Algorithm design and simulation of Rectilinear Steiner Tree problems based on concrete class Line

Man feng[1], Mingyang Li[2], Pengyuan Chen[3], Zhenping Lan[4],Ping Li(✉)
{ fengman2020@163.com[1],523155344@qq.com[2],554855487@qq.com[3],lanzp@dlpu.edu.cn[4],liping@dlpu.edu.cn (✉) }

Department of Information Science and Engineering Dalian Polytechnic University Dalian, P. R. China

Abstract. This graduation thesis mainly focuses on the specific RST, namely the Minimum Rectilinear Steiner tree, to study and improve the algorithm, and finally simulation and output results. Steiner tree problem belongs to combinatorial optimization discipline and is NP-hard problem. Over the past three centuries, many mathematicians have extended and extended this problem, making the steiner tree tree problem further developed. This paper mainly deals with a sub-problem of steiner tree, the Minimum Rectilinear Steiner tree problem. In this paper, based on dynamic programming, a classical technique to solve the steiner tree problem, two algorithms, L-MRST and Z-MRST, are proposed to construct the Minimum Rectilinear Steiner tree on a plane. Both algorithms use point data discretization for pre-tree preparation. However, in the same solution mode, the algorithm l-mrst directly iterates on the current subtree to build the final tree. The algorithm zmrst preprocesses the feasible z-layout standby of all points and enumerates all possible layouts when traversing the nodes. This algorithm increases the algorithm complexity, but the results are obviously optimized. Experimental results also bear this out. For the Minimum Rectilinear Steiner tree problem, the two algorithms in this paper have their own advantages and disadvantages, which can help VLSI problem to some extent, and can effectively optimize the large-scale wire network routing problem. The next research direction about the line steiner tree is also proposed.

Keywords: Minimum Rectilinear Steiner tree, Dynamic programming, VLSI, Algorithm Optimization.

1 Overview

1.1 A Sample Introduction

The Steiner tree (ST) problem is a combinatorial optimisation problem. It first sprouted in the early 18th century and appeared in the Fermat problem, which was popularised in the 19th century by the Swiss mathematician Steiner as "the problem of finding a point on a plane such that the sum of the distances from this point to a given number of points on the plane is minimised" [1]. Steiner had no further contribution to make to the Steiner Tree problem, but in his honour the problem was named after him.

✉ Corresponding author: Ping Li
(1969-),female, Professor, master, main research directions for optical communications and intelligence systems. E-mail:liping@dlpu.edu.cn

EAI GreeNets 2021, June 06-07, Dalian, People's Republic of China
Copyright © 2021 EAI
DOI 10.4108/eai.6-6-2021.2307527

Over the centuries, several mathematicians have extended the problem, leading to a deeper development of the Steiner tree problem and a revision of the term "introduction of a point" to "introduction of several points". This extends the scope of its application and significantly increases the difficulty of the problem. The Steiner tree problem has been identified as an NP difficult problem [11]. Initially, researchers were only able to propose an approximation scheme (Polynomial-time Approximation Scheme, PTAS) for the Steiner tree problem, using heuristic algorithms such as the Genetic Algorithm or ant algorithm to find an approximation in the worst-case scenario. The solution is compared with the minimum spanning tree as a measure. It is only in recent times that exact algorithms for the Steiner tree problem have been developed. The first exact algorithms were proposed by Hakimi, a foreign researcher, and were based on an enumerative implementation, but with considerable limitations and high time complexity. Subsequently, new exact algorithms have been proposed and the Steiner tree problem has evolved by expanding its definition, enriching its structure and branching out. The Steiner tree problem has also been shown to be NPcomplete [1].

In recent years, the Steiner tree problem has been defined in detail as a graph consisting of n points on a plane, connecting these points and allowing the addition of up to N-2 points (the additional points are called Steiner points), with the shortest connected lines forming a tree called a Steiner tree. If no new points are added, the Steiner tree problem is equivalent to the classical computational minimum spanning tree problem. For example, it can become an Edge-Weighted Steiner tree (EWST) depending on whether the edges have value or not; similarly, it can become a VertexWeighted Steiner tree (Vertex-Weighted) depending on whether the vertices are assigned value or not. (Steiner tree,VWST). In general, the default weighted Steiner tree problem is the edge-weighted Steiner tree (EWST).

In addition, the Steiner tree problem is subdivided into sub-problems such as directed Steiner trees, undirected Steiner trees, special Steiner trees, bottleneck Steiner trees and parametric Steiner trees. In the case of the European Steiner Tree, which is a Steiner tree on the European plane, the two lines connecting the same point must be at an obtuse angle of 120°, whereas in the case of the Rectirectilinear Steiner Tree (RST), the two connecting lines through the same point must be at 90°, and are therefore also known as right-angled Steiner trees [4].

This thesis studies a sub-case of the rectilinear staner tree (RST) - the minimum rectilinear staner tree!

1.2 Concept and nature of the Steiner tree problem

The concept of a Steiner tree has been defined by many researchers at home and abroad and is basically the same. The shortest tree that connects all these terminal vertices is a Steiner tree (V is the set of vertices and E is the set of edges). The maximum number of related edges over a vertex is three (set). In addition to terminal vertices, there are also Steiner points in V. Steiner points are new points that are added during the connection process in order to have shorter connecting edges and are also known as auxiliary vertices. The Steiner point is the most fundamental feature of the Steiner tree. This is because if there were no Steiner points, i.e. V=R, the Steiner tree would be identical to the minimum spanning tree. There are no more than n-2 Steiner points in a Steiner tree [2].

The Steiner tree basically follows the above definition and is divided according to different requirements.

The Euclidean steiner tree (EST) is a steiner tree on the Euclidean plane, which requires an obtuse angle of 120° on both sides past a point and all steiner points must be within a convex bundle if the area enclosed by the connection of n terminal vertices is a convex bundle.

The line distance between two points a (x_1,y_1) and b (x_2,y_2) is calculated using the European distance d.

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

However, in the case of the rectilinear staner tree (RST), it is required that the two sides past a point must be at 90° and that all lines are perpendicular, which is why it is also called the right angle staner tree or vertical static distance staner tree. The connection distance d_1 between two points a1 (x_1,y_1) and b1 (x_2,y_2) is calculated using the absolute distance (also known as the Manhattan distance, where two points can only be connected by a straight or polyline segment).

$$d_1 = |x_1 - x_2| + |y_1 - y_2|$$

I have used figures 1 and 2 to show the difference between a European Steiner tree and a straight Steiner tree.

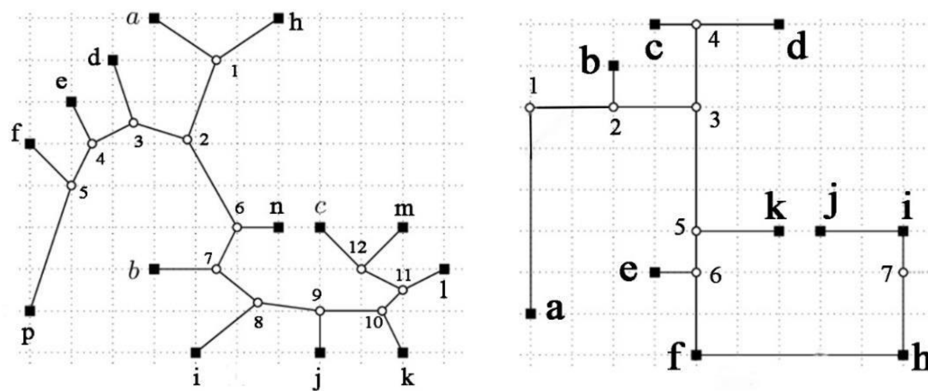

Fig. 1. Euclidean steiner tree and Rectilinear Steiner tree. The solid square dot marked by a letter represents the terminal vertex R; the hollow dot marked by a number represents the Steiner point S. The hollow dot is a hollow dot.

This thesis investigates a sub-case of the rectilinear Steiner tree (RST) - the minimum rectilinear Steiner tree (MRST). As the name implies, the minimum recti-linear Steiner tree is the Steiner tree with the shortest total length and the lowest total cost of connection among many rectilinear Steiner trees.

1.3 Key point analysis of the least rectilinear Steiner tree

The key to solving general Steiner trees is the determination of the Steiner point (S). The rectilinear Steiner tree is no exception, and even the Steiner point of a rectilinear Steiner tree can be better determined. A new concept, the Hanan grid, needs to be introduced here. At each point in a given set of terminal vertices R, a horizontal line and a vertical line lead to a point where the intersection is a Hanan point, and the resulting mesh is a Hanan grid. The mathematician Hanan, who first formulated the minimal rectilinear Steiner tree problem, has a famous Hanan's theorem. Hanan's theorem means that for any rectilinear Steiner tree, when the Steiner points are all located at Hanan's points, it is certain that there must be an optimal rectilinear Steiner tree, i.e. a minimum rectilinear Steiner tree [2].

Thus, the problem of finding a minimum rectilinear Steiner tree on a plane can be completed equivalent to finding a minimum rectilinear Steiner tree on a Hanan grid. The positions of the Steiner points are constrained. The key to the minimum Steiner tree becomes the filtering of

the Hanan grid points to determine the Steiner point positions and finally the rectilinear tree construction to solve the optimal tree [10].

1.4 Various known algorithms and their performance measures

First of all, it is clear that the Steiner tree problem is an NP Hard problem (nondeterministic polynomial (NP), so there may or may not be an infinite number of solutions. Of course, several researchers have proved that the Steiner tree problem is solvable, so there is only an optimal solution and not a single final solution to the problem. This is because as the mathematical body of knowledge and computer technology continues to develop, researchers continue to discover new optimisation algorithms and solutions, and the problem takes on a new lease of life. This is the beauty of the NP problem.

The minimum rectilinear Steiner problem discussed in this thesis and the proposed optimisation algorithm is only one of many methods, and the results are not necessarily optimal, but only partially based on an effective algorithm.

Approximate and exact algorithms. When the rectilinear Steiner tree problem was first studied, researchers were not able to come up with an exact algorithm, so research has focused on the Polynomial-time Approximation Scheme (PTAS) for polynomials. The performance measure of an approximation algorithm is the algorithm approximation, which is the ratio of the approximate solution found in the least ideal case to the optimal solution. The performance of the Steiner tree algorithm in comparison to the minimum spanning tree algorithm is also one of the key research questions. Berman et al. found that

$W_{ST}/W_m \leq 2(1 - 1/m)$

W_{ST} : Stannard tree weights W_m: the minimum spanning tree weight m: the minimum number of leaf nodes in the spanning tree

From 1980 to the 21st century, the approximation algorithm has been improved by many authors, resulting in a reduction in approximation from 2 to 1.55 and a reduction in time complexity from O(nlogn+m) to n^k (k approaching infinity)[2].However, approximate algorithms can only give approximate solutions and many problems do not have approximate algorithms, so researchers have started to investigate exact algorithms. The earliest exact algorithms were based on enumerative implementations, either based on spanning tree enumeration or on topology enumeration, and I have collated the exact algorithms proposed by some of the researchers in Table 1.

As can be seen from the table, branch definition algorithms have become the key to solving the problem, but the various algorithms have their limitations and deficiencies, are highly complex and are not suitable for situations where the number of vertices is large. Therefore research on the Steiner tree problem is still ongoing.

Table 1. Comparison of the exact Steiner tree algorithm.

Author	Methods	Time complexity	Limitations or deficiencies
Hakimi	Enumeration	$O(n^2 logn+mn+min\{n^{k-2},2^{n-k}\}k^2)$	Only applicable to R=V
Shore	Branch definition	$O(24^n)$	High time complexity
Aneja,Maculan	Branch definition	$O(1.62^n)$	No time complications
Fomin	Treatment technology	$O(1.59^n)$	Clusters phenomenon

Parameterisation algorithms. Because of the large number of terminal vertices n and the complexity of the diagram in practical use, parametric algorithms have been developed.

Parameterisation algorithms are used to parameterise the Steiner tree problem. The classical parametric algorithm is the Dreyfus-Wagner algorithm, which is based on dynamic planning techniques. The implementation is based on the idea of decomposition, with three points as a small tree, from which a subset of the small trees is ultimately connected to form the entire large optimal Steiner tree, iteratively for each connected subtree. The complexity of the algorithm is O ($^{3rn+2rn2}$). The dynamic planning technique is also one of the most widely used techniques for solving Steiner tree problems. Later researchers applied tree decomposition, also known as path decomposition, to the investigation of the Steiner tree problem, which led to further optimisation of the Steiner tree parametric algorithm problem. Subset convolution and dynamic planning and capacitive theorem and dynamic planning techniques were later applied to optimise both spatial and temporal complexity, and the Steiner tree problem continues to progress.

Heuristic algorithms. A Heuristic Algorithm is an intelligent optimisation algorithm. Heuristic algorithms are usually found in the traditional problem solving experience of seeking a problem oriented strategy, which can then be used to find a relatively good solution in a feasible time frame. At this stage, heuristic algorithms are mainly nature-like algorithms, such as ant colony algorithms, simulated annealing algorithms, genetic algorithms and so on[5].

Simulated annealing algorithms refer to the problem of similar and combinatorial optimisation of the annealing process of crystalline material substances. If the annealing process is modelled, the objective function agrees with the energy function and the control parameter agrees with the important parameter temperature, then the solution space is regarded as the state space and the final simulation of the annealing algorithm's search for the base state (the lowest energy state in which a solid substance is formed) is the optimisation of the minimal (optimal) value of the objective function.

A colony algorithm is the ability of ants to release a pheromone on a path when searching for food. The more ants that travel a certain path, the more pheromone trails are left behind and the higher the probability that the ants will later choose that path. By means of this internal synergy, the ant colony gradually develops an optimum route. This optimal route is the optimal solution.

Genetic algorithms are the most basic type of intelligent optimisation algorithm. It is based on the Darwinian theory of evolution, which is based on the principle of survival of the fittest. "Genetic algorithms simulate the phenomena of mating, reproduction and mutation that occur in the process of natural selection and, according to the principle of survival of the fittest, use genetic operators (there are different genetic operators for different problems, i.e. variable parameters) to carry out calculations and select the best individual from generation to generation. Individual"[6], the optimal solution to be found.

Particle swarm optimisation algorithms are derived from genetic algorithms. It is an intelligent optimisation algorithm based on iterative thinking and can be used to solve most optimisation problems. Particle swarm optimisation is an improvement on the legacy algorithm, which is simpler and less complex than the genetic algorithm.

There are many other effective algorithms for solving the minimum rectilinear Steiner tree problem, but only some of them are listed here as a reference for optimising the algorithms.

2 Algorithm design

2.1 Algorithm design principles and processes

Principle: The algorithm is based on dynamic planning techniques. Dynamic planning is one of the most widely used techniques for solving Steiner tree problems. The algorithm uses decomposition thinking to find the optimal rectilinear Steiner tree for each subset, taking into account all decomposition cases[8].

Dynamic programming is a commonly used mathematical method, developed in the 20th century by mathematicians such as Bellman, which involves taking a large multistage problem and transforming it into a series of small single-stage problems, solving them one by one using the relationships between the stages, and ultimately optimising the solution.

The basic idea of a dynamic planning algorithm is to decompose the problem to be solved into a number of sub-problems, solve the sub-problems first and then derive the solution to the original problem from the solutions to these sub-problems. However, the sub-problems resulting from the decomposition are not normally independent of each other. Therefore, using dynamic planning techniques, it is possible to create an array of variables that hold the answers to the solved sub-problems (e.g. the edges of the spanning tree in this article) and then find the answers to the solved subproblems when needed, i.e. when performing the next recursion, thus avoiding the increased complexity and time costs associated with a large number of repeated calculations.

This is how the algorithm in this thesis works, first finding the root node, then recursively finding all the solutions, finding the optimal solution, moving on to the child nodes and continuing the recursion, but this is not a simple repetition, as each recursion is related to the solution already found, and does not generate a separate subtree, but ultimately a complete tree. Therefore, by using the 'table' idea of dynamic planning techniques, the complexity of the algorithm can be reduced, the algorithm can be run faster and redundancy reduced.

2.2 Design and research of algorithms

The algorithm uses the Strategy design pattern and the display specifies a class, named RST, which can use the changeStrategy function to change the solution mode used (L-MRST or Z-MRST), thus enabling both algorithms to run in the same situation. The ZRST class and the LRST class are used to solve these two different algorithms. At the same time the ZRST and LRST implementations have different interfaces, so the Strategy class also acts as an Adapter.

The RST class is a common solution mode for solving rectilinear Steiner trees. The programme starts with a randomly generated number of terminal vertices in the main function, which are added to the RST class and changed by the changeStrategy function into the L-MRST or Z-MRST solution mode, respectively. Some of the pseudocode is shown below.

The changeStrategy function is first defined to be a member of the RST class and, after a simple if statement and switch selection statement, ultimately returns myStrategyIdx_.

```
1 int RST ::changeStrategy(int s_idx) //define the change function
2 {
3       if (s_idx == myStrategyIdx_) return 0; //judgement
4       if (myStrategyIdx_)
5             delete myStrategy_;
6       myStrategyIdx_ = s_idx; //assign a value to the parameter s_idx
7       switch (s_idx)
8       case 1 :
9             myStrategy_ = new LRSTStrategy;
10                  break;
11          case 2 :
12                myStrategy_ = new ZRSTStrategy;
13                  break;
14      return myStrategyIdx_; //return tree
15 }
```

A new class is then defined, named the MST class, which is used to solve the rectilinear minimum spanning tree in preparation for the subsequent algorithm, which is used to solve for all the edges of the tree after generating the minimum rectilinear Steiner tree (as the minimum spanning tree was originally used when solving the minimum Steiner tree, and although the two trees are different, they have something in common).

The following is part of the code for the MST function. The first step is to traverse all nodes and find all edges using a two-layer for loop.

```
1       void MST::mst() {
2       const size_t n = m_vertexs.size();
3       Dist dist dist_max(std::numeric_limits<int>::max(), 0, 0);
4       vector<Dist> dist(n, dist_max);
5       vector<size_t> parent(n, 0);
6       vector<bool> in_tree(n, false);
7       size_t t = 0; // dist[t] = Dist(m_vertexs[t], m_vertexs[t]);
8       for (size_t k = 0; k < n - 1; k++) {
9             in_tree[t] = true;
10            for (size_t j = 0; j < n; j++)
11                  if (j ! = t) {
12                        Dist tmp(m_vertexs[t], m_vertexs[j]);
13                        if (tmp < dist[j] && !in_tree[j]) {
14                              dist[j] = tmp;
15                              parent[j] = t; // exchange nodes
16                        } }
17            Dist min = dist_max;
18            size_t min_v = 0;
19            for (size_t i = 0; i < n; i++) {
20                  if (dist[i] < min && !in_tree[i]) {
21                        min = dist[i];
22                        min_v = i;
23                  } }
24            t = min_v;
25            m_lines.push_back(Line(parent[t], t));
26      } }
```

A number of constants used in the algorithm are then defined in the Common function, including NUM, i.e. the number of vertices for a given endpoint. By changing the value of NUM in Common.h, a minimum rectilinear Steiner tree can be generated with the corresponding number of vertices.

Finally, the Visualizer class is also specified, which plays a role in the display. The pseudo-code is displayed as follows.

All generated edges are first processed with a for loop, stored and coloured black, and then all points, including terminal vertices and steiner points, are processed with a for loop, also coloured black, for display in the final generated tree.

```
1 void Visualizer::show(const RST* rst)
2 {
3   Mat picture(name);
4   for (const auto& seg : rst   ->v_seg)//processing the generated edge
5   {
6       line(picture,
7                   horizontal axis coordinates, vertical axis coordinates),
8          cv::Point(seg.v.x, seg.v.y),
9          //store as black
10    }
11   for (const auto& p : rst   ->v_op) // Process NUM points
12   {
13       circle(picture,
14            cv::            Point(p.x, p.y),
15                       cv::Scalar(0, 0, 0));
16    }
17 namedWindow("RST");//Naming
18 imshow("RST", picture);//Show
19 }
```

2.3 Implementation and comparison of the two algorithms

The following discussion focuses on the implementation of the LRST and ZRST algorithms. Both algorithms rely on the base class Line and the derived class Line_L. Line_L and Line_Z are L- and Z-shaped Layout. segment is used mainly in the peripheral RST and Visualizer classes to represent each horizontal or vertical line segment (where the polyline has been split). This differs when implementing the construction of a rectilinear Steiner tree in concrete terms.

L-MRST based on the derived class Line_L. LRST is to select an "L-shaped" or "U-shaped" layout for each edge and, out of all the layouts, find the one with the largest overlap. Due to the separable nature, the sub-problems can in fact be independent and the exhaustive search can be translated into a dynamic planning process in the tree.The tree therefore first needs to be created. A node of degree 1 needs to be chosen as the root (this is to ensure that there is initially only one edge). The tree is then built recursively using the list of edges found in the MST function. The tree is represented by two arrays: tree and parent. each element of the tree is a vector, which represents the successor node of a node. parent represents the parent node.Since any resulting edge is only possible on a grid line with dotted x- or y-coordinates (Hanan grid), it is advantageous to discretize the coordinates for subsequent processing. All that is required for discretization is to obtain the coordinates of all points, then sort and de-

weight all x,y coordinates, the resulting ordinate number being the discretized coordinate number. This is done using the map function in STL (the internal implementation of map is an ordered red and black tree). x_coord and y_coord variables record the original coordinates of all the discretized coordinates and discr_points records the discretized coordinates of all the points, respectively.The functions find_layout_L and find_layout_U recursively solve for all cases. Starting from the only child node of the root, each call solves for the optimal solution (the solution is the choice that the children of the current node should adopt) in the case where the edge between the current point and the parent node is chosen L (or U) in the current subtree. The result of the solution for each choice at each point is placed in the layout_l and layout_u arrays, using binary coding, for each bit of code for the choice of a child node (0 for L, 1 for U).An exhaustive search of all possible case points for all children of a node is done recursively using the dfs function. Part of the pseudo-code is shown below.

```
1 dfs(int parent, std::vector<int> &kids,
2           size_t num,
3           int &value, int &result, int choice, int &decision) {
4       if (num == kids.size()) {
5            if (result < value)
6                 { result = value; decision = choice;}
7        // if the shape is L
8        paint(discr_points[parent], discr_points[kids[num]], false, 1, value);
9        value += layout_l[kids[num]];
10         dfs(parent, kids, num + 1, value, result, choice, decision);
11         value -= layout_l[kids[num]];
12         paint(discr_points[parent], discr_points[kids[num]], false, -1, value);
13 // if the shape is U
14       Repeat,false replase true.
```

Z-MRST based on the derived class Line_Z. The first step of the second algorithm requires the discretization of the terminal vertices as well as tree building. These two processes are the same as for L-MRST.

The difference is that for each edge, all possible Z-layouts are pre-processed first. This is achieved with the dfs function.

After the tree has been built with the build_tree function, it is called recursively from the root (with the find_root function) (here with the find_layout function). For each node that is traversed, all of its children are traversed and then all possible layouts for the current point are enumerated (i.e. all combinations of layouts for the children of the current node). The other processes and coordinate points are stored in the same way as the L-MRST algorithm.

The optimal layout for the current point is obtained by the get_result function, which is iterated over and over again to obtain the minimum rectilinear Steiner tree.

```
1 void ZRST ::dfs(int root, int father, size_t stat, layout &lay, vector<size_t>& stack) {
2 if (tree[root].size() == 0)
3 lay.sub_ans = 0;//initialisation
4       int id = stat, son;
5     if (stat == tree[root].size()) {// Traverse from the root node
6 int ans = 0;
7 vector<Line_Z> overLap;
8 for (int i = 0; i < id; i++) {
9          ans += layouts[son = tree[root][i]][stack[i]].sub_ans;
          overLap.push_back(Line_Z(root, layouts[son][stack[i]].mid_point));
10          overLap.push_back(Line_Z(father, root, lay.mid_point));
11          ans += overlap(points(), overLap)
12 point(root).distance(point(father));
13     if (sans < lay.sub_ans) {
14          lay.sub_ans = ans;
15          copy(stack.begin(), stack.begin() + id, lay.best_lay);}
 else {son = tree[root][stat];
16          for (size_t i = 0; (stack[id] = i) < layouts[son].size(); i++) {
17                  dfs(root, father, stat + 1, lay, stack);//continue iterating
18 }
```

2.4 Algorithmic simulation implementation

This algorithm uses command line parameters to control the behaviour of the programme. "LRST" and "ZRST" run the two algorithms respectively. "test.txt" is used to generate the new test data. This means that the input values are the same (same randomly generated number of end vertices), the output form is the same and the spanning tree attribution class is the same, the only difference is the tree generation process.

Output form: Each run displays the resulting graph of the solution and the final total tree length using opencv.

The number of terminal vertices and the constants of the file names can be changed in the code main.cpp and common.h.

3 Analysis of simulation results

3.1 Experimental platforms and use cases

The algorithm experiments are carried out on the same experimental platform in both a hardware and a software environment.

(1) Hardware environment

Mainframe: Intel® Core™ i7-7500U CPU, 8GB RAM

(2) Software environment

Host: Microsoft Windows 10 (64 bit edition), Microsoft Visual Studio 2019, OpenCv.

(3) Software used

Microsoft Visual Studio 2019, OpenCv

The simulation of the two algorithms with different numbers of vertices is based on the same randomly generated Hanan grid structure, where the horizontal and vertical distances of

the lattice points are considered to be of unit length 1. Six test cases with 5, 10, 20, 20, 100 and 500 vertices are selected for the experiment. The resulting graph of the minimum rectilinear Steiner tree is recorded and represented as opencv, the total tree length for each case is output for both algorithms and a txt document is generated to record the coordinates of each point.

3.2 Experimental results

Six groups of experimental cases with a number of terminal vertices of 5, 10, 20, 20, 100 and 500 were selected for the experiment. Below is the total tree length generated for each group in both algorithms.

Table 2. Simulation results.

Case studies	Number of terminal vertices	L-MRST total length	Z-MRST total length
1	5	18	18
2	10	65	65
3	20	159	159
4	50	168	168
5	100	381	379
6	500	8480	8432

The graphical results on the final spanning tree are represented by cases 3, 4 and 6.

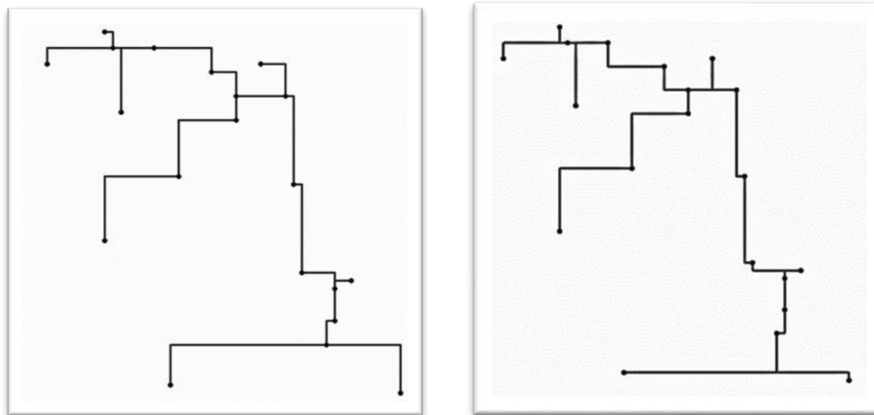

Fig. 2. Case 3.L-MRST & Z-MRST

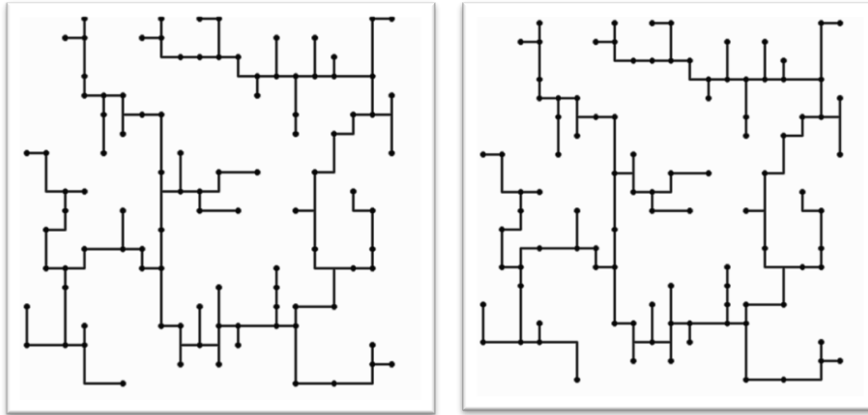

Fig. 3. Case 4.L-MRST & Z-MRST

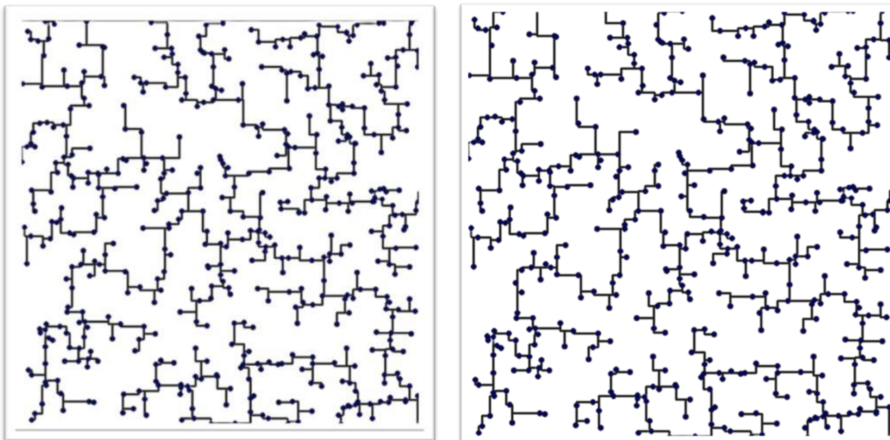

Fig. 4. Case 6.L-MRST & Z-MRST

3.3 Comparative analysis of simulation results

Simulation results: For both the L-MRST and Z-MRST algorithms, the total length of the rectilinear Steiner spanning tree is practically indistinguishable for a small number of vertices (n) (less than 50) and there is a slight difference in the graph of the spanning tree. With a larger number of vertices, Z-MRST has a shorter total rectilinear Steiner spanning tree length and produces better results than the L-MRST algorithm. This advantage is even more pronounced when the number of vertices is higher. Analysis of the results: It can be seen that Z-MRST gives slightly better calculation results than L-MRST. This is due to the fact that Z-MRST takes more cases into account and traverses more leaf node layouts. z-MRST pre-processes all the nodes with possible z-layout spares, because a z-layout can be uniquely identified by its inflection point and so a layout can be represented by saving the coordinates of this inflection point. for each node traversed. All of its children are traversed and then all possible layouts for the current point are enumerated (i.e. all combinations of layouts for the children of the current node). This results in a more optimised rectilinear Steiner tree, but the complexity of the algorithm increases accordingly and it takes longer to get the results from

the running program.

In summary.

When the number of terminal vertices is small, the two algorithms produce similar rectilinear Steiner trees with the same total length.

The Z-MRST calculation outperforms the L-MRST calculation with a larger number of terminal vertices. this advantage becomes more pronounced as the number of terminal vertices increases.

The Z-MRST algorithm is more complex and L-MRST is faster than Z-MRST in terms of algorithm running time, but this advantage is not obvious if the number of terminal vertices is small.

4 Summary and outlook

4.1 Summary

With regard to the rectilinear Steiner tree problem, two algorithms are presented in this thesis based on dynamic planning techniques, MRST for line_L and MRST for line_Z. Both algorithms are based on the base class line derivation. This is because Hanan's theorem states that an optimal rectilinear Steiner tree exists when every Steiner point lies at the intersection of two vertical lines containing terminal vertices. The key to MRST is therefore the determination of the Steiner point, which must lie on the Hanan grid. So the algorithm starts by discretizing and ordering the positions of the given terminal vertices, and segment is used to represent each horizontal or vertical line that passes through these points, which in combination is the Hanan grid. I have put both algorithmic approaches in the same program, the input and output functions are common, differ only in the subtree iteration and are displayed via opencv using a common Visualizer function.

From the analysis of the final results it can be seen that the Z-MRST calculations are slightly better than the L-MRST calculations, an advantage that is more obvious when the number of vertices is higher. This is because Z-MRST takes more scenarios into account during iteration by pre-compiling all possible edges. However, the time cost of harvesting a better solution is increasing, especially with a given number of terminal vertices.

Comparing the two algorithms, each with its own advantages and disadvantages, the time and cost of optimising the results is increasing. Therefore, I believe that the algorithm written in this paper has a lot of room for improvement. It can be further optimised from the point of view of reducing redundancy and unnecessary recursions in order to obtain better results.

4.2 Outlook

This paper has achieved some success in the study of the algorithm for the smallest tree of the rectilinear Steiner tree, but there is still much room for improvement. Because the Steiner tree problem is an NP hard problem, this paper presents only the simplest branch of the problem concerning rectilinear Steiner trees. The Steiner tree problem has a very wide range of applications in many fields. They play an important role in computer networks, in the field of biology, in the humanities and in genetic engineering. The rectilinear Steiner tree problem is a very basic problem in VLSI design and has applications in the design phases of VLSI such as physical synthesis, cabling planning, interconnection planning, global and detailed cabling, etc. It can be used to generate optimal cabling solutions and significantly reduce cabling time. Although some minimum rectilinear Steiner tree algorithms and optimisations are given in this paper, there are many more factors that can interfere in real life use. For example, with real

integrated circuit wiring, there are not only endpoints on the circuit board, but also many other components such as chips, buzzers etc. For these "obstacles" on the circuit board, a minimum rectilinear Steiner tree can be used in the MRST. These 'obstacles' on the circuit board can be further investigated on the basis of the MRST (Minimum Rectilinear Steiner Tree), called the Obstacle-avoiding Rectirectilinear Steiner Minimum Tree (MRST). ORSMT also known as the Restricted Version Bottleneck Rectilinear Steiner Tree[7].

There is still a long way to go with regard to the research on the rectilinear Steiner tree problem for realistic VLSI wiring. How to ensure quality, even if the resulting lines are the shortest and the wiring time is reduced, and how to make the algorithm stable and optimised, are key issues for future research on the rectilinear Steiner tree problem.

It is hoped that future researchers will continue to explore the rectilinear Steiner book problem and optimise the algorithm for application in real production[9].

References

[1] Viet Min-yi. Introduction to combinatorial optimization. Hangzhou: Zhejiang Science and Technology Press(2001).

[2] The Steiner Tree problem[J]. Computer Science and Exploration, 2011,38(10):16-22.

[3] Yusheng Zhang , Zhongbin Wang. Minimum rectilinear Steiner tree and its application in detailed wiring[J]. Journal of Hefei University of Technology (Natural Sciences), 1992(S1):100-107.

[4] Huimin Jin, Liang Ma, Zhoumen Wang. Fast algorithm for Euclidean Steiner's optimal tree[J].Computer Applications Research,2006,23(5):60-62.

[5] Jin Huimin, Ma Liang, Wang Zhoumen. Intelligent optimization algorithm for the Euclidean Steiner minimal tree problem[J]. Computer Engineering,2006,32(10):201-203. DOI:10.3969/j.issn.1000-3428.2006.10.075.

[6] Chen Xiuhua, Zhu Natural. A PSO algorithm for solving the RSMT wiring problem[J]. Journal of Minjiang University, 2014,35(5):39-44.

[7] Zhang Hao, Ye Dongyi, Guo Wenzhong. A multilayer barrier-wound right-angle Steiner minimal tree heuristic[J]. Small Microcomputer Systems, 2016, 37(08):1760-1764.

[8] Zhang Hao, Ye Dongyi, Guo Wenzhong. A multilayer barrier-wound right-angle Steiner minimal tree heuristic[J]. Small Microcomputer Systems, 2016, 37(08):1760-1764.

Yang Changling, Yan Xiaolang. MRST hybrid genetic algorithm based on tree coding and its parallel processing[J]. Microelectronics, 1999,29(2):89-95.

[9] Jun Ma,Bo Yang,Shaohan Ma. A practical algorithm for the minimum rectirectilinear steiner tree[J]. Journal of Computer Science and Technology,2000,15(1).

[10] S. D. Lin and D. H. Kim, "Construction of All Rectirectilinear Steiner Minimum Trees on the Hanan Grid and Its Applications to VLSI Design," in IEEE Transactions on ComputerAided Design of Integrated Circuits and Systems.

[11] Garey M R, Johnson D S. The rectirectilinear Steiner tree problem is NP-complete[J]. SIAM Journal on Applied Mathematics, 1977, 32(4): 826-834.