An Efficient Rectilinear Steiner Minimum Tree Algorithm Based on Ant Colony Optimization*

Yu Hu, Tong Jing, Xianlong Hong, Zhe Feng
Tsinghua University
Beijing 100084, P. R. China
Email: matrix98@mails.tsinghua.edu.cn

Xiaodong Hu, Guiying Yan
Institute of Applied Mathematics, CAS
Beijing 100080, P. R. China
Email: xdhu@public.bta.net.cn

Abstract--**The rectilinear Steiner minimum 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 a practical heuristic for RSMT construction based on ant colony optimization (ACO). This algorithm has been implemented on a Sun workstation with Unix operating system and the results have been compared with the GeoSteiner 3.1 and a recent work using batched greedy triple construction (BGTC). Experimental results show that our algorithm, named ACO-Steiner, can get a very short run time and keep the high performance.**

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

I. INTRODUCTION

Routing plays an important role in very large scale integrated circuit/ultra large scale integrated circuit (VLSI/ULSI) physical design [1]. The rectilinear Steiner minimum tree (RSMT) problem is one of the fundamental problems in routing [2]. However, Garey and Johnson [3] prove that the RSMT problem is NP-complete, indicating 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.

Ref. [4] gave an extensive survey of RSMT heuristics in 1992. Kahng and Robins [5] introduced the Batched Iterated 1-Steiner (BI1S) heuristic with an average improvement over the minimum spanning tree (MST) on terminals of almost 11%. Two good works appeared recently. Kahng and Mandoiu *et al* [6] proposed a batched greedy triple construction (BGTC) algorithm, which speedups the run time of Zelikovsky's algorithm [7] while keeping its performance. Zhou [8] introduced the spanning graph as a base for MST and then constructed the RSMT from the MST. Another work is the $O(n\log n)$ algorithms proposed in [9], which is in octilinear plane instead of rectilinear. Warme *et al* released the GeoSteiner [10, 17], which is an exact algorithm. The shortcoming of GeoSteiner is the long run time. So, There is room for obtaining high efficiency.

The main contribution of this paper is to propose a practical heuristic, called ACO-Steiner, to construct a RSMT, by which we can get a very short run 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) but keeping the fast speed. When the number of terminals is more than 50, ACO-Steiner can achieve near optimal results (within 1% wire length increments compared with GeoSteiner) but keeping the very short run time.

The rest of this paper is organized as follows. In Section II, we introduce the ant colony optimization (ACO) and some basic definitions of RSMT problem. In Section III, the ACO-Steiner heuristic is described in detail. Section IV gives performance improvements based on some special strategies. Then, Section V shows the experiment results. Finally, Section VI concludes the whole paper.

II. PRELIMINARIES

A. ACO algorithm

As we know, ants live in colonies and have evolved to exhibit very complex patterns of social interaction. Besides the simplistic behavior of individual ants, they can communicate with one another through secretions called pheromones, and this cooperative activity of the ants in a nest gives rise to an emergent phenomenon known as swarm intelligence. ACO algorithms are a class of algorithms that mimic the cooperative behavior of real ant behavior to achieve complex computations [11].

The ACO consists of multiple iterations. In the iteration of the algorithm, one or more ants are allowed to execute a move, 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 [12].

The main flow of ACO algorithm is shown in Fig.1.

B. Basic definitions of RSMT problem

The RSMT problem is described as follows [13].

Given a set T of n points called terminals in the plane, find a set S of additional points called Steiner points such that the length of a rectilinear minimum spanning tree of

* This work was partially supported by NSFC under Grant No.60373012, Hi-Tech Research and Development (863) Program of China under Grant 2002AA1Z1460, SRFDP of China under Grant No.20020003008, and NSFC under Grant 60121120706.

$T \cup S$ is minimized.

```
ACO_Algorithm()
1.    Initialization;
1.1       Place all ants in the initial positions;
1.2       Set the intensity of trails as a initial value;
2.    While loopNum < MAXLOOP;
3.        Construct a complete solution by ants moving;
3.1           Select an ant by some rule;
3.2           Make decision based on trail intensity and some
                 greedy rules;
4.        Update the intensity of trails based on the solution;
5.        loopNum ++;
```

Fig.1 The ACO algorithm

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

Hanan [14] has shown that there always exists an RSMT with Steiner points chosen from the intersections (i.e., vertices) of all the horizontal and vertical lines, which is called *Hanan grid*, passing through all the points in terminal set T [15]. So, we can design our ACO-Steiner in the graph based on Hanan grid.

III. Our ACO-Steiner Algorithm

In this section, we use ant colony to construct the RSMT. Firstly, we draw the Hanan graph 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 rule and move to that vertex via an edge in Hanan graph. 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.

An ant determines its next vertex it wants to move stochastically, but 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, and the desirability of vertex j (j must be the neighbor of i in Hanan graph) is defined as

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

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

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

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

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

where $c(S_t)$is the total cost of the current result tree S_t, E_t is 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.

$$p_{i,j} = \begin{cases} \dfrac{[\tau_{i,j}]^\alpha [\eta_j^m]^\beta}{\sum\limits_{k \notin tabu-list(m)} [\tau_{i,k}]^\alpha [\eta_k^m]^\beta}, & j \in A \\ 0, & otherwise \end{cases} \qquad (4)$$

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

The pseudo-code of our ACO-Steiner algorithm is shown is Fig.2.

```
ACO_Steiner_Algorithm()
1. Initialization();
2. While loopNum < MAXLOOP;
3.     Construct_Steiner_Tree_By_ACO();
4.     loopNum ++;
```

Fig.2 ACO-Steiner Algorithm

The MAXLOOP in Fig.2 is the maximum loop number. The **Initialization()** sub-procedure does some initial work for the algorithm and the pseudo-code is shown in Fig3.

```
Initialization()
1. Create Hanan graph G based on T;
2. Set the intensity in each edge in G to be $p_0$;
```

Fig.3 The Initialization() sub-procedure

The sub-procedure **Construct_Steiner_Tree_By_ACO()** is to construct a Steiner tree using ant colony. The pseudo-code is shown in Fig.4.

```
Construct_Steiner_Tree_By_ACO()
1. Place an ant on each vertex in the terminal set T and put
      the vertex into its tabu-list;
2. While ant number > 1
3.     Select an ant m randomly;
4.     AntMove(m);
5.     If m meets m' then
6.         Add vertices in tabu-list of m into that of m';
7.         m died;
8. Update the trail intensity in every edge by equation (2);
```

Fig.4 The Construct_Steiner_Tree_By_ACO() sub-procedure

The sub-procedure **AntMove(*m*)** decides the next vertex that the current ant will move to. The input of this procedure is the ant's current position. The pseudo-code is shown in Fig.5.

```
AntMove(m)
1. Compute the $p_j$ of $m$ by equation (1) and (4);
2. If $p_i == 0$ ($i$ = 1, 2, 3, 4) then
3.     deconfuse();
4. Ant $m$ moves to $j$ under the probability of $p_j$;
5. Add vertex $j$ into $m$'s tabu-list;
```

Fig.5 The AntMove(*m*) sub-procedure

The sub-procedure **deconfuse()** solves the following problem.

Sometimes, an ant may have no available vertices to move (see Fig.6). 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.6. There are two ants (*m* and *m*') left after some 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 the ant *m* can't move any more but to 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 closest to ant *m*'.

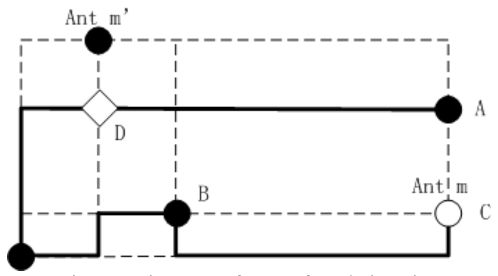

Fig.6 An instance for confused situation

Since ACO-Steiner algorithm is performed based on Hanan graph, we can reduce the Hanan graph so as to reduce the searching space, which bases on the following theorem.

Theorem 1: Let $G'(V', E')$ be a sub-graph of $G(V, E)$, where $T \subset V'$ is the set of terminals. Let $q \in V'-T$ such that the degree of q is equal to two with respect to G, and $v_1, v_2 \in V'$ are its two adjacent vertices. If edges (v_1, q) and (v_2, q) belong to the solution of the Steiner problem in $G(V, E)$, then there exists another solution such that (v_1, q) and (v_2, q) are not in it if a vertex $v_3 \in V'$ exists which is adjacent to both v_1 and v_2.

Theorem 1 has been proved by Yang and Wing in [16]. From theorem 1, we know that any non-terminal vertex that is adjacent to exactly two orthogonal edges *e1* and *e2* can be deleted if the other two edges forming a rectangle with *e1* and *e2* are present. The vertices remaining after this reduction (see Fig.7(b)) has been performed are precisely those that lie within the rectilinear convex hull of the terminals.

In pathological cases the convex-hull reduction may have no effects, but for small, randomly generated sets of terminals it is typically quite effective. The convex-hull reduction often leaves many terminals of degree 1. Such terminals can be deleted (along with their adjacent edge) and their neighbor made a terminal, and the appropriate edge added back into the final solution.

We call this the terminal reduction (see Fig.7(c)). The most striking 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 [13].

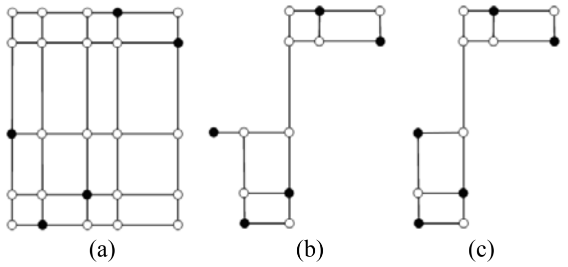

Fig.7 (a) The Hanan graph, (b) The Convex-hull reduction, (c) The terminal reduction

By using this reduction approach, we can speedup our algorithm to some extent. However, the ACO-Steiner algorithm is still much time consuming because the ants must move based on the Hanan grid and move only small segment in the iteration. So, we will shorten the run time to get high efficiency by means of some strategies, which will be introduced in the next section.

IV. Performance Improvements

We extend the tabu-list of each ant to record the edges instead of the vertices that this ant has visited. Every movement does not be 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. If the shortest path is a line, there is only one way to move. However, if the shortest path is a L-shape, there are two possible way to move, which are TOP_ORIENT and BOTTOM_ORIENT. We choose one orient to move based on both the trail intensity and the topology.

Here, the trail will deposit in the four directions of each terminal vertices instead of edges. Thus, we can decide which way to move by comparing the trail intensity in different directions of the current vertex. The updating rule of trail intensity is still based on equation (2). Here, $\tau_{i,j}$ denotes the trail intensity in direction $i \rightarrow j$.

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

Firstly, for a given edge orient we find the closest vertex (out of the current ant's tabu-list) to the edge only in the phrase shown in Fig.8.

Then, compute the distance D_c between this vertex to the edge in this orient. Compute the distance D_f between this vertex to the edge in the opposed orient. The gain in this orient is D_f - D_c. We can see this rule from the following instance shown in Fig.9.

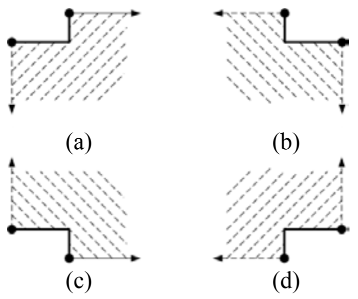

Fig.8 The different phrase in different orient
(a) and (b) are two types of BOTTOM_ORIENT,
(c) and (d) are two types of TOP_ORIENT

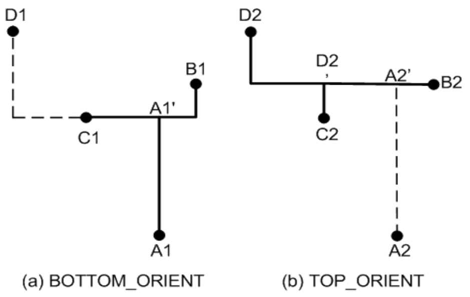

Fig.9 Two orients of edge connect the vertex *B* and *C*

The suffix of the vertex label is just to distinguish the two orients. For example, *A1* and *A2* are the same vertex only in different situation.

In Fig.9, the current position 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 Fig.9(a) and Fig.9(b). The path between *B* and *C* in Fig.9(a) is BOTTOM_ORIENT, and the one in Fig.9(b) is TOP_ORIENT. Now the ant wants to decide the orient of edge *(B, C)*. The closest vertex to edge *(B1, C1)* is *A1* with the distance |*A1A1'*| (see Fig.9(a)) and the distance between *A2* and edge *(B2, C2)* is |*A2A2'*| (see Fig.9(b)). So, the gain in BOTTOM_ORIENT is |*A2'A2*| - |*A1A1'*| and the gain in TOP_ORIENT is |*D1C1*| - |*D2D2'*|.

When we compute the gain of one orient we rewrite equation (1) as following.

$$\eta_d = \frac{[gain_d]^\lambda}{dist_d} \qquad (5)$$

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

Now, an ant can decide the orient with the value given in the formula (4). We can find that it's much greedy in each movement.

We keep the main flow of ACO-Steiner algorithm given in Fig.2 and Fig.4, but improve the following two sub-procedures (shown in Fig.10 and Fig.11) to get a higher performance.

```
Initialization()
1. Set the intensity in each direction of every terminal to be
   $p_0$;
```

Fig.10 The improved Initialization() sub-procedure

```
AntMove(m)
1. Compute the $p_j$ of $m$ by equation (5) and (4);
2. Ant $m$ moves to $j$ based on $p_i$;
3. Add the path from $m$ to $j$ into $m$'s tabu-list;
```

Fig.11 The improved AntMove(*m*) sub-procedure

V. 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 $Q = 10000$ in our experiments.

In the experiment, we find that our improvement method described in Section IV is efficient. An ant decides the next node or the edge orient greedily instead of stochastically with the possibility *p*. By using this "greedy approach", we can maintain a quick convergence to satisfactory solutions while keeping a good performance. Fig.12 shows the performance improvements in a 200 terminal RSMT instance. It shows that the algorithm get the most improvement in the first 10 iterations, and improve little after 50 iterations.

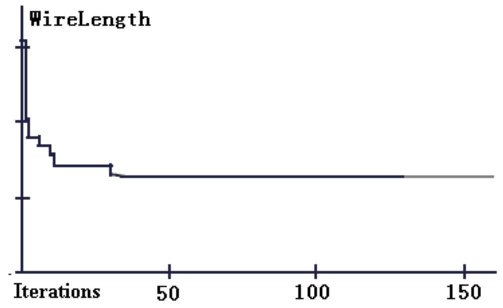

Fig.12 Performance improvement in a 200 terminal RSMT instance

Table 1: Wire length

Terminal Number	BGTC	ACO-Steiner			GeoSteiner
		Best	Ave	Worst	
9	19913	19799	19799	19799	19799
10	21259	21143	21143	21143	21143
20	34767	34767	34778	34878	34767
30	40226	40037	40072	40215	40037
50	51878	51674	51800	52094	51595
70	59564	59531	59701	60093	59503
100	73289	73356	73751	73929	72979
200	104750	105277	105567	105701	104178
500	161875	164440	164484	164565	160844

Table1 shows the experiment results of three algorithms. We set the MAXLOOP = 10.

Table2 shows the runtime of the three algorithms. Each algorithm has performed ten times.

Table 2: Run time

Terminal Number	BGTC	ACO-Steiner	GeoSteiner
9	< 0.001	< 0.001	< 0.001
10	< 0.001	< 0.001	< 0.001
20	< 0.001	< 0.001	< 0.001
30	< 0.001	< 0.001	< 0.001
50	< 0.001	< 0.001	< 0.001
70	< 0.001	0.004	0.020
100	< 0.001	0.018	0.050
200	0.01	0.387	0.880
500	0.050	3.380	38.880

From Table1 and Table2, we can see that ACO-Steiner performs well when the number of the terminal is no more than 50. It can always achieve the optimal results and keep short runtime. When the number of the terminal increases, our algorithm can keep the performance within 1% worse than the optimal (GeoSteiner) with a very short runtime.

The Fig.13 shows the result of ACO-Steiner for a 500 terminals tree.

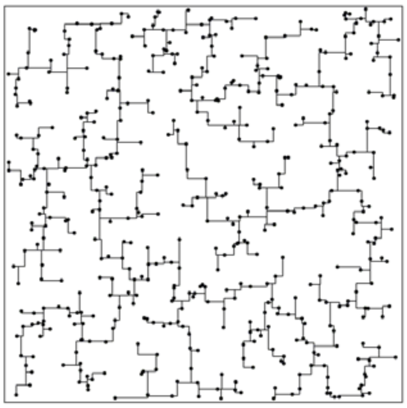

Fig.13 The result of ACO-Steiner for a 500 terminals tree

The Fig.14 shows the result of BGTC vs. ACO-Steiner in a case with 30 terminals. Fig.14(a) shows the result of BGTC with the wire length of 40226 and Fig.14(b) shows the result of ACO-Steiner with the wire length of 40037.

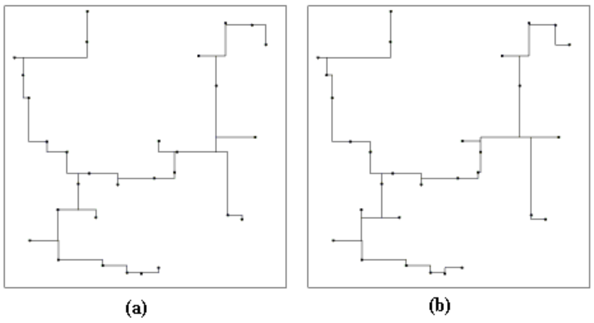

Fig.14 BGTC vs. ACO-Steiner for a 30 terminal tree

VI. Conclusions

In this paper, we propose a practical heuristic for RSMT construction based on ACO. Then, we use a fast-ant strategy to speedup the algorithm. The experimental results show that our heuristic ACO-Steiner keeps the high performance with a very short run time.

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

References

[1] T. Jing, X. L. Hong, Y. C. Cai, H. Y. Bao, J. Y. Xu, "The Key Technologies and Related Research Work of Performance-Driven Global Routing", *J. of Software*, 12(5), pp.677-688, 2001.

[2] T. Jing, X. L. Hong, "The Key Technologies of Performance Optimization for Nanometer Routing", In: *Proc. of IEEE ASICON*, Beijing, China, 2003, pp.118-123.

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

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

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

[6] A. B. Kahng, I. I. Mandoiu, A. Z. Zelikovsky, *et al*, "Highly Scalable Algorithms for Rectilinear and Octilinear Steiner Trees", In: *Proc. of Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2003: pp.827-833.

[7] A. Zelikovsky. An 11/6-approximation for the network Steiner tree prob-lem, *Algorithmica* 9: pp.463-470, 1993.

[8] H. Zhou, "Efficient Steiner Tree Construction Based on Spanning Graphs", *In: Proc. of ACM ISPD,* Monterey, CA, USA, 2003: pp.152-157.

[9] Q. Zhu, H. Zhou, T. Jing, X. L. Hong, and Y. Yang. "Efficient Octilinear Steiner Tree Construction Based on Spanning Graphs", In: *Proc. of IEEE/ACM ASP-DAC, 2004,* Yokohama, Japan, pp.687-690.

[10] 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, April 1998

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

[12] 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.

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

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

[15] J. Griffith, G. Robins, J. S. Salowe, and T. Zhang. "Closing the gap: Near-optimal steiner trees in polynomial time", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 13(11): pp.1351-1365, 1994.

[16] Y. Y. Yang and O. Wing, "Suboptimal Algorithm for a Wire Routing Problem", *IEEE Trans. on Circuit Theory,* September 1972: pp.508-510.

[17] http://www.diku.dk/geosteiner/