An Exact Algorithm for the Construction of Rectilinear Steiner Minimum Trees among Complex Obstacles

Tao Huang
Dept. of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, NT, Hong Kong
thuang@cse.cuhk.edu.hk

Evangeline F. Y. Young
Dept. of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, NT, Hong Kong
fyyoung@cse.cuhk.edu.hk

ABSTRACT

In this paper, we present an exact algorithm for the construction of obstacle-avoiding rectilinear Steiner minimum trees (OARSMTs) among complex rectilinear obstacles. This is the first work 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 (FSTs) among complex obstacles, which are proven to be of simple structures in this paper. The algorithm 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.

Categories and Subject Descriptors

B.7.2 [**Hardware**]: Integrated Circuits—*Design Aids*

General Terms

Algorithms, Theory, Design

Keywords

Rectilinear Steiner Minimum Tree, Obstacle-avoiding, Full Steiner Tree, Pruning

1. INTRODUCTION

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). Modern VLSI designs often contain rectilinear obstacles such as macro cells, IP blocks, and pre-routed 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 [1], 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 [2, 3, 4] try to dissect a rectilinear obstacle into several rectangular obstacles. However, the cutting lines on the rectilinear obstacles can allow wires to go through which may result in an infeasible solution. The maze-routing approach proposed by Li *et al.* [5] 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.* [6] proposed a new framework to generate OARSMT based on the critical paths, which guarantees the existence of desirable solutions. Recently, Ajwani *et al.* [7] proposed a fast lookup table based algorithm to route a multi-terminal 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.

In terms of exact algorithms, Ganley *et al.* [8] proposed a strong connection graph called escape graph for the OARSMT problem and prove 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 [9] and [10] extended GeoSteiner [11] to an obstacle aware version. Their algorithms are able to generate optimal OARSMTs for multi-terminal 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. 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 to a graph problem which can be solved optimally by using some graph based algorithms [12, 13], these approaches are believed to be less efficient than the geometric approaches for solving the geometric Steiner tree problem. An example is GeoSteiner [11], which remains to be 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 to construct OARSMTs among rectilinear obstacles including both convex and concave polygon obstacles. 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. Our work has the following contributions:

1. This is the first work to propose an geometric approach to exactly solve the OARSMT problem when there are complex rectilinear obstacles. By using the proposed algorithm,

DAC '11, Jun 05-10 2011, San Diego, California, USA

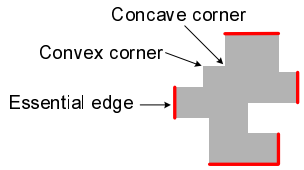

Figure 1: Corners and essential edges within an obstacle.

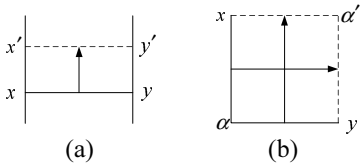

Figure 2: Two operations on a rectilinear Steiner tree. (a) Shifting and (b) Flipping.

we report optimal solutions to several benchmarks which are commonly used in the literature. Some of them are the first known optimal solutions. This work provides key insights into this difficult problem.

2. We observe that reduction techniques can be very powerful for the OARSMT problem, and hence design an efficient pruning procedure which can greatly increase the largest solvable problem size.

The rest of this paper is organized as follows. In Section 2, we give preliminaries on the problem. In Section 3, we study the structures of FSTs among complex obstacles. Section 4 describes the exact algorithm in detail. Experiment results are presented in Section 5, followed by a conclusion in Section 6.

2. PRELIMINARIES

2.1 Problem formulation

In a two-dimensional routing region, we are given a set V of terminals and a set O of obstacles.

Definition 1. 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*.

As shown in Figure 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 degree angle inside the polygon, the corner is called a *convex corner*. Otherwise, if the two neighboring edges of a corner form a 270 degree angle inside the polygon, the corner is called a *concave corner*. If both end points of an edge are convex corners, this edge is called an *essential edge* (Figure 1).

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.

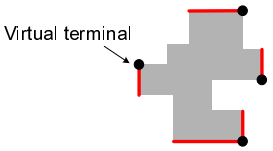

Figure 3: Virtual terminals added to an obstacle.

2.2 Equivalent trees

As define in [14], there are two basic operations on a tree that will not change the total length: *Shiftings* and *Flippings*, as shown in Figure 2. 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.

2.3 Full Steiner tree

A full Steiner tree (FST) is a rectilinear Steiner minimum tree in which every terminal is a leaf node (of degree one). Any Steiner minimum tree can be decomposed into a set of edge-disjoint FSTs by splitting at terminals with degree more than one. In the absence of obstacles, a FST has one of the two generic forms as described in [14]. Since FSTs are much easier to construct than Steiner minimum trees, most of the exact algorithms for the construction of a Steiner minimum tree will first generate its FST components.

2.4 GeoSteiner

GeoSteiner is a software package for computing Steiner minimum trees. The algorithm makes use of a two-phase approach, consisting of a FST generation phase and a FST concatenation phase, to construct a Steiner minimum tree. In the first phase, a set of FSTs are generated such that there is at least one Steiner minimum tree composed of the FSTs in the set only. In the second phase, a subset of FSTs are selected and combined to form a Steiner minimum tree. 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.

3. FULL STEINER TREES AMONG COMPLEX OBSTACLES

To construct OARSMTs among complex obstacles, we first study the FSTs among complex obstacles. The structures of these FSTs can be complicated due to the existence of rectilinear obstacles. Therefore, virtual terminals are added to simplify their structures in this paper. It should be noted that although virtual terminals are also used in [9, 10], there are critical differences when dealing with rectilinear obstacles. 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. For simplicities, we assume the virtual terminal on an essential edge is located at one of its end points. An example is shown in Figure 3. Note that for two essential edges sharing a common endpoint at a corner, we only need to add one virtual terminal at this corner. We use V' to denote the set of virtual terminals we added. A FST among complex obstacles is defined as follows.

Definition 2. A 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 (has degree one) in f and all its equivalent trees. Moreover, all the equivalent trees of f cannot contain forbidden edges that pass through a virtual terminal. Otherwise, we can split this FST into smaller FSTs at this virtual terminal.

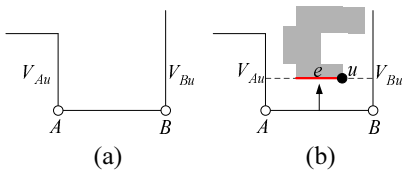

Figure 4: Two adjacent Steiner points in a FST.

Note that the definition of FST in this paper is similar to the definitions in [9, 10]. However, the obstacles considered in this paper are rectilinear polygons, which are more general and complicated than the rectangles considered in [9, 10]. If all obstacles are rectangles, the FSTs defined in this paper degenerate to the ones as defined in [10].

To derive the structures of a FST, we mainly follow the steps as described in [14]. The main difference is that there can be rectilinear obstacles in the routing region. For the two operations (shiftings and flippings) used in the proofs, it is possible that some of the operations cannot be done due to obstacles. We will show in the following how this problem can be solved by adding virtual terminals.

In the following lemmas, we use V_{xu} (V_{xd}) to denote the vertical line at point x which extends above (below) x excluding x itself. We use H_{xr} (H_{xl}) denotes the horizontal line at point x which extends to the right (left) of x excluding x itself. In the following figures, we use a solid circle to denote a terminal (real or virtual) and an empty circle to denote a Steiner point.

LEMMA 1. *All Steiner points in a FST either have degree three or degree four.*

LEMMA 2. *Let A and B be two adjacent Steiner points in a 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 Figure 4(a). Suppose A is to the left of B.

(i) V_{Au} contains no terminal on it, 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 Figure 4(b), we can shift AB up until it overlaps with an edge e of the obstacle. According to the definition, the two endpoints of e are convex corners. Therefore, 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.

(ii) 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.

(iii) 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.

(iv) 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 overlap with an edge of the obstacle and this edge must be a essential edge), the line will pass through a virtual terminal, an absurdity. □

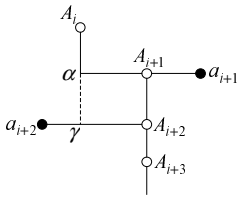

Figure 5: A structure of Steiner chain when A_i and A_{i+1} are connected by a corner.

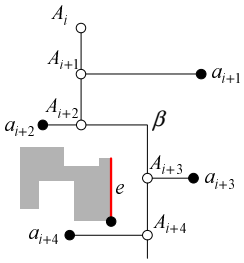

Figure 6: A structure of Steiner chain when the corner between A_{i+2} and A_{i+3} can not be flipped due to obstacles.

The rest of the lemmas are analogous to the lemmas in [14]. In the proofs, we can use similar method as illustrated in the proof of Lemma 2 to deal with possible obstacles. The only exception is Lemma 10 in [14], in which flippings are required for the proof. In this paper, a slightly different lemma is used, which we refer to as Lemma 3.

LEMMA 3. *Let A_i be the i^{th} Steiner point on the Steiner chain of a 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 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 8 and Lemma 9 in [14], when A_i and A_{i+1} are connected by a corner, the graph must be the one given in Figure 5. 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 β. 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} exist, the graph must be the one given in Figure 6, in which there are some obstacles inside the bounded rectangular defined by A_{i+2} and A_{i+3}. We use β to denote the corner connecting A_{i+2} and A_{i+3}. Shift βA_{i+4} to the left until it meets an edge e on one of the obstacle inside the rectangular region. The same as 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 pass through a virtual terminal, a contradiction. If shifting βA_{i+4} meet 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.

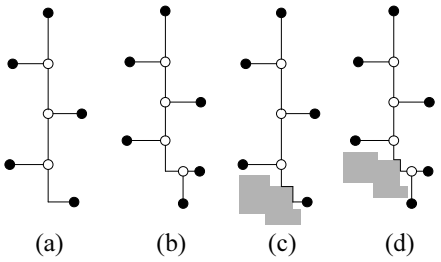

Figure 7: Possible structures of a FST among complex obstacles.

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

By pushing the corners along the direction according to Lemma 3, there will be at most one corner connecting the last two Steiner points. Summing up all the lemmas, the structures of a FST must be in one of the four forms as shown in Figure 7. The first two structures are exactly the same as the structures of the FSTs in the absence of obstacles. The last two structures differ from the first two in that the last corner may be blocked by some obstacles.

An OARSMT can be partitioned into a set of FSTs by splitting at real terminals or virtual terminals of degree more than one. 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.

4. OARSMT CONSTRUCTION

Similar to [10], the proposed algorithm uses the following three phases to construct an OARSMT.

1. FST generation: Generate a set of FSTs.
2. FST pruning: Use some necessary conditions to eliminate a fraction of the FSTs generated in the previous phase.
3. FST concatenation: Combine a subset of FSTs survived in the previous phase to construct an OARSMT.

Since the number of obstacles can be very large, we do not include all obstacles and finish the algorithm in one round. Instead, an iterative approach is adopted. The algorithm starts by solving the OARSMT problem with an empty set of obstacles. In the subsequent iterations, an obstacle will be taken into consideration if it intersects with the OARSMTs generated in previous iterations. This process repeats until no overlapping obstacle can be found. The resulting OARSMT in the final iteration is then an optimal solution to the problem.

4.1 Generation of FSTs

We modify the FST generation approach in [10] to generate the FSTs defined in this paper. To generated the FSTs spanning three or more terminals, a modified version of the recursive algorithm in [15] 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. Implementation details are omitted due to the limitation of space. The result of the FST generation phase is a set of FSTs denoted by F.

Algorithm PRUNE(f)
Input: f
Output: $true$ or $false$
1: initialize a queue Q
2: push $f \rightarrow Q$
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** $T' \in \begin{pmatrix} S \\ 1 \end{pmatrix} \cup \begin{pmatrix} S \\ 2 \end{pmatrix} \cup \begin{pmatrix} S \\ 3 \end{pmatrix}$ **do**
12: push $T \cup T' \rightarrow Q$
13: **if** Q is full **then**
14: **return** $false$
15: **end if**
16: **end for**
17: **end if**
18: **end while**
19: **return** $true$

Figure 8: Pseudocode of the pruning algorithm

4.2 Pruning of FSTs

We propose an efficient pruning procedure to reduce the number of FSTs needed to construct an OARSMT. The pruning algorithm works by growing a FST f to larger trees and test if these larger trees can exist in the optimal solution. We know that virtual terminals in the OARSMT must have degree two, three or four. Therefore, it is possible to grow a tree at a leaf node which is a virtual terminal. The rationale behind is that a FST f can be eliminated if no tree grew from f can exist in an OARSMT. The pseudocode of the pruning algorithm is shown in Figure 8.

The input of the function PRUNE(f) is a FST f. The function returns a value $true$ 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 compute 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. 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 grew 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, 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.

The first test tries to construct a shorter tree that spans the same set of terminals.

LEMMA 4. *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 in a Steiner minimum tree, we can replace T with the OARSMT over V_T, yielding a tree with shorter length, an absurdity. □

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).

The second test make use of the bottleneck Steiner distance. Let $(V + V', E, c)$ be the distance graph of $V + V'$, with E being the set of edges between every pair of terminals in $V + V'$ and $c : E \rightarrow \Re^+$ 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 5. *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 on $(V + V', E, s)$ (the graph that uses distances $s_{u,v}$ as a measure of the edge weight for every pair of terminals).*

PROOF. The proof for this lemma is omitted due to space limitation. A proof for a similar but simpler lemma can be found in [13]. □

The third test compares the tree distance and the bottleneck Steiner distance between two terminals in T.

LEMMA 6. *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. □

The fourth test 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 [10]. However, this approach is not practical due to the fact of its high computational cost. An alternative choice is the dual ascent heuristic proposed in [16], 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 FST. E_F is the set of directed edges which is generated by transferring each edge in a FST to its two directed versions. $d : E_F \rightarrow \Re^+$ is the edge length function. It can be 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 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 [5] 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 function on E_F, and $r(u, v)$ be the reduced cost distance between u and v in the graph. Let $l_1, l_2, \ldots, l_k$ 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 $\overrightarrow{T}_i$.

Table 2: Results of the exact algorithm

Bench mark	m	n	FST reduction (%)	Pruning time (s)	Iteration number	OARSMT length	Total time (s)
RC01	10	10	76.1	<1	2	25980	<1
RC02	30	10	63.8	<1	2	41350	<1
RC03	50	10	59.6	<1	3	54160	<1
RC04	70	10	72.5	<1	2	59070	<1
RC05	100	10	63.7	<1	2	74070	1
RC06	100	500	60.3	180	6	79714	335
RC07	200	500	62.6	324	7	108740	541
RC08	200	800	67.1	4549	7	112564	24170
RC09	200	1000	72.8	5026	7	111005	14174
RC10	500	100	63.7	90	5	164150	176
RC11	1000	100	66.4	345	3	230837	706
RT1	10	500	72.0	10	6	2146	25
RT2	50	500	61.3	23	5	45852	31
RT3	100	500	71.6	794	5	7964	840
RT4	100	1000	63.7	7939	11	9693	34521
RT5	200	2000	64.4	26772	13	51313	276621
IND1	10	32	63.3	<1	1	604	<1
IND2	10	43	61.4	<1	3	9500	<1
IND3	10	59	68.5	<1	2	600	<1
IND4	25	79	55.7	<1	4	1086	1
IND5	33	71	43.9	<1	4	1341	1

LEMMA 7. *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. The proof for this lemma is omitted due to space limitation. A proof for a similar lemma can be found in [12]. □

If T fails any of these four tests, PASS_TEST(T) returns $false$, and T can be eliminated.

4.3 Concatenation of FSTs

We use the same algorithm as described in [10] for the concatenation of FSTs. An integer linear programming (ILP) formulation is developed for the problem. The ILP is solved via a modified version of the branch-and-cut search in [17] 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.

5. EXPERIMENTS

We implemented our algorithm in C based on GeoSteiner-3.1 [11]. The experiments are conducted on a Sun Blade 2500 workstation with two 1.6GHz processors and 2GB memory. Our program runs sequentially on a single processor. There are 21 benchmark circuits which are commonly used as test cases for the OARSMT problem. IND1-IND5 are industrial test cases from Synopsys. RC01-RC11 are benchmarks used in [18]. RT1-RT5 are randomly generated circuits used in [3]. Note that there are overlapping obstacles in these benchmarks. We regard overlapping obstacles as one rectilinear obstacle.

Table 2 shows the results of the proposed exact algorithm. Column "m" provides the number of terminals in each benchmark. Column "n" provides the number of obstacles in each benchmark. 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

Table 1: Comparison of heuristics based on the OARSMT length

Benchmark	m	n	OARSMT length (L)	Wirelength			$\frac{(X-L)}{X}$ (%)		
				Li[5] (A)	Ajwani[7] (B)	Liu[6] (C)	$X=A$	$X=B$	$X=C$
RC01	10	10	25980	25980	25980	26740	0.00	0.00	2.84
RC02	30	10	41350	42010	42110	42070	1.57	1.80	1.71
RC03	50	10	54160	54390	56030	54550	0.42	3.34	0.71
RC04	70	10	59070	59740	59720	59390	1.12	1.09	0.54
RC05	100	10	74070	74650	75000	75430	0.78	1.24	1.80
RC06	100	500	79714	81607	81229	81903	2.32	1.87	2.67
RC07	200	500	108740	111542	110764	111752	2.51	1.83	2.70
RC08	200	800	112564	115931	116047	118349	2.90	3.00	4.89
RC09	200	1000	111005	113460	115593	114928	2.16	3.97	3.41
RC10	500	100	164150	167620	168280	167540	2.07	2.45	2.02
RC11	1000	100	230837	235283	234416	234097	1.89	1.53	1.39
RT1	10	500	2146	2231	2191	2259	3.81	2.05	5.00
RT2	50	500	45852	47297	48156	48684	3.06	4.78	5.82
RT3	100	500	7964	8187	8282	8347	2.72	3.84	4.59
RT4	100	1000	9693	9914	10330	10221	2.23	6.17	5.17
RT5	200	2000	51313	52473	54598	53745	2.21	6.02	4.53
IND1	10	32	604	619	604	626	2.42	0.00	3.51
IND2	10	43	9500	9500	9500	9700	0.00	0.00	2.06
IND3	10	59	600	600	600	600	0.00	0.00	0.00
IND4	25	79	1086	1096	1129	1095	0.91	3.81	0.82
IND5	33	71	1341	1360	1364	1364	1.40	1.69	1.69
Average							1.74	2.40	2.76

time is in minutes. We can also observe that the total running 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 running time over all iterations. For all benchmarks, around 60% of the FSTs can be eliminated and the running 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 cause by the pruning procedure is small compared to the savings in the concatenation phase.

Table 1 compares the performance of some recently published heuristics [5, 6, 7] based on the optimal solutions provided by the proposed exact algorithm. The results are quoted from the corresponding papers. We can see that all three heuristics works better on small problems, obtaining optimal solutions in several cases. The performance gradually decreases with the increasing number of obstacles.

6. CONCLUSIONS

An exact algorithm for the construction of OARSMTs among complex rectilinear obstacles is presented in this paper. The OARSMT is generated by the concatenation of FSTs among complex obstacles. We prove that by adding virtual terminals to the essential edges of obstacles, the FSTs structures can be greatly simplified. Therefore, they can be generated efficiently. We also develop a pruning procedure as a problem reduction method to cut down the number of required FSTs. Experiment results on several benchmarks demonstrate the effectiveness of the proposed algorithm. This is the first work to propose a geometric approach to construct OARSMTs among complex rectilinear obstacles and it gives key insights into this difficult problem.

7. REFERENCES

[1] M. Garey and D. Johnson, "The rectilinear steiner tree problem is np-complete," *SIAM Journal of Applied Mathematics*, pp. 826–834, 1977.

[2] Z. Shen, C. Chu, and Y. Li, "Efficient rectilinear steiner tree construction with rectilinear blockages," in *Proceedings ICCD*, pp. 38–44, 2005.

[3] C. W. Lin, S. Y. Chen, C. F. Li, Y. W. Chang, and C. L. Yang, "Efficient obstacle-avoiding rectilinear steiner tree construction," in *Proc. Int. Symp. Phys. Des.*, pp. 380–385, 2007.

[4] J. Y. Long, H. Zhou, and S. O. Memik, "Eboarst: An efficient edge-based obstacle-avoiding rectilinear steiner tree construction algorithm," *IEEE Transaction on Computer-Aided Design*, vol. 27, no. 12, pp. 2169–2182, 2008.

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

[6] 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. of DAC*, pp. 314–319, 2009.

[7] G. Ajwani, C. Chu, and W. K. Mak, "FOARS: FLUTE Based Obstacle-Avoiding Rectilinear Steiner Tree Construction," in *Proc. Int. Symp. Phys. Des.*, pp. 27–34, 2010.

[8] J. L. Ganley and J. P. Cohoon, "Routing a multi-terminal critical net: Steiner tree construction in the presence of obstacles," in *Proc. of IEEE ISCAS*, (London, UK).

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

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

[11] GeoSteiner — software for computing Steiner trees. http://www.diku.dk/geosteiner/.

[12] T. Polzin, *Algorithms for the Steiner Problem in Networks*. PhD thesis, Universität des Saarlandes, 2003.

[13] C. Duin, "Preprocessing the steiner problem in graph," in *Advances in Steiner Trees* (D. Z. Du, J. M. Smith, and J. H. Rubinstein, eds.), pp. 81–116, Boston: Kluwer Academic Publishers, 2000.

[14] F. K. Hwang, "On steiner minimal trees with rectilinear distance," *Proceedings SIAM Journal on Applied Mathematics*, vol. 30, pp. 104–114, 1976.

[15] M. Zachariasen, "Rectilinear full steiner tree generation," *Networks*, vol. 33, pp. 125–143, 1999.

[16] R. T. Wong, "A dual ascent approach for steiner tree problems on a directed graph," *Mathematical Programming*, vol. 28, pp. 271–287, 1984.

[17] D. M. Warme, *Spanning Trees in Hypergraphs with Applications to Steiner Trees*. PhD thesis, Computer Science Dept., The University of Virginia, 1998.

[18] 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.*, pp. 48–55, 2006.