CDCTree: Novel Obstacle-Avoiding Routing Tree Construction based on Current Driven Circuit Model

Yiyu Shi[1], Tong Jing[2], Lei He[1],

Electrical Engineering Department[1]
UCLA[1]
Los Angeles, California, 90095-1594, USA
Tel: 310-206-2037
Fax: 310-206-4685
e-mail: {yshi, lhe}@ee.ucla.edu[1]

Zhe Feng[2] and Xianlong Hong[2]

Computer Science & Technology Department[2]
Tsinghua University[2]
Beijing, P.R.China 10084
Tel: +86-10-62785564
Fax: +86-10-62781489
e-mail: {jingtong, hxl-dcs}@tsinghua.edu.cn[2]

Abstract— Routing tree construction is a fundamental problem in modern VLSI design. In this paper we propose CDCTree, an Obstacle-Avoiding Rectilinear Steiner Minimum Tree (OARSMT) heuristic algorithm to construct an OARSMT. CDCTree is based on the current driven circuit (CDC) model mapped from an escape graph. The circuit structure comes from the topology of the escape graph, with each edge replaced by a resistor indicating the wirelength of that edge. By performing DC analysis on the circuit and selecting the edges according to the current distribution to construct an OARSMT, the wirelength of the resulting tree is short. The algorithm has been implemented and tested on cases of different scales and with different shapes of obstacles. Experiments show that CDCTree can achieve shorter wirelength than the existing best algorithm, An-OARSMan, when the terminal number of a net is less than 50.

I. Introduction

Routing tree construction is one of the major tasks in the routing phase. When wirelength is of interest, it is in essence an RSMT (*Rectilinear Steiner Minimum Tree*) construction problem on a given terminal set. In practical full-chip routing or detailed routing applications, we consider macro cells, IP blocks, and pre-routed nets as obstacles. Therefore, powerful algorithms of OARSMT (*Obstacle-Avoiding Rectilinear Steiner Minimum Tree*) construction are required to get short wirelength.

The RMST problem has been proved to be NP-complete [1]. However, the OARSMT problem is even more complicated and no polynomial-time algorithms have been proposed to solve it precisely. Maze algorithm was proposed in [2] and several improvements on searching efficiency were made in [3]-[6] later.

The method of line search routing was introduced in [7] and [8]. However, these algorithms are only suitable for small-scale problems. Most existing OARSMT algorithms use the multi-terminal variant of the maze algorithm, which incurs the same space demand as that of the two-terminal variant with a result far from the optimal. [9] proposed an algorithm to construct optimal three-terminal or four-terminal OARSMT. Then, G3S, G4S, and B3S heuristics were proposed for the cases with less than twenty terminals. [10] provided an exact algorithm to find an obstacle-avoiding Euclidean Steiner tree with less than 150 terminals. [11] introduced an *O(mn)* two-step heuristic of OARSMT, in which *m* is the number of obstacles and *n* is the number of terminals. The two-step heuristic works well when the terminal number is less than seven and the obstacles are convex. The most recent works are FORst [12] and An-OARSMan [13]. FORst can tackle large scale problems efficiently. An-OARSMan, based on the track graph put forward in [14], can achieve shorter wirelength than FORst when the terminal number is less than 100.

In practice, most nets in circuits are in common scale. If we research on powerful OARSMT algorithms to get short wirelength for such nets, we can have more opportunities to get better routing results. Meanwhile, there is still much room to improve the wirelength performance.

The main contribution of this paper is CDCTree, a heuristic algorithm of OARSMT construction based on a current driven circuit (CDC) model. The idea of CDCTree is different from those of existing algorithms. It maps the edges of escape graph into resistors, and adds a current source at each terminal. Then it makes use of Coulomb's Law, which indicates the repellency of currents, to construct an RSMT. Experimental results show that CDCTree can achieve shorter wirelength than An-OARSMan when the terminal number is less than 50. In addition, CDCTree can route among both convex and concave

The work at UCLA was partially supported by NSF award CCR-0093273/0401682. The work at Tsinghua University was partially supported by the NSFC under Grant No.60373012, the SRFDP of China under Grant No.20050003099.

0-7803-9451-8/06/$20.00

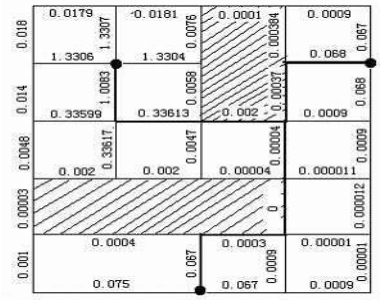

Fig. 1. Current distribution for a three-terminal current driven circuit model

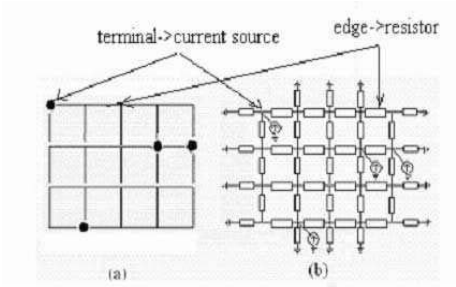

Fig. 2. An example of topology mapping from (a) escape graph to (b) circuit structure

polygon obstacles.

The rest of the paper is organized as follows: In Section II, we discuss how *Coulomb's Law* can be applied to the OARSMT construction. In Section III, the detailed procedure of tree construction based on CDC model is described in detail. Section IV shows the experimental results and some discussions. Conclusions and remarks are given in Section V.

II. Coulomb's Law in OARSMT Construction

The foundation of our CDCTree algorithm is different from the existing algorithms. In order to clearly describe the idea, we first describe how Coulomb's Law can be applied to the OARSMT problem.

Coulomb's Law indicates that charges of the same kind (negative or positive) are repellent. [15] first employed this law to solve planning problems. But we notice that this law can also be used in optimization problems. Basically speaking, electrical currents are composed of electrons, all of which are negatively charged. According to Coulomb's Law, the currents repel from each other. Here, we introduce this idea into the tree construction problem.

If we map an escape graph [9] into a circuit and inject current at each terminal, the current distribution of this current driven circuit can be obtained by performing DC analysis. Then the edges with the minimum currents are selected to construct the Steiner tree. The total length of the tree thus constructed should be short because the currents along the edges which are close to all the terminals are significantly repelled by current injections at those terminals. We observe that RSMT is usually constructed by the edges close to all the terminals. Edges far from some of the terminals are seldomly selected. Therefore, the tree constructed by the edges with minimum current has a short wirelength.

Figure 1 illustrates the current distribution in a three-terminal current driven circuit model. Currents are injected at the three vertices marked in black. Each edge represents a resistor with a given value. The RMST is composed by the edges in bold black lines. It can be easily verified that the currents flowing along the edges of the RSMT are small. Note that not all the edges with minimum currents are selected because of the *path selection* rules, which is discussed in the next section.

III. The CDCTree Algorithm

CDCTree algorithm is composed of four steps: *topology mapping, resistor selection, circuit simulation, and path selection*. The four steps are discussed in detail below.

A. Topology Mapping

CDCTree algorithm is based upon the escape graph [9], which enables us to get short wirelength. This step deals with the mapping from escape graph to circuit structure.

We first place a resistor at each edge of the escape graph, the value of which is to be decided in the next step, i.e. *resistor selection*. Next, we add a current source at each terminal. The value of the current source can be arbitrarily chosen as it does not influence the relative distribution of the currents. The edge with larger current always has larger current regardless of the current source value. In our experiments we set the current sources to be *5A*.

Finally, to let the circuit function correctly, we have to decide how to connect the circuit to the ground. One method is to extend the escape graph to infinite size. Then the nodes at infinite are automatically connected to ground. However, this method requires a large memory as well as high computation cost because we have to calculate the currents at each edge while the total edge number is infinite. Therefore, this method is impractical.

An alternative way is to connect the periphery nodes of the circuit to ground. Though much simplified, this method brings about another problem. Unlike the infinite structure, the finite one influences the current distribution of the circuit significantly. If we let the periphery nodes connect to the ground via a resistor, the side effects can be alleviated to some extent. In the fourth step (*path selection*), further techniques are employed to compensate for the side effects.

An example of topology mapping from the escape graph to the circuit structure is given in Figure 2. The terminals and edges in (a) are mapped into current sources and resistors in (b), respectively.

B. Resistor Selection

There are three different types of resistors in our circuit model. The first type, GND resistors, are used to connect the periphery nodes of the circuit to the ground. The second type, EDGE resistors, refer to the resistors on the edges of

the graph except for the obstacle boundaries. The last type, OBST resistors, are the resistors on the edges of the obstacle boundaries.

1) GND resistors: GND resistors are used to provide paths for currents to flow to the ground. In our experiments, we set GND resistors to be a fixed small value, 0.1.

2) EDGE resistors: According to Ohm's Law, the larger the resistor, the smaller the current flowing through it. Since we choose edges based on minimum currents, we want less current to flow through paths with larger wirelength. Therefore, the longer the edge, the smaller the EDGE resistor should be. A mapping function $f(x)$ should thus be decided by which EDGE resistor R can be calculated as

$$R = f(L) \tag{1}$$

where L is the length of the edge. The slope of $f(x)$ cannot be too large, otherwise it may significantly influence the current distribution. This causes the resulting tree to be constructed simply by the shortest edges without considering the topology of the tree. On the other hand, the slope can not be too small, which leads to neglecting the impact of the edge length. After experiments, we select the following function.

$$f(x) = K - ln(x) \tag{2}$$

where K must be large enough to guarantee $f(x)$ to be positive in the range of x, i.e. edge lengths. In our experiments, the edge lengths are set to be between 100 and 10,000, so we choose K to be 10. The mapping function turns out to be

$$R = 10 - ln(L) \tag{3}$$

In general cases, K should be larger than the logarithm of the maximum wirelength to keep the resistor value positive.

3) OBST Resistors: The existence of obstacles to some extent destroys the characteristics of the original circuit model. From our experiments, we find out that currents tend to gather at the edges of obstacles, which can be called as *Current Crowding Effect (CCE)*. To compensate for CCE, the resistors on obstacle edges should be set larger. In our experiments, we simply add a fixed value 10 to the value calculated from (3).

Following the four steps, we construct a current driven circuit model.

C. Circuit Simulation

The current through each edge can be solved by using NA (*Nodal Analysis*) method [16]. The main idea of this method is based upon KCL (*Kirchhoff's Current Law*), which states that the algebraic sum of the currents flowing into each node of the circuit must be zero. Set the voltages at the nodes as variables and the current through each edge can be expressed by those variables according to *Ohm's Law*. Then, apply KCL to get a set of linear equations. These equations can be written in matrix form as *Ax=b*, where *A* is a positive definite diagonally dominant sparse matrix and *b* is the port incidence matrix [16]. There are numerous efficient algorithms to solve it, such as *ICCG* (Incomplete Cholesky-conjugate Gradient method) [17], etc. For convenience's sake, we simply use Gaussian-Seidel iteration method in our experiments to verify the theory of CDCTree.

D. Path Selection

This step is of great importance as it directly decides the quality of the tree. The step is composed of three sub-functions: *preprocess*, *growth*, and *reduction*. The overall procedure of the *path selection* is shown in Algorithm 1. It first moves inside the terminals at periphery to alleviate the deficiency brought by GND resistors (in sub-function *preprocess*). Then, it decides what to do in each iteration according to the situation of surrounding terminals, whether *reduction* or *growth*.

The key point for this algorithm is to select the edges with the minimum currents. This is also the goal of sub-function $growth$. However, as we mentioned above, there are other factors that influence the current distribution. Some strategies must be employed to make up for the disturbance. The first and second sub-functions, *preprocess* and *reduction*, are put forward for this purpose.

This algorithm cannot guarantee that all the terminals can be connected. Therefore, a maximum iteration number must be set. If the above algorithm does not converge successfully after a certain number of iterations, it is forced to exit. The remaining unconnected terminals are connected using Maze Algorithm. This situation happens when there are minimum current loops in the circuit. In our experiments, we observe the unconnected terminal number is always less than three.

Algorithm 1 Overall Procedure of *Path Selection*

```
INPUT: terminal_number, terminal_list and current distribution of the escape graph;
OUTPUT: OARSMT;
initialization: active_number = terminal_number;
initialization: active_list = terminal_list;
initialization: dead_list = Φ
call: preprocess;
while active_number ≥ 2 do
  i = 0;
  Δactive_number = 0;
  while i < active_number do
    current_active_vertex = active_list(i);
    if Any neighboring vertices of current_active_vertex has already been selected then
      call: reduction;
    else
      call: growth;
    end if
    i ++;
  end while
  active_number = active_number - Δactive_number;
end while
if dead_list ≠ Φ then
  Use maze algorithm to connect the vertices in the dead_list sequentially;
end if
```

The sub-function *preprocess* moves the periphery terminals inside. The periphery vertices are connected to ground via a small resistor. So the current distribution at periphery is

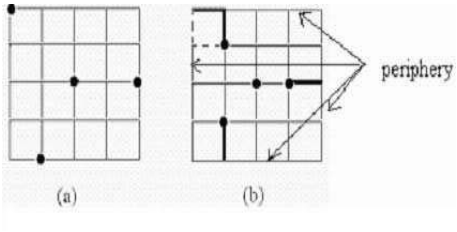

Fig. 3. Terminal distribution on a 5×5 grid: (a) Before preprocess and (b) After preprocess

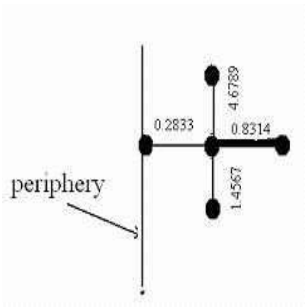

Fig. 4. An illustration of subfunction *growth at a vertex near periphery*

influenced. To avoid this, we move the periphery terminals inside to a new position, as shown in Figure 3. The terminals are denoted by black dots and the edges selected in the *preprocess* stage are denoted by bold lines. Note that the terminal on the upper-left corner is moved right and then down along the edges in black. It can also be moved inside along the dashed edges as the total wirelength remains the same.

A vertex is "active" when it is newly added to the Steiner tree during tree *growth*. For each active vertex, the subfunction *growth* selects the available neighboring edge with minimum current, adds the vertex at the other end of the edge to the current active vertex list, and removes itself from the active vertex list. An edge is not available when it connects the current active vertex with a vertex on the periphery. This is because the selection of this edge will cause the new active vertex to be moved to the periphery, where the current distribution is signifi cantly influenced by the GND resistors.

Algorithm 2 provides the procedure of sub-function *growth*. Figure 4 shows an example of *growth* where the vertex in the center is the current active vertex and the value beside each edge is the current flowing through it. Although the edge heading left has the minimum current (0.2833), it can not be selected because it is not available. Therefore, the edge heading right is selected as it has the minimum current among the available edges.

Algorithm 2 Procedure of the sub-function $growth$

INPUT: current_active_vertex and the current distribution of the escape graph;
OUTPUT: updated dead_list, updated active_list, updated current_active_vertex and updated Δactive_number;
search for the edge e with the minimum current among all the available edges residing on the current_active_vertex;
if no such edge exists **then**
  remove current_active_vertex from active_list;
  add current_active_vertex to dead_list;
  Δactive_num - -;
else
  update current_active_vertex along edge e ;
  mark edge e as selected;
  mark current_active_vertex(i) as selected;
end if

The sub-function *reduction* is used to alleviate the influence of GND resistors. The periphery vertices are connected to ground via a small resistor, thus influencing the current distribution. To compensate for this, *reduction* changes the tree structure when necessary. It functions when any of the neighboring vertices has already been selected. When one of the neighboring vertices has already been selected but has not been connected with the current active vertex yet, add the edge connecting them to the tree. Otherwise if it has already been connected with the current active vertex, add the edge connecting them to the tree and delete the edges on the path connecting these two edges until the connectivity of the tree is destroyed.

The detailed procedure of *reduction* is shown in Algorithm 3. Figure 5 is an example of *reduction*. The original tree is $A-B-C-D-E-F-G-H-I-J$. The improved tree after *reduction* is $A-B-C-F-G-H-I-J$, showing edges $C-D-E-F$ being replaced by edge $C-F$.

Algorithm 3 Procedure of the sub-function *reduction*

INPUT: current_active_vertex and the current distribution of the escape graph
OUTPUT: updated dead_list, updated active_list, updated current_active_vertex and updated Δactive_number;
if any neighboring vertex is selected but it is not connected with current_active_vertex **then**
  select the edge e connecting them;
  mark edge e as selected;
  delete current_active_vertex from active_list;
  Δactive_number - -;
end if
if any neighboring vertex is selected and it is already connected with current_active_vertex **then**
  select the edge e connecting them;
  delete the edges on the path connecting these two vertices until the connectivity of the tree is destroyed;
end if

IV. Experimental Results and Discussions

We have implemented the CDCTree algorithm in C language. It was reported that the An-OARSMan algorithm produces the shortest wirelength among existing algorithms when routing less than 100 terminals [13]. Therefore, we compare our results with those of the An-OARSMan on an 800MHz Sun V880 fi re workstation with Unix operating system. We

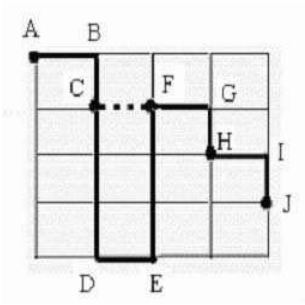

Fig. 5. An illustration of sub-function *reduction on a 5×5 grid*

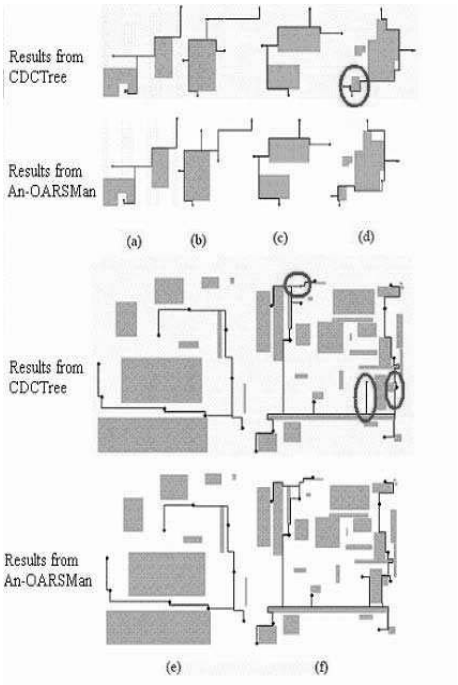

Fig. 6. Results comparison between CDCTree and An-OARSMan

test the cases presented in [13] using the CDCTree algorithm and the comparison between their results are shown in Figure 6. Both algorithms achieve the same optimal results in the case (a), (b), (c), and (e). In the cases (d) and (f), the results of CDCTree are better than those of An-OARSMan as shown in red circles.

Figure 7 provides more examples to show the advantages of CDCTree over An-OARSMan. In Figure 7, (a) and (b) are the results of CDCTree, while (c) and (d) are their respective counterparts produced by An-OARSMan. Improvements are shown in red circles. It is obvious that CDCTree algorithm produces shorter wirelength than An-OARSMan does.

Table 1 shows the comparison results upon test cases with a different number of terminals and obstacles. For each scale, we randomly select several different cases and represent the average results. The first two columns are the terminal number and obstacle number of the test cases. The next four columns are the wirelength and runtime of An-OARSMan and CDCTree, respectively. From Table 1, we can see that when the terminal number is less than 50, CDCTree can achieve better result than An-OARSMan.

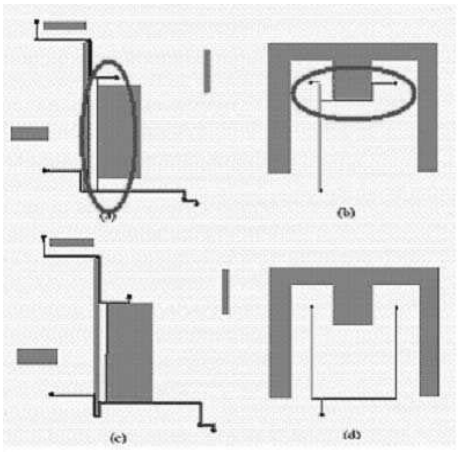

Fig. 7. Further results comparison between CDCTree and An-OARSMan

TABLE I
COMPARISON BETWEEN AN-OARSMAN AND CDCTREE

Terminal #	Obstacle #	An-OARSMan Length	An-OARSMan Runtime (s)	CDCTree Length	CDCTree Runtime (s) I	CDCTree Runtime (s) II
3	3	2350	<0.01	2350	<0.01	<0.01
5	3	4380	0.01	4350	0.01	<0.01
7	5	9610	0.04	9610	0.02	0.01
10	5	11340	0.06	10980	0.08	0.02
20	7	14790	0.28	13110	0.84	0.13
30	10	21220	0.77	19970	1.38	0.25
40	10	28600	1.98	27300	4.32	0.77
50	15	31330	3.22	31310	10.25	1.02

In physical design, most nets have less than 50 terminals, with the exception of clock trees and power grids. In addition, the terminals of the same net are usually placed close to each other in the placement stage. Therefore, CDCTree algorithm is practical in routing applications.

The time consumed by CDCTree can be divided into two parts. Part I is the time used to solve linear equations and Part II is the time used to construct OARSMT. CDCTree consumes a majority of the time in Part I. This is because we use the Gaussian-Seidel algorithm to solve the equations, which is extremely inefficient. As the coefficient matrix for the equations are sparse, positive-definite and symmetric, ICCG or other efficient algorithms can be employed to dramatically reduce the run time. It is reported that ICCG is about $8,000X$ faster than the point Gaussian-Seidel method in solving linear equations [17]. In addition, more advanced algorithms can be employed to further reduce the runtime. For example, an improved ICCG method was put forward in [18], which can reduce the computation time by $30\% - 50\%$ compared with the ICCG method. Therefore, by using these algorithms, the runtime for equation-solving can be reduced to be small enough to be neglected.

V. SUMMARY AND CONCLUSIONS

This paper focuses on OARSMT routing tree construction. An heuristic algorithm named CDCTree is presented based on a current driven circuit model. The algorithm maps the

escape graph into a circuit structure and replaces each edge with one resistor. By performing DC analysis on the circuit and selecting among the edges with the minimum currents to construct the OARSMT, the resulting tree has a short wirelength. Experimental results show that CDCTree achieves better results than the existing best algorithm An-OARSMan when the terminal number of a net is less than 50.

ACKNOWLEDGEMENT

The paper describes the research work performed cooperatively at University of California, Los Angeles (UCLA), USA and Tsinghua University, Beijing, P.R.China. The authors wish to thank the anonymous reviewers for their helpful comments.

REFERENCES

[1] 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.

[2] C.Y.Lee, "An algorithm for path connections and its applications", *IRE Trans. on Electronic Computers,* 1961, 10: pp. 345-346.

[3] S.B.Akers, "A Modification of Lee's Path Connection Algorithm", *IEEE Trans. on Electronic Computer,* 1967, 16 (4): pp. 97-98.

[4] J.Soukup, "Fast Maze Router", In *Proc. of the 15th IEEE/ACM Design Automation Conferece,* 1978: pp. 100-102.

[5] Hadlock, "A Shortest Path Algorithm for Grid Graphs", *Networks,* 1977(7): pp. 323-334.

[6] F.Rubin, "The Lee Connection Algorithm", *IEEE Trans. on Computer,* 1974(23): pp. 907-914.

[7] D.W.Hightower, "A Solution to the Line Routing Problem on the Continous Plane", in *Proc. of the 6th Design Automation Conference,* 1969: pp. 1-24.

[8] K.Mikami and K.Tabuchi, "A Computer Program for Optimal Routing of Printed Circuit Connectors", in *Proc. of IFIPS*, 1968, H47: pp. 1475-1478.

[9] 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.

[10] 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.

[11] 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.

[12] Yu Hu, Zhe Feng, Tong Jing, Xianlong Hong, Yang Yang, Ge Yu, Xiaodong Hu, Guiying Yan, "FORst: A 3-Step Heuristic for Obstacle-Avoiding Rectilinear Steiner Minimal Tree Construction", *Journal of Information and Computational Science,* 2004, 1(3): 107-116.

[13] Yu Hu, Tong Jing, Xianlong Hong, Zhe Feng, Xiaodong Hu, and Guiying Yan, "An-OARSMan: Obstacle-Avoiding Routing Tree Construction with Good Length Performance", in *Proc. of IEEE/ACM Asia and South Pacific Design Automation Conference,* 2005, Shanghai, China, pp. 7-12.

[14] 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.

[15] Yiyu Shi, Bike Xie and Yanjie Mao, "Circuit Model in Mathematical Modeling", *Journal of Mathematical Engineering,* 2004, 21(7): pp. 43-48.

[16] Ho C, Ruehli, Brennan P. "The Modified Nodal Approach to Network Analysis", *IEEE Trans. Circuits and Systems,* 1975, 39(22): pp. 504-509.

[17] D. S. Kershaw, "The incomplete Cholesky-conjugate gradient method for the iterative solution of systems of linear equations", *Journal of Computational Physics,* 26 I (Jan.) (1978), 43-65.

[18] J. Wang, D. Xie and Y. Yao, "The Modified Solutoin for Large Sparse Symmetric Linear Systems in Electromagnetic Field Analysis", *Transactions of China Electrotechnical Society,* 2001 16(2): pp. 26-29.