



This article appeared in a journal published by Elsevier. The attached copy is furnished to the author for internal non-commercial research and education use, including for instruction at the authors institution and sharing with colleagues.

Other uses, including reproduction and distribution, or selling or licensing copies, or posting to personal, institutional or third party websites are prohibited.

In most cases authors are permitted to post their version of the article (e.g. in Word or Tex form) to their personal website or institutional repository. Authors requiring further information regarding Elsevier's archiving and manuscript policies are encouraged to visit:

<http://www.elsevier.com/copyright>



ELSEVIER

Available online at www.sciencedirect.com

INTEGRATION, the VLSI journal 41 (2008) 413–425

INTEGRATION
 the VLSI journal

www.elsevier.com/locate/vlsi

A full-scale solution to the rectilinear obstacle-avoiding Steiner problem

Tom Tong Jing^{a,b,*}, Yu Hu^{a,b}, Zhe Feng^{a,b}, Xian-Long Hong^b, Xiaodong Hu^c, Guiying Yan^c

^aElectrical Engineering Department, UCLA, Los Angeles, CA 90095, US

^bComputer Science and Technology Department, Tsinghua University, Beijing 100084, PR China

^cInstitute of Applied Mathematics, Chinese Academy of Sciences, Beijing 100080, PR China

Received 21 November 2006; received in revised form 20 August 2007; accepted 18 October 2007

Abstract

Routing is one of the important steps in very/ultra large-scale integration (VLSI/ULSI) physical design. Rectilinear Steiner minimal tree (RSMT) construction is an essential part of routing. Macro cells, IP blocks, and pre-routed nets are often regarded as obstacles in the routing phase. Obstacle-avoiding RSMT (OARSMT) algorithms are useful for practical routing applications. However, OARSMT algorithms for multi-terminal net routing still cannot meet the requirements of practical applications. This paper focuses on the OARSMT problem and gives a solution to full-scale nets based on two algorithms, namely An-OARSMan and FORSTer. (1) Based on ant colony optimization (ACO), An-OARSMan can be used for common scale nets with less than 100 terminals in a circuit. An heuristic, greedy obstacle penalty distance (OP-distance), is used in the algorithm and performed on the track graph. (2) FORSTer is a three-step heuristic used for large-scale nets with more than 100 terminals in a circuit. In Step 1, it first partitions all terminals into some subsets in the presence of obstacles. In Step 2, it then connects terminals in each connected graph with one or more trees, respectively. In Step 3, it finally connects the forest consisting of trees constructed in Step 2 into a complete Steiner tree spanning all terminals while avoiding all obstacles. (3) These two graph-based algorithms have been implemented and tested on different kinds of cases. Experimental results show that An-OARSMan can handle both convex and concave polygon obstacles with short wire length. It achieves the optimal solution in the cases with no more than seven terminals. The experimental results also show that FORSTer has short running time, which is suitable for routing large-scale nets among obstacles, even for routing a net with one thousand terminals in the presence of 100 rectangular obstacles. © 2007 Elsevier B.V. All rights reserved.

Keywords: Routing; Rectilinear Steiner minimal tree; Obstacle avoiding; Ant colony optimization; Track graph; Hypergraph; Full Steiner tree; Detour

1. Introduction

Routing a net, finding a rectilinear Steiner minimal tree (RSMT) for a given terminal set, is one of the fundamental problems in very/ultra large-scale integration (VLSI/ULSI) physical design. Many algorithms have been proposed focusing on the RSMT problem. However, only a few RSMT algorithms take obstacles into consideration. In practical routing applications, macro cells, IP blocks, and pre-routed nets are often regarded as obstacles. Obstacle-avoiding RSMT (OARSMT) construction is often used as an estimation for wire length even delay throughout the process of routing. Therefore, we need efficient OARSMT

algorithms in physical design. Garey and Johnson [1] proved that the RSMT problem is NP-complete. Thus, OARSMT problem is more complicated and polynomial-time algorithms can unlikely solve it exactly.

The OARSMT problem has been well studied for the case of two-terminal net. Lee et al. [2] presented maze algorithm to route two-terminal nets optimally. Some improvements on Lee's algorithm were proposed later [3–6]. The line-probe routing algorithm was introduced in [7,8]. This kind of algorithms is suitable for small-scale problems. Zheng et al. [9] proposed an algorithm, in which the searching space is restricted to a strong connection graph (implicit connection graph). The time and space complexity depend on the instance (the number of terminals and obstacle border segments). Wu et al. [10] introduced a sparse connection graph (track graph) to reduce the searching space efficiently and routed the shortest path based on the graph.

*Corresponding author. Electrical Engineering Department, UCLA, Los Angeles, CA 90095, US. Tel.: +1 310 267 4950.

E-mail address: tomjing@ucla.edu (T.T. Jing).

We need to route multi-terminal nets in the routing phase. However, routers often use the multi-terminal variant of the maze routing algorithm, which incurs the same space demand as that of the two-terminal variety and usually obtains solutions far from the optimal.

Ganley et al. [11] proposed an algorithm to construct the optimal three-terminal or four-terminal OARSMT. Then G3S, G4S, and B3S heuristics [11] were proposed for the cases with less than 20 terminals. Zachariasen et al. [12] gave an exact algorithm for finding an obstacle-avoiding Euclidean Steiner tree with less than 150 terminals. The $O(mn)$ two-step heuristic [13] (m is the number of obstacles and n is the number of terminals) works well when the number of terminals is less than seven and obstacles are convex ones. There are some recent achievements [14–16]. Shi et al. [14] achieved short wire length but could only route the nets with less than 50 terminals, which could not meet the needs of routing full-scale nets. A rectilinear approach was proposed in [15], which has the limitation of handling blockages with complex shapes such as concave polygons and handling large-scale cases.

Thus, it still needs full-scale solutions to the OARSMT problem, which can handle more terminals in the present of convex and concave polygon obstacles with short length performance and high time efficiency. The major contribution of this paper is a full-scale solution based on two heuristics, namely An-OARSMAN and FORSTER, to the obstacle-avoiding RSMT problem. A special data structure, track graph, is used in An-OARSMAN. We present the T -reduction to delete redundant vertices and edges in the track graph. Then the searching space is reduced. An-OARSMAN can obtain the optimal solution while the terminal number is less than seven and it is suitable for the common scale net routing with less than 100 terminals. FORSTER takes short running time. The full Steiner tree (FST) construction and detour method are used in the FORSTER algorithm, which makes it more practical for large-scale net routing with more than 100 terminals.

The rest of this paper is organized as follows. In Section 2, we introduce some basic definitions of the OARSMT problem, connection graphs, and the ant colony optimization (ACO). Section 3 proposes the An-OARSMAN heuristic. In Section 4, the FORSTER algorithm is described in detail. Section 5 presents the experimental results and Section 6 concludes the paper. The partial of this work, i.e., the An-OARSMAN heuristic, has been published in ASP-DAC 2005 [17]. This paper presents progress in large-scale extension and includes detailed analysis.

2. Preliminaries

2.1. Basic definitions

Definition 1. The space under consideration is the plane with x - and y -coordinate axes and the L_1 -metric, i.e., the

distance between two points (x_1, y_1) and (x_2, y_2) is defined as $|x_1 - x_2| + |y_1 - y_2|$. If all the edges of a polygon are either horizontal or vertical ones, then the polygon is called a *rectilinear polygon*. Here, we consider arbitrary rectilinear polygon obstacles without holes.

Definition 2. Considering polygon S as a set of points, a rectilinear polygon is *x-convex* if for any two points p_1 and p_2 in set S that are on the same horizontal line, the line segment $\overline{p_1 p_2}$ is also contained in this set (*y-convex* rectilinear polygon is defined similarly). And if a rectilinear polygon is both *x-convex* and *y-convex*, it is called *convex rectilinear polygon*. Otherwise, it is called *concave rectilinear polygon*.

Definition 3. *Corner points* are end points in edges of a polygon. If the interior angle at the corner point c is equal or less than 90° , point c is a *convex corner point* (e.g., point c_1 in Fig. 1). Otherwise, point c is a *concave corner point* (e.g., point c_2 in Fig. 1).

Definition 4. An *extreme edge* [10] of a rectilinear polygon is an edge on the polygon whose two neighboring edges lie on one side of the supporting line of the edge, and the immediate neighborhood [10] of the edge on that side is in the interior of the polygon. According to the definition, the rectilinear polygon in Fig. 1 has six extreme edges (dashed lines), and edge e is not an extreme edge.

Definition 5. *OARSMT*: Given a set T of n points, called terminals, and a set O of m rectilinear obstacles¹ in the plane, find a set S of additional points, called Steiner points, such that the length of a rectilinear minimum spanning tree (MST) of $T \cup S$, which avoids all obstacles in O , is minimized.

Definition 6. A rectilinear Steiner tree is called a *FST* if every terminal is a leaf of the tree.

Definition 7. FST f is *compatible* with Steiner minimal tree (SMT) s , if f and s share one and only one terminal, and they can appear simultaneously in any SMT. f and s are *incompatible*, if f and s share more than one terminal or share partial edges.

2.2. Two kinds of connection graphs

Ganley et al. [11] introduced a strong connection graph, called *escape graph*, and proved that all Steiner points in the optimal solution can be found in this graph (see Fig. 2(a)). Wu et al. [10] introduced *track graph*, a grid-like structure, which consists of rectilinear tracks defined by the obstacles and the terminals (see Fig. 2(b)).

The number of vertex and edge in an escape graph is $O((l+n)^2)$, where l is the boundary number of all obstacles and n is the terminal number. There are near 100 vertices and 200 edges in the escape graph in Fig. 2(a). The number

¹An obstacle is rectilinear, if its borders are either horizontal or vertical segments.

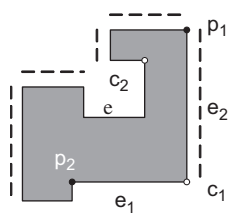


Fig. 1. Corner points and extreme edges of a polygon.

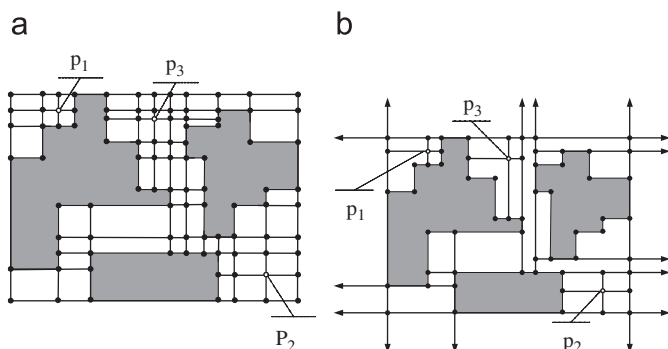


Fig. 2. (a) An escape graph, (b) a track graph. Generated by three obstacles and three terminals

of vertex and edge in a track graph is $O(r^2)$, where r is the number of extreme edge of all obstacles. There are only about 50 vertices and 100 edges in the track graph in Fig. 2(b).

The track graph is not a strong connection graph, which means that the optimal solution may not be found in some cases. It contains optimal or approximate optimal solutions in most situations. In large-scale cases, the number of edges and vertices in escape graph is much larger than that in track graph. Therefore, finding a solution in escape graph is time consuming. In the An-OARSMAN algorithm, we use track graph as the connection graph.

2.3. Ant colony optimization

The basic motivation of ACO is to mimic the cooperative behavior of ants to achieve complex computations [18], which consists of multiple iterations. In each time of iteration, one or more ants are allowed to execute a movement, while leaving behind a pheromone trail for other ants to follow. An ant traces out a single path, probabilistically selecting only one edge at a time (in a graph), until the final solution is obtained. Each separate path can be assigned a cost metric and the sum of all the individual cost defines the function to be minimized by ACO [19].

3. The An-OARSMAN algorithm

For common scale net obstacle-avoiding routing, we introduce the An-OARSMAN algorithm based on track

graph here. An efficient method, *T-reduction*, is used to reduce the searching space.

3.1. Preprocess for An-OARSMAN

Theorem 1. Let $T(V,E)$ be a track graph, q be a two-degree non-terminal point in graph T , and v_1 and v_2 be two vertices adjacent to vertex q in T . If edge (v_1, q) and edge (v_2, q) belong to the solution S to the OARSMT problem in T , and if there exists another vertex p adjacent to both vertex v_1 and vertex v_2 in graph T , then there exists another solution S' with the same cost as S such that edge (v_1, q) and edge (v_2, q) are not in S' .

Proof. Since the degree of q is equal to two, there are only two edges, (v_1, q) and (v_2, q) , incident to vertex q . These two edges in the solution S can be substituted by edge (v_1, p) and edge (v_2, p) with the same cost since we have $c(v_1, q) = c(v_2, p)$ and $c(v_2, q) = c(v_1, p)$. \square

Corollary 1 (Case 1 in *T-reduction*). Any non-terminal vertex only adjacent to two orthogonal edges, edge e_1 and e_2 , can be deleted if the other two segments forming a rectangle with e_1 and e_2 exist in the track graph. The other vertices in edge e_1 and edge e_2 are the next two vertices to be considered.

In Fig. 3, edge (v_1, q) and edge (v_2, q) form a rectangle (v_1, p, v_2, q) . The other two segments in the rectangle is $\overline{v_1p}$ and $\overline{v_2p}$, where $\overline{v_2p}$ consists two edges, (v_2, v_3) and (v_3, p) . Therefore, from Corollary 1, edge (v_1, q) and edge (v_2, q) can be deleted. In fact, $path2 = (A, \dots, v_1, p, v_3, v_2, \dots, B)$ can be the substitution of $path1 = (A, \dots, v_1, q, v_2, \dots, B)$ with the same cost while $path1$ is in a solution of an OARSMT problem. Vertices v_1 and v_2 are the next two vertices to be considered.

Corollary 2 (Case 2 in *T-reduction*). Let c be a non-terminal convex corner point (see Fig. 4) in an obstacle, v_1 and v_2 be adjacent vertices with c , v_1 and v_2 be both concave corner points. Let v_3 and v_4 be the other adjacent vertices to v_1 and v_2 besides vertex c , respectively. Let R be the rectangle formed by v_3 and v_4 . If two segments $(\overline{v_3v_5}$ and $\overline{v_4v_5})$ of R exist in graph T , which do not contain edge (v_3, v_1) and edge (v_2, v_4) , then edge (v_3, v_1) , edge (v_1, c) , edge (c, v_2) , and edge

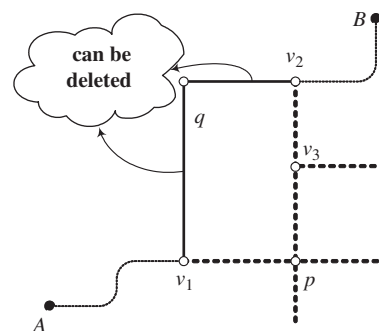


Fig. 3. An example for Corollary 1.

(v_2, v_4) can be deleted. So do vertex v_1 , vertex c , and vertex v_2 .

Case 3. After the reductions in Cases 1 and 2, it often leaves many terminals of one-degree. Such terminals can be deleted along with their adjacent edges. Then, some vertices that were neighbors of such terminals become terminals.

From Fig. 5(b), we observe that terminal p_2 becomes a one-degree vertex in the reduced graph after reductions in Cases 1 and 2. Therefore, p_2 and edge (p_2, p_2') can be deleted, which makes vertex p_2' be a terminal in the reduced graph. When we get the completed solution in the reduced graph, we add edge (p_2, p_2') back into the final solution.

Note that the advantage of the terminal reduction is that not only non-terminal vertices can be removed, but also two or more terminals can often merge into a single new terminal.

Now, an efficient graph reduction algorithm, called *T-reduction*, is proposed based on the above theorems and corollaries. The pseudo-code of this algorithm is shown in Fig. 6. We maintain an first in, first out (FIFO) queue in *T-reduction*. Firstly, we push all two-degree vertices into the queue. Secondly, we pop the first vertex from the queue, delete the vertex and its adjacent edges based on Corollary 1, and push its adjacent non-terminal vertices into the queue while the queue is not empty. Thirdly, we reduce non-terminal convex corner points by Corollary 2. Finally, we reduce one-degree terminals following the idea in Case 3.

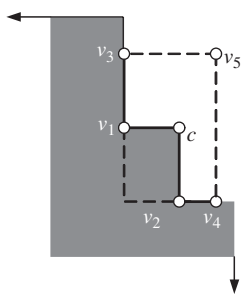


Fig. 4. An example for Corollary 2.

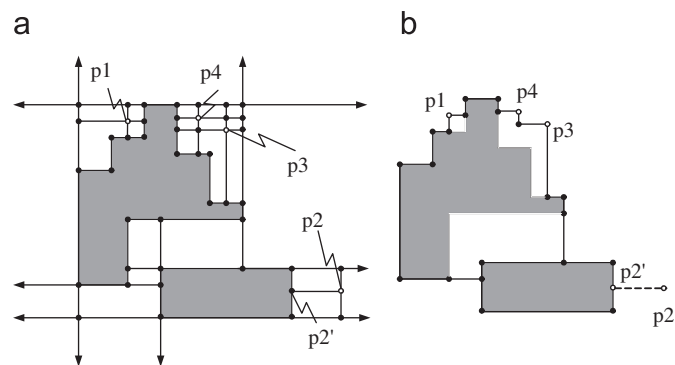


Fig. 5. (a) The track graph generated by two obstacles and four terminals, (b) the graph after reduction.

The T-Reduction Algorithm

Input: track graph $T(V,E)$

Output: reduction graph of T

Create a FIFO queue q , and push all two-degree non-terminal vertices into q ;

While q is not empty **do**

$v \leftarrow$ pop from q ;

Reduce v by **Corollary 1**;

Push the next non-terminal vertices to be considered into q ;

For each non-terminal convex corner point u **do**

Reduce u by **Corollary 2**;

Reduce all one-degree terminals;

Fig. 6. The *T-reduction* algorithm.

The track graph in Fig. 5(a) has about 50 vertices and 70 edges. There are only about 20 vertices and 20 edges left after reduction (see Fig. 5(b)).

The *T-reduction* algorithm is efficient when the case scale is not large. The timing complexity is $O(n)$, where n is the vertex number in the track graph. Tested on random generated rectilinear polygon obstacles and terminals, we found that the remain vertices and edges after reduction were only less than 40% when the terminal number was less than 10 and the extreme edge number of all obstacles was less than 20. The *T-reduction* algorithm is also suitable for escape graph reduction. *T-reduction* can guarantee the same optimality of wire length between before and after the reduction in track graph.

The alternative-based reduction technologies in [20,21] focus on the Steiner problem in a given network and graph, respectively. It means that the graphs are already the part of the instances. In our work, the instances are just terminals and obstacles. We first construct the track graph and then use *T-reduction* technology to reduce the graph size. In track graph, when we meet alternative-based situation, *T-reduction* does in a similar way to the alternative-based reduction. But *T-reduction* can also handle non-alternative-based situations in track graph, including deleting other kinds of redundant edges besides alternative ones, such as deleting edges with terminals of one-degree.

3.2. Details of An-OARSMAN

Das et al. [19] proposed an ant colony approach for constructing a Steiner tree, which is for the general Steiner problem in the Euclidean plane. There is not any obstacle considered in [19]. In this section, we construct an OARSMT on the reduced track graph based on ACO method. We first construct the connection graph in the Manhattan plane. Then, to handle obstacles, we introduce OP-distance and the related techniques used in our method.

We first generate the track graph T based on all obstacles and terminals. Then, we get the reduced graph T' by the *T-reduction* algorithm. After that, we place ants in each

terminal that needs to be connected. An ant determines a new vertex by some rules and moves to that vertex via an edge in T' . Each ant maintains its own *tabu-list* that records the vertices already visited to avoid revisiting them. When ant A meets ant B , ant A dies, and add the vertices from ant A 's *tabu-list* into ant B 's. After every movement, an ant leaves some trail in the edge just passed, and the trail evaporates in a constant rate.

There are two strategies in [19] to choose the next wanted edge, which are the greedy way and the stochastic way. An ant used the stochastic with a 90% probability in the experiments in [19]. However, based on many experiments, we found that the stochastic way got a solution with a little shorter wire length but many bends. In VLSI/ULSI physical design, more bends in a routing tree mean more vias. Vias are harmful for timing performance and reliability. Therefore, we use greedy way here.

An ant m chooses the next wanted edge that has a higher trail intensity τ_{v_i, v_k} and desirability η_{v_i, v_k}^m . That is to say, an ant travels edge (v_i, v_j) , when vertex v_j is defined as follows:

$$v_j = \max_{v_k} [\tau_{v_i, v_k}]^\alpha [\eta_{v_i, v_k}^m]^\beta, \quad (1)$$

where v_k is the neighbor vertex with v_i , and v_k is not in the *tabu-list* of m , α and β are constants. The trail intensity and desirability will be defined later.

Before giving the definition of *desirability*, we introduce a metric, called obstacle penalty distance (*OP-distance*), to estimate the distance between two vertices in the presence of obstacles. Suppose two vertices, v_1 and v_2 , in the Manhattan plane with some obstacles. The rectangle formed by v_1 and v_2 is R . If obstacle B is intersected with R , and the total length of the horizontal and vertical segments of B in interior of R is w and h , respectively, the *OP-distance* is L_1 -metric of v_1 and v_2 plus $\theta\sqrt{wh}$ (see Fig. 7), where θ is a constant. We denote the *OP-distance* between v_1 and v_2 as $op(v_1, v_2)$ in the following statement.

Given an ant m in vertex v_i , the desirability of edge (v_i, v_j) (v_j must be the neighbor of v_i in graph T') is defined as

$$\eta_{v_i, v_j}^m = \frac{1}{op(v_i, v_j) + \gamma \psi_{v_j}^m}, \quad (2)$$

where $op(v_i, v_j)$ is the *OP-distance* between vertex v_i and v_j , γ is a constant, and $\psi_{v_j}^m$ is the shortest distance from vertex v_j to all the vertices in the *tabu-list* of other ants, which makes the current ant join others as quickly as possible.

The updating of the trail intensity of edge (v_i, v_j) in graph T' is defined as follows:

$$\tau_{v_i, v_j} = (1 - \rho)\tau_{v_i, v_j} + \rho\Delta\tau_{v_i, v_j}, \quad (3)$$

where ρ is a constant, called the trail evaporation rate, which measures how rapidly the trails evolve. The increment of updating is given by the following formula:

$$\Delta\tau_{v_i, v_j} = \begin{cases} \frac{Q}{c(S_t)} & \text{if } (v_i, v_j) \in E_t, \\ 0 & \text{otherwise,} \end{cases} \quad (4)$$

where $c(S_t)$ is the total cost of the current solution tree S_t , E_t is edge set of tree S_t , and Q is a constant that matches the quantity of the tree cost.

The essence of An-OARSMAN is shown in Fig. 8. The *MAXLOOP* is the maximum number of iteration times.

The procedure **Construction_RSMT** (see Fig. 9) constructs a rectilinear Steiner tree in the track graph, where the procedure **AntMove** (see Fig. 10) decides the next wanted vertex.

In experiments, we found that when two ants met, if the survival one was still in the original location, the solution was far from the optimal. Therefore, we use the procedure **Relocate** to relocate the survival ant into a new location that is the vertex in the ant's *tabu-list* and is closest to locations of other ants.

When there is one ant left, a tree is constructed. But some leaves in the tree may not be terminals. Here, we use the procedure **Prune** (see Fig. 11) to delete all one-degree non-terminal vertices in *tree*.

The procedure **Deconfuse** in Fig. 10 solves the unavailable vertex problem. Sometimes, an ant may have unavailable vertices to move (see Fig. 12). We move this ant to some other vertices in its own *tabu-list*. But this new location should be the closest one to another ant. For example, there are two ants (m and m') left after some times of iteration shown in Fig. 12. The bold line denotes the *tabu-list* of ant m and the black solid vertices

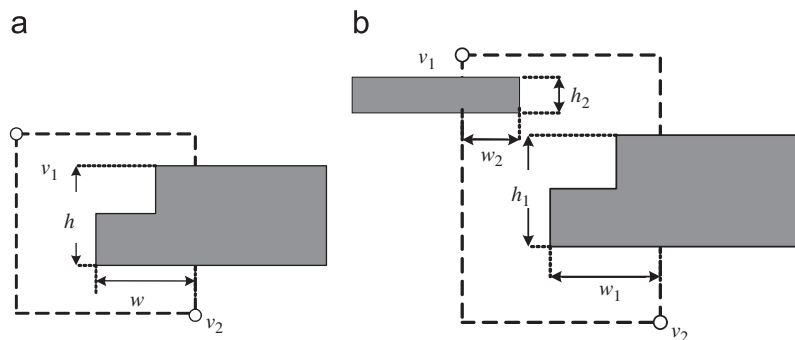


Fig. 7. (a) the *OP-distance* between v_1 and v_2 is the sum of the L_1 -metric of them and $\theta\sqrt{wh}$ for single obstacle, (b) the *OP-distance* between v_1 and v_2 is the sum of the L_1 -metric of them, $\theta_1\sqrt{w_1h_1}$ and $\theta_2\sqrt{w_2h_2}$ for multiple obstacles.

The An-OARSMAN Algorithm
Input: obstacle set O and terminal set N
Output: a rectilinear Steiner minimal tree to connect points in N and to avoid obstacles in O

Generate track graph T by O and N ;
Reduce T into T' using T-Reduction algorithm;
Set the intensity in each edge in T' to be p_0 ;
 $curBest \leftarrow \infty$, $curBestTree \leftarrow \text{NULL}$;
While $loopNum < \text{MAXLOOP}$ **do**
 $tree \leftarrow \text{Construction_RSMT}$;
 Update the trail intensity in every edge in $tree$ by Eq. (3) and Eq. (4);
 If cost of $tree < curBest$, then
 $curBest \leftarrow \text{cost of } tree$;
 $curBestTree \leftarrow tree$;
 $loopNum ++$;
Return $curBestTree$;

Fig. 8. The An-OARSMAN algorithm.

The Construction_RSMT Procedure
Input: track graph $T(V,E)$ and terminal set N
Output: a rectilinear Steiner minimal tree in T

Place an ant on each vertex in the terminal set N and put the vertex into its tabu-list;
 $tree \leftarrow \text{NULL}$;
While ant number > 1 **do**
 Select an ant m randomly;

$tree \leftarrow tree + \text{AntMove}(m)$;
If m meets m' **then**
 Add vertices from tabu-list of m into that of m' ;
 m died;
 Relocate m' ;
Prune ($tree$);
Return $tree$;

Fig. 9. The Construction_RSMT() procedure.

The AntMove Procedure
Input: ant m who is in vertex v_i currently
Output: the edge between vertex v_i and the next wanted vertex v_j

Compute the next wanted edge (v_i, v_j) of ant m by Eq. (1) and Eq. (4);
If there are no edges can be chosen then
 Deconfuse;
 Ant m moves to v_j ;
 Add vertex v_j into m 's tabu-list;
Return the edge connect vertex v_i and v_j ;

Fig. 10. The AntMove() procedure.

denote terminals. The current ant m is in vertex C whose only two neighbor vertices (vertex A and B) are both in its *tabu-list*. Therefore, ant m cannot move any more. We should find a new location in the *tabu-list*

of ant m and make the new location be the closest one to another ant. Following this idea, we consider vertex X as the new location of ant m , which is closest to ant m' .

The Prune Procedure

Input: a Steiner tree t with some non-terminal leaves

Output: a Steiner tree t' without non-terminal leaves

```

For each vertex  $s$  in tree  $t$  do
    Set  $s$  valid;
For each vertex  $s$  in tree  $t$  do
    While  $s$  is valid do
        If  $s$  is terminal
            Break;
        If the degree of  $s$  is not equal to one
            Break;
         $e \leftarrow$  the edge connect  $s$  in  $t$ ;
        Set  $e$  invalid;
        Set  $s$  invalid;
    }
 $t' \leftarrow$  NULL;
For each edge  $e$  in tree  $t$  do
    Add  $e$  into tree  $t'$ ;
Return  $t'$ ;
    
```

Fig. 11. The Prune() procedure.

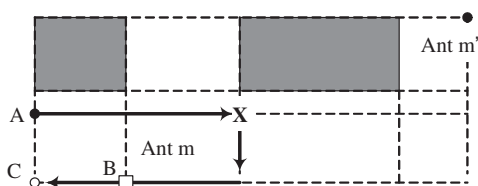


Fig. 12. An example of the unavailable-vertex problem.

4. The FORSTer algorithm

An-OARSMAN works well for common scale nets. But to get a full-scale solution, we still need efficient method for large-scale nets. In this Section, we give a detailed description of the FORSTer algorithm for large-scale net obstacle-avoiding routing.

4.1. Outline of FORSTer

Our FORSTer algorithm is a three-step heuristic. In Step 1, we partition all the terminals into some subsets in the presence of obstacles by some rules. After Step 1, connected components, regarded as hyper graphs [22], are generated. In Step 2, we connect terminals in each connected graph with one or more trees, respectively. As a result, the separated sub-trees are constructed, respectively, after this step. In Step 3, we connect the forest consisting of the trees constructed in Step 2 into a completed Steiner tree spanning all terminals while avoiding all obstacles. GFST-RSMT is proposed to construct OARSMT in a connected graph in Step 2.

The flow chart and an illustration of FORSTer are shown in Figs. 13 and 14, respectively.

4.2. Details of FORSTer

4.2.1. Step 1: Terminal partition

The process of terminal partition is as follows.

Firstly, we construct the FSTs (see Definition 6) for all terminals by Hwang's theorems [23], the rules of Warme [24], and the method in [25] with $O(n^2)$ time. In this operation, we do not consider all obstacles.

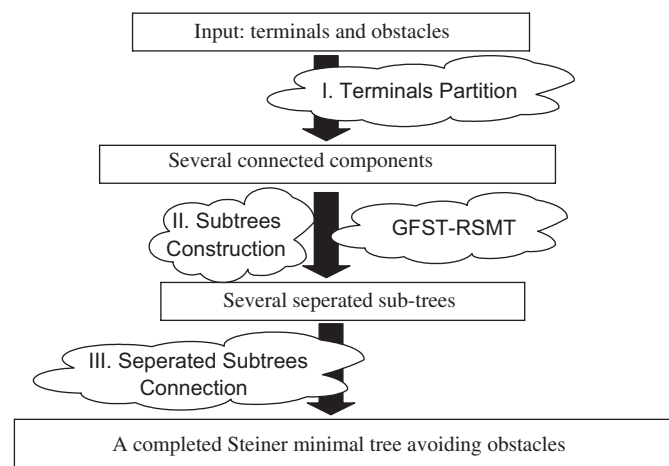


Fig. 13. The flow chart of FORSTer algorithm

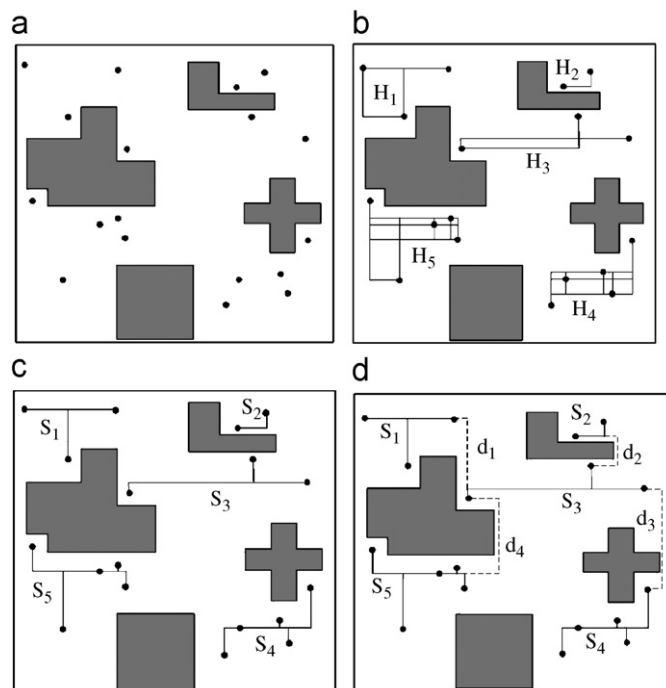


Fig. 14. An illustration of FORSTer algorithm, (a) the input of the problem: some terminals and four obstacles, (b) the result after Step 1 (terminal partition), (c) the result after Step 2 (sub-tree construction), (d) the result after Step 3 (separated sub-tree connection)

Secondly, we add all obstacles on the graph with generated FSTs. Some FSTs intersect with obstacles and we delete them. After this operation, some FSTs (all non-intersected with obstacles) survived, which is called *survival set* denoted by F .

Finally, we use *hypergraph* [22] to describe all the terminals and the survival set F . Consider the hypergraph $H = (T, F)$ with the set of terminals T as its vertices and the survival set F as its hyperedges. Obviously, the hypergraph H is partitioned into some connected sub-graphs denoted by $H_1 = (T_1, F_1)$, $H_2 = (T_2, F_2), \dots$, and $H_n = (T_n, F_n)$, where $T_1, T_2, \dots, T_n \in T$, and $F_1, F_2, \dots, F_n \in F$.

In Fig. 14(b), we can see that the hypergraph H is partitioned into five connected sub-graphs, H_1, H_2, \dots , and H_5 . Also, the set T and set F are partitioned into five subsets correspondingly.

4.2.2. Step 2: Sub-tree construction

In this step, the major task is to connect the terminals in each connected sub-graph H_i with one completed Steiner tree or several separated Steiner trees. In Fig. 14(c), RSMTs are constructed in each of the connected sub-graph, which are S_1, S_2, \dots , and S_5 , respectively.

We now propose a method to construct an OARSMT in a given connected sub-graph, named greedy FST (GFST)-RSMT.

GFST-RSMT selects *compatible* (see Definition 7) FSTs in F_i greedily, and constructs several separated SMTs, such as $s_{i1}, s_{i2}, \dots, s_{in}$, etc.

$$T(s_{ij}) \cap T(s_{ik}) = \emptyset \quad \text{for each pair of } j \text{ and } k.$$

$$T(s_{i1}) \cup T(s_{i2}) \cup \dots \cup T(s_{in}) = T_i,$$

where $T(s)$ is the set of terminal spanned by SMTs.

Each FST is initialized with the measure η , and the measure η of a FST f is defined as

$$\eta(f) = c(f)^\lambda / n(f)^\omega \quad (\lambda > 0, \omega > 0), \quad (5)$$

where $c(f)$ is the wire-length of f , and $n(f)$ is the number of terminals in f , λ and ω are constants.

Setting suitable value for λ and ω in Eq. (5) plays an important role in the experiments. If we just use the function $\eta(f) = c(f)/n(f)$ instead of Eq. (5), a FST with smaller η indicates it covers more terminals with a shorter wire-length, by which we can get the shortest wire length and make use of the greedy strategy. But the shortcoming is that it has the tendency to choose FST with two terminals. That is, $c(f)/n(f)$ cannot play well on a FST covering more terminals though it has a short wire length. When the function $\eta(f) = c(f)/n(f)$ is changed into Eq. (5), the result of covering more terminals is better. After we tested all types of convex and concave rectilinear polygons, such as rectangle, L-shaped polygon, cross-shaped polygon, and more complicated shapes, we set $\lambda = 1$ and $\omega = 4$, by which we can archive the best result.

For each sub-graph H_i , the following method is used to construct one completed Steiner tree or several separated Steiner trees.

We maintain a triple (D, Q, Y) , where Y is the set of current generated Steiner minimal trees in H_i , D is the set of candidate FSTs in F_i , and Q are the terminals that are not covered by Y in T_i . In the initialization, Y is set to be empty, D is set to be F_i , and Q is set to be T_i .

At the beginning of GFST-RSMT algorithm, we select the FST f with minimal η from D , and add a new SMT s made up of f into Y . We check every FST in D , if the FST is *incompatible* with Y , it should be removed from D . If there are some FSTs in D compatible with s in Y , the one whose η is minimal should be selected and added into s , and the terminals covered by this FST should be removed from Q . When there is no FST compatible with Y , if D is not empty, select the FST f with minimal η , and add a new SMT made up of f into Y , and repeat the above process. If D is empty, every terminal left in Q makes up of a degenerated SMT, and add them into Y .

The pseudo-code of GFST-RSMT algorithm is shown in Fig. 15.

The number of edges and vertices in the connection graph C_i constructed by F_i algorithm is $O(n)$ [24], where n is the number of terminals in T_i . For GFST-RSMT algorithm, we compute the η for all FSTs in F_i , and sort these FSTs by η , which requires $O(n \log n)$ time. Then, we select $O(n)$ FSTs one by one greedily. Thus, GFST-RSMT runs in $O(n \log n)$ time.

4.2.3. Step 3: Separated sub-tree connection

After the process of Steps 1 and 2, some separated sub-trees are constructed, respectively. In Step 3, we need to connect these sub-trees into a completed tree avoiding all obstacles. We route the exact shortest path of each pair of nodes in each pair of trees, and then choose the minimum ones to connect all the Steiner trees. The dot lines in Fig. 14(d) show the results of Step 3 for the given instance, and the shortest paths connecting the trees in Fig. 14(c) are d_1, d_2, d_3 , and d_4 , respectively.

Zheng et al. [9] introduced the concept of *detour* value, and proved that the shortest path avoiding obstacle is the Manhattan distance plus twice of the detour value.

We compute the shortest path between the source node and the destination node in the presence of obstacles by using our *obstacle-avoiding shortest path (OASP)* algorithm. The OASP algorithm is based on the idea of the *detour* technique proposed in [9].

Zheng et al. [9] proved that the time-complexity to route the shortest path between a node pair with detour algorithm is $O(e \log(e))$, where e is the number of edges of obstacles. We must compute every node-pair of all different sub-trees, which is $O(n^2)$ node-pairs. Therefore, the time-complexity of OASP algorithm is $O(n^2 e \log(e))$.

a

The GFST-RSMT Algorithm for One Sub-graph

Input: Terminal set T_i in H_i

Output: One or more separated rectilinear Steiner minimal trees in H_i spanning T_i

Initiate FST set F_i in partition H_i (computing η for every FST, and ordering all the FST ascendantly);

$D \leftarrow F_i; Q \leftarrow T_i; Y \leftarrow \emptyset;$

Select the first FST f in D ;

$Y \leftarrow s \leftarrow f;$

Remove (D, f, Q);

While $D \neq \emptyset$ **do**

From first FST f to the last one in D

If f is incompatible with Y

Remove f from D ;

else

$s \leftarrow s + f;$

Remove (D, f, Q);

If $D \neq \emptyset$

Select the first FST f in D

$s \leftarrow f;$

$Y \leftarrow Y + s;$

Remove (D, f, Q);

For each terminal n left in Q **do**

$Y \leftarrow n;$

Return Y ;

b

The Remove Procedure

Input: D, f, Q

Remove FST f from D ;

Remove all the terminals covered by f from Q ;

Fig. 15. (a) The GFST-RSMT algorithm for one sub-graph, (b) the Remove() procedure.

Table 1
Comparison between An-OARSMAN and Yang's two-Step

Net ID	Number of obstacles	Number of terminals	Yang's two-step (%)	An-OARSMAN		
				Best (%)	Average (%)	Worst (%)
C2:22	2	2	0	0	0	0
C2:33	2	2	0	0	0	0
C2:2	2	3	0	0	0	0
C2:17	2	3	1.87	0	0	0
C2:504	2	4	3.80	0	0	0
C2:59	2	3	4.46	0	0	0
C2:609	2	4	4.89	0	0	0
C2:67	2	5	5.48	0	0	0
C2:177	2	6	7.28	0	0.19	1.32
C2:98	2	4	21.4	0	0	0
C2:117	2	3	1.87	0	0	0
C2:18	2	4	3.80	0	0.06	0.45
Average	/	/	5.31	0	0.02	0.15

Table 2
Comparison between An-OARSMAN and FORSTer

Number of terminals	Number of obstacles	Wire length		Running time (s)	
		An-OARSMAN	FORSTer	An-OARSMAN	FORSTer
7	7	19,380	22,700	0.04	<0.01
8	7	21,040	24,090	0.05	<0.01
9	9	25,380	27,450	0.08	<0.01
10	9	26,990	27,330	0.09	<0.01
12	10	31,630	33,770	0.23	<0.01
16	10	41,350	43,780	0.25	<0.01
20	10	43,630	45,790	0.39	<0.01
30	10	44,970	46,120	0.77	0.01
50	10	53,260	55,410	3.22	0.26
70	10	68,390	70,140	8.84	0.87
100	10	80,040	81,830	31.3	3.12
500	10	—	223,415	—	559.00
1000	10	—	285,821	—	1240.0

The time-complexities of the three steps are $O(n^3)$, $O(n \log(n))$, and $O(n^2 e \log(e))$, respectively, which suggests that the total time-complexity of FORSTer

algorithm is $\max(O(n^3), O(n^2 e \log(e)))$, where n is the number of terminals and e is the number of edges of obstacles.

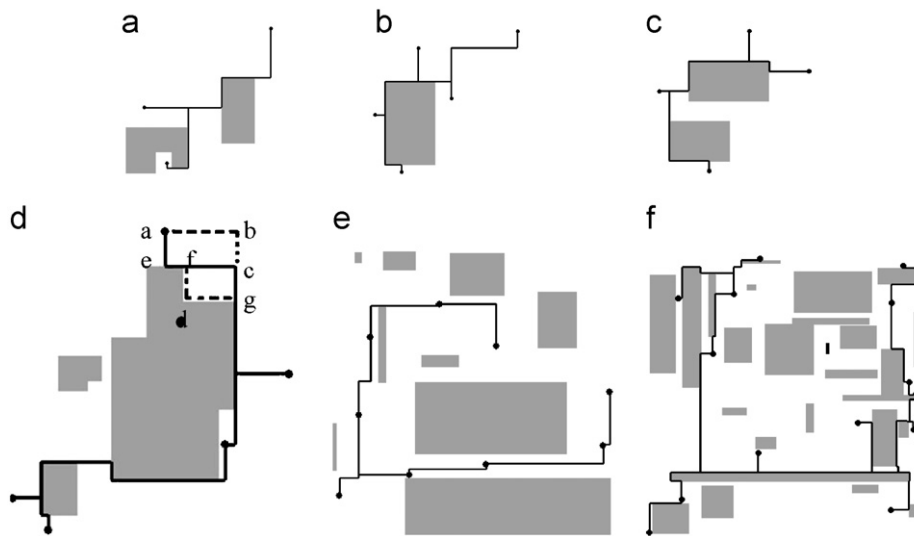


Fig. 16. Some typical results by An-OARSMAN.

5. Experimental results

We have implemented these two algorithms in C language and performed it on a computer with Linux operating system. We first randomly generate the terminals by the usual random function. Then, we use the same random function to generate the boundaries of the obstacles. When an obstacle is constructed, we check if there are terminals inside it. If so, the obstacle is deleted. The obstacles are constructed to include all types of convex and concave rectilinear polygons, such as rectangle, L-shaped polygon, cross-shaped polygon, and more complicated shapes.

We set $\alpha = 5$, $\beta = 1$, $\gamma = 1$, $\theta = 1$, and $Q = 10,000$ (the scale of all benchmarks is $10,000 \times 10,000$) in An-OARSMAN, and set $\lambda = 1$ and $\omega = 4$ in GFST-RSMT for all experiments, by which the average best performance is archived.

Having tested all cases with 3000 times of iteration, we found that the algorithm got the most improvements in the first 50 iterations and improved little after 500 iterations. Therefore, we set $MAX_LOOP = 500$ for every case in the final testing considering both the length quality of the solution and the running time.

To compare our An-OARSMAN with recent progress, i.e., Yang's two-step heuristic [13], by MCNC (C2) benchmarks, $redundancy^2$ metric introduced in [13] is used here to judge the solution quality. We run each case introduced in [13] for 10 times. All cases can be finished within 0.1 s. The result comparisons are shown in Table 1.

² $redundancy = (sol - opt) / opt \times 100\%$, where sol is the result of compared algorithms (such as An-OARSMAN and Yang's two-step heuristic), opt is the optimal solution generated by manual construction due to a few terminals.

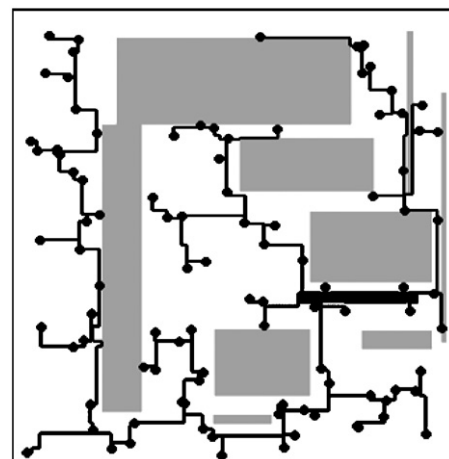


Fig. 17. The result by An-OARSMAN for a 100-terminal net among 10 rectangle obstacles.

In Table 1, we can see that An-OARSMAN can achieve the optimal routing solution and is better than Yang's two-step obviously.

Table 2 shows the length performance comparison between An-OARSMAN and FORSTER on cases with more than seven terminals. The first two columns of Table 2 are the number of terminals and obstacles, respectively. The next two columns show the wire length of trees produced by An-OARSMAN and FORSTER, respectively. The last two columns show the running time of An-OARSMAN and FORSTER, respectively.

From Table 2, we can see that the length performance of An-OARSMAN is better than that of FORSTER. And An-OARSMAN takes about 30 s to handle cases with 100 terminals. We think it is an acceptable time, if we want to get short wire length. We also can see that for large-scale nets with more than 100 terminals, FORSTER works in an

acceptable time since there is not any other algorithm can be available.

An exact algorithm for obstacle-avoiding Euclidean Steiner tree construction was presented in [12]. It takes over 1000, 3000 and 4500 s, respectively, to construct a Steiner tree with 10, 50 and 100 terminals among six polygon obstacles by a 200 MHz processor, while An-OARSMAN takes about 0.1, 3 and 32 s, respectively, to construct a tree with 10, 50 and 100 terminals among over nine obstacles by a 755 MHz processor (from Table 2). Meanwhile, FORSTer handling 1000 terminals among 10 obstacles has the similar running time to the algorithm in [12] handling 10 terminals among six obstacles.

Typical results by An-OARSMAN are shown in Fig. 16.

Tree construction is the fundamental in VLSI/ULSI routing and serves routing. Therefore, when we design tree (Steiner tree) algorithms, we have the motivation of considering the requirements in routing. Here, we consider wire length, running time, and routability. We take Fig. 16(d) as an example. (1) When we construct track graph, we delete edge fd and edge dq to get a speedup since the searching space is reduced. (2) During the reduction of track graph, all the two-degree vertices are deleted. The

vertices on the boundary of obstacles are mostly Three-degree and Four-degree ones and they are survived. Then the path is more prone to be near the boundary of obstacles. That is, edge ab and edge bc are deleted and the path from vertex a to vertex c is connected by edge ae and edge ec . The benefit is that it leaves more routing resource and is helpful for routability.

We can regard this strategy as the tradeoff between running time, routability, and wire length. We sacrifice wire length to some extent and get shorter running time and higher routability. As for Fig. 16(f), the suboptimal wire length is due to the same reason.

There are concave and other obstacles in these cases. Fig. 16(e) shows the results for a nine-terminal net among nine obstacles. Fig. 16(f) shows the result of a 13-terminal net among 30 obstacles. Fig. 17 shows the result by An-OARSMAN for a 100-terminal net among 10 rectangle obstacles. Fig. 18 shows the result by FORSTer to route 1000 terminals in the presence of 100 rectangular obstacles.

6. Conclusions

Based on An-OARSMAN and FORSTer, this paper introduces a full-scale solution to the rectilinear obstacle-avoiding

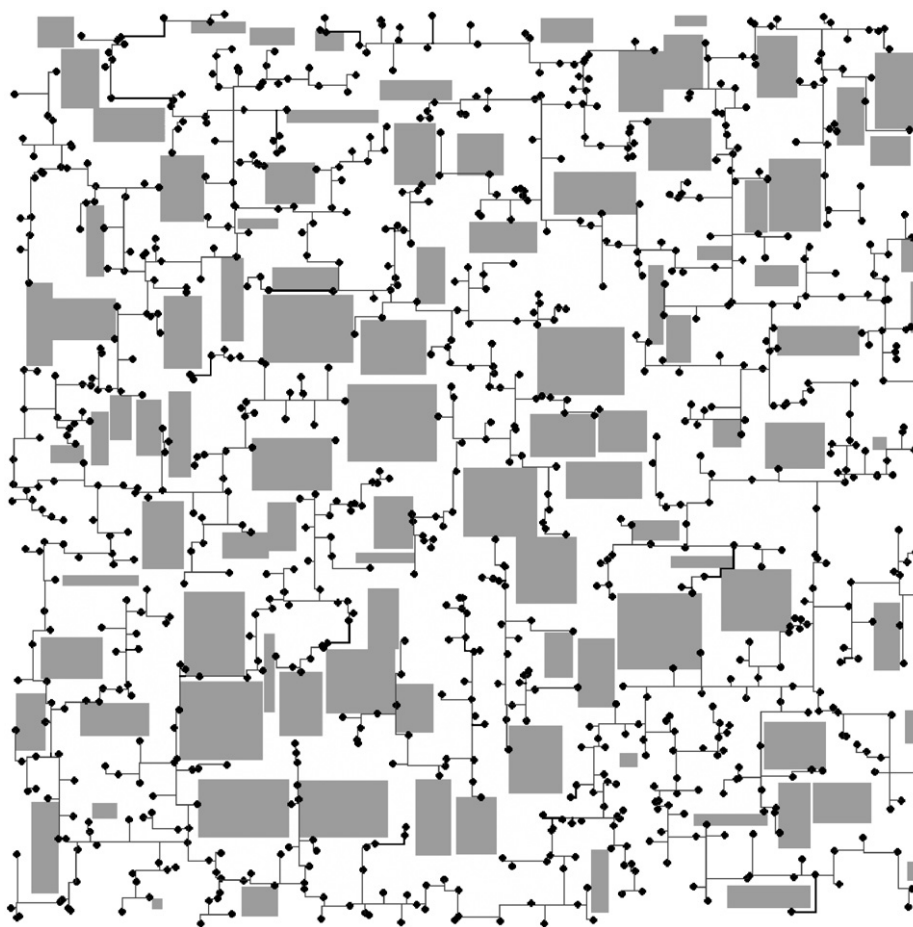


Fig. 18. The result by FORSTer for 1000 terminals in the presence of 100 obstacles.

Steiner minimal tree problem. With an efficient reduction method on the track graph, An-OARSMan keeps high wire length performance for common scale net obstacle-avoiding routing. The three-step heuristic, FORSTer, takes short running time for large-scale net obstacle-avoiding routing.

Acknowledgments

This work was supported in part by the Key Project of Chinese Ministry of Education under Grant no. 106008, the Specialized Research Fund for the Doctoral Program of Higher Education (SRFDP) of China under Grant no. 20050003099, and the National Natural Science Foundation of China (NSFC) under Grant nos. 90607001, 10531070 and 10771209.

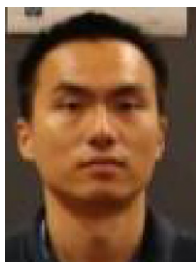
References

- [1] M.R. Garey, D.S. Johnson, The rectilinear Steiner tree problem is NP-complete, *SIAM J. Appl. Math.* (32) (1977) 826–834.
- [2] C.Y. Lee, An algorithm for path connections and its applications, *IRE Trans. Electron. Comput.* 10 (1961) 346–365.
- [3] S.B. Akers, A modification of Lee's path connection algorithm, *IEEE Trans. Electron. Comput.* 16 (4) (1967) 97–98.
- [4] J. Soukup, Fast Maze Router. In: Proceedings of 15th DAC, 1978, pp. 100–102.
- [5] F.O. Hadlock, A shortest path algorithm for grid graphs, *Networks* (7) (1977) 323–334.
- [6] F. Rubin, The Lee connection algorithm, *IEEE Trans. Comput.* (23) (1974) 907–914.
- [7] D.W. Hightower, A solution to the line routing problem on the continuous plane. in: Proceedings of the Sixth DAC, 1969, pp. 1–24.
- [8] K. Mikami, K. Tabuchi, A computer program for optimal routing of printed circuit connectors. in: Proceedings of IFIPS, H47, 1968, pp. 1475–1478.
- [9] S.Q. Zheng, J.S. Lim, S.S. Iyengar, Finding obstacle-avoiding shortest paths using implicit connection graphs, *IEEE Trans. CAD* 15 (1) (1996) 103–110.
- [10] Y.F. Wu, P. Widmayer, M.D.F. Schlag, C.K. Wong, Rectilinear shortest paths and minimum spanning trees in the presence of rectilinear obstacles, *IEEE Trans. Comput.* 36 (3) (1987) 321–331.
- [11] J.L. Ganley, J.P. Cohoon, Routing a multi-terminal critical net: steiner tree construction in the presence of obstacles, in: Proceedings of IEEE ISCAS, London, UK, 1994, pp. 113–116.
- [12] M. Zachariassen, P. Winter, Obstacle-avoiding Euclidean Steiner trees in the plane: an exact algorithm, in: International Workshop on Algorithm Engineering and Experimentation, Lecture Notes in Computer Science 1619, 1999, pp. 282–295.
- [13] Y. Yang, Q. Zhu, T. Jing, X.L. Hong, Y. Wang, Rectilinear Steiner minimal tree among obstacles. in: Proceedings of IEEE ASICON, Beijing, China, 2003, pp. 348–351.
- [14] Y.Y. Shi, T. Jing, L. He, Z. Feng, X.L. Hong, CDCTree: Novel obstacle-avoiding routing Tree construction based on current driven circuit model, in: Proceedings of IEEE/ACM ASP-DAC, January Yokohama, Japan, 2006, pp. 630–635.
- [15] Z. Shen, C.N. Chu, Y.M. Li, Efficient rectilinear Steiner Tree construction with rectilinear blockages, in: Proceedings of IEEE ICCD, 2005, pp. 38–44.
- [16] M. Min, S.C.H. Huang, J. Liu, E. Shragowitz, W. Wu, Y. Zhao, Y. Zhao, An approximation scheme for the rectilinear Steiner minimum tree in presence of obstructions, novel approaches to hard discrete optimization, fields institute communications series, *Am. Math. Soc.* 37 (2003) 155–163.
- [17] Y. Hu, T. Jing, X.L. Hong, Z. Feng, X.D. Hu, G.Y. Yan, An-OARSMan: Obstacle-avoiding Routing Tree construction with good length performance, in: Proceedings of IEEE/ACM ASP-DAC, Shanghai, China, 2005, pp. 7–12.
- [18] M. Dorigo, V. Maniezzo, A. Colomi, The ant system: optimization by a colony of cooperating agents, *IEEE Trans. Syst. Man Cybern.–Part B* 26 (1) (1996) 1–13.
- [19] S. Das, S.V. Gosavi, W.H. Hsu, S.A. Vaze, An ant colony approach for the Steiner tree problem, in: Proceedings of Genetic and Evolutionary Computation Conference (GECCO), New York City, NY, USA, 2002, pp. 135.
- [20] T. Polzin, S. Vahdati Daneshmand, Extending reduction techniques for the Steiner Tree problem: a combination of alternative and bound-based approaches, Research report, 2001. <<http://citeseer.ist.psu.edu/polzin01extending.html>>.
- [21] E. Uchoa, M. Poggi de Arago, C.C. Ribeiro, Preprocessing Steiner problems from VLSI layout, Technical Report MCC. 32/99, Departamento de Informtica, PUC-Rio, Rio de Janeiro, Brazil, 1999, <<http://citeseer.ist.psu.edu/447157.html>>.
- [22] M. Zachariassen, The Rectilinear Steiner Tree problem: A Tutorial, steiner trees in industries, Kluwer Academic Publishers, 2001, pp. 467–507.
- [23] F.K. Hwang, D.S. Richards, P. Winter, The Steiner Tree problem, *Annals of Discrete Mathematics*, North-Holland, Amsterdam, The Netherlands, 1992.
- [24] D. M. Warme, P. Winter, M. Zachariassen, "Exact algorithms for plane Steiner tree problems: a computational study, Technical Report DIKU-TR-98/11, Department of Computer Science, University of Copenhagen, April 1998.
- [25] M. Zachariassen, Rectilinear full Steiner tree generation, *Networks* 33 (1999) 125–143.

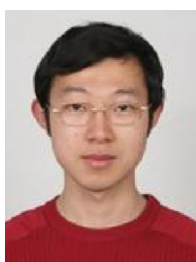


Tom Tong Jing received the B.S. degree in electronic engineering and the M.S. and Ph.D. degrees in computer science from Northwestern Polytechnical University, Xi'an, China, in 1989, 1992 and 1999, respectively. He is currently a Research Associate at Electrical Engineering Department, University of California at Los Angeles. From September 1999 to April 2001, he was a Postdoctoral Researcher at Computer Science and Technology Department, Tsinghua University, Beijing, China. He was a Faculty

Member (Associate Professor from 2001 to 2004, Tenured Associate Professor from 2004 to 2006) at Computer Science and Technology Department, Tsinghua University. He was a Visiting Scholar at University of California at San Diego and Chinese University of Hong Kong. He has authored or coauthored more than 100 papers published in technical journals and conference proceedings. His research interests include electronic design automation (EDA), combinational optimization and algorithms, graph theory, and programming. Dr. Jing is a recipient of IEEE/ACM ASP-DAC Best Paper Award in 2005, ACM/IEEE ISQED Best Paper Nomination in 2005, and IEEE ASICON Outstanding Student Paper Award in 2003. He is a recipient of the Second Class Science and Technology Award by Ministry of Education of China in 2005. He is a recipient of the First Class Award for Excellence in Teaching by Beijing Municipal Education Commission in 2004 and the First Class Awards for Excellence in Teaching by Tsinghua University in 2002 and 2004, respectively. He has served as a TPC member of IEEE/ACM ASP-DAC 2006; the Secretary General, Chair of Physical Design and Interconnect Optimization TPC-Subcommittee, and Session Chair of IEEE/ACM ASP-DAC 2005; the Session Chair of IEEE ASAP 2005; a TPC member, a Panel Speaker, and Session Co-Chair of IEEE ICCAS 2004; a Session Chair of ISCI 2004; the Secretary General and TPC member of IEEE ASICON 2003; and the Session Co-Chair of IEEE/ACM ASP-DAC 2003.



Yu Hu received the B.S. degree and M.S. degree both in computer science from Tsinghua University, Beijing, China, in 2002 and 2005, respectively. Currently, he is a Ph.D. degree candidate in the Electrical Engineering Department at the University of California, Los Angeles. He worked with Xilinx Research Laboratory in the summer of 2006. He is currently a Graduate Student Researcher (GSR) with the Electrical Engineering Department at the University of California, Los Angeles. His current research interests include CAD for FPGA synthesis and ASIC physical synthesis. He is the author of over 20 technical papers in journals and international conferences, and the inventor of 4 patents in the field of CAD for VLSI designs. Mr. Hu received the outstanding graduate student award in 2005 from Tsinghua University. He has been a full member of Sigma Xi since 2007, and a student member of IEEE since 2004.



Zhe Feng received the M.S. degree in computer science and technology from Tsinghua University, Beijing, China, in 2007. He is currently working towards the Ph.D. degree in the Department of Electrical Engineering, University of California, Los Angeles. His research interest focuses on synthesis and physical design for FPGAs, and physical design for ASICs.



Xian-Long Hong received the B.S. degree from Tsinghua University, Beijing, China, in 1964. Since 1988, he has been a Professor in the Department of Computer Science and Technology, Tsinghua University, Beijing, China. He has published 5 books and more than 400 papers. His research interests include very large-scale integration (VLSI) layout algorithms and design automation systems. Prof. Hong has served as the steering member of Asia and South Pacific Design Automation Conference (ASPDAC) and

the co-chair of technical program committee of ASPDAC in 1999, 2004 and 2005. He has been an Associate Editor of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART I since 2002.



Xiaodong Hu received his B.S. degree in applied mathematics from Tsinghua University, Beijing, China, in 1985, and his Ph.D. degree in operations research and cybernetics from Institute of Applied Mathematics, Chinese Academy of Sciences, in 1989. He was a post-doctor fellow at Rutgers Center for Operations Research, Rutgers University, in 1990–1991, a visiting associate professor at graduate school of information science, Japan Advanced Institute of Science and Technology, in 1993–1994, and a research fellow at Computer Science Department, City University of Hong Kong, in 1998–2000. Currently, he is a research professor at Institute of Applied Mathematics, Chinese Academy of Sciences. His research interests include combinatorial optimization and computer communication networks. He is the member of American Mathematical Society (since 1992) and member of IEEE Computer Society (since 1997).



Guiying Yan received her B.S., M.S. and Ph.D. degrees in operational research from Shandong University, in 1989, 1992 and 1995, respectively. Currently, she is a Professor of Academy of Mathematics and Systems Science. From 1995 to 1997, she was a postdoctor fellow of Institute of applied mathematics, Chinese Academy of Sciences. She is now serving as an Editor of the IEEE Acta Mathematicae Applicatae Sinica.