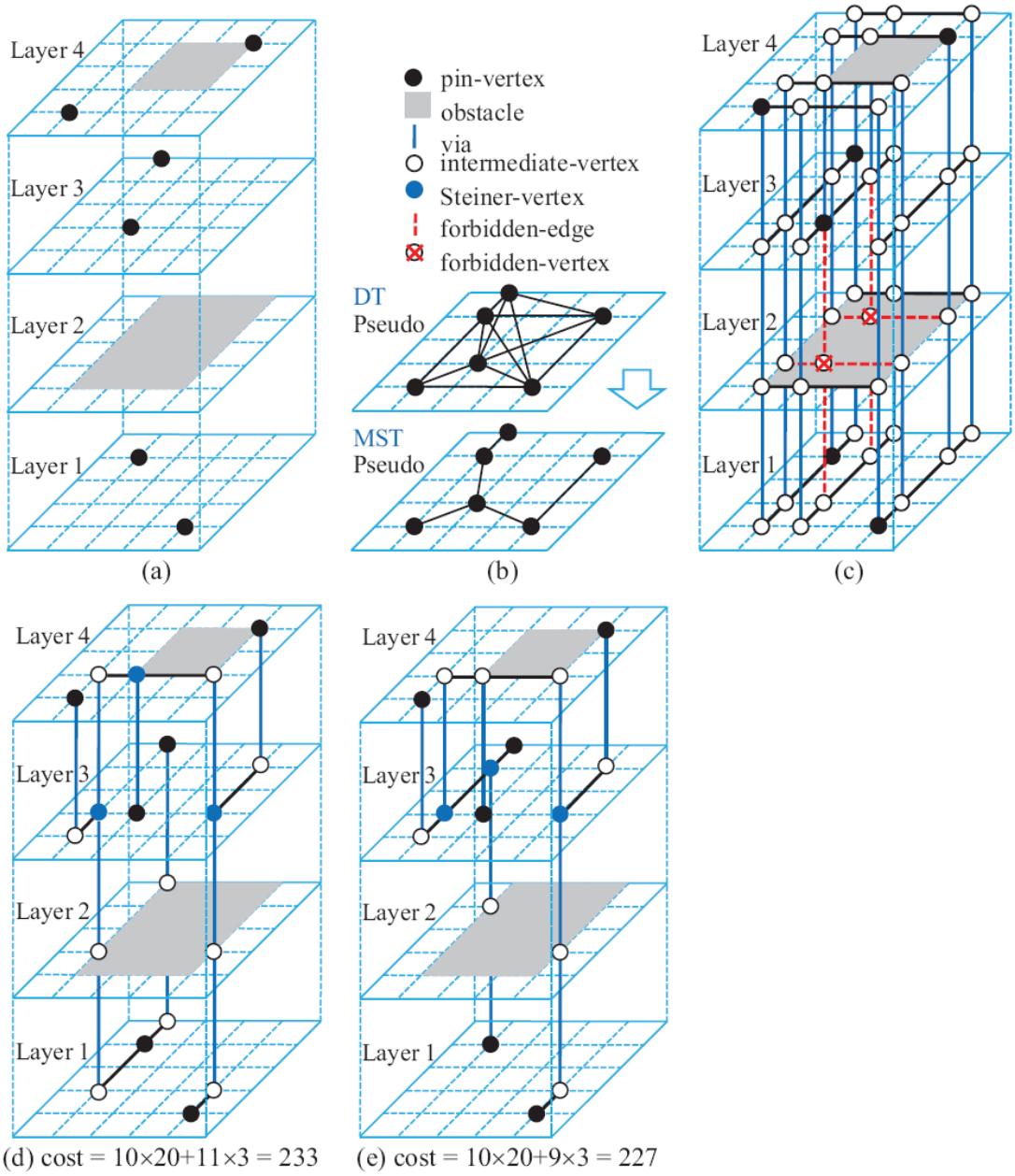


Fig. 10. (a) An instance of OAPDST, where C_v = 3, each grid size is 20×20 (unit)^2, UC_i =1 for all layers. (b)(c) The corresponding DT, obstacle-weighted MST, and 3D extended escape graph. (d) The resulting OAPDST without refinement. (e) The resulting OAPDST with refinement.

Chapter 3

EXTENSIONS

A. Preferred Directions

In this section, we shall demonstrate the flexibility of our algorithm. As shown in Fig. 2, our algorithm for ML-OARSMT can easily be extended to consider preferred directions. We adopt the formulation of the obstacle-avoiding preferred direction Steiner tree (OAPDST) problem in [13].

Problem: Obstacle-Avoiding Preferred Direction Steiner Tree (OAPDST): Given the equivalent wirelength cost C_v of a via, the number N_l of layers, a set $P = \{p_1, p_2, \ldots, p_m\}$ of pins, a set $O = \{o_1, o_2, \ldots, o_k\}$ of obstacles, the layer-specific routing cost UC_i, $1 \leqq i \leqq N_l$, the PD constraints, construct a Steiner tree to connect all pins in P, such that no tree edge or via intersects any obstacle in O and the total cost of the tree is minimized.

The definitions and restrictions of an *obstacle*, a *pin-vertex*, a *via* are the same as those in Section II. Here, a routing layer I has a specific routing cost UC_i, the unit cost of wires in layer i. Without loss of generality, assume the PD constraints as follows: the odd (even) layers only allow vertical (horizontal) edges [12, 13].

To adapt our algorithm for ML-OARSMT to OAPDST, we apply simple and effective modifications to the DT, the 3D extended escape graph, and 3D U-shaped pattern refinement.

1) The DT: For each edge, the part of edge weight contributed by the Manhattan distance is multiplied by UC_i, and α is changed to be a function of obstacles and UC_i. For the edge between p_i and p_j (located at layers z_i and z_j), the UC_i for vertical (horizontal) segments is the minimum value among vertical (horizontal) layers from

layer min(z_i, z_j)-1 to layer max(z_i, z_j)+1.

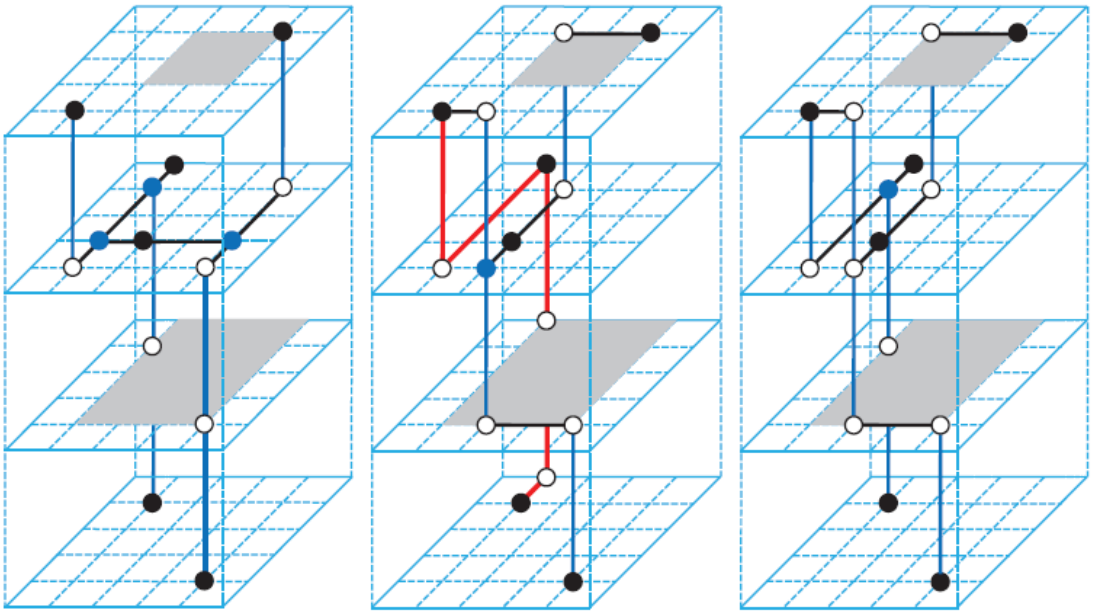
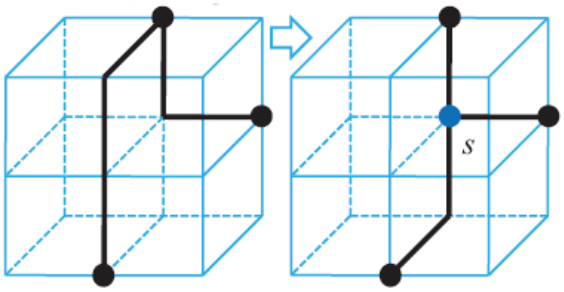
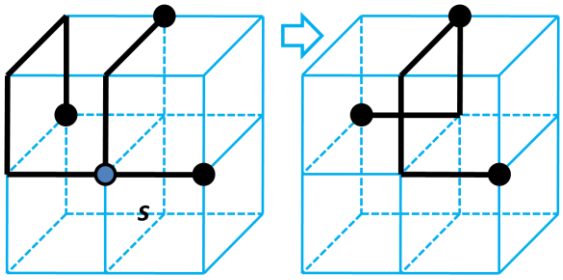


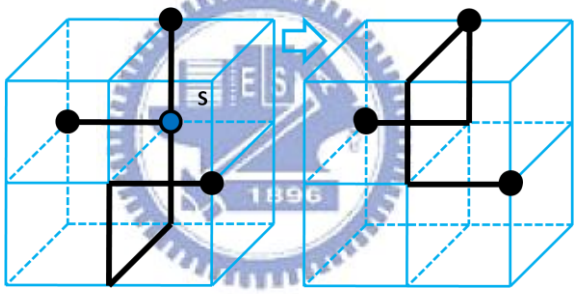
Fig. 11. (a) Our ML-OARSMT. (b) OAPDST in [13]. (c) The refined tree of (b).



(a) degenerated basic case: I + L



(b) degenerated case 2: I + L



(c) degenerated case 3: I + 2L

Fig. 12. Degenerated cases for 3D U-shaped pattern refinement in OAPDST.

2) The 3D extended escape graph: The horizontal (vertical) edges on odd (even) layers are removed. (They are forbidden.) The edge cost on layer i is magnified by UC_i, $1 \leqq i \leqq N_l$.

3) 3D U-shaped pattern refinement: Considering the PD constraints, a Steiner-vertex can only connect vias either with vertical edges or with horizontal edges. For a given U-shaped pattern formed by three vertices, the median of their coordinates may not be valid for a Steiner-vertex. However, the median point still can be a reference point to reroute the L-shaped segments on the pattern, so the strategy is

the same as that in ML-OARSMT.

Fig. 11(a) shows the instance given in [13]; assume C_v = 3, each grid size is 20×20 (unit wirelength)^2, UC_i =1 for all layers. Fig. 11(b)(c) show the corresponding DT, the obstacle-weighted MST, and the 3D extended escape graph. Fig, 10(d)(e) depicts the resulting OAPDST without refinement (cost = 233 (=10×20+11×3)), with refinement (cost = 227 (=10×20+9×3)), respectively. Fig. 12(a) shows the corresponding ML-OARSMT generated by our algorithm, cost = 218 (=10×20+6×3); it can be viewed as the lower bound of the cost of OAPDST. Fig. 11(b) shows the OAPDST given in [13], cost = 281 (=13×20+7×3), where a standard pattern is highlighted by bold lines. After refining this pattern, we can obtain a better tree in Fig. 11(c), cost = 261 (=12×20+7×3). Fig. 12 lists degenerated cases for refinement in OAPDST.

B. Global Routing

To include our Steiner tree construction to global routing, we shall consider the capacity of each edge on the global routing graph. Without loss of generality, assume the net ordering is given. It can be seen that on the 3D escape graph, if the capacity of some edge is full, then this edge can be set as forbidden; otherwise, this edge can still be used. After the RSMT is constructed, the capacity of the corresponding routed edges reduces. Moreover, considering the grids on upper metal layers are larger than lower ones, we may slightly shift the lines of the3D escape graph to align their nearest grids.

Chapter 4

EXPERIMENTAL RESULTS

We implemented our algorithm in C++ language and executed the program on a PC with an Intel Pentium4 3.0 GHz CPU and 1 GB memory under Windows XP OS. Our results show our algorithm outperforms state-of-the-art works on SL-OARSMT, ML-OARSMT, and OAPDST. Meanwhile, our runtimes are also stable, not increasing much from SL-OARSMT to ML-OARSMT and OAPDST. In addition, the comparison between DT without and with obstacles is provided. Furthermore, the results of timing-driven RSMT are also provided.

TABLE II

SL-OARSMT: THE COMPARISONS ON THE TOTAL WIRELENGTH BETWEEN PREVIOUS WORK [6-10] AND OURS

Test cases	m / k	HPBB[1]	Total wirelength							Imp.(%) (X-G)/(X-Y)						Time (s)			
			[6]	[7]	[8]	[9]	[10]	Ours_SL[2]		[6]	[7]	[8]	[9]	[10]	Ours_SL	[9]	Ours_SL		
		(Y)	(A)	(B)	(C)	(D)	(E)	Nref[3] (F)	Full[4] (G)	(A)	(B)	(C)	(D)	(E)	Nref (F)	(D)	Mst[5] (H)	Nref (F)	Full (G)
sl-ind1	10 / 32	501	-	644	626	632	649	622	609	-	24.48	13.60	17.56	27.03	10.74	<0.01	<0.01	<0.01	0.02
sl-indx	74 / 625	493	-	1,731	1,640	1,674	1,897	1,640	1,616	-	9.29	2.09	4.91	20.01	2.09	0.28	0.03	0.08	0.09
sl-indy	115/1204	630	-	3,011	2,872	2,969	3,009	2,844	2,804	-	8.69	3.03	7.05	8.62	1.81	0.76	0.08	0.25	0.25
sl-rc1	10 / 10	17,890	30,410	29,320	27,250	27,790	28,380	28,080	25,980	35.38	29.22	13.57	18.28	22.88	20.61	<0.01	0.01	0.02	0.03
sl-rc2	30 / 10	19,470	45,640	43,400	43,220	42,240	42,030	42,710	41,945	14.12	6.08	5.37	1.30	0.38	3.29	<0.01	<0.01	0.05	0.05
sl-rc3	50 / 10	19,380	58,570	57,020	56,500	56,140	56,270	57,370	54,690	9.90	6.19	4.88	3.94	4.28	7.05	<0.01	<0.01	0.05	0.09
sl-rc4	70 / 10	19,850	63,340	61,910	61,090	60,800	59,730	63,280	61,190	7.24	4.09	2.18	1.49	-1.15	7.11	<0.01	<0.01	0.07	0.13
sl-rc5	100 / 10	19,600	83,150	78,240	76,870	76,760	76,330	77,340	75,300	12.35	5.01	2.74	2.55	1.82	3.53	0.01	0.02	0.13	0.20
sl-rc6	100 / 500	19,593	149,750	86,770	84,327	84,197	87,588	83,080	80,947	52.86	8.67	5.22	5.03	9.77	3.36	0.16	0.05	9.03	22.34
sl-rc7	200 / 500	19,882	181,470	118,169	115,461	114,173	118,229	113,657	112,241	42.84	6.03	3.37	2.05	6.09	1.51	0.31	0.09	7.14	15.64
sl-rc8	200 / 800	19,803	202,741	123,360	122,574	120,492	124,896	118,054	116,378	47.21	6.74	6.03	4.09	8.11	1.71	0.49	0.11	15.30	34.78
sl-rc9	200/1000	19,964	214,850	120,567	120,017	117,659	120,629	116,885	114,988	51.24	5.55	5.03	2.73	5.60	1.96	0.63	0.14	19.13	44.58
sl-rc10	500 / 100	19,900	198,010	174,420	172,490	171,520	169,129	172,940	168,710	16.45	3.70	2.48	1.85	0.28	2.76	0.32	0.17	3.28	15.33
sl-rc11	1000/100	19,984	250,570	242,840	238,377	237,794	235,704	243,518	236,436	6.07	2.81	0.83	0.62	-0.34	3.11	1.36	0.38	9.98	43.70
Avg. (%)[6] (X-G)/X	- / -	-	-	-	-	-	-	-	-	21.57	5.16	2.85	2.77	4.57	2.76	-	-	-	-
Avg. (%)[7] (X-G)/(X-Y)	- / -	-	-	-	-	-	-	-	-	26.88	9.04	5.03	5.25	8.10	5.05	-	-	-	-

[1]HPBB: The half-perimeter of the bounding box of all pin-vertices (which is a lower bound of total wirelength), and "-" refers to "not available."

[2]Ours_SL: Our algorithm for SL-OARSMT.

[3]Nref: All steps of our algorithm are applied, but 3D U-shaped pattern refinement is turned off.

[4]Full: All steps of our algorithm are applied.

[5]Mst: Step 1 (DT) and step 2 (MST) of our algorithm are applied.

[6,7]Avg. (%): Average improvement is computed by averaging (X-G)/X, and (X-G)/(X-Y) for all cases, X = A, B, C, D, E, F.

A. SL-OARSMT

For SL-OARSMT, totally 14 benchmark circuits were provided by [8]; the first 3 from industry, the rest from [6]. We compared our algorithm with those presented in [6], [7], [8], [9], [10]. The results of [6] and [8] are quoted from their papers; those of [7] are quoted from [8]; those of [9] and [10] were conducted on our platform using their binary codes. (In addition, the parameter α was set to [0.50, 1.30] depending on the congestion.) As listed in TABLE II, considering the differences from the half-perimeter of the bounding box of all pins (which is a lower bound of the optimal solution), our algorithm achieved average 5.03% up to 26.88% improvement on wirelength over them. Moreover, we had the best results for 12 out of 14 cases. Fig. 13 shows the resulting SL-OARSMTs of sl-rc6 and sl-rc9. Without refinement, on average, we still have a small win to [8] and [9] on total wirelength. Novel 3D U-shaped pattern refinement worked well in planar cases and contributed 2.76% reduction on wirelength. Because our method mainly focuses on multi-layer, the overhead on runtimes for single-layer is reasonable.

B. ML-OARSMT

For ML-OARSMT, totally 10 test cases were provided by [11]. ml-ind4 and ml-ind5 simulate the environment for single-layer routing, where all pins and obstacles are located in a layer, and the upper and lower adjacent layers are entirely occupied by another two large obstacles. Fig. 14 displays the ML-OARSMT of ml-ind2 generated

by our algorithm as $C_v = 3$.

We compared our algorithm with [11]. (In addition, the parameter _ was set to [0.70, 1.15].) As listed in TABLE III (IV), as $C_v = 3$ (5), the average improvements on the number of vias and total costs are 7.69% (4.76%), 2.77% (2.74%), respectively. Our algorithm has smaller total costs in 9 out of 10 cases. In addition, our algorithm always generated a smaller total cost as $C_v = 3$ than that as $C_v = 5$ for each case; it can be seen that our algorithm is indeed stable.

TABLE III

ML-OARSMT: THE COMPARISONS ON THE NUMBER OF VIAS, THE TOTAL COSTS, AND CPU TIMES BETWEEN [11] AND OURS UNDER $C_V = 3$

Test cases	m / k / N_t	Total cost (#via)				Time (s)				
		[11]	[11]_ref[2]	Ours_ML[3]		[11][1]	[11]_ref	Ours_ML		
		(A)	(B)	Nref (C)	Full (D)	(A)	(B)	Nref (C)	Full (D)	Mst (E)
ml-ind1	50 / 6 / 5	55,537 (49)	55,537 (49)	55,323 (51)	53,915 (45)	0.07	0.03	0.03	0.05	<0.01
ml-ind2	200 / 85 / 6	12,512 (224)	12,420 (224)	12,491 (232)	12,179 (210)	3.05	0.52	0.39	0.97	0.03
ml-ind3	250 / 13 / 10	10,973 (359)	10,695 (356)	10,996 (357)	10,677 (325)	3.32	2.09	2.19	5.56	0.06
ml-ind4	500 / 100 / 5	77,033 (0)	76,969 (0)	77,594 (0)	77,275 (0)	8.12	2.33	0.88	2.34	0.20
ml-ind5	1000 / 20 / 5	14,515,511 (0)	14,455,316 (0)	14,981,554 (0)	14,496,361 (0)	45.63	18.16	12.06	82.88	0.48
ml-rt1	25 / 10 / 10	4,334 (76)	4,244 (69)	4,247 (68)	4,042 (67)	0.06	0.11	0.11	0.09	<0.01
ml-rt2	100 / 20 / 10	9,434 (215)	9,222 (220)	9,485 (192)	9,234 (188)	0.88	1.22	0.97	1.78	0.02
ml-rt3	250 / 50 / 10	15,569 (490)	15,223 (502)	15,408 (465)	14,996 (456)	6.86	6.97	4.84	12.35	0.05
ml-rt4	500 / 50 / 10	22,034 (918)	21,548 (951)	21,782 (925)	21,151 (915)	17.52	26.63	10.78	50.25	0.16
ml-rt5	1000/100/5	27,890 (869)	27,252 (840)	27,632 (857)	27,028 (817)	55.61	32.63	11.00	75.33	0.53
Imp.(%)[3]	-	-	1.48 (0.5)	0.10 (2.48)	2.77 (7.69)	-	-	-	-	-

[1]The runtimes of [11] are quoted from the paper, generated by a 2.8GHz AMD-64 machine with 8GB memory under Ubuntu 6.06 OS. They are listed for reference because the machine is different.

[2][11]_ref: 3D U-shaped pattern refinement is directly applied to the resulting ML-OARSMT of [11]. The runtimes of [11]_ref only count for refinement and are measured on our platform.

[3]Ours_ML: Our algorithm for ML-OARSMT. 3Imp. (%): Average improvement is computed by averaging (A-X)/A for all cases, X =B, C or D.

TABLE IV

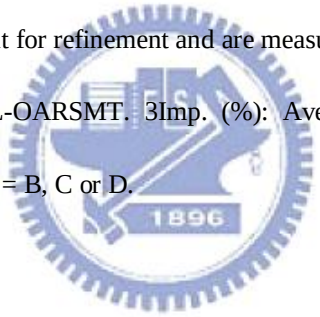
ML-OARSMT: THE COMPARISONS ON THE NUMBER OF VIAS, THE TOTAL COSTS, AND CPU TIMES BETWEEN [11] AND OURS UNDER $C_V = 5$

Test cases	m / k / N_t	Total cost (#via)				Time (s)				
		[11]	[11]_ref[2]	Ours_ML[3]		[11][1]	[11]_ref	Ours_ML		
		(A)	(B)	Nref (C)	Full (D)	(A)	(B)	Nref (C)	Full (D)	Mst (E)
ml-ind1	50 / 6 / 5	55,635 (49)	55,635 (49)	55,425 (51)	54,005 (45)	0.07	0.05	0.05	0.05	0.02
ml-ind2	200 / 85 / 6	12,899 (208)	12,862 (220)	12,927 (219)	12,599 (210)	3.01	0.42	0.38	0.94	0.05
ml-ind3	250 / 13 / 10	11,698 (343)	11,559 (334)	11,794 (328)	11,327 (325)	3.35	2.03	2.36	5.84	0.05
ml-ind4	500 / 100 / 5	77,033 (0)	76,969 (0)	77,594 (0)	77,275 (0)	8.21	2.34	0.91	2.53	0.19
ml-ind5	1000 / 20 / 5	14,515,511 (0)	14,455,316 (0)	14,981,554 (0)	14,496,361 (0)	45.71	16.50	14.28	83.09	0.48
ml-rt1	25 / 10 / 10	4,486 (76)	4,377 (68)	4,392 (78)	4,176 (67)	0.06	0.09	0.09	0.17	<0.01
ml-rt2	100 / 20 / 10	9,812 (200)	9,653 (210)	9,784 (189)	9,616 (191)	0.90	1.16	0.92	1.78	0.02
ml-rt3	250 / 50 / 10	16,384 (429)	15,861 (432)	16,240 (447)	15,858 (431)	6.84	6.95	4.38	13.16	0.05
ml-rt4	500 / 50 / 10	23,883 (879)	23,325 (893)	23,667 (883)	22,981 (848)	17.56	25.89	9.83	50.66	0.16
ml-rt5	1000/100/5	29,598 (814)	28,972 (801)	29,470 (802)	28,626 (763)	56.45	33.94	10.83	76.33	0.50
Imp.(%)[3]	-	-	1.37 (0.17)	0.00 (-0.66)	2.74 (4.76)	-	-	-	-	-

[1]The runtimes of [11] are quoted from the paper, generated by a 2.8GHz AMD-64 machine with 8GB memory under Ubuntu 6.06 OS. They are listed for reference because the machine is different.

[2][11]_ref: 3D U-shaped pattern refinement is directly applied to the resulting ML-OARSMT of [11]. The runtimes of [11]_ref only count for refinement and are measured on our platform.

[3]Ours_ML: Our algorithm for ML-OARSMT. 3Imp. (%): Average improvement is computed by averaging (A-X)/A for all cases, X = B, C or D.



C. OAPDST

For OAPDST, totally 10 test cases are used. 7 cases are exactly the same as those used in ML-OARSMT. We did not use ml-ind3 because it is invalid under the PD constraints. pd-ind4, pd-ind5a, pd-ind5b were modified from ml-ind4 and ml-ind5. For pd-ind4 and pd-ind5a, we inserted one empty layer right above the working layer. For pd-ind5b, we further duplicated the obstacles in the working layer onto the inserted layer. In addition, the routing cost UC_i was set to 1 for all i. By doing so, we can see how worse an OAPDST can be with respect to its ML-OARSMT counterpart. (In addition, the parameter α was set to [0.70, 1.00].) Fig. 15 displays the OAPDST of ml-ind2 generated by our algorithm as $C_v = 3$.

We compared our algorithm with [13]; because we cannot obtain the test cases

and the program of [13], we implemented their algorithm and executed it on the same machine described above.

As listed in TABLE V, as C_v = 3, the average degradation of the total costs from ML-OARSMT to OAPDST is 6.47%, but the average speedup of CPU times is 60.18%. The average improvement of the total costs over [13] is 3.20%, while the CPU times are almost the same. Our algorithm has smaller total costs in 9 out of 10 cases.

TABLE VI compares the impacts of the obstacle-weighted MST and 3D U-shaped pattern refinement of our algorithm. It can be seen that without the guidance from the MST, on average, we may have an 8.76% degradation on the total costs, and the CPU times surprisingly become much worse (38.88% slower). Hence, steps 1 and 2 are necessary; actually, they are efficient and effective. On the other hand, although 3D U-shaped pattern refinement does not influence much on our results, it does improve the total costs of [13] by 2.66% on average. The refined results of [13] are still slightly worse than ours when refinement is turned off. Although not presented here, we have similar results for C_v = 5, and $UC_i \neq 1$.

TABLE V

OAPDST: THE COMPARISONS ON THE NUMBER OF VIAS, THE TOTAL COST, AND CPU TIMES BETWEEN [13] AND OURS UNDER $C_V = 3$, $UC_I = 1$, $1 \leqq I \leqq N_L$

Test cases	m / k / N_t	Total cost (#via)			Time (s)		
		Ours_ML[1] (A)	Ours_PD[2] (B)	[13] (C)	Ours_ML (A)	Ours_PD (B)	[13] (C)
ml-ind1	50 / 6 / 5	53,915 (45)	64,401 (77)	66,033 (81)	0.05	0.03	0.02
ml-ind2	200 / 85 / 6	12,179 (210)	13,260 (364)	13,575 (363)	0.97	0.49	0.39
pd-ind4	500 / 100 / 6	60,298 (137)	62,459 (661)	61,624 (651)	5.63	1.58	1.28
pd-ind5a	1000 / 20 / 6	14,381,940 (16)	14,717,269 (1,623)	15,503,281 (1,592)	111.37	15.97	18.46
pd-ind5b	1000 / 40 / 6	14,496,361 (0)	14,768,268 (1,624)	15,873,298 (1,592)	82.88	16.11	18.91
ml-rt1	25 / 10 / 10	4,042 (67)	4,117 (82)	4,289 (83)	0.09	0.08	0.09
ml-rt2	100 / 20 / 10	9,234 (188)	9,528 (268)	9,803 (257)	1.78	0.94	0.97
ml-rt3	250 / 50 / 10	14,996 (456)	15,776 (619)	16,295 (619)	12.35	5.09	5.36
ml-rt4	500 / 50 / 10	21,151 (915)	22,366 (1,217)	23,222 (1,144)	50.25	13.87	12.11
ml-rt5	1000 / 100 / 5	27,028 (817)	30,431 (1,462)	31,328 (1,457)	75.33	11.58	12.16
Imp. (%)[3]	-	-6.47	-	3.20	60.18	-	-0.50

[1]Ours_ML: Our algorithm is applied without the PD constraints; it can be viewed as the lower bound of the total cost for OAPDST.

[2]Ours_PD: All steps of our algorithm for OAPDST are applied. 3Imp. (%): Average improvement is computed by averaging (X-B)/X for all cases, where X = A, C.

TABLE VI

OAPDST: THE COMPARISONS ON THE IMPACTS OF OUR ALGORITHM ON THE TOTAL COST AND CPU TIMES UNDER $C_V = 3$, $UC_I = 1$, $1 \leqq I \leqq N_L$

Test cases	m / k / N_t	Total cost			Time (s)		
		Nmst[1] (D)	Nref[2] (E)	[13]_ref[3] (F)	Nmst (D)	Nref (E)	[13]_ref (F)
ml-ind1	50 / 6 / 5	69,913	64,401	66,166	0.05	0.03	0.05
ml-ind2	200 / 85 / 6	14,448	13,275	13,227	0.75	0.41	1.17
pd-ind4	500 / 100 / 6	67,348	63,120	58,820	1.94	1.48	12.41
pd-ind5a	1000 / 20 / 6	16,663,711	14,721,978	14,976,592	32.00	21.36	128.56
pd-ind5b	1000 / 40 / 6	16,667,652	14,773,077	15,274,934	31.28	17.63	101.88
ml-rt1	25 / 10 / 10	4,466	4,146	4,146	0.13	0.09	0.11
ml-rt2	100 / 20 / 10	10,502	9,594	9,599	1.72	0.97	1.02
ml-rt3	250 / 50 / 10	17,030	15,825	16,011	9.42	4.72	6.39
ml-rt4	500 / 50 / 10	24,149	22,458	22,495	24.47	11.16	21.89
ml-rt5	1000 / 100 / 5	33,546	30,467	30,622	16.69	11.70	95.61
Imp. (%)[4]	-	8.76	0.34	0.54	38.88	-0.41	53.22

[1]Nmst: Only step 3 of our algorithm is applied, i.e., the tree is directly constructed from the 3D extended escape graph.

[2]Nref: All steps of our algorithm are applied, but 3D U-shaped pattern refinement is turned off.

[3][13]_ref: 3D U-shaped pattern refinement is applied to [13].

[4]Imp. (%): Average improvement is computed by averaging (X-B)/X for all cases, where B is Ours_PD

in TABLE IV, X = D, E, F.

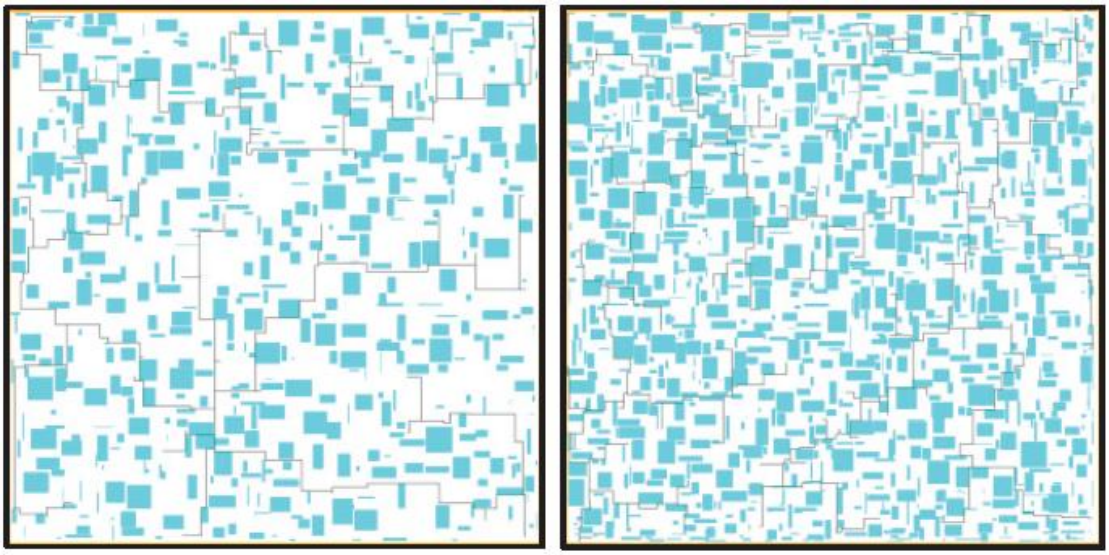


Fig. 13. (a) The SL-OARSMT of sl-rc6. (b) The SL-OARSMT of sl-rc9.

Fig. 14. The ML-OARSMT of ml-ind2 under C_v = 3. (a) The DT without illegal edges. (b) The MST. (c)-(g) Layers 2-6, respectively. (h) All pin-vertices are projected onto a pseudo plane, without showing the obstacles.

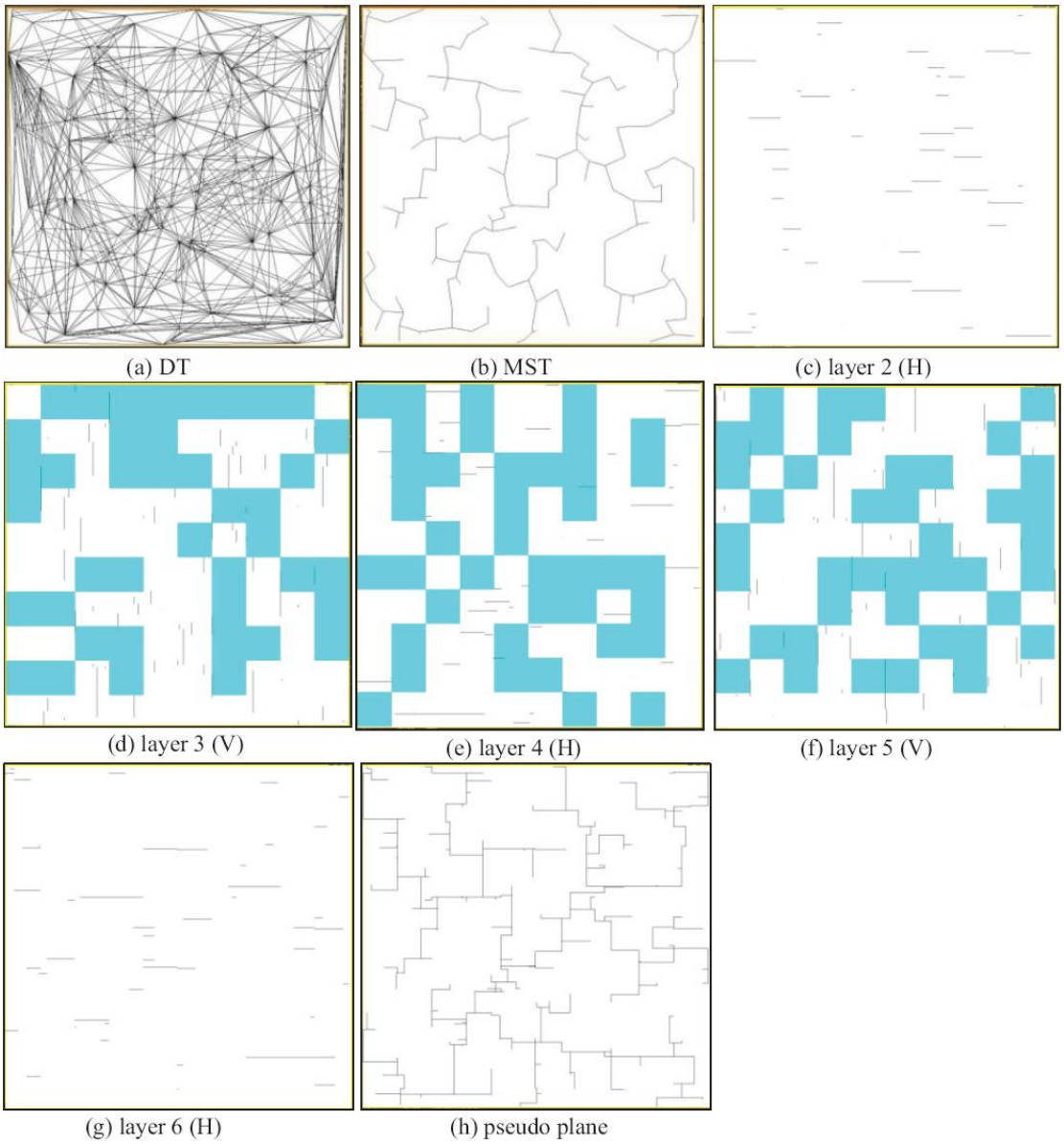


Fig. 15. The OAPDST of ml-ind2 under C_v = 3. (a) The DT with illegal edges. (b) The MST. (c)-(g) Layers 2-6, respectively. The odd (even) layers allow vertical (horizontal) edges. Some line segments are at obstacle boundaries; they are feasible according to the problem formulation. (h) All pin-vertices are projected onto a pseudo plane, without showing the obstacles.

Chapter 5

CONCLUSION

In this thesis, we solved ML-OARSMT and OAPDST by the same strategy. In addition, we also showed our method can be extended to construct timing-driven RSMTs. Previous work tackles one configuration at a time, while our algorithm can easily handle various configurations. Experimental results showed that our algorithm outperformed the state-of-the-art works. Future work includes the extensions to clock trees and manufacturability-aware trees.

REFERENCES

[1] The International Technology Roadmap for Semiconductors (ITRS), 2007. Available: http://www.itrs.net/

[2] 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, 1977.

[3] J. L. Ganley and J. P. Cohoon, "Routing a multi-terminal critical net: Steiner tree construction in the presence of obstacles," in *Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS'94)*, vol. 1, May 1994, pp.113-116.

[4] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed., Springer-Verlag, 2008.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed., MIT Press, 2001.

[6] Z. Feng, Y. Hu, T. Jing, X. Hong, X. Hu, and G. Yan, "An O(nlogn) algorithm for obstacle-avoiding routing tree construction in the lambda geometry plane," in *Proc. ACM Int. Symp. on Physical Design (ISPD'06)*, Apr. 2006, pp. 48-55.

[7] Z. Shen, C. C. N. Chu, and Y.-M. Li, "Efficient rectilinear Steiner tree construction with rectilinear blockages," in *Proc. IEEE Int. Conf. on Computer Design (ICCD'05)*, Oct. 2005, pp. 38-44.

[8] 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. ACM/IEEE Asia and South Pacific Design Automation Conf. (ASP-DAC'07)*, Jan. 2007, pp. 262-267.

[9] 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. Computer-Aided Design*, vol. 27, no. 4, pp.643-653, Apr. 2008. Also see *Proc. ACM Int. Symp. on Physical Design (ISPD'07)*, pp.127-134.

[10] J. Long, H. Zhou, and S. O. Memik, "An O(nlogn) edge-based algorithm for obstacle-avoiding rectilinear Steiner tree construction," in *Proc. ACM Int. Symp. on Physical Design (ISPD'08)*, Apr. 2008, pp. 126-133

[11] C.-W. Lin, S.-L. Huang, K.-C. Hsu, M.-X. Lee, and Y.-W. Chang, "Multilayer obstacle-avoiding rectilinear Steiner tree construction based on spanning graphs," *IEEE Trans. Computer-Aided Design*, vol. 27, no.11, pp. 2007-2016, Nov. 2008. Also see *Proc. IEEE/ACM Int. Conf. on Computer-aided Design (ICCAD'07)*, pp.380-385.

[12] M. C. Yildiz and P. H. Madden, "Preferred direction Steiner trees," *IEEE Trans. Computer-Aided Design*, vol. 21, no. 11, pp. 1368-1372, Nov.2002.

[13] 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. ACM Int. Symp. on Physical Design (ISPD'08)*, Apr. 2008, pp.118-125.

[14] I. H.-R. Jiang, S.-W. Lin, and Y.-T. Yu, "Unification of obstacle-avoiding rectilinear Steiner tree construction," in *Proc. IEEE Int. SOC Conf. (SOCC'08)*, Sep. 2008.

[15] I. H.-R. Jiang and Y.-T. Yu, "Configurable rectilinear Steiner tree construction for SoC and nano technologies," in *Proc. IEEE Int. Conf. on Computer Design (ICCD'08)*, Oct. 2008, pp. 34-39.

APPENDIX

Timing-Driven Steiner Trees

In addition to preferred directions, our method can also be applied to other types of Steiner trees. For example, a timing-driven RSMT targets to minimize the path length between each pin to the designated source pin. It can be done by constructing a shortest-path tree instead of MST at step 2. The shortest-path tree is constructed by Dijkstra's shortest path algorithm. Moreover, if step 3 still remains the same, we may obtain a compromise between the path length and the total cost.

For timing-driven RSMT, the source pin is randomly assigned in our experiments. TABLE VII lists the results for timing-driven RSMT. Fig. 16 displays the timing-driven OAPDST of ml-ind2 generated by our algorithm as C_v = 5.

TABLE VII

TIMING-DRIVEN RSMT

SL-OARSMT Total wirelength			ML-OARSMT Total cost (#via) as $C_v = 3$			OAPDST Total cost (#via as $C_v = 3$		
Test cases	Original[1]	Timing-Driven[2]	Test cases	Original	Timing-Driven	Test cases	Original	Timing-Driven
sl-rc1	25,980	25,520	ml-ind1	54,005 (45)	57,588 (43)	ml-ind1	64,401 (77)	67,906 (86)
sl-rc2	41,945	43,850	ml-ind2	12,599 (210)	13,714 (206)	ml-ind2	13,260 (364)	15,864 (358)
sl-rc3	54,690	58,190	ml-ind3	11,327 (325)	12,570 (379)	pd-ind4	62,459 (661)	90,118 (735)
sl-rc4	61,190	66,330	ml-ind4	77,275 (0)	85,669 (0)	pd-ind5a	14,717,269 (1,623)	16,341,548 (1,603)
sl-rc5	75,300	80,160	ml-ind5	14,496,361 (0)	16,323,737 (0)	pd-ind5b	14,768,268 (1,624)	16,524,615 (1,617)
sl-rc6	80,947	85,664	ml-rt1	4,176 (67)	4,660 (66)	ml-rt1	4,117 (82)	4,911 (87)
sl-rc7	112,241	120,202	ml-rt2	9,616 (191)	10,711 (215)	ml-rt2	9,528 (268)	11,102 (285)
sl-rc8	116,378	124,111	ml-rt3	15,858 (431)	17,330 (464)	ml-rt3	15,776 (619)	18,594 (652)
sl-rc9	114,988	127,085	ml-rt4	22,981 (848)	25,469 (961)	ml-rt4	22,366 (1,217)	28,189 (1,339)
sl-rc10	168,710	183,850	ml-rt5	28,626 (763)	32,185 (843)	ml-rt5	30,431 (1,462)	37,237 (1,473)

[1]Original: The results of our algorithms to minimize total cost are quoted from TABLE II, III, and V.

[2]Timing-Driven: To minimize the delay from source to each sink, the minimum spanning tree is replaced with shortest path tree at step 2. The rest of our algorithm is unchanged.

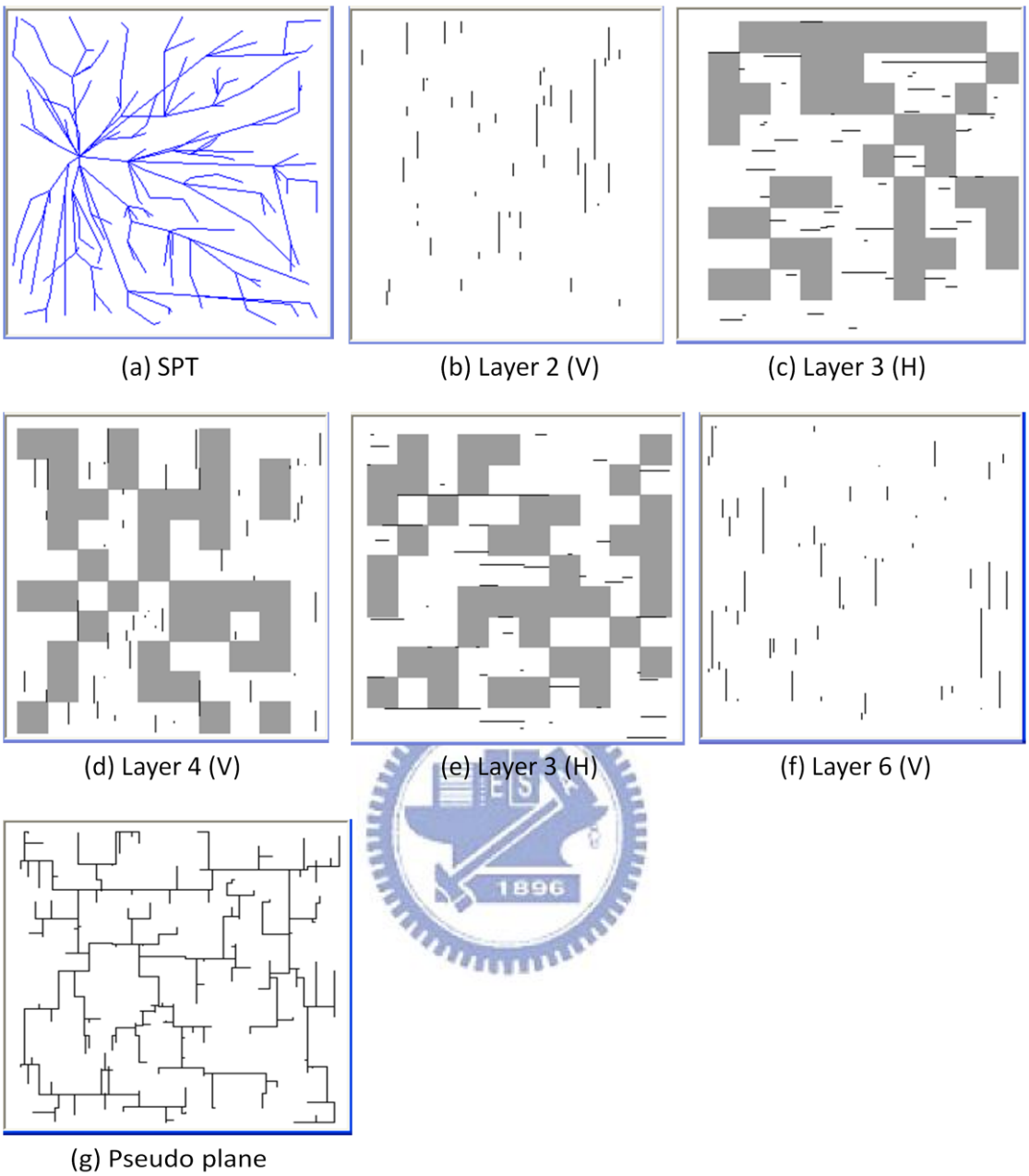


Fig. 16. The timing driven OAPDST of ml-ind2 under C_v = 5. DT is the same as Fig. 15 (a). (a) The SPT. (b)-(f) Layers 2-6, respectively. The even (odd) layers allow vertical (horizontal) edges. Some line segments are at obstacle boundaries; they are feasible according to the problem formulation. (g) All pin-vertices are projected onto a pseudo plane, without showing the obstacles.