- **Algorithm Flow**

The procedure of my program consists of **three** parts, that is, spanning graph construction, minimum spanning tree construction, and rectilinear Steiner tree construction.

1. Spanning graph (SG) construction

Spanning graph is a sparse graph, every vertex has its eight regions (shown in Fig 1) and there is at most one edge in each region. Therefore, there are at most 4n edges in SG, where n is the number of pin vertices.

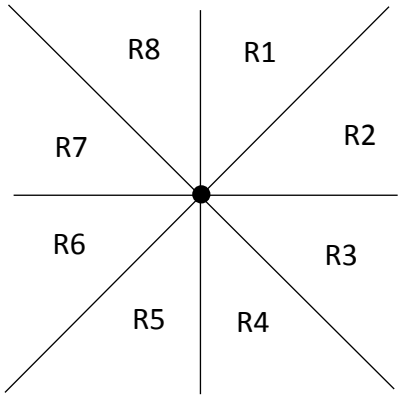


Fig 1

Based on the method in [1], we can construct a spanning graph efficiently by following the two directions, (x + y) and (x - y), and maintaining two active sets (implemented by binary tree). The pseudocode of SG construction is shown in Fig 2. First, sort all the vertices (pins) by (x + y) in non-decreasing order. Then, find the potential neighbors in the two active sets (A[1] and A[2]). The search, deletion, and insertion a vertex to the active sets is quite efficient since these operations on binary tree have time complexity $O(\log_2 n)$ respectively.

```
Algorithm Rectilinear Spanning Graph (RSG)
  for (i = 0; i < 2; i++) {
    if (i == 0) sort points according to x + y;
    else sort points according to x − y;
    A[1] = A[2] = ∅;
    for each point p in the order {
      find points in A[1], A[2] such that p is in their
        R_{2i+1} and R_{2i+2} regions, respectively;
      connect p with points in each subset;
      delete the subsets from A[1], A[2], respectively;
      add p to A[1], A[2];
    }
  }
```

Fig 2

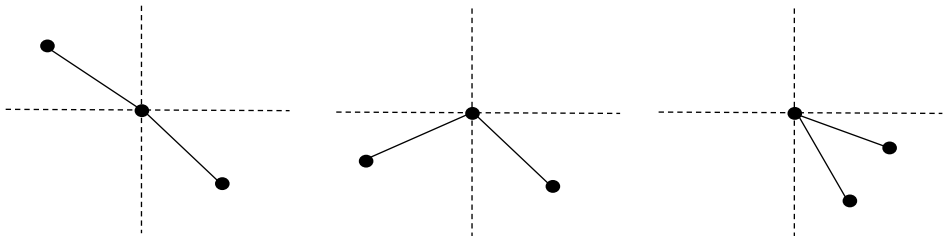
2. Minimum spanning tree (MST) construction

After the SG construction, build a MST from the SG. To find a MST, I apply **Kruskal's algorithm**. The algorithm is shown in Fig 3.

```
KRUSKAL (G):
1 A = ∅
2 for each v ∈ G.V:
3       MAKE-SET (v)
4 for each (u, v) in G.E ordered by weight (u, v), increasing:
5       if FIND-SET(u) ≠ FIND-SET(v):
6             A = A ∪ {(u, v)}
7             UNION (u, v)
8 return A
```

3. Rectilinear Steiner tree (RST) construction

After the construction of MST, we transform the slant edges into either vertical lines or horizontal lines. Referred to the method used in obstacle-avoiding rectilinear Steiner tree (OARST) in [2], there are three cases between two neighbor edges, **opposite regions**, **near regions**, and **the same region** (Fig 4). Notice that the definition of region is different from the one which we discussed in SG construction. Fig 5 is the definition of region in this step.



Case 1: Opposite regions | Case 2: Near regions | Case 3: The same regions

Fig 4

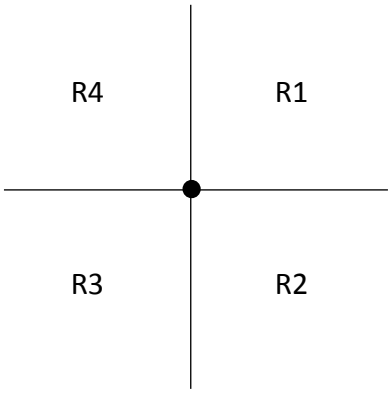


Fig 5

First, sort all the edges by Manhattan distance in non-decreasing order, then select the longest edge **e** and its longest neighbor edge **ne** which has not been done. Transform the two selected edges by the three transformation shown in Fig 6.

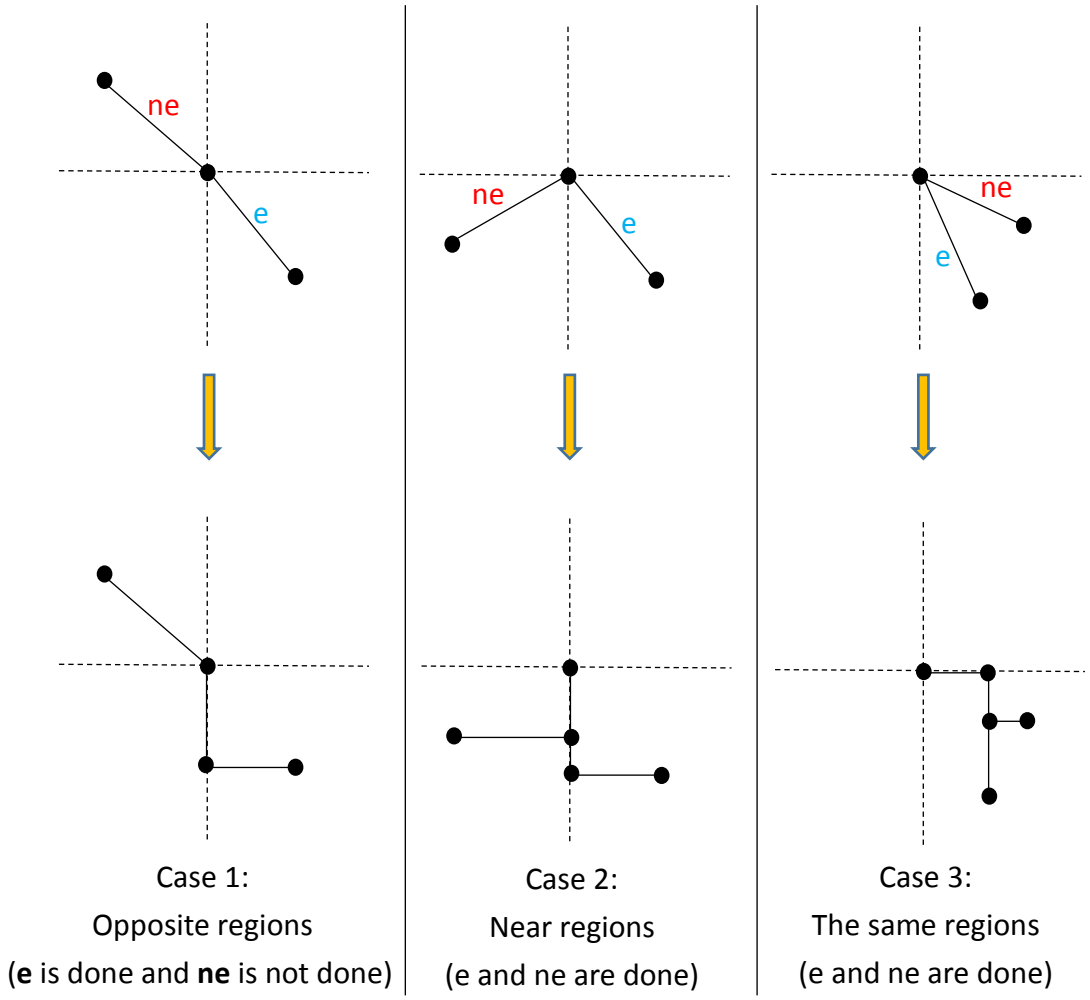
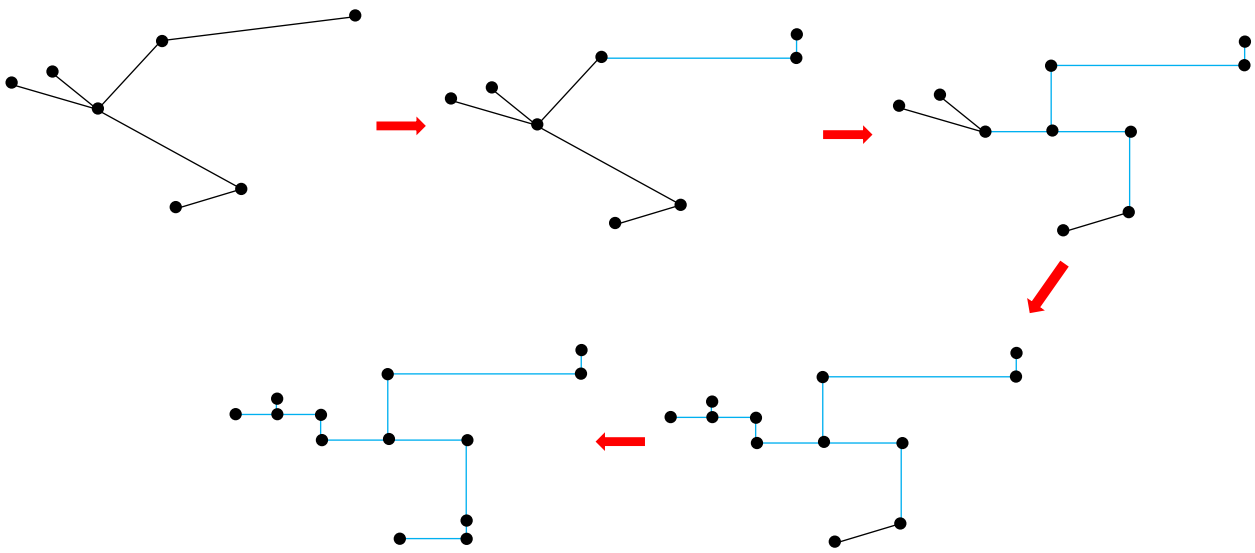


Fig 6

Example:



- **Data Structure**

  The data structure in this project is basic and simple. First of all, since the main steps are all operating on the spanning graph, I maintain *class Edge* and *class Pin*. *Class Edge* stores its two terminals and its neighbor edges when it is generated. In addition, *Class Pin* stores its coordinates and also the edges which is connected to it.

  The active sets implemented in SG construction is binary trees (STL Map). Binary trees can efficiently delete, insert, and search a node, which reduces the time complexity.

- **Problem & Discussion**

  Q1: It is known that spanning graph has at most 4n edges. In practice, what is the average number of edges in a spanning tree? (n: number of pins)

  A: In the experiments, the average number of edges in a spanning trees is **2n ~ 2.5n**.

  Q2: It is proved that the ratio of the total length of a MST and that of a rectilinear minimal Steiner tree is at most 1.5 [3]. What is the improvement (in percentage) from a MST to a RSMT in this program?

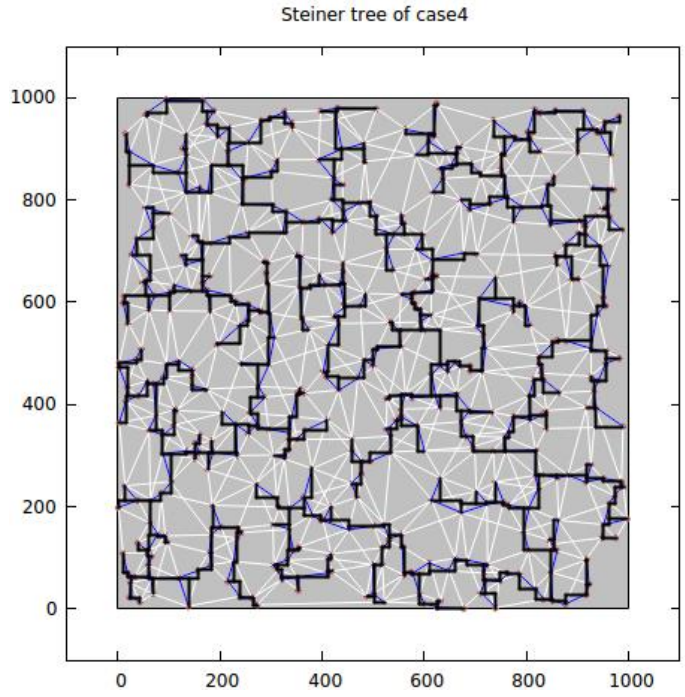
  A: From the results, the improvement is, in average, **8.5% ~ 9.5%**.

- **Supplement**

  This program provide two **Gnuplot** output formats. (See readme.txt)
  Here shows the example plot.

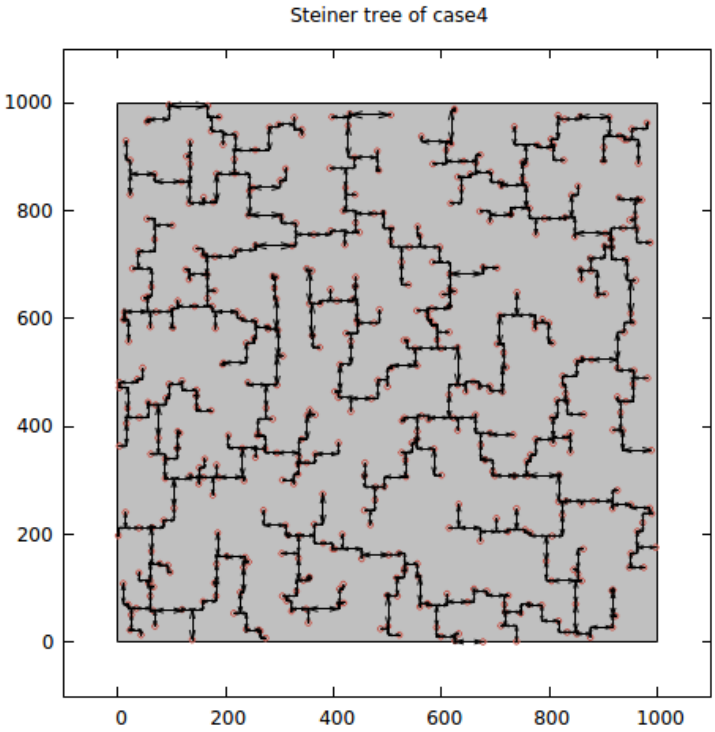
  1. ***.graph.plt**:

     This plot shows the spanning graph (**white** lines), minimum spanning tree (**blue** lines), and rectilinear Steiner tree (**black** lines).



2. ***.tree.plt**:

This plot shows the rectilinear Steiner tree (two-head arrows).



Reference

[1] H. Zhou, N. Shenoy, and W. Nicholls. Efficient spanning tree construction without delaney triangulation. Information Processing Letter, 81(5), 2002.

[2] C.-W. Lin, S.-Y. Chen, C.-F. Li, Y.-W. Chang, and C.-L. Yang, "Obstacleavoiding rectilinear Steiner tree construction based on spanning graphs," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 27, no. 4, pp. 643–653, Apr. 2008..

[3] F.K. Hwang, "On Steiner minimal trees with rectilinear distance", SIAM J. Appl. Math. 30 (1976) 104-114.