FH-OAOS: A Fast Four-Step Heuristic for Obstacle-Avoiding Octilinear Steiner Tree Construction

XING HUANG, WENZHONG GUO, GENGGENG LIU, and GUOLONG CHEN,
Fu Zhou University

With the sharp increase of very large-scale integrated (VLSI) circuit density, we are faced with many knotty issues. Particularly in the routing phase of VLSI physical design, the interconnection effects directly relate to the final performance of circuits. However, the optimization capability of traditional rectilinear architecture is limited; thus, both academia and industry have been devoted to nonrectilinear architecture in recent years, especially octilinear architecture, which is the most promising because it can greatly improve the performance of modern chips. In this article, we design FH-OAOS, an obstacle-avoiding algorithm in octilinear architecture, by constructing an obstacle-avoiding the octilinear Steiner minimal tree (OAOSMT). Our approach first constructs an obstacle-free Euclidean minimal spanning tree (OFEMST) on the given pins based on Delaunay triangulation (DT). Then, two lookup tables about OFEMST's edge are generated, which can be seen as the information center of FH-OAOS and can provide information support for algorithm operation. Next, an efficient obstacle-avoiding strategy is proposed to convert the OFEMST into an obstacle-avoiding octilinear Steiner tree (OAOST). Finally, the generated OAOST is refined to construct the final OAOSMT by applying three effective strategies. Experimental results on various benchmarks show that FH-OAOS achieves 66.39 times speedup on average, while the average wirelength of the final OAOSMT is only 0.36% larger than the best existing solution.

CCS Concepts: • **Hardware** → **Integrated circuits**; *Interconnect*; **Electronic design automation**; *Physical design (EDA)*; Wire routing; • **Theory of computation** → Design and analysis of algorithms;

Additional Key Words and Phrases: Octilinear architecture, VLSI design, router, Steiner tree, obstacle

ACM Reference Format:
Xing Huang, Wenzhong Guo, Genggeng Liu, and Guolong Chen. 2016. FH-OAOS: A fast four-step heuristic for obstacle-avoiding octilinear Steiner tree construction. ACM Trans. Des. Autom. Electron. Syst. 21, 3, Article 48 (April 2016), 31 pages.
DOI: http://dx.doi.org/10.1145/2856033

1. INTRODUCTION

Physical design of modern very large-scale integrated (VLSI) circuits includes several steps. As one of the most important steps, global routing plays a decisive role during the chip design. On the other hand, the construction of a Steiner minimum tree (SMT),

This work was supported in part by the National Basic Research Program of China under Grant 2011CB808000, the National Natural Science Foundation of China under Grant 11271002 and Grant 11501114, the Fujian Province High School Science Fund for Distinguished Young Scholars under Grant JA12016, the Fujian Natural Science Funds for Distinguished Young Scholars under Grant 2014J06017, the Program for New Century Excellent Talents in Fujian Province University under Grant JA13021, and the development foundation of the Education Committee of Fujian Province under Grant JA13356.
Authors' addresses: X. Huang, W. Guo (corresponding author), G. Liu, and G. Chen, College of Mathematics and Computer Sciences, Fuzhou University, Fuzhou, Fujian, China, 350116, and Key Laboratory of Network Computing and Intelligent Information Processing, Fuzhou, Fujian, China, 350116; emails: fzu_hx@163.com, fzugwz@163.com, liu_genggeng@126.com, fzucgl@163.com.

DOI: http://dx.doi.org/10.1145/2856033

which seeks to connect a set of terminals on a silicon chip ensuring the minimum wirelength between them, is essential for each signal net in the global routing process [Hashimotao and Stevens 1988]. In addition, in the current technology, interconnects play a significant role in determining several metrics such as timing, congestion, and vias [Ho et al. 2007; Hu and Sapatnekar 2001]. Hence, it is critical to have a router that can produce high-quality routing results.

Ever since the Hanan grid was proposed in 1966 [Hanan 1966], the rectilinear Steiner minimal tree (RSMT) has been widely applied in the VLSI circuit routing problem [Ganley and Cohoon 1996; Ho et al. 1989], and it has been shown to be NP-complete [Garey and Johnson 1977]. On the other hand, due to the sharply increasing density of the VLSI circuit, many reusable components are integrated in modern chips, such as IP cores and macro blocks. These components cannot be run through during the routing process. Consequentially, for the past several years, the obstacle-avoiding RSMT (OARSMT) construction problem has been widely studied [Liu et al. 2009; Wu et al. 1987; Ajwani et al. 2011; Long et al. 2008; Lin et al. 2007; Liu et al. 2009; Chow et al. 2014; Huang and Young 2010, 2011; Huang et al. 2011].

However, because the routing directions of rectilinear architecture are restricted to be only horizontal and vertical, rectilinear architecture has limited ability to optimize many key performance indicators, such as wirelength and congestion. Thus, the routing technology is now standing at a crossroads and arousing wide attention from many research institutions [Koh and Madden 2000]. It follows that as manufacturing technology of VLSI circuits has advanced, nonrectilinear architectures have been developed. As one of the most promising nonrectilinear architectures, octilinear architecture is fully supported by several current manufacturing technologies [Teig 2002]. In other words, this means that 45° and 135° diagonal segments can be permitted in an octilinear routing plane. Compared with traditional rectilinear architecture, octilinear architecture can greatly optimize the wirelength, and thereby lower the cost of time delay, wire capacitance, and congestion. For example, the octilinear Steiner minimal tree (OSMT) in Teig [2002] has shown that the number of vias, total wirelength, and die size are reduced by 40%, 20%, and 11% compared to RSMT, respectively. Thus, both academia and industry have been devoted to octilinear architecture in recent years. However, to the best of our knowledge, most papers currently mainly assume an obstacle-free octilinear routing plane [Kahng et al. 2003; Coulston 2003; Ho et al. 2005; Arora and Mose 2009; Zhu et al. 2005; Yan 2008; Huang et al. 2009; Liu et al. 2015]; there are only two works for the obstacle-avoiding OSMT (OAOSMT) construction problem [Huang et al. 2015; Jing et al. 2007]. Therefore, designing a great OAOSMT construction algorithm becomes very important.

For the reasons mentioned, we develop a new heuristic algorithm for OAOSMT construction in this article.

The rest of this article is organized as follows. Section 2 introduces the related work and gives the motivation of our work. The OAOSMT problem model and some definitions are given in Section 3. We illustrate the framework of the proposed algorithm and describe the details of the proposed algorithm in Section 4. Section 5 shows experimental results, and Section 6 concludes the article.

2. RELATED WORK

In this section, we introduce four categories of works related with our work, namely, RSMT, OARSMT, OSMT, and OAOSMT.

2.1. RSMT

As one of the basic models of VLSI physical design, RSMT can be applied in floorplanning, placement, and routing stages. There are lots of research achievements focusing

on the RSMT problem. For example, Borah et al. [1994] proposed an edge substitution approach, which starts with a minimal spanning tree (MST) and iteratively connects a point to a nearby edge, and then deletes the longest edge on the formed cycle. Warme et al. [2000] proposed the GeoSteiner algorithm, which is currently the fastest exact algorithm for constructing the optimal RSMT in the plane Steiner tree problem. However, GeoSteiner has a high time complexity and is impractical with the increasing scale of the routing problem. Tong and Wu [1995] and Schmiedle et al. [2003] applied rectilinear routing tree to the three-dimensional channel routing and detailed routing problem, respectively. Koren and Koren [2000] proposed to optimize the total cost and yield at the same time based on rectilinear architecture during the floorplanning process. By combining the edge substitute concept in Borah et al. [1994] with a spanning graph, an efficient RSMT construction algorithm was proposed by Zhou [2004]. In addition, Cinel and Bazlamacci [2008] used Zhou's rectilinear Steiner tree [Zhou 2004] and Kahng et al.'s batched greedy algorithm (BGA) [Kahng et al. 2003] as the basis to propose an effective RSMT algorithm to generate great results in much shorter time. In particular, a very fast and accurate RSMT algorithm called FLUTE was proposed by Chu and Wong [2008]. Based on a precomputed lookup table, FLUTE can construct an optimal RSMT for low-degeree nets (up to degree 9). For high-degree nets, they also presented a net-breaking technique to reduce the net size until the table can be used.

2.2. OARSMT

In addition to the RSMT algorithms mentioned in Section 2.1, OARSMT has been widely studied in recent years. Since the RSMT problem has been proved to be NP-complete, the presence of obstacles further increases the difficulty in finding a satisfactory solution. However, there are many heuristics that can achieve great results in a short time. Moreover, several exact algorithms have been proposed to generate the optimal OARSMT. For example, Wu et al. [1987] defined a grid-like track graph, and then a hybrid method to search for the shortest path while simultaneously constructing an MST. The obtained routing tree can be seen as an approximation of the Steiner tree. Ajwani et al. [2011] proposed a top-down approach called FOARS. FOARS first partitions a set of pins into several subsets. Then, by using an obstacle-aware version, FLUTE, an obstacle-avoiding Steiner tree is generated for each pin subset. Finally, the final OARSMT is generated by merging these subtrees. An efficient four-step algorithm called EBOARST was proposed by Long et al. [2008]. Compared to the state-of-the-art algorithm at that time [Lin et al. 2007], EBOARST achieves 16.56 times speedup on average, and -0.46% wirelength reduction. Recently, based on a maze-routing-based sequential approach, Chow et al. [2014] proposed a new parallel approach with the aid of graphic processing units (GPUs). Furthermore, there are several exact OARSMT algorithms [Huang and Young 2010, 2011; Huang et al. 2011], which can get the optimal routing tree among obstacles. However, it is not practical for a large input size because any exact algorithm is expected to have an exponential worst-case time complexity.

2.3. OSMT

For octilinear architecture, due to the advantages stated in Section 1, a special industry alliance has arisen to promote octilinear architecture in recent years. Under this background, Hong et al. [2003] pointed out that octilinear architecture would become a hot issue of VLSI physical design. In particular, they said that OSMT was one of the most critical problems under octilinear architecture. Moreover, some challenges and opportunities offered by octilinear architecture were presented in Teig [2002] at the same time. Therefore, as a new fundamental problem, some algorithms have been proposed for solving the OSMT construction problem. For example, Kahng et al. [2003] proposed a near-optimal RSMT construction algorithm based on triple contraction,

which can be easily modified to an OSMT construction algorithm with a small increase in runtime. Zhu et al. [2005] proposed two OSMT construction algorithms that are based on octilinear spanning graph and combined with edge substitution and triple contraction, respectively. Coulston [2003] proposed an exact OSMT construction algorithm, but it is not practical since it has an exponential complexity. The algorithm in Ho et al. [2005] was proposed to construct an OSMT for the multilevel full-chip routing problem. Since the swarm intelligence algorithm has been shown to have good application prospects in solving the NP-hard problem, Arora and Mose [2009] proposed an ant colony optimization-based algorithm to minimize wirelength, vias, and capacitance in octilinear architecture. The experimental results in Arora and Mose [2009] obtain an overall improvement of 8% in terms of wirelength and 7% in terms of vias and capacitance, but runtime is unsatisfactory due to the nature of the swarm-based algorithm. Moreover, there are several algorithms [Yan 2008; Huang et al. 2009; Liu et al. 2015] that are proposed to construct a timing-driven OSMT and can notably improve the timing performance of chips compared to RSMT. For instance, compared to RSMT, experimental results in Liu et al. [2015] obtain a 24.07% and 6.59% reduction in radius and wirelength, respectively.

2.4. OAOSMT

However, all algorithms mentioned in Section 2.3 are based on an obstacle-free plane. To the best of our knowledge, Jing et al. [2007] and Huang et al. [2015] are the only two related works for solving the octilinear obstacle-avoiding problem. Jing et al. [2007] first constructs a fully connected tree based on obstacle-avoiding constrained Delaunay triangulation (DT). Then the algorithm embeds the tree into a λ-obstacle-avoiding Steiner minimal tree by zonal combination. It is efficient because the time complexity is only $O(n \log n)$. But since it is designed for solving the λ-geometry problem, the routing results for octilinear architecture are not satisfying. Especially when the scale of the problem is large, the obtained results are even worse than rectilinear architecture. Huang et al. [2015] proposed a novel particle swarm optimization (PSO)-based OAOSMT construction algorithm, which can generate great results on various benchmarks. However, due to the nature of PSO, which is a kind of swarm intelligence algorithm, it often needs multiple iterations. Moreover, the number of candidate edges of Huang et al. [2015] is $O(n^2)$; therefore, the runtime of this algorithm is relatively poor: even when the scale of problem becomes larger, the runtime can reach tens of seconds.

In view of the shortages of the existing works and the requirements of VLSI physical design, this article designs an efficient and effective OAOSMT construction algorithm. In other words, Jing et al. [2007] can generate an OAOSMT in an efficient way, but the quality of the result is relatively poor. Huang et al. [2015] can generate an excellent OAOSMT, but it needs much more time. Our algorithm makes up for the drawbacks of these two algorithms. The main contributions of this article can be summarized as follows:

—We propose a new lookup-table-based OAOSMT construction technique, which can generate an excellent solution in an efficient way.
—Based on the method of speediness of the lookup table, an efficient and effective obstacle-avoiding strategy is proposed, which can efficiently transform an initial obstacle-free Euclidean minimal spanning tree (OFEMST) into an obstacle-avoiding octilinear Steiner tree (OAOST). In particular, we propose a pseudo Steiner point selection method. By catching the global view of obstacles and selecting some points on obstacles as intermediate nodes, this method can help OFEMST edges to accurately avoid all obstacles.

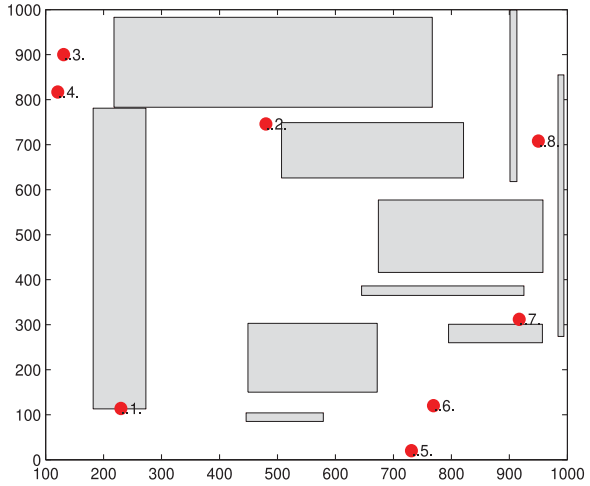

Fig. 1. An example of input information [Huang et al. 2015].

—We develop an effective refinement method, which can fully leverage the principle of shared edges and unused routing resources. By computing the optimal interconnection structure of each terminal and then applying these structures to the original OAOST, an OAOSMT can be achieved efficiently.
—Both the wirelength and runtime of our algorithm are inspiring when compared to the previous obstacle-avoiding algorithms. It is valuable for both scientific research and industry production.

3. PROBLEM FORMULATION

The input of our algorithm consists of n pin vertices $P = \{p_1, p_2, p_3 \dots p_n\}$ and m rectangular obstacles $B = \{b_1, b_2, b_3 \dots b_m\}$, where p_i and b_i are the serial number of the i^{th} pin and the i^{th} obstacle, respectively. A pin cannot be located inside any obstacle, but it could be located on the boundary of an obstacle. Any two obstacles cannot overlap with each other except for the boundary.

Figure 1 shows an example of input information. There are eight pin vertices and 10 obstacles. No pin vertex can be located inside an obstacle, but pin 1 and pin 7 are located on the boundary of the obstacle.

Definition 3.1 (*Routing Architecture*). In two-dimensional λ-geometry, only orientations with angles $i\pi/\lambda$, for all i, are allowed, and λ is an integer ($\lambda > 2$).

(1) Rectilinear architecture: when the value of λ is set to 2, the routing direction is $i\pi/2$, which includes 0° and 90°.
(2) Octilinear (X) architecture: when the value of λ is set to 4, the routing direction becomes $i\pi/4$, which includes 0°, 45°, 90°, and 135°.

Definition 3.2 (*Pseudo Steiner Point*). For convenience, we assume that the endpoints except for the pins are collectively referred to as pseudo Steiner points.

Definition 3.3 (*Half-Perimeter Wirelength*). For a rectangular b with length l_1 and width l_2, the half-perimeter wirelength (HPWL) of b is $l_1 + l_2$.

Definition 3.4 (*Choice 0*). In Figure 3(a), let $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ be two endpoints of a line segment l, where $x_1 < x_2$. Choice 0 of the pseudo Steiner point

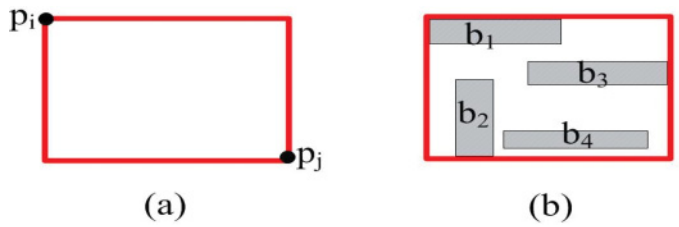

Fig. 2. An example of rectilinear bounding box. (a) Rectilinear bounding box of two points. (b) Rectilinear bounding box of a group of obstacles.

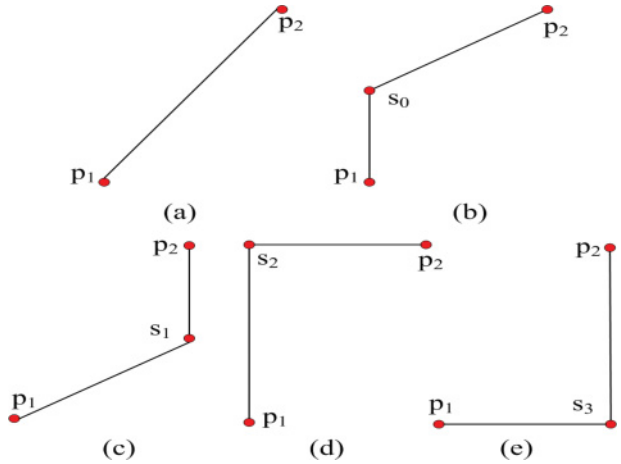

Fig. 3. Four choices of pseudo Steiner point. (a) Line segment. (b) Choice 0. (c) Choice 1. (d) Choice 2. (e) Choice 3.

corresponds to edge l and is given in Figure 3(b). It first leads to the rectilinear side from p_1 to pseudo Steiner point s_0 and then leads the nonrectilinear side to p_2.

Definition 3.5 (*Choice 1*). Choice 1 of the pseudo Steiner point corresponds to edge l as shown in Figure 3(c). It first leads the nonrectilinear side from p_1 to pseudo Steiner point s_1 and then the leads rectilinear side to p_2.

Definition 3.6 (*Choice 2*). Choice 2 of the pseudo Steiner point corresponds to edge l as shown in Figure 3(d). It first leads the vertical side from p_1 to pseudo Steiner point s_2 and then leads the horizontal side to p_2.

Definition 3.7 (*Choice 3*). Choice 3 of the pseudo Steiner point corresponds to edge l as shown in Figure 3(e). It first leads the horizontal side from p_1 to pseudo Steiner point s_3 and then leads the vertical side to p_2.

Definition 3.8 (*Rectilinear Bounding Box*). The rectilinear bounding box of two points p_i and p_j is a rectangular formed by p_i and p_j. The rectilinear bounding box of a group of obstacles is the smallest rectangle that completely contains all obstacles. Figures 2(a) and 2(b) show two examples, respectively.

OAOSMT problem: Given a set of pin vertices and a set of obstacles, a Steiner tree is constructed that connects all the given pins with 0°, 45°, 90°, and 135° wires, and avoids all the given obstacles so that the length of the tree is minimized.

Tables I and II summarize the main notations and abbreviations in this article.

Table I. List of Notations

Notation	Description
n	The number of pin vertices
m	The number of obstacles
p_i	The serial number of the i^{th} terminal
b_i	The serial number of the i^{th} obstacle
$p_i p_j k$	An octilinear edge of terminal p_i and p_j when using routing choice k
$p_i s_k$	The first line segment of $p_i p_j k$
$s_k p_j$	The second line segment of $p_i p_j k$
$rec(p_i, p_j)$	The wirelength of $p_i p_j 2$ or $p_i p_j 3$
$oct(p_i, p_j)$	The wirelength of $p_i p_j 0$ or $p_i p_j 1$
$dis(p_i, p_j)$	The Euclidean distance between terminal p_i and p_j
$\{B_{ijk}\}$	The obstacle set that $p_i p_j k$ runs through
$box(t)$	The rectilinear bounding box of t
$sp(t)$	The HPWL of t
L_{ijk}	The HPWL of the $box(\{B_{ijk}\})$
$len(p_i, p_j)$	The obstacle-avoiding wirelength between terminal p_i and p_j
os_i	The optimal structure for terminal p_i's interconnection
se_i	The length of shared edges of os_i

Table II. List of Abbreviations

Abbreviation	Description
SMT	Steiner minimum tree
OST	Octilinear Steiner tree
OSMT	Octilinear Steiner minimal tree
RSMT	Rectilinear Steiner minimal tree
OAOST	Obstacle-avoiding octilinear Steiner tree
OAOSMT	Obstacle-avoiding octilinear Steiner minimal tree
OARSMT	Obstacle-avoiding rectilinear Steiner minimal tree
OFEMST	Obstacle-free Euclidean minimal spanning tree
DT	Delaunay triangulation
MST	Minimal spanning tree
HPWL	Half-perimeter wirelength
EOT	Edge-obstacle table
EST	Edge-segment table
TDST	Two-dimensional segment tree

4. DETAILS OF THE PROPOSED ALGORITHM

The input pins and obstacles are shown in Figure 4(a). There are four pin vertices and four obstacles. Our algorithm can be divided into the following four steps:

Step 1: OFEMST construction. In this step, an OFEMST is used to construct the initial skeleton of the octilinear Steiner tree (OST) since it can be easily generated and converted to an OST. We first construct a DT [Fortune 1987] on the given pin vertices (Figure 4(b)). Then we employ Kruskal's algorithm to generate an OFEMST (Figure 4(c)).

Step 2: Lookup table generation. In this step, two lookup tables are generated, which record the connection information about the OFEMST's edge. Both tables can provide information support for the whole algorithm.

Step 3: Obstacle-avoiding strategy. In this step, OFEMST is first converted to an OST based on table lookup (Figure 4(d)). Then some points on the routing plane are selected as pseudo Steiner points to generate an OAOST. In Figure 4(e), s_1 and c_1 are

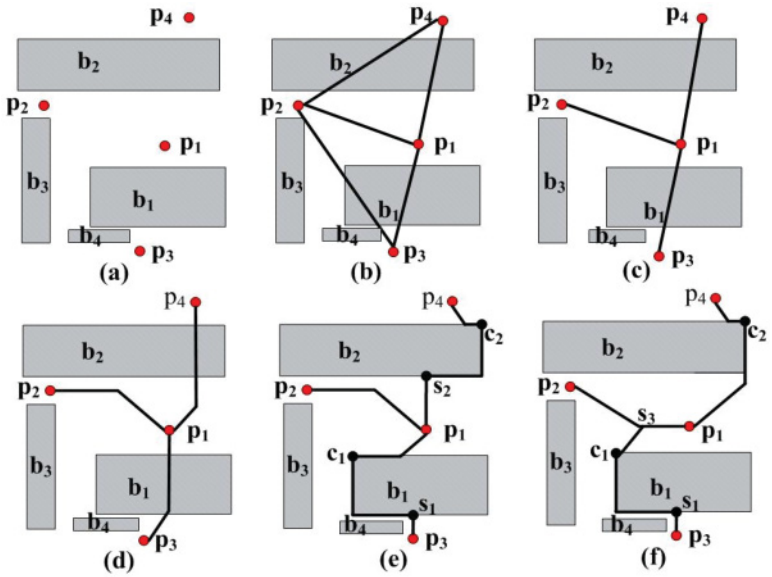

Fig. 4. Framework. (a) P&O. (b) DT. (c) OFEMST. (d) OST. (e) OAOST. (f) OAOSMT.

selected as two pseudo Steiner points between p_1 and p_3, and s_2 and c_2 are selected as two pseudo Steiner points between p_1 and p_4.

Step 4: Refinement. In this step, a refinement method that includes three strategies, namely, redundant pseudo-Steiner elimination, pseudo-Steiner connection optimization, and routing choice optimization, is employed to generate the final OAOSMT. For example, by sharing the routing path between s_3 and p_1 and removing the redundant pseudo Steiner point s_2, the OAOST in Figure 4(e) is converted to an OAOSMT in Figure 4(f).

Figure 5 shows the flow of our algorithm. In this section, the details of the proposed algorithm are given in order according to the framework.

4.1. OFEMST Construction

An OFEMST is usually used to construct the initial skeleton of an OST in an obstacle-free plane since it can be easily generated and converted to an OST [Liu et al. 2015; Huang et al. 2015], so we study the OFEMST construction problem first. There are many existing OFEMST construction algorithms. Jing et al. [2007] used an obstacle-avoiding constrained DT (OACDT) to construct an obstacle-avoiding MST (OAMST) since OACDT contains at least one OAMST and has only $O(n)$ candidate edges. Therefore, in this article, we first construct a DT based on the given pins. Then either Prim's or Kruskal's algorithm can be used to construct an OFEMST in $O(n)$ time. Because the sweepline algorithm of Fortune [1987] is a very mature technology for Voronoi diagram (VD) construction, we directly employ it in this step to construct a VD, and then a DT can be generated by converting VD to its dual graph.

The time complexity of the sweepline algorithm is $O(nlogn)$. In addition, since a DT only contains $O(n)$ edges, the final complexity of OFEMST generation is also $O(nlogn)$.

4.2. Lookup Table Generation

We calculate the connection information of each OFEMST edge in an octilinear plane. Two precomputed lookup tables are generated in this step. The first one is called an

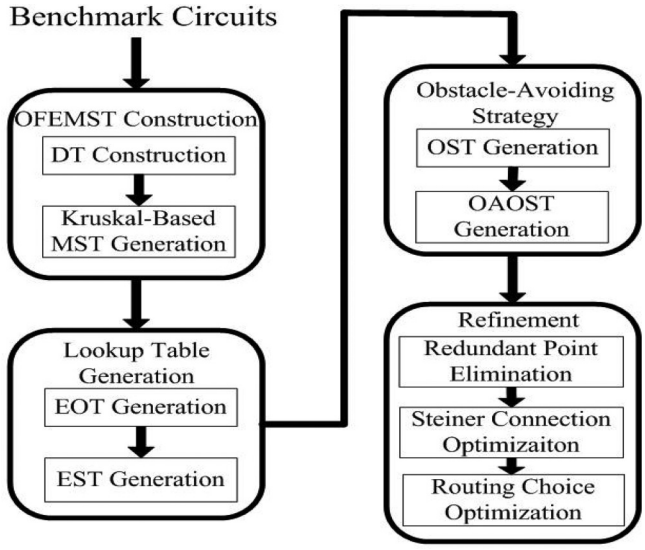

Fig. 5. The flow of FH-OAOS.

ALGORITHM 1: Lookup Table Generation

Input: OFEMST.
Output: EOT, EST.
begin
  Construct a TDST for all obstacles;
  for *each edge* $p_i p_j$ *of OFEMST in routing choice* k **do**
    for *each active obstacle* b *of* $p_i p_j$ **do**
      if $run\_through(p_i s_k, b) = True$ *OR* $run\_through(s_k p_j, b) = True$ **then**
        Add b to $\{B_{ijk}\}$;
      end
    end
    $L_{ijk} = sp(box(\{B_{ijk}\}))$;
    if $p_i s_k = 45^\circ$ *or* $p_i s_k = 135^\circ$ **then**
      $Rotate(p_i s_k, 45^\circ)$;
    end
    if $s_k p_j = 45^\circ$ *or* $s_k p_j = 135^\circ$ **then**
      $Rotate(s_k p_j, 45^\circ)$;
    end
    record the coordinates of $p_i s_k$ and $s_k p_j$;
  end
end

edge-obstacle table (EOT), which mainly records obstacles that each octilinear edge runs through. The other is called an edge-segment table (EST), which records the coordinates of two line segments for each octilinear edge.

Since OFEMST has n-1 edges, and each edge has four routing choices, there are 4*(n-1) octilinear edges in total. For each octilinear edge $p_i p_j k$, we record obstacles that $p_i p_j k$ runs through as a set $\{B_{ijk}\}$. Meanwhile, we compute the rectilinear bounding box of each $\{B_{ijk}\}$ and set the L_{ijk} value to the HPWL of this bounding box. All 4*(n-1) groups of $\{B_{ijk}\}$ and L_{ijk} constitute the final EOT. In addition, according to Definitions 3.4 through 3.7, each $p_i p_j k$ consists of two line segments ($p_i s_k$ and $s_k p_j$). We record the

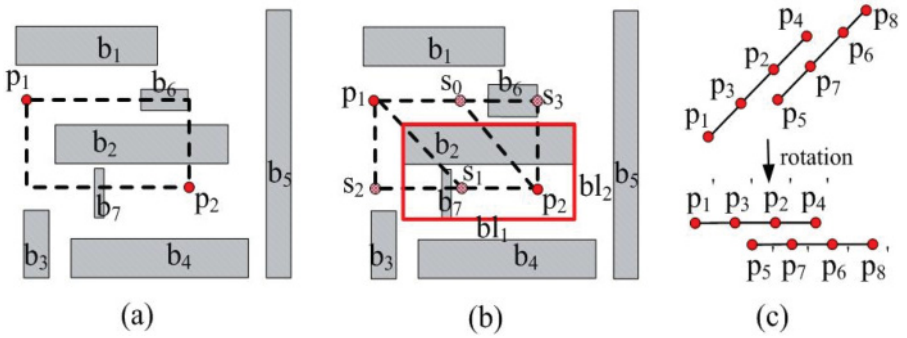

Fig. 6. Lookup table generation. (a) An example of "active obstacle." (b) An example of table generation. (c) Segment rotation.

coordinates of both two segments. Note, for a 45° and 135° line segment, we rotate it 45° clockwise around the origin, then record the coordinates of the new segment. This information of all 4*(n-1) octilinear edges constitutes the final EST. The pseudocode of table generation is shown in Algorithm 1. The detailed steps are stated as follows:

(1) Initialize $t = 1$.
(2) Inspect the t^{th} edge $p_i p_j$ of OFEMST. The starting coordinates and ending coordinates of two line segments $p_i s_k$ and $s_k p_j$ are calculated for each routing choice k.
(3) For each active obstacle b_p ($0 < p < m$), if $p_i s_k$ or $s_k p_j$ runs through b_p, b_p is added to corresponding set $\{B_{ijk}\}$.
(4) Calculate L_{ijk} of each $\{B_{ijk}\}$. Then record both L_{ijk} and $\{B_{ijk}\}$ into EOT.
(5) Inspect each $p_i s_k$ and $s_k p_j$; if it is a 45° or 135° segment, rotate it 45° clockwise around the origin to form a new horizontal or vertical segment.
(6) Record the coordinates of each $p_i s_k$ and $s_k p_j$ into EST. Then $t = t+1$; if $t < n$, return (2). Otherwise, exit the process.

There are two points to note. First, during step (3), for an obstacle b_p, we inspect it only when b_p overlaps with the rectilinear bounding box of $p_i p_j$. This obstacle is also called an "active obstacle." For example, as shown in Figure 6(a), there are only two pin vertices p_1 and p_2 for edge $p_1 p_2$; b_2, b_6, and b_7 are three active obstacles; and the others do not need to be inspected because it is impossible to intersect with any routing choice of edge $p_1 p_2$. Since a two-dimensional segment tree (TDST) [De et al. 2000] is often used to record some attribute information of the plane region, in order to find the active obstacle set for a given pair of pins in an efficient way, a TDST is employed to organize all obstacles. More specifically, the outer segment tree of TDST is used to organize the intervals on the horizontal axis, and the inner segment tree is used to organize the intervals on the vertical axis. In addition, according to the problem model described in Section 3, each obstacle b_p has a unique serial number p; thus, for any one region of a constructed TDST, if the region is covered by an obstacle b_p, then we add the serial number p to this region. In this way, after all obstacles are added to the corresponding regions of TDST, the final TDST will contain all information about obstacles on the whole routing plane, and when given a pair of pin vertices p_i and p_j, we can quickly find all active obstacles by querying the regions that are covered by the rectilinear bounding box of $p_i p_j$. Note, once the TDST is constructed, it does not need to update, and can provide an active obstacle inquiry for all subsequent steps of the algorithm. Since a segment tree is usually a complete binary tree, each operation can be done in logarithmic time, such as insert and query; this is efficient. Second, in step (5), if there exists a 45° or 135° segment, we

Table III. An Example of EOT and EST

EOT	EST
$\{B_{120}\} = \{b_2\}$, $L_{120} = sp(b_2)$	$p_1 s_0\{(3, 4), (4, 4)\}$ $s_0 p_2^*\{(4\sqrt{2}, -\sqrt{2}), (4\sqrt{2}, 0)\}$
$\{B_{121}\} = \{b_2, b_7\}$, $L_{121} = sp(box(\{B_{121}\}))$	$p_1 s_1^*\{(7/\sqrt{2}, -\sqrt{2}/2), (7/\sqrt{2}, \sqrt{2})/2\}$ $s_1 p_2\{(4, 3), (5, 3)\}$
$\{B_{122}\} = \{b_7\}$, $L_{122} = sp(b_7)$	$p_1 s_2\{(3, 4), (3, 3)\}$ $s_2 p_2\{(3, 3), (5, 3)\}$
$\{B_{123}\} = \{b_2, b_6\}$, $L_{123} = sp(box(\{B_{123}\}))$	$p_1 s_3\{(3, 4), (5, 4)\}$ $s_3 p_2\{(5, 4), (5, 3)\}$

rotate it 45° clockwise around the origin to form a new horizontal or vertical segment, then record the coordinates of the new segment. This can facilitate the total and local wirelength calculation, because shared 45° and 135° segments can be easily identified without wirelength changes. For example, as shown in Figure 6(c), there are four 45° segments p_1p_2, p_3p_4, p_5p_6, and p_7p_8. When calculating the wirelength, one needs to simultaneously consider the position of all four 45° segments, and path p_2p_3 and p_6p_7 should be calculated only once since both are shared paths. Since the input scale of real circuits may be very large, there may exist quite a lot of 45° segments in the routing plane, and relatively more computing quantity is needed to obtain those shared paths (it can reach quadratic complexity in the worst case if we consider the position of all possible pairs of 45° segments). However, if we rotate all 45° segments 45° clockwise around the origin in advance, all shared paths can be easily calculated, because all generated segments are horizontal (Figure 6(c)), and if two 45° segments have a shared path, the two generated horizontal segments must have the same y-coordinates; in this way, we can first sort these rotated segments according to its y-coordinates, and then the length of the shared path can be calculated only by scanning the sorted segments once (sort process dominates this process; it can be performed in $O(eloge)$ time, where e is the number of 45° segments). Rotation for 135° segments has a similar principle.

$$\begin{cases} y_{new} = \sqrt{2}/2 * (y_i - x_i) \\ x_{new1} = \sqrt{2}/2 * (x_i + y_i) \\ x_{new2} = \sqrt{2}/2 * (x_j + y_j) \end{cases} \tag{1}$$

$$\begin{cases} x_{new} = \sqrt{2}/2 * (x_i + y_i) \\ y_{new1} = \sqrt{2}/2 * (y_i - x_i) \\ y_{new2} = \sqrt{2}/2 * (y_j - x_j) \end{cases} \tag{2}$$

We assume that $v_i(x_i, y_i)$ and $v_j(x_j, y_j)$ are two endpoints of a 45° or 135° segment. Equation (1) is used to rotate a 45° segment to a horizontal segment, where y_{new} is the y-coordinate of the new segment, and x_{new1} and x_{new2} are the x-coordinates of two new endpoints, respectively. Similarly, Equation (2) is used to rotate a 135° segment to a vertical segment, where x_{new} is the x-coordinate of the new segment, and y_{new1} and y_{new2} are the y-coordinates of two new endpoints, respectively.

For EOT and EST generation, we take Figure 6(b) as a simple example. Without loss of generality, we assume that there are two pins p_1 (3,4) and p_2 (5,3) and seven obstacles (b_1—b_7) in an octilinear routing plane. Obviously, in this example, the OFEMST only contains one edge p_1p_2. Table III shows the generated EOT and EST. All connection information about OFEMST can be inquired quickly. For example, it can be seen that the octilinear edge p_1p_21 runs through obstacles b_2 and b_7. The coordinates of two

ALGORITHM 2: OST Generation

Input: OFEMST, EOT.
Output: OST.
begin
  for *each edge* $p_i p_j$ *of OFEMST* **do**
    if $\{B_{ij0}\} = \emptyset$ **then**
      $p_i p_j k = p_i p_j 0$;
    end
    else if $\{B_{ij1}\} = \emptyset$ **then**
      $p_i p_j k = p_i p_j 1$;
    end
    else if $\{B_{ij2}\} = \emptyset$ *OR* $\{B_{ij3}\} = \emptyset$ **then**
      if $L_{ij0} <= L_{ij1}$ **then**
        $p_i p_j k = p_i p_j 0$;
      end
      else
        $p_i p_j k = p_i p_j 1$;
      end
    end
    else
      select_min(L_{ijt}), t=0,1,2,3;
      $p_i p_j k = p_i p_j t$;
    end
  end
end

segments $p_1 s_1$ and $s_1 p_2$ are $\{(7/\sqrt{2}, -\sqrt{2}/2), (7/\sqrt{2}, \sqrt{2})/2\}$ and $\{(4, 3), (5, 3)\}$, respectively. It should be noted that $p_1 s_1^*$ means the coordinates of $p_1 s_1$ are the rotated vertical segment. In addition, if an octilinear edge $p_i p_j k$ can avoid all the obstacles ($\{B_{ijk}\} = \emptyset$), then $L_{ijk} = 0$. If $p_i p_j k$ runs through only one obstacle b ($\{B_{ijk}\} = \{b\}$), L_{ijk} is directly set to the HPWL of b. Otherwise, if $p_i p_j k$ runs through more than one obstacle, for example, $p_1 p_2 1$ runs through b_2 and b_7 in Figure 6(b) ($\{B_{ijk}\} = \{b_2, b_7\}$), then L_{121} is set to the HPWL of $box(\{B_{121}\})$, which is equal to $bl_1 + bl_2$, as shown in Figure 6(b) with a red line.

The functions of two lookup tables can be summarized as follows:

1. In the obstacle-avoiding strategy, the conversion process (OFEMST $\rightarrow$ OST $\rightarrow$ OAOST) can be efficiently performed based on the information provided by EOT.
2. During the refinement process, the optimal structure of each terminal can be efficiently calculated based on EOT and EST. In addition, all connection information regarding redundant the pseudo Steiner points elimination and pseudo-Steiner connection optimization process needed can be obtained by looking up tables.
3. Local and global wirelength can be efficiently calculated with EST.

4.3. Obstacle-Avoiding Strategy

4.3.1. OST Generation. In this part, the OFEMST generated in step 1 is converted to an OST based on looking up EOT.

We inspect each edge $p_i p_j$ of OFEMST. If octilinear edge $p_i p_j 0$ or $p_i p_j 1$ can avoid all obstacles, we directly set the routing choice of $p_i p_j$ as choice 0 or choice 1. Otherwise, if either $p_i p_j 2$ or $p_i p_j 3$ can avoid all obstacles, we still select choice 0 or choice 1 as a result according to the value of L_{ij0} and L_{ij1}. If all four routing choices run through obstacles, we select the one with the smallest L_{ijk}. It should be noted that all judgments can be

performed by directly looking up EOT. Thus, this process is very fast. The pseudocode of OST generation is shown in Algorithm 2. The detailed steps are stated as follows:

(1) Initialize $t = 1$.
(2) For the t^{th} edge $p_i p_j$ of OFEMST, inspect choice 0 and choice 1; if either of them can avoid all obstacles, then set the routing choice of $p_i p_j$ as the current one. Repeat this step to inspect the $(t+1)^{th}$ edge until $t + 1 >= n$. Otherwise, go to step (3).
(3) If either choice 2 or choice 3 can avoid all obstacles, go to step (4). Otherwise, go to step (5).
(4) If $L_{ij0} < L_{ij1}$, set the routing choice of $p_i p_j$ as choice 0. Otherwise, set it as choice 1. Then $t = t + 1$; if $t < n$, go to step (2). Otherwise, exit process.
(5) Elect the smallest L_{ijk} ($k = 0, 1, 2, 3$), and set the routing choice of $p_i p_j$ as choice k. Then $t = t + 1$; if $t < n$, go to step (2). Otherwise, exit process.

Step (2) adopts a strategy that choice 0 and choice 1 are preferred in the OST generation process (Theorem 4.1). In addition, step (3) and step (4) mean that even if choice 2 or choice 3 can avoid all obstacles, we still select choice 0 or choice 1 as a result (Theorem 4.4). Although it may introduce some redundant points according to our OAOST generation technique in Section 4.3.2, it does not matter because of the function of refinement.

THEOREM 4.1. *For any two terminals p_i and p_j, $oct(p_i, p_j)$ can reduce wirelength by $(2 - \sqrt{2}) * Min(\Delta x, \Delta y)$ compared with $rec(p_i, p_j)$.*

PROOF. We assume that $\Delta x < \Delta y$. Then $oct(p_i, p_j) = \Delta y - \Delta x + \sqrt{2} * \Delta x$, and $rec(p_i, p_j) = \Delta x + \Delta y$. Thus, $rec(p_i, p_j) - oct(p_i, p_j) = (2 - \sqrt{2}) * \Delta x$. For the case that $\Delta x > \Delta y$, a similar method can prove the result. □

LEMMA 4.2. *Terminal c is an intermediate node of edge $p_i p_j$, and then $rec(p_i, p_j) = rec(p_i, c) + rec(c, p_j)$ if and only if $c \subset box(p_i p_j)$.*

LEMMA 4.3. *Terminal c is an intermediate node of edge $p_i p_j$, and then $oct(p_i, p_j) = oct(p_i, c) + oct(c, p_j)$ if and only if $c \subset para(p_i p_j)$, where $para(p_i p_j)$ is the parallelogram formed by p_i and p_j with 45° or 135° direction.*

According to the parallel relationship among segments formed by p_i, c and c, p_j, the equation in Lemmas 4.2 and 4.3 can be achieved easily. Thus, proofs are omitted here.

THEOREM 4.4. *For an edge $p_i p_j$, if $\{B_{ij0}\} \neq \emptyset$ and $\{B_{ij1}\} \neq \emptyset$, but $\{B_{ij2}\} = \emptyset$ or $\{B_{ij3}\} = \emptyset$, selecting some pseudo Steiner points on the boundary of $box(\{B_{ij0}\})$ or $box(\{B_{ij1}\})$ as intermediate nodes can reduce wirelength by $[0, (2 - \sqrt{2}) * Min(\Delta x, \Delta y))$.*

PROOF. As shown in Figure 7(a), $\{B_{ij0}\} = \{B_{ij0}\} = \{b_1, b_2\}$, but $\{B_{ij2}\} = \emptyset$. However, as shown in Figure 7(b), if we select a pseudo Steiner point c as an intermediate node, we will have $oct(p_i, c) <= rec(p_i, c)$ and $oct(c, p_j) <= rec(c, p_j)$, and the equality holds up if and only if p_i, c and c, p_j are collinear, respectively. According to Lemma 4.2, if $c \subset box(p_i p_j)$, then $rec(p_i, c) + rec(c, p_j) = rec(p_i, p_j)$. Thus, we have $oct(p_i, c) + oct(c, p_j) <= rec(p_i, p_j)$. At best, $(2 - \sqrt{2}) * Min(\Delta x, \Delta y) * (oct(p_i, c) + oct(c, p_j)) = rec(p_i, p_j)$ according to Theorem 4.1. Because formula $c \subset para(p_i p_j)$ is not valid otherwise, $p_i p_j$ can be directly connected by choice 0 or choice 1; thus, the right boundary of result cannot be reached. Note, at worst, that both $p_i c$ and $c p_j$ can avoid all obstacles by only using choice 2 or choice 3, and then the wirelength reduction is zero. □

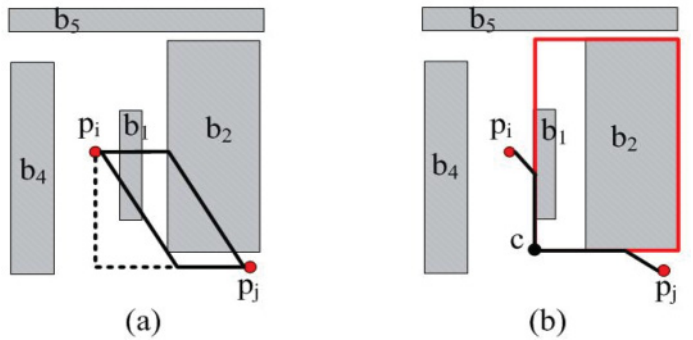

Fig. 7. Illustration of Lemma 4.4. (a) The original graph. (b) The graph after selecting pseudo Steiner point.

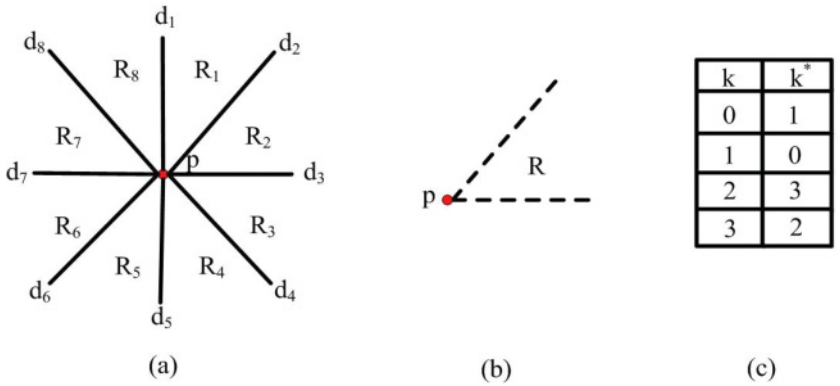

Fig. 8. (a) Octilinear partition. (b) Boundary of each region. (c) Routing choice chart.

4.3.2. OAOST Generation. Obviously, some edges of OST may run through obstacles according to the OST generation method. In fact, these edges can be identified by directly looking up EOT. But it may need much more technical support so as to avoid obstacles. Thus, we propose a pseudo Steiner point selection method, which can efficiently help all OST edges avoid all obstacles.

Similar to the literature [Zhu et al. 2005], we divide the plane into eight octilinear regions with reference to each point p as shown in Figure 8(a); each region does not include its two bounding half lines as shown in Figure 8(b). It is obvious that each bounding line ($d_1 - d_8$) in Figure 8(a) is a legal routing path of point p in an octilinear routing plane.

Before selecting pseudo Steiner points, we first scan all edges of OST once, for each edge $p_i p_j k$ (k is an exact routing choice in OST); if $x_i > x_j$, then we change the position of point p_i and p_j, namely, transform $p_i p_j k$ to $p_j p_i k^*$, where the corresponding k^* value is shown in Figure 8(c). In this way, for each OST edge, we can ensure that the start point locates to the left side of the endpoint, and thus simplify the possible scenarios of OST edges. After the transformation, when we inspect an OST edge $p_i p_j k$, we only need to consider the right half axis of point p_i's octilinear partition and the left half axis of point p_j's octilinear partition.

Next, we inspect each octilinear edge $p_i p_j k$ of transformed OST; if it runs through obstacles ($\{B_{ijk}\} \neq \emptyset$), we directly delete octilinear edge $p_i p_j k$ and compute the rectilinear bounding box bx of $\{B_{ijk}\}$. Then we project p_i to its right half axis along five octilinear directions; these directions include $d_1 - d_5$ in Figure 8(a). We assume that the first

intersection between $p_i d_x$ and bx is t_x ($x = 1, 2, 3, 4, 5$); it should be noted that t_x may not exist because $p_i d_x$ does not intersect with bx. In addition, we calculate the angle α_x between straight line $p_i d_x$ and $p_i p_j$, and then select t_x in existence with the smallest α_x as a pseudo Steiner point s. Because s might not be able to be directly connected to p_j, we need to select more pseudo Steiner points so p_i and p_j can be connected. To achieve this goal, we inspect each corner point c_y ($y = 1, 2, 3, 4$) of bx and select the one such that $rec(s, c_y) + rec(c_y, p_j)$ is minimum as the other pseudo Steiner point c. Note, there may exist two corner points of bx such that $rec(s, c_y) + rec(c_y, p_j)$ is minimum; for this case, we select the one with larger $rec(s, c_y)$. Finally, we compute the connection information of edge $p_i s$, sc, and cp_j and add this information into EOT and EST. At this point, we can generate the octilinear edge $p_i sk_1$, sck_2, and $cp_j k_3$ by directly looking up the table.

Considering that the obstacles of a set $\{B_{ijk}\}$ may be scattered widely in the routing plane, in this case, the new generated routing path between p_i and p_j may be long since it detours around the rectilinear bounding box of $\{B_{ijk}\}$. Thus, in order to ensure high routing quality, we employ the other pseudo Steiner point selection strategy in the following two cases. (1) The routing path between two selected pseudo Steiner points runs through the obstacles. (2) We set a parameter β if we divide the length of the new generated path between p_i and p_j by $rec(p_i, p_j)$ and the obtained value is bigger than β (β is set to 2 in our experiments). Then, we sort all obstacles of $\{B_{ijk}\}$ in increasing order according to the distance between p_i and the obstacle, and select pseudo Steiner points on each obstacle in order according to the aforementioned projection method. In this way, an obstacle-avoiding path that can escape from the large bounding box of $\{B_{ijk}\}$ is generated.

The detailed steps of the OAOST generation technique are stated as follows:

(1) For each octilinear edge $p_i p_j k$ of OST, if $x_i > x_j$, transform $p_i p_j k$ to $p_j p_i k^*$.
(2) Initialize $t = 1$.
(3) For the $t-th$ octilinear edge $p_i p_j k$ of transformed OST, if it runs through obstacles, go to step (4). Otherwise, repeat this step to inspect the $(t + 1) - th$ edge until $t + 1 >= n$.
(4) Delete edge $p_i p_j k$. Compute $bx = box(\{B_{ijk}\})$. Project p_i along $d_1 - d_5$ and compute the first intersection t_x between each $p_i d_x$ ($x = 1, 2, 3, 4, 5$) and bx.
(5) Compute the angle α_x between each $p_i d_x$ and straight line $p_i p_j$. Then select a t_x with the smallest α_x as a pseudo Steiner point s.
(6) Select a corner point c from bx such that $rec(s, c_y) + rec(c_y, p_j)$ is minimum, where c_y is the y^{th} corner point of bx, y = 1, 2, 3, 4.
(7) Calculate the connection information of edge $p_i s$, sc, and cp_j. Generate octilinear paths $p_i s$, sc, and cp_j.
(8) If $(len(p_i, s) + len(s, c) + len(c, p_j))/rec(p_i, p_j) > \beta$ or octilinear path sc runs through obstacles, then sort $\{B_{ijk}\}$ and select pseudo Steiner points on each obstacle in order, and generate a new path from p_i to p_j.

LEMMA 4.5. *For an edge $p_i p_j k$ of transformed OST, if $\{B_{ijk}\} \neq \emptyset$, then $projection(p_i) \cap box(\{B_{ijk}\}) \neq \emptyset$ or $projection(p_j) \cap box(\{B_{ijk}\}) \neq \emptyset$, where $projection(p_i)$ and $projection(p_j)$ are the projection of p_i and p_j on its half axis along five directions, respectively.*

PROOF. We use reductio ad absurdum to prove this lemma. Figures 9(a) and 9(b) show the half axis of p_i and p_j according to our definition, respectively. If $projection(p_i) \cap box(\{B_{ijk}\}) = \emptyset$, then $box(\{B_{ijk}\})$ must completely locate in one region of p_i's right half axis. Similarly, $box(\{B_{ijk}\})$ must completely locate in one region of p_j's left half axis if $projection(p_j) \cap box(\{B_{ijk}\}) = \emptyset$. Thus, if both $projection(p_i) \cap box(\{B_{ijk}\}) = \emptyset$ and

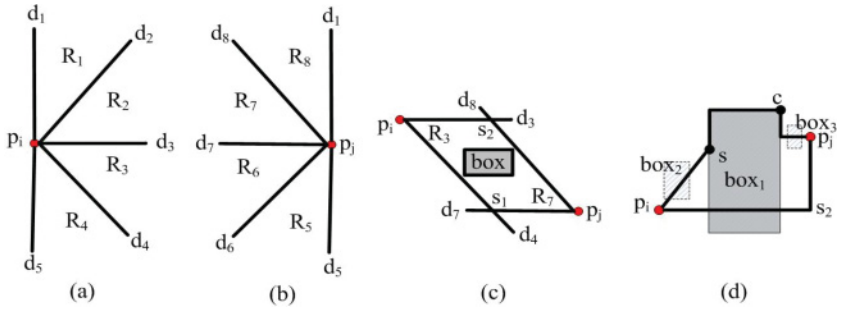

Fig. 9. Illustration of theory. (a) Half axis of p_i. (b) Half axis of p_j. (c) The formed legal path of two projections. (d) Bounding box of new paths.

$projection(p_j) \cap box(\{B_{ijk}\}) = \emptyset$ are true, without loss of generality, we assume that $box(\{B_{ijk}\}) \subset R_i$ ($R_i \in \{R_1, R_2, R_3, R_4\}$ of p_i) and $box(\{B_{ijk}\}) \subset R_j$ ($R_j \in \{R_5, R_6, R_7, R_8\}$ of p_j), and then the two boundary lines of R_i and R_j must form a legal octilinear routing path, which means that p_i and p_j can be directly connected without running though any obstacle, a paradox. For example, as shown in Figure 9(c), $box(\{B_{ijk}\}) \subset R_3$ of p_i and $box(\{B_{ijk}\}) \subset R_7$ of p_j, and then p_i and p_j can be connected by path $p_i s_1 p_j$ or $p_i s_2 p_j$. □

We have pointed that t_x may not exist because $p_i d_x$ does not intersect with bx in step (4); thus, there may exist the scenario that all five intersection points t_x do not exist. In that case, according to Lemma 4.5, we will project p_j to its left half axis along five octilinear directions $(d_1, d_5 - d_8)$ in Figure 8(a). In addition, since there are other obstacles that we do not consider, the new generated path between p_i and p_j may still run through obstacles; thus, this OAOST generation technique may be operated several times until all octilinear edges avoid all obstacles. Fortunately, when a new generated octilinear path runs through obstacles, at least two boundaries of the rectilinear bounding box of these obstacles are usually restricted to a certain range. For example, as shown in Figure 9(d), p_i and p_j are connected by choice 3, and it runs through box_1, and then s and c are selected as two pseudo Steiner points. Since we have ensured that the path between s and c can avoid all obstacles, only paths $p_i s$ and $c p_j$ may still run through obstacles. Assume that box_2 is the rectilinear bounding box of obstacles that octilinear path $p_i s$ runs through; then the bottom side of box_2 must locate above $p_i s_2$, and the right side of box_2 is restricted by the right side of box_1. box_3 is also restricted by $p_i s_2$ and box_1; obviously, this case can be solved in constant time. The other three choices follow a similar principle. Similarly, when we select pseudo Steiner points on each obstacle, the bounding box of a new generated octilinear path is also restricted in at least two directions. It also can be solved in constant time. It should be noted that this OAOST generation process also may generate redundant pseudo Steiner points. The complete pseudo code of OAOST generation is shown in Algorithm 3.

We take Figure 10 as a simple example to further explain this process. Figure 10(a) is the original routing graph, where $p_1 p_2 0$ runs through obstacles b_1, b_2, and b_3 (namely, $\{B_{120}\} = \{b_1, b_2, b_3\}$). After edge $p_1 p_2 0$ is deleted, we first compute $box(\{B_{120}\})$; the result bx is shown with a red line in Figure 10(b). Then we project p_1 along its five directions; it is obvious that $p_1 d_1$ and $p_1 d_5$ do not intersect with bx, and thus, $T_1 = \{t_2, t_3, t_4\}$. Next, we compute the angle between straight line $p_1 p_2$ and each projection line of p_1, and we have $\alpha_2 = \angle d_2 p_1 p_2$, $\alpha_3 = \angle d_3 p_1 p_2$, and $\alpha_4 = \angle d_4 p_1 p_2$. Because $\alpha_4 < \alpha_3 < \alpha_2$, t_4 is selected as a pseudo Steiner point s. In Figure 10(c), we inspect four corner points of bx; we have $rec(sc_1) + rec(c_1 p_2) = rec(sc_2) + rec(c_2 p_2) < rec(sc_3) + rec(c_3 p_2) = rec(sc_4) + rec(c_4 p_2)$, and because $rec(sc_1) < rec(sc_2)$, c_2 is selected as the other pseudo

ALGORITHM 3: OAOST Generation

Input: OST, EOT.
Output: OAOST.
begin
  Generate the transformed OST;
  while *there exists edges run through obstacle* **do**
    for *each octilinear edge* $p_i p_j k$ *with* $\{B_{ijk}\} \neq \emptyset$ **do**
      Initialize $mark1 = False$ and $mark2 = False$;
      delete octilinear edge $p_i p_j k$;
      $bx = box(\{B_{ijk}\})$;
      project point p_i to its right half axis along $d_1 - d_5$;
      $T_1 = \{t_1, t_2, t_3, t_4, t_5\} = \{p_i d_1 \cap bx,\ p_i d_2 \cap bx,\ p_i d_3 \cap bx,\ p_i d_4 \cap bx,\ p_i d_5 \cap bx\}$;
      if $T_1 \neq \emptyset$ **then**
        $mark1 = True$;
      end
      if $mark1 = False$ **then**
        project point p_j to its left half axis along $d_1, d_5 - d_8$;
        $T_2 = \{t_1, t_5, t_6, t_7, t_8\} = \{p_j d_1 \cap bx,\ p_j d_5 \cap bx,\ p_j d_6 \cap bx,\ p_j d_7 \cap bx,\ p_j d_8 \cap bx\}$;
        $mark2 = True$;
      end
      if $mark1 = True$ **then**
        $\theta = \{\alpha_x\} =$ {the angle between $p_i d_x$ and $p_i p_j$}, $x = 1, 2, 3, 4, 5$;
        for *each* α_x *in* θ **do**
          select t_x exists in T_1 with the smallest α_x;
        end
        $s =$ the selected t_x;
        for *each corner point* c_y *(*$y = 1, 2, 3, 4$*) of* bx **do**
          $c =$ select the one such that $rec(s, c_y) + rec(c_y, p_j)$ is minimum;
        end
      end
      else
        $\theta = \{\alpha_x\} =$ {the angle between $p_j d_x$ and $p_i p_j$}, $x = 1, 5, 6, 7, 8$;
        for *each* α_x *in* θ **do**
          select t_x exists in T_2 with the smallest α_x;
        end
        $s =$ the selected t_x;
        for *each corner point* c_y *(*$y = 1, 2, 3, 4$*) of* bx **do**
          $c =$ select the one such that $rec(s, c_y) + rec(c_y, p_i)$ is minimum;
        end
      end
      Generate octilinear path $p_i s$, sc, and cp_j;
      if $(len(p_i, s) + len(s, c) + len(c, p_j)))/rec(p_i, p_j) > \beta$ *or path* sc *is not illegal* **then**
        Delete $p_i s$, sc, and c, p_j, and sort all obstacles of $\{B_{ijk}\}$;
        Generate a path by selecting pseudo Steiner points on each obstacle in order;
      end
    end
  end
end

Steiner point. Finally, we calculate the connection information of edge $p_1 s$, sc_2, and $c_2 p_2$ and add this information to EOT and EST. The final octilinear edge can be generated by table lookup (Figure 10(d)). Note, the generated OAOST may include some routing paths that also have space to further optimize.

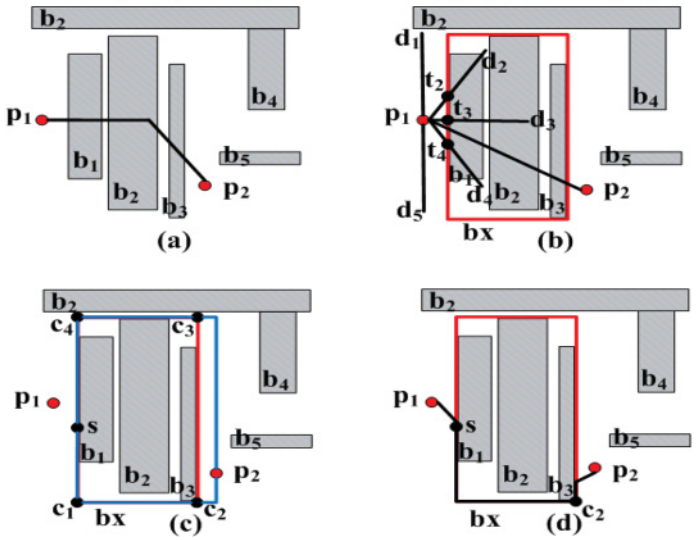

Fig. 10. An example of OAOST generation. (a) Original routing graph. (b) The first pseudo Steiner point selection. (c) The second pseudo Steiner point selection. (d) New routing path.

4.4. Refinement

Since the generated OAOST may still include some suboptimal routing paths, in this part, three refinement strategies are proposed to further reduce wirelength.

4.4.1. Redundant Pseudo-Steiner Elimination. During the process of OAOST generation, some redundant pseudo Steiner points may be introduced according to our OAOST generation method. For example, as shown in Figure 4(e), for octilinear edge $p_1p_4$1, we select s_2 and c_2 as two pseudo Steiner points. However, s_2 is redundant because p_1 and c_2 can be directly connected and $len(p_1s_2)+len(s_2c_2) > len(p_1c_2)$; thus, s_2 is deleted, and p_1 is connected to c_2 in Figure 4(f). In order to eliminate these redundant points, we scan each selected pseudo Steiner point s in OAOST and inspect its two octilinear edges p_isk_1 and sp_jk_2; if p_ip_j can be directly connected and $len(p_is) + len(sp_j) >= len(p_ip_j)$, we delete pseudo Steiner point s and the related edge p_isk_1 and sp_jk_2, then connect p_i to p_j. Note, the connection information of the original edges p_isk_1 and sp_jk_2 can be gotten by directly looking up tables except the new edge p_ip_j; thus, this process is also very fast.

4.4.2. Pseudo Steiner Connection Optimization. Although the OAOST generation technique can make the OST edge avoid all the obstacles, the routing path between pseudo Steiner points may not be good enough. For example, the routing path between s and c_2 in Figure 10(d) can be further optimized, because there exists some unused routing resources in the rectilinear bounding box of obstacles. Of course, if an OST edge only runs through one obstacle, or the pseudo Steiner points are selected on each obstacle, the routing path between the selected pseudo Steiner points have no space to optimize. On the other hand, if two selected pseudo Steiner points are collinear, then the optimal routing path is a straight line, and it also cannot be optimized. Therefore, for any pair of pseudo Steiner points, if it can be further optimized, it should meet at least two points: (1) the original OST edge runs through more than one obstacle, and the pseudo Steiner points are selected on the boundary of the rectilinear bounding box of obstacles, and (2) the two selected pseudo Steiner points are not collinear. For this kind of pseudo Steiner point, we use an operation called sliding to try to reduce wirelength. Because

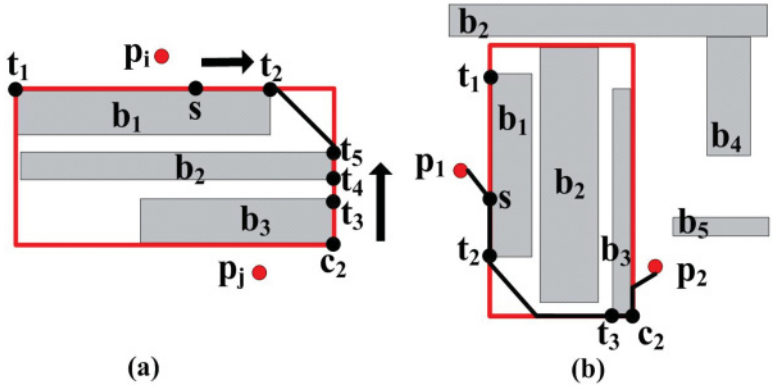

Fig. 11. Pseudo Steiner connection optimization. (a) Sliding operation. (b) Final routing graph of Figure 10(d).

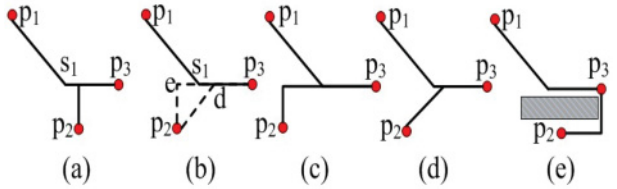

Fig. 12. The optimal structure of a 2-degree terminal.

the two selected pseudo Steiner points are located on two sides of the rectilinear bounding box, there must exist a legal routing path along two sides of this bounding box. In addition, both two sides of the bounding box must be line touched with at least one obstacle. For example, as shown in Figure 11(a), the OST edge between p_i and p_j runs through obstacles b_1, b_2, and b_3; according to the OAOST generation technique, we select point s and corner c_2 as two pseudo Steiner points. In this case, s and c_2 can only be connected by choice 3. Furthermore, b_1, b_2, and b_3 are line touched with a rectilinear box between t_1t_2, t_4t_5, and c_2t_3, respectively. However, if we slide two pseudo Steiner points along the routing path between s and c_2 and stop each one at a line-touched obstacle corner t_i, this t_i is the farthest one from the original pseudo Steiner point. Then these two corners may be connected by choice 0 or choice 1, because the unused routing resources can be used. For example, we slide s to t_2 and c_2 to t_5, and then t_2 and t_5 can form an octilinear edge t_2t_50; thus, wirelength is reduced compared with the previous result. Figure 11(b) shows the routing graph after the result in Figure 10(d) is optimized. Of course, when there exists other obstacles between two selected t_i, or these two t_is overlap with each other, there is also no space to optimize wirelength.

4.4.3. Routing Choice Optimization. Apart from the previous two points, we have not considered an important issue, namely, shared edges. In other words, the obtained routing tree can be further optimized.

FACT. *For any one terminal p_i of OAOST, there is at least one optimal structure for its interconnection.*

We only discuss the 2-degree terminal; other degree terminals follow a similar principle. Since each edge has four routing choices, there are 16 substructures in total for a 2-degree terminal. Figure 12(a) shows a 2-degree terminal p_3 in an obstacle-free plane, where p_1 and p_2 are connected to p_3. Without loss of generality, we assume $\Delta x > \Delta y$

between two terminals. When p_2 is between s_1 and p_3, where s_1 is the pseudo Steiner point of octilinear edge p_1p_31, Figure 12(a) is the only optimal structure for p_3. When p_2 is between p_1 and s_1 (Figure 12(b)), if $p_2e + es_1 < p_2d$ or $es_1 > ed$, Figure 12(c) becomes the optimal structure. Otherwise, Figure 12(d) is the optimal one. Aobstacle-exist plane also follows a similar principle. But the optimal structure may become any one of the 16 substructures because of the existence of the obstacle. For example, as shown in Figure 12(e), p_2 can be connected to p_3 only by choice 3. Thus, Figure 12(e) is the optimal structure when there exists an obstacle. Of course, p_1 also can be connected to p_3 by choice 0 in Figure 12(e). In conclusion, no matter what degree a terminal is, there is at least one optimal structure for its interconnection.

Based on this fact, we propose a terminal-oriented routing choice optimization method. By scanning the edges of OAOST once, we first calculate the degree of each terminal p_i and record the terminals that are connected to p_i as a list. Then we calculate the optimal structure (os_i) for each terminal p_i. Assume that the degree of p_i is d. We enumerate all 4^d routing choice combinations of p_i and select the obstacle-avoiding one with minimal wirelength as os_i. Moreover, we calculate the length of the shared edge (se_i) of each os_i and sort all terminals in decreasing order according to the value of se_i. Finally, we apply each terminal's os_i to the original OAOST according to the sorted terminal list until the routing choice of all OAOST edges have been decided. The detailed steps are stated as follows:

(1) Scan the edges of the OAOST to count the degree of each terminal p_i, and record the terminals that are connected to p_i as a list at the same time.
(2) For each terminal p_i, if the degree of p_i is d, enumerate all 4^d routing choice combinations of p_i. Then select the obstacle-avoiding one with minimal wirelength as os_i and meanwhile calculate the se_i of each os_i.
(3) Sort all terminals of OAOST in decreasing order according to se_i.
(4) For each terminal p_i in order, apply the routing choice combination of os_i to OAOST until all OAOST edges have been decided.

There are two points to note. First, in step (2), whether a routing choice combination avoids all obstacles or not can be determined by directly looking up EOT. Second, in step (4), if the routing choice of an OAOST edge has been decided, then it will not be changed even if the current os_i has a different choice for this edge. Moreover, both local and total wirelength can be calculated by directly looking up EST.

Algorithm 4 shows the pseudocode of the whole refinement, which includes redundant pseudo Steiner elimination, pseudo Steiner connection optimization, and routing choice optimization.

4.5. Complexity Analysis

THEOREM 4.6. *The time complexity of the proposed algorithm is* $O((m+n)\log x_m \log y_m + nm\log m)$, *where n and m are the number of pin vertices and obstacles, respectively, and* x_m *and* y_m *are the maximum x-coordinates and y-coordinates of the routing plane, respectively.*

PROOF. In step 1, a DT can be generated in $O(n\log n)$ time [Fortune 1987]. Then the OFEMST can be generated in $O(n)$ time with Kruskal's algorithm, because there are only $O(n)$ edges in DT.

In step 2, the TDST can be constructed in $m*(\log x_m \log y_m)$ time. Then there are two for-loops; n dominates the first loop, because there are $O(n)$ edges. For the second loop, it is a constant 4, which is the number of routing choices. And each query inside the second loop can be performed in $O(\log x_m \log y_m)$ time. Therefore, the time complexity of step 2 is $O((m+n)*(\log x_m \log y_m))$.

ALGORITHM 4: Refinement

Input: OAOST, OST, EOT, EST.
Output: OAOSMT.
begin
  for *each terminal* p_i *of OAOST* **do**
    if p_i *is a pseudo Steiner point* **then**
      take p_i's two endpoints p_s and p_t from the edge set;
      if $len(p_s p_i) + len(p_i p_t) >= len(p_s p_t)$ **then**
        delete terminal p_i, octilinear edge $p_s p_i k_1$ and $p_i p_t k_2$;
        add the connection information of edge $p_s p_t$ to EOT and EST;
        connect p_s to p_t;
      end
    end
  end
  for *each edge* $p_i p_j k$ *of OST* **do**
    if $\{B_{ijk}\} \neq \emptyset$ *And* $| \{B_{ijk}\} \neq \emptyset | > 1$ **then**
      if *there are two pseudo Steiner points s and c And s and c are not collinear* **then**
        compute $T = \{t_1, t_2, ...t_n\}$;
        slide s to t_i on the path from s to c such that $dis(s, t_i)$ is max;
        slide c to t_j on the path from c to s such that $dis(c, t_j)$ is max;
        if $t_i t_j 0$ *Or* $t_i t_j 1$ *can avoid all the obstacles* **then**
          delete octilinear edge sc;
          generate octilinear edge st_i, $t_i t_j$, and $t_j c$;
        end
      end
    end
  end
  Initialize $degree() = 0$; $list() = \emptyset$;
  for *each edge* $p_i p_j k$ *of OAOST* **do**
    $degeree(p_i) = degree(p_i) + 1$;
    $degeree(p_j) = degree(p_j) + 1$;
    add p_j to $list(p_i)$;
    add p_i to $list(p_j)$;
  end
  Initialize $length(os_i) = +\infty$;
  for *each terminal* p_i *of OAOST* **do**
    for *each substructure st of* p_i **do**
      if $obstacle - avoiding(st) = True$ *And* $length(st) < length(os_i)$ **then**
        $os_i = st$; //According to $degree(p_i)$ and $list(p_i)$
      end
    end
    $se_i = length\_shared\_edges(os_i)$;
  end
  $Sort(P)$ in decreasing order according se_i;
  for *each terminal* p_i *of OAOST* **do**
    Apply os_i to OAOST;
    if *the routing choice of all OAOST edges have been decided* **then**
      break;
    end
  end
end

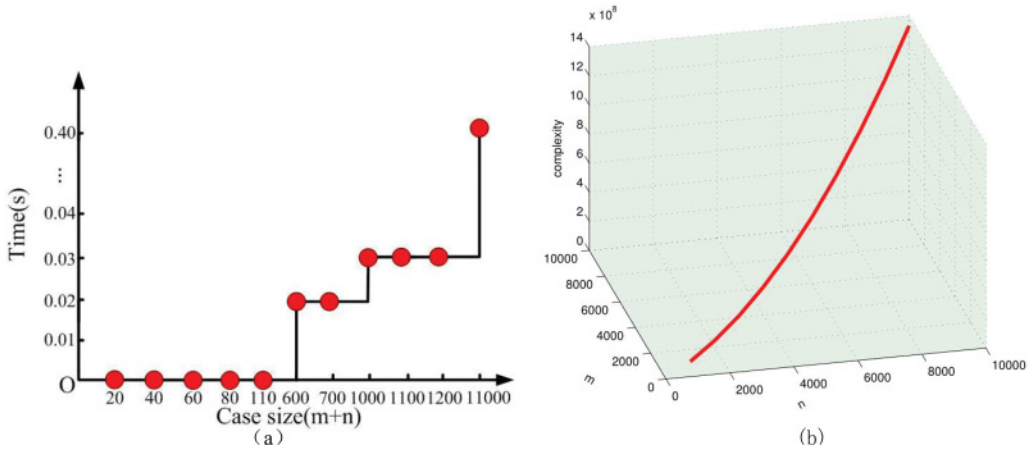

Fig. 13. Illustration for time complexity analysis. (a) Runtime versus input size of FH-OAOS. (b) Curve of complexity.

In step 3, OST generation needs $O(n)$ time, because there are $O(n)$ edges and table lookup only needs $O(1)$ time. For the OAOST generation process, the outer for-loop is dominated by n, and then for each OST edge, if it runs through obstacles, finding the bounding box in the worst case needs $O(m)$ time. If we select pseudo Steiner points on each obstacle, sort obstacles dominate this process, which can be performed in $O(m \log m)$ time. Furthermore, when the connection information of new paths is generated, each query needs $O(\log x_m \log y_m)$ time. Thus, the time complexity of step 3 is $O(n * (m \log m + \log x_m \log y_m))$.

In step 4, both redundant pseudo Steiner elimination and pseudo Steiner connection optimization need $O(n * (\log x_m \log y_m))$ time since there are $O(n)$ edges, and the connection information of new edges should be calculated. During the routing choice optimization process, scanning each edge of the OAOST needs $O(n)$ time. When the os_i of each terminal p_i is calculated, two for-loops are dominated by n and 4^d, respectively. In addition, all terminals can be sorted in $O(n \log n)$ time by using a quick sort algorithm, and applying the optimal structure of each terminal to the OAOST also needs $O(n)$ time; thus, routing choice optimization is dominated by $O(n \log n)$. In conclusion, the time complexity of step 4 is $O(n * (\log x_m \log y_m + \log n))$.

All in all, the time complexity of the proposed algorithm in the worst case is $O((m + n) \log x_m \log y_m + nm \log m)$. Moreover, we use $m + n$ as the case size and select the benchmark circuits from Section 5 of our article (rc01–rc12). The input size of these benchmarks is increasing. Figure 13(a) shows the curve of runtime versus case size. It can be seen that our algorithm runs fast for most of the benchmarks; for the largest input size with 1,000 pins and 10,000 obstacles, our algorithm can generate an OAOSMT in only 0.41s. In addition, both x_m and y_m are 10,000 in all benchmark circuits used in Section 5. Figure 13(b) shows the simulated complexity curve with some integer points. □

5. EXPERIMENTAL RESULTS

All algorithms have been implemented and executed in C language. Moreover, Matlab has been used to simulate the final routing diagram. The experiments were performed on a PC with 2.9GHz CPU and 2Gb memory. There are 17 benchmark circuits in total. ind1 through ind5 are industrial test cases from Synopsys. rc01 through rc12 are benchmarks of the obstacle-avoiding problem. In addition, some randomly generated circuits

Table IV. Validation of Refinement Process

Bench	*Pin#*	*Obs#*	OAOST	OAOSMT	Imp(%)
ind1	10	32	584	568	2.74
ind2	10	43	10,066	9,548	5.15
ind3	10	50	601	574	4.49
ind4	25	79	1,179	1,069	9.33
ind5	33	71	1,402	1,334	4.85
rc01	10	10	27,716	25,084	9.50
rc02	30	10	43,758	39,488	9.76
rc03	50	10	56,048	54,177	3.34
rc04	70	10	64,011	59,643	6.82
rc05	100	10	79,844	72,738	8.90
rc06	100	500	83,842	77,592	7.45
rc07	200	500	113,528	105,480	7.09
rc08	200	800	122,917	113,110	7.98
rc09	200	1,000	120,048	110,642	7.84
rc10	500	100	161,346	155,579	3.57
rc11	1,000	100	222,132	216,401	2.58
rc12	1,000	10,000	726,837	702,544	3.34
Average					6.16

are used to further test the scalability of the proposed algorithm. In all these test cases, the number of pins and obstacles is from 10 to 1,000 and from 10 to 10,000, respectively.

5.1. Experiment Validation of Refinement

The refinement process is of great importance for the final routing quality of the algorithm. It constructs the final OAOSMT by increasing the wirelength of shared edges as much as possible while fully leveraging the routing resources to generate more diagonal segments. In order to study the effectiveness of the refinement process for optimizing wirelength, we illustrate the function of refinement by comparing the wirelength before and after using this method. Table IV shows the results, where Pin# and Obs# represent the number of pins and obstacles, respectively. OAOST and OAOSMT represent the wirelength before and after using the refinement method, respectively. $Imp\% = 100*(OAOST-OAOSMT)/OAOST\%$. It can be seen from Table IV that a 2.74% to 9.76% wirelength reduction can be achieved after using refinement. Furthermore, the average wirelength reduction can also reach 6.16%.

5.2. Comparison with the Best OAOSMT Algorithm

To the best of our knowledge, the literature [Huang et al. 2015] achieves the best results for specially solving the OAOSMT construction problem so far. In Huang et al. [2015], a PSO-based framework was proposed, which can generate an excellent OAOSMT in a reasonable time. In order to validate the effectiveness of FH-OAOS, we compare our algorithm with Huang et al. [2015] in the aforementioned benchmark circuits. We run both algorithms on the same PC. Table V shows the comparison of wirelength and runtime with Huang et al. [2015]. Column 4 and column 5 show the wirelength of FH-OAOS and Huang et al. [2015], respectively. $\Delta w\% = 100*(Huang-FH-OAOS)/Huang$. We observed that FH-OAOS produces better results when the scale of input circuits is large (i.e., larger scale net with many obstacles). For example, for the test cases rc06 through rc12, FH-OAOS has a smaller wirelength than those of Huang et al. [2015]. In addition, it can be seen from Table V that a -4.18% to 3.10% reduction can be achieved when compared with Huang et al. [2015]. On average, FH-OAOS performs comparably to that of Huang et al. [2015]; the wirelength of FH-OAOS is only 0.36% longer than

Table V. Comparison with the Best OAOSMT Algorithm [Huang et al. 2015]

			Wire_length			CPU Time	
Bench	*Pin#*	*Obs#*	FH-OAOS	Huang	$\Delta w\%$	FH-OAOS	Huang
ind1	10	32	568	562	−1.07	0.00	0.02
ind2	10	43	9,548	9,431	−1.24	0.00	0.02
ind3	10	50	574	574	0.00	0.00	0.02
ind4	25	79	1,069	1,033	−3.48	0.00	0.02
ind5	33	71	1,334	1,288	−3.57	0.00	0.05
rc01	10	10	25,084	24,717	−1.34	0.00	0.02
rc02	30	10	39,488	40,751	3.10	0.00	0.02
rc03	50	10	54,177	52,033	−4.12	0.00	0.10
rc04	70	10	59,643	57,250	−4.18	0.00	0.17
rc05	100	10	72,738	72,738	0.00	0.00	0.26
rc06	100	500	77,592	78,643	1.34	0.02	0.57
rc07	200	500	105,480	105,542	0.06	0.02	1.91
rc08	200	800	113,110	116,204	2.66	0.03	2.93
rc09	200	1,000	110,642	111,385	0.07	0.03	3.20
rc10	500	100	155,579	157,520	1.23	0.02	5.42
rc11	1,000	100	216,401	219,037	1.20	0.03	9.33
rc12	1,000	10,000	702,544	724,425	3.02	0.41	13.12
Average ($\Delta w\%$)/Total (Time)					−0.36%	0.56	37.18
Norm(Time)					-	1.00	66.39

that in Huang et al. [2015]. We analyze this result mainly for two reasons. First, when the scale of input circuits is small (i.e., small scale net with only a few obstacles), Huang et al. [2015] can search an excellent result by modestly increasing the iteration times of the PSO algorithm while maintaining a reasonable runtime. Furthermore, Huang et al. [2015] introduce two operators of the genetic algorithm, which can further improve the search capability of the PSO algorithm. Second, when the scale of input circuits becomes large, if Huang et al. [2015] modestly increase iteration times, it has limited effects. If Huang et al. [2015] substantially increase iteration times, although it may get a great result, the runtime will become unacceptable; thus, there needs to be a compromise between wirelength and runtime.

We provide the runtime of two algorithms in column 7 and column 8. Although the final results of Huang et al. [2015] seem pretty great, the runtime of this PSO-based algorithm is less than satisfactory. This is mainly due to the nature of the PSO algorithm; it is a swarm-based algorithm, and PSO needs more iteration times to search the solution space. Moreover, the number of candidate edges of Huang et al. [2015] is $O(n^2)$; thus, it needs much more time to compute this edge information. The last row of Table V is normalized over FH-OAOS. It can be seen that FH-OAOS is very fast for each group of circuits. For example, rc12 includes 1,000 pin vertices and 10,000 obstacles; FH-OAOS can generate a great result in only 0.41 seconds, while Huang et al. [2015] need 13.12 seconds. On average, FH-OAOS runs 66.39 times faster than the method of Huang et al. [2015], so it is more practical in industrial production.

5.3. Comparison with λ-Geometry Algorithm

In this part, we compare FH-OAOS with a λ-geometry algorithm [Jing et al. 2007]. According to Definition 3.1, when λ is set to two, Jing et al. [2007] can generate an OARSMT; when λ is set to four, Jing et al. [2007] can generate an OAOSMT. Table VI shows the comparison results. It can been seen that FH-OAOS outperforms Jing et al. [2007] with all test cases when $\lambda = 2$, and the average wirelength reduction can reach 27.83%. In addition, for the case when $\lambda = 4$, when the scale of the problem is small,

Table VI. Comparison with λ-Geometry Algorithm [Jing et al. 2007]

				Wire length		$\Delta w\%$		CPU Time		
Bench	*Pin#*	*Obs#*	FH-OAOS	$\lambda = 2$	$\lambda = 4$	$\lambda = 2$	$\lambda = 4$	FH-OAOS	$\lambda = 2$	$\lambda = 4$
rc01	10	10	25,084	30,410	27,279	17.51	8.05	0.00	0.00	0.00
rc02	30	10	39,488	45,640	41,222	13.48	4.21	0.00	0.00	0.00
rc03	50	10	54,177	58,570	52,432	7.50	−3.33	0.00	0.00	0.00
rc04	70	10	59,643	63,340	57,699	5.84	−3.37	0.00	0.00	0.01
rc05	100	10	72,738	83,150	73,090	12.52	0.48	0.00	0.00	0.01
rc06	100	500	77,592	149,725	135,454	48.18	42.72	0.02	0.06	0.07
rc07	200	500	105,480	181,470	162,762	41.87	35.19	0.02	0.06	0.08
rc08	200	800	113,110	202,741	182,056	44.21	37.87	0.03	0.10	0.12
rc09	200	1,000	110,642	214,850	193,228	48.50	42.74	0.03	0.13	0.15
rc10	500	100	155,579	198,010	176,497	21.43	11.85	0.02	0.03	0.03
rc11	1,000	100	216,401	250,570	222,758	13.64	2.85	0.03	0.04	0.05
rc12	1,000	10,000	702,544	1,723,990	1,564,170	59.25	55.09	0.41	2.82	3.03
Average ($\Delta w\%$)/Total (Time)						27.83	19.53	0.56	3.24	3.55
Norm(Time)						-	-	1.00	5.79	6.34

the results in Jing et al. [2007] have a similar performance to ours on average (rc01–rc05). However, when the scale of the problem is large (rc06–rc12), especially when the number of obstacles is more than the one of pins, our algorithm has a substantial advantage with a −3.37% to 55.09% wirelength reduction. The average improvement can also reach 19.53%. In other words, the final routing trees generated by Jing et al. [2007] are relatively poor, the main reason being that Jing et al. [2007] may introduce much more redundant points, and the routing tree of Jing et al. [2007] cannot share the same routing path as much as possible. In addition, the the last three columns show the runtime of the two algorithms. It can be seen that FH-OAOS is 5.79 and 6.34 times faster than Jing et al. [2007] on average when $\lambda = 2$ and $\lambda = 4$, respectively. It is obvious that Jing et al. [2007] is better than Huang et al. [2015] in terms of runtime. However, for the quality of the final routing tree, Huang et al. [2015] is better than Jing et al. [2007]. Thus, FH-OAOS makes up for the drawback of Huang et al. [2015] and Jing et al. [2007], respectively.

5.4. Comparison with the Latest Three OARSMT Algorithms

In order to further verify the reality, that is, that octilinear architecture has enormous advantages for wirelength optimization when compared to rectilinear architecture, we compare our algorithm with three state-of-the-art OARSMT algorithms in this part, which were all proposed in recent years [Chow et al. 2014; Ajwani et al. 2011; Long et al. 2008]. Table VII shows the comparison results. Column 4 is the wirelength of our algorithm. $\Delta w\% = 100 * (others - ours)/others\%$. Chow et al. [2014] is the latest OARSMT literature, which proposed a parallel approach with the aid of GPU and achieved great results in an efficient way. Compared with this parallel algorithm, we get a −0.51% to 8.25% improvement in wirelength and a 3.96% reduction on average, with only one test case (ind2) being worse than it. In addition, our algorithm outperforms Ajwani et al. [2011] by 4.23% and Long et al. [2008] by 6.77% on average, respectively.

Although the introduction of 45° and 135° routing directions increases the complexity of the problem, our algorithm is also faster than those OARSMT algorithms. There are four main reasons. First, because step 1 generates an MST, the rectangular bounding box of two pins is small unless they are very distant from each other. Thus, most obstacles need not be inspected in step 2 and step 3. Second, for OAOST generation in step 3, only edges that run through obstacles need to be inspected. Third, all judgments in step 3 and step 4 can be performed in O (1) time by directly looking up tables. In

Table VII. Comparison with Three OARSMT Algorithms [Chow et al. 2014; Ajwani et al. 2011; Long et al. 2008]

				Wire_length			$\Delta w\%$			CPU Time			
Bench	*Pin#*	*Obs#*	FH-OAOS	Chow	Ajwani	Long	Chow	Ajwani	Long	FH-OAOS	Chow	Ajwani	Long
ind1	10	32	568(618)	609	604	639	6.73(−1.48)	5.96(−2.32)	11.11(3.29)	0.00(0.00)	0.05	0.00	0.01
ind2	10	43	9548(9800)	9500	9500	10000	−0.51(−3.16)	−0.51(−3.16)	4.52(2.00)	0.00(0.00)	0.06	0.00	0.01
ind3	10	50	574(613)	600	600	623	4.33(−2.17)	4.33(−2.17)	7.87(1.61)	0.00(0.00)	0.05	0.00	0.01
ind4	25	79	1069(1146)	1092	1129	1126	2.11(−4.95)	5.31(−1.51)	5.12(−1.78)	0.00(0.00)	0.09	0.00	0.02
ind5	33	71	1334(1412)	1345	1364	1379	0.82(−4.98)	2.20(−3.52)	3.26(−2.39)	0.00(0.00)	0.08	0.00	0.02
rc01	10	10	25084(27630)	25980	25980	27540	3.45(−6.35)	3.45(−6.35)	8.92(−0.33)	0.00(0.00)	0.05	0.00	0.01
rc02	30	10	39488(43290)	41740	42110	41930	5.40(−3.71)	6.23(−2.80)	5.82(−3.24)	0.00(0.00)	0.06	0.00	0.01
rc03	50	10	54177(56940)	55500	56030	54180	2.38(−2.59)	3.31(−1.62)	0.00(−5.09)	0.00(0.00)	0.07	0.00	0.01
rc04	70	10	59643(61990)	60120	59720	59050	0.79(−3.11)	0.13(−3.80)	−1.00(−4.98)	0.00(0.00)	0.06	0.00	0.02
rc05	100	10	72738(75685)	75390	75000	75630	3.52(−0.39)	3.02(−0.91)	3.82(−0.07)	0.00(0.00)	0.09	0.00	0.02
rc06	100	500	77592(84662)	81340	81229	86381	4.61(−4.08)	4.48(−4.23)	10.17(1.99)	0.02(0.02)	0.38	0.03	0.13
rc07	200	500	105480(113598)	110952	110764	117093	4.93(−2.38)	4.77(−2.56)	9.92(2.98)	0.02(0.02)	0.31	0.03	0.15
rc08	200	800	113110(119177)	115663	116047	122306	2.21(−3.04)	2.53(−2.70)	7.52(2.56)	0.03(0.02)	0.46	0.05	0.27
rc09	200	1000	110642(117074)	114275	115593	119308	3.18(−2.45)	4.28(−1.28)	7.26(1.87)	0.03(0.02)	0.61	0.06	0.36
rc10	500	100	155579(167219)	167830	168280	167978	7.30(0.37)	7.55(0.63)	7.38(0.45)	0.02(0.01)	0.20	0.02	0.08
rc11	1000	100	216401(234107)	235866	234416	232381	8.25(0.75)	7.69(0.13)	6.87(−0.74)	0.03(0.02)	0.34	0.03	0.14
rc12	1000	10000	702544(775263)	762089	756998	842689	7.81(−1.73)	7.19(−2.41)	16.63(8.00)	0.41(0.36)	21.78	1.19	5.88
Average ($\Delta w\%$)/Total (Time)							3.96(−2.66)	4.23(−2.38)	6.77(0.36)	0.56(0.47)	24.74	1.41	7.15
Norm(Time)							-	-	-	1.00 (1.00)	44.18 (52.64)	2.52 (3.00)	12.77 (15.21)

addition, compared to many MST-oriented algorithms, our n is very small, because the maximum value of n is equal to the number of pin vertices plus the number of corner points of all obstacles in their algorithms. We provide the runtime of different algorithms in Table VII. It can be seen that compared to the OARSMT algorithms, we are 44.18 times faster than Chow et al. [2014] and 12.77 times faster than Long et al. [2008]. In particular, the algorithm proposed in Ajwani et al. [2011] achieves the fastest runtime among three algorithms, because it is a precomputed lookup-table-based algorithm. Compared to it, we also achieve 2.52 times speedup on average.

Moreover, in order to make an apples-to-apples comparison, we try to generate an OARSMT by making the following changes to FH-OAOS. First, we delete choice 0 and choice 1 from the whole algorithm; thus, step 3 of the algorithm will convert the generated OFEMST to a rectilinear Steiner tree (RST). Then we generate an obstacle-avoiding RST (OARST) by selecting some pseudo Steiner points from the routing plane. It should be noted that the routing plane is divided into four rectilinear regions with reference to each point p in step 3 (namely, d_1, d_3, d_5, and d_7 are available directions in Figure 8(a)). Second, the pseudo Steiner connection optimization process becomes unavailable since it is based on the octilinear architecture, and only rectilinear combinations should be considered for each point during the routing choice optimization process. Moreover, since there are only rectilinear paths, redundant pseudo Steiner elimination also cannot reduce wirelength, because it is designed to try to replace the redundant rectilinear edge by a legal octilinear edge as much as possible. It can be seen that the refinement process almost has no effect after choice 0 and choice 1 are deleted. This is mainly because the optimization techniques for octilinear architecture and rectilinear architecture are usually not universal. For example, Borah et al. [1994]'s edge-substitution method and "U-shape pattern refinement" method are widely used to optimize rectilinear architecture [Chow et al. 2014; Long et al. 2008], but it is not suitable for octilinear architecture. We show the optimized OARSMT results in parentheses in Table VII; it can be seen that the performance of FH-OAOS is worse than Chow et al. [2014] (−2.66%) and Ajwani et al. [2011] (−2.38%) on average, and 0.36% better than Long et al. [2008]. In addition, for another important indicator of VLSI routing, namely, runtime, FH-OAOS still has a huge advantage. FH-OAOS is 52.64, 3.00, and 15.21 times faster than Chow et al. [2014], Ajwani et al. [2011], and Long et al. [2008], respectively.

5.5. Comparison with the Optimal OARSMT Algorithms

The proposed algorithm in Huang and Young [2011] is an exact algorithm, which can construct an optimal OARSMT by the concatenation of full Steiner trees [Fortune 1987] among obstacles. Table VIII shows our results as compared with Huang and Young [2011]. Column 4 of Table VIII shows that FH-OAOS outperforms Huang and Young [2011] by 2.35% on average. These results are further evidence that octilinear architecture is of great advantage to reduce routing cost in VLSI design. In addition, Column 6 shows that FH-OAOS is 267,387.53 times faster than Huang and Young [2011]. This is mainly because the method of Huang and Young [2011] is expected to have an exponential worst-case time complexity.

5.6. Comparison on OSMT Generation

An OSMT can be seen as a special case for an OAOSMT. In fact, FH-OAOS can generate an OSMT without requiring any modification. We directly performed experiments on our existing benchmarks after deleting obstacles. As shown in Table IX, we compare our results with Huang et al. [2015]. FH-OAOS* represents the wirelength before using the refinement method. We could not compare our algorithm with Jing et al. [2007] as this algorithm cannot directly be applied to cases with no obstacles. It can be seen

Table VIII. Comparison with Exact Algorithm [Huang and Young 2011]

	Wirelength			CPU time(s)	
Bench	FH-OAOS	Huang	$\Delta w\%$	FH-OAOS	Huang
ind1	568	604	5.96	0.00	0.11
ind2	9,548	9,500	−0.52	0.00	0.25
ind3	574	600	4.33	0.00	0.19
ind4	1,069	1,086	1.57	0.00	0.87
ind5	1,334	1,341	0.52	0.00	1.09
rc01	25,084	25,980	3.45	0.00	0.16
rc02	39,488	41,350	4.50	0.00	0.52
rc03	54,177	54,160	−0.03	0.00	0.68
rc04	59,643	59,070	−0.97	0.00	0.95
rc05	72,738	74,070	1.80	0.00	1.31
rc06	77,592	79,714	2.66	0.02	335
rc07	105,480	108,740	3.00	0.02	541
rc08	113,110	112,564	−0.49	0.03	24,170
rc09	110,642	111,005	0.33	0.03	14,174
rc10	155,579	164,150	5.22	0.02	176
rc11	216,401	230,837	6.25	0.03	706
Average($\Delta w\%$)/Total(Time)			2.35	0.15	40,108.13
Norm			-	1.00	267,387.53

Table IX. Wirelength and CPU Time Comparison for Benchmarks with No Obstacles

				Wire_length		$\Delta w\%$		CPU Time	
Bench	*Pin#*	*Obs#*	FH-OAOS	FH-OAOS*	Huang	FH-OAOS*	Huang	FH-OAOS	Huang
ind1	10	0	563	578	559	2.60	−0.72	0.00	0.01
ind2	10	0	8,814	8,838	8,814	0.27	0.00	0.00	0.01
ind3	10	0	547	559	547	2.15	0.00	0.00	0.01
ind4	25	0	955	975	963	2.05	0.83	0.00	0.01
ind5	33	0	1,152	1,163	1,147	0.95	−0.44	0.00	0.02
rc01	10	0	24,098	24,311	24,123	0.88	−0.10	0.00	0.01
rc02	30	0	35,734	36,450	36,203	1.96	1.30	0.00	0.01
rc03	50	0	48,553	49,816	48,819	2.54	0.54	0.00	0.07
rc04	70	0	51,737	52,992	50,627	2.37	−2.19	0.00	0.13
rc05	100	0	67,814	69,358	69,105	2.23	1.87	0.00	0.20
rc06	100	0	72,247	73,479	72,996	1.68	1.03	0.00	0.22
rc07	200	0	98,341	100,137	97,541	1.79	−0.82	0.00	1.02
rc08	200	0	101,239	103,002	101,479	1.71	0.24	0.00	1.23
rc09	200	0	98,724	100,193	99,774	1.47	1.05	0.00	1.37
rc10	500	0	150,080	152,713	151,443	1.72	0.90	0.02	2.29
rc11	1000	0	214,841	219,484	214,073	2.12	−0.36	0.03	4.35
rc12	1000	0	698,164	712,154	707,993	1.96	1.39	0.03	5.15
Average ($\Delta w\%$)/Total (Time)						1.79	0.27	0.08	16.11
Norm(Time)						-	-	1.00	201.38

from Table IX that the refinement method gets 1.79% wirelength reduction on average. In addition, our results for wirelength are also better than Huang et al. [2015]. A −2.19% to 1.87% wirelength reduction can be achieved when compared with Huang et al. [2015]. The average wirelength reduction is 0.27%. For runtime, we are 201.38 times faster than Huang et al. [2015].

Table X. Testing on Randomly Generated Cases with More Obstacles

Circuit	*Pin#*	*Obs#*	Wirelength		CPU time(s)	
			OSMT	OAOSMT	OSMT	OAOSMT
random1	10	500	1,710	2,320	0.00	0.01
random2	50	500	41,948	46,013	0.00	0.01
random3	100	500	7,084	8,235	0.00	0.02
random4	100	1,000	7,190	12,233	0.00	0.03
random5	200	2,000	40,257	50,037	0.00	0.13

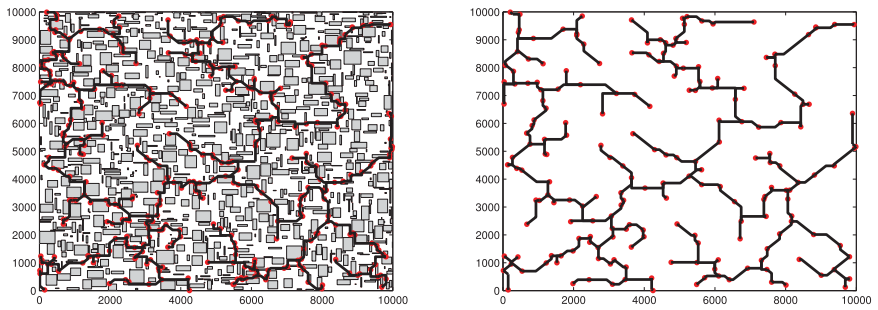

Fig. 14. Routing diagrams. (a) OAOSMT for rc09. (b) OSMT for rc09.

5.7. Experiment on Randomly Generated Circuits and Simulation Results

Moreover, to study the scalability of our algorithm, we test some randomly generated cases with additional obstacles. In these cases, the number of obstacles is far more than the one of pins. These cases are more similar to the practical routing applications, such as detailed routing or engineering change order (ECO) routing. As shown in Table X, our algorithm can efficiently generate OSMT and OAOSMT for all test cases.

Finally, for a better understanding of our algorithm, we use Matlab to simulate the final routing diagrams. We choose rc09 as a representative. Both OAOSMT and OSMT are given in Figure 14.

6. SUMMARY AND CONCLUSIONS

With the rapid development of VLSI manufacturing technology, 45° and 135° diagonal segments can be permitted in an octilinear routing plane. In this article, we have designed an efficient and effective obstacle-avoiding algorithm in octilinear architecture, namely, a fast four-step heuristic for OAOSMT and OSMT construction. We employ C language to perform the experiment and use Matlab to simulate the final routing diagram. Multiple sets of standard test cases and some randomly generated cases have been tested. Compared to several state-of-the-art algorithms, experimental results show that the designed algorithm achieves great results in terms of both wirelength and runtime, and it is extremely practical and useful in the process of VLSI routing.

REFERENCES

J. L. Ganley and J. P. Cohoon. 1996. Rectilinear Steiner trees on a checkerboard. *ACM Trans. Des. Autom. Electron. Syst.* 1, 4. 512–522.

A. Hashimotao and J. Stevens. 1988. Wire routing by optimizing channel assignment with in large apertures. In *Proceedings of the 8th Design Autom Conference (DAC'88)*. ACM Press, 35–49.

T. Y. Ho, Y. W. Chang, and S. J. Chen. 2007. *Full-Chip Nanometer Routing Techniques.* Springer, Berlin.

J. Hu and S. Sapatnekar. 2001. A survey on multi-net global routing for integrated circuits. *Integration, VLSI J.* 31, 1–49.

M. Hanan. 1966. On Steiners problem with rectilinear distance. *SIAM J. Appl. Math.* 14, 255–265.

M. Garey and D. Johnson. 1977. The rectilinear Steiner tree problem is NP-complete. *SIAM J. Appl. Math.* 32, 826–834.

Y. F. Wu, P. Wirdmayer, M. D. F. Schlag, and C. K. Wong. 1987. Rectilinear shortest paths and minimum spanning trees in the presence of rectilinear obstacles. *IEEE Trans. Comput.* 36, 321–331.

G. Ajwani, C. Chu, and W. K. Mak. 2011. FOARS: FLUTE based obstacle-avoiding rectilinear Steiner tree construction. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 30, 194–204.

J. Long, H. Zhou, and S. O. Memik. 2008. EBOARST: An efficient edge-based obstacle-avoiding rectilinear Steiner tree construction algorithm. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 27, 2169–2182.

C. W. Lin, S. Y. Chen, C. F. Li, Y. W. Chang, and C. L. Yang. 2007. Efficient obstacle-avoiding rectilinear Steiner tree construction. In *Proceedings of the International Symposium on Physical Design (ISPD'07)*. 127–134.

C. H. Liu, S. Y. Yuan, S. Y. Kuo, and Y. H. Chou. 2009. An O(nlogn) path based obstacle-avoiding algorithm for rectilinear Steiner tree construction. In *Proceedings of the Design Automation Conference (DAC'09)*. 314–319.

W. K. Chow, L. Li, E. F. Y. Young, and C. W. Sham. 2014. Obstacle-avoiding rectilinear Steiner tree construction in sequential and parallel approach. *Integration, the VLSI J.* 47, 105–114.

T. Huang and E. F. Y. Young. 2010. Obstacle-avoiding rectilinear Steiner tree construction: An optimal approach. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'10)*. ACM Press, 610–613.

T. Huang and E. F. Y. Young. 2011. An exact algorithm for the construction of rectilinear Steiner minimum trees among complex obstacles. In *Proceedings of the Design Automation Conference (DAC'11)*. ACM Press, 164–169.

T. Huang, L. Li, and E. F. Y. Young. 2011. On the construction of optimal obstacle-avoiding rectilinear Steiner minimum trees. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 30, 718–731.

C. K. Koh and P. H. Madden. 2000. Manhattan or non-Manhattan? A study of alternative VLSI routing architectures. In *Proceedings of the 10th Great Lakes Symposium on VLSI (GLSVLSI'00)*. ACM Press, 47–52.

S. Teig. 2000. The X architecture: Not your fathers diagonal wiring. In *Proceedings of the International Workshop on System-Level Interconnect Prediction (SLIP'02)*. ACM Press, New York, 33–37.

A. B. Kahng, I. I. Mandoiu, and A. Z. Zelikovsky. 2003. Highly scalable algorithms for rectilinear and octilinear Steiner trees. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC'03)*. IEEE, 827–833.

C. S. Coulston. 2003. Constructing exact octagonal Steiner minimal tree. In *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI'03)*. ACM Press, 597–602.

T. Y. Ho, C. F. Chang, Y. W. Chang, and S. J. Chen. 2005. Multilevel full-chip routing for the X-based architecture. In *Proceedings of the 42th Annual Design Automation Conference (ADAC'05)*. 28–29.

T. Arora and M. E. Mose. 2009. Ant colony optimization for power efficient routing in Manhattan and non-Manhattan VLSI architectures. In *Proceedings of the Swarm Intelligent Symposium (SIS'09)*. 137–144.

Q. Zhu, H. Zhou, T. Jing, X. L. Hong, and Y. Yang. 2005. Spanning graph-based nonrectilinear Steiner tree algorithm. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 24, 1066–1075.

J. T. Yan. 2008. Timing-driven octilinear Steiner tree construction based on Steiner-point reassignment and path reconstruction. *ACM Trans. Des. Autom. Electron. Syst.* 13, 2.

H. H. Huang, S. P. Chang, Y. C. Lin, and T. M. Hsieh. 2009. Timing-driven non-rectangular obstacle-avoiding routing algorithm for the X-architecture. In *Proceedings of the 8th WSEAS International Conference on Instrumentation, Measurement, Circuits and Systems (IMCAS'09)*. 31–34.

G. G. Liu, W. Z. Guo, Y. Z. Niu, G. L. Chen, and X. Huang. 2015. A pso-based-timing-driven octilinear Steiner tree algorithm for VLSI routing considering bend reduction. *Soft Comput.* 1153–1169.

X. Huang, G. G. Liu, W. Z. Guo, Y. Z. Niu, and G. L. Chen. 2015. Obstacle-avoiding algorithm in X-architecture based on discrete particle swarm optimization for VLSI design. *ACM Trans. Des. Autom. Electron. Syst.* 20, 2.

T. T. Jing, Z. Feng, Y. Hu, X. L. Hong, X. D. Hu, and G. Y. Yan. 2007. λ-OAT: λ-geometry obstacle-avoiding tree construction with O (n log n) complexity. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 26, 2073–2079.

M. Borah, R. M. Owens, and M. J. Irwin. 1994. An edge-based heuristic for Steiner routing. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 13, 1563–1568.

D. M. Warme, P. Winter, and M. Zachariasen. 2000. Exact algorithms for plane Steiner tree problems: A computational study. *Advances in Steiner Trees* 81–116.

C. C. Tong and C. L. Wu. 1995. Routing in a three-dimensional chip. *IEEE Trans. Comput.* 44, 106–117.

F. Schmiedle R. Drechsler, and B. Becker. 2003. Exact routing with search space reduction. *IEEE Trans. Comput.* 52, 815–825.

F. Schmiedle R. Drechsler, and B. Becker. 2000. Incorporating yield enhancement into the floorplanning process. *IEEE Trans. Comput.* 49, 532–541.

H. Zhou. 2004. Efficient Steiner tree construction based on spanning graphs. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 22, 704–710.

S. Cinel and C. Bazlamacci. 2008. A distributed heuristic algorithm for the rectilinear Steiner minimal tree problem. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 27, 2083–2087.

C. Chu and Y. C. Wong. 2008. FLUTE: Fast lookup table based rectilinear Steiner minimal tree algorithm for VLSI design. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 27, 70–83.

X. L. Hong, Q. Zhu, T. Jing, Y. Wang, and Y. C. Cai. 2003. Non-rectilinear on-chip interconnects: An efficient routing solution with high performance. *Chin. J. Semicond.* 24, 225–233.

S. Fortune. 1987. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2, 153–174.

M. B. De, M. K. Van, M. Overmars, and C. O. Schwarzkopf. 2000. *Computational Geometry: Algorithms and Applications.* Springer, Berlin, 2000.

J. Ho, G. Vijayan, and C. K. Wong. 1989. A new approach to the rectilinear Steiner tree problem. In *Proceedings of the 26th Design Automation Conference.* ACM, 161–166.

C. H. Liu, S. Y. Yuan, S. Y. Kuo, and S. C. Wang. 2009. High-performance obstacle-avoiding rectilinear Steiner tree construction. *ACM Trans. Des. Autom. Electron. Syst.* 14, 3, 45.

Received June 2015; revised October 2015; accepted December 2015