Configurable Rectilinear Steiner Tree Construction for SoC and Nano Technologies

Iris Hui-Ru Jiang and Yen-Ting Yu
Department of Electronics Engineering & Institute of Electronics
National Chiao Tung University, Hsinchu 30010, Taiwan
Email: hrjiang@faculty.nctu.edu.tw, keekeewawa.ee96g@nctu.edu.tw

***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 nano technologies. This paper proposes a construction-by-correction approach for obstacle-avoiding preferred direction rectilinear Steiner tree construction. Experimental results show that our algorithm is promising and outperforms the state-of-the-art works.**

I. Introduction

Rectilinear Steiner minimal tree (RSMT) construction has been extensively studied and considered as a fundamental problem in physical design; it is frequently performed for interconnect estimation during floorplanning, placement, routing stages. To make the estimation practical, we shall consider the fabrication issues for modern SoC and nano technologies. Advanced nano technology offers an abundance of routing layers, e.g., 11 in 65 nm [1], and normally assigns a preferred routing direction to each layer; large-scale SoC designs often contain a tremendous number of obstacles.

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 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 (CGG): Step 1 generates a connection graph to connect all pins. Obstacle boundaries may also be included. This graph contains geometrical proximity information among pins, and even obstacle boundaries. The initial connection graph can be a complete graph, a spanning graph, an escape graph [3], or a Delaunay triangulation (DT) [4].

2) Minimum spanning tree construction (MST): Step 2 constructs a minimum spanning tree (MST) [5] over all pins based on the connection graph. The MST may be obstacle-avoiding (all tree edges bypass obstacles), obstacle-weighted (tree edges may run through obstacles but consider the impacts of obstacles into edge weights), or mixed (tree edges are obstacle-weighted first and then obstacle-avoiding).

3) Rectilinearization and refinement (R&R): Step 3 transforms the MST into a rectilinear Steiner tree and refines the total cost. The total cost includes wirelength and vias. Planar U-shaped pattern refinement is usually applied.

As listed in Table 1, we compare the available 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 "essential edges" into their spanning graph. (The essential edges can 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). Even so, it is still somewhat impractical because it cannot directly be extended to

Table 1. The comparison between recent works on RSMT.

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

[1]ML: Multi-layer; SL: Single-layer.
[2]OA: Obstacle-avoidance.
[3]PD: Preferred direction.

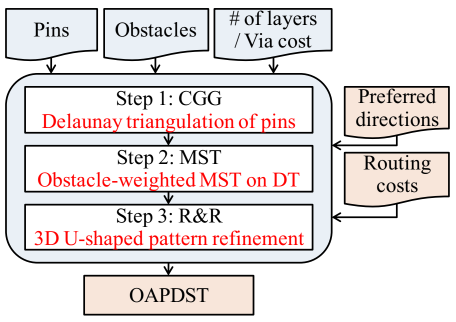

Fig. 1. The overview of our algorithm; it can be configured to handle different constraints.

consider preferred directions. [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). [13] directly constructed a rectilinear MST over a 3D improved escape graph. However, the MST was not furthered refined, so the solution quality may be limited.

As shown in Fig. 1, in this paper, we extend a previously presented heuristic [14] to consider preferred directions. Steps 1 and 2 construct an obstacle-weighted MST on the DT of pins only. Step 3 rectilinearizes each tree edge on a 3D extended escape graph and then refines it. Our features include:

1) We can easily handle 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 extra edges that may be essential during DT construction and adequately associating the impacts of obstacles into edge weights.

3) We construct the obstacle-weighted MST 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 present novel 3D U-shaped pattern refinement.

Experimental results show that our algorithm achieves smaller total costs in 9 out of 10 cases and on average outperforms the state-of-the-art work. 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.

II. Problem Formulation

We adopt the definitions and restrictions of an *obstacle*, a *pin-vertex*, a *via* in [11]. An *obstacle* is a rectangle on a layer, indicated by its four *corner-vertices*. A *pin-vertex* p_i 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.

We adopt the formulation of the obstacle-avoiding preferred direction Steiner tree problem in [13]. 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].

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, $1 \le i \le N_l$, 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.

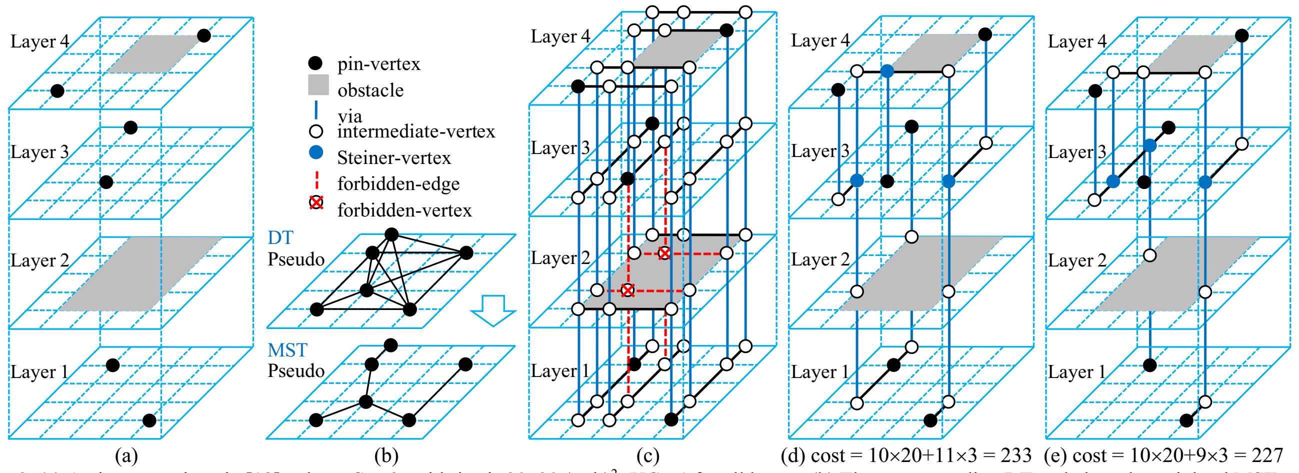

Fig. 2. (a) An instance given in [13], where $C_v = 3$, grid size is 20x20 (unit)^2, $UC_i = 1$ for all layers. (b) The corresponding DT and obstacle-weighted MST. (c) The corresponding 3D extended escape graph. (d) The resulting OAPDST without refinement. (e) The resulting OAPDST with refinement.

III. Algorithm

Our algorithm is based on construction-by-correction. Here, we use the example given in [13], depicted in Fig. 2(a), to demonstrate our algorithm. Assume $C_v = 3$, $UC_i = 1$ for all layers, and the 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 the process, some extra edges that may be essential are added. (see Fig. 2(b))

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. 2(b))

3) Step 3: Each tree edge is rectilinearized on a 3D extended escape graph (see Fig. 2(c)), and then novel 3D U-shaped pattern refinement is applied. (see Fig. 2(e))

Our OAPDST for this instance (see Fig. 2(e)) is of cost 227 (=10×20+9×3), while the OAPDST generated by [13] (see Fig. 6(b)) is of cost 281 (=13×20+7×3). It can be improved by our refinement method; as shown in Fig. 6(c), the refined tree is of cost 261 (=12×20+7×3).

As shown in Fig. 1, our algorithm can easily handle variant configurations. Without the PD constraints, this problem becomes ML-OARSMT; moreover, if $N_l = 1$ and without the PD constraints, it becomes SL-OARSMT. We detail each step and analyze the time complexity of our algorithm as follows.

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, using this pseudo plane, we can extract the geometrical proximity among pins.

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.

During DT construction, if two triangles violate the definition of DT, the common edge of them, an illegal edge, as depicted in Fig. 3(a), is then flipped to a legal edge, as depicted in Fig. 3(b). The typical process discards these illegal edges; instead, we preserve them since they contain more global information than legal ones and may lead to better solutions. Fig. 2(b) gives the corresponding DT of the instance in Fig. 2(a), where illegal/legal edges are both shown.

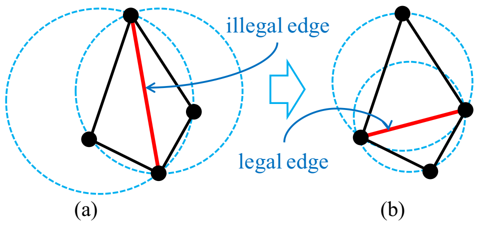

Fig. 3. During DT construction, an illegal edge in (a) is flipped into a legal one in (b).

B. Obstacle-Weighted MST on DT

As shown in Fig. 2(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].

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 is simplified from [8], where only the obstacles completely passing through the bounding box between p_i, p_j horizontally or vertically are counted. In addition, we introduce a parameter α to further reflect the congestion of obstacles and the average routing costs among layers *i* to *j*. The edge weight $w(p_i, p_j)$ is computed as follows.

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

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.

C. Rectilinearization and 3D U-Shaped Pattern Refinement

Based on the guidance of the obstacle-weighted MST, pins are connected by rectilinear segments and then the total cost is further reduced by 3D U-shaped pattern refinement. These two operations are compounded into one and iteratively applied to the MST edge. By doing so, the refinement done for early edges can benefit consequent edges, thus our refinement does not always hurt runtimes.

Rectilinearization is performed on a 3D extended escape graph based on Dijkstra's shortest path algorithm [5]. We extend the planer 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. To make the implementation flexible, we adequately associate forbidden flags to the vertices that are connected with unavailable edges. These unavailable edges include the line segments intersecting or passing through obstacles. Subject to the PD constraints, the horizontal (vertical) edges on odd (even) layers are removed. This removal is equivalent to associate forbidden flags to unavailable edges on each layer. The

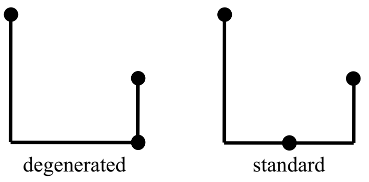

Fig. 4. All 3D U-shaped patterns are classified into (a) degenerated and (b) standard ones.

edge cost on layer i is magnified by UC_i, $1 \le i \le N_l$. Fig. 2(c) shows the 3D extended graph for the instance in Fig. 2(a).

By extending the proof done in [3], we can prove that at least one optimal OAPDST is embedded in the 3D extended escape graph; on the other hand, the proof of the 2:1 performance bound of MST to SMT (Steiner minimal tree) for general graphs can be applied here.

When a tree edge is rectilinearized and connected to the partially constructed rectilinear Steiner tree, a 3D U-shaped pattern may be formed. We generalize the definition of a U-shaped pattern and then propose novel 3D U-shaped pattern refinement to fix it. It can be proved that all 3D U-shaped patterns fall into two types:

1) Degenerated U-shape: The middle vertex is located in one turning corner of U. (see Fig. 4(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 on 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. 4(b)) This type can be identified by several L-shaped segments plus several L-shaped segments. For a given U-shaped pattern formed by three vertices, the median of their coordinates is the optimal Steiner-vertex if the PD constraints are not considered. Under the PD constraints, a Steiner-vertex can only connect vias either with vertical edges or with horizontal edges. Thus, the median 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

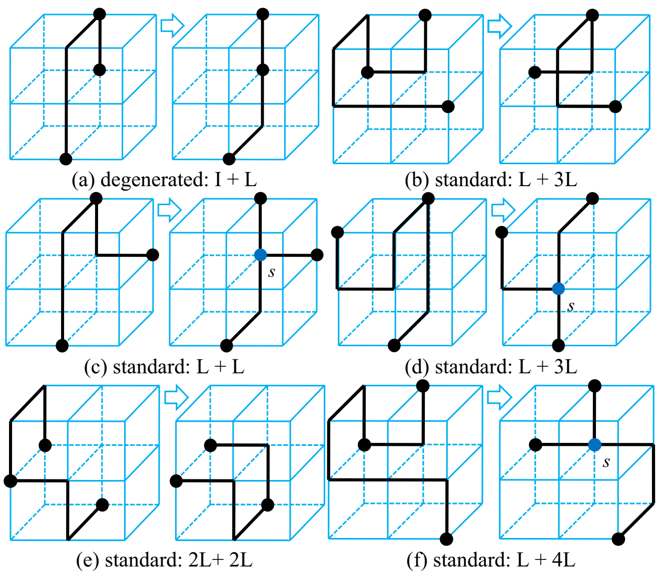

Fig. 5. Several cases for 3D U-shaped pattern refinement in OAPDST. (*s*: Steiner-vertex)

pattern.

More complicated cases can be decomposed into smaller ones. Fig. 5 lists several examples for 3D U-shaped pattern refinement. Please note that our classification is complete.

Corresponding to the instance in Fig. 2(a), Fig, 2(d)(e) depicts the resulting OAPDST without refinement (cost = 233 (=10×20+11×3)), with refinement (cost = 227 (=10×20+9×3)), respectively. Fig. 6(b) shows the ML-OARSMT generated by our algorithm without considering the PD constraints, cost = 218 (=10×20+6×3). The cost of ML-OARSMT can be viewed as the lower bound of that of OAPDST. On the other hand, Fig. 6(c) shows the OAPDST generated by [13], cost = 281 (=13×20+7×3), where a standard pattern is highlighted by bold lines. (see Fig. 6(d)) After refining this pattern, we can obtain a better tree in Fig. 6(e), cost = 261 (=12×20+7×3).

D. *Time Complexity Analysis*

As defined in Section II, m is the number of pins and k is the number of obstacles. Let $n=m+4k$. Step 1 takes O($m$lg$m$)

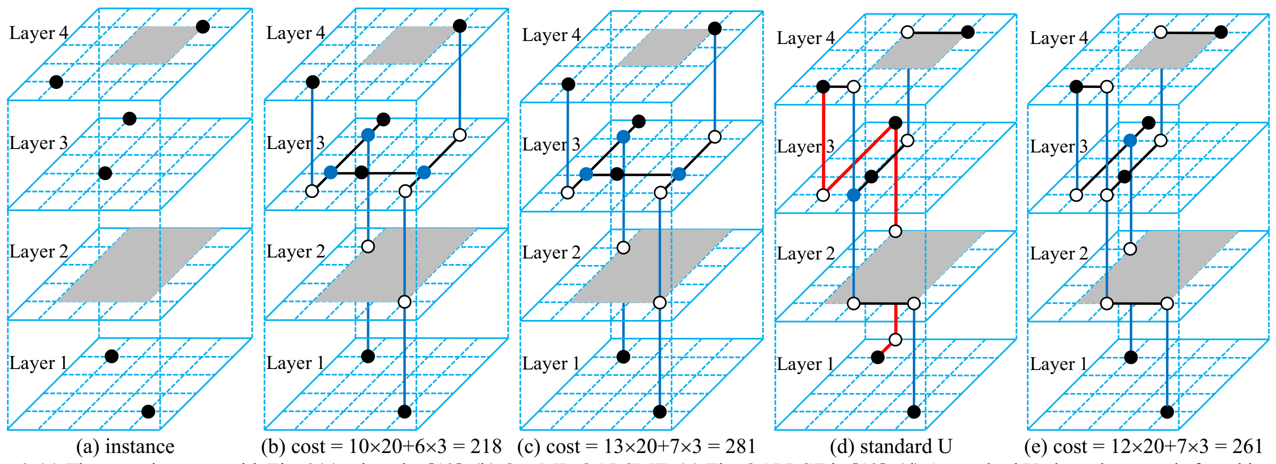
Layer 4
Layer 3
Layer 2
Layer 1
(a) instance
(b) cost = 10×20+6×3 = 218
(c) cost = 13×20+7×3 = 281
(d) standard U
(e) cost = 12×20+7×3 = 261

Fig. 6. (a) The same instance with Fig. 2(a), given by [13]. (b) Our ML-OARSMT. (c) The OAPDST in [13]. (d) A standard U-shaped pattern is found in (c). (e) The refined tree of (c).

time for DT construction [3]; 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 are necessary for guiding step 3, and they have low time complexities. Although 3D U-shaped pattern refinement in step 3 has a high time complexity, it can be expected to produce good solutions. Our results prove our strategy is promising.

IV. 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.

Totally 10 test cases are used. To demonstrate the differences between OAPDST and ML-OARSMT (which is a lower bound of OAPDST), we use almost the same set of test cases in [11]. ind3 is invalid under the PD constraints; thus it is unused in our experiments. ind4 (ind5) simulates 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. We insert one empty layer right above the working layer to ind4 (ind5) and obtain pd-ind4 (pd-ind5a). We then further duplicate the obstacles in the working layer onto the inserted layer of pd-ind5a and obtain pd-ind5b. In addition, the routing cost UC_i was set to 1 for all i. The parameter α was set to [0.7, 1.0]. Fig. 7 displays the OAPDST of ind2 as $C_v = 3$ and $UC_i = 1$.

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 2, as $C_v = 3$ and $UC_i = 1$, 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 3 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 have an 8.76% degradation on the total costs, and the CPU times surprisingly become much worse (38.8% 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 costs of [13] are still slightly worse (0.26% larger) than ours when refinement is turned off. Although not presented here, we have similar results for $C_v = 5$, and $UC_i \neq 1$.

V. Conclusion

This paper solved OAPDST based on construction-by-correction. Our algorithm can easily handle variant constraints, including multiple routing layers, obstacle-avoidance, preferred routing directions. Experimental results showed that our algorithm on average outperformed the state-of-the-art work. Future work includes performance-driven RSMT construction with variant constraints.

Acknowledgment

The authors would like to thank Prof. Yao-Wen Chang of National Taiwan University for sharing the test cases and gratefully acknowledge reviewers' valuable and insightful comments.

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($n\log n$) 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 *ISPD'07*, pp.127–134.

[10] J. Long, H. Zhou, and S. O. Memik, "An O($n\log n$) edge-based algorithm for obstacle-avoiding rectilinear Steiner tree construction,"in *Proc. 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. Li, and Y.-W. Chang, "Efficient multi-layer obstacle-avoiding rectilinear Steiner tree construction," in *Proc. IEEE/ACM Int. Conf. on Computer-aided Design (ICCAD'07)*, Nov. 2007, 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.

Table 2. OAPDST: The comparisons on the number of vias, the total cost, and CPU times between [13] and ours under $C_v = 3$, $UC_i = 1$, $1 \le i \le N_l$.

Test cases	m / k / N_l	Total cost (#via)			Time (s)		
		Ours_ML[1] (A)	Ours_PD[2] (B)	[13] (C)	Ours_ML (A)	Ours_PD (B)	[13] (C)
ind1	50 / 6 / 5	53,915 (45)	64,401 (77)	66,033 (81)	0.05	0.03	0.02
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
rt1	25 / 10 / 10	4,042 (67)	4,117 (82)	4,289 (83)	0.09	0.08	0.09
rt2	100 / 20 / 10	9,234 (188)	9,528 (268)	9,803 (257)	1.78	0.94	0.97
rt3	250 / 50 / 10	14,996 (456)	15,776 (619)	16,295 (619)	12.35	5.09	5.36
rt4	500 / 50 / 10	21,151 (915)	22,366 (1,217)	23,222 (1,144)	50.25	13.87	12.11
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.
[3]Imp. (%): Average improvement is computed by averaging (X-B)/X for all cases, where X = A, C.

Table 3. OAPDST: The comparisons on the impacts of our algorithm on the total cost and CPU times under $C_v = 3$, $UC_i = 1$, $1 \le i \le N_l$.

Test cases	m / k / N_l	Total cost			Time (s)		
		Nmst[1] (D)	Nref[2] (E)	[13]_ref[3] (F)	Nmst (D)	Nref (E)	[13]_ref (F)
ind1	50 / 6 / 5	69,913	64,401	66,166	0.05	0.03	0.05
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
rt1	25 / 10 / 10	4,466	4,146	4,146	0.13	0.09	0.11
rt2	100 / 20 / 10	10,502	9,594	9,599	1.72	0.97	1.02
rt3	250 / 50 / 10	17,030	15,825	16,011	9.42	4.72	6.39
rt4	500 / 50 / 10	24,149	22,458	22,495	24.47	11.16	21.89
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 2, X = D, E, F.

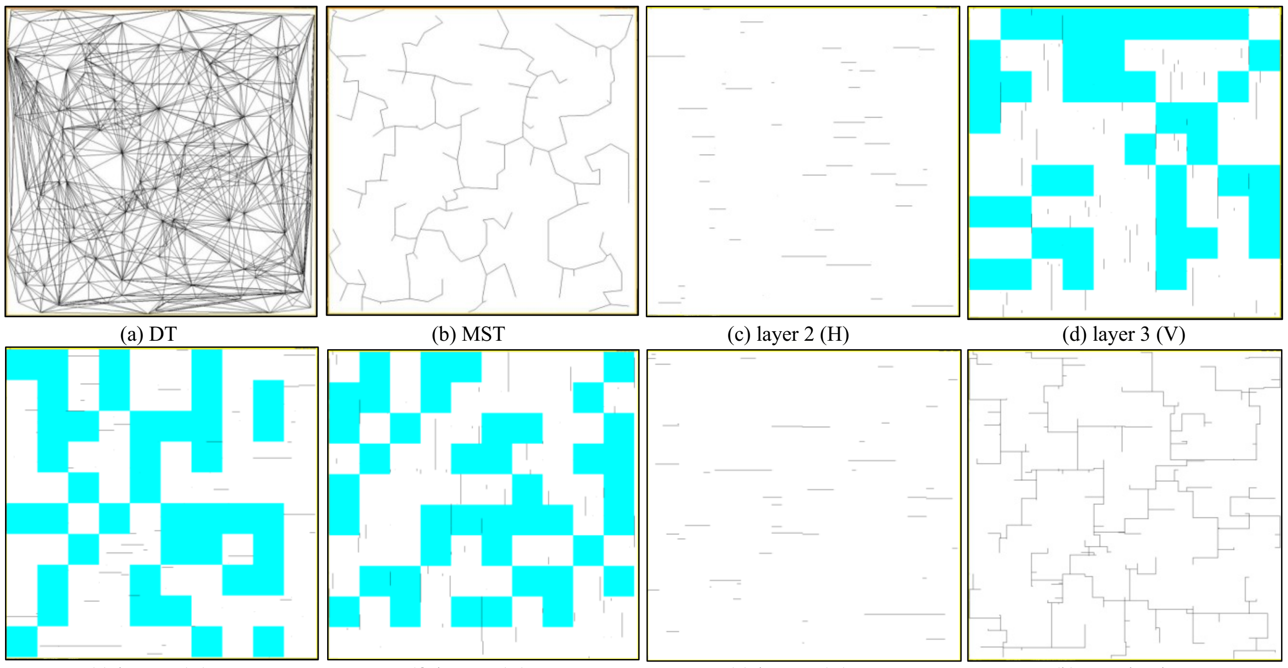

Fig. 7. The OAPDST of ind2 under $C_v = 3$. (a)(b) show the DT and MST. (c)–(g) show the results of layers 2–6, respectively. The odd layers allow vertical edges, while the even ones horizontal. Some line segments are at obstacle boundaries; they are feasible according to the problem formulation. (h) All pin-vertices are projected onto a pseudo plane, without showing the obstacles.