Timing-Driven X-Architecture Routing Tree Construction Among Rectangular and Non-Rectangular Obstacles

Shu-Ping Chang[1], Hsin-Hsiung Huang[2], Cheng-Chiang Lin[3] and Tsai-Ming Hsieh[3]
[1]Dept.of CAD, Genesys Logic Company, Taipei Country, Taiwan, R.O.C.
[2]Dept. of Electronic Engineering, Lunghwa Univ. of Science and Technology, Taoyuan, Taiwan, R.O.C.
[3]Dept. of Information and Computer Engineering, Chung Yuan Christian University, Chung-Li, Taiwan, R.O.C.
Email: Patty.Chang@genesyslogic.com.tw; pp022@mail.lhu.edu.tw; matrix_lin2002@yahoo.com; hsieh@cycu.edu.tw

Abstract—**In this paper, we formulate a new X-architecture routing problem in presence of non-rectangular obstacles, and propose an X-architecture timing-driven routing algorithm to minimize the maximum source-to-sink delay and the total wirelength simultaneously. First, a spanning graph is constructed by the terminals and the corners of the obstacles. A minimal spanning tree is then produced by performing searching algorithm to the spanning graph. The feasible X-architecture is constructed by transforming all slant edges of the minimal spanning tree. For the initial X-architecture routing tree, the delay of source-to-terminal is estimated by the modified Elmore delay model. According to the user defined delay threshold, an efficient rerouting algorithm is used to fix the timing violated nets. The critical terminals iteratively are rerouted by splitting two sub-trees and merging into one tree. Compared to the routing result without rerouting, the maximum source-to-sink delay is improved by 49.1% and only 2.5% of additional total wirlength is increased.**

Key-Words: Timing-driven, Non-rectangular obstacle, A-shaped pattern routing, Routing, X-architecture.

1 Introduction

Routing plays a very important role in physical design such as the synthesis [14], placement [3] and routing [9] stages, since it is a very complex step in nanometer IC. Traditional routers are designed for the Manhattan-based architecture (M-based for short). Modern routers present various challenges, including the novel X-based architecture (X-based for short), the obstacles in SOC design and the serious timing issues.

Construction for Steiner minimal tree is proved to be NP-hard, and many heuristic algorithms are developed to reduce the wirelength for X-architecture [17]. Teig [18] implemented a Toshiba microprocessor by using the octagonal technique, reducing the total wirelength, via count and die size, by 20%, 40% and 11%, respectively. Coulston [2] presented a two-step algorithm that generates all possible full components and merges them into an optimal tree. Kahng et al. [10] presented the wirelength-driven heuristic algorithm that adopts the batched-based triple contraction with complexity $O(mlg^2m)$. Chiang et al. [1] presented the octilinear Steiner tree by using the edge-conversion and Steiner-sliding technique. In contrast to indirect transformation, some graph-based methods have been presented to build routing trees directly [5][6][16][20]. But, some cannot handle the obstacles and timing issues.

Many IPs, macros and routed paths are involved in SOC design. Modern routers must handle obstacles in routing. Many algorithms have presented spanning- graph based methods to build routing trees with rectangular obstacles [12][16][19]. Feng et al. [5] presented a three-step method to construct an obstacle-avoiding routing tree that minimizes the total wirelength under the λ-Geometry plane with the non-rectangular obstacles. Some algorithms cannot deal with non-rectangular obstacles, and some algorithms do not consider the timing issues.

Many algorithms that consider timing issues have been presented for nanometer VLSI design. Some algorithms attempt to reduce detours made during routing. These detours result in the large maximum source-to-terminal delays [8][11][15]. Some algorithms have been presented to adjust the initial routing tree based on timing constraints [13]. Hu et al. [7] presented the concept of soft edge to move the Steiner points, and Lin et al. [13] proposed a recalling function to update the critical paths. However, some algorithms cannot manage the routing with obstacles. Some cannot construct the timing-driven initial tree, and only reduce the delay by passively adjusting the tree.

This study makes the following major contributions. First, the rectangular and non-rectangular obstacle-avoiding X-based Steiner minimal tree problem is formulated and solved by a new algorithm. To the best of our knowledge, no existing literature discusses X-based timing-driven trees with non-rectangular obstacles. Second, the effective rerouting method, which splits violated terminals and merges two subtrees, is proposed to enhance the timing of the violated terminals. Experimental results indicate that if the initial routing tree is not optimal, then rerouting not only minimizes the delay but also reduces the wirelength.

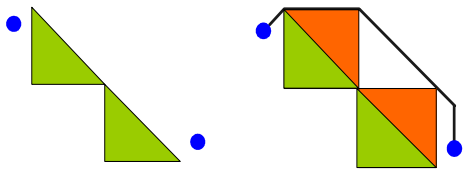


Fig.1 Drawback of the traditional method.

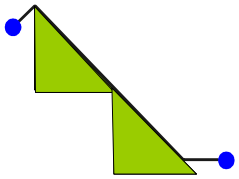


Fig. 2 Non-rectangular obstacle-avoiding routing.

Third, non-rectangular obstacles are handled by superimposing a bounding rectangle for the non-rectangular obstacles. The concept of virtual nodes is presented to further reduce the wirelength. An extension is shown to reveal that the proposed method handle obstacles with any geometric shapes.

The remainder of this paper is organized as follows. Section 2 describes the problem domain, including the motivation and problem definition. Section 3 presents the proposed algorithm. Experimental results are shown in section 4. Finally, conclusions are drawn in Section 5.

2 Preliminary

This section describes the motivation to study the importance of non-rectangular obstacles. For the X-based routing tree, the maximum source-to-target delay is calculated and the delay threshold is defined as the timing constraint for rerouting. Finally, the rectangular obstacle-avoiding X-based Steiner minimal tree (ROA-XSMT) and rectangular and non-rectangular obstacle-avoiding X-based Steiner minimal tree (NOA-XSMT) problems are formulated.

2.1 Motivation

A routed path of an X-architecture for terminals and obstacles becomes a non-rectangular obstacle for the un-routed paths, as shown in Fig. 1(a). Hence, to minimize the wirelength and delay, the modern router must handle routing with non-rectangular obstacles. No routers for X-architectures have yet been reported in the literature. A router generally treats a non-rectangular obstacle as a rectangular obstacle. However, this approach does not produce the optimal wirelength or delay, as indicated in Fig. 1(b). For the Fig. 1(a), the traditional router, which treats a non-rectangular obstacle as a rectangular obstacle, has the results in Fig. 1(b) but the total wire length is long. However, the short total wirelength in Fig. 2 is obtained by the modern router which can handle routing with non-rectangular obstacles. We observe that the algorithms which can handle the routing with non-rectangular are necessary. This observation indicates that a router that can handle non-rectangular obstacles minimizes the wirelength and the maximum source-to-terminal delay.

2.2 Problem Definition

Due to the lack of timing constraints, the delay threshold is given by $D_{\max} \times r$, where $D_{\max}$ and r represent the maximum source-to-terminal delay and user-defined ratio ($0 \le r \le 1$), respectively. A larger (smaller) ratio denotes looser (tighter) timing constraints. Clearly, more timing violations occur when a tighter timing constraint is applied. Due space limitations in this article, only the result of r=0.7 is addressed herein. The critical nodes with delays over the threshold are rerouted them to further reduce the delay. Rerouting is further discussed in Section 3.4. The total wirelength (abbreviated as L_{tal}) is increased after performing the rerouting algorithm.

This study formulates two problems, and develops algorithms to solve them. The first ROA-XSMT problem is defined as follows:

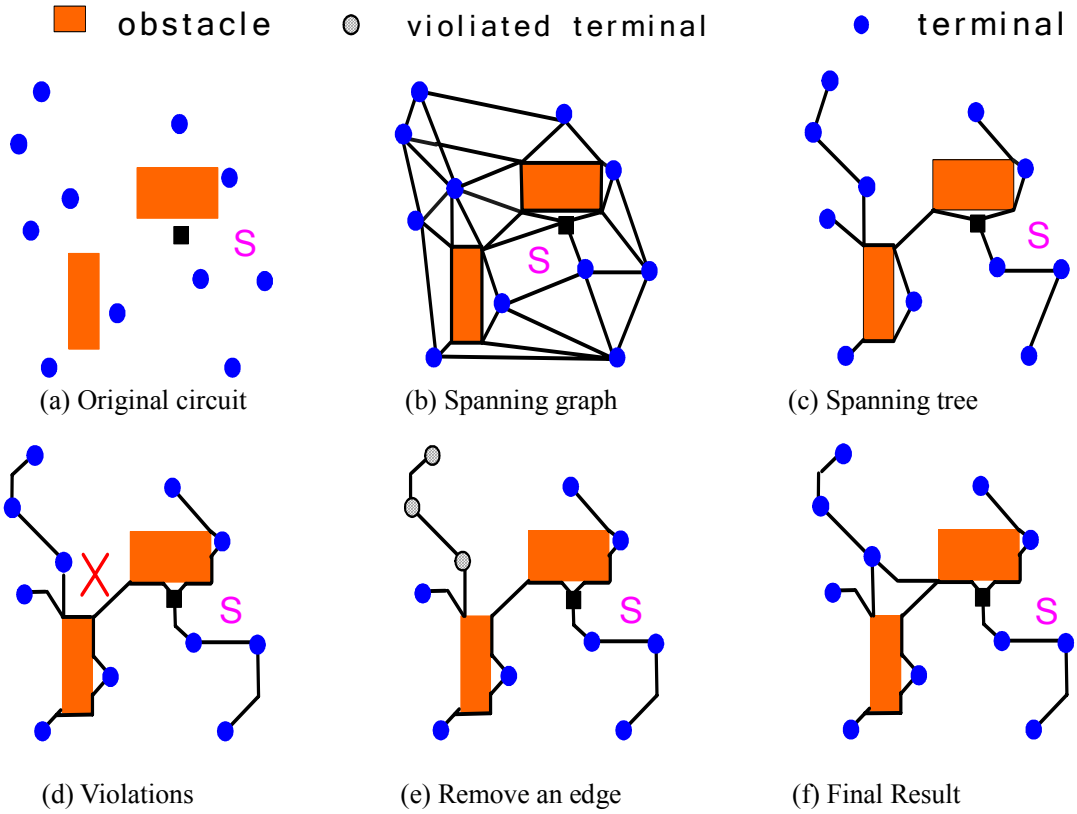
Given a set of obstacles and a set of terminals with the source under the given delay threshold, the objective of the paper is to construct the timing-driven X-architecture routing tree that minimizes the $D_{\max}$ among the rectangular obstacles.

As well as rectangular obstacles, circuits also contain some non-rectangular obstacles. The differences between these types of obstacles are discussed in Sub-section 3.2. The second NOA-XSMT problem is defined as follows:

Given a set of rectangular and non-rectangular obstacles, and a set of terminals with the source under the user defined delay threshold, the objective of the paper is to minimize the $D_{\max}$ among all obstacles.

3 Timing-Driven X-based Routing Tree with Obstacles

This section describes two algorithms for the routing with obstacles. Algorithm1 in Sub-section 3.1 is directly applied for the rectangular obstacles and Algorithm1 reroutes the initial tree to improve the $D_{\max}$. Algorithm2, which is applied to both rectangular and non-rectangular obstacles, consists of three steps, namely spanning graph construction (Sub-section 3.2), transformation for X-based routing tree (Sub-section 3.3) and rerouting all violated terminals (Sub-section 3.4).



(a) Original circuit (b) Spanning graph (c) Spanning tree

(d) Violations (e) Remove an edge (f) Final Result

Fig. 3 An X-based tree with the *rectangular* obstacles.

3.1 Routing Tree with Rectangular Obstacles

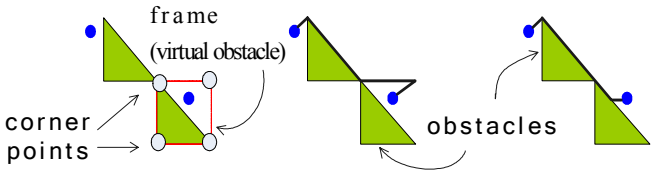
The X-based routing tree is constructed using a spanning graph without obstacles [17]. In contrast with the work of Lin *et al.* [10], the proposed algorithm can not handle the X-based routing. Our proposed algorithm comprises three steps, which are described as follows. First, a spanning graph is built from the terminals and the obstacles. Second, the slant edge of the minimal spanning tree in the spanning graph is transformed into the X-based feasible routing tree. Finally, the violated terminals are rerouted if their delays are over the delay threshold discussed in Sub-section 2.2.

The spanning graph (Fig. 3(b)) is built from the terminal and obstacles (Fig. 3(a)), to yield the spanning tree (Fig.3(c)) in the spanning graph. The spanning tree is then transformed into the feasible X-based routing tree (Fig. 3(d)). For the violated terminals (marked as gray color) over delay threshold, the original tree is split into two sub-trees by removing one edge, as shown in Fig. 3(e). Finally, the two sub-trees are merged, see Fig. 3(f). The proposed algorithm is shown in Fig.4.

Algorithm1: Rectangular Obstacle-Avoiding X-based Steiner Minimal Tree (ROA-XSMT)
Input: (1) A set of *n* pins and a source
(2) A set of *m* rectangular obstacles
(3) User-defined delay threshold
Output: A timing-driven X-based routing tree
begin

1 Spanning graph construction for obstacles;
2 Construct the feasible X-based routing tree;
2.1 Obtain the minimal spanning tree;
2.2 Transform the spanning tree into the X-based routing tree;
3 Reroute the violated terminals;
3.1 Estimate the delay by applying the modified Elmore delay model;
3.2 Split the routing tree by the delay threshold;
3.3 Merge two sub-trees into one tree by the shortest feasible paths with obstacles;
3.4 Goto Step 3.2 and 3.3 iteratively until all violated terminals are rerouted;
end.

Fig. 4 Algorithm1 for ROA-XSMT.



(a) (b) (c)

Fig. 5 Terminals inside the virtual obstacles.

Algorithm1 can handle routing with rectangular obstacles, but cannot handle non-rectangular obstacles. To solve this limitation, a bounding rectangle-based transformation is presented for non-rectangular obstacles, and the routing tree is then constructed by algorithm1. However, algorithm1 cannot handle the case that some terminals are inside the virtual obstacles. If some terminals are located in the bounding rectangle as shown in Fig. 5(a), a correct and good solution might not be found, in Fig. 5(b). Our proposed algorithm can find the result in Fig. 5(c). In next sub-section, the bounding rectangle-based transformation and A-shaped pattern routing, are presented to build a spanning graph with both rectangular and non-rectangular obstacles.

3.2 Spanning Graph Construction for Rectangular and Non-rectangular Obstacles

For the NOA-XSMT problem, we can simplify superimpose a square bounding rectangle for the non-rectangular obstacle and then transform the NOA-XSMT to the ROA-XSMT problem, and use algorithm1 to solve it.

First, a virtual obstacle is produced for each non-rectangular obstacle by superimposing a square bounding rectangle for each non-rectangular obstacle. This process is called bounding rectangle-based transformation.

Second, the spanning graph is built based on the terminals and boundaries of obstacles. Terminals are classified as being inside or outside the virtual obstacles. The spanning graph is first built for terminals outside the virtual obstacles. Terminals inside the virtual obstacles are then added.

Third, A-shaped pattern routing is adopted to reduce the L_{tal}. For the terminals inside the virtual square obstacles, some extra edges are added into the spanning graph of the previous paragraph. A-shaped pattern routing method that produces two virtual nodes on the boundary of a virtual obstacle can guarantee to generate one routing result. In Fig.6(a), the spanning graph (Fig.6(b)) is generated by applying the two-virtual-node A-shaped routing, yielding the routing result in Fig. 6(c). Fig. 6(d) and (e) respectively show the spanning graph and routing result generated using three-virtual-node A-shaped routing. Fig. 6(f) illustrates the *n*-virtual-node A-shaped pattern routing. A comparison between Fig. 6(c) and (e) indicate that two-virtual-node A-shaped routing leads to efficient spanning graph construction, since a shorter routing path cannot be found with three (or *n*) virtual nodes. Similarly, A-shaped pattern routing can handle *n* terminals inside a virtual obstacle.

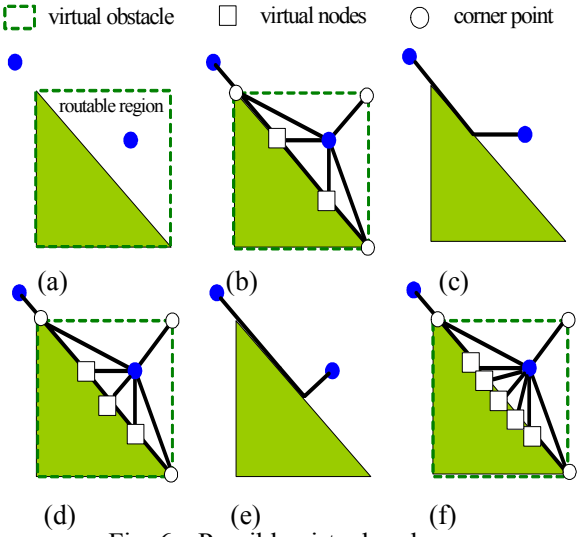
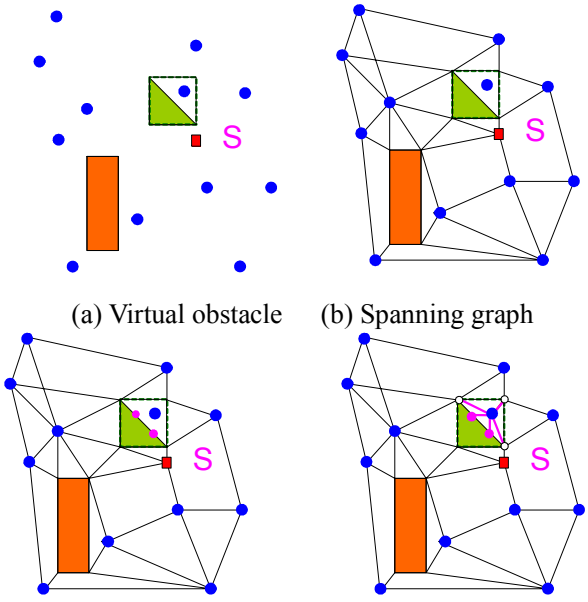


Fig. 6 Possible virtual nodes.



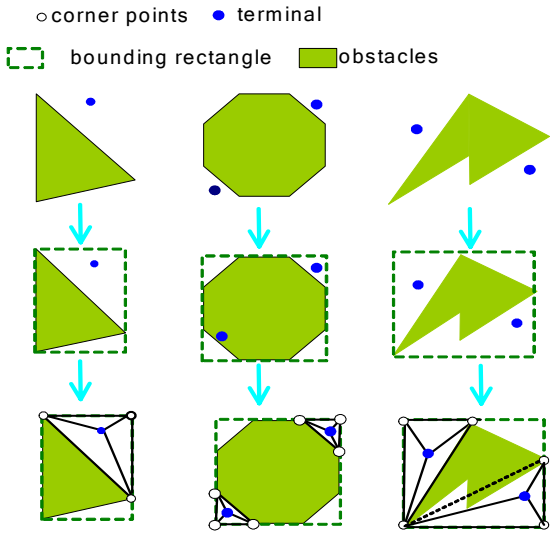
(a) Virtual obstacle (b) Spanning graph

(c) Inside virtual obstacle (d) A-shaped pattern routing

Fig. 7 Spanning graph construction.

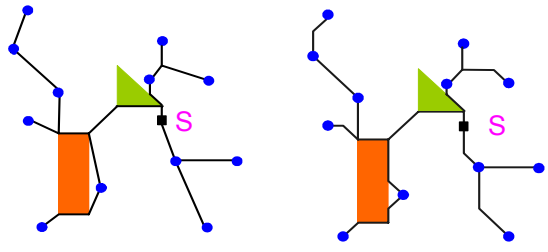
To summarize, this sub-section explains spanning graph construction using a simple example. In Fig.7, the method first builds the spanning graph for terminals, obstacle and the virtual obstacles (Fig. 7(a) and (b)), and then builds the spanning graph for the terminals inside the virtual square obstacles (Fig. 7(c)). To further minimize L_{tal} , A-shaped pattern routing is then adopted to generate three additional edges to reduce the total wirelength, as shown in Fig. 7(d).

In Sub-section 3.2, the spanning graph construction is only addressed to triangles. Other polygons, such as hexagons and octagons are handled by the bounding rectangle-based transformation, which superimposes the smallest feasible bounding rectangle to the obstacles, and builds the spanning graph for the corner points and terminals inside the virtual obstacle. A bounding rectangle is superimposed for the triangle to produce the spanning graph, as shown in Fig. 8(a). Fig. 8(b) and 8(c) respectively show the handling of hexagonal and octagonal obstacles.



(a) (b) (c)

Fig. 8. Bounding rectangle-based transformation.



(a) The spanning tree (b) X-based routing tree

Fig. 9 Transformation of the minimal spanning tree.

In this step, a spanning graph is simplified by bounding rectangle-based transformation and A-shaped pattern routing. The next step attempts to obtain the X-based routing tree with the shortest wirelength.

3.3 Construct the Feasible X-Based Routing Tree

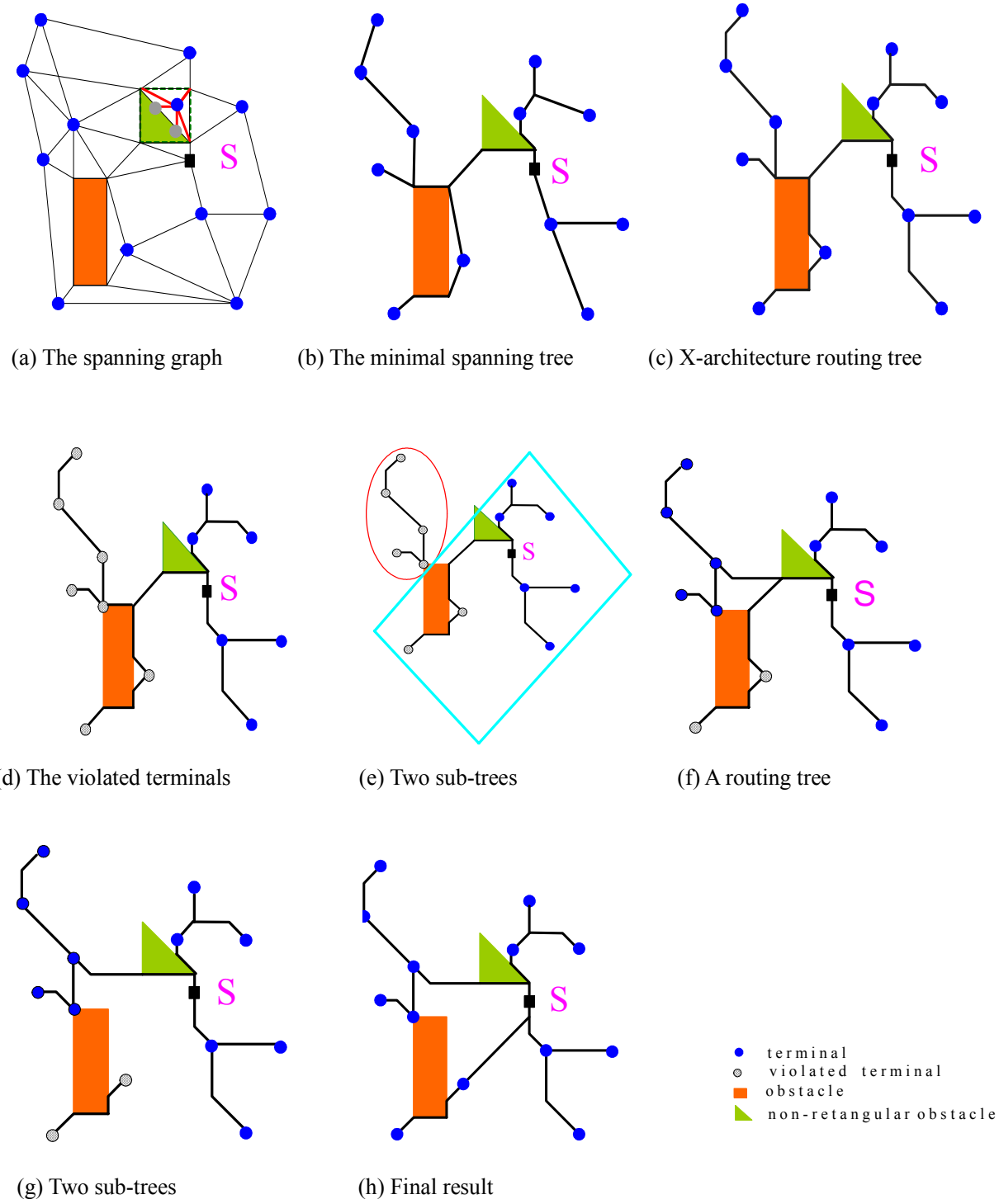
In this step, the slant edges of the minimal spanning tree are transformed in order to obtain a feasible X-based routing tree. Each feasible edge is formed by combining the horizontal, vertical and ±45° edges outside the obstacles. Fig. 9(a) shows the minimal spanning tree in the spanning graph of Fig.7(c). Post-processing is performed on the X-based routing tree to further reduce the L_{tal} . For each feasible X-based edge, the wirelength is longer if the common edges are not considered. The common edges are then merged after the neighbor edges are checked. Therefore, the post-processing successfully minimizes the wirelnegth. Each slant edge of Fig. 9(a) is transformed into a set of the feasible edges, namely the horizontal, vertical and ±45° edges. Finally, the X-based routing tree (Fig. 9(b)) is constructed by transforming the minimal spanning tree.

The delay of each terminal is calculated by the X-based delay model. In next step, we will explain rerouting for the violated terminals by delay threshold.

3.4 Reroute for All Violated Terminals

To minimize $D_{\max}$, this step reroutes the terminals with delays over the given delay threshold for the initial X-based routing tree in Sub-section 2.2. In Fig. 10, the delays of all terminals are estimated by using the delay model [4]. Due the lack of delay constraints, the delay threshold (r=0.7) is taken as the timing constraint. If we apply the large delay threshold, there are few terminals are rerouted. Hence, the maximum source-to-target delay is not very good.

To summarize, an example in Fig. 10 is used to show the whole algorithm. In Fig. 10 (a), (b) and (c), the initial routing tree is constructed by the stages of the spanning graph, the minimal spanning tree and the X-architecture routing tree. For the initial routing tree, the violated terminals are identified from the delay threshold, and labeled as the gray points (Fig. 10(d)). At the first rerouting iteration, the delays of the terminals marked in gray are over delay threshold ($r \times D_{\max}$). These terminals are labeled as violated terminals, and a proper edge is removed to produce two sub-trees, see Fig.10(e).



(a) The spanning graph (b) The minimal spanning tree (c) X-architecture routing tree

(d) The violated terminals (e) Two sub-trees (f) A routing tree

(g) Two sub-trees (h) Final result

Fig. 10 An X-based tree with the *non-rectangular* obstacles.

To minimize the $D_{\max}$, two sub-trees are merged by the selected edge, which connects the one sub-tree to the node close to the source, as indicated in Fig.10(f). A proper edge is removed to split the routing tree into two sub-trees. Similarly, in the next rerouting step, two sub-trees are produced by removing a proper (see Fig. 10(g)) edge and merging by an edge (see Fig. 10(h)). Hence, each violated terminal of the initial X-based routing tree is rerouted once.

In summary, the X-based routing tree is built by performing the steps in Sub-sections 3.2~3.4. To indicate the effect of the proposed methods, the $D_{\max}$ before and after rerouting were compared. We observe that the $D_{\max}$ of the original routing tree (Fig. 9(c)) was better than the $D_{\max}$ of the rerouted results in Fig. 10(d). Obviously, the additional wirelength was increased because the short routing paths are updated to the long routing path. These findings indicate that the improvement on the $D_{\max}$ is significant, while the additional total wirelength is small. Fig.11 shows the proposed algorithm for solving the NOA-XSMT problem.

Algorithm2: Rectangular and Non-rectangular Obstacle-Avoiding X-based Steiner Minimal Tree (NOA-XSMT)
Input: (1) A set of *n* pins and a source
(2) A set of *m* rectangular obstacles
(3) A set of *w* non-rectangular obstacles
(4) User-defined delay threshold
Output: A timing-driven X-based routing tree
begin
1 Spanning graph construction for nonrectangular and rectangular obstacles;
1.1 Superimpose a bounding rectangle for the non-rectangular obstacles;
1.2 Construct spanning graph for terminals, rectangular and non-rectangular obstacles;
1.3 A-shaped pattern routing for terminals inside the non-rectangular obstacles;
2 Construct the feasible X-based routing tree;
2.1 Obtain the minimal spanning tree;
2.2 Transform the spanning tree into the X-based routing tree;
3 Reroute the violated terminals;
3.1 Estimate the delay by applying the modified Elmore delay model;
3.2 Split the routing tree by the delay threshold;
3.3 Merge two sub-trees into one tree by the shortest feasible paths with obstacles;
3.4 Goto Step 3.2 and 3.3 iteratively until all violated terminals are rerouted;
end.

Fig. 11 Algorithm2 for NOA-XSMT.

4 Experimental Results

All experiments were implemented by the C++ language on Intel Core2 CPU 1.86GHz 1.87GHz machine with 3GB memory. The objective of was to construct the routing tree with the smallest $D_{\max}$. The X-based delay model was adopted to estimate the delay for each terminal by the 0.18*um* technology parameters. In Table 1, "Term", "OB1" and "OB2" denote the numbers of terminals, rectangular obstacles and non-rectangular obstacles, respectively. Because of the lack of source information in these benchmarks, Huang's approach [17] was adopted to test the stability under three sources. Due to limited space, Table 2 only shows the results of source located at the center of gravity. The user-defined delay threshold is the delay value above which rerouting is performed.

First, the L_{tal} and $D_{\max}$ produced by the two proposed methods were compared. Because there is no algorithm available for NOA-XSMT problem, only the two proposed algorithms without and with rerouting were compared. Because of paper limit, Table 2 only shows the result of source type I. In fact, rerouting stably improved the $D_{\max}$ under three source types. In Table 2, the second and third column respectively represent the methods without the rerouting (steps 1~2 in Fig.11 and with the rerouting (steps 1~3 in Fig.11). "L_{tal}" and "$D_{\max}$" denote the total wirelength and the maximum source-to-terminal delay, respectively. The rerouting improved the $D_{\max}$ by 49.1% with only 2.5% additional L_{tal}. Moreover, the rerouting reduced the L_{tal} of T6, T8 and T9. These results indicate that the proposed rerouting technique can effectively minimize both the L_{tal} and $D_{\max}$, even when the initial X-based routing tree is not the optimal solution. For the small benchmark, Fig.12 shows the rerouting effect on delay, indicating that the detour in routing (the red line) is reduced. For the larger benchmark, Fig.13 shows the rerouting significantly improves the detour in routing (the red line). If the rerouting does not improve the $D_{\max}$ of the initial solution, the initial routing tree is retained (T10 and T12). In other words, rerouting guarantees a feasible solution under all cases.

Second, the CPU runtimes of two proposed algorithms were compared. We observe that the proposed algorithms are quite efficient, since the additional CPU runtime for performing the rerouting is about 27.35 minutes (2519-878 second) for the largest testcase. In Sub-section 3.4, the delay threshold is given by the formula $D_{\max} \times r$, where $D_{\max}$ and r denote the maximum source-to-terminal delay and ratio, respectively. Due to limited space, only the results of r=0.7 are shown. Actually, the proposed method is stably improved the $D_{\max}$ under any given delay threshold. Furthermore,