

# FORst: A 3-step heuristic for obstacle-avoiding rectilinear Steiner minimum tree construction<sup>\*</sup>

Yu Hu<sup>a</sup>, Zhe Feng<sup>a</sup>, Tong Jing<sup>a,†</sup>, Xianlong Hong<sup>a</sup>, Yang Yang<sup>a</sup>,  
Ge Yu<sup>b</sup>, Xiaodong Hu<sup>c</sup>, Guiying Yan<sup>c</sup>

<sup>a</sup> Department of Computer Science and Technology, Tsinghua University, Beijing 100084, P. R. China

<sup>b</sup> School of Information Sci. and Eng., Northeastern University, Shenyang 110004, P. R. China

<sup>c</sup> Institute of Applied Mathematics, Chinese Academy of Sciences, Beijing 100080, P. R. China

---

## Abstract

Macro cells, IP blocks, and pre-routed nets are often regarded as obstacles in VLSI routing phase. Obstacle-avoiding rectilinear Steiner minimum tree (OARSMT) algorithms are often used to meet the needs of practical routing applications. However, OARSMT algorithms with multi-terminal nets routing still do not satisfy the requirements of practical applications. This paper presents a 3-step heuristic, named FORst, to tackle the OARSMT problem. In Step1, we partition all terminals into some subsets in the presence of obstacles. Then in Step2, we connect terminals in each connected graph with one or more trees, respectively. In Step3, we connect the forest consisting of trees constructed in Step2 into a completed Steiner tree spanning all terminals while avoiding all obstacles. Two algorithms, called ACO-RSMT and GFST-RSMT, are proposed to construct an OARSMT in a connected graph in Step2, which are suitable for different situations. This algorithm has been implemented and tested on cases with typical obstacles. The experimental results show that FORst is with great efficiency and can get good performance. Moreover, it can tackle large scale nets among complex obstacles, such as a net with 1000 terminals in the presence of 100 rectangular obstacles.

*Keywords:* Rectilinear Steiner Minimum Tree (RSMT); Obstacle-Avoiding; IC CAD; VLSI; routing.

---

## 1. Introduction

Routing a net, finding a rectilinear Steiner minimum tree (RSMT) for a given terminal set, is one of the fundamental problems in integrated circuit computer-aided design (IC CAD). However, only a few RSMT algorithms take obstacles into consideration. Macro cells, IP blocks, and pre-routed nets are often regarded as obstacles and RSMT construction among obstacles is often used as the estimation for wire length and delay throughout the process of routing. So, we need efficient obstacle-avoiding RSMT (OARSMT) algorithms in IC CAD. Ref. [1] proved that the

---

<sup>\*</sup> 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

<sup>†</sup> Corresponding author.

E-mail address: jingtong@tsinghua.edu.cn

RSMT problem is NP-complete. So, OARSMT problem is more complicated and no polynomial-time algorithm can solve it exactly.

Since it has been paid less attention to multi-terminal net routing among obstacles due to great complexity and difficulty, the VLSI designers always use the multi-terminal variant of maze routing [2] algorithms. Ganley *et al* [3] proposed an algorithm to compute the optimal 3-terminal or 4-terminal RSMT in the presence of obstacles. Then, G3S, B3S, and G4S heuristics [3] are proposed for larger scale cases (but less than 20 terminals). The running time is still long. Zachariassen *et al* [4] proposed an efficient exact algorithm to find an obstacle-avoiding Euclidean Steiner tree with less than 150 terminals. But the complexity of this algorithm is exponential and no rectilinear exact algorithms are reported at present. Zhou *et al* introduced a generalized connection graph [5] and proposed an efficient algorithm to compute OARSMT for a 3-terminal net. The recent progress in OARSMT is an  $O(mn)$  2-step heuristic presented in [6], which works well when the number of terminals is less than 7 and the obstacles are convex polygons. We find that the existing algorithms tackling OARSMT problem are more suitable for small scale and simple cases. That is, it contains only a few terminals and obstacles. The shape of obstacles is always constrained to be convex polygons.

The main contribution of this paper is the 3-step heuristic, called FORst, for the obstacle-avoiding rectilinear Steiner minimum tree construction. FORst can handle complex cases such as many terminals and concave polygon obstacles, keep the high performance, and take short running time. Two algorithms, called ACO-RSMT and GFST-RSMT, are proposed to construct an OARSMT in a connected graph in Step2, which are suitable for different situations. The FST (full Steiner tree) construction, ant colony optimization (ACO) technology, and detour method are used in our algorithm, which makes it be more practical for many cases.

The rest of this paper is organized as follows. In Section 2, we give our FORst algorithm in detail. Section 3 shows the experimental results and Section 4 concludes the whole paper.

## 2. Our FORst Algorithm

**Definition 1. (OARSMT)** Given a set  $T$  of  $n$  points, called terminals, and a set  $O$  of  $m$  rectilinear obstacles in the plane, find a set  $S$  of additional points, called Steiner points, such that the length of a rectilinear minimum spanning tree of  $T \cup S$ , which avoids all obstacles in  $O$ , is minimized.

**Definition 2. (FST, compatible and incompatible)** A rectilinear Steiner tree is called a *full Steiner tree* (FST) if every terminal is a leaf of the tree. FST  $f$  is *compatible* with 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 terminals or share partial edges.

Our FORst algorithm is a 3-step heuristic. In *Step1*, we partition all the terminals into some subsets in the presence of obstacles by some rules. After *Step1*, connected components, regarded as hyper-graphs in Section 3, are generated. Then in *Step2*, we connect terminals in each connected graph with one or more trees, respectively. As a result, the separated sub-trees are constructed after this step. In *Step3*, we connect the forest consisting of the trees constructed in *Step2* into a completed Steiner tree spanning all terminals while avoiding all obstacles. Two kinds of algorithms, called ACO-RSMT and GFST-RSMT, are proposed to construct an OARSMT in a

connected graph in *Step2*, which are suitable for different situations. The flow chart and an illustration of FORst are shown in Fig.1 and Fig.2, respectively.

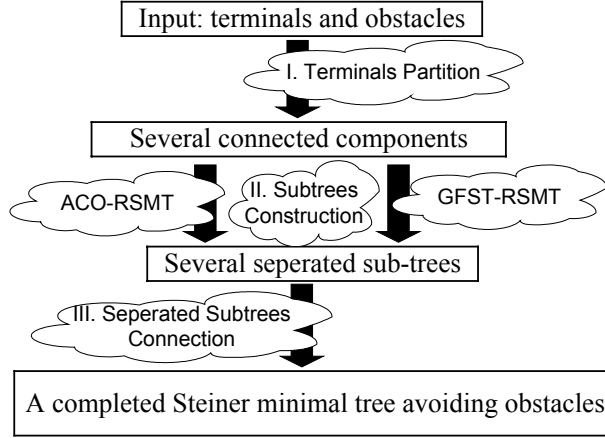


Fig. 1. The flow chart of FORst algorithm.

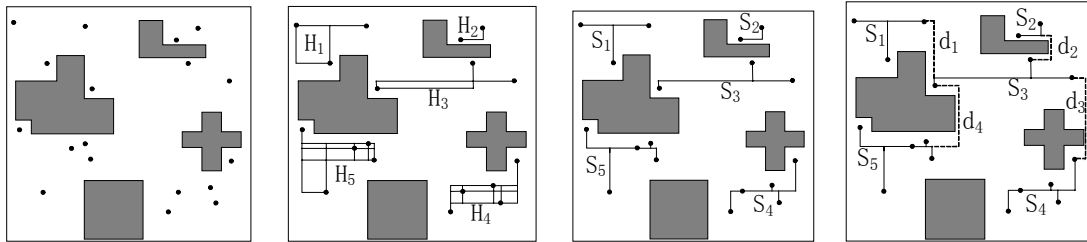


Fig. 2. An illustration of FORst algorithm, (a) the input of the problem: some terminals and four obstacles, (b) the result after *Step1* (terminal partition), (c) the result after *Step2* (sub-tree constructions), (d) the result after *Step3* (separate sub-tree connection).

### 2.1. Step1: Terminal partition

We try to translate OARSMT into RSMT to simplify the problem in this step. We partition the terminals into some subsets and construct a graph for each subset. Here, we construct the FSTs by Hwang’s theorems [8] and the rules of Warme [7]. Then we delete those FSTs that intersect with obstacles. After these operations, there are some FSTs survived, which is called *survival set* denoted by  $F$ .

Consider the *hypergraph*  $H = (T, F)$  with the set of terminals  $T$  as its vertices and the survival set as its hyperedges. Obviously, the hypergraph is partitioned into some connected components, such as  $H_1, H_2, \dots,$  and  $H_n$ , where every pair of vertices are connected by hyperedges in  $H_i$ . After the partition, the set  $T$  and set  $F$  is partitioned into some corresponding subsets, such as  $T_1, \dots, T_n$ , and  $F_1, \dots, F_n$ , where  $T_i$  is the vertex set of  $H_i$  and  $F_i$  is the edge-set of  $H_i$ . The set of hyperedges in  $H_i$ , i.e.  $F_i$ , forms a connection graph  $C_i$ , in which the set of all terminals and Steiner points in all FSTs form the set of vertices and the line segments of the edges. We can see this formation in Fig.2(b), in which the hypergraph  $H$  is partitioned into five connected components,  $H_1, H_2, \dots,$  and  $H_5$ , and the sets  $T$  and  $F$  are partitioned into five subsets correspondingly.

## 2.2. Step2: Sub-tree construction

In this step, the main task is to connect the terminals in each connected components  $H_i$  with one completed Steiner tree or several separated Steiner trees. In Fig.2(c), RSMTs are constructed in each of the connected components, which are  $S_1, S_2, \dots$ , and  $S_5$ , respectively. We now propose two methods to construct an OARSMT in a given connected component, named ACO-RSMT and GFST-RSMT, respectively. Each has its own advantages, which enables us to archive better performances in different situations.

ACO-RSMT is an efficient algorithm based on ant colony optimization [9, 10]. It finds a near-optimal RSMT in the connection graph  $C_i$ , which spans all terminals in  $T_i$ . Because the connection graph  $C_i$  does not intersect with any obstacles, the RSMT is obstacle-avoiding.

GFST (Greedy FST)-RSMT selects compatible FSTs in  $F_i$  greedily, and constructs several separated SMTs. We maintain a triple  $(D, Q, Y)$ , where  $Y$  are the SMTs in the partial solution, and  $D$  are candidate FSTs which are compatible with  $Y$ , and  $Q$  are the terminals that do not covered by  $Y$ . In the initialization,  $Y$  is set to be empty, and  $D$  is set to be  $F_i$ , and  $Q$  is set to be  $T_i$ . Let  $t$  be the current tree.  $\eta$  is the desirability of FST, which is defined as  $\eta(t) = C(t)^\lambda/B(t)^\omega$ , where  $C(t)$  is the wire-length of tree  $t$ ,  $B(t)$  is the number of terminals in tree  $t$ ,  $\lambda$  and  $\omega$  are constants.

ACO-RSMT can produce better solution than GFST-RSMT but in a longer running time. In practice, we can use GFST-RSMT while the terminal number in  $H_i$  is more than 20. Otherwise, we will use ACO-RSMT. The domain complexity of Step2 is  $O(n \log n)$ .

## 2.3. Step3: Separate sub-tree connection

In Step3, we compute the exact shortest path of each pair of nodes in each pair of trees using the *detour method* [11], and then to choose the minimum ones to connect all the Steiner trees. The dashed lines in Fig.2(d) show the results of Step3 for the given instance, and the shortest path we found to connect the trees in Fig.2(c) are  $d_1, d_2, d_3$ , and  $d_4$ , respectively. The time-complexity of Step3 is  $O(n^2 \cdot e \cdot \log(e))$ , where  $e$  is the edge number of obstacles.

## 3. Experimental Results

The FORst algorithm has been implemented in C++ language and performed on a Sun V880 fire workstation. The obstacles are constructed to include all types of convex rectilinear polygons, which includes rectangle, L-shaped polygon, cross-shaped polygon, and more complicated shapes.

Table 1 shows the performance results (percentage improvements over the minimum spanning tree) for our FORst and G4S [3] for randomly generated instances containing 10 rectangular obstacles and the indicated numbers of terminals. Fig.3 shows the result of FORst algorithm to route 1000 terminals in the presence of 100 rectangular obstacles.

## 4. Conclusions

In this paper, we propose a 3-step heuristic for OARSMT construction. The experimental results show that FORst keeps the high performance with a short running time. Two kinds of routing algorithms in Step2 make the FORst work well on the cases with different scales.

Furthermore, the performance of FORst algorithm keeps stable in very large scale cases. The time-complexity of FORst algorithm is  $\max(O(n^3), O(n^2 \cdot e \cdot \log(e)))$ , where  $n$  is the number of terminals and  $e$  is the number of edges of obstacles.

Table 1. The results of FORst algorithm compared with G4S

Term#	G4S	Ours / CPU(s)					
		ACO	CPU	GFST	CPU	FORst	CPU
5	9.2	9.8	0.02	4.9	< 0.01	9.8	0.02
10	9.1	9.1	0.10	5.2	< 0.01	9.1	0.10
15	9.4	9.4	0.25	3.8	< 0.01	9.4	0.25
20	9.0	9.2	0.38	4.2	< 0.01	9.2	0.38
50	-	9.3	3.22	7.4	0.26	9.1	2.75
100	-	8.9	31.2	8.8	3.12	8.8	3.66
500	-	8.7	1244	9.2	482	9.1	559
1000	-	8.2	1343	9.0	1112	8.9	1240

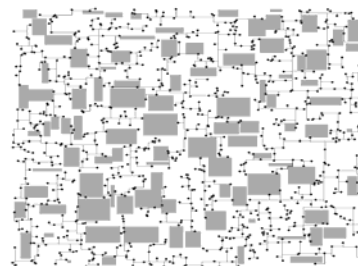


Fig. 3. A RSMT of 1000 terminals avoiding 100 rectangular obstacles.

## References

- [1] M. R. Garey and D. S. Johnson, "The rectilinear Steiner tree problem is NP-complete", *SIAM Journal on Applied Mathematics*, (32) (1977) 826-834.
- [2] F. Rubin, "The Lee Connection Algorithm", *IEEE Trans. on Computer*, (23) (1974) 907-914.
- [3] 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) 113-116.
- [4] 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).
- [5] Z. Zhou, C. D. Jiang, L. S. Huang, *et al*, "Finding Obstacle-Avoiding Shortest Path Using Generalized Connection Graph with  $\Theta(t)$  Edges", *Chinese J. of Software*, 14(2) (2003) 166-174.
- [6] Y. Yang, Q. Zhu, T. Jing, X. L. Hong, *et al*, "Rectilinear Steiner Minimal Tree among Obstacles", In: *Proc. of IEEE ASICON*, Beijing, China, (2003) 348-351.
- [7] D. M. Warme, P. Winter, and M. Zachariasen, "Exact Algorithms for Plane Steiner Tree Problems: A Computational Study", Technical Report DIKU-TR-98/11, Department of Computer Science, University of Copenhagen, (1998).
- [8] F. K. Hwang, D. S. Richards, and P. Winter, "The Steiner Tree Problem, *Annals of Discrete Mathematics*", Amsterdam, The Netherlands: North-Holland, (1992).
- [9] 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*, 26(1) (1996) 1-13.
- [10] S. Das, S. V. Gosavi, W. H. Hsu, *et al*, "An Ant Colony Approach for the Steiner Tree Problem", In: *Proc. of Genetic and Evolutionary Computing Conference*, New York City, New York, (2002).
- [11] S. Q. Zheng, J. S. Lim, and S. S. Iyengar, "Finding obstacle-avoiding shortest paths using implicit connection graphs". *IEEE Trans. on Computer Aided Design*, 15(1) (1996) 103-110.