Jan. 2006, Vol.21, No.1, pp.147–152 J. Comput. Sci. & Technol.

ACO-Steiner: Ant Colony Optimization Based Rectilinear Steiner Minimal Tree Algorithm*

Yu Hu[1,3] (胡 昱), Tong Jing[1] (经 彤), Zhe Feng[1] (冯 哲), Xian-Long Hong[1] (洪先龙)
Xiao-Dong Hu[2] (胡晓东), and Gui-Ying Yan[2] (严桂英)

[1] *Department of Computer Science and Technology, Tsinghua University, Beijing 100084, P.R. China*

[2] *Institute of Applied Mathematics, Chinese Academy of Sciences, Beijing 100080, P.R. China*

[3] *Electrical Engineering Deparment, UCLA, Los Angeles, CA 90095, U.S.A.*

E-mail: hu@ee.ucla.edu; jingtong@tsinghua.edu.cn

Received November 11, 2004; revised June 3, 2005.

Abstract The rectilinear Steiner minimal tree (RSMT) problem is one of the fundamental problems in physical design, especially in routing, which is known to be NP-complete. This paper presents an algorithm, called ACO-Steiner, for RSMT construction based on ant colony optimization (ACO). An RSMT is constructed with ants' movements in Hanan grid, and then the constraint of Hanan grid is broken to accelerate ants' movements to improve the performance of the algorithm. This algorithm has been implemented on a Sun workstation with Unix operating system and the results have been compared with the fastest exact RSMT algorithm, GeoSteiner 3.1 and a recent heuristic using batched greedy triple construction (BGTC). Experimental results show that ACO-Steiner can get a short running time and keep the high performance. Furthermore, it is also found that the ACO-Steiner can be easily extended to be used to some other problems, such as rectilinear Steiner minimal tree avoiding obstacles, and congestion reduction in global routing.

Keywords rectilinear Steiner minimal tree (RSMT), routing, physical design, ant colony optimization (ACO)

1 Introduction

Routing plays an important role in the very large scale integrated circuit/ultra large scale integrated circuit (VLSI/ULSI) physical design. Useful algorithms have been proposed focusing on routability[1,2], timing issue[3−5], coupling effects[6], and rectilinear routing tree construction[7,8].

The rectilinear Steiner minimal tree (RSMT) problem is one of the fundamental problems in routing. However, Garey and Johnson[9] proved that the RSMT problem is NP-complete, which indicates that a polynomial-time algorithm to compute an optimal RSMT is unlikely to exist. So, many helpful algorithms continue to focus on the RSMT problem to get high efficiency.

[10] gave an extensive survey of RSMT heuristics in 1992. Kahng and Robins[11] introduced the Batched Iterated 1-Steiner (BI1S) heuristic with an average improvement over the minimum spanning tree (MST) on terminals of almost 11%. The edge based heuristic of Borah[12] has a time complexity of $O(n^2)$ and similar performance as the BI1S. In recent years, several efficient heuristics were presented. Kahng and Mandoiu *et al.*[13] proposed a batched greedy triple construction (BGTC) algorithm with a very short runtime while keeping the performance. Zhou[14] introduced the spanning graph as a base for MST and then constructed the RSMT from the MST. Another work is an $O(n \log n)$ octilinear Steiner minimal tree heuristic[15]. Besides these heuristics, Warme *et al.* released the GeoSteiner[16,17], which is the most efficient existing exact algorithm. The shortcoming of GeoSteiner is the long runtime. So, there is still some room for researchers to design some algorithms to obtain higher efficiency.

The main contribution of this paper is to propose an efficient heuristic, called ACO-Steiner, to construct an RSMT, by which we can get a short running time and keep the high performance. When the number of terminals is no more than 50, ACO-Steiner can achieve exact results (better than BGTC) while keeping the short running time. When the number of terminals is more than 50, ACO-Steiner can achieve approximate optimal results (within 1% wire length increments compared with GeoSteiner) but keeping short running time. Furthermore, our ACO-Steiner is easy to be extended to solve other tree construction problems, such as rectilinear Steiner minimal tree avoiding obstacles, and congestion reduction in global routing.

The rest of this paper is organized as follows. In Section 2, we introduce the ant colony optimization (ACO) and some basic definitions of RSMT problem. In Section 3, the ACO-Steiner heuristic is described in detail. Section 4 gives performance improvements based on some special strategies. Then, Section 5 gives the experimental results and some discussions. Finally, Section 6 concludes the paper.

Short Paper

This work was partially supported by the National Natural Science Foundation of China (NSFC) under Grant No. 60373012, and the Specialized Research Fund for the Doctoral Program of Higher Education (SRFDP) of China under Grant No. 20050003099.

*Some preliminary results of this work were presented at IEEE International Conference on Communications, Circuits and Systems (ICCCAS), Chengdu, China, 2004.

2 Preliminaries

2.1 Ant Colony Optimization

Ant Colony Optimization (ACO) is a class of algorithms that mimic the cooperative behavior of real ant behavior to achieve complex computations consisting of multiple iterations[18,19].

2.2 Denotations in RSMT Problem[20]

In the rest of this paper, T always denotes the terminal set and S always denotes the Steiner point set. The cost ($c(i,j)$) between vertex i and vertex j is the Manhattan distance between vertexes i and j.

3 Our ACO-Steiner Algorithm

3.1 Tree Construction Based on ACO

In this section, we construct the RSMT based on ACO. Firstly, we generate the Hanan grid[21] of the terminal set T. Then, we place the 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 Hanan grid. Each ant maintains its own tabu-list, which records the visited vertices to avoid revisiting it again. When ant A meets ant B, ant A dies, and the vertices in the A's tabu-list are added 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.

An ant determines its next vertex it wants to move stochastically, but the process is biased on a higher value $p_{i,j}$, which is a trade-off between the desirability and the trail intensity.

Given an ant m in vertex i, the desirability of vertex j (j must be the neighbor of i in Hanan grid) is defined as follows.

$$\eta_j^m = \frac{1}{c(i,j) + \gamma \cdot \psi_j^m} \tag{1}$$

where γ is a constant, and ψ_j^m is the shortest distance from vertex i 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 in Hanan edge (i,j) is defined as follows.

$$\tau_{i,j} = (1-\rho) \cdot \tau_{i,j} + \rho \cdot \Delta\tau_{i,j} \tag{2}$$

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.

$$\Delta\tau_{i,j} = \begin{cases} \dfrac{Q}{c(S_t)}, & \text{if } (i,j) \in E_t \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

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

The probability of an ant using edge (i,j) to move is defined as follows.

$$p_{i,j} = \begin{cases} \dfrac{[\tau_{i,j}]^\alpha [\eta_j^m]^\beta}{\sum_{k \notin tabu-list(m)} [\tau_{i,k}]^\alpha [\eta_k^m]^\beta}, & \text{if } j \in A \\ 0, & \text{otherwise} \end{cases}$$

where A is the set made up of all vertices which are connected with i and are not in the tabu-list of ant m. It is obvious that an ant has at most three possible vertices to move in the Hanan grid when it has set off, that is, the four neighbor vertices except the vertex that the ant comes from.

The essence of our ACO-Steiner algorithm is shown in Fig.1.

```
ACO-Steiner
INPUT: terminal set T
OUTPUT: an RSMT spanning T
1.  Generate Hanan Grid G based on T;
2.  Set the intensity in each edge in G to be p0;
3.  While loopNum < MAXLOOP do
4.    ConstructSteinerTreeByACO (T, G);
5.    Update the trail intensity in every edge by (2);
6.    Update the current best solution;
7.    loopNum++;
8.  Return the current best solution;
```

Fig.1. ACO-Steiner algorithm.

In Fig.1, the MAXLOOP is the maximum loop number, and the sub-procedure ConstructSteinerTreeByACO is to construct a Steiner tree using ant colony optimization, which is shown in Fig.2.

```
ConstructSteinerTreeByACO
INPUT: terminal set T, connection graph
OUTPUT: a rectilinear Steiner tree spanning T
1.  Place an ant on each vertex in the terminal set T
      and put the vertex into its tabu-list;
2.  Set the current sub-tree t empty;
3.  While ant number > 1 do
4.    Select an ant m randomly;
5.    AntMove (m);
6.    Add the edge m passing into t;
7.    If m meets m1 then
8.      Add vertices in tabu-list of m to that of m1;
9.      m dies;
10.     Relocate(m1);
11. Prune(t);
12. Return t;
```

Fig.2. Sub-procedure of ConstructSteinerTreeByACO.

In Fig.2, the sub-procedure AntMove decides the next vertex that the current ant will move to. The input of this procedure is the ant's current position. AntMove is shown in Fig.3.

In the 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 sub-procedure Relocate to relocate the survival ant to 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 not be a terminal. So we use the procedure Prune to delete all 1-degree non-terminal vertices in the tree.

AntMove
INPUT: current location of ant m
1. Compute the p_j of m by (1) and (4);
2. **If** $p_i == 0$ $(i = 1, 2, 3, 4)$ **then**
3. **Deconfuse** (m);
4. Ant m moves to j under the probability of p_j;
5. Add vertex j to the tabu-list of m;

Fig.3. Sub-procedure of AntMove.

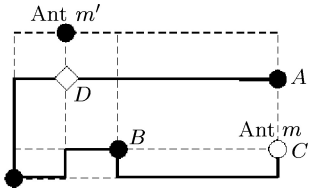

Fig.4. Instance for confused situation.

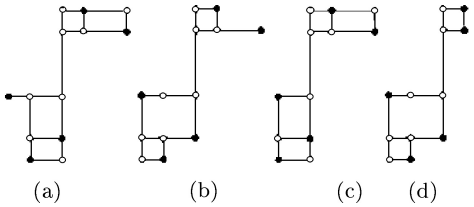

Fig.5. (a) Convex-hull reduction. (b) FST reduction. (c), (d) Terminal reduction.

The sub-procedure Deconfuse in Fig.3 solves the following problem. Sometimes, an ant may have no available vertices to move (see Fig.4). We solve this problem by moving this ant to some other vertex in its own tabu-list. But this new location should be closest to one of other ants.

We can see an example in Fig.4. There are two ants (m and m') left after several iterations. 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, which makes ant m not move any more but stay in the current position. So, we must find a new position for ant m. We should find this new position in the tabu-list of ant m and make the new position be the closest one to the other ants. Following this rule, we find vertex D as the new position of ant m, which is the closest to ant m'.

3.2 Graph Reduction Methods

Since ACO-Steiner algorithm is performed based on Hanan grid, we can reduce the Hanan grid so as to reduce the searching space, which is based on the following theorem. We use the following three methods, convex-hull reduction, fulsome Steiner tree (FST) reduction, and terminal reduction.

3.2.1 Convex-Hull Reduction

[22] proved that any non-terminal vertex that is adjacent to exactly two orthogonal edges e_1 and e_2 can be deleted if the other two edges forming a rectangle with e_1 and e_2 are present. The vertices remaining, after this reduction (see Fig.5(a)) is performed, are precisely those that lie within the rectilinear convex-hull of the terminals. For small and randomly generated sets of terminals, it is typically quite effective.

3.2.2 FST Reduction

M. Zachariasen *et al.*[17] proposed a graph reduction method by overlaying the generated FSTs on Hanan grid. They concluded that the number of Steiner points left in the grid is almost linear, approximately $3n$. In large scale problem, this reduction is very efficient. An example of FST reduction is shown in Fig.5(b).

3.2.3 Terminal Reduction

The convex-hull reduction and FST reduction often leave many terminals of 1-degree. Such terminals can be deleted (along with their adjacent edge), their neighbor can be made a terminal, and the appropriate edge can be added back into the final solution.

We call this the terminal reduction (see Fig.5(c)). The most significant effect of the terminal reduction is not the non-terminals it removes, but rather the fact that often two or more terminals collapse into a single new terminal[20].

We can perform convex-hull reduction or FST reduction followed by terminal reduction. The Hanan grid in Fig.1 can be reduced by these methods as shown in Fig.5. In practice, we find that convex-hull reduction is easier to implement and efficient in small scale problems, while FST reduction is more complex but very effective for large scale problems (only 0.3% Steiner points left in 1,000-terminal cases).

By using these reduction approaches, we can speed up our algorithm to some extent. However, the ACO-Steiner algorithm is still time consuming because the ants must move based on the Hanan grid and move only a small segment in the iteration. So, we make effort to shorten the running time to get high efficiency, which will be introduced in the next section.

4 Performance Improvement

4.1 Performance Improvement Approaches

We extend the tabu-list of each ant to record the edges instead of the vertices that this ant has visited. Every movement is not constrained by Hanan grid. Each time, an ant will choose the closest edge (here consider the vertex as a degenerate edge) out of its tabu-list, and move to this edge with the shortest path.

We define the distance between two edges as shown in Fig.6, in which, the distances between edge L_1 and edge L_2 in (a), (b), (c), (d) are h, $h + w$, h, and $h + w$, respectively.

If the shortest path is a line, there is only one way to move. However, if the shortest path has an L-shape, there are two possible ways to move, which are TOP_ORIENT and BOTTOM_ORIENT. We choose one orientation to move based on both the trail intensity and the topology.

Here, the trail will deposit in the four directions of terminal vertices instead of edges. Thus, we can decide which way to move by comparing the trail intensities in

different directions of the current vertex. The updating rule of trail intensity is still based on (2). Here, $\tau_{i,j}$ denotes the trail intensity in direction $i \to j$.

The topology is another factor in deciding the moving orientation. We compute the gains for each of the two possible orientations based on the following rules.

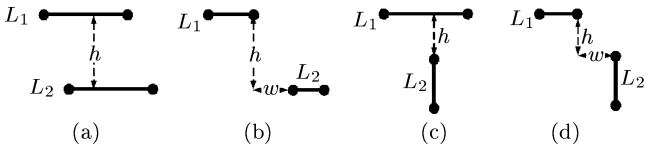

Fig.6. Definition of distance between two edges.

Firstly, for a given edge orientation we find the closest vertex (out of the current ant's tabu-list) to the edge as shown in Fig.7. Then, compute the distance D_c between this vertex to the edge in this orientation. Compute the distance D_f between this vertex and the edge in the opposed orientation. The gain in this orientation is D_f-D_c. We can see this rule from the instance shown in Figs. 7(c) and 7(d).

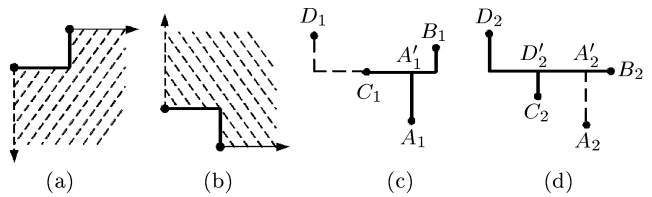

Fig.7. (a) BOTTOM_ORIENT. (b) TOP_ORIENT. (c) BOTTOM_ORIENT segment. (d) TOP_ORIENT segment.

The suffix of the vertex label is just to distinguish the two orientations. For example, A_1 and A_2 are the same vertex but in different situations.

In Figs. 7(c) and 7(d), the current location of an ant is in vertex B, and the closest vertex out of its tabu-list is vertex C. However, the path between B and C has two possible shapes, which are shown in Figs. 7(c) and 7(d), respectively. The path between B and C in Fig.7(c) is BOTTOM_ORIENT, and the one in Fig.7(d) is TOP_ORIENT. Now the ant wants to decide the orientation of edge (B, C). The closest vertex to edge (B_1, C_1) is A_1 with the distance $|A_1A'_1|$ and the distance between A_2 and edge (B_2, C_2) is $|A_2A'_2|$. So, the gain in BOTTOM_ORIENT is $|A_2A'_2| - |A_1A'_1|$ and the gain in TOP_ORIENT is $|D_1C_1| - |D_2D'_2|$.

When we compute the gain of one orientation we rewrite (1) as follows:

$$\eta_d = \frac{[gain_d]^\lambda}{dist_d} \tag{5}$$

where d is the two orientations (BOTTOM_ORIENT and TOP_ORIENT), $gain_d$ is the gain in orientation d, $dist_d$ is the distance from the closest vertex out of its tabu-list to the edge in orientation d, and λ is a constant that is the trade-off between the closest distance and gain.

Now, an ant can decide the orientation with the value given in (4).

We keep the main flow of ACO-Steiner algorithm given in Figs. 2 and 3, but improve the sub-procedure AntMove (see Fig.3) to obtain a higher performance (see Fig.8).

Since each leave vertex in the tree must be a terminal, the sub-procedure Prune in Fig.2 is unnecessarily any longer in the improved algorithm.

AntMove2
INPUT: current location of ant m
1. Compute the p_j of m by (4) and (5);
2. Ant m moves to j based on p_j;
3. Add the path from m to j into the tabu-list of m;

Fig.8. Improved sub-procedure of AntMove(m).

4.2 Discussions on the Improvement

In the original ACO-Steiner algorithm, an ant will step over one edge of Hanan grid in each iteration. In the worst case, ants' movements may cover the whole Hanan grid, so it will take $O(n^2)$ steps to complete a tree construction. In the improved algorithm, an ant may step over several edges of Hanan grid in each iteration, and one ant will be removed from the set of alive ants in each movement. So it will take only $O(n)$ steps to construct a tree.

To show the efficiency of the improvement approaches intuitively, we use the following case to simulate the processes of tree construction for the original ACO-Steiner algorithm given in Section 3 (see Fig.9) and the improved algorithm given in this section (see Fig.10).

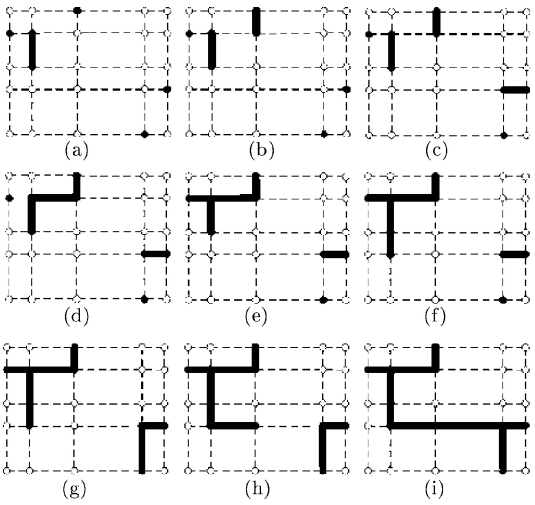

Fig.9. Original tree construction process.

We can find that the original one needs 9 steps to construct a tree, while the improved one needs only 4 steps. We can also find that these two algorithms can generate the same topologies of the final trees.

5 Experimental Results and Discussions

We have implemented ACO-Steiner algorithm in C++ and generated testing cases by using a sub-program provided by GeoSteiner 3.1. Then, perform GeoSteiner,

BGTC, and ACO-Steiner on a Sun V880 fire workstation with Unix operating system, respectively. We set $\alpha = 5$, $\beta = 1$, $\gamma = 1$, $\lambda = 3$, and $Q = 10{,}000$ in our experiments.

In the experiment, we find that the improvement method described in Section 4 is efficient. An ant decides the next node or the edge orientation greedily instead of stochastically with the possibility p. By using "greedy approach", we can maintain a quick convergence while keeping a good performance. Fig.11 shows the performance improvements in 50-, 100-, and 200-terminal RSMT instances. It shows that the algorithm gets the most improvement in the first 10 iterations, and improves little after 50 iterations. To get the best tradeoff between running time and performance, we set MAXLOOP as 10.

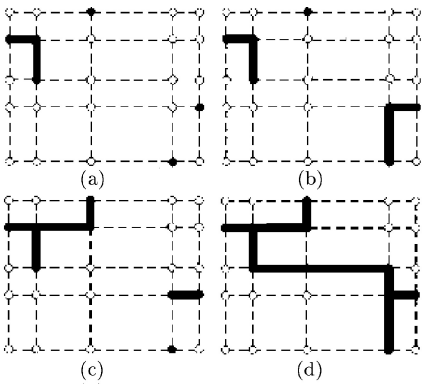

Fig.10. Improved tree construction process.

Table 1 shows the wire length and running time of results produced by the three algorithms.

From Table 1, we can see that ACO-Steiner performs well when the number of terminals is no more than 50. It can always achieve the optimal results and keep short runtime. When the number of terminals increases our algorithm can keep the performance within 1% worse than the optimal (GeoSteiner) with a very short runtime. When the number of terminals reaches 1,000, the GeoSteiner 3.1 fails to produce a result in our workstation.

Fig.12 Shows the result of ACO-Steiner for a 1,000-terminal tree.

Fig.13 shows the result of BGTC vs. ACO-Steiner in a case with 50 terminals. Fig.13(a) shows the result of BGTC with a wire length of 51,878 and Fig.13(b) shows the result of ACO-Steiner with a wire length of 51,595. The circles show the differences between two results.

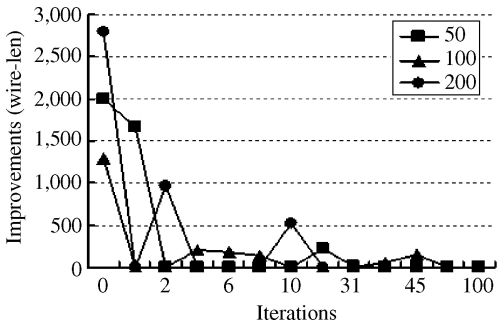

Fig.11. Performance improvements.

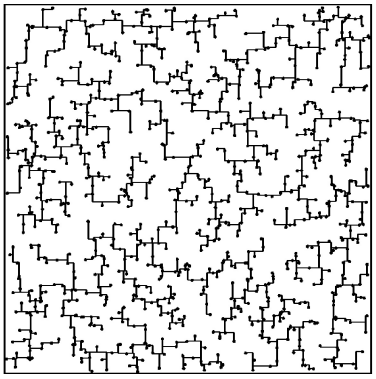

Fig.12. Result of ACO-Steiner for a 1,000-terminal tree.

In VLSI/ULSI routing, most of the nets in a circuit have no more than 50 terminals (clock tree/mesh design needs special methods). When the problem scale becomes larger, our algorithm can keep good performance in a short running time. So our ACO-Steiner can be used in practical applications well.

Our ACO-Steiner algorithm is based on graph, so it can be easily extended to solve other tree problems which can be formulated as problems in graph, such as obstacle avoiding rectilinear Steiner minimal tree (OARSMT) and congestion reduction in global routing.

To solve OARSMT problem, we can construct an Escape Graph[23] or other connection graphs based on the given terminals and obstacles, and perform our ACO-Steiner on a connection graph. The result tree on the graph is an RSMT avoiding all obstacles.

Table 1. Experimental Results of ACO-Steiner and the Comparison Results with BGTC and GeoSteiner 3.1

Terminal number	BGTC		ACO-Steiner				GeoSteiner	
	Wire length	CPU (s)	Wire length			CPU (s)	Wire length	CPU (s)
			Best	Ave.	Worst			
9	19,913	< 0.001	19,799	19,799	19,799	< 0.001	19,799	< 0.001
10	21,259	< 0.001	21,143	21,143	21,143	< 0.001	21,143	< 0.001
20	34,767	< 0.001	34,767	34,778	34,878	< 0.001	34,767	< 0.001
30	40,226	< 0.001	40,037	40,072	40,215	< 0.001	40,037	< 0.001
50	51,878	< 0.001	51,595	51,800	52,094	< 0.001	51,595	< 0.001
70	59,564	< 0.001	59,531	59,701	60,093	0.004	59,503	0.020
100	73,289	< 0.001	73,356	73,751	73,929	0.018	72,979	0.050
200	104,750	0.01	105,277	105,567	105,701	0.387	104,178	0.880
500	161,875	0.050	164,440	164,484	164,565	3.380	160,844	38.880
1,000	203,653	0.54	230,873	234,998	238,219	54.00	–	–

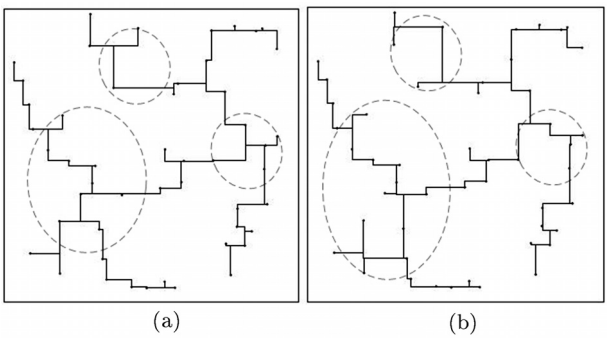

Fig.13. BGTC vs. ACO-Steiner for a 50-terminal tree. (a) BGTC. (b) ACO-Steiner.

In global routing, our ACO-Steiner can be used not only to construct initial trees for all nets, but also to improve routing trees in iterations in routing algorithms. We should note that ACO-Steiner can produce different topologies in different runs because of the random selection of an ant in each iteration in our algorithm (see Line 4 in Fig.2). Meanwhile, these result trees in different topologies keep almost the same performance (wire length). This property is useful to generate various routing trees for a net in global routing, so as to reduce congestion scarifying only a little wire length. Fig.14 shows the three different result trees for a 70-terminal case, the wire lengths of these three topologies are 59,737, 59,551, and 59,531 respectively.

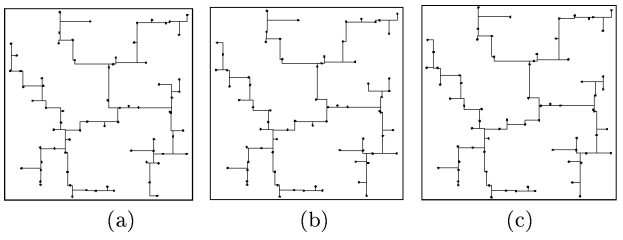

Fig.14. Different topologies for a 70-terminal case. (a) cost=59,737. (b) cost=59,551. (c) cost=59,531.

6 Conclusions

In this paper, we propose a heuristic for RSMT construction based on ACO. Then, we use some strategies to speed up the algorithm. The experimental results show that our heuristic ACO-Steiner keeps high performance with a very short running time. Furthermore, our ACO-Steiner can be easily extended to solve other tree-construction problems in graph.

We find that there is still room for improvement in our work. We will continue to improve the performance in wire length of ACO-Steiner while keeping short running time, which is regarded as our future work.

References

[1] Carden R C IV, Li J M, Cheng C K. A global router with a theoretical bound on the optimal solution. *IEEE Trans. Computer-Aided Design*, Feb. 1996, 15: 208–216.

[2] Jing T, Hong X L, Bao H Y, Xu J Y, Gu J. SSTT: Efficient local search for GSI global routing. *J. Computer Science and Technology*, 2003, 18(5): 632–639.

[3] Jing T, Hong X L, Xu J Y, Bao H Y, Cheng C K, Gu J. UTACO: A unified timing and congestion optimization algorithm for standard cell global routing. *IEEE Trans. CAD*, 2004, 23(3): 358–365.

[4] Xiang H, Tang X P, Wong D F. An algorithm for integrated pin assignment and buffer planning. In *Proc. ACM/IEEE Design Automation Conf.* (*DAC*), 2002, pp.584–589.

[5] Hong X L, Jing T, Xu J Y, Bao H Y, Gu J. CNB: A critical-network-based timing optimization method for standard cell global routing. *J. Computer Science and Technology*, 2003, 18(6): 732–738.

[6] Xu J Y, Hong X L, Jing T, Cai Y C, Gu J. A novel timing-driven global routing algorithm considering coupling effects for high performance circuit design. *IEICE Trans. Fundamentals of ECCS*, 2003, E86-A(12): 3158–3167.

[7] Wang Y, Hong X L, Jing T, Yang Y, Hu X D, Yan G Y. An efficient low-degree RMST algorithm for VLSI/ULSI physical design. *Lecture Notes in Computer Science* (*LNCS*) *3254 – Integrated Circuits and System Design*, Santorini, Greece, Sept. 2004, pp.442–452.

[8] Xu J Y, Hong X L, Jing T, Cai Y C, Gu J. An efficient hierarchical timing-driven Steiner tree algorithm for global routing. *INTEGRATION, VLSI J.*, 2003, 35(2): 69–84.

[9] Garey M R, Johnson D S. The rectilinear Steiner tree problem is NP-complete. *SIAM Journal on Applied Mathematics*, 1977, 32: 826–834.

[10] Hwang F K, Richards D S, Winter P. The Steiner Tree Problem, Annals of Discrete Mathematics. Amsterdam: North-Holland, The Netherlands, 1992.

[11] Kahng A B, Robins G. A new class of iterative Steiner tree heuristics with good performance. *IEEE Trans. Computer-Aided Design*, July 1992, 11: 893–902,

[12] Borah M, Owens R M, Irwin M J. An edge-based heuristic for Steiner routing. *IEEE Trans. Computer Aided Design*, 1994, 13: 1563–1568.

[13] Kahng A B, Mandoiu I I, Zelikovsky A Z. Highly scalable algorithms for rectilinear and octilinear Steiner trees. In *Proc. Asia and South Pacific Design Automation Conference* (*ASP-DAC*), Kitakyushu, Japan, 2003, pp.827–833.

[14] Zhou H. Efficient Steiner tree construction based on spanning graphs. In *Proc. ACM ISPD*, Monterey, CA, USA, 2003, pp.152–157.

[15] Qi Zhu, Hai Zhou, Tong Jing, Xianlong Hong, Yang Yang. Spanning graph-based nonrectilinear Steiner tree algorithms. *IEEE Trans. CAD*, 2005, 24(7): 1066–1075.

[16] Warme D M, Winter P, Zachariasen M. 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.

[17] Zachariasen M. Rectilinear full Steiner tree generation. Technical Report DIKU-TR-97/29, Department of Computer Science, University of Copenhagen, December 1997.

[18] Dorigo M, Maniezzo V, Colorni A. The Ant System: Optimization by a colony of cooperating agents. *IEEE Trans. Systems, Man, and Cybernetics—Part B*, 1996, 26(1): 1–13.

[19] Das S, Gosavi S V, Hsu W H, Vaze S A. An ant colony approach for the Steiner tree problem. In *Proc. Genetic and Evolutionary Computing Conference*, New York City, New York, 2002.

[20] Ganley J L. Computing optimal rectilinear Steiner trees: A survey and experimental evaluation. *Discrete Applied Mathematics*, 1998, 89: 161–171.

[21] Hanan M. On Steiner's problem with rectilinear distance. *SIAM Journal on Applied Mathematics*, 1966, 14: 255–265.

[22] Yang Y Y, Wing O. Suboptimal algorithm for a wire routing problem. *IEEE Trans. Circuit Theory*, September 1972, 19: 508–510.

[23] Ganley J L, Cohoon J P. Routing a multi-terminal critical net: Steiner tree construction in the presence of obstacles. In *Proc. IEEE International Symposium on Circuits and Systems*, London, UK, 1994, pp.113–116.