

there are multiple types of buffers and the candidate locations for buffering are specified. We also implemented a net-based buffering approach, called NetBIN. NetBIN buffers nets one by one until the timing constraint is satisfied. We also use a refinement step at the end of NetBIN to reduce the buffers, and our experience is that the refinement can reduce the buffers by at least 50%. Table II shows the comparison results of CostBIN, CutBIN, NetBIN, and [8]. There are four types of buffers, and the buffering candidate locations are specified. The timing constraints are 0.2 times greater than the minimal achievable delays from  $s$  to  $t$  by buffering, computed by the min-delay buffering algorithm in [8]. We observed that NetBIN inserts more buffers than [8] for most of those cases. NetBIN does not budget the timing in a global view, so the required time at sinks may mislead the buffering. Thus, buffering each net such that the required time at the root of the net is maximized does not guarantee that the delay from  $s$  to  $t$  is minimized. For example, for the last four test cases, it is difficult for NetBIN to obtain a feasible solution in 5 h. We also observed that the final solutions computed by the min-cost buffering algorithm in [8] often do not satisfy the timing constraint, so we select the solution that satisfies the timing constraint and has the minimal buffer area during the iterations. We also implemented the refinement step in CostBIN. The CostBIN without the refinement is denoted as CostBINv1, and the CostBIN with the refinement is denoted as CostBINv2. Column 3 shows the number of wire segments with negative slacks. Note that the wires in these test cases are separated into many small wire segments in this part, and the number of buffering locations (shown in column 2) is equal to the number of wire segments. The results indicate that CostBINv1 achieves a 52% reduction on the total area of buffers on average. The refinement step in CostBINv2 reduces an additional 12% of the buffering area on average, but “c7” and “c8” cannot be finished for CostBINv2 because of the big memory overhead. Also, the CostBIN inserts fewer buffers than the CutBIN in this situation.

## REFERENCES

- [1] C. J. Alpert and A. Devgan, “Wire segmenting for improved buffer insertion,” in *Proc. Des. Autom. Conf.*, 1997, pp. 588–593.
- [2] C. J. Alpert, M. Hrkic, and S. T. Quay, “A fast algorithm for identifying good buffer insertion candidate locations,” in *Proc. Int. Symp. Phys. Des.*, 2004, pp. 47–52.
- [3] R. Chen and H. Zhou, “Efficient algorithms for buffer insertion in general circuits based on network flow,” in *Proc. Int. Conf. Comput.-Aided Des.*, 2005, pp. 322–326.
- [4] J. Cong and Z. Pan, “Interconnect performance estimation models for design planning,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 6, pp. 739–752, Jun. 2001.
- [5] J. R. Ford and D. R. Fulkerson, *Flows in Networks*. Princeton, NJ: Princeton Univ. Press, 1962.
- [6] A. V. Karzanov and S. T. McCormick, “Polynomial methods for separable convex optimization in unimodular linear spaces with applications,” *SIAM J. Comput.*, vol. 26, no. 4, pp. 1245–1275, 1997.
- [7] J. Lillis, C. K. Cheng, and T. Y. Lin, “Optimal wire sizing and buffer insertion for low power and a generalized delay model,” *IEEE J. Solid-State Circuits*, vol. 31, no. 3, pp. 437–447, Mar. 1996.
- [8] I.-M. Liu, A. Aziz, and D. F. Wong, “Meeting delay constraints in DSM by minimal repeater insertion,” in *Proc. DATE*, 2000, pp. 436–440.
- [9] I.-M. Liu, A. Aziz, D. F. Wong, and H. Zhou, “An efficient buffer insertion algorithm for large networks based on Lagrangian relaxation,” in *Proc. Int. Conf. Comput. Des.*, 1999, pp. 210–215.
- [10] P. Saxena, N. Menezes, P. Cocchini, and D. A. Kirkpatrick, “The scaling challenge: Can correct-by-construction design help?” in *Proc. Int. Symp. Phys. Des.*, 2003, pp. 51–58.
- [11] W. Shi and Z. Li, “A fast algorithm for optimal buffer insertion,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 6, pp. 879–891, Jun. 2005.
- [12] W. Shi, Z. Li, and C. J. Alpert, “Complexity analysis and speedup techniques for optimal buffer insertion with minimum cost,” in *Proc. Asia South Pacific Des. Autom. Conf.*, 2004, pp. 609–614.
- [13] C. N. Sze, C. J. Alpert, J. Hu, and W. Shi, “Path based buffer insertion,” in *Proc. Des. Autom. Conf.*, 2005, pp. 509–514.
- [14] L. P. P. van Ginneken, “Buffer placement in distributed RC-tree networks for minimal Elmore delay,” in *Proc. Int. Symp. Circuits Syst.*, 1990, pp. 865–868.
- [15] H. Zhou, D. F. Wong, I.-M. Liu, and A. Aziz, “Simultaneous routing and buffer insertion with restrictions on buffer locations,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 19, no. 7, pp. 819–824, Jul. 2000.

## $\lambda$ -OAT: $\lambda$ -Geometry Obstacle-Avoiding Tree Construction With $O(n \log n)$ Complexity

Tom Tong Jing, Zhe Feng, Yu Hu, Xianlong L. Hong,  
Xiaodong D. Hu, and Guiying Y. Yan

**Abstract**—Obstacle-avoiding rectilinear Steiner minimal tree (OARSMT) construction is an essential part of routing. Recently, IC routing and related researches have been extended from Manhattan architecture ( $\lambda_2$ -geometry) to Y-/X-architecture ( $\lambda_3$ -/ $\lambda_4$ -geometry) to improve the chip performance. This paper presents an  $O(n \log n)$  heuristic,  $\lambda$ -OAT, for obstacle-avoiding Steiner minimal tree construction in the  $\lambda$ -geometry plane ( $\lambda$ -OASMT). In this paper, based on obstacle-avoiding constrained Delaunay triangulation, a full connected tree is constructed and then embedded into  $\lambda$ -OASMT by zonal combination. To the best of our knowledge, this is the first work addressing the  $\lambda$ -OASMT problem. Compared with most recent works on OARSMT problem,  $\lambda$ -OAT obtains up to 30-Kx speedup with quality solution. We have tested randomly generated cases with up to 10 K terminals and 10-K rectilinear obstacles within 4 seconds on a Sun V880 workstation (755-MHz CPU and 4-GB memory). The high efficiency and accuracy of  $\lambda$ -OAT make it extremely practical and useful in the routing phase.

**Index Terms**—Physical design, routing, Steiner tree, very large scale integration (VLSI).

## I. INTRODUCTION

Routing a net, finding a rectilinear Steiner minimal tree (RSMT) for a given terminal set, is a fundamental problem in physical design. In practical routing applications, macrocells, IP blocks, and prerouted nets are often regarded as obstacles. Thus, obstacle-avoiding rectilinear Steiner minimal tree (OARSMT) construction is often used as an accurate wire length even the delay estimation throughout the process of routing. It was proved that the RSMT problem is NP-complete [1].

Manuscript received April 30, 2006; revised December 18, 2006. This work was supported by the Key Project of Chinese Ministry of Education under Grant 106008, by the Specialized Research Fund for the Doctoral Program of Higher Education (SRFDP) of China under Grant 20050003099, and by the NSFC under Grant 90607001. This paper was recommended by Associate Editor P. H. Madden.

T. T. Jing was with the Computer Science and Technology Department, Tsinghua University, Beijing 100084, China. He is now with the Electrical Engineering Department, University of California at Los Angeles (UCLA), Los Angeles, CA 90095 USA (e-mail: tomjing@ucla.edu).

Z. Feng and X. L. Hong are with the Computer Science and Technology Department, Tsinghua University, Beijing 100084, China.

Y. Hu is with the Electrical Engineering Department, University of California at Los Angeles (UCLA), Los Angeles, CA 90095 USA.

X. D. Hu and G. Y. Yan are with the Institute of Applied Mathematics, Chinese Academy of Sciences, Beijing 100080, China.

Digital Object Identifier 10.1109/TCAD.2007.896291

Therefore, it is hardly to find polynomial-time algorithms to solve OARSMT problem.

The OARSMT problem has been well studied. The maze routing [2], [3] and line-probe [4], [5] algorithms were proposed for this problem. They need much space and take long running time since the complexity depends on the size of the routing area. Ganley and Cohoon [6] proposed some heuristics, G3S, G4S, and B4S, for cases with less than 20 terminals. Zhou *et al.* [7], [8] introduced a generalized connection graph and proposed an efficient algorithm to compute OARSMT, but only for a 3-terminal net. A 2-step heuristic [9] was proposed, which works well when the terminal number is less than 7 and the obstacles are convex polygons. More recently, a rectilinear approach was proposed in [22], which has the limitation of handling blockages with complex shapes such as concave polygons and large scale cases. FORst [10] can tackle large scale problem efficiently. An-OARSMan [11] has a good length performance when the terminal number is less than 100. FORst and An-OARSMan can route among both convex and concave polygon obstacles. CDCTree [21] can obtain a better length performance than An-OARSMan when the terminal number is less than 50.

$\lambda$ -geometry routing [15], [23]–[26] allows along  $\lambda > 2$  orientations forming consecutive angles of  $\pi/\lambda$ . In particular,  $\lambda = 2, 3, 4$ , and  $\infty$  correspond to Manhattan architecture, Y-architecture, X-architecture, and Euclidean geometry, respectively. Most attention from both academia and industry has been devoted to  $\lambda$ -geometry (especially  $\lambda = 3$  and 4) routing recently since the total wire length can be reduced up to 30% on some extreme cases compared with Manhattan routing, and the crosstalk can also be reduced [12], [13]. Note that the OARSMT problem is a subproblem of the obstacle-avoiding Steiner minimal tree (OASMT) construction problem in the  $\lambda$ -geometry plane ( $\lambda$ -OASMT) while  $\lambda$  is equal to 2. However, no algorithm exists to handle obstacle-avoiding Steiner minimal tree (SMT) problem in arbitrary  $\lambda$ -geometry.

In this paper, we propose an efficient algorithm,  $\lambda$ -OAT, to handle the problem of  $\lambda$ -OASMT with quality solution. The first contribution of this paper is to propose an  $O(n \log n)$  algorithm for OARSMT construction, where  $n$  is the sum of the terminal number and obstacle number. The second contribution is that we extend the algorithm to handle  $\lambda$ -OASMT with the same time complexity,  $O(n \log n)$ . To the best of our knowledge, this is the first work addressing the  $\lambda$ -OASMT problem.

The rest of this paper is organized as follows. In Section II, the outline of the  $\lambda$ -OAT algorithm is introduced. In Section III,  $\lambda$ -OAT is described in detail. Section IV shows the experimental results and Section V concludes the whole paper.

## II. OUTLINE OF THE HEURISTIC

The input of the  $\lambda$ -OAT algorithm is the set of terminals and the set of obstacles (T&O) shown in Fig. 1(a). The output is  $\lambda$ -OASMT shown in Fig. 1(f). The flow of  $\lambda$ -OAT is as follows, which falls into three steps.

Step 1) In the obstacle-free tree construction scenario, minimum spanning tree (MST) is usually used as an initial tree since it can be easily constructed and transformed into SMT. There are many existing MST algorithms. But the fast way is based on Delaunay triangulation (DT), which only needs  $O(n \log n)$  time ( $n$  is the number of the terminals) [16]. Moreover, DT has some desired properties that are useful in Step 2) and Step 3).

Therefore, we construct a DT on the set of terminals and *corner points* [18]. This DT is an Euclidean one [see Fig. 1(b)], which will not influence the following tree construction steps in different geometries. Then, all the

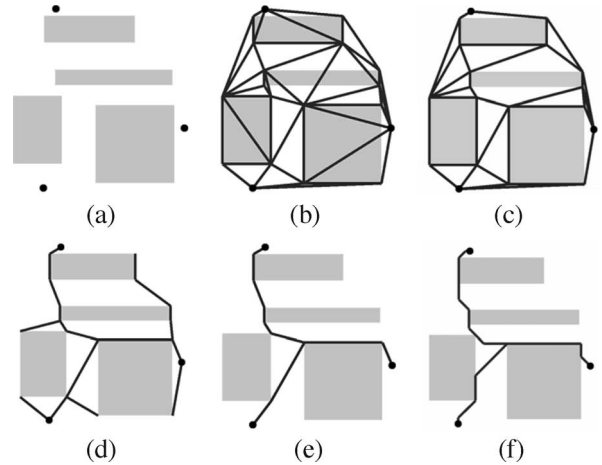


Fig. 1. Flow of  $\lambda$ -OAT. (a) T&O. (b) DT. (c) OACDT. (d) OAMST. (e) FCT. (f) 4-OASMT.

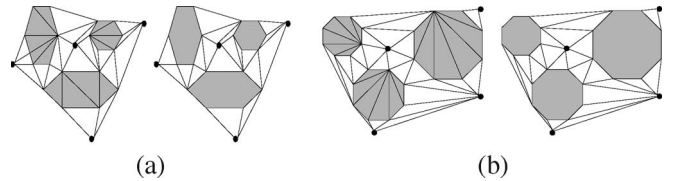


Fig. 2. Edge deleting in the  $\lambda$ -geometry plane. (a) Hexagon. (b) Octagon.

edges of DT which intersect with the obstacles are removed by *edge deleting*. As a result, obstacle-avoiding constrained DT (OACDT) is generated, which is the desired graph [see Fig. 1(c)]. Note that the edge-deleting-based OACDT construction can handle arbitrary-shape obstacles in the  $\lambda$ -geometry plane (see Fig. 2).

Step 2) The goal of this step is to construct an MST, which is suitable to be embedded into SMT. An obstacle-avoiding MST (OAMST) is generated on OACDT by Prim algorithm [17] [see Fig. 1(d)]. Then, we delete all the nonterminal leaves of the OAMST, which results in a full connect tree (FCT) [see Fig. 1(e)]. The definition of FCT will be given in Section III-B. The constructions of OAMST and FCT are performed on the Euclidean OACDT. Therefore, they are independent of geometry for the construction of  $\lambda$ -OASMT.

Step 3) The goal of this step is to embed an FCT into a  $\lambda$ -OASMT [see Fig. 1(f)]. Based on *zonal combination*, firstly, the coordinate space of each node in FCT is divided into  $2\lambda$  phases, where  $\lambda$  varies with different geometries. Then, in each phase, we combine all the geometries into one form to handle according to the corresponding  $\lambda$ . Finally, we can generate arbitrary geometry OASMT. Fig. 1(f) shows the result in 4-geometry.

## III. DETAILS OF THE $\lambda$ -OAT ALGORITHM

### A. Step 1: Construct an OACDT

To handle obstacles, an obstacle-avoiding DT is needed. In this step, we propose an efficient heuristic to construct an obstacle-avoiding DT. First, the definition of obstacle-avoiding DT is given as follows.

*Definition 1 (OACDT):* Obstacle-avoiding constrained DT (OACDT) is a kind of constrained DT. It is connective and has no edge intersecting with obstacles.

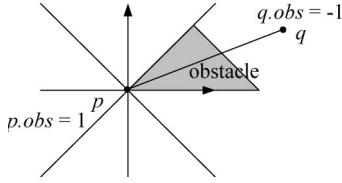


Fig. 3. Edge deleting in the 4-geometry.

To construct an OACDT, we first generate a DT based on both the terminal set and the corner point set since the graph may not be connective if DT is constructed only based on the terminal set. Then, we delete all the edges intersecting with the obstacles. Here, a technique, edge deleting, is provided to keep the process of deleting edges running in  $O(n \log n)$  time, where  $n$  is the sum of terminal number and corner point number.

*Technique of edge deleting:* There are two cases when an edge of DT intersects with a boundary of an obstacle. We'll delete all these intersecting-edges.

1) Case I: At least one end point of the edge is also a corner point of the intersecting obstacle.

We maintain a domain *obs* for each point, which indicates whether the point is a terminal or a corner point of an obstacle. If the point is a terminal, we set  $obs = -1$ . If the point is a corner point, we first divide the coordinate space of the point into  $2\lambda$  phases, and then set the *obs* a value from 1 to  $2\lambda$  to indicate the phase number that the obstacle is inside.

After we set the domain *obs* of each point according to the above declaration, we check each edge of DT whether it intersects with the obstacle by checking the domain *obs* of the point. If so, the intersecting edge will be deleted. Fig. 3 shows an example in 4-geometry.

2) Case II: Neither end point of the edge is a corner point of the intersecting obstacle.

Definitions and some notations used in this Subsection are given as follows.

- $CP_1$  First corner point of a boundary of an obstacle.
- $CP_2$  Second corner point of the boundary of an obstacle.
- $C$  Current handled point in the route.
- $L$  Last handled point before point  $C$  in the route.
- $CL$  Edge starting from point  $L$  to point  $C$ .
- $CM$  Point which is temporally found to be the other end point of the edge most possibly intersects with the obstacle, which connects with point  $C$ .
- $CA_1, CA_2, \dots$  Candidates as the other end point of the edge most possibly intersects with the obstacle, which connects with point  $C$ .
- $ML$  Edge starting from point  $C$  to point  $CM$ .

**Definition 2 (Route):** A route is the most adjacent path outside the obstacle, which connects a pair of corner points of a boundary of an obstacle.

**Property 1:** There is no other path connecting the pair of corner points, one of whose edge is inside the route (see Fig. 4).

To delete all the intersecting-edges in Case II, we check the edges along the route. Also, we should check the edges connecting with the route according to how acute the angle  $L-C-CA$  (starting from the least degrees) is (see Fig. 5). All these can be guaranteed by the following theorems.

**Theorem 1:** If the edge to be deleted in Case II connects with certain path connecting a pair of corner points of a boundary of an obstacle, the path must be a route. Otherwise, the edge must connect with an edge inside the polygon surrounded by the route and corresponding boundary of the obstacle.

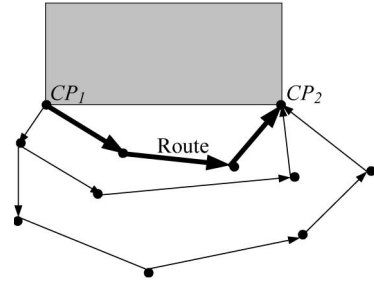


Fig. 4. Route.

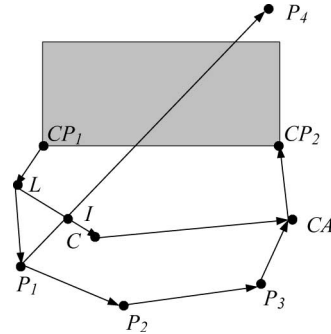


Fig. 5. Proof for Case a in Theorem 1.

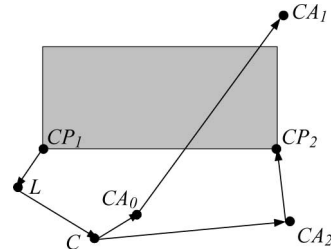


Fig. 6. Proof for Case b in Theorem 1.

*Proof:* Case a: Suppose that there is an edge to be deleted in Case II connect with a path connecting a pair of corner points of a boundary of an obstacle other than the route. Without loss of generality (shown in Fig. 5),  $P_1P_4$  is the edge intersecting with the obstacle, which connects with the path  $CP_1-L-P_1-P_2-P_3-CA-CP_2$  other than the route  $CP_1-L-C-CA-CP_2$ .

Since the checking is performed on the DT with some edge deleted, the graph should still be a planar plane. Therefore,  $P_1P_4$  must intersect with the route according to the definition of route and *Property 1* (there is no other path who has edge inside the polygon surrounded by the route and corresponding boundary). Without loss of generality, suppose that there is a point  $I$  on the intersection of edge  $P_1P_4$  and edge  $L-C$ , which is on the route. As a result,  $I-P_4$  is the edge intersecting with the obstacle, which connects with the route. It contradicts with the assumption.

Case b: If the edge to be deleted does not connect any path connecting with a pair of corner points of a boundary of an obstacle, the edge must connect with an edge inside the polygon surrounded by the route and corresponding boundary of the obstacle shown in Fig. 6.

The proof for Case b is similar to that for Case a. It is omitted here. ■

**Theorem 2:** If an edge  $A$  connecting with the route intersects with an obstacle, the edge  $B$  sharing the same end point of the route with less acute angle  $L-C-CA$  than  $A$  intersects with the obstacle, or the edge  $C$  following edge  $B$  intersects with the obstacle.

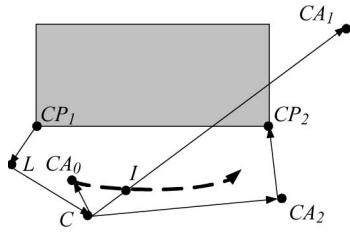


Fig. 7. Proof for Theorem 2.

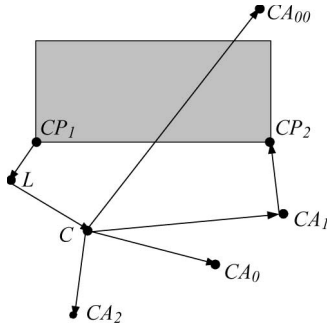


Fig. 8. Checking along the route.

*Proof:* Suppose that there is an edge  $C-CA_1$  connecting with the route intersects with the obstacle (shown in Fig. 7). An edge  $C-CA_0$  sharing the same end point  $C$  with edge  $C-CA_1$  of the route, but neither  $C-CA_0$  nor the edges following  $C-CA_0$  intersects with the obstacle.

Then, the edges following  $C-CA_0$  must intersect with the edge  $C-CA_1$ , suppose that the intersection is the point  $I$ . Therefore, the path segments  $C-CA_0-I$  will be instead of  $C-I$  becoming part of the route, which contradict with the assumption. ■

As showed in Fig. 8, we want to delete all the edges intersect with the boundary  $CP_1-CP_2$  in Case II. Our strategy is as follows.

It starts from one corner point  $CP_1$ . Suppose the current handled point is point  $C$ . We check all the edges connecting to point  $C$  to get the one with least acute angle  $L-C-CA$ . If it does not intersect with the obstacle, we will select it to form the path. Also, then the point  $CA_1$  becomes the next one to be handled. But if the edge intersects with the obstacle, it will be deleted. We will select the one with the second/third/... least acute angle  $L-C-CA$  to be checked until we find one nonintersecting the obstacle to form the path.

Here, we pay more attention to the following two issues.

- 1) How to find the edge with the least acute angle  $L-C-CA$  among all the edges connecting to the current handled point  $C$ ?

The edge with the least acute angle  $L-C-CA$  must be on the left side of the  $CL$ , which is because as for a closed polygon path along counterclockwise, the left side of any edge on the path is the interior of the polygon.

In Fig. 9(a), the edge  $C-CA_1$  is the one to be selected and detected since point  $CA_1$  is the other point besides point  $C$ , which is the one with the least acute  $L-C-CA$  and the edge  $C-CA_1$  is on the left side of  $CL$ .

Fig. 9(a) and (b) show different locations of point  $CM$ . In Fig. 9(a),  $C-CM$  is on the left side of  $L-C$ . In Fig. 9(b),  $C-CM$  is on the right side of  $L-C$ . We check all the edges inside the angle of  $CM-C-L$  for cases shown in Fig. 9(a) and (b), respectively.

- 2) How to determine whether an edge intersects with the obstacle?

After selecting the edge, e.g., the edge  $C-CA_1$  in Fig. 9, we check whether it intersects with the obstacle. There are totally

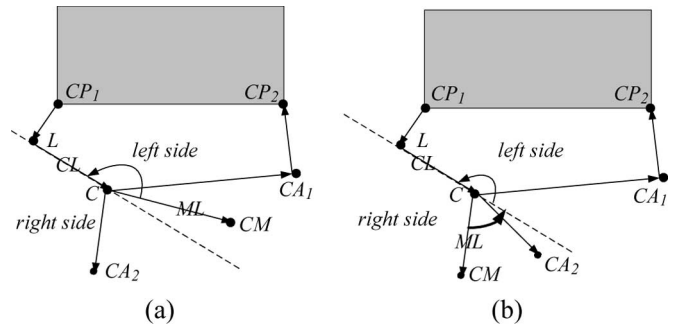


Fig. 9. Find the edge with the least acute angle  $L-C-CA$ .

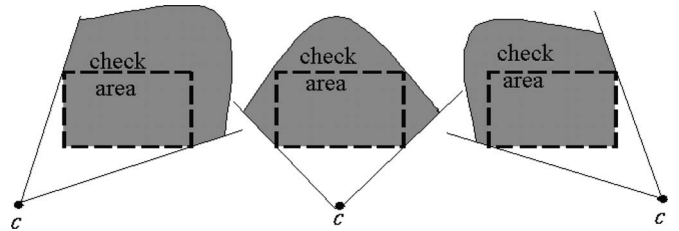


Fig. 10. Three situations of checking the intersection.

three situations as shown in Fig. 10. If the edge intersects with the obstacle, point  $CA_1$  must be in the *check area* (the shadow in Fig. 10), which can be checked out in constant time. As a result, all the edges intersecting with obstacles in both cases can be detected and deleted.

In order to keep the connectivity of OACDT, whenever a *cut edge* [20] is deleted during the process of edge deleting, we connect the separated parts by *detour method* [14]. The experimental results show that cut edges are never deleted in the algorithm. Therefore, the detour work does not impact the complexity in this step.

### B. Step 2: Construct Full Connected Tree (FCT)

*Definition 3 (FCT):* FCT is the tree connecting all terminals and some corner points on the OACDT, and all the leaves are terminals.

We construct an MST on the OACDT got from *Step 1* to connect all terminals and corner points, i.e., an OAMST, which is easy to be implemented since existing algorithms can be used. In this paper, OAMST is obtained by using Prim algorithm.

But, we find that some of the corner points of OASMT are not useful because they are leaves, which means that such corner points do not span other terminals. Then, to get better length performance, we arbitrarily choose a terminal as the root instead of a corner point. We traverse OAMST by depth first search (DFS) and delete all the nonterminal leaves. Finally, we get the FCT from OAMST.

### C. Step 3: Embed FCT Into $\lambda$ -OASMT

In this step, we embed FCT into  $\lambda$ -OASMT. *Zonal combination* is presented to tackle this problem.

The Embedding method [19] is used to transform MST into SMT. In [19], a special shape, L-shape for 2-geometry embedding, was given firstly. Then, step by step, the given shape pattern replaced the edges in the tree/graph to shorten the total wire length.

Our zonal combination method can unify all the geometries. The coordinate space of the current node is divided into  $2\lambda$  phases, which sets a shape frame for the corresponding geometry. Then, in each phase, we construct edges according to the geometry.

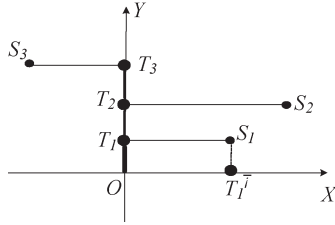


Fig. 11. Shared edges for 2-geometry.

Moreover, our zonal combination method makes the most use of shared edges in order to shorten the total wire length, and tackles obstacle-avoiding problem by means of bend technique (see *Definition 4*). The detailed description of the zonal combination method is as follows.

First, we traverse the FCT got from *Step 2* and handle all the nodes in FCT one by one. For each node  $O$ , there are several child nodes, such as  $S_1, S_2, S_3, \dots$  (see Fig. 11). We divide the coordinate space of the current node  $O$  into  $2\lambda$  phases and allocated all the child nodes of node  $O$  into the phases according to their relative position to point  $O$ .

Second, phases are handled one by one as follows. Since the path connecting the current node  $O$  and its parent node  $p$  has already been fixed, if we try to share the edges in the path to shorten the total wire length, we should choose the phase, including the path as a start-up. If there is not any constructed edge, we will handle the root. In this case, we will start in the first phase. Then, the rest phase is handled by the counterclockwise order. In each phase, all the child nodes are connected to point  $O$  by new edges.

There are following two key points.

- 1) How to make full use of shared edges to keep the wire length of the  $\lambda$ -OASMT as short as possible? There are two cases.

Case a: All the child nodes to be connected to point  $O$  are in the same phase, such as point  $S_1$  and  $S_2$  for 2-geometry in Fig. 11. Four points:  $T_1, T'_1, T_2$ , and  $T_3$  are turning points.

We strike the path  $O \rightarrow T_1 \rightarrow S_1$  and path  $O \rightarrow T_2 \rightarrow S_2$  instead of path  $O \rightarrow T'_1 \rightarrow S_1$  and path  $O \rightarrow T_2 \rightarrow S_2$  to share the same edge  $OT_1$ . As a result, the way that all the child nodes choose the same path turning direction (e.g., the path turning direction of path  $O \rightarrow T_1 \rightarrow S_1$  is the same as that of path  $O \rightarrow T_2 \rightarrow S_2$ ) will make use of the shared edge.

Case b: The child nodes are in the neighbor phases, such as point  $S_1$  and point  $S_3$  in Fig. 11. We handle all phases in counterclockwise. That is, the phase that point  $S_1$  is inside will be handled before the one  $S_3$  is inside. Then, if all the child nodes in the  $S_1$ -phase choose up-right turning, the child nodes in the  $S_3$ -phase will choose up-left turning instead of left-up turning to share edges and vice versa.

The method can be easily extended to any arbitrary geometry by replacing dividing the coordinate space into four phases by  $2\lambda$  phases. Then, the situation is similar to 2-geometry for edge sharing.

- 2) How to get new edges connecting point  $O$  and its child nodes avoiding obstacles?

The following concept of *bend* is helpful to solve this problem.

**Definition 4 (Bend):** During the process of embedding, if a new generated path intersects with a boundary of an obstacle, the path has to be altered in order to avoid the obstacle. The event is called bend. The new point, added into the path to alter it, is also called bend (see Fig. 12). How to find a bend?

In Fig. 12, node  $O$  is the current node, while node  $p$  is the node to be connected to node  $O$ . The angle  $op_2-O-op_1$  is the current

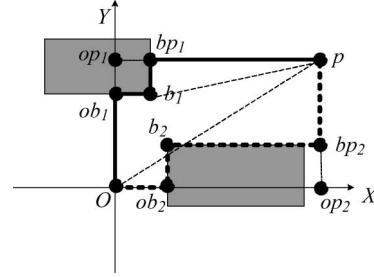


Fig. 12. Bend for 2-geometry.

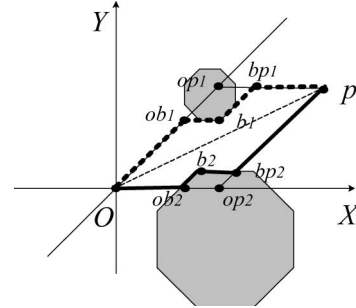


Fig. 13. Bend for 4-geometry.

phase of node  $O$ . From key point 1), we can get the turning. If it chooses the up-right turning ( $O \rightarrow op_1 \rightarrow p$ ) to connect node  $O$  and node  $p$ , we can find the bend in this way. We check in the rectangle  $O-op_1-p-op_2$  to find whether there is a neighbor of node  $p$  happening to be a corner point of an obstacle, and find out the corner number of the obstacle where the point is. In Fig. 12, if we find that the neighbor of the node  $p$ ,  $b_1$ , the right-down corner point of the obstacle, is in the rectangle, which means that the obstacle is in the rectangle  $O-op_1-p-op_2$  and it needs to be avoided. Thus,  $b_1$  is identified as a bend according to *Definition 4*.

After finding the bend, we need to avoid the obstacle. We construct new edges and generate new nodes to do so. New node  $ob_1$  and  $bp_1$  are generated along the right-down corner of the obstacle, where node  $ob_1$  is the intersection of  $Y$ -axis and the boundary while node  $bp_1$  is the intersection of the horizontal line passing  $p$  and the boundary. Thus, when a bend happens, the obstacle-avoiding path connecting node  $O$  and node  $p$  is made out, i.e.,  $O-ob_1-bend_1-bp_1-p$  (see Fig. 12).

From key point 1), if it chooses the right-up turning ( $O \rightarrow op_2 \rightarrow p$ ) to connect node  $O$  and node  $p$ , we can finally make out the obstacle-avoiding path as  $O-ob_2-bend_2-bp_2-p$ .

The method can be easily extended to any arbitrary geometry by replacing dividing the coordinate space into four phases by  $2\lambda$  phases firstly. Then, as shown in Fig. 13 for 4-geometry, suppose point  $O$  and point  $P$  are the two points to be connected, thus their positions are available. Also, we choose right-up as the turning direction to connect point  $O$  and point  $P$ . Bend  $b_2$  can be found in corresponding parallelogram just as 2-geometry. Hence, the position of point  $b_2$  is also available. The positions of the points to be added in order to avoid the bend can be got in the following formulas (for the first phase).

$$\begin{cases} ob_2.x = b_2.x - b_2.y \\ ob_2.y = b_2.y \end{cases} \quad \begin{cases} bp_2.x = p.x - (p.y - b_2.y) \\ bp_2.y = b_2.y \end{cases}$$

For different phases, the formulas have slight difference. As a result, the path avoiding the bend can be made out.

TABLE I  
COMPARISON BETWEEN FORst [10], An-OARSMAN [11], 2-OASMT, AND 4-OASMT ON WIRE LENGTH

Term#	Obs#	Wire length						
		[10]	[11]	2-OAT		4-OAT		
		Wire length	Wire length	Wire length	Improve to [10]/[11] (%)	Wire length	Improve to [10]/[11] (%)	Improve to 2-OAT (%)
10	10	29630	27840	30410	-9.23	27279	2.02	10.30
30	10	56880	43090	45640	-5.92	41222	4.34	9.68
50	10	64020	63250	58570	7.34	52432	17.10	10.48
70	10	69450	66310	63340	4.48	57699	12.99	8.91
100	10	84590	82320	83150	-1.01	73090	11.21	12.10
500	100	223415	-	198010	11.37	176497	21.00	10.86
1000	100	285821	-	250570	12.33	222758	22.06	11.10
10000	10000	-	-	3653240	-	3284620	-	10.09

#### D. Complexity of $\lambda$ -OAT Algorithm

In *Step 1*, we can generate DT by an  $O(n \log n)$  algorithm [16], where  $n$  is the sum of the terminal number and the corner point number in obstacles. That is,  $n = C_O \times n_O + n_T$ , where  $n_O$  is the number of obstacles,  $n_T$  is the number of terminals, and  $C_O$  is maximum of edges that one obstacle has.

During the process of transforming DT into OACDT, the edge deleting falls into two cases. In Case I, the process of edge deleting is with that of DT construction, which takes  $O(n \log n)$  time. In Case II, the time complexity is  $C_2 \times n_O$ , where  $n_O$  is the number of obstacles,  $C_2 = C_1 \times C'_1 \times C''_1$ ,  $C_1$  is the maximum number of the points in the shortest path connecting two neighbor corner points,  $C'_1$  is the maximum number of edges in the shortest path,  $C_1 \times C'_1$  is the period time for constructing the shortest path, and  $C''_1$  is the period time for checking the intersection. As a result,  $O(n \log n)$  dominate the time complexity of this part. Therefore, the time complexity of *Step 1* is  $O(n \log n)$ .

In *Step 2*, there exist  $O(n)$  edges in OACDT. Therefore, OAMST can be generated from OACDT by the  $O(n \log n)$  Prim algorithm [17]. During the process of deleting nonterminal leaves, the time complexity is dominated by DFS traversing. That is,  $C_3 \times n$ , where  $C_3$  is the maximum constant period time of the operations that every node is handled by the stack. Therefore, the time complexity of *Step 2* is  $O(n \log n)$ .

In *Step 3*, three For-loops in this step dominate the complexity. As for the first loop,  $n$  dominates. As for the second one,  $2\lambda$  dominates. As for the third one, it is a constant  $C_4$ , which is the maximum node number in one phase. Therefore, the time complexity of *Step 3* is  $O(n \times 2 \times \lambda \times C_4 \times C_5)$ , where  $C_5$  is the period time that generate new paths matching bend or nonbend, i.e.,  $O(n)$ .

All in all, the time complexity of our  $\lambda$ -OAT algorithm is  $O(n \log n)$ , where  $n$  is the sum of the terminal number and the corner point number of obstacles. We also can say that we get an  $O(n \log n)$  algorithm, where  $n$  is the sum of the terminal number and obstacle number.

## IV. EXPERIMENTAL RESULTS AND DISCUSSIONS

### A. Results

We have implemented the  $\lambda$ -OAT algorithm in C++ language and performed it on a Sun V880 fire workstation with 755-MHz CPU and 4-GB memory. We generate test cases in a  $32767 \times 32767$  plane.

Tables I and II show the results compared with FORst [10] and An-OARSMAN [11].

Table III shows the results of  $\lambda$ -OAT testing on cases with more obstacles.

TABLE II  
COMPARISON BETWEEN FORst [10], An-OARSMAN [11], 2-OASMT, AND 4-OASMT ON RUNNING TIME

Term#	Obs#	Runtime (s)					
		[10]	[11]	2-OAT		4-OAT	
		Time	Time	Time	Speedup	Time	Speedup
10	10	0.095	0.164	0.002	47x	0.003	32x
30	10	0.584	1.075	0.003	195x	0.003	195x
50	10	0.868	3.504	0.004	217x	0.004	217x
70	10	2.759	10.552	0.004	690x	0.005	552x
100	10	6.312	26.974	0.004	1578x	0.005	1262x
500	100	559.000	-	0.026	21500x	0.033	16939x
1000	100	1240.00	-	0.037	33514x	0.045	27556x
10000	10000	-	-	3.220	-	3.653	-
						Average:8249x	

TABLE III  
TESTING ON CASES WITH MORE OBSTACLES

Term#	Obs#	Wire length		Runtime (s)	
		2-OAT	4-OAT	2-OAT	4-OAT
100	500	149725	135454	0.057	0.070
200	500	181470	162762	0.062	0.075
200	800	202741	182056	0.095	0.119
200	1000	214850	193228	0.129	0.154
1000	10000	1723990	1564170	2.823	3.030

### B. Discussions

- 1) In Table I, the performance of 2-OAT is better than that of FORst on larger test cases since FORst divides the problem into several subproblems (*Step 1* in FORst). Therefore, it just searches in a smaller searching space and loses optimality. However, our algorithm tries a global search. That is, constructing DT, OACDT, OAMST, FCT, and  $\lambda$ -OASMT pay respect to all obstacles and all terminals. Hence, 2-OAT can produce better solution for larger scale problems.
- 2) DT is constructed on terminals and corner points. When the test case (NOT a net) is small (i.e., *small scale net with only a few obstacles*), the corner points have a worse effect on DT topology since the triangulation has to span the corner points. But actually, it may not need such corner points to perform obstacle-avoiding in this case. Therefore, this worse effect of corner points leads to a longer wire length. When the test case is large, it means not only more terminals, but also more obstacles. To get an obstacle-avoiding tree, our algorithm has the technique of shared edge. In such situation, there are many shared edges due to many nodes. Then, we can get a more short wire length. Thus,  $\lambda$ -OAT is suitable for practical applications since there are at least many obstacles actually.

3) For small test cases, FORst and An-OARSMAN can produce better solutions with more expensive computational time. An-OARSMAN generates Steiner tree in track graph, which preserves most of the optimal Steiner point candidates (for the special case where all obstacles are rectangles, the track graph is equivalent to the Hanan graph, which contains all optimal Steiner point candidates). However, the complexity of constructing a Steiner tree in such a connection graph is high and the algorithm is basically an iterative approach, where the iteration number, for example  $N$ , is fixed. For small-scale cases, the track graph is small, then a near optimal solution can be achieved within  $N$  iterations. However, for a large connection graph,  $N$  iterations can only explore a small part of the solution space and therefore lead to low performance (longer wire length).

In Table II, we can see that  $\lambda$ -OAT achieves 8249x speedup on FORst/An-OARSMAN on average. The speedup increases for larger test cases. We can obtain up to 30-Kx speedup for the largest test cases.

4) To study on the scalability of  $\lambda$ -OAT, we test some cases with additional more obstacles, which are more similar to the practical routing applications, such as detailed routing, or engineering change order (ECO) routing. From Table III, we can see that  $\lambda$ -OAT consistently runs efficiently for all test cases. Particularly,  $\lambda$ -OAT can route 10 000 terminals in the presence of 10 000 rectangular obstacles in both 2-geometry and 4-geometry within 4 seconds.

## V. CONCLUSION

In this paper, we propose an efficient and unified heuristic,  $\lambda$ -OAT, for  $\lambda$ -OASMT construction.  $\lambda$ -OAT is fit for arbitrary interconnect architecture (e.g., Y/X-Architecture), which is helpful for further shortening the wire length and reducing crosstalk compared with the traditional Manhattan Architecture. The time-complexity of  $\lambda$ -OAT is  $O(n \log n)$ , where  $n$  is the sum of the terminal number and obstacle number. Compared with two recent works focusing on OARSMT problem,  $\lambda$ -OAT obtains up to 30-Kx speedup with quality solutions. We have tested the randomly generated cases with up to 10-K terminals and 10-K rectilinear obstacles within 4 seconds on a Sun V880 workstation (755-MHz CPU and 4-GB memory). The high efficiency and accuracy of  $\lambda$ -OAT make it is extremely practical and useful in the process of routing.

## REFERENCES

- [1] M. R. Garey and D. S. Johnson, "The rectilinear Steiner tree problem is NP-complete," *SIAM J. Appl. Math.*, vol. 32, no. 4, pp. 826–834, Jun. 1977.
- [2] C. Y. Lee, "An algorithm for path connection and its application," *IRE Trans. Electron. Comput.*, vol. 10, no. 3, pp. 346–365, Mar. 1961.
- [3] F. Rubin, "The Lee path connection algorithm," *IEEE Trans. Comput.*, vol. C-23, no. 9, pp. 907–914, Sep. 1974.
- [4] K. Mikami and K. Tabuchi, "A computer program for optimal routing of printed circuit connectors," in *Proc. IFIPS*, 1968, pp. 1475–1478.
- [5] D. W. Hightower, "A solution to line-routing problems on the continuous plane," in *Proc. DAC*, 1969, pp. 1–24.
- [6] J. L. Ganley and J. P. Cohoon, "Routing a multi-terminal critical net: Steiner tree construction in the presence of obstacles," in *Proc. IEEE ISCAS*, 1994, pp. 113–116.
- [7] Z. Zhou, C. D. Jiang, L. S. Huang, and J. Gu, "Finding obstacle-avoiding shortest path using generalized connection graph with  $\Theta(t)$  edges," *J. Softw.*, vol. 14, no. 2, pp. 166–174, Feb. 2003.
- [8] Z. Zhou, C. D. Jiang, L. S. Huang, and J. Gu, "On optimal rectilinear shortest paths and 3-Steiner tree routing in presence of obstacles," *J. Softw.*, vol. 14, no. 9, pp. 1503–1514, Sep. 2003.
- [9] Y. Yang, Q. Zhu, T. Jing, and X. L. Hong, "Rectilinear Steiner minimal tree among obstacles," in *Proc. IEEE ASICON*, 2003, pp. 348–351.
- [10] Y. Hu, Z. Feng, T. Jing, X. L. Hong, Y. Yang, G. Yu, X. D. Hu, and G. Y. Yan, "FORst: A 3-Step heuristic for obstacle-avoiding rectilinear Steiner minimal tree construction," *J. Inf. Comput. Sci.*, vol. 1, no. 3, pp. 107–116, Dec. 2004.
- [11] Y. Hu, T. Jing, X. L. Hong, Z. Feng, X. D. Hu, and G. Y. Yan, "An-OARSMAN: Obstacle-avoiding routing tree construction with good length performance," in *Proc. ASP-DAC*, 2005, pp. 7–12.
- [12] *X Initiative Home Page*, 2001. [Online]. Available: <http://www.xinitiative.com>
- [13] H. Y. Chen, C. K. Cheng, A. B. Kahng, I. Mandoiu, and Q. K. Wang, "Estimation of wirelength reduction for  $\lambda$ -geometry vs. Manhattan placement and routing," in *Proc. Int. Workshop Syst.-Level Interconnect Prediction*, 2003, pp. 71–76.
- [14] S. Q. Zheng, J. S. Lim, and S. S. Iyengar, "Finding obstacle-avoiding shortest paths using implicit connection graphs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 15, no. 1, pp. 103–110, Jan. 1996.
- [15] M. Sarrafzadeh and C. K. Wong, "Hierarchical Steiner tree construction in uniform orientations," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 11, no. 9, pp. 1095–1103, Sep. 1992.
- [16] S. Fortune, "A sweepline algorithm for Voronoi diagrams," *Algorithmica*, vol. 2, no. 2, pp. 153–174, 1987.
- [17] R. C. Prim, "Shortest connection networks and some generalizations," *Bell Syst. Tech. J.*, vol. 36, no. 6, pp. 1389–1401, Jun. 1957.
- [18] M. Zachariasen, "Rectilinear full Steiner tree generation," *Networks*, vol. 33, no. 3, pp. 125–143, Mar. 1999.
- [19] J. M. Ho, G. Vijayan, and C. K. Wang, "A new approach to the rectilinear Steiner tree problem," in *Proc. DAC*, 1989, pp. 161–166.
- [20] H. Schenck, *Computational Algebraic Geometry*. Cambridge, U.K.: Cambridge Univ. Press, Nov. 2003.
- [21] Y. Y. Shi, T. Jing, L. He, Z. Feng, and X. L. Hong, "CDCTree: Novel obstacle-avoiding routing tree construction based on current driven circuit model," in *Proc. ASP-DAC*, 2006, pp. 630–635.
- [22] Z. Shen, C. N. Chu, and Y. M. Li, "Efficient rectilinear Steiner tree construction with rectilinear blockages," in *Proc. IEEE ICCD*, 2005, pp. 38–44.
- [23] C. S. Coulston, "Constructing exact octagonal Steiner minimal trees," in *Proc. GLSVLSI*, 2003, pp. 1–6.
- [24] A. B. Kahng, I. Mandoiu, and A. Z. Zelikovsky, "Highly scalable algorithms for rectilinear and octilinear Steiner trees," in *Proc. ASP-DAC*, 2003, pp. 827–833.
- [25] D. T. Lee and C. F. Shen, "The Steiner minimal tree problem in the lambda-geometry plane," in *Proc. Int. Symp. Algorithms and Comput.*, 1996, pp. 247–255.
- [26] C. K. Koh and P. H. Madden, "Manhattan or non-Manhattan? A study of alternative VLSI routing architectures," in *Proc. GLSVLSI*, 2000, pp. 47–52.