

An-OARSMAN: Obstacle-Avoiding Routing Tree Construction with Good Length Performance *

Yu Hu, Tong Jing, Xianlong Hong, Zhe Feng

Xiaodong Hu, Guiying Yan

Computer Science & Technology Department
Tsinghua University
Beijing 100084, P. R. China

E-mail: {matrix98, z-feng04}@mails.tsinghua.edu.cn
{jingtong, hxl-dcs}@tsinghua.edu.cn

Institute of Applied Mathematics
Chinese Academy of Sciences
Beijing 100080, P. R. China

E-mail: {xdhu, yangy}@public.bta.net.cn

Abstract - Routing is one of the important steps in VLSI/ULSI physical design. The rectilinear Steiner minimum tree (RSMT) construction is an essential part of routing. Since 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. This paper focuses on the OARSMT problem and presents an algorithm, named An-OARSMAN, based on ant colony optimization. A greedy obstacle penalty distance (OP-distance) local heuristic is used in the algorithm and performed on the track graph. The algorithm has been implemented and tested on different kinds of obstacles. Experimental results show that An-OARSMAN can handle complex obstacle cases including both convex and concave polygon obstacles with good length performance. It can always achieve the optimal solution in the cases with no more than 7 terminals.

I. INTRODUCTION

Routing a net, finding a rectilinear Steiner minimum tree (RSMT) for a given terminal set, is a fundamental task in very/ultra large scale integrated circuit (VLSI/ULSI) physical design. Many algorithms have been proposed focusing on the RSMT problem. In practical routing applications, macro cells, IP blocks, and pre-routed nets are often regarded as obstacles. Then, obstacle-avoiding RSMT (OARSMT) construction is often used as an accurate estimation for wire length even delay throughout the process of routing.

The OARSMT problem has been well studied for the case of two-terminal nets. Lee *et al* [1] presented maze algorithm to route two-terminal nets optimally. Some improvements on Lee's algorithm proposed later [9-12]. The method of line search routing is introduced in [14] and [15]. This kind of algorithms is suitable for small scale problems.

Zheng *et al* [2] proposed an algorithm, in which the searching space is restricted to a strong connection graph (implicit connection graph). The time and space complexity depends on the instance (the number of terminals and obstacle border segments). Wu *et al* [3] introduced a sparse connection graph (track graph) reducing the searching space efficiently and computed the shortest path based on the graph.

We often need to route multi-terminal nets in the routing

phase. However, only a few existing OARSMT algorithms take multi-terminal nets into consideration. The routers always use the multi-terminal variant of the maze routing algorithm, which incurs the same space demands as that of the two-terminal variety and usually obtains solutions far from the optimal. Ref. [13] proved the RSMT problem is NP-Complete. So, the OARSMT problem is more complicated and no polynomial-time algorithm can solve it exactly.

Ganley *et al* [4] proposed an algorithm to construct the optimal 3-terminal or 4-terminal OARSMT. Then G3S, G4S, and B3S heuristics [4] are proposed for the cases with less than 20 terminals. Zachariasen *et al* [5] gave an exact algorithm for finding an obstacle-avoiding Euclidean Steiner tree with less than 150 terminals. Then a progress in OARSMT is an $O(mn)$ 2-step heuristic presented in [6]. It works well when the terminal number is less than 7 and obstacles are convex ones. The recent progress, FORst [16], can tackle large scale problem efficiently.

It still needs to research on OARSMT algorithms with good length performance for routing multi-terminal nets. Such algorithms need to handle more terminals and obstacles, concave polygon obstacles, and complex obstacles. The main contribution of this paper is a heuristic, called An-OARSMAN, for obstacle-avoiding rectilinear Steiner minimum tree construction, by which we can tackle complex obstacles and keep high length performance. 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 always obtain the optimal solution while the terminal number is less than 7. It can route among both convex and concave polygon obstacles. It routes a 100-terminal net among 10 rectangular obstacles with about thirty seconds.

The rest of this paper is organized as follows. In Section II, we introduce some basic definitions of the OARSMT problem and the ant colony optimization. Section III proposes an efficient reduction method on track graph. In Section IV, An-OARSMAN heuristic is described in detail. Section V shows the experimental results and Section VI concludes the whole work.

II. PRELIMINARIES

A. Basic definitions

The space under consideration is the plane with x and y

* This work was supported in part by the NSFC under Grant No.60373012, the SRFDP of China under Grant No.20020003008, and the Hi-Tech Research and Development (863) Program of China under Grant No.2004AA1Z1050.

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.

Consider 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 which 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*.

Corner points are end points in edges of a polygon. Let e_1 and e_2 be the two edges adjacent to a corner point c , p_1 and p_2 be the two points on e_1 and e_2 , respectively. If the segment $\overline{p_1 p_2}$ passes through the interior of the polygon, c is a *convex corner point* (e.g. c_1 in Fig.1). Otherwise, c is a *concave corner point* (e.g. c_2 in Fig.1).

An *extreme edge* [3] 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 [3] of the edge on that side is in the interior of the polygon. According to the definition, the rectilinear polygon in Fig.1 has 5 extreme edges, and edge e is not an extreme edge.

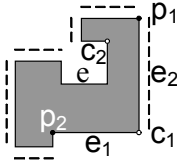


Fig.1. Extreme edges of a polygon.

B. Two kinds of connection graphs

Ganley *et al* [4] 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* [3] introduced *track graph*, a grid-like structure, which consists of rectilinear tracks defined by the obstacles and the terminals (see Fig.2 (b))

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

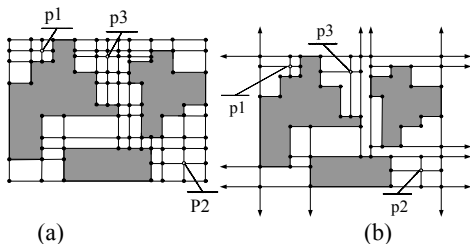


Fig.2. (a) An escape graph. (b) A track graph. Generated by 3 obstacles and 3 terminals.

The track graph is not a strong connection graph, which means that in some cases the optimal solution can not be found. 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. So, finding a solution in escape graph is time consuming. In this paper, we use track graph as the connection graph.

C. Ant colony optimization

The basic motivation of ant colony optimization (ACO) is to mimic the cooperative behavior of ants to achieve complex computations [7], which consists of multiple iterations. In each iteration, one or more ants are allowed to execute a movement, leaving behind a pheromone trail for others to follow. An ant traces out a single path, probabilistically selecting only one edge at a time (in a graph), until an entire solution is obtained. Each separate path can be assigned a cost metric and the sum of all the individual costs defines the function to be minimized by ACO [8].

III. PREPROCESS FOR AN-OARSMAN

Our An-OARSMAN algorithm is based on track graph, we use an efficient method to reduce the searching space before we describe the algorithm.

Theorem1: Let $T(V, E)$ be a track graph, q be a 2-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 of the OARSMT problem in T , then there exists another solution S' with the same cost such that edge (v_1, q) and edge (v_2, q) are not in S' if there exists another vertex p adjacent to both v_1 and v_2 in graph T .

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 . On the other hand, 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 $c(v_2, q) = c(v_1, p)$. ■

Corollary 1 (Case 1): Any non-terminal vertices only adjacent to two orthogonal edges, 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 e_1 and 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) . And the other two segments in the rectangle is $\overline{v_1 p}$ and $\overline{v_2 p}$, where $\overline{v_2 p}$ consists two edges, (v_2, v_3) and (v_3, p) . So, from Corollary 1, edge (v_1, q) and edge (v_2, q) can be deleted. In fact, we can also find that $path2 = (A, \dots, v_1, p, v_3, v_2, \dots, B)$ 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. v_1 and v_2 are the next two vertices to be considered.

Corollary 2 (Case 2): 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 point*. Let v_3 and v_4 be the other adjacent vertices to v_1 and v_2 respectively besides vertex c . 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 (v_2, v_4) can be deleted. So do vertex v_1 , vertex c , and vertex v_2 .

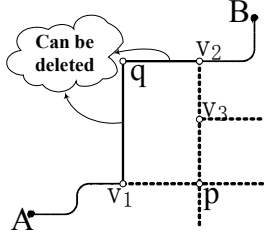


Fig. 3. An example for Corollary 1.

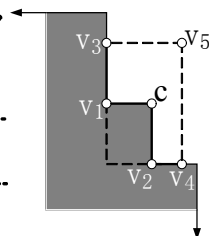


Fig. 4. An example for Corollary 2.

Case 3: After the reduction in Case 1 and Case 2, it often leaves many terminals of degree 1. 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 can see that terminal p_2 becomes a 1-degree vertex in the reduced graph after the reduction in Case 1 and Case 2. So 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 not only non-terminal vertices can be removed, but also two or more terminals can often collapse into a single new terminal.

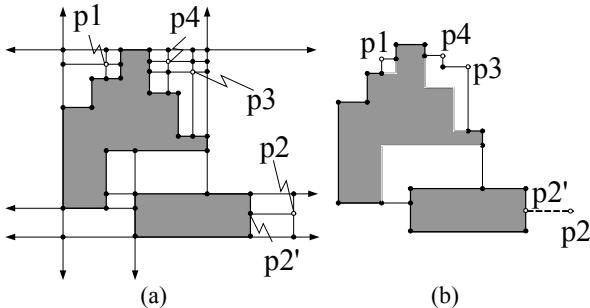


Fig. 5. (a) The track graph generated by two obstacles and four terminals, (b) The graph after reduction.

Now, an efficient graph reduction algorithm, called *T-Reduction*, is proposed based on the above theorem and corollaries. The pseudo-code of this algorithm is shown in Fig. 6. We maintain an FIFO queue in *T-Reduction*. Firstly, we push all 2-degree vertices into the queue. Then 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. After that, we reduce non-terminal convex corner points by Corollary 2. Finally, reduce 1-degree terminals following the idea in Case 3.

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)).

Input: track graph $T(V, E)$
Output: reduction graph of T
1. Create a FIFO queue q , and Push all 2-degree non-terminal vertices into q ;
2. 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 ;
3. For each non-terminal convex corner point u do Reduce u by Corollary 2;
4. Reduce all 1-degree terminals;

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

The *T-Reduction* algorithm is very efficient when the case scale is not large. The timing complexity is $O(n)$, where n is the vertices number in the track graph. Testing by random generated rectilinear polygon obstacles and terminals, we find that the remain vertices and edges after reduction is only less than 40% when the terminal number is less than 10 and the extreme edges number in all the obstacles is less than 20. The *T-Reduction* algorithm is also suitable for escape graph reduction.

IV. THE AN-OARSMAN ALGORITHM

Das *et al* [8] proposed an ant colony approach for constructing a Steiner tree in a given graph. Having made some improvements and changes on the basic approach, we construct an OARSMT on the reduced track graph in this Section.

We firstly generate the track graph T based on all obstacles and terminals. Then, 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 will determine a new vertex by some rules and move to that vertex via an edge in T' . Each ant maintains its own *tabu-list*, which records the vertices already visited to avoid revisiting it. When ant A meets ant B , ant A dies, and add the vertices in the A 's *tabu-list* into B 's. After every movement, an ant will leave some trail in the edge just passed, and the trail will evaporate in a constant rate.

There are two strategies introduce in [8] to choose the next wanted edge, which are a greedy way and a stochastic way. An ant will use the stochastic with a 90% probability in the experiments in [8]. However, based on many experiments, we found that the stochastic will produce a solution with a little shorter wire length and much more bends. In VLSI/ULSI physical design, more bends in a routing tree will produce more vias. Vias are harmful for timing performance and reliability. So, we use greedy way instead of the stochastic one here.

An ant m will choose the next wanted edge which have a higher trail intensity τ_{v_i, v_k} and desirability η_{v_i, v_k}^m , that is to say, an ant will travel edge (v_i, v_j) , when vertex v_j is defined as follows.

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

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 *OP-distance* (obstacle penalty 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 $\theta \cdot \sqrt{w \cdot h}$ (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.

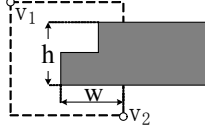


Fig.7. the OP-distance between v_1 and v_2 is the sum of the L_1 -metric of them and $\theta \cdot \sqrt{w \cdot h}$.

<p>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</p>
<ol style="list-style-type: none"> 1. Generate track graph T by O and N; 2. Reduce T into T' using T-Reduction algorithm; 3. Set the intensity in each edge in T' to be p_0; 4. $curBest \leftarrow \infty$, $curBestTree \leftarrow NULL$; 4. While $loopNum < MAXLOOP$ do; $tree \leftarrow$ Construction_RSMT; Update the trail intensity in every edge in $tree$ by equation (3) and (4); If cost of $tree < curBest$, then $curBest \leftarrow$ cost of $tree$; $curBestTree \leftarrow tree$; $loopNum ++$; 5. Return $curBestTree$;

Fig.8. The An-OARSMAN algorithm.

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:

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

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 into others as quickly as possible.

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

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

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 S_t , and Q is a constant which 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(m)** (see Fig.10) decides the next wanted vertex.

<p>Input: track graph $T(V, E)$ and terminal set N Output: a rectilinear Steiner minimal tree in T</p>
<ol style="list-style-type: none"> 1. Place an ant on each vertex in the terminal set N and put the vertex into its tabu-list; 2. $tree \leftarrow NULL$; 2. While ant number > 1 do Select an ant m randomly; $tree \leftarrow tree +$ AntMove(m); If m meets m' then; Add vertices in tabu-list of m into that of m'; m died; Relocate m'; 3. Prune($tree$); 4. Return $tree$;

Fig.9. The Construction_RSMT procedure.

<p>Input: ant m who is in vertex v_i currently Output: the edge between vertex v_i and the next vertex v_j</p>
<ol style="list-style-type: none"> 1. Compute the next wanted edge (v_i, v_j) of m by equation (1) and (4); 2. If there are no edges can be chosen then Deconfuse; 3. Ant m moves to v_j; 4. Add vertex v_j into m's tabu-list; 5. Return the edge connect vertex v_i and v_j;

Fig.10. The AntMove(m) procedure.

<p>Input: a Steiner tree t with some non-terminal leaves Output: a Steiner tree t' without non-terminal leaves</p>
<ol style="list-style-type: none"> 1. For each vertex s in tree t do Set s valid 2. 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 1 Break; $e \leftarrow$ the edge connect s in t; Set e invalid; Set s invalid; 3. $t' \leftarrow NULL$; 3. For each edge e in tree t do Add e into tree t' 5. Return t';

Fig.11. The Prune(t) procedure.

In experiments, we found that when two ants meet, if the survival one is still in the original location, the solution will be far from the optimal. So we use the procedure **Relocate** to relocate the survival ant into a new location, which is the vertex in the ant's tabu-list closest to those of other ants.

When there is one ant left, a tree is constructed. But some leaves in the tree may be not a terminal. So we use the procedure **Prune** (see Fig.11) to deletes all 1-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 iterations shown in Fig.12. The bold line denotes the *tabu-list* of ant m and the black solid vertices denote terminals. The current ant m is in vertex C whose only two neighbor vertices (A and B) are both in its *tabu-list*. So, ant m can not move any more. We should find a new position in the *tabu-list* of ant m and make the new position be the closest one to another ant. Following this idea, we consider vertex X as the new position of ant m , which is closest to ant m' .

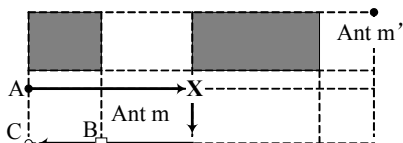


Fig.12. An example of the unavailable-vertex problem.

V. EXPERIMENTAL RESULTS AND DISCUSSIONS

We have implemented the algorithm in C language and performed it on a Sun V880 fire workstation with Unix operating system. We randomly generate some terminals among several kinds of rectilinear polygon obstacles to test the proposed algorithm. The obstacles are constructed to include all types of convex and concave rectilinear polygons, which consist of rectangle, L-shaped polygon, cross-shaped polygon, and more complicated shapes. We set $\alpha = 5$, $\beta = 1$, $\gamma = 1$, $\theta = 1$, and $Q = 10000$ (the scale of all benchmarks is 10000×10000) here.

Having tested all cases with 3000 iterations, we find that the algorithm gets the most improvements in the first 50 iterations and improves little after 500 iterations. So we set $MAX_LOOP = 500$ in every cases in the final testing considering both the length quality of the solution and the running time.

To compare our An-OARSMAN with Yang's 2-step heuristic [6] by MCNC benchmarks, *redundancy** metric introduced in [6] is used here to judge the solution quality. We run each case ten times. All cases can be finished with 0.1 second. The result comparisons are shown in TABLE I.

In TABLE I, we can see that An-OARSMAN can always achieve the optimal routing solution and is better than Yang's 2-step obviously.

TABLE II shows the performance comparison between FORst [16] and our An-OARSMAN on cases with more than 7

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

terminals. The first two columns of TABLE II are the number of terminals and obstacles, respectively. The next two columns show the wire length of trees produced by our An-OARSMAN and FORst, respectively. The last column shows the running time of our An-OARSMAN.

TABLE I
COMPARISON BETWEEN AN-OARSMAN AND YANG'S 2-STEP

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

TABLE II
COMPARISON BETWEEN AN-OARSMAN AND FORST

Terminal Number	Obstacle Number	Wire Length		Running Time (s)
		OARSMAN	FORst	
7	7	19380	22700	0.04
8	7	21040	24090	0.05
9	9	25380	27450	0.08
10	9	26990	27330	0.09
12	10	31630	33770	0.23
16	10	41350	43780	0.25
20	10	43630	45790	0.39
30	10	44970	46120	0.77
50	10	53260	55410	3.22
70	10	68390	70140	8.84
100	10	80040	81830	31.3

In VLSI/ULSI routing, most of the nets in a circuit have no more than 100 terminals (clock network design needs special methods). From TABLE II, we can see that it takes about 30 seconds for An-OARSMAN to handle cases with 100 terminals, furthermore, the performance of our An-OARSMAN is better than FORst, which makes it practical for routing applications.

An exact algorithm for obstacle-avoiding Euclidean Steiner tree construction is presented in [5]. It takes over 1000, 3000, and 4500 seconds respectively to construct a Steiner tree with 10, 50, and 100 terminals among 6 polygon obstacles by a 200MHz processor, while our An-OARSMAN takes about 0.1, 3, and 32 seconds respectively to construct a tree with 10, 50, and 100 terminals among over 9 rectangle obstacles by a 755MHz processor (from TABLE II). Although, the comparison is not so significant, these facts also show that our An-OARSMAN is more practical.

Typical routing results are shown in Fig.13. There are concave and other obstacles in these cases. Fig.13 (e) shows the results for a 9-terminal net among 9 obstacles. Fig.13 (f)

shows the result for a 13-terminal net among 30 obstacles. Fig.14 (g) shows the result for a 100-terminal net among 10 rectangle obstacles.

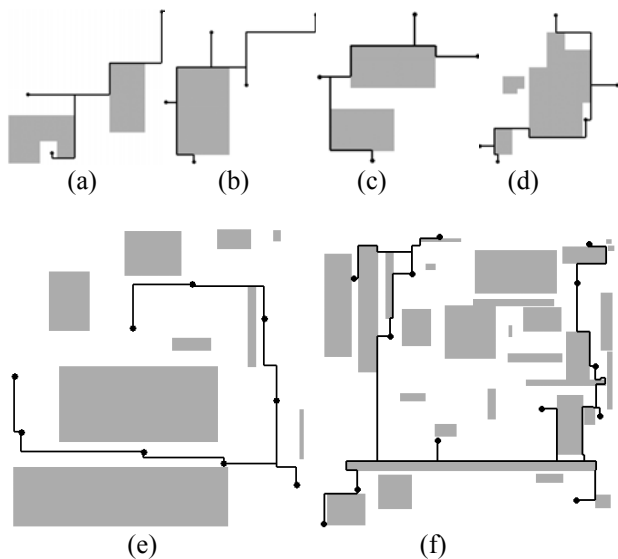


Fig.13 Some typical results of An-OARSMAN

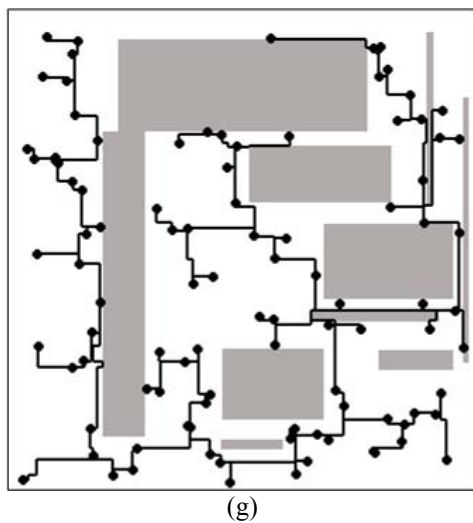


Fig.14 the result for a 100-terminal net among 10 rectangle obstacles

VI. CONCLUSIONS

In this paper, we give an efficient reduction method on the track graph. Then, we present a heuristic for OARSMT problem based on ACO. The experimental results show that our heuristic, An-OARSMAN, keeps the high length performance.

As future work, we will speed up our An-OARSMAN algorithm while keeping an approximation optimal wire length.

REFERENCES

- [1] C.Y.Lee, "An algorithm for path connections and its applications". IRE Trans on Electronic Computers, 1961, 10: pp. 346-365.
- [2] S.Q.Zheng, J.S.Lim, and S.S.Iyengar, "Finding obstacle-avoiding shortest paths using implicit connection graphs". IEEE Trans on CAD, 1996,15(1): pp. 103-110.
- [3] Y.F.Wu, P.Widmayer, M.D.F.Schlag, and C.K.Wong. "Rectilinear Shortest Paths and Minimum Spanning Trees in the Presence of Rectilinear Obstacles", IEEE Trans on Computers, 1987, 36(3): pp. 321-331.
- [4] 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, 1994: pp. 113-116.
- [5] M.Zachariasen and P.Winter, "Obstacle-avoiding Euclidean Steiner trees in the plane: an exact algorithm", extended abstract presented at the Workshop on Algorithm Engineering and Experimentation (ALENEX), 1999.
- [6] Y.Yang, Q.Zhu, T.Jing, X.L.Hong, and Y.Wang, "Rectilinear Steiner Minimal Tree among Obstacles". In Proc. of IEEE ASICON, Beijing, China, 2003: pp. 348-351.
- [7] M.Dorigo, V.Maniezzo, and A.Colomi, "The Ant System: Optimization by a colony of cooperating agents", IEEE Trans on Systems, Man, and Cybernetics-Part B, 1996, 26(1): pp.1-13.
- [8] S.Das, S.V.Gosavi, W.H.Hsu, and S.A.Vaze, "An Ant Colony Approach for the Steiner Tree Problem", In Proc. of Genetic and Evolutionary Computing Conference, New York City, New York, 2002.
- [9] S.B.Akers, "A Modification of Lee's Path Connection Algorithm", IEEE Trans on Electronic Computer, 1967, 16(4): pp. 97-98.
- [10] J.Soukup, "Fast Maze Router", In Proc. of 15th DAC, 1978: pp. 100-102.
- [11] Hadlock, "A Shortest Path Algorithm for Grid Graphs", Networks, 1977(7): pp. 323-334.
- [12] F.Rubin, "The Lee Connection Algorithm", IEEE Trans on Computer, 1974(23): pp. 907-914.
- [13] M.R.Garey and D.S.Johnson, "The rectilinear Steiner tree problem is NP-complete", SIAM Journal on Applied Mathematics, 1977(32): pp. 826-834.
- [14] D.W.Hightower, "A Solution to the Line Routing Problem on the Continuous Plane", In Proc. of the 6th DAC, 1969: pp. 1-24.
- [15] K.Mikami and K.Tabuchi, "A Computer Program for Optimal Routing of Printed Circuit Connectors", in Proc of IFIPS, 1968, H47: pp. 1475-1478.
- [16] Yu Hu, Zhe Feng, Tong Jing, Xianlong Hong, Yang Yang, Ge Yu, Xiaodong Hu, and Guiying Yan, "A 3-Step Heuristic for Obstacle-Avoiding Rectilinear Steiner Minimum Tree Construction", in Proc. of ISC&I'04, Zhuhai, China, pp.1017-1021.