ObSteiner: An Exact Algorithm for the Construction of Rectilinear Steiner Minimum Trees in the Presence of Complex Rectilinear Obstacles

Tao Huang and Evangeline F. Y. Young

***Abstract*—In this paper, we present ObSteiner, an exact algorithm for the construction of obstacle-avoiding rectilinear Steiner minimum trees (OARSMTs) among complex rectilinear obstacles. This is the first paper to propose a geometric approach to optimally solve the OARSMT problem among complex obstacles. The optimal solution is constructed by the concatenation of full Steiner trees among complex obstacles, which are proven to be of simple structures in this paper. ObSteiner is able to handle complex obstacles, including both convex and concave ones. Benchmarks with hundreds of terminals among a large number of obstacles are solved optimally in a reasonable amount of time.**

***Index Terms*—Full Steiner tree (FST), obstacle avoiding, rectilinear Steiner minimum tree (OARSMT), routing.**

I. Introduction

THE RECTILINEAR Steiner minimum tree (RSMT) problem asks for a shortest tree spanning a set of given terminals using only horizontal and vertical lines. It is a fundamental problem in physical design of very large scale integrated circuits (VLSI). Finding an RSMT is useful for routing and global wire length estimation, and is also important for congestion and timing estimation in floor planning and placement. Modern VLSI designs often contain rectilinear obstacles such as macrocells, IP blocks, and prerouted nets. Therefore, the obstacle-avoiding rectilinear Steiner minimum tree (OARSMT) problem has received a lot of research attentions in recent years. The RSMT problem is well known to be NP-complete [3], and the presence of obstacles further increases the problem complexity.

In recent years, many heuristics have been proposed for the OARSMT problem. In order to deal with rectilinear obstacles, some heuristic approaches [4]–[6] try to dissect a rectilinear obstacle into several rectangular obstacles. However, the cutting lines on the rectilinear obstacles allow wires to go through, which may result in an infeasible solution. The maze-routing approach proposed by Li and Young [7] constructs OARSMTs based on the extended Hanan grid. This maze-routing-based approach, by its nature, can handle complex obstacles, but is computationally expensive for large scale designs. Liu *et al.* [8] proposed a new framework to generate OARSMT based on the critical paths, which guarantees the existence of desirable solutions. Ajwani *et al.* [9] proposed a fast lookup table-based algorithm to route a multiterminal net in the presence of rectilinear obstacles. Their algorithm uses the obstacle-avoiding spanning graph to guide the partitioning of the initial solution and constructs the final OARSMT based on an obstacle-aware version of fast lookup table. Recently, Liu *et al.* [10] presented a Steiner-point-based framework to construct OARSMTs and showed that their algorithm can achieve best practical performance among existing heuristics.

In comparison with the heuristics, few exact algorithms have been proposed. Maze routing, proposed in [11], can give an optimal solution to two-terminal nets. Ganley and Cohoon [12] proposed a strong connection graph called an escape graph for the OARSMT problem and proved that there is an optimal solution composed only of escape segments in the graph. Based on the escape graph, they proposed an algorithm to construct optimal three-terminal and four-terminal OARSMTs. The works presented in [13] and [14] extended GeoSteiner [15] to an obstacle-aware version. Their algorithms are able to generate optimal OARSMTs for multiterminal nets in the presence of rectangular obstacles. However, these approaches cannot be applied when there are complex rectilinear obstacles in the routing region, as is often the case in the routing problem. Moreover, their algorithms can only handle benchmarks with less than 100 obstacles, while modern VLSI design may contain over 1000 obstacles. To the best of our knowledge, no previous algorithm can generate optimal solutions to the OARSMT problem with a large number of terminals among complex rectilinear obstacles. Although the escape graph model can transform the OARSMT problem into a graph problem that can be solved optimally by using some graph-based algorithms [16], [17], these approaches are believed to be less efficient than the geometric approaches for

Manuscript received August 24, 2012; revised October 30, 2012; accepted December 15, 2012. Date of current version May 15, 2013. This work was supported in part by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China, under Project CUHK418611. Preliminary results of this paper were presented at the Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, San Jose, CA, in November 2010 [1] and at the Proceedings of the ACM/IEEE Design Automation Conference, San Diego, CA, in June 2011 [2]. This paper was recommended by Associate Editor Y.-W. Chang.

The authors are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, Hong Kong (e-mail: thuang@cse.cuhk.edu.hk; fyyoung@cse.cuhk.edu.hk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2013.2238291

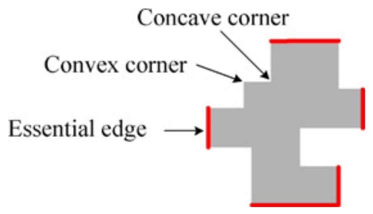

Fig. 1. Corners and essential edges of an obstacle.

solving the geometric Steiner tree problem.[1] An example is GeoSteiner [15], which remains the most efficient approach to solve the RSMT problem when no obstacle exists. Therefore, it is necessary to develop an efficient exact algorithm that allows the presence of complex obstacles. The aim of this paper is to propose an algorithm called ObSteiner to construct OARSMTs among rectilinear obstacles of both convex and concave shapes. To generate OARSMTs, we first study the full Steiner trees (FSTs) among complex obstacles and verify how their structures can be simplified by adding virtual terminals. We then propose an iterative three-phase approach to construct optimal OARSMTs based on GeoSteiner. The contributions of this paper can be summarized as follows.

1) This is the first paper to propose a geometric approach to exactly solve the OARSMT problem when there are complex rectilinear obstacles. Our algorithm is able to handle both convex and concave rectilinear obstacles. By using the proposed algorithm, benchmarks with up to 2000 obstacles are solved to optimal in a reasonable amount of time. This paper provides key insights into this difficult problem.
2) We design an efficient pruning procedure that can greatly reduce the size of the solution space and therefore improve the performance of the algorithm.
3) We adopt an incremental way to handle obstacles. An obstacle will be considered only if it is necessary. This approach is very effective when there are a large number of obstacles in the routing region.
4) Based on the theorem we developed in this paper, we further propose a simple graph model that can transfer the geometric OARSMT problem into a graph problem.

The rest of this paper is organized as follows. In Section II, we give preliminaries on the OARSMT problem and an exact algorithm for the RSMT problem. In Section III, we study the structures of FSTs among complex obstacles. Section IV and V describe the proposed exact algorithm in detail. Experiment results are presented in Section VI, followed by a conclusion in Section VII.

II. Preliminaries

A. OARSMT Problem Formulation

In this problem, we are given a set V of terminals and a set O of obstacles. An obstacle is a rectilinear polygon. All edges of an obstacle are either horizontal or vertical. Rectilinear polygons can be classified into two types: convex polygons and concave polygons. A rectilinear polygon is a convex rectilinear polygon if any two points in the polygon have a shortest Manhattan path lying inside the polygon. Otherwise, it is called a concave rectilinear polygon.

[1]Standard benchmarks for the Steiner tree problem in graphs also include rectilinear graphs that correspond to the RSMT problems. When solving these problems, most of the algorithms [16], [17] will preprocess them by using the first phase of GeoSteiner [15] to reduce the problem size. Otherwise, the problem will be much more difficult to solve. This is mainly because the algorithms for Steiner tree problem in graphs cannot exploit the geometric of the RSMT problem.

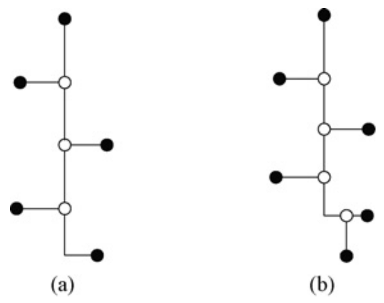

Fig. 2. FST structures in the absence of obstacles. (a) Type I structure. (b) Type II structure.

As shown in Fig. 1, a corner of an obstacle is the meeting point of two neighboring edges. If the two neighboring edges of a corner form a 90° angle inside the polygon, the corner is called a convex corner. Otherwise, if the two neighboring edges of a corner form a 270° angle inside the polygon, the corner is called a concave corner. If both endpoints of an edge are convex corners, this edge is called an essential edge (see Fig. 1). Note that the essential edge defined in this paper is also known as extreme edge in [18] and [19]. However, the way we make use of essential edges is very different.

A terminal cannot be located inside an obstacle, but it can be at the corner or on the edge of an obstacle. The OARSMT problem asks for a rectilinear Steiner tree with minimum total length that connects all terminals. No edge in the tree can intersect with any obstacle, but it can be point touched at a corner or line touched on an edge of an obstacle. This tree is known as an OARSMT.

In the following figures, we use a solid circle to denote a terminal and an empty circle to denote a Steiner point.

B. Exact RSMT Algorithm

The RSMT problem in the absence of obstacles has been studied excessively over the years [20]. Among various approaches, GeoSteiner [15] is the most efficient exact algorithm in practice. The algorithm is developed based on the construction of FSTs. An FST is an RSMT in which every terminal is a leaf node (i.e., of degree one). In the absence of obstacles, it is proven in [21] that an FST has one of the two generic forms as shown in Fig. 2, consisting of a backbone and alternating incident legs connecting the terminals. A folk theorem states that any RSMT can be decomposed into a set of edge-disjoint FSTs by splitting at terminals with degree more than one. Since FSTs are much easier to construct than RSMTs, most of the exact algorithms for the construction of an RSMT will first generate its FST components. GeoSteiner makes use of a two-phase approach, consisting of an FST generation phase and an FST concatenation phase, to construct an RSMT. In the first phase, a set of FSTs are generated such that there is

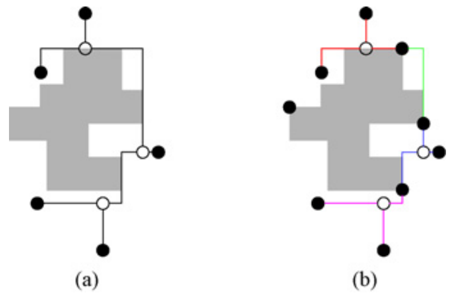

Fig. 3. (a) FST structure in the presence of obstacles. (b) Decomposition of FST after adding virtual terminals.

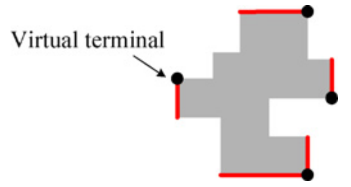

Fig. 4. Example of adding virtual terminals.

at least one RSMT composed of the FSTs in the set only. In the second phase, a subset of FSTs is selected and combined to form an RSMT. The key observation is that the FST concatenation problem is equivalent to the spanning tree in hypergraph problem and can be formulated as an integer linear programming. On the rectilinear plane, GeoSteiner remains the fastest exact algorithm for the RSMT problem, but it cannot be applied when obstacles exist in the routing plane.

III. OARSMT Decomposition

The exact algorithm for the RSMT problem indicates the importance of studying the structures of FSTs when there are obstacles in the routing region. However, the structures of these FSTs can be complicated due to the existence of rectilinear obstacles. We will show in this section how we can simplify the FST structures in the presence of complex obstacles by adding the so-called virtual terminals. In addition, we will propose a new simple graph model that contains at least one optimal solution for the OARSMT problem. This section gives the theoretical foundations for the exact algorithm for the OARSMT problem.

A. FSTs Among Complex Obstacles

To construct OARSMTs among complex obstacles, we first study the FSTs in the presence of complex obstacles. An example of such an FST in the presence of one obstacle is shown in Fig. 3(a). As we can observe from the figure, the structure of FSTs in the presence of obstacles can be very complicated. In such a case, the construction of FSTs can itself be a hard problem that limits the application of the two-phase approach to the OARSMT problem. Therefore, virtual terminals are added to simplify their structures in our approach. These terminals are called virtual because they can be connected by the OARSMT or not. It should be noted that, although virtual terminals are also used in [13] and [14], there are critical differences when dealing with rectilinear obstacles.

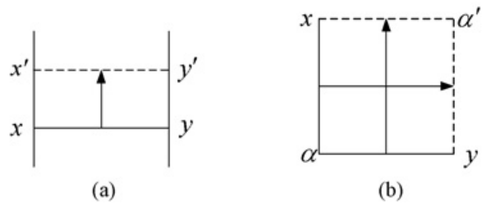

Fig. 5. Two operations on a rectilinear Steiner tree. (a) Shifting. (b) Flipping.

In [13] and [14], virtual terminals are added to every corner of each of the rectangular obstacles, whereas, in this paper, the virtual terminals are added in such a way that there is at least one virtual terminal on every essential edge of all the obstacles. This is a simplified but sufficient way of adding virtual terminals in comparison with those in [13] and [14]. Note that the location of a virtual terminal on an essential edge is not restricted. It will not affect the optimality of the solution. For simplicity, in the following proofs, we assume that the virtual terminal on an essential edge is located at one of its endpoints. An example is shown in Fig. 4. Note that for two essential edges sharing a common endpoint at a corner, we only need to add one virtual terminal at that corner. We use V' to denote the set of virtual terminals we added. The direct impact of adding a virtual terminal is that we can further decompose the complicated FSTs [see Fig. 3(a)] into smaller and simpler FSTs by splitting at the virtual terminals [see Fig. 3(b)]. We call these smaller trees FSTs among complex obstacles. In the following, we will give a formal definition to the FSTs among complex obstacles and prove that they will follow some very simple structures.

To define FSTs among complex obstacles, we introduce two operations as follows. As defined in [21], there are two basic operations on a tree that will not change the total length: shiftings and flippings, as shown in Fig. 5. Shifting a line means moving a line between two parallel lines to a new position. Flipping a corner means moving the two perpendicular lines of the corner so as to move the corner to the opposite position diagonally. A rectilinear Steiner tree t is equivalent to another tree t' if and only if t can be obtained from t' by flipping and shifting some lines that have no node on them. With these two operations, an FST among complex obstacles can be defined as follows.

Definition 1: An FST f over a set $V_f \subseteq V+V'$ of terminals is an OARSMT of V_f such that every terminal $v \in V_f$ is a leaf node in f and in all its equivalent trees. Moreover, all the equivalent trees of f cannot contain forbidden edges. A forbidden edge is an edge that passes through a virtual terminal. If an FST f or its equivalent trees contain forbidden edges, we can split this FST into smaller FSTs at this virtual terminal.

Note that the definition of FST in this paper is similar to the definitions in [13] and [14]. However, the obstacles considered in this paper are rectilinear polygons, which are more general and complicated than the rectangles considered in [13], [14]. In the following, we use FSTs to refer FSTs among complex obstacles for simplicity.

To derive the structures of an FST, we mainly follow the steps as described in [14] and [21]. The main difference is

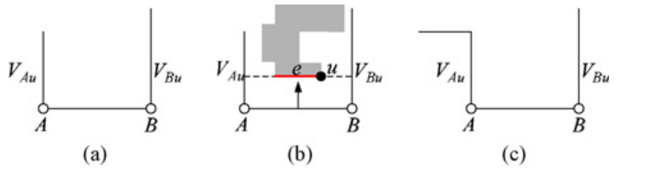

Fig. 6. Structure of two neighboring Steiner points when both V_{Au} and V_{Bu} exist and $|V_{Bu}| \geq |V_{Au}|$. (a) In the absence of obstacles. (b) In the presence of an obstacle. (c) Resulting structure of Lemma 2.

that there can be rectilinear obstacles in the routing region. For the two operations (i.e., shiftings and flippings) used in the proofs, it is possible that some of the operations cannot be performed due to obstacles. We will show in the following how this problem can be solved by adding virtual terminals.

In the following lemmas, a vertex can be a node (real terminal or virtual terminal) or a Steiner point. An edge between two vertices is a sequence of alternating vertical and horizontal lines, and each turning point is a corner. A line has only one direction but may contain a number of vertices on it. We use V_{xu} (V_{xd}) to denote the longest possible vertical line at point x, which extends above (below) x, excluding x itself. We use H_{xr} (H_{xl}) to denote the longest possible horizontal line at point x, which extends to the right (left) of x, excluding x itself. If a line ends at a node and contains no other vertices, we call it a node line. If it ends at a corner and contains no vertices, we call it a corner line.

Lemma 1: All Steiner points in an FST either have degree three or degree four.

Lemma 2: Let A and B be two adjacent Steiner points in an FST. Suppose that AB is a horizontal line and both V_{Au}, V_{Bu} exist. Then, $|V_{Bu}| \geq |V_{Au}|$ implies that V_{Au} is a line that ends at a corner turning away from V_{Bu}.

Proof: See Fig. 6(a). Suppose A is to the left of B.

1) V_{Au} contains no terminal at its endpoint, for otherwise we can shift AB to that terminal and obtain an equivalent tree in which a terminal has degree more than one. If the line AB cannot be shifted due to some obstacles as shown in Fig. 6(b), we can shift AB up until it overlaps with an edge e of the obstacle. According to the definition, since the two endpoints of e are convex corners, e is an essential edge. Let u be the virtual terminal added on e. As a result, AB will pass through u and thus is a forbidden edge, which is a contradiction to the definition of FSTs.

2) No Steiner points on V_{Au} can have a line going right, for otherwise we can replace AB by extending that line to meet V_{Bu} and reduce the total length. If the line cannot be extended due to obstacles, we can repeat the operation described in the previous step and result in a contradiction.

3) Therefore, the upper endpoint of V_{Au} cannot be a Steiner point since it has no lines going right or upward, hence it must be a corner turning left, as shown in Fig. 6(c).

4) V_{Au} can contain no Steiner point, for let C be such a Steiner point which is nearest to the corner point. Since H_{Cr} does not exist, H_{Cl} must exist. We can then shift the line between point C and the corner point to the left to reduce the total length, a absurdity. If the line cannot be shifted due to some obstacles (this line overlaps with an edge of the obstacle and this edge must be a essential edge), the line will pass through a virtual terminal, an absurdity. ■

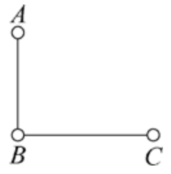

Fig. 7. Impossible Steiner chain structure in an FST.

Lemma 3 to Lemma 10 are analogous to Lemma 3 to Lemma 10 in [14]. We can use a similar method as the one illustrated in the proof of Lemma 2 to deal with possible obstacles. Therefore, the proofs are omitted here.

Corollary: Suppose V_{Bu} contains a vertex, then V_{Au} is a corner line that ends at a corner turning away from V_{Bu} and $|V_{Au}| < |V_{Bu}|$.

Lemma 3: Suppose V_{xu} (where x is a vertex) is a corner line ends at a corner turning left (right), then H_{xl} (H_{xr}) does not exist.

Lemma 4: No Steiner point can have more than one corner line.

Lemma 5: If f is an FST, the Steiner points in f form a chain.

We call the chain of Steiner points the *Steiner chain*.

Lemma 6: Suppose f is an FST. Then, its Steiner chain cannot contain the subgraph shown in Fig. 7.

Define a staircase to be a continuous path of alternating vertical lines and horizontal lines such that their projections on the vertical and horizontal axes have no overlaps.

Lemma 7: Suppose f is an FST. The Steiner chain of f is then a staircase.

Lemma 8: Suppose f is an FST. The Steiner chain of f cannot contain a corner with more than one Steiner points on the two neighboring lines.

Lemma 9: Suppose f is an FST. If the number of Steiner points is greater than two, either every vertical line on the Steiner chain contains more than one Steiner points (except possibly the first and the last vertical lines) and every horizontal line on the Steiner chain contains no Steiner point except at the endpoint, or vice versa.

Note that the structure of an FST is not affected by 90° rotation. In the following lemmas and theorems, we assume that if f is an FST, the corresponding Steiner chain will consist of a set of vertical lines, and adjacent vertical lines are connected by corners. We label the ith Steiner point on the chain counting from above by A_i.

Lemma 10: Suppose f is an FST. Every Steiner point on f must have a horizontal node line and the node lines alternate in the left–right directions.

The proofs of the above lemmas are similar to those in [14], except that of Lemma 11 in which flippings are required. In this paper, a different lemma is proposed.

Lemma 11: Let A_i be the ith Steiner point on the Steiner chain of an FST. A corner connecting A_i and A_{i+1} can be transferred to either one connecting A_{i-2} and A_{i-1}, or one connecting A_{i+2} and A_{i+3}, regardless of whether the place it transfers to has a corner of not. If the corner cannot be

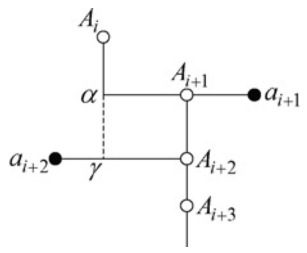

Fig. 8. Structure of Steiner chain when A_i and A_{i+1} are connected by a corner.

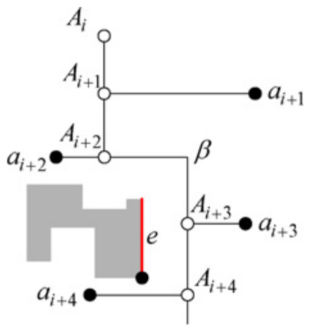

Fig. 9. Structure of Steiner chain when the corner between A_{i+2} and A_{i+3} cannot be flipped due to obstacles.

transferred due to obstacles, A_{i+3} is the last Steiner point or A_i is the first Steiner point on the chain (if A_{i+3} or A_i exist).

Proof: From Lemma 9 and Lemma 10, when A_i and A_{i+1} are connected by a corner, the graph must be the one given in Fig. 8. We use a_i to denote the node connected by A_i.

Necessarily $|a_{i+2}A_{i+2}| > |\alpha A_{i+1}|$, for otherwise we can shift $A_{i+1}A_{i+2}$ to a_{i+2} and obtain an equivalent tree in which a_{i+2} has degree two. Now shift $A_{i+1}A_{i+2}$ to α and suppose this line meets $a_{i+2}A_{i+2}$ at γ. Flip the corner A_{i+2} between γ and A_{i+3}. The corner connecting A_i and A_{i+1} is then transferred to one connecting A_{i+2} and A_{i+3}.

If the corner A_{i+2} cannot be flipped due to obstacles and A_{i+4} exists, the graph must be the one given in Fig. 9, in which there are obstacles inside the bounded rectangular region defined by A_{i+2} and A_{i+3}. We use β to denote the corner connecting A_{i+2} and A_{i+3}. We can shift βA_{i+4} to the left until it meets an edge e on one of the obstacles inside the rectangular region. Similar to the proof for Lemma 1, e is an essential edge and has a virtual terminal on it. Therefore, the FST has an equivalent tree that passes through a virtual terminal, a contradiction. If shifting βA_{i+4} meets a_{i+4} first, the FST has an equivalent tree in which a_{i+4} has degree two, again a contradiction. As a result, if the corner cannot be transferred due to obstacles, A_{i+3} is the last Steiner point on the chain if it exists. Similarly, we can transfer the corner to one connecting A_{i-2} and A_{i-1}. If the corner cannot be transferred, A_i is the first Steiner point on the chain if it exists.

Note that if A_{i+3} does not exist, the above operation eliminates the corner between A_i and A_{i+1}. ■

Lemma 12: Suppose f is an FST and let m be the number of Steiner points on f. There exists a f' equivalent to f such that:

1) if m is odd, the Steiner chain of f' is a straight line;

2) if m is even, all the Steiner points are on a straight line except possibly the last one.

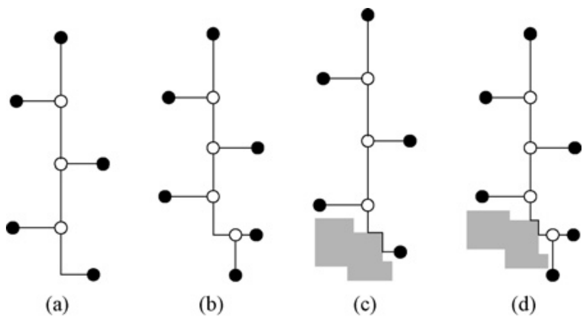

Fig. 10. Possible structures of an FST among complex obstacles. (a) Type I structure. (b) Type II structure. (c) Type III structure. (d) Type IV structure.

To summarize, if the Steiner chain is a straight line, the horizontal node line linked to the sequence of Steiner points must alternate in the left–right directions. Hence, each Steiner point has exactly one horizontal node line except for the first and the last one. Similar to what was done in [14], by putting all of the above lemmas together, we can have the following conclusion.

Theorem 1: The structures of an FST among complex obstacles must be in one of the four forms as shown in Fig. 10.

As we can observe from the figures, the structures of FSTs in the presence of rectilinear obstacles are very similar to those in [14] and [21]. The first two structures are exactly the same as those in [14] and [21]. However, in the presence of complex obstacles, the FSTs have two additional structures. A main characteristic of these two additional structures is that the last corner connecting two Steiner points or one Steiner point and one terminal is blocked by some obstacles. The similarities indicate that we can use the same method to construct the FSTs defined in this paper efficiently, making it possible to use the two-phase approach to solve an OARSMT problem in the presence of complex rectilinear obstacles.

B. More Theoretical Results

We mentioned in Section I that the OARSMT problem can be transformed into a graph problem by using the escape graph. The escape graph is the simplest existing graph model that contains at least one optimal solution to the OARSMT problem [22]. In this section, we will propose a new graph called a virtual graph that is simpler than the escape graph. Based on the theorem that we presented in the previous section, we will show that the virtual graph is a strong connection graph that contains at least one optimal solution.

The escape graph consists of two types of segments. The first type is the segments that extend from the terminals in the vertical and horizontal directions, and end at an obstacle boundary or the boundary of the whole routing region. The other type of segments is obtained by extending boundary segments of each obstacle until an obstacle boundary or the boundary of the whole routing region is met. The vertices of the graph are the terminals and the segment intersection points. An example is given in Fig. 11 where there are three terminals in the presence of three rectilinear obstacles. Therefore, the size of the escape graph is $O((m + b)^2)$, where m is the number of terminals and b is the number of obstacle boundary

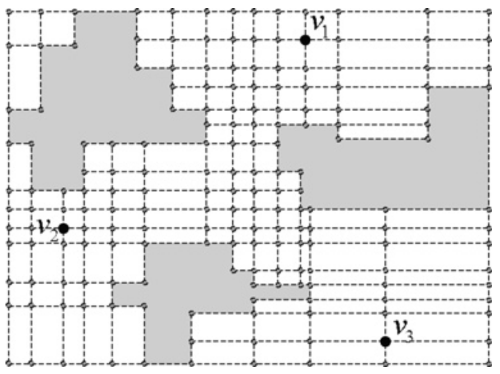

Fig. 11. Escape graph.

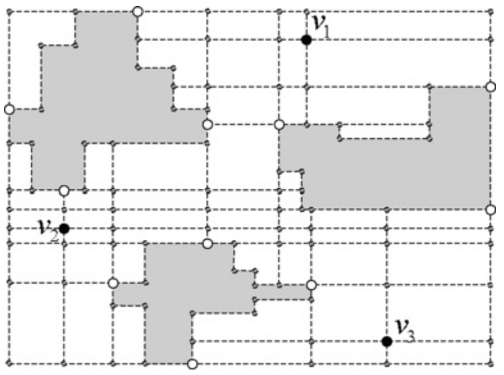

Fig. 12. Virtual graph.

segments. It is proven in [12] that for any OARSMT problem, there is at least one optimal solution composed only of the escape segments in the escape graph. The importance of the escape graph is that, with this model, one can transform the geometric OARSMT problem into a graph problem. As a result, some graph-based searching algorithms [16], [17] can also be applied to this problem. The introduction of the escape graph has also led to a set of heuristics [23], [24] for the OARSMT problem.

In the following, we will introduce a new graph called a virtual graph based on the virtual terminals we added to the problem. The virtual graph is composed of two types of segments. The first type is the segments that extend from the terminals and virtual terminals in the vertical and horizontal directions, and end at an obstacle boundary or the boundary of the whole routing region. The second type is the obstacle boundary segments. The vertices of the graph are the terminals, virtual terminals, and the segment intersection points. An example is shown in Fig. 12.

Theorem 2: For any OARSMT problem, there is at least one optimal solution contained in the virtual graph.

Proof: Any optimal OARSMT can be decomposed into a set of FSTs among complex obstacles. By Theorem 1, there are only two types of segments in the FSTs. The first type is the segments that extend from either a terminal or virtual terminal horizontally or vertically. The second type is the segments that go around obstacles. By the definition of the virtual graph, it can be easily verified that all FSTs can be further decomposed into segments in the virtual graph. Therefore, there is at least one optimal solution contained in the virtual graph. ■

By Theorem 2, we can see that virtual graph is also a strong connection graph. The size of the graph is $O((m + e)^2 + b)$, where e is the number of essential edges. In comparison with the escape graph, the size of the virtual graph is smaller. In the particular examples shown in Figs. 11 and 12, the escape graph consists of 184 nodes and 319 edges while the virtual graph only consists of 104 nodes and 158 edges. The simplicity of virtual graph also benefits from the flexibility in choosing the positions of the virtual terminals. Note that we only require one virtual terminal on each essential edge. As shown in Fig. 12, three virtual terminals are chosen to be internal points of essential edges to align with real terminals or other virtual terminals. This can further reduce the size of the graph.

Proposing a simple graph model is of vital importance for the OARSMT problem. Since the problem is NP-complete, a simpler graph can lead to a dramatic reduction of the solution space. Moreover, this graph model is also promising for the graph-based heuristics to improve their performance.

IV. OARSMT Construction

An OARSMT can be partitioned into a set of FSTs by splitting at real terminals or virtual terminals of degree more than one. Therefore, any OARSMT is a union of FSTs. As we can observe, FSTs are much easier to generate than OARSMTs. Therefore, one possible way to construct an OARSMT is to first construct its FSTs components and then combine a subset of them.

Similarly to [14], we adopt a two-phase approach to construct an OARSMT. The first phase is to generate a set of FSTs. The second phase is to combine a subset of FSTs generated in the first phase to construct an OARSMT. In our early experiments, we found that the FST concatenation phase usually dominates the total run time. Therefore, we propose a pruning algorithm to further eliminate useless FSTs resulting from the FST generation phase. This can reduce the number of FSTs that needs to be considered in the second phase leading to a significant improvement in the total run time.

A. Generation of FSTs

Since the structures of FSTs defined in this paper are very similar to those in [14], we can easily modify the FST generation approach in [14] to generate the FSTs defined in this paper. To generate the FSTs spanning three or more terminals, a modified version of the recursive algorithm in [25] is used. The FSTs spanning exactly two terminals are generated by using a different but more efficient approach. During the construction, some necessary conditions such as empty diamonds, empty corner rectangles and empty inner rectangles are used to eliminate those FSTs that cannot be a part of any OARSMTs. The result of the FST generation phase is a set of FSTs denoted by F.

To construct an OARSMT, for each set $V_f \subseteq V + V'$ of terminals, we need at most one FST. Therefore, a trivial upper bound for the number of FSTs is $O(2^{(m+e)})$ where m is the number of terminals and e is the number of essential edges (i.e., the number of virtual terminals).

For the RSMT problem, Ganley and Cohoon [26] proved an upper bound $O(m(\frac{1+\sqrt{5}}{2})^m)$ for the number of FSTs. By using the same approach as in [26], we can achieve a tighter bound $O((m+e)(\frac{1+\sqrt{5}}{2})^{(m+e)})$ for the number of FSTs defined in this paper. Note that although the worst-case bound is exponential, the actual number of FSTs generated is much smaller than this bound.

B. Pruning of FSTs

We propose an efficient pruning procedure to reduce the number of FSTs needed to construct an OARSMT. Although some pruning is also done in the FST generation phase, these tests consider only one FST at a time. To further eliminate useless FSTs, a set of FSTs should be considered simultaneously. The proposed pruning algorithm works by growing an FST f to larger trees and testing whether these larger trees can exist in the optimal solution. We know that virtual terminals in an OARSMT must have degree two, three, or four. Therefore, it is possible to grow a tree at a leaf node that is a virtual terminal. The growing is done by combining FSTs at virtual terminals. The rationale behind is that an FST f can be eliminated if no tree grown from f can exist in an OARSMT. The pseudocode of the pruning algorithm is shown in Algorithm 1.

Algorithm 1 Pseudocode of the pruning algorithm

```
Input: $f$
Output: $true$ or $false$
 1: initialize a queue $Q$
 2: push $f \to Q$
 3: while $Q$ is not empty do
 4:     pop $T \leftarrow Q$
 5:     if PASS_TEST($T$) then
 6:         if ALL_REAL($T$) then
 7:             return $false$
 8:         end if
 9:         $l$ = LEAST_DEGREE($T$)
10:         $S = \{f_i : (f_i \in F) \wedge (l \in f_i) \wedge (f_i \nsubseteq T)\}$
11:         for all $T' \in \binom{S}{1} \cup \binom{S}{2} \cup \binom{S}{3}$ do
12:             push $T \cup T' \to Q$
13:             if $Q$ is full then
14:                 return $false$
15:             end if
16:         end for
17:     end if
18: end while
19: return $true$
```

The input of the function PRUNE(f) is an FST f. The function returns a value of true or false to indicate whether f can be eliminated or not. A queue Q is used to store all the trees we can grow from f during the test. Initially, Q contains f only. The algorithm repeatedly removes a tree T from Q and tests if T can be a part of any OARSMT. The function PASS_TEST(T) returns true if T passes all the tests used to eliminate useless trees. In this case, the function LEAST_DEGREE(T) is used to select a virtual terminal l that is also a leaf node of T. If there are more than one such virtual terminals, the function returns the one that is connected by the least number of FSTs. The algorithm then tries to grow T by connecting T with combinations of FSTs at l. Let S be the set of FSTs not in T and with l as a leaf node. We know that in an OARSMT, a virtual terminal must have degree two, three, or four. Therefore, all two, three, and four combinations of the FSTs in S are added to T to form larger trees. All such expansions are added to the queue. If PASS_TEST(T) returns false, T can be eliminated and no more expansion is needed. The algorithm stops when Q is empty, which means that no tree grown from f can be in an OARSMT. We can then safely eliminate f. If at some point Q is full or all leaf nodes of T are real terminals [i.e., if ALL_REAL(T) returns true], the algorithm terminates and returns false.

Four tests are used in the function PASS_TEST(T) to eliminate useless trees. In the following, we let $V_T \subseteq V + V'$ be the set of terminals connected by T.

1) *Test* 1*:* Test 1 tries to construct a shorter tree that spans the same set of terminals.

Lemma 13: T cannot be a part of any OARSMT over V, if the length of T is larger than the length of an OARSMT over V_T.

Proof: If T is part of a Steiner minimum tree, we can replace T with the OARSMT over V_T, yielding a tree with shorter length, an absurdity. ■

To compute an OARSMT over V_T, we will include all FSTs that span exactly a subset of V_T and pass them to the FST concatenation phase. Since the computation of OARSMT over V_T can be expensive, this test is performed only when the number of terminals in T is less than a predefined number (this number is set to 30 in our implementation).

2) *Test* 2*:* Test 2 makes use of the bottleneck Steiner distance. Let $(V + V', E, c)$ be the distance graph[2] of $V + V'$, with E being the set of edges between every pair of terminals in $V + V'$ and $c : E \to \Re^+$ being a positive length function on E. A path P in the distance graph is an elementary path if both of its two endpoints are in V. The Steiner distance of P is the length of the longest elementary path in P. The bottleneck Steiner distance $s_{u,v}$ between u and v is the minimum Steiner distance over all the paths from u to v in $(V + V', E, c)$. Such a path is known as a bottleneck Steiner path.

Lemma 14: T cannot be a part of any OARSMT over V, if the length of the tree $c(T)$ is larger than the length of the minimum spanning tree over V_T in $(V + V', E, s)$ (the graph that uses distances $s_{u,v}$ as a measure of the edge weight for every pair of terminals).

Proof: It is proven in [16] and [17] that if $c(T)$ is larger than the length of the minimum spanning tree over V_T in $(V + V', E, s)$, a tree shorter than $c(T)$ spanning V_T exists in $(V + V', E, c)$. As a result, in such a case, T cannot be a part of any OARSMT. ■

3) *Test* 3*:* Test 3 compares the tree distance and the bottleneck Steiner distance between two terminals in T.

[2]A distance graph is a graph formed from a collection of points in the plane by connecting every two points by an edge, and the edge weight is equal to the distance between the two points.

Lemma 15: Let u and v be two terminals in T and $t_{u,v}$ be the length of the longest edge on the path between u and v in T. T cannot be a part of any OARSMT over V, if $t_{u,v} > s_{u,v}$.

Proof: Assume the contrary that T is in a Steiner minimum tree. Remove the longest edge on the path between u and v in T and the Steiner minimum tree is divided into two components. Along the bottleneck Steiner path between u and v, let P' be an elementary path such that its two endpoints are in different components. Note that the length of P' should be no larger than $s_{u,v}$. Therefore, we can reconnect the two components by P' yielding a shorter tree, a contradiction. ■

Note that the second and third tests both make use of the bottleneck Steiner distance between pairs of vertices. The bottleneck Steiner distance between any pair of vertices u and v in the graph $(V + V', E, c)$ can be obtained by determining the Steiner distance on the path between these two vertices in the spanning tree over V.

4) *Test 4:* Test 4 exploits the lower and upper bounds on the length of a Steiner minimum tree. To obtain the lower bound on the length of an OARSMT, one way is to solve the linear programming relaxation of the FST concatenation problem formulation as described in [14]. However, this approach is not practical due to its high computational cost. An alternative is the dual ascent heuristic proposed in [27], which is a fast heuristic that provides a lower bound for the Steiner arborescence problem in a directed graph. To apply this method, we first construct a directed graph $(V+V'+S, E_F, d)$. S is the set of all Steiner points in all FSTs. E_F is the set of directed edges that is generated by transferring each edge in an FST to its two directed versions. $d : E_F \rightarrow \Re^+$ is the edge length function. It can be easily verified that the FST concatenation problem is equivalent to finding a shortest arborescence tree in $(V+V'+S, E_F, d)$ that rooted at a terminal z and spans all the other terminals in V. As a result, we can use the dual ascent heuristic to compute the lower bound and the associated reduced cost for each edge. To compute an upper bound, the maze routing-based heuristic proposed in [7] is used. In the following, let lower be the lower bound, upper be the upper bound, $r : E_F \rightarrow \Re^+$ be the reduce cost[3] function on E_F, and $r(u, v)$ be the reduced cost distance between u and v in the graph. Let $l_1, l_2, \dots, l_k$ be the leaves of T and $\overrightarrow{T}_i$ be the directed version of T rooted at l_i. We use $r(T_i)$ to denote the reduced cost of $\overrightarrow{T}_i$.

Lemma 16: T cannot be a part of any OARSMT over V, if lower + $\min_i\{r(z, l_i) + r(T_i) + \sum_{j\neq i} \min_{v\in V-\{z\}} r(l_j, v)\} >$ upper in which z is the root node.

Proof: It is proven in [16] that lower$_{\text{constrained}}$ = lower + $\min_i\{r(z, l_i)+r(T_i)+\sum_{j\neq i} \min_{v\in V-\{z\}} r(l_j, v)\}$ is a lower bound for the length of any Steiner tree with the additional constraint that it contains T. Therefore, if lower$_{\text{constrained}}$ > upper, T cannot be a part of any OARSMT. ■

If T fails any of these four tests, PASS_TEST(T) returns false, and T can be eliminated. Note that these four tests are arranged according to their efficiencies. A test that is able to prune more trees in a shorter time will be performed first. In our current implementation, test 2 is the first one to be used, then test 3 and test 4. Test 1 is the most expensive one and it is put at the end.

[3]Finding a shortest arborescence tree in a graph can be formulated as an integer linear programming. In linear programming, the reduced cost value indicates how much the objective function coefficient on the corresponding variable must be improved before the value of the variable will be positive in the optimal solution.

Algorithm 2 Pseudocode of ObSteiner

```
Input: $V$, $O$
Output: OARSMT
1:  initialize the obstacle list $OL$ to $\emptyset$
2:  while true do
3:      FST generation
4:      FST pruning
5:      FST concatenation
6:      for all FSTs in the current solution do
7:          for all line segments in the FST do
8:              check if the line segment intersects with any
                obstacles
9:              if it intersects with obstacles then
10:                 add the dominating obstacle to $OL$
11:             end if
12:         end for
13:     end for
14:     if no overlapping obstacle exists then
15:         goto line 18
16:     end if
17: end while
18: return the OARSMT
```

C. Concatenation of FSTs

We use the same algorithm as described in [14] for the concatenation of FSTs. An integer linear programming (ILP) formulation is developed for the problem. In the ILP formulation, each FST and virtual terminal is associated with a binary variable to indicate whether it is selected as a part of the OARSMT. The objective is to minimize the total wire length, subject to a set of constraints such as the total degree constraint, the cut-set constraints, and the subtour elimination constraints. The ILP is solved via a modified version of the branch-and-cut search in [28] with lower bounds provided by the LP relaxation. The input of the algorithm is a set F of FSTs after the pruning procedure. The output is an OARSMT spanning all real terminals in V.

In the worst case, the branch-and-cut algorithm needs to explore the whole search tree, and therefore, the worst-case complexity of the branch-and-cut algorithm is exponential. Let k be the number of FSTs. The number of nodes in the search tree is then $2^{(k+1)} - 1$. Since the number of FSTs k is bounded by $O((m + e)(\frac{1+\sqrt{5}}{2})^{(m+e)})$, an upper bound for the number of nodes in the search tree is $O(2^{((m+e)(\frac{1+\sqrt{5}}{2})^{(m+e)})})$.

V. Incremental Construction

By using the two-phase approach, we can solve the OARSMT problem optimally. However, considering all obstacles together may result in a large number of virtual terminals. In our early experiments, we found that adding all obstacles simultaneously would result in an explosion of FSTs. A more

TABLE I
DETAILED RESULTS OF OBSTEINER

Benchmark	m	n	OARSMT length	t_{total} (s)	FST reduction (%)	t_{prune} (s)	$\frac{t_{\text{prune}}}{t_{\text{total}}}$ (%)	Number of iterations	$\|OL\|$	$\frac{\|OL\|}{n}$(%)
RC01	10	10	25 980	0.16	76.1	0.02	12.5	2	3	30.0
RC02	30	10	41 350	0.52	63.8	0.18	34.6	2	3	30.0
RC03	50	10	54 160	0.68	59.6	0.21	30.9	3	6	60.0
RC04	70	10	59 070	0.95	72.5	0.37	38.9	2	5	50.0
RC05	100	10	74 070	1.31	63.7	0.51	38.9	2	6	60.0
RC06	100	500	79 714	335	60.3	180	53.7	6	89	36.0
RC07	200	500	108 740	541	62.6	324	59.9	7	100	20.0
RC08	200	800	112 564	24 170	67.1	4549	18.8	7	161	20.1
RC09	200	1000	111 005	14 174	72.8	5026	35.5	7	192	19.2
RC10	500	100	164 150	176	63.7	90	51.1	5	28	28.0
RC11	1000	100	230 837	706	66.4	345	48.9	3	18	18.0
RT1	10	500	2146	25	72.0	10	40.0	6	33	6.6
RT2	50	500	45 852	31	61.3	23	74.2	5	42	8.4
RT3	100	500	7964	840	71.6	794	94.5	5	61	12.2
RT4	100	1000	9693	34 521	63.7	7939	23.0	11	197	19.7
RT5	200	2000	51 313	276 621	64.4	26 772	9.7	13	388	19.4
IND1	10	32	604	0.11	63.3	0.02	18.2	1	0	0
IND2	10	43	9500	0.25	61.4	0.05	20.0	3	5	11.6
IND3	10	50	600	0.19	68.5	0.04	21.1	2	2	3.4
IND4	25	79	1086	0.87	55.7	0.25	28.7	4	11	13.9
IND5	33	71	1341	1.03	43.9	0.27	26.2	4	14	19.7
Average					64.5		37.1			23.1

efficient way is to consider an obstacle only when it is necessary. Therefore, we adopt an incremental approach to construct an OARSMT. An obstacle list is maintained during the generation of the OARSMT. The list is responsible for keeping track of the obstacles we need to avoid during the construction. Initially, the OARSMT problem with an empty list of obstacles is solved resulting in an RSMT. We then check for obstacles that overlap with the solution. For each FST used to build the current solution, we decompose it into line segments. For each line segment, we will check whether it intersects with any obstacles. Among all overlapping obstacles we will choose the dominating one. For example, for a vertical segment, we will choose an obstacle that has the largest width. All chosen obstacles are added to the obstacle list. A new iteration then starts again by solving the OARSMT problem with the obstacles in the renewed list. This procedure repeats until no overlapping obstacle can be found. This approach is effective as in most cases only a fraction of the obstacles will affect the final OARSMT. The pseudocode of this OARSMT construction framework is shown in Algorithm 2.

VI. EXPERIMENTS

We implemented ObSteiner in C based on GeoSteiner-3.1 [29]. The experiments are conducted on a Sun Blade 2500 workstation with two 1.6-GHz processors and 2 GB memory. Our program runs sequentially on a single processor. There are 21 benchmark circuits that are commonly used as test cases for the OARSMT problem. IND1–IND5 are industrial test cases from Synopsys. RC01–RC11 are benchmarks used in [30]. RT1–RT5 are randomly generated circuits used in [5]. Note that there are overlapping obstacles in these benchmarks. We regard overlapping obstacles as one rectilinear obstacle.

TABLE II
RUN TIME OF OBSTEINER WITH AND WITHOUT THE PRUNING PROCEDURE AND THE INCREMENTAL APPROACH

Benchmark	ObSteiner w/o PN & IN	ObSteiner w/o IN	ObSteiner w/o PN	ObSteiner
RC01	0.38	0.23	0.17	0.16
RC02	0.21	0.19	0.65	0.52
RC03	0.18	0.20	0.78	0.68
RC04	0.50	0.32	0.96	0.95
RC05	0.70	0.52	1.63	1.31
RC06	>345 600	>345 600	876	335
RC07	>345 600	>345 600	1796	541
RC08	>345 600	>345 600	61 005	24 170
RC09	>345 600	>345 600	40 150	14 174
RC10	>345 600	>345 600	855	176
RC11	>345 600	>345 600	21 242	706
RT1	>345 600	>345 600	81	25
RT2	>345 600	>345 600	32	31
RT3	>345 600	>345 600	478	840
RT4	>345 600	>345 600	120 552	34 521
RT5	>345 600	>345 600	>345 600	276 621
IND1	29.88	20.78	0.13	0.11
IND2	23.25	18.92	0.27	0.25
IND3	8.78	6.07	0.18	0.19
IND4	133 852	1089	0.96	0.87
IND5	43.59	4.24	1.20	1.03
Average	8755×	1481×	3.19×	1.00×

A. Overall Performance

Table I shows the results obtained by ObSteiner. Column m provides the number of terminals in each benchmark. Column n provides the number of obstacles in each benchmark. Column t_{total} provides the total run time of the algorithm. Column t_{prune} provides the run time of the pruning procedure.

TABLE III
RESULTS OF OBSTEINER IN COMPARISON WITH THE APPROACH IN [14]

Benchmark	ObSteiner		Huang [14]		$\frac{t_2}{t_1}$
	L_1	t_1	L_2	t_2	
RC1	25 980	0.16	25 980	0.58	3.63×
RC2	41 350	0.52	41 350	0.55	1.06×
RC3	54 160	0.68	54 160	0.58	0.85×
RC4	59 070	0.95	59 070	1.10	1.16×
RC5	74 070	1.31	74 070	2.09	1.60×
RC6	79 714	335	–	–	–
RC7	108 740	541	–	–	–
RC8	112 564	24 170	–	–	–
RC9	111 005	14 174	–	–	–
RC10	164 150	176	–	–	–
RC11	230 837	706	–	–	–
RT1	2146	25	–	–	–
RT2	45 852	31	–	–	–
RT3	7964	840	–	–	–
RT4	9693	34 521	–	–	–
RT5	51 313	276 621	–	–	–
IND1	604	0.11	604	0.46	4.18×
IND2	9500	0.25	9500	3.44	13.76×
IND3	600	0.19	600	1.31	6.89×
IND4	1086	0.87	1086	3.15	3.62×
IND5	1341	1.03	1341	24.73	24.01×
RC6_40	76 946	3.20	76 946	264	82.5×
RC7_40	105 956	20	105 956	179	8.95×
RC8_30	107 833	17	107 833	495	29.12×
RC9_30	106 139	5.89	106 139	174	29.54×
RC10_30	163 050	48	163 050	1463	30.48×
RT1_40	1872	0.16	1872	1.11	6.94×
RT2_30	44 294	0.50	44 294	45	90.00×
RT3_30	7580	1.02	7580	179	179.49×
RT4_30	7825	6.05	7825	63	10.41×
RT5_30	42 879	10	42 879	40	4.00×
RC6_rand_40	76 840	3.03	76 840	538	177.56×
RC7_rand_40	105 358	14	105 358	154	11.00×
RC8_rand_30	107 811	5.55	107 811	385	69.37×
RC9_rand_30	105 875	4.44	105 875	84	18.92×
RC10_rand_30	162 470	147	162 470	733	4.99×
RT1_rand_40	1817	0.14	1817	2.02	14.43×
RT2_rand_30	44 358	0.54	44 358	23	42.59×
RT3_rand_30	7595	1.04	7595	33	31.73×
RT4_rand_30	7681	4.23	7681	64	15.13×
RT5_rand_30	42 821	5.26	42 821	97	18.44×
Average					31.08×

Column $|OL|$ provides the number of obstacles considered in the algorithm. We can see that all benchmarks are solved to optimal in a reasonable amount of time. For small benchmarks (RC01–RC05, IND1–IND5), it takes only seconds to obtain the optimal solution. For the benchmarks with less than or equal to 500 obstacles, the required time is in minutes. We can also observe that the total run time is closely related to the number of obstacles, and more obstacles usually lead to more iterations of the algorithm. In the table, we also list the average FST reduction achieved by the FST pruning procedure and the run time over all iterations. For all benchmarks, around 60% of the FSTs can be eliminated and the run time of the pruning procedure in most cases is less than half of the total time. This can greatly reduce the search space of the branch-and-cut algorithm, and therefore leads to a significant improvement in performance. The computational overhead caused by the pruning procedure is small compared to the savings in the concatenation phase. We can also observe from the table that the incremental construction is very effective. On average, only 23.1% obstacles needs to be considered. This can greatly reduce the number of additional virtual terminals and the resulting FSTs.

B. Run-Time Analysis

In order to clearly show the effectiveness of the pruning procedure and the incremental approach, we compare the run time of ObSteiner with and without these two techniques. We run each test case for 96 h (345 600 s) at most. Results are listed in Table II. In the table, >345 600 means that the solution cannot be achieved within the run-time limit. Column "ObSteiner w/o PN & IN" provides the run time of ObSteiner without both techniques. Column "ObSteiner w/o IN" provides the run time of ObSteiner without the incremental approach. Column "ObSteiner w/o PN" provides the run time of ObSteiner without the pruning procedure. Column "ObSteiner" provides the run time of ObSteiner with both techniques. To compute the average run time, for those benchmarks that cannot be finished in the time limit, we will simply use 345 600 s instead. Considering the incremental approach, we can see that, without using this technique, RC06–RC11 and RT1–RT5 will not be solvable within the run-time limit. Although for the small benchmarks with ten obstacles, the incremental approach may worsen the run time, the speedup on large benchmarks is tremendous. Considering the pruning procedure, although it is not as effective as the incremental approach, a considerable speedup can still be achieved. Without using the technique, RT5 cannot be solved within time limit. For small benchmarks, the benefit of using pruning procedure is limited. However, the technique can be very useful for difficult cases. This is because the parameters in the pruning procedure (e.g., queue size) are set according to the large benchmarks, which may not be necessary for small cases.

C. Comparison With Previous Exact Algorithm

To show the efficiency of ObSteiner, we compare our method with the approach in [14]. The results are tabulated in Table III. We execute the algorithm in [14] on our platform. Since [14] can only handle rectangular obstacles, we change the benchmarks by dissecting rectilinear obstacles into several rectangular obstacles. For completeness, we also tabulate the results of 20 additional test cases that are used in [14]. These test cases can be divided into two categories. The test cases in the first category are generated by taking the first few obstacles in the corresponding benchmarks. We use "benchmark_number" to denote them, in which "benchmark" is the original benchmark and "number" is the number

TABLE IV
COMPARISON OF HEURISTICS BASED ON THE OARSMT LENGTH

Bench	OARSMT	Wirelength				$\frac{(X-L)}{X}$ (%)				Time (s)			
mark	length (L)	[10] (A)	[7] (B)	[9] (C)	[8] (D)	$X = A$	$X = B$	$X = C$	$X = D$	[10]	[7]	[9]	[8]
RC01	25 980	26 040	25 980	25 980	26 740	0.23	0.00	0.00	2.84	0.001	0.030	0.003	<0.01
RC02	41 350	41 570	42 010	42 110	42 070	0.53	1.57	1.80	1.71	0.001	0.080	0.004	<0.01
RC03	54 160	54 620	54 390	56 030	54 550	0.84	0.42	3.34	0.71	0.002	0.080	0.008	<0.01
RC04	59 070	59 860	59 740	59 720	59 390	1.32	1.12	1.09	0.54	0.002	0.080	0.008	<0.01
RC05	74 070	74 770	74 650	75 000	75 440	0.94	0.78	1.24	1.80	0.003	0.070	0.011	<0.01
RC06	79 714	81 854	81 607	81 229	81 903	2.61	2.32	1.87	2.67	0.041	0.380	0.039	0.020
RC07	108 740	110 851	111 542	110 764	111 752	1.90	2.51	1.83	2.70	0.044	0.440	0.048	0.020
RC08	112 564	116 132	115 931	116 047	118 349	3.07	2.90	3.00	4.89	0.064	0.760	0.066	0.030
RC09	111 005	113 559	113 460	115 593	114 928	2.25	2.16	3.97	3.41	0.084	0.970	0.078	0.040
RC10	164 150	167 310	167 620	168 280	167 540	1.89	2.07	2.45	2.02	0.022	0.190	0.028	0.020
RC11	230 837	236 018	235 283	234 416	234 097	2.20	1.89	1.53	1.39	0.039	0.490	0.042	0.020
RT1	2146	2193	2231	2191	2259	2.14	3.81	2.05	5.00	0.031	0.110	0.006	0.020
RT2	45 852	47 488	47 297	48 156	48 684	3.45	3.06	4.78	5.82	0.037	0.360	0.039	0.020
RT3	7964	8231	8187	8282	8347	3.24	2.72	3.84	4.59	0.038	0.120	0.041	0.020
RT4	9693	9893	9914	10 330	10 221	2.02	2.23	6.17	5.17	0.072	0.200	0.084	0.040
RT5	51 313	52 509	52 473	54 598	53 745	2.28	2.21	6.02	4.53	0.160	1.720	0.179	0.080
IND1	604	604	619	604	626	0.00	2.42	0.00	3.51	0.002	<0.01	0.015	<0.01
IND2	9500	9600	9500	9500	9700	1.04	0.00	0.00	2.06	0.002	0.030	0.005	<0.01
IND3	600	600	600	600	600	0.00	0.00	0.00	0.00	0.002	<0.01	0.037	<0.01
IND4	1086	1092	1096	1129	1095	0.55	0.91	3.81	0.82	0.004	<0.01	0.005	<0.01
IND5	1341	1374	1360	1364	1364	2.40	1.40	1.69	1.69	0.004	<0.01	0.004	<0.01
Average						1.66	1.74	2.40	2.76				

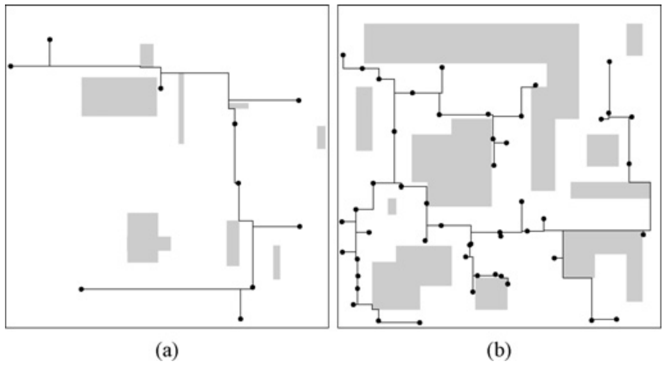

Fig. 13. OARSMTs for two benchmarks. (a) RC01. (b) Manual test case with complex rectilinear obstacles.

of obstacles taken. The test cases in the second category are generated by taking the obstacles randomly. We use "benchmark_rand_number" to denote them. In the table, "–" means that the solution cannot be achieved within the run time limit. As can be observed from the table, the run time required for the OARSMT construction has been improved a lot by our algorithm. Comparing with the approach in [14], ObSteiner can solve problems with up to 2000 obstacles, while the approach in [14] can only deal with cases with less than 100 obstacles. For small solvable cases, our approach is 31 times faster than the approach in [14] on average.

D. Comparison of Heuristics

Table IV compares the performance of some recently published heuristics [7]–[10] based on the optimal solutions provided by the proposed exact algorithm. We obtain and run the binaries from corresponding authors. We can see that all four heuristics works better on small problems (e.g., RC01–RC05, IND1–IND5), obtaining optimal solutions in several cases. However, for larger benchmarks (e.g., RC06–RC11, RT1–RT5), there is still a gap (around 2% to 3%) between the best heuristic solutions and the optimal solutions.

Fig. 13 shows two resulting OARSMTs generated by ObSteiner for two test cases.

VII. CONCLUSION

This paper presented ObSteiner, an exact algorithm for the construction of OARSMTs among complex rectilinear obstacles. The OARSMT was generated by the concatenation of FSTs among complex obstacles. We proved that by adding virtual terminals to the essential edges of obstacles, the FSTs' structures can be simplified. Therefore, they can be generated efficiently. An iterative three-phase algorithm was then proposed for the construction of OARSMTs. Between the FST generation phase and the FST concatenation phase, a pruning procedure was developed to cut down the number of required FSTs. Experiment results on several benchmarks demonstrated the effectiveness of the proposed algorithm. This is the first paper to propose a geometric approach to construct OARSMTs among complex rectilinear obstacles, and it gives key insights into this difficult problem.

ACKNOWLEDGMENT

The authors would like to thank Li *et al.* [7], Liu *et al.* [8], Ajwani *et al.* [9], and Liu *et al.* [10] for sending the executables of the corresponding papers.

References

[1] T. Huang and E. F. Y. Young, "Obstacle-avoiding rectilinear Steiner minimum tree construction: An optimal approach," in *Proc. Int. Conf. Comput.-Aided Des.*, 2010, pp. 610–613.

[2] T. Huang and E. F. Y. Young, "An exact algorithm for the construction of rectilinear Steiner minimum trees among complex obstacles," in *Proc. Des. Automat. Conf.*, 2011, pp. 164–169.

[3] M. Garey and D. Johnson, "The rectilinear Steiner tree problem is NP-complete," *SIAM J. Appl. Math.*, vol. 32, no. 4, pp. 826–834, 1977.

[4] Z. Shen, C. Chu, and Y. Li, "Efficient rectilinear Steiner tree construction with rectilinear blockages," in *Proc. Int. Conf. Comput. Des.*, 2005, pp. 38–44.

[5] C. W. Lin, S. Y. Chen, C. F. Li, Y. W. Chang, and C. L. Yang, "Obstacle-avoiding rectilinear Steiner tree construction based on spanning graphs," *IEEE Trans. Comput.-Aided Des.*, vol. 27, no. 4, pp. 643–653, Apr. 2008.

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

[7] L. Li and E. F. Y. Young, "Obstacle-avoiding rectilinear Steiner tree construction," in *Proc. Int. Conf. Comput.-Aided Des.*, 2008, pp. 523–528.

[8] C. H. Liu, S. Y. Yuan, S. Y. Kuo, and Y. H. Chou, "An $O(n \log n)$ path-based obstacle-avoiding algorithm for rectilinear Steiner tree construction," in *Proc. Des. Automat. Conf.*, 2009, pp. 314–319.

[9] G. Ajwani, C. Chu, and W. K. Mak, "FOARS: FLUTE based obstacle-avoiding rectilinear Steiner tree construction," *IEEE Trans. Comput.-Aided Des.*, vol. 30, no. 2, pp. 194–204, Feb. 2011.

[10] C. H. Liu, S. Y. Kuo, D. T. Lee, C. S. Lin, J. H. Weng, and S. Y. Yuan, "Obstacle-avoiding rectilinear Steiner tree construction: A Steiner-point-based algorithm," *IEEE Trans. Comput.-Aided Des.*, vol. 31, no. 7, pp. 1050–1060, Jul. 2012.

[11] C. Y. Lee, "An algorithm for connections and its application," *IRE Trans. Electron. Comput.*, vol. 10, no. 3, pp. 346–356, 1961.

[12] J. L. Ganley and J. P. Cohoon, "Routing a multi-terminal critical net: Steiner tree construction in the presence of obstacles," in *Proc. IEEE Int. Symp. Circuits Syst.*, Jun. 1994, pp. 113–116.

[13] L. Li, Z. Qian, and E. F. Y. Young, "Generation of optimal obstacle-avoiding rectilinear Steiner minimum tree," in *Proc. Int. Conf. Comput.-Aided Des.*, 2009, pp. 21–25.

[14] T. Huang, L. Li, and E. F. Y. Young, "On the construction of optimal obstacle-avoiding rectilinear Steiner minimum trees," *IEEE Trans. Comput.-Aided Des.*, vol. 30, no. 5, pp. 718–731, May 2011.

[15] D. M. Warme, P. Winter, and M. Zachariasen, "Exact algorithms for plane Steiner tree problems: A computational study," in *Advances in Steiner Trees*, D. Z. Du, J. M. Smith, and J. H. Rubinstein, Eds. Boston, MA: Kluwer Academic, 2000, pp. 81–116.

[16] T. Polzin, "Algorithms for the Steiner problem in networks," Ph.D. dissertation, Univ. Saarlandes, Saarbrücken, Germany, 2003.

[17] C. Duin, "Preprocessing the Steiner problem in graph," in *Advances in Steiner Trees*, D. Z. Du, J. M. Smith, and J. H. Rubinstein, Eds. Boston, MA: Kluwer Academic, 2000, pp. 173–233.

[18] Y. F. Wu, P. Widmayer, M. D. F. Schlag, and C. K. Wong, "Rectilinear shortest paths and minimum spanning trees in the presence of rectilinear obstacles," *IEEE Trans. Comput.-Aided Des.*, vol. 36, no. 3, pp. 321–331, Mar. 1987.

[19] C. D. Yang, D. T. Lee, and C. K. Wong, "Rectilinear path problems among rectilinear obstacles revisited," *IEEE Trans. Comput.-Aided Des.*, vol. 24, no. 3, pp. 457–472, Jun. 1995.

[20] F. K. Hwang, D. S. Richards, and P. Winter, *The Steiner Tree Problem*. Amsterdam, The Netherlands: Elsevier, 1992, no. 53.

[21] F. K. Hwang, "On Steiner minimal trees with rectilinear distance," *SIAM J. Appl. Math.*, vol. 30, no. 1, pp. 104–114, 1976.

[22] M. Zachariasen, "A catalog of hanan grid problems," *Networks*, vol. 38, no. 2, pp. 76–83, 2001.

[23] Y. Hu, Z. Feng, T. Jing, X. Hong, Y. Yang, G. Yu, X. Hu, and G. Yan, "FORst: A 3-step heuristic for obstacle-avoiding rectilinear Steiner minimal tree construction," *J. Informat. Comput. Sci.*, vol. 1, no. 3, pp. 107–116, 2004.

[24] Y. Shi, P. Mesa, H. Yao, and L. He, "Circuit simulation based obstacle-aware Steiner routing," in *Proc. Asia South Pacific Des. Automat. Conf.*, 2006, pp. 385–388.

[25] M. Zachariasen, "Rectilinear full Steiner tree generation," *Networks*, vol. 33, no. 2, pp. 125–143, Mar. 1999.

[26] J. L. Ganley and J. P. Cohoon, "Optimal rectilinear Steiner minimal trees in $O(n^2 2.62^n)$ time," in *Proc. Canad. Conf. Comput. Geometry*, 1994, pp. 308–313.

[27] R. T. Wong, "A dual ascent approach for Steiner tree problems on a directed graph," *Math. Programming*, vol. 28, no. 3, pp. 271–287, 1984.

[28] D. M. Warme, "Spanning trees in hypergraphs with applications to Steiner trees," Ph.D. dissertation, Comput. Sci. Dept., Univ. Virginia, Charlottesville, 1998.

[29] *GeoSteiner − Software for Computing Steiner Trees*. (2003) [Online]. Available: http://www.diku.dk/geosteiner/

[30] Z. Feng, Y. Hu, T. Jing, X. Hong, X. Hu, and G. Yan, "An $O(n \log n)$ algorithm for obstacle-avoiding routing tree construction in the λ-geometry plane," in *Proc. Int. Symp. Phys. Des.*, 2006, pp. 48–55.

Tao Huang received the B.Eng. and M.Eng. degrees in electronic engineering from Sun Yat-Sen University, Guangzhou, China, in 2007 and 2009, respectively. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, Hong Kong.

His current research interests include computer-aided design of very large scale integration circuits, physical design, interconnect optimization, and combinatorial optimization.

Mr. Huang has won the Championship in the International Symposium on the Physical Design (ISPD) 2011 Routability-Driven Placement Contest, the First Runner-Up in the Design Automation Conference 2012 Routability-Driven Placement Contest, the First Runner-Up in the IEEE/ACM International Conference on Computer-Aided Design 2012 Routability-Driven Placement Contest, and the First Runner-Up in the ISPD 2010 High-Performance Clock Network Synthesis Contest.

Evangeline F. Y. Young received the B.Sc. and M.Phil. degrees in computer science from the Chinese University of Hong Kong (CUHK), Shatin, Hong Kong, and the Ph.D. degree from the University of Texas, Austin, in 1999.

She is currently a Professor with the Department of Computer Science and Engineering, CUHK. Her current research interests include algorithms and computer-aided design of very large scale integration circuits. She is engaged actively in floorplanning, placement, routing, and algorithmic designs.

Dr. Young has served on the program committees of several major conferences, including DAC, ICCAD, ASP-DAC, ISPD, DATE, and GLSVLSI. She has also served on the editorial boards of the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems and *Integration, the VLSI Journal*.