12-2016

Rectilinear Steiner Tree Construction

Zhiliu Zhang
Computer Sciences, zhzhang@cse.unl.edu

RECTILINEAR STEINER TREE CONSTRUCTION

by

Zhiliu Zhang

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professor Jitender S. Deogun

Lincoln, Nebraska, USA

December 2016

RECTILINEAR STEINER TREE CONSTRUCTION

Zhiliu Zhang, M.S.

University of Nebraska, 2016

Advisor: Jitender S. Deogun

The Minimum Rectilinear Steiner Tree (MRST) problem is to find the minimal spanning tree of a set of points (also called terminals) in the plane that interconnects all the terminals and some extra points (called Steiner points) introduced by intermediate junctions, and in which edge lengths are measured in the L1 (Manhattan) metric. This is one of the oldest optimization problems in mathematics that has been extensively studied and has been proven to be NP-complete, thus efficient approximation heuristics are more applicable than exact algorithms.

In this thesis, we present a new heuristic to construct rectilinear Steiner trees (RSTs) with a close approximation of minimum length in $O(n \log n)$ time. To this end, we recursively divide a plane into a set of sub-planes of which optimal rectilinear Steiner trees (optRSTs) can be generated by a proposed exact algorithm called Const_optRST. By connecting all the optRSTs of the sub-planes, a suboptimal MRST is eventually constructed.

We show experimentally that for topologies with up to 100 terminals, the heuristic is 1.06 to 3.45 times faster than RMST, which is an efficient algorithm based on Prim's method, with accuracy improvements varying from 1.31 % to 10.21 %.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we give the background of Steiner tree problem, previous related work, and the organization of this thesis.

1.1 Background

The Steiner tree problem is named after the Swiss mathematician Jacob Steiner and is one of the oldest optimization problems in mathematics. Hanan [5] first considered the concept of minimal rectilinear Steiner tree which is constructed by Manhattan distance due to its importance in VLSI routing and printed circuit boards.

The Minimum Rectilinear Steiner Tree (MRST) problem is to find the minimal spanning tree of a set of points (also called terminals) in the plane that interconnects all the terminals and some extra points (called Steiner points) introduced by intermediate junctions, and in which edge lengths are measured in the L1 (Manhattan) metric. In 1976, M. Garey and D. S. Johnson [4] proved that the rectilinear Steiner problem is NP-complete. Therefore, polynomial-time algorithms for this problem are unlikely to exist. The MRST problem is fundamental to VLSI design, phylogenetic tree reconstruction in biology, network routing, civil engineering, and so on.

1.2 Related Work

In 1966, Hanan [5] introduced the rectilinear Steiner problem in which optimal rectilinear trees can be obtained by constructing horizontal and vertical lines through each terminal. Hanan also proposed an optimal solution for $n \leq 5$ terminals.

Yang and Wing [17] reported the first exact algorithm in 1972 which solves the rectilinear Steiner problem with up to 9 terminals. Hwang [6] first introduced the rectilinear Fullsome Steiner Tree (FSTs) in 1976, which is popular for later research for designing exact and optimal algorithms based on it, and developed the well-known rectilinear FST generators. In 1981, Winter [27] reported an exact algorithm called GeoSteiner which solves the rectilinear Steiner problem with up to 15 terminals. In 1989, Sidorenko [21] made a further progress and proposed an algorithm which is applicable up to 11 terminals. And in 1992, Thomborson, Alpern and Carter [29] reported a similar algorithm for solving the rectilinear Steiner problem with up to 16 terminals. Ganley and Cohoon [34] reported an algorithm which solves the problem with up to 28 terminals in 1994. Salowe and Warme [48] reported an algorithm for solving the problem with 30 terminals in an average of 30 minutes in 1993.

Since the exact algorithms only exist in exponential time, especially for a large number of terminals, thus efficient approximation heuristics are more applicable than exact algorithms. The Batched Iterative 1-Steiner (BI1S) algorithm proposed by Kahng and Robins [30] computes rectilinear Steiner trees efficiently. The improved BI1S [38] is currently the most near-optimal approximation algorithm for the rectilinear Steiner problem. However, the BI1S has a time complexity of $O(n^4 \log n)$. In practice, RMST [32], which is Prim's algorithm ($O(n^2)$) for computing minimal rectilinear spanning trees, is frequently used for computing minimal rectilinear Steiner trees.

1.3 About This Thesis

1.3.1 Motivation

Many previous exponential-time algorithms tackle the rectilinear Steiner problem for large instances, however, in VLSI routing applications, a typical instance often contains a few terminals [13]. In practice, most VLSI applications have 30 terminals or less, and the performance, including CPU runtime and accuracy of total length, is clearly critical. Therefore, our objective in this thesis is to seek a heuristic algorithm which is both efficient and accurate for VLSI applications with small instances (30 terminals or less).

1.3.2 Contributions

Our major contributions are as follows:

1. We propose a mathematical modeling method to reformulate the minimum rectilinear Steiner tree problem as a problem to be more tractable that only contains permutation sequences in Hanan grid.

2. We create an optimal algorithm $const\_optRST$ which can efficiently solve the rectilinear Steiner problem with up to 7 points.

3. We introduce a new heuristic $RSTC$ which is excellent in both efficiency and accuracy for constructing rectilinear Steiner trees with up to 100 terminals in the plane.

1.3.3 Outline of Thesis

This thesis is structured as follows. Chapter 2 reformulates the minimum rectilinear Steiner tree problem and gives some definitions and properties. Chapter 3 proposes an exact solution for growing optimal rectilinear Steiner trees. Chapter 4 introduces a heuristic algorithm for constructing the rectilinear Steiner trees with a set of terminals in the plane, and describes our experiments and compares our results with a very efficient Prim's algorithm RMST [32]. Finally, Chapter 5 concludes this thesis with future work.

Chapter 2

The Rectilinear Steiner Tree Problem

In this chapter, we propose a mathematical modeling method to reformulate the minimum rectilinear Steiner tree problem to be more tractable. In Section 2.1 we define the rectilinear Steiner tree problem formally. In Section 2.2 we redefine the problem as a reduced problem that only contains a set of permutation points instead. In Section 2.3 we introduce some structural properties of rectilinear Steiner tree in Hanan grid.

2.1 Problem Formulation

Given a finite set T of terminals in the plane, the Hanan grid $H(T)$ is a grid net which is composed of vertical and horizontal lines through each terminal in T.

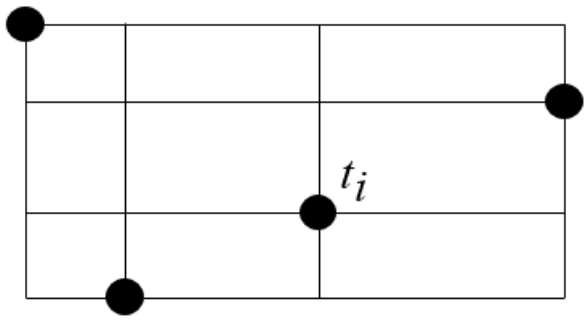

Figure 2.1: Hanan grid $H(T)$

In Figure 2.1, terminal t_i in T is the i-th terminal in the Hanan grid and has two attributes which are x-coordinate and y-coordinate: $t_i(t_i.x, t_i.y)$. In Hanan grid, the distance between any two terminals is measured in the L1 (Manhattan) metric: $|t_1.x - t_2.x| + |t_1.y - t_2.y|$.

The Minimum Rectilinear Steiner Tree (MRST) Problem: Given a finite set V of n ($n \in Z^+$, $n > 1$) points in the plane, determine a minimal spanning tree T with a set S ($V \subset S$) of nodes in the rectilinear distance.

A point of V is called a terminal and a node of $S - V$ is called a Steiner point. We denote RST to be rectilinear Steiner tree and $optRST$ to be optimal rectilinear Steiner tree.

2.2 Problem Reformulation

Here, we introduce a set of n points with permutation order. By sorting n points according to their x-coordinates or y-coordinates, a permutation order can be generated.

Lemma 2.1. *For a given set of terminals $T = \{t_1, t_2, ..., t_n\}$ in the plane, there exists a corresponding permutation $C = c_1c_2...c_n$.*

Proof. Let T be sorted into $T = \{t_i | \, t_i.x \leq t_{i+1}.x, 1 \leq i \leq n\}$ or $T = \{t_i | \, t_i.y \leq t_{i+1}.y, 1 \leq i \leq n\}$, therefore, $T_y = \{t_i.y | 1 \leq i \leq n\} = \{t_1.y, t_2.y, ..., t_n.y\}$ or $T_x = \{t_i.x | 1 \leq i \leq n\} = \{t_1.x, t_2.x, ..., t_n.x\}$ is an order which can be mapped to a permutation according to their relative positions. □

We design an algorithm for mapping a set of terminals to a permutation as follows:

Algorithm 2.1: $TerMapPermut$

Input : Given n terminals in the plane
Output: the corresponding permutation

```
1 sort T by y-coordinates
2 for i = 1 to n do
3 |   t_i.order = i
4 end
5 sort T by x-coordinates
6 for each terminal t_i ∈ T do
7 |   add t_i.order into vector C
8 end
9 return C
```

In the algorithm $TerMapPermut$, each terminal in $T = \{t_1, t_2, ..., t_n\}$ has three attributes ($t_i.x$, $t_i.y$, $t_i.order$): x-coordinate, y-coordinate, and the order of sorted position. C is the permutation of n given terminals. In time complexity, line 2 - 4 and line 6 - 8 are in $O(n)$ time. Therefore, the entire runtime depends on line 1 and line 5 which can be done by the typical merge-sort algorithm in $O(n \log n)$ time.

A permutation C can be structured into a set of permutation points $P = \{p_i | 1 \leq i \leq n\}$ of which x-coordinates / y-coordinates to be the permutation and y-coordinates / x-coordinates to be an increasing order.

Lemma 2.2. *For a given permutation $C = c_1 c_2 ... c_n$, there exists a corresponding graph $G(P)$ that consists of a set of permutation points $P = \{p_1, p_2, ..., p_n\}$.*

Proof. For $G(V) = \{v_1, v_2, ..., v_n\}$, let $V_x = \{v_1.x = c_1, v_2.x = c_2, ..., v_n.x = c_n\}$ or $V_y = \{v_1.y = c_1, v_2.y = c_2, ..., v_n.y = c_n\}$, and $V_y = \{v_1.y = 1, v_2.y = 2, ..., v_n.y = n\}$ or $V_x = \{v_1.x = 1, v_2.x = 2, ..., v_n.x = n\}$, therefore the vertices of G are a set of permutation points. □

We design an algorithm for structuring a permutation to a graph $G(P)$ with permutation points as follows:

Algorithm 2.2: $Permut - G(P)$

Input : a permutation C
Output: the corresponding $G(P)$ with permutation points

```
1 create a vector P = {p_i | 1 ≤ i ≤ n}
2 for i = 1 to n do
3 |   p_i.x = c_i
4 |   p_i.y = i
5 end
6 return P
```

In the algorithm $Permut - G(P)$, $G(P)$ is a graph which only consists of a set of permutation points $P = \{p_i | p_i.x = c_i, p_i.y = i, 1 \leq i \leq n\}$. Its obvious that the time complexity is $O(n)$.

Theorem 1. *Given a set of terminals $T = \{t_1, t_2, ..., t_n\}$ in the plane, there exists a graph $G(P)$ with a set of permutation points $P = \{p_i | 1 \leq i \leq n\}$ which derives from T.*

Proof. By Lemma 2.1 and Lemma 2.2, a set of terminals T can be transformed into a graph $G(P)$ which only contains a set of permutation points. □

The minimum rectilinear Steiner tree problem seeks an optimal rectilinear Steiner tree for a given set of n terminals in the plane, however, if we can transform the problem with real distances into a problem with relative distances, the given n terminals can be modeled as a set of permutation points, and thus the original problem is reformulated to be finding an optimal rectilinear Steiner tree for a set of permutation points.

Here, we reformulate the problem as: Given a finite set V of permutation points in the plane, determine a minimal spanning tree T with a set S ($V \subset S$) of nodes in the rectilinear distance.

According to Theorem 1, in the rest of this thesis, we only study graph $G(P)$ with given n permutation points in the plane.

2.3 Structural Properties

We define $G(P, E, L)$ to be a graph with a set of permutation points in the Hanan grid, P to be the set of vertices of G, E to be the set of edges of G which are nonnegative, and L to be the minimal length of G. We assume that graph $G(P, E, L)$ is always underlined in Hanan grid $H(G)$.

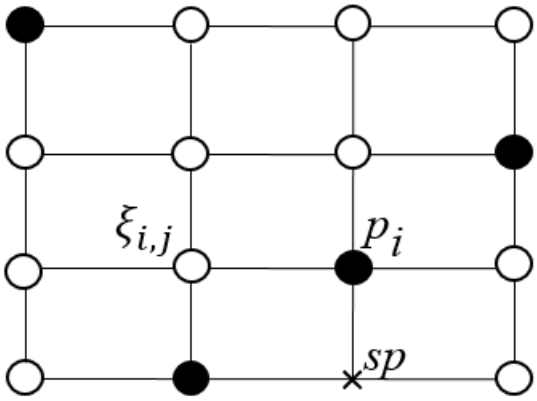

Figure 2.2: Hanan grid $H(G(P))$

Figure 2.2 shows that $\xi_{i,j}$ is a grid point located in i-th horizontal line and j-th vertical line, and p_i is the i-th vertex of graph $G(P, E, L)$. We use solid circles ($\bullet$) to denote vertices, open circles ($\circ$) to denote grid points, and crosses ($\times$) to denote Steiner points.

The rectilinear Steiner tree of graph G grows in Hanan grid $H(G)$ with horizontal lines $H = \{h(i)|1 \leq i \leq m\}$, vertical lines $V = \{v(j)|1 \leq j \leq n\}$, and intersection points $\zeta = \{\xi_{i,j}|1 \leq i \leq m,\ 1 \leq j \leq n\}$. Therefore, we have

$h(i)$: the i-th ($1 \leq i \leq m$) horizontal grid line;

$v(j)$: the j-th ($1 \leq j \leq n$) vertical grid line;

$\xi_{i,j}$: the grid point intersected by the i-th horizontal grid line and the j-th vertical grid line.

Suppose we have two grid points a and b, we define

$|a - b|$ = rectilinear distance between a and b;

$|a.x - b.x|$ = horizontal distance between a and b;

$|a.y - b.y|$ = vertical distance between a and b.

We use upper letters $(A, B, C, ...)$ to denote permutation points, lower case letters $(a, b, c, ...)$ to denote grid points, $e(p_i, p_j)$ to denote an edge between two vertices p_i and p_j, $\varphi(a, b)$ to denote a segment between two horizontally or vertically adjacent grid points a and b. For any two grid points a and b (not necessarily adjacent), a rectilinear path $\delta(a, b)$ is a segment chain connecting a and b. $|\delta(a, b)|$ is the length of the path starting from a and ending in b. $|\varphi(a, b)|$ is the length between a and b. $|G(P)|$ is the number of vertices in G, and in this thesis we only discuss graph G in which $|G| \geq 2$.

Definition 2.1. *For any two adjacent grid points a and b, $|\varphi(a, b)| = 1$.*

We assume that, in this thesis, any segment between two grid points a and b has 1 unit length.

Chapter 3

Generation of Optimal Rectilinear Steiner Trees

In this chapter, we present an optimal solution to solve the rectilinear Steiner problem with up to 7 points. In Section 3.1 we introduce basic concepts and preliminaries about optimal Steiner trees. In Section 3.2 we propose the method of growing Steiner trees. In Section 3.3 we describe the exact algorithm $Const\_optRST$ for generating optimal rectilinear Steiner trees efficiently.

3.1 Preliminaries

We denote $optRST(G)$ to be an optimal rectilinear Steiner tree of $G(P)$, $B(G)$ to be a subgraph which are the boundaries of $G(P)$, and $R(G)$ to be an enclosing rectangle which contains all the terminals of $G(P)$ in the plane. To compute an optimal rectilinear Steiner tree of a set of permutation points $P = \{p_i | 1 \leq i \leq n\}$, we use a grid matrix to represent Hanan grid points. A grid matrix consists of all the grid points which are the intersections of Hanan grid $H(G)$. Therefore, we have

$$\Lambda = \begin{bmatrix} \xi_{1,1} & \xi_{1,2} & \cdots & \xi_{1,n} \\ \xi_{2,1} & \xi_{2,2} & \cdots & \xi_{2,n} \\ \cdots & \cdots & \xi_{i,j} & \cdots \\ \xi_{m,1} & \xi_{m,2} & \cdots & \xi_{m,n} \end{bmatrix}$$

In the matrix Λ, $\xi_{i,j}$ is the grid point located in the i-th row and the j-th column. An edge/path is called absolute edge/path if there is no other terminals or Steiner points within the edge/path.

Definition 3.1. *Optimal rectilinear Steiner trees with the same length in each two x-coordinates and in each two y-coordinates are equivalent trees.*

3.2 Growing Steiner Trees

In this section, we propose a method for how to grow optimal rectilinear Steiner trees with $n \leq 7$ points in Hanan grid.

Lemma 3.1. *Let $G' = \{\xi_{i,j}, \xi_{s,t}\}$ be a subgraph of $G(P)$ for some $1 \leq i \leq m$, $1 \leq j \leq n$, $1 \leq s \leq m$, $1 \leq t \leq n$, $i \neq s$ and $j \neq t$, if there is an absolute path $\delta(\xi_{i,j}, \xi_{s,t})$ that belongs to an $optRST(G')$, then there exists an alternative path $\delta'(\xi_{i,j}, \xi_{s,t})$ that belongs to another $optRST(G')$.*

Proof. Suppose $\delta(\xi_{i,j}, \xi_{s,t}) = \delta(\xi_{i,j}, \xi_{i,t}) \cup \delta(\xi_{i,t}, \xi_{s,t})$, then there exists another path $\delta'(\xi_{i,j}, \xi_{s,t}) = \delta(\xi_{i,j}, \xi_{s,j}) \cup \delta(\xi_{s,j}, \xi_{s,t})$, and we have

$$|\delta(\xi_{i,j},\xi_{s,t})| = |\delta(\xi_{i,j},\xi_{i,t})| + |\delta(\xi_{i,t},\xi_{s,t})|$$
$$= \sum_{x=i}^{i}\sum_{y=j}^{t}|\varphi(x,y)| + \sum_{x=i}^{s}\sum_{y=t}^{t}|\varphi(x,y)|$$
$$= \sum_{y=j}^{t}|\varphi(i,y)| + \sum_{x=i}^{s}|\varphi(x,t)|$$
$$|\delta'(\xi_{i,j},\xi_{s,t})| = |\delta(\xi_{i,j},\xi_{s,j})| + |\delta(\xi_{s,j},\xi_{s,t})|$$
$$= \sum_{x=i}^{s}\sum_{y=j}^{j}|\varphi(x,y)| + \sum_{x=s}^{s}\sum_{y=j}^{t}|\varphi(x,y)|$$
$$= \sum_{x=i}^{s}|\varphi(x,j)| + \sum_{y=j}^{t}|\varphi(s,y)|$$

Because $\sum_{y=j}^{t}|\varphi(s,y)| = \sum_{y=j}^{t}|\varphi(i,y)|$ and $\sum_{x=i}^{s}|\varphi(x,j)| = \sum_{x=i}^{s}|\varphi(x,t)|$, therefore $|\delta(\xi_{i,j}, \xi_{s,t})| = |\delta'(\xi_{i,j},\xi_{s,t})|$. □

Lemma 3.2. *Let $G' = \{e(\xi_{i,j},\xi_{s,j}), e(\xi_{p,k},\xi_{t,k})\}$ ($s \neq p$ and $i \neq t$) to be a subgraph of $G(P)$, if there exists an $optRST(G')$ that contains an absolute edge e_q incident to G', then by replacing e_q with an alternative edge e_r ($e_q \neq e_r$) which is perpendicular to $e(\xi_{i,j},\xi_{s,j})$ and $e(\xi_{p,k},\xi_{t,k})$ results in another $optRST(G')$.*

Proof. Without loss of generality, assume e_q is incident to $\xi_{q,j}$ and $\xi_{q,k}$, and e_r is incident to $\xi_{r,j}$ and $\xi_{r,k}$, therefore we have

$$|e_q| = |\delta(\xi_{q,j},\xi_{q,k})| = \sum_{x=q}^{q}\sum_{y=j}^{k}|\varphi(x,y)| = \sum_{y=j}^{k}|\varphi(q,y)|$$
$$|e_r| = |\delta(\xi_{r,j},\xi_{r,k})| = \sum_{x=r}^{r}\sum_{y=j}^{k}|\varphi(x,y)| = \sum_{y=j}^{k}|\varphi(r,y)|$$

Because $\sum_{y=j}^{k}|\varphi(q,y)| = \sum_{y=j}^{k}|\varphi(r,y)|$, therefore $|e_q| = |e_r|$. □

Lemma 3.3. *Let $G' = G \cap \bigcup_{\substack{1\le x\le i\\ 1\le y\le n}}\{\xi_{i,j}\}$ ($1 \le i \le m$), if $G' = \{\xi_{i,j} | \xi_{i,j} \in \Lambda\}$ for some $1 \le i \le m$ and $1 \le j \le n$, then there exists an $optRST(G)$ in which the segment $\varphi(\xi_{i,j},\xi_{i+1,j})$ must be present.*

Proof. Since $\xi_{i+1,j}$ must be connected to $R(G\backslash G')$ it is trivial to prove that the segment $\varphi(\xi_{i,j},\xi_{i+1,j})$ must be present in an $optRST(G)$ if $\xi_{i,j}$ is incident to $\xi_{i+1,j}$.

If $\xi_{i,j}$ is incident to a random grid point $\xi_{i+1,k}$ for some $1 \le k \le n$, without loss of generality, assume that $j \le k$, then by Lemma 3.1, $\delta(\xi_{i,j},\xi_{i+1,k}) = \delta(\xi_{i,j},\xi_{i,k}) \cup \delta(\xi_{i,k},\xi_{i+1,k})$ can be replaced with $\delta'(\xi_{i,j},\xi_{i+1,k}) = \delta(\xi_{i,j},\xi_{i+1,j}) \cup \delta(\xi_{i+1,j},\xi_{i+1,k})$, and we have

$$
\begin{aligned}
&|\delta(\xi_{i,j},\xi_{i+1,k})| = |\delta(\xi_{i,j},\xi_{i,k})| + |\delta(\xi_{i,k},\xi_{i+1,k})| \\
&= \sum_{x=i}^{i}\sum_{y=j}^{k}|\varphi(x,y)| + \sum_{x=i}^{i+1}\sum_{y=k}^{k}|\varphi(x,y)| \\
&= \sum_{y=j}^{k}|\varphi(i,y)| + \sum_{x=i}^{i+1}|\varphi(x,k)| \\
&|\delta'(\xi_{i,j},\xi_{i+1,k})| = |\delta(\xi_{i,j},\xi_{i+1,j})| + |\delta(\xi_{i+1,j},\xi_{i+1,k})| \\
&= \sum_{x=i}^{i+1}\sum_{y=j}^{j}|\varphi(x,y)| + \sum_{x=i+1}^{i+1}\sum_{y=j}^{k}|\varphi(x,y)| \\
&= \sum_{x=i}^{i+1}|\varphi(x,j)| + \sum_{y=j}^{k}|\varphi(i+1,y)|
\end{aligned}
$$

Because $\sum_{y=j}^{k}|\varphi(i,y)| = \sum_{y=j}^{k}|\varphi(i+1,y)|$ and $\sum_{x=i}^{i+1}|\varphi(x,k)| = \sum_{x=i}^{i+1}|\varphi(x,j)|$, therefore $|\delta(\xi_{i,j},\xi_{i+1,k})| = |\delta'(\xi_{i,j},\xi_{i+1,k})|$.

If $\xi_{i,j}$ is incident to multiple random grid points $\gamma = \{\xi_{i+1,r},...,\xi_{i+1,s},...,\xi_{i+1,t}\}$ $(1 \leq j \leq r \leq s \leq n)$, according to Lemma 3.2, $\delta(\xi_{i,s},\xi_{i,t})$ can be replaced with $\delta(\xi_{i+1,s},\xi_{i+1,t})$, consequently leading $\varphi(\xi_{i,t},\xi_{i+1,t})$ to being redundant. Likewise, $\delta(\xi_{i,r},\xi_{i,s})$ can be replaced with $\delta(\xi_{i+1,r},\xi_{i+1,s})$ leaving $\varphi(\xi_{i,s},\xi_{i+1,s})$ to being redundant. Let $G_1 = G \cap \bigcup_{\substack{i\leq x\leq i+1\\1\leq y\leq n}}\{\xi_{x,y}\}$, $G_2 = \delta(\xi_{i,j},\xi_{i,r}) \cup \delta(\xi_{i,r},\xi_{i,s}) \cup \delta(\xi_{i,s},\xi_{i,t})$, $G_3 = \{\varphi(\xi_{i,y},\xi_{i+1,y})|r \leq y \leq n, \xi_{i+1,y} \in G_1\}$, $G_4 = \delta(\xi_{i,j},\xi_{i+1,r}) \cup \delta(\xi_{i+1,r},\xi_{i+1,s}) \cup \delta(\xi_{i+1,s},\xi_{i+1,t})$, $\alpha = G_1 \cup G_2 \cup G_3$ and $\beta = G_1 \cup G_4$, and we have $|\alpha| = |\delta(\xi_{i,j},\xi_{i,r})| + |\varphi(\xi_{i,r},\xi_{i+1,r})| + |\delta(\xi_{i,r},\xi_{i,s})| + |\delta(\xi_{i,s},\xi_{i,t})| + |G_3 \backslash \varphi(\xi_{i,r},\xi_{i+1,r})|$, $|\beta| = |\delta(\xi_{i,j},\xi_{i+1,j})| + |\varphi(\xi_{i+1,j},\xi_{i+1,r})| + |\delta(\xi_{i+1,r},\xi_{i+1,s})| + |\delta(\xi_{i+1,s},\xi_{i+1,t})|$. Because $|\varphi(\xi_{i,r},\xi_{i+1,r})| = |\delta(\xi_{i,j},\xi_{i+1,j})|$, $|\delta(\xi_{i,j},\xi_{i,r}) = |\varphi(\xi_{i+1,j},\xi_{i+1,r})|$, $|\alpha| = |\beta| + |G_3|$. And also because $|G_3 \backslash \varphi(\xi_{i,r},\xi_{i+1,r})| > 1$, $|\beta| < |\alpha|$. Therefore, in this case, $\varphi(\xi_{i,j},\xi_{i+1,j})$ must be present in an $optRST$ for G which contains $|\beta|$.

If $\xi_{i,j}$ is incident to $G \backslash \{\xi_{i,j}\}$ through a Steiner point, then that Steiner point can be removed. Thus, the segment $\varphi(\xi_{i,j},\xi_{i+1,j})$ must be present in an $optRST$ for G. □

Corollary 3.1. *Let* $G' = G \cap \bigcup_{\substack{1\leq x\leq m\\1\leq y\leq j}}\{\xi_{i,j}\}$ $(1 \leq j \leq n)$, *if* $G' = \{\xi_{i,j}|\xi_{i,j} \in \Lambda\}$ *for some* $1 \leq i \leq m$ *and* $1 \leq j \leq n$, *then there exists an* $optRST(G)$ *in which the segment* $\varphi(\xi_{i,j},\xi_{i,j+1})$ *must be present.*

Proof. When $G' = G \cap \bigcup_{\substack{1\leq x\leq m\\1\leq y\leq j}}\{\xi_{i,j}\}$ $(1 \leq j \leq n)$, it can be clockwise rotated to $G' =$

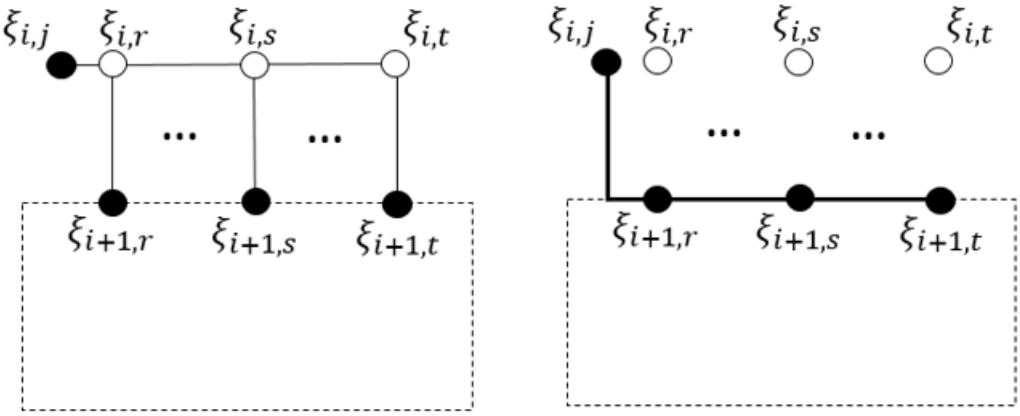

Figure 3.1: A terminal incident to multiple grid points

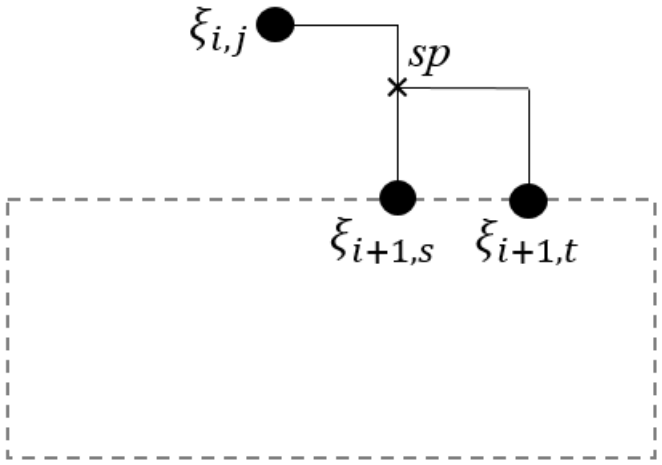

Figure 3.2: A path through a Steiner point

$G \cap \bigcup_{\substack{1\le x\le i\\ 1\le y\le n}} \{\xi_{i,j}\}$ $(1 \le i \le m)$ by 90 degrees in geometry. According to Lemma 3.3, the segment $\varphi(\xi_{i,j}, \xi_{i,j+1})$ must be present in an $optRST(G)$. □

Corollary 3.2. *Let* $G' = G \cap \bigcup_{\substack{1\le x\le m\\ j\le y\le n}} \{\xi_{i,j}\}$ $(1 \le j \le n)$, *if* $G' = \{\xi_{i,j} | \xi_{i,j} \in \Lambda\}$ *for some* $1 \le i \le m$ *and* $1 \le j \le n$, *then there exists an* $optRST(G)$ *in which the segment* $\varphi(\xi_{i,j}, \xi_{i,j-1})$ *must be present.*

Proof. When $G' = G \cap \bigcup_{\substack{1\le x\le m\\ j\le y\le n}} \{\xi_{i,j}\}$ $(1 \le j \le n)$, it can be anticlockwise rotated to $G' = G \cap \bigcup_{\substack{1\le x\le i\\ 1\le y\le n}} \{\xi_{i,j}\}$ $(1 \le i \le m)$ by 90 degrees in geometry. According to Lemma 3.3, the segment $\varphi(\xi_{i,j}, \xi_{i,j-1})$ must be present in an $optRST(G)$. □

Corollary 3.3. *Let* $G' = G \cap \bigcup_{\substack{i\le x\le m\\ 1\le y\le n}} \{\xi_{i,j}\}$ $(1 \le i \le m)$, *if* $G' = \{\xi_{i,j} | \xi_{i,j} \in \Lambda\}$ *for*

some $1 \leq i \leq m$ *and* $1 \leq j \leq n$*, then there exists an* $optRST(G)$ *in which the segment* $\varphi(\xi_{i,j}, \xi_{i-1,j})$ *must be present.*

Proof. When $G' = G \cap \bigcup\limits_{\substack{i \leq x \leq m \\ 1 \leq y \leq n}} \{\xi_{i,j}\}$ $(1 \leq i \leq m)$, it can be anticlockwise rotated to $G' = G \cap \bigcup\limits_{\substack{1 \leq x \leq i \\ 1 \leq y \leq n}} \{\xi_{i,j}\}$ $(1 \leq i \leq m)$ by 180 degrees in geometry. According to Lemma 3.3, the segment $\varphi(\xi_{i,j}, \xi_{i-1,j})$ must be present in an $optRST(G)$. □

Lemma 3.4. *Let* $L(x) = G(P) \cap \bigcup\limits_{j=1}^{n} \{\xi_{x,j}\}$ *for some* $1 \leq x \leq m$ *and* $R'(G) = R(G) \cap \bigcup\limits_{x=1}^{i} \{L(x)\} (1 \leq i \leq m)$*, if* $R'(G) = \{\xi_{i,j}, \xi_{i,k}\}$*, for some* $1 \leq i \leq m$*,* $1 \leq j \leq n$ *and* $1 \leq k \leq n$*, then there exists an* $optRST(R(G))$*:*

$$optRST(R(G)) = min \begin{cases} optRST(R_1(G)) \cup \varphi(\xi_{i,j}, \xi_{i+1,j}) \cup \varphi(\xi_{i,k}, \xi_{i+1,k}) \\ optRST(R_2(G)) \cup \varphi(\xi_{i,j}, \xi_{i+1,j}) \cup \varphi(\xi_{i,j}, \xi_{i,k}) \\ optRST(R_3(G)) \cup \varphi(\xi_{i,j}, \xi_{i,k}) \cup \varphi(\xi_{i,k}, \xi_{i+1,k}) \end{cases}$$

$$where \begin{cases} R_1(G) = R(G) \backslash \{\xi_{i,j}, \xi_{i,k}\} \cup \{\xi_{i+1,j}, \xi_{i+1,k}\} \\ R_2(G) = R(G) \backslash \{\xi_{i,j}, \xi_{i,k}\} \cup \{\xi_{i+1,j}\} \\ R_3(G) = R(G) \backslash \{\xi_{i,j}, \xi_{i,k}\} \cup \{\xi_{i+1,k}\} \end{cases}$$

Proof. We first prove that there are at most 2 edges for the grid points $\xi_{i,j}$ and $\xi_{i,k}$ to be incident to $R(G \backslash \{\xi_{i,j}, \xi_{i,k}\})$. Suppose $\xi_{i,j}$ and $\xi_{i,k}$ are incident to more than 2 points in $R(G \backslash \{\xi_{i,j}, \xi_{i,k}\})$, say $\alpha = \{\xi_{i+1,s}, \xi_{i+1,s+1}, ..., \xi_{i+1,t-1}, \xi_{i+1,t}\}$. According to Lemma 3.2, we notice that $\delta(\xi_{i,t-1}, \xi_{i,t})$ and $\delta(\xi_{i,t-2}, \xi_{i,t-1})$ can be replaced with $\delta(\xi_{i+1,t-1}, \xi_{i+1,t})$ and $\delta(\xi_{i+1,t-2}, \xi_{i+1,t-1})$, resulting in $\varphi(\xi_{i,t-1}, \xi_{i+1,t-1})$ to be redundant. Likewise, it goes until the path $\varphi(\xi_{i,s+2}, \xi_{i,s}) = \varphi(\xi_{i,s+2}), \xi_{i,s+1}) \cup \varphi(\xi_{i,s+1}, \xi_{i,s})$ is replaced with the path $\varphi(\xi_{i+1,s+2}, \xi_{i+1,s}) = \varphi(\xi_{i+1,s+2}), \xi_{i+1,s+1}) \cup \varphi(\xi_{i+1,s+1}, \xi_{i+1,s})$ making the segment $\varphi(\xi_{i,s+1}, \xi_{i+1,s+1})$ be redundant. Finally, a set of segments $\beta = \{\varphi(\xi_{i,s+1}, \xi_{i+1,s+1}),$ $\varphi(\xi_{i,s+2}, \xi_{i+1,s+2}), ..., \varphi(\xi_{i,t-1}, \xi_{i+1,t-1})$ can be removed, resulting in two paths $\varphi(\xi_{i,j}, \xi_{i+1,s})$ and $\varphi(\xi_{i,k}, \xi_{i+1,t})$ connected to $R(G \backslash \{\xi_{i,j}, \xi_{i,k}\})$. Therefore, there exists an $optRST(G)$ in which $\xi_{i,j}$ and $\xi_{i,k}$ are incident to $R(G \backslash \{\xi_{i,j}, \xi_{i,k}\})$ with at most two edges ($e(\xi_{i,s}, \xi_{i+1,s})$ and $e(\xi_{i,t}, \xi_{i+1,t})$).

Case I: $\xi_{i,j}$ and $\xi_{i,k}$ are connected to $R(G \backslash \{\xi_{i,j}, \xi_{i,k}\})$ with two edges.

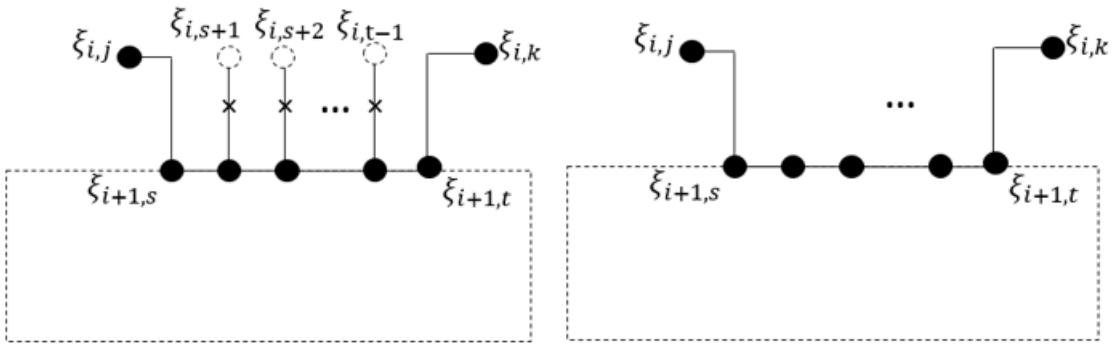

Figure 3.3: Two terminals incident to multiple grid points

(a) If $\xi_{i,j}$ and $\xi_{i,k}$ are incident to $R(G\backslash\{\xi_{i,j}, \xi_{i,k}\})$ through grid points $\xi_{i,s}(j < s)$ and $\xi_{i,t}(t < k)$ respectively. In this case, the path $\delta(\xi_{i,j}, \xi_{i+1,s}) = \delta(\xi_{i,j}, \xi_{i,s}) \cup \varphi(\xi_{i,s}, \xi_{i+1,s})$ can be replaced with the path $\delta'(\xi_{i,j}, \xi_{i+1,s}) = \varphi(\xi_{i,j}), \xi_{i+1,j}) \cup \delta(\xi_{i+1,j}, \xi_{i+1,s})$. In the same way, the path $\delta(\xi_{i,k}, \xi_{i+1,t}) = \delta(\xi_{i,k}), \xi_{i,t}) \cup \varphi(\xi_{i,t}, \xi_{i+1,t})$ also can be replaced with the path $\delta'(\xi_{i,k}, \xi_{i+1,t}) = \varphi(\xi_{i,k}), \xi_{i+1,k}) \cup \delta(\xi_{i+1,k}, \xi_{i+1,t})$.

(b) If $\xi_{i,j}$ and $\xi_{i,k}$ are incident to $R(G\backslash\{\xi_{i,j}, \xi_{i,k}\})$ through grid points $\xi_{i,s}(s < j)$ and $\xi_{i,t}(k < t)$ respectively. In this case, the path $\delta(\xi_{i,j}, \xi_{i+1,s}) = \delta(\xi_{i,j}, \xi_{i,s}) \cup \varphi(\xi_{i,s}, \xi_{i+1,s})$ can be replaced with the path $\delta'(\xi_{i,j}, \xi_{i+1,s}) = \varphi(\xi_{i,j}), \xi_{i+1,j}) \cup \delta(\xi_{i+1,j}, \xi_{i+1,s})$. In the same way, the path $\delta(\xi_{i,k}, \xi_{i+1,t}) = \delta(\xi_{i,k}), \xi_{i,t}) \cup \varphi(\xi_{i,t}, \xi_{i+1,t})$ also can be replaced with the path $\delta'(\xi_{i,k}, \xi_{i+1,t}) = \varphi(\xi_{i,k}), \xi_{i+1,k}) \cup \delta(\xi_{i+1,k}, \xi_{i+1,t})$.

Case II: $\xi_{i,j}$ and $\xi_{i,k}$ are connected to $R(G\backslash\{\xi_{i,j}, \xi_{i,k}\})$ with one edge.

(a) If $\xi_{i,j}$ is incident to $R(G\backslash\{\xi_{i,j}, \xi_{i,k}\})$ through the grid point $\xi_{i+1,j}$. In this case, $optRST(R_2(G)) \cup \varphi(\xi_{i,j}, \xi_{i+1,j}) \cup \varphi(\xi_{i,j}, \xi_{i,k})$ $(R_2(G) = R(G)\backslash\{\xi_{i,j}, \xi_{i,k}\} \cup \{\xi_{i+1,j}\})$ is a candidate for $optRST(R(G))$. (b) If $\xi_{i,j}$ is incident to $R(G\backslash\{\xi_{i,j}, \xi_{i,k}\})$ through the grid point $\xi_{i+1,k}$. In this case, $optRST(R_3(G)) \cup \varphi(\xi_{i,j}, \xi_{i,k}) \cup \varphi(\xi_{i,k}, \xi_{i+1,k})$ $(R_3(G) = R(G)\backslash\{\xi_{i,j}, \xi_{i,k}\} \cup \{\xi_{i+1,k}\})$ is a candidate for $optRST(R(G))$.

Case III: $\xi_{i,j}$ and $\xi_{i,k}$ are connected to $R(G\backslash\{\xi_{i,j}, \xi_{i,k}\})$ with no edge. Since there must be at least one edge between $\alpha = \{\xi_{i,j}, \xi_{i,k}\}$ and $G\backslash\{\xi_{i,j}, \xi_{i,k}\}$, therefore this case never happens. □

Corollary 3.4. *Let* $L(y) = G(P) \cap \bigcup_{i=1}^{m} \{\xi_{i,y}\}$ *for some* $1 \leq y \leq n$ *and* $R'(G) = R(G) \cap \bigcup_{y=1}^{j} \{L(y)\}(1 \leq j \leq n)$, *if* $R'(G) = \{\xi_{i,j}, \xi_{k,j}\}$, *for some* $1 \leq i \leq m$, $1 \leq k \leq m$, *and*

$1 \leq j \leq n$ *then there exists an* $optRST(R(G))$*:*

$$optRST(R(G)) = min \begin{cases} optRST(R_1(G)) \cup \varphi(\xi_{i,j}, \xi_{i,j+1}) \cup \varphi(\xi_{k,j}, \xi_{k,j+1}) \\ optRST(R_2(G)) \cup \varphi(\xi_{i,j}, \xi_{i,j+1}) \cup \varphi(\xi_{i,j}, \xi_{k,j}) \\ optRST(R_3(G)) \cup \varphi(\xi_{i,j}, \xi_{k,j}) \cup \varphi(\xi_{k,j}, \xi_{k,j+1}) \end{cases}$$

$$\textit{where} \begin{cases} R_1(G) = R(G) \backslash \{\xi_{i,j}, \xi_{k,j}\} \cup \{\xi_{i,j+1}, \xi_{k,j+1}\} \\ R_2(G) = R(G) \backslash \{\xi_{i,j}, \xi_{k,j}\} \cup \{\xi_{i,j+1}\} \\ R_3(G) = R(G) \backslash \{\xi_{i,j}, \xi_{k,j}\} \cup \{\xi_{k,j+1}\} \end{cases}$$

Proof. When $R'(G) = R(G) \cap \bigcup_{y=1}^{j} \{L(y)\} (1 \leq j \leq n)$, it can be clockwise rotated to $R'(G) = R(G) \cap \bigcup_{x=1}^{i} \{L(x)\} (1 \leq i \leq m)$ by 90 degrees in geometry. According to Lemma 3.4, $optRST(R_1(G))$, $optRST(R_2(G))$ and $optRST(R_3(G))$ are candidates for $optRST(R(G))$. □

Corollary 3.5. *Let* $L(y) = G(P) \cap \bigcup_{i=1}^{m} \{\xi_{i,y}\}$ *for some* $1 \leq y \leq n$ *and* $R'(G) = R(G) \cap \bigcup_{j \leq y}^{n} \{L(y)\} (1 \leq j \leq n)$*, if* $R'(G) = \{\xi_{i,j}, \xi_{k,j}\}$*, for some* $1 \leq i \leq m$*,* $1 \leq k \leq m$*, and* $1 \leq j \leq n$ *then there exists an* $optRST(R(G))$*:*

$$optRST(R(G)) = min \begin{cases} optRST(R_1(G)) \cup \varphi(\xi_{i,j}, \xi_{i,j-1}) \cup \varphi(\xi_{k,j}, \xi_{k,j-1}) \\ optRST(R_2(G)) \cup \varphi(\xi_{i,j}, \xi_{i,j-1}) \cup \varphi(\xi_{i,j}, \xi_{k,j}) \\ optRST(R_3(G)) \cup \varphi(\xi_{i,j}, \xi_{k,j}) \cup \varphi(\xi_{k,j}, \xi_{k,j-1}) \end{cases}$$

$$\textit{where} \begin{cases} R_1(G) = R(G) \backslash \{\xi_{i,j}, \xi_{k,j}\} \cup \{\xi_{i,j-1}, \xi_{k,j-1}\} \\ R_2(G) = R(G) \backslash \{\xi_{i,j}, \xi_{k,j}\} \cup \{\xi_{i,j-1}\} \\ R_3(G) = R(G) \backslash \{\xi_{i,j}, \xi_{k,j}\} \cup \{\xi_{k,j-1}\} \end{cases}$$

Proof. When $R'(G) = R(G) \cap \bigcup_{j \leq y}^{n} \{L(y)\} (1 \leq j \leq n)$, it can be anticlockwise rotated to $R'(G) = R(G) \cap \bigcup_{x=1}^{i} \{L(x)\} (1 \leq i \leq m)$ by 90 degrees in geometry. According to Lemma 3.4, $optRST(R_1(G))$, $optRST(R_2(G))$ and $optRST(R_3(G))$ are candidates for $optRST(R(G))$. □

Corollary 3.6. *Let* $L(x) = G(P) \cap \bigcup_{j=1}^{n} \{\xi_{x,j}\}$ *for some* $1 \leq x \leq m$ *and* $R'(G) = R(G) \cap \bigcup_{i \leq x}^{m} \{L(x)\} (1 \leq i \leq m)$*, if* $R'(G) = \{\xi_{i,j}, \xi_{i,k}\}$*, for some* $1 \leq i \leq m$*,* $1 \leq j \leq n$ *and* $1 \leq k \leq n$*, then there exists an* $optRST(R(G))$*:*

$$optRST(R(G)) = min \begin{cases} optRST(R_1(G)) \cup \varphi(\xi_{i,j}, \xi_{i-1,j}) \cup \varphi(\xi_{i,k}, \xi_{i-1,k}) \\ optRST(R_2(G)) \cup \varphi(\xi_{i,j}, \xi_{i-1,j}) \cup \varphi(\xi_{i,j}, \xi_{i,k}) \\ optRST(R_3(G)) \cup \varphi(\xi_{i,j}, \xi_{i,k}) \cup \varphi(\xi_{i,k}, \xi_{i-1,k}) \end{cases}$$

$$\textit{where} \begin{cases} R_1(G) = R(G) \backslash \{\xi_{i,j}, \xi_{i,k}\} \cup \{\xi_{i-1,j}, \xi_{i-1,k}\} \\ R_2(G) = R(G) \backslash \{\xi_{i,j}, \xi_{i,k}\} \cup \{\xi_{i-1,j}\} \\ R_3(G) = R(G) \backslash \{\xi_{i,j}, \xi_{i,k}\} \cup \{\xi_{i-1,k}\} \end{cases}$$

Proof. When $R'(G) = R(G) \cap \bigcup_{i \leq x}^{m} \{L(x)\} (1 \leq i \leq m)$, it can be clockwise rotated to $R'(G) = R(G) \cap \bigcup_{x=1}^{i} \{L(x)\} (1 \leq i \leq m)$ by 180 degrees in geometry. According to Lemma 3.4, $optRST(R_1(G))$, $optRST(R_2(G))$ and $optRST(R_3(G))$ are candidates for $optRST(R(G))$. □

3.3 Optimal Algorithm

In this section, we design an optimal algorithm called $const\_optRST$ which can generate the rectilinear Steiner trees with $n \leq 7$ points.

We denote $MinX(G)$ to be a set of points which are extracted from G and have the minimal x-coordinate, $MinY(G)$ to be a set of points which are extracted from G and have the minimal y-coordinate, $MaxX(G)$ to be a set of points which are extracted from G and have the maximal x-coordinate, and $MaxY(G)$ to be a set of points which are extracted from G and have the maximal y-coordinate.

Algorithm 3.1: $Const\_optRST$

Input : A given $G(P)$ in which $|G(P)| \leq 7$
Output: the $optRSTs$ of $G(P)$

```
1  set TreeList = ∅
2  set G.grown = false
3  add G into TreeList
4  TreeList.generated = false
5  for each G ∈ TreeList do
6      if G.grown = false then
7          if extreme(G) = true then
8              fork(G, TreeList)
9          end
10         else
11             set G.grown = true
12         end
13     end
14 end
15 set TreeList.generated = true
16 return min(TreeList)
```

In the algorithm $Const\_optRST$, $TreeList$ is a linked list of which each node stores a graph. When a graph is constructed as a rectlinear tree, $G.grown$ is set to be ture. At first, we add the given graph $G(P)$ into the $TreeList$. And then we reduce the graph by $extreme(G)$. When $extreme(G)$ is finished and returns true, we further reduce the graph with three forked subgraphs by $fork(G)$. And then add the subgraphs into $TreeList$ and delete the original graph. When $extreme(G)$ returns false, that means the graph has been formed as a subtree. When each subgraph in the $TreeList$ has became a subtree, $TreeList.generated$ is set to be true. At last, we return the subtree with the minimal length. In time complexity, it depends on $extreme(G)$ and $fork(G)$ which can be implemented in $O(n)$ and $O(3^n)$ respectively, therefore the entire algorithm is in $O(3^n)$ time.

Algorithm 3.2: $extreme(G)$

Input : A given $G(P)$
Output: True or false

if $G.size == 2$ **then**
  $G.P = G.p_2$
  $G.E = G.E \cup e(G.p_1, G.p_2)$
  $G.L = G.L + |e(G.p_1, G.p_2)|$
  return false
end
if $MinY(MinX(G)) = MaxY(MinX(G))$ **then**
  $G.P = G.P \backslash \{\xi_{i,j}\} \cup \{\xi_{i,j+1}\}$
  $G.E = G.E \cup \varphi(\xi_{i,j}, \xi_{i,j+1})$
  $G.L = G.L + |\varphi(\xi_{i,j}, \xi_{i,j+1})|$
end
else if $MinY(MaxX(G)) = MaxY(MaxX(G))$ **then**
  $G.P = G.P \backslash \{\xi_{i,j}\} \cup \{\xi_{i,j-1}\}$
  $G.E = G.E \cup \varphi(\xi_{i,j}, \xi_{i,j-1})$
  $G.L = G.L + |\varphi(\xi_{i,j}, \xi_{i,j-1})|$
end
else if $MinX(MaxY(G)) = MaxX(MaxY(G))$ **then**
  $G.P = G.P \backslash \{\xi_{i,j}\} \cup \{\xi_{i-1,j}\}$
  $G.E = G.E \cup \varphi(\xi_{i,j}, \xi_{i-1,j})$
  $G.L = G.L + |\varphi(\xi_{i,j}, \xi_{i-1,j})|$
end
else if $MinX(MinY(G)) = MaxX(MinY(G))$ **then**
  $G.P = G.P \backslash \{\xi_{i,j}\} \cup \{\xi_{i+1,j}\}$
  $G.E = G.E \cup \varphi(\xi_{i,j}, \xi_{i+1,j})$
  $G.L = G.L + |\varphi(\xi_{i,j}, \xi_{i+1,j})|$
end
else
  return true;
end
extreme(G);

In the routine $extreme(G)$, we recursively reduce a graph according to Lemma 3.3, Corollary 3.1, Corollary 3.2 and Corollary 3.3. In time complexity, it can be implemented in $O(n)$ time.

Algorithm 3.3: $fork(G, TreeList)$

Input : G, $TreeList$
Output: the updated $TreeList$ with forked subgraphs from G

1 create G_1, G_2 and G_3
2 set $G_1 = G$, $G_2 = G$ and $G_3 = G$
3 **if** $MinX(G) = MinY(MinX(G)) \cup MaxY(MinX(G))$ **then**
4 $G_1.P = G_1 \backslash \{\xi_{i,j}, \xi_{i,k}\} \cup \{\xi_{i+1,j}, \xi_{i+1,k}\}$
5 $G_1.E = G_1.E \cup \varphi(\xi_{i,j}, \xi_{i+1,j}) \cup \varphi(\xi_{i,k}, \xi_{i+1,k})$
6 $G_1.L = G_1.L + |\varphi(\xi_{i,j}, \xi_{i+1,j})| + |\varphi(\xi_{i,k}, \xi_{i+1,k})|$
7 add G_1into $TreeList$
8 $G_2.P = G_2 \backslash \{\xi_{i,j}, \xi_{i,k}\} \cup \{\xi_{i+1,j}\}$
9 $G_2.E = G_2.E \cup \varphi(\xi_{i,j}, \xi_{i+1,j}) \cup \varphi(\xi_{i,j}, \xi_{i,k})$
10 $G_2.L = G_2.L + |\varphi(\xi_{i,j}, \xi_{i+1,j})| + |\varphi(\xi_{i,j}, \xi_{i,k})|$
11 add G_2 into $TreeList$
12 $G_3.P = G_3 \backslash \{\xi_{i,j}, \xi_{i,k}\} \cup \{\xi_{i+1,k}\}$
13 $G_3.E = G_3.E \cup \varphi(\xi_{i,k}, \xi_{i+1,k}) \cup \varphi(\xi_{i,j}, \xi_{i,k})$
14 $G_3.L = G_3.L + |\varphi(\xi_{i,k}, \xi_{i+1,k})| + |\varphi(\xi_{i,j}, \xi_{i,k})|$
15 add G_3 into $TreeList$
16 $TreeList = TreeList \backslash G$
17 return $TreeList$;
18 **end**
19 **if** $MaxX(G) = MinY(MaxX(G)) \cup MaxY(MaxX(G))$ **then**
20 ... ...
21 **end**
22 **if** $MinY(G) = MinX(MinY(G)) \cup MaxX(MinY(G))$ **then**
23 ... ...
24 **end**
25 **if** $MaxY(G) = MinX(MaxY(G)) \cup MaxX(MaxY(G))$ **then**
26 ... ...
27 **end**

Line 20 in the routine $fork(G, TreeList)$:

1 $G_1.P = G_1 \backslash \{\xi_{i,j}, \xi_{i,k}\} \cup \{\xi_{i-1,j}, \xi_{i-1,k}\}$
2 $G_1.E = G_1.E \cup \varphi(\xi_{i,j}, \xi_{i-1,j}) \cup \varphi(\xi_{i,k}, \xi_{i-1,k})$
3 $G_1.L = G_1.L + |\varphi(\xi_{i,j}, \xi_{i-1,j})| + |\varphi(\xi_{i,k}, \xi_{i-1,k})|$
4 add G_1into $TreeList$
5 $G_2.P = G_2 \backslash \{\xi_{i,j}, \xi_{i,k}\} \cup \{\xi_{i-1,j}\}$
6 $G_2.E = G_2.E \cup \varphi(\xi_{i,j}, \xi_{i-1,j}) \cup \varphi(\xi_{i,j}, \xi_{i,k})$
7 $G_2.L = G_2.L + |\varphi(\xi_{i,j}, \xi_{i-1,j})| + |\varphi(\xi_{i,j}, \xi_{i,k})|$
8 add G_2 into $TreeList$
9 $G_3.P = G_3 \backslash \{\xi_{i,j}, \xi_{i,k}\} \cup \{\xi_{i-1,k}\}$
10 $G_3.E = G_3.E \cup \varphi(\xi_{i,k}, \xi_{i-1,k}) \cup \varphi(\xi_{i,j}, \xi_{i,k})$
11 $G_3.L = G_3.L + |\varphi(\xi_{i,k}, \xi_{i-1,k})| + |\varphi(\xi_{i,j}, \xi_{i,k})|$
12 add G_3 into $TreeList$
13 $TreeList = TreeList \backslash G$
14 return $TreeList$;

Line 23 in the routine $fork(G, TreeList)$:

---

1 $G_1.P = G_1 \backslash \{\xi_{i,j}, \xi_{i,k}\} \cup \{\xi_{i,j+1}, \xi_{i,k+1}\}$
2 $G_1.E = G_1.E \cup \varphi(\xi_{i,j}, \xi_{i,j+1}) \cup \varphi(\xi_{i,k}, \xi_{i,k+1})$
3 $G_1.L = G_1.L + |\varphi(\xi_{i,j}, \xi_{i,j+1})| + |\varphi(\xi_{i,k}, \xi_{i,k+1})|$
4 add G_1into $TreeList$
5 $G_2.P = G_2 \backslash \{\xi_{i,j}, \xi_{i,k}\} \cup \{\xi_{i,j+1}\}$
6 $G_2.E = G_2.E \cup \varphi(\xi_{i,j}, \xi_{i,j+1}) \cup \varphi(\xi_{i,j}, \xi_{i,k})$
7 $G_2.L = G_2.L + |\varphi(\xi_{i,j}, \xi_{i,j+1})| + |\varphi(\xi_{i,j}, \xi_{i,k})|$
8 add G_2 into $TreeList$
9 $G_3.P = G_3 \backslash \{\xi_{i,j}, \xi_{i,k}\} \cup \{\xi_{i,k+1}\}$
10 $G_3.E = G_3.E \cup \varphi(\xi_{i,k}, \xi_{i,k+1}) \cup \varphi(\xi_{i,j}, \xi_{i,k})$
11 $G_3.L = G_3.L + |\varphi(\xi_{i,k}, \xi_{i,k+1})| + |\varphi(\xi_{i,j}, \xi_{i,k})|$
12 add G_3 into $TreeList$
13 $TreeList = TreeList \backslash G$
14 return $TreeList$;

Line 26 in the routine $fork(G, TreeList)$:

---

1 $G_1.P = G_1 \backslash \{\xi_{i,j}, \xi_{i,k}\} \cup \{\xi_{i,j-1}, \xi_{i,k-1}\}$
2 $G_1.E = G_1.E \cup \varphi(\xi_{i,j}, \xi_{i,j-1}) \cup \varphi(\xi_{i,k}, \xi_{i,k-1})$
3 $G_1.L = G_1.L + |\varphi(\xi_{i,j}, \xi_{i,j-1})| + |\varphi(\xi_{i,k}, \xi_{i,k-1})|$
4 add G_1into $TreeList$
5 $G_2.P = G_2 \backslash \{\xi_{i,j}, \xi_{i,k}\} \cup \{\xi_{i,j-1}\}$
6 $G_2.E = G_2.E \cup \varphi(\xi_{i,j}, \xi_{i,j-1}) \cup \varphi(\xi_{i,j}, \xi_{i,k})$
7 $G_2.L = G_2.L + |\varphi(\xi_{i,j}, \xi_{i,j-1})| + |\varphi(\xi_{i,j}, \xi_{i,k})|$
8 add G_2 into $TreeList$
9 $G_3.P = G_3 \backslash \{\xi_{i,j}, \xi_{i,k}\} \cup \{\xi_{i,k-1}\}$
10 $G_3.E = G_3.E \cup \varphi(\xi_{i,k}, \xi_{i,k-1}) \cup \varphi(\xi_{i,j}\xi_{i,k})$
11 $G_3.L = G_3.L + |\varphi(\xi_{i,k}, \xi_{i,k-1})| + |\varphi(\xi_{i,j}, \xi_{i,k})|$
12 add G_3 into $TreeList$
13 $TreeList = TreeList \backslash G$
14 return $TreeList$;

In the routine $fork(G)$, we further reduce the graph by forking three subgraphs according to Lemma 3.4, Corollary 3.4, Corollary 3.5 and Corollary 3.6. In time complexity, it can be implemented in $O(3^n)$ time.

Theorem 2. *The algorithm $Const\_optRST$ is optimal for growing rectilinear Steiner trees for any $G(P)$ in which $|G(P)| \leq 7$.*

Proof. We prove that Theorem 2 is correct for 3, 4, 5, 6 and 7 points respectively.

Type 1: $|G(P)| = 3$.

When there are 3 points in the plane, without loss of generality, suppose they are

randomly distributed as (a). By corollary 3.1, the segment f must be present in an $optRST(G)$ as shown in (b). By seeding a quasi-terminal A' into $G(P)\backslash\{A\}$, then $G(P)$ can be reduced to $G' = G(P)\backslash\{A\} \cup \{A'\} = \{A', B, C\}$ as shown in (c). Likewise, the segment g is also present in the $optRST(G)$, and by seeding a quasi-terminal B', G' can be further reduced to $G'' = G'\backslash\{B\} \cup \{B'\} = \{A', B', C\}$ as shown in (e). Finally, by recursively applying $extreme(G)$, G'' can be reduced to $G''' = \{C\}$ as shown in (f). Therefore, $extreme(G)$ can generate all the $optRSTs$ for $G(P)$ in which $|G(P)| = 3$ and reduce it to a single point.

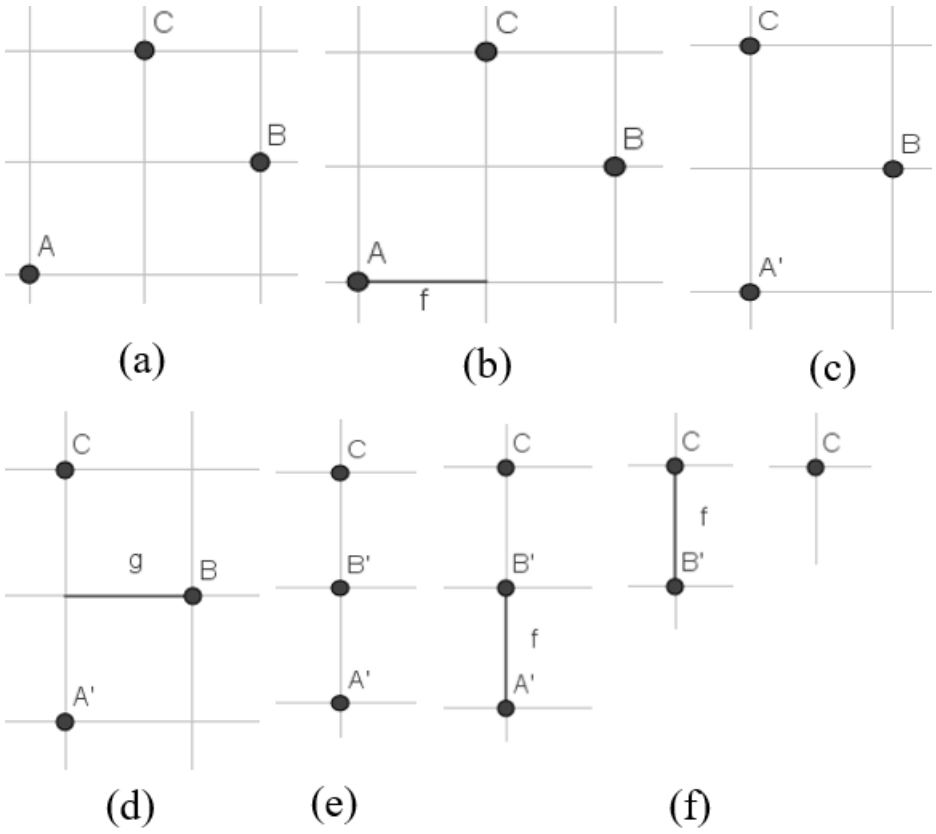

Figure 3.4: Type 1

Type 2: $|G(P)| = 4$.

Case I: $|B(G)| < 4$. When there are 4 points in the plane, without loss of generality, suppose they are randomly distributed as shown in (a). First of all, by the line 7 - 11 and line 12 - 16 of $extreme(G)$, two segments f and g are added into $G.E$ and $G(P)$ is reduced to $G' = \{A', B, C, D'\}$. Likewise, G' can be reduced to $G'' = \{A'', B, C, D''\}$ by the line 17 - 21 and line 22 - 26. If $G(P)$ is reduced to three points in the plane,

then according to the conclusion of Type 1, it can be further reduced to a single point. Therefore, in this case, any $G(P)$ can be reduced to a subgraph with $|B(G')| = 4$ or a single point.

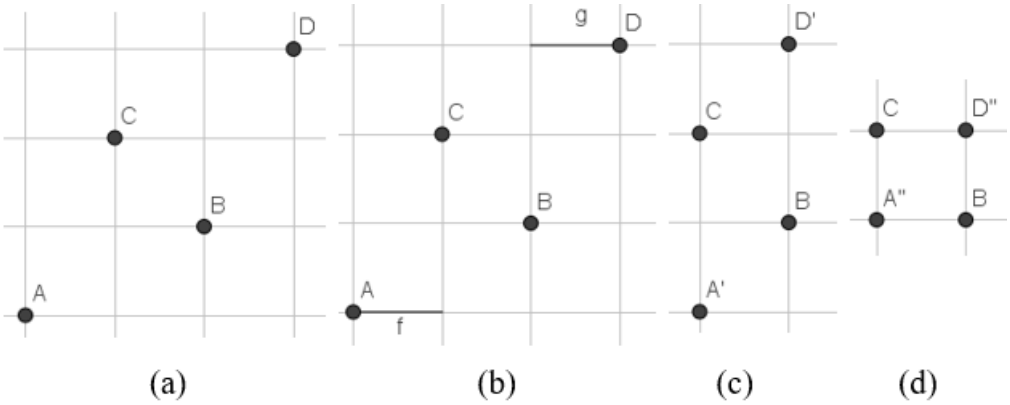

Figure 3.5: Type 2, Case 1

Case II: $|B(G)| = 4$. In this case, assume the vertices of $G(P)$ are distributed as shown in (a), the $optRSTs$ of $G(P)$ can be mapped into two topologies ((b) and (c)). For growing the $optRST$ as shown in (b), we apply the line 3 - 18 of $fork(G)$, and two paths f and g are added into $G.E$. Therefore $G(P)$ is reduced to $G' = \{D, B\}$ which can be further reduced to $G'' = \{D\}$ by the line 22 - 26 of $extreme(G)$. Likewise, for growing the $optRST$ as shown in (c), we apply the line 22 - 24 of $fork(G)$, and path f and path g are added into $G.E$. Hence $G(P)$ is reduced to $G' = \{C, D\}$ which can be further reduced to $G'' = \{D\}$ by the line 7 - 11 of $extreme(G)$.

Type 3: $|G(P)| = 5$.

Case I: $|B(G)| = 5$. In this case, there are only two possible topologies as shown in (b) and (c). By applying the line 22 - 24 of $fork(G)$, $G(P) = \{A, B, C, D, E\}$ can be reduced to $G' = \{C, E, D\}$ as shown in (d), and according to the conclusion of Type 1, G' can be further reduced to a single point. Likewise, by applying the line 3 - 18 of $fork(G)$, $G(P)$ can be reduced to $G' = \{F, B, E, D\}$ as shown in (e), and according to the conclusion of Type 2, G' can be further reduced to a single point. If there is an edge which leaves point E horizontally and enters into edge $e(A, B)$ by introducing a Steiner point F, such an $optRST$ is equivalent to the topology as shown in (c).

Case II: $|B(G)| < 4$. In this case, according to the conclusions of Type 1 and Type 2,

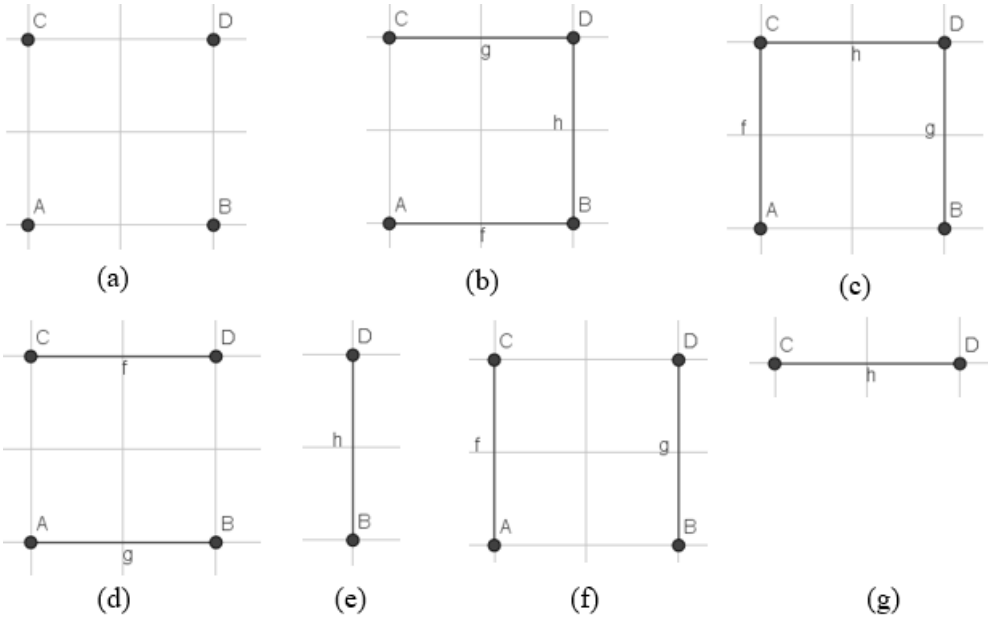

Figure 3.6: Type 2, Case 2

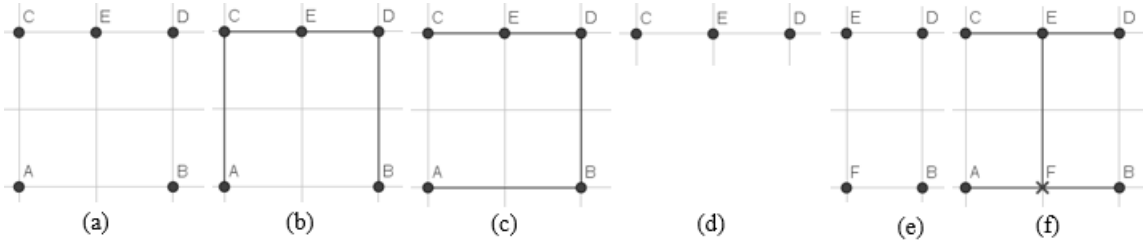

Figure 3.7: Type 3, Case 1

$G(P)$ can be transformed to a subgraph with $|B(G)| = 4$ or reduced to a single point.

Case III: $|B(G)| = 4$. In this case, there is only one point inside boundaries. If there is an edge which is incident to point E and perpendicular to edge $e(B, D)$ by introducing a Steiner point F as shown in (b), then there is only one possible topology as (c) illustrates. Likewise, if there is an edge which is incident to point E and perpendicular to edge $e(C, D)$ by introducing a Steiner point F as shown in (d), then there is also only one possible topology as (e) illustrates.

Type 4: $|G(P)| = 6$.

Case I: $|B(G)| = 6$. When all the points on boundaries, without loss of generality, suppose the vertices of $G(P)$ are distributed as shown in (a). Since it is impossible that there are two edges which are perpendicular to $e(A, C)$ and $e(A, B)$ both in the middle, if there is no edge perpendicular to $e(A, C)$ in the middle, then by applying the line 3 - 18 of

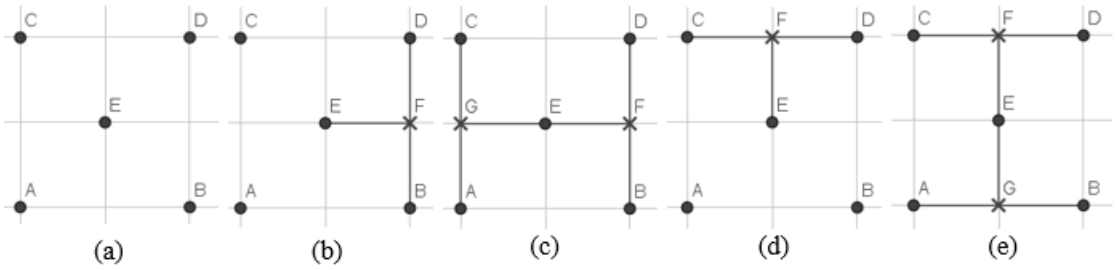

Figure 3.8: Type 3, Case 3

$fork(G)$, $G(P)$ can be reduced to $G' = \{E, D, F, G, B\}$ as (b) illustrates, and according to the conclusion of Type 3, it can be eventually reduced to a single point. Likewise, if there is no edge perpendicular to $e(A, B)$ in the middle, then by applying the line 22 - 24 of $fork(G)$, $G(P)$ can be reduced to $G' = \{C, E, D, A, F\}$ as (c) illustrates, and according to the conclusion of Type 3, G' can be eventually reduced to a single point.

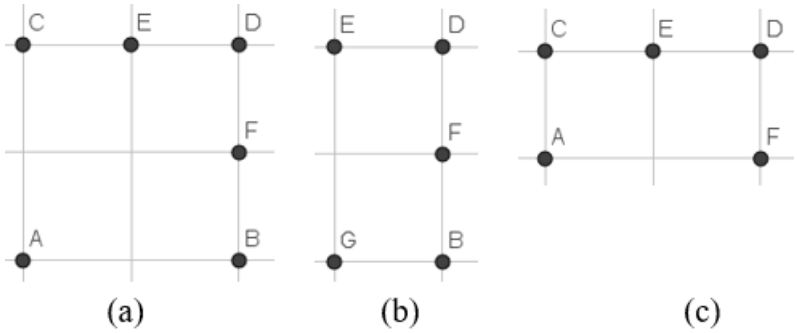

Figure 3.9: Type 4, Case 1

Case II: $|B(G)| = 5$, and $|G(P) \backslash B(G)| = 1$. Without loss of generality, suppose the vertices of $G(P)$ are distributed as shown in (a), therefore it is impossible for point F is perpendicular to $e(A, C)$ and $e(A, B)$ both in the middle. If F is only perpendicular to $e(A, C)$ ((b)), then by applying the line 22 - 24 of $fork(G)$, $G(P)$ can be reduced to $G' = \{C, E, D, A, F, B\}$, and according to the conclusion of Case I, G' can be eventually reduced to a single point. If point F is only perpendicular to $e(A, B)$ as shown in (d), then by applying the line 3 - 18 of $fork(G)$, $G(P)$ can be reduced to $G' = \{C, E, D, F, A, B\}$ as shown in (e), and according to the conclusion of Case I, G' can be eventually reduced to a single point.

Case III: $|B(G)| = 4$, and $|G(P) \backslash B(G)| = 2$. If point E and F are perpendicular to $e(A, C)$ as shown in (b), it is impossible for E or F is perpendicular to $e(C, D)$ or

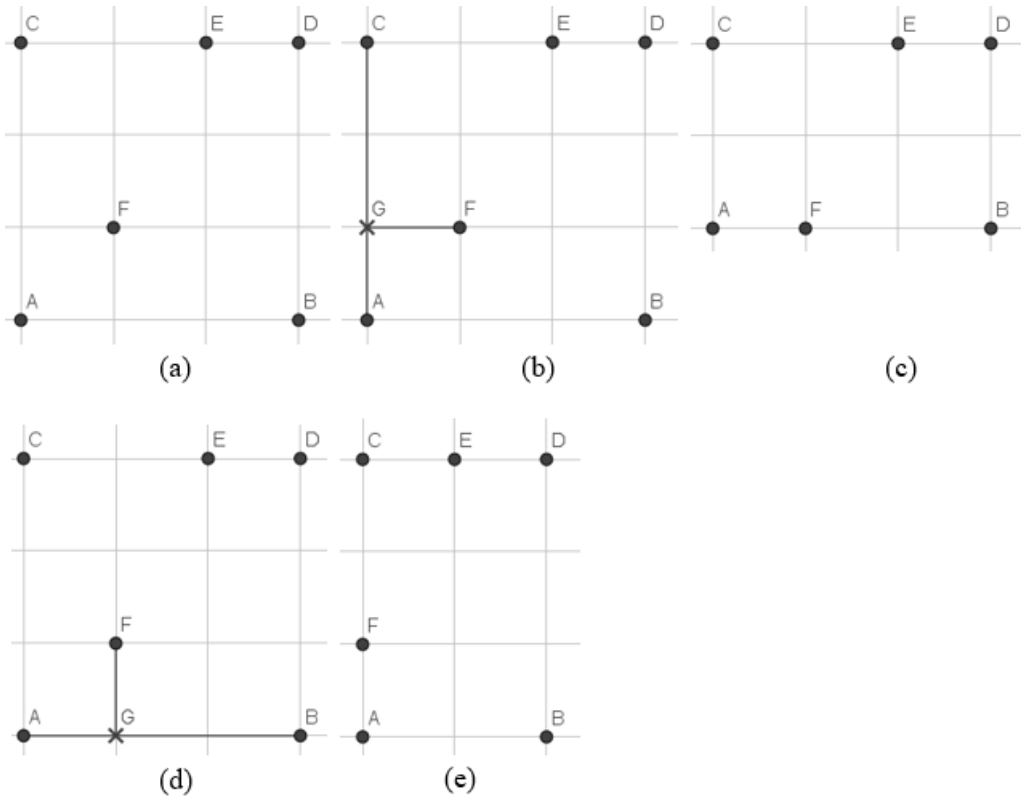

Figure 3.10: Type 4, Case 2

$e(A, B)$. Then by applying the line 25 - 27 or line 22 - 24 of $fork(G)$, $G(P)$ can be reduced to (c) or (d), and according to the conclusion of Case II, it can be eventually reduced to a single point.

If point E and point F are perpendicular to different edges, assume $e(A, C)$ and $e(B, D)$, it is impossible for E perpendicular to $e(A, B)$ and F perpendicular to $e(C, D)$. Then by applying the line 25 - 27 or line 22 - 24 of $fork(G)$, $G(P)$ can be reduced to (c) or (d), and according to the conclusion of Case II, it can be eventually reduced to a single point. If there is an edge which is perpendicular to $e(C, D)$ and $e(A, B)$ through point F, then by applying the line 19 - 21 of $fork(G)$, $G(P)$ can be reduced to (g), and according to the conclusion of Case II, it can be eventually reduced to a single point.

Type 5: $|G(P)| = 7$.

Case I: $|B(G)| = 7$. Suppose there is an edge which is perpendicular to $e(D, E)$ as shown in (b), if point G is incident to $e(D, E)$ as shown in (c), it can be transformed to (d). By applying the line 25 - 27 of $fork(G)$, (d) can be reduced to (g). If point G is incident to $e(D, E)$ as shown in (e), it can be transformed to (f). By applying the line 25 -

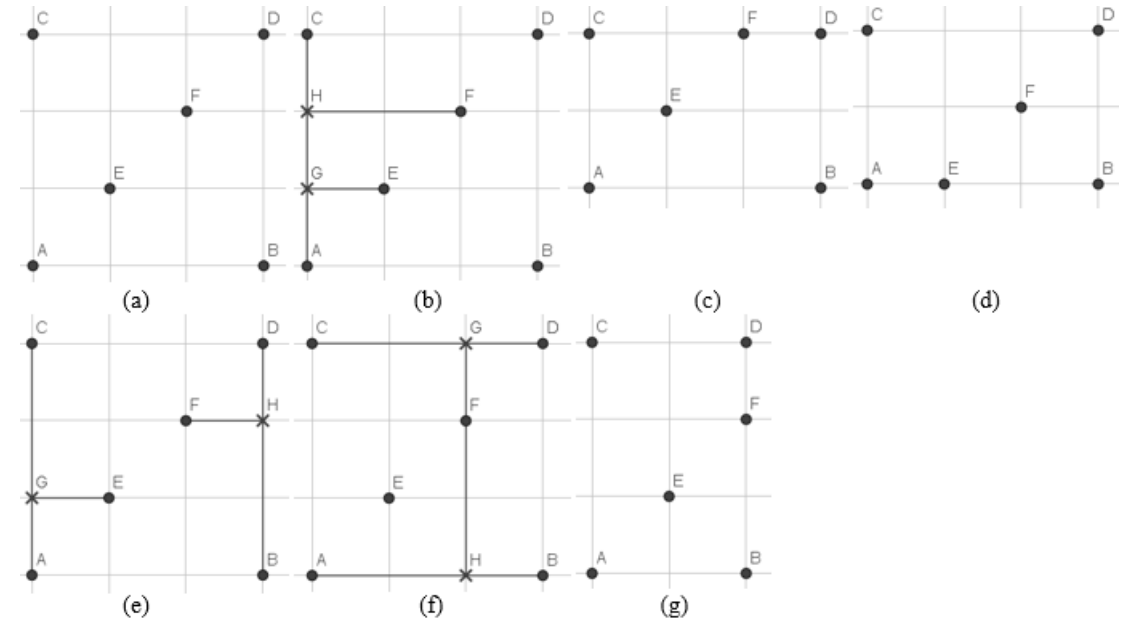

Figure 3.11: Type 4, Case 3

27 of $fork(G)$, (d) can be reduced to (g). According to the conclusion of Type 4, (g) can be eventually reduced to a single point.

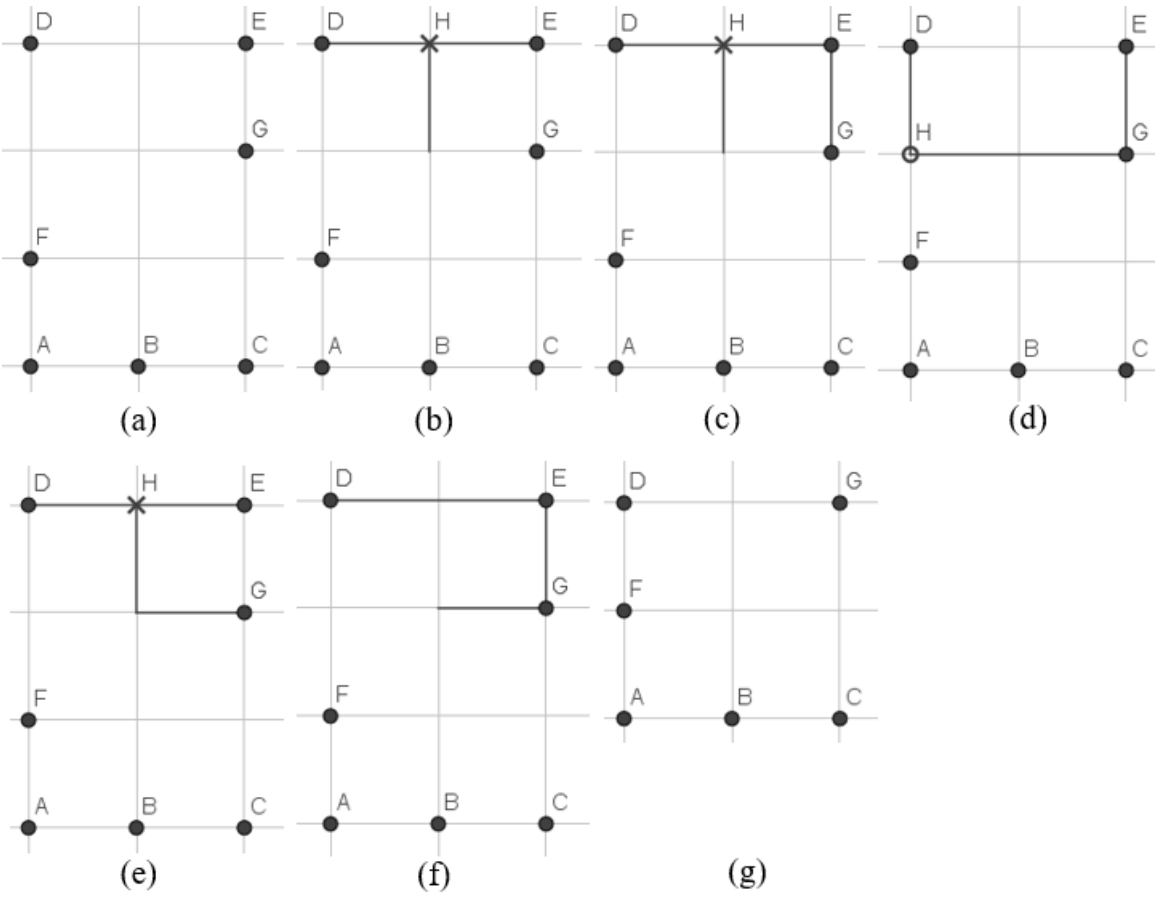

Figure 3.12: Type 5, Case 1

Case II: $|B(G)| = 6$. Suppose point G is perpendicular to edge $e(A, F)$ as shown in (b), it is impossible for point G is also perpendicular to $e(F, E)$, therefore by applying the line 25 - 27 of $fork(G)$, (b) can be reduced to (c), and according to the conclusion of Type 3, it can be eventually reduced to a single point.

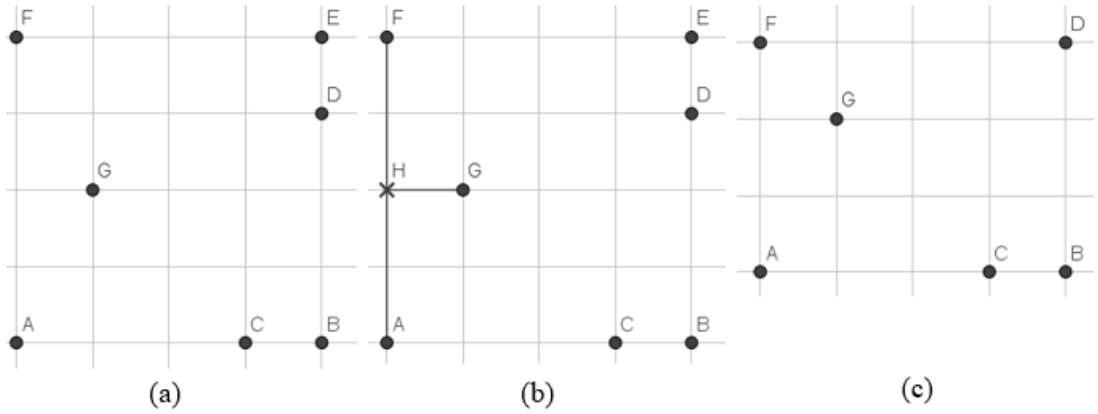

Figure 3.13: Type 5, Case 2

Case III: $|B(G)| = 5$. Suppose point G is perpendicular to $e(A, F)$ and point D is perpendicular to $e(F, E)$ as shown in (b), it is impossible for point G or point D is perpendicular to $e(E, B)$. Then by applying the line 19 - 21 of $fork(G)$, (b) can be reduced to (c), and according to the conclusion of Type 4, it can be eventually reduced to a single point.

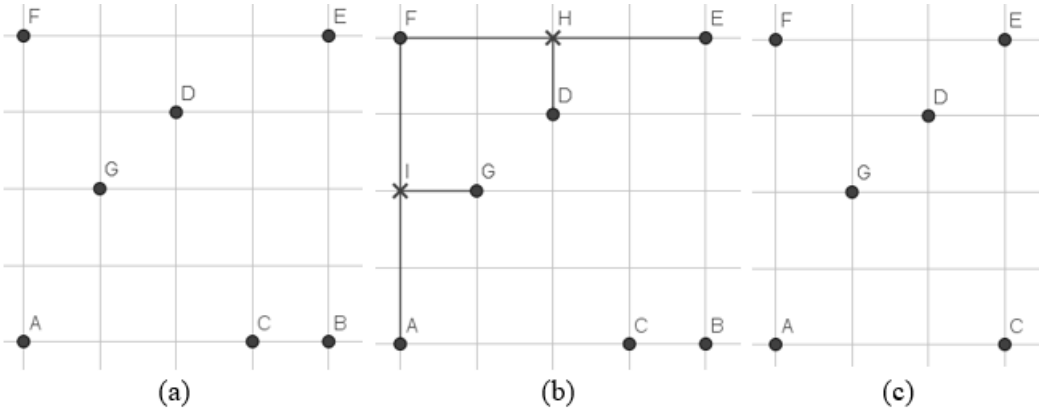

Figure 3.14: Type 5, Case 3

Case IV: $|B(G)| = 4$. Suppose point E is perpendicular to $e(A, B)$, point F is perpendicular to $e(C, B)$, and point G is perpendicular to $e(D, C)$ as shown in (b), it is impossible for point E, F or G is perpendicular to $e(D, A)$ too. Then by applying the line

3 - 18 of $fork(G)$, (b) can be reduced to (c), and according to the conclusion of Case II, it can be eventually reduced to a single point.

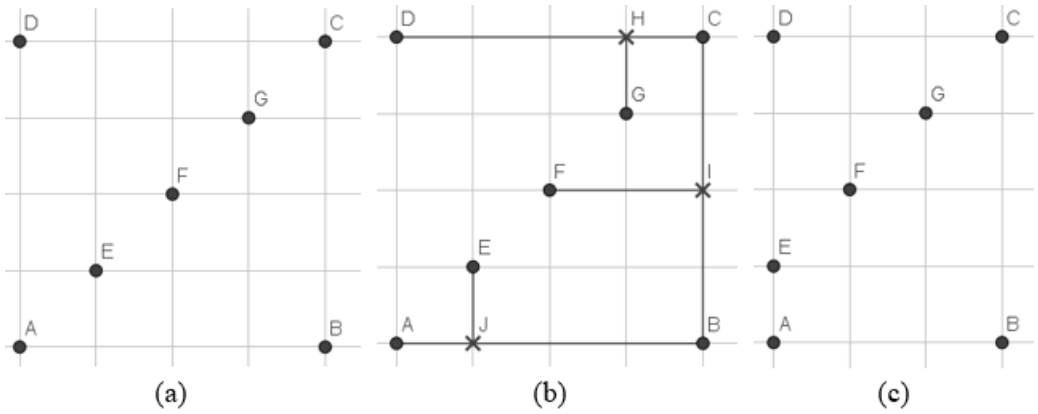

Figure 3.15: Type 5, Case 4

□

Remark 3.1. *In this thesis, equivalent optimal rectilinear Steiner trees are considered as one topology.*

Chapter 4

Heuristic for Rectilinear Steiner Tree Construction

In this chapter, we introduce a new heuristic to construct rectilinear Steiner trees efficiently in $O(n \log n)$ time. In Section 4.1 we investigate approaches to split a large graph and optimize it. In Section 4.2 we describe the heuristic algorithm $RSTC$ for constructing rectilinear Steiner trees. In Section 4.3 we report the experiments and results.

4.1 Partitioning and Optimization

4.1.1 Partitioning

Here, we use $G(\lambda)$ to denote a graph with $n > 7$ terminals and $G(\sigma)$ to denote a graph with $n \leq 7$ terminals, therefore we have

$$G(P) = \begin{cases} G(\sigma)\ where\ |G(P)| \leq 7 \\ G(\lambda)\ where\ |G(P)| > 7 \end{cases}$$

In Chapter 3, we design the algorithm $Const\_optRST$ for generating the rectilinear trees for a given graph with $n \leq 7$ terminals. However, for a graph larger than that, we need to partition it into a set of subgraphs. If a subgraph is not small enough to be a $G(\sigma)$, then we further divide it until each subgraph of the subgraph is a $G(\sigma)$. By recursively dividing a $G(\lambda)$ into a set of $G(\sigma)s$, that is, $G(\lambda) = \{G(\sigma_1), G(\sigma_2), ..., G(\sigma_r)\}$, the $optRSTs$ of all subgraphs can be easily generated by the algorithm $Const\_optRST$.

There are two kinds of partitioning, one is lossless partitioning which divides a $G(\lambda)$ into a set of $G(\sigma)s$ and the $optRST$ of $G(\lambda)$ can be achieved by connecting all the $optRSTs$ of $G(\sigma)s$. The other one is loss partitioning which means when a $G(\lambda)$ is split into a set of $G(\sigma)s$ and the $optRST$ of $G(\lambda)$ is not achievable by connecting all the $optRSTs$ of $G(\sigma)s$.

If a $G(\lambda)$ can be partitioned into two subgraphs $G(\sigma_1)$ and $G(\sigma_2)$ which are not intersected with each other, then $G(\lambda)$ can be optimally constructed by connecting $G(\sigma_1)$ and $G(\sigma_2)$. This means when $G(\sigma_1)$ and $G(\sigma_2)$ are independent of each other, the $optRST$ of $G(\lambda)$ can be achieved by bridging them. However, this does not always happen. For instance, when two subgraphs $G(\sigma_1)$ and $G(\sigma_2)$ are overlapped in x-coordinate, y-coordinate or both, the $optRST$ of $G(\lambda)$ can or cannot be found, depending on different situations. For a graph $G(\lambda)$ which cannot be split into any independent subgraphs, it is hard to know whether the $optRST$ of $G(\lambda)$ can be achievable or not by combining the subtrees of those subgraphs. And more importantly, we cannot predict which cases have such a property. Therefore, only the partitioning which results in independent subgraphs is guaranteed to be lossless.

In this thesis, we partition a $G(\lambda)$ in a way that each subgraph of $G(\lambda)$ has a joint terminal with its neighbor subgraph. When the $optRSTs$ of two subgraphs $G(\sigma_1)$ and $G(\sigma_2)$ are generated, the RST of subgraph $\{G(\sigma_1), G(\sigma_2)\}$ is already formed so that we do not need to bridge the two $optRSTs$ of $G(\sigma_1)$ and $G(\sigma_2)$ into a single subtree. Likewise, when all the subtrees of subgraphs are generated, the entire RST of $G(\lambda)$ is also constructed.

Here, we propose three partitioning ways:

1. Sequential partitioning

A simple way to divide a $G(\lambda)$ is to line up all the terminals of $G(\lambda)$ according to their ascending/descending x-coordinates or y-coordinates, and then divide it into a set of $G(\sigma)$ sequentially. For instance, Figure 4.1 shows that $G(\lambda) = \{(1,5),(5,1),(6,3),(3,2),(2,4),(4,6),(7,7)\}$ is sorted by their x-coordinates and then becomes $G(\lambda) = \{(1,5),(2,4),(3,2),(4,6),(5,1),(6,3),(7,7)\}$. In this case, $G(\lambda)$ is partitioned as $G(\lambda) = \{G(\sigma_1), G(\sigma_2)\}$, $G(\sigma_1) = \{(1,5),(2,4),(3,2),(4,6),(5,1)\}$ and $G(\sigma_2) = \{(5,1),(6,3),(7,7)\}$. And then we generate the $optRSTs$ of $G(\sigma_1)$ and $G(\sigma_2)$ respectively by the algorithm $Const\_optRST$. Finally, the rectilinear Steiner tree of entire $G(\lambda)$ is also constructed.

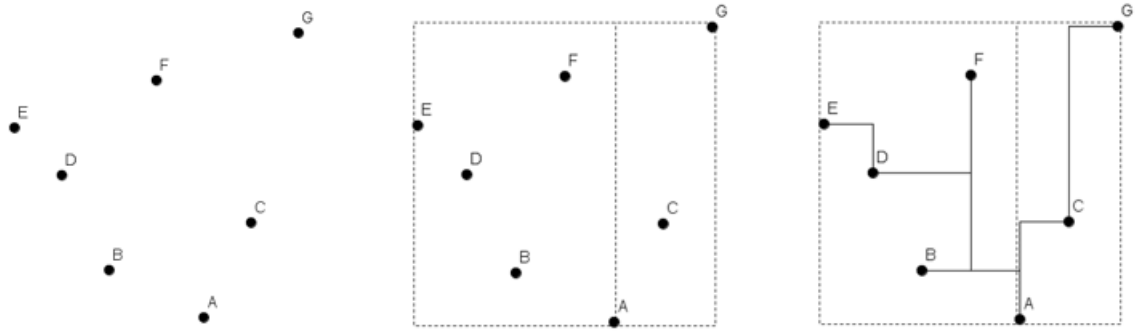

Figure 4.1: Sequential partitioning

However, this can only happen when splitting along with one direction - horizontal or vertical. By sorting all the terminals horizontally/vertically, we can linearly divide a $G(\lambda)$ into a set of subgraphs ($G(\lambda) = \{G(\sigma_1), G(\sigma_2), ..., G(\sigma_r)\}$), starting from left/right side to right/left side or from top/bottom side to bottom/top side (Figure 4.2). Therefore, there is no dividing inside a subgraph, like Figure 4.3 shows.

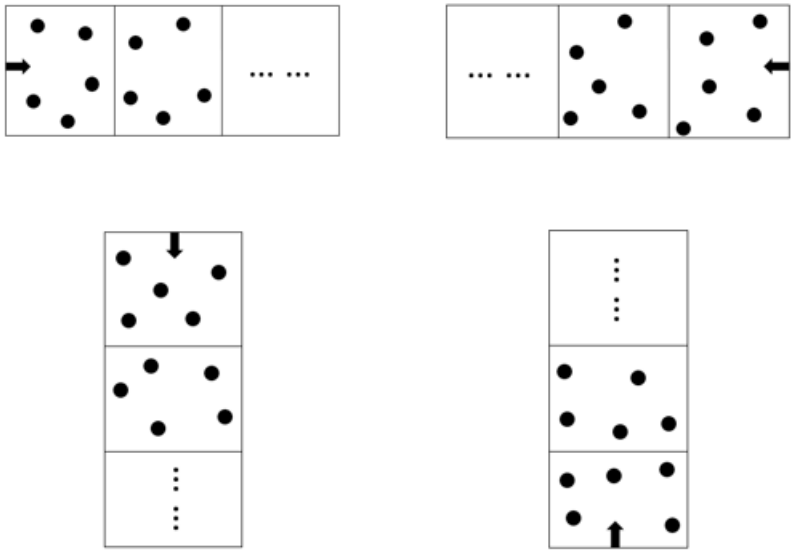

Figure 4.2: Sequential partitioning schemes

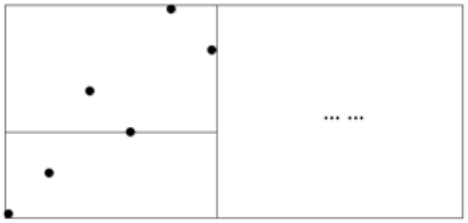

Figure 4.3: Sequential partitioning inside a subgraph

2. Median partitioning

Another partitioning way is to divide the graph $G(\lambda)$ up into a set of subgraphs, each containing at most 7 terminals. For the dividing, we always select the median terminal for the splitting point, and this applies to each subgraph too unless it is already a $G(\sigma)$. Therefore, by recursively dividing subgraphs until each is small enough to fit into a $G(\sigma)$ of which $optRST$ can be directly generated. And then the $optRSTs$ of subgraphs are connected as a rectilinear Steiner tree of $G(\lambda)$. Suppose we divide a $G(\lambda)$ into subgraphs $G(\sigma_1)$ and $G(\sigma_2)$ as Figure 4.4 shows, first of all, we need to find the median terminal according to their x-coordinates or y-coordinates, and then we further divide $G(\sigma_1)$ up into $G(\sigma_{11})$ and $G(\sigma_{12})$, and $G(\sigma_2)$ up into $G(\sigma_{21})$ and $G(\sigma_{22})$ so that each resulting subgraph ($G(\sigma_{11})$/$G(\sigma_{12})$/$G(\sigma_{21})$/$G(\sigma_{22})$) is a $G(\sigma)$ of which $optRST$ can be generated by the algorithm $Const\_optRST$.

The differences between this partitioning and sequential partitioning are that firstly

the former allows you to divide a subgraph in a different direction. For instance, Figure 4.5 shows that a $G(\lambda)$ is divided into $G(\sigma_1)$ and $G(\sigma_2)$ horizontally, and then $G(\sigma_1)$ is further divided into $G(\sigma_{11})$ and $G(\sigma_{12})$ vertically. Secondly, any subgraph may be split into a $G(\sigma)$ with $1 \leq |G(\sigma)| \leq 7$, not necessarily to be $|G(\sigma)| = 7$, and this brings more flexibility to the dividing.

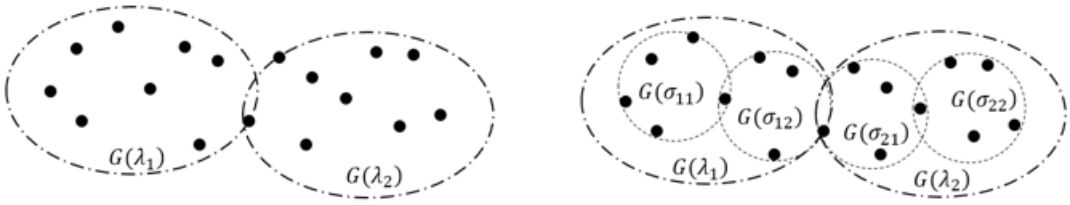

Figure 4.4: Median partitioning

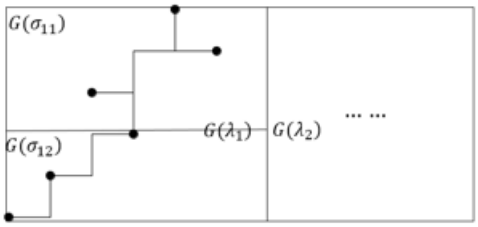

Figure 4.5: Median partitioning with different directions

However, the dividing points cannot be arbitrarily chosen, because they are always the median terminals of the subgraphs. In addition, for most cases, we cannot fully utilize the function of algorithm $Const\_optRST$. For example, only when the subgraph has 13 terminals it will be divided into two subgraphs with $|G(\sigma)| = 7$, otherwise the generating function of 7 terminals will never be employed.

3. Group partitioning

Here we introduce a partitioning way to divide a $G(\lambda)$ into a set of groups. We firstly recursively partition a $G(\lambda)$ into a set of $G(\omega)$ ($|G(\sigma)| < |G(\omega)| < 2|G(\sigma)|$). And then we further split each $G(\omega)$ into two subgraphs $G(\sigma_1)$ and $G(\sigma_2)$.

Suppose $G(\omega) = \{B, F, A, D, G, C, E\}$, as Figure 4.6 shows, we partition $G(\omega)$ into $G(\sigma_1) = \{D, E, F, G\}$ and $G(\sigma_2) = \{A, B, C, D\}$. It can be obviously observed that $G(\sigma_1) \cap G(\sigma_2) = \{D\}$. And then we generate the $optRSTs$ of $G(\sigma_1)$ and $G(\sigma_2)$ respectively. Finally, the rectilinear Steiner tree of $G(\omega)$ is already constructed by the connecting

function of terminal D. As we can see, in the beginning of partitioning, an appropriate ω is needed to be given. ω is a parameter and we confine it to be ranging between σ to 2σ , eliminating the situation when $|\omega| = |\sigma|$ and $\omega = 2|\sigma|$, and different ω yields a different accuracy of constructing rectilinear Steiner tree.

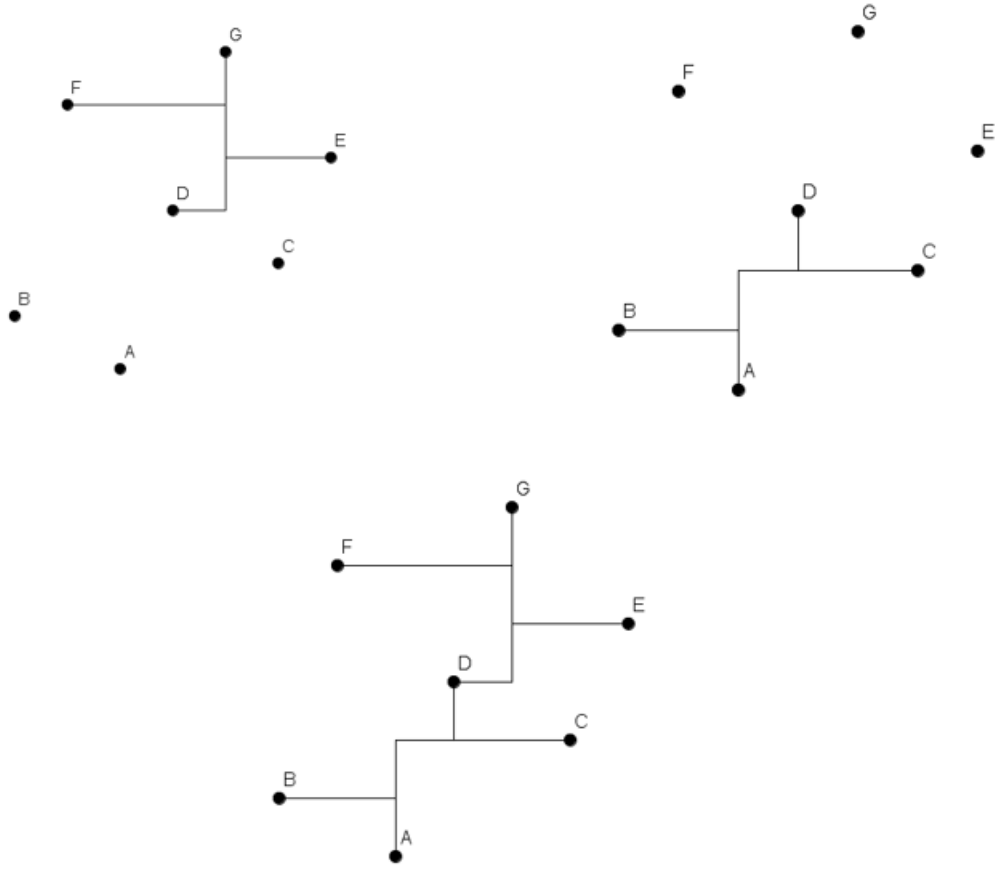

Figure 4.6: Group partitioning

For instance, for splitting $G(\omega) = \{D, A, F, B, E, C\}$ as Figure 4.7 shows, suppose it is divided into $G(\sigma_1) = \{A, B, C\}$, $G(\sigma_2) = \{D, F, E, C\}$, and $G(\sigma_1) \cap G(\sigma_2) = \{C\}$. As Figure 4.7 shows, two optimal subtrees $optRST_1$ and $optRST_2$ are generated. And then a suboptimal RST of $G(\omega)$ is formed by the common point C.

However, the RST in Figure 4.7(a) is not an optimal RST for $G(\omega)$. In fact, we have another way to build an RST for $G(\omega)$. By splitting $G(\omega)$ into $G(\sigma_1) = \{A, D, B, C\}$, $G(\sigma_2) = \{D, F, E\}$, and $G(\sigma_1) \cap G(\sigma_2) = \{D\}$, we have two optimal subtrees $optRST_1'$ and $optRST_2'$ generated, and finally an RST' for $G(\omega)$ is also built by the connecting terminal D.

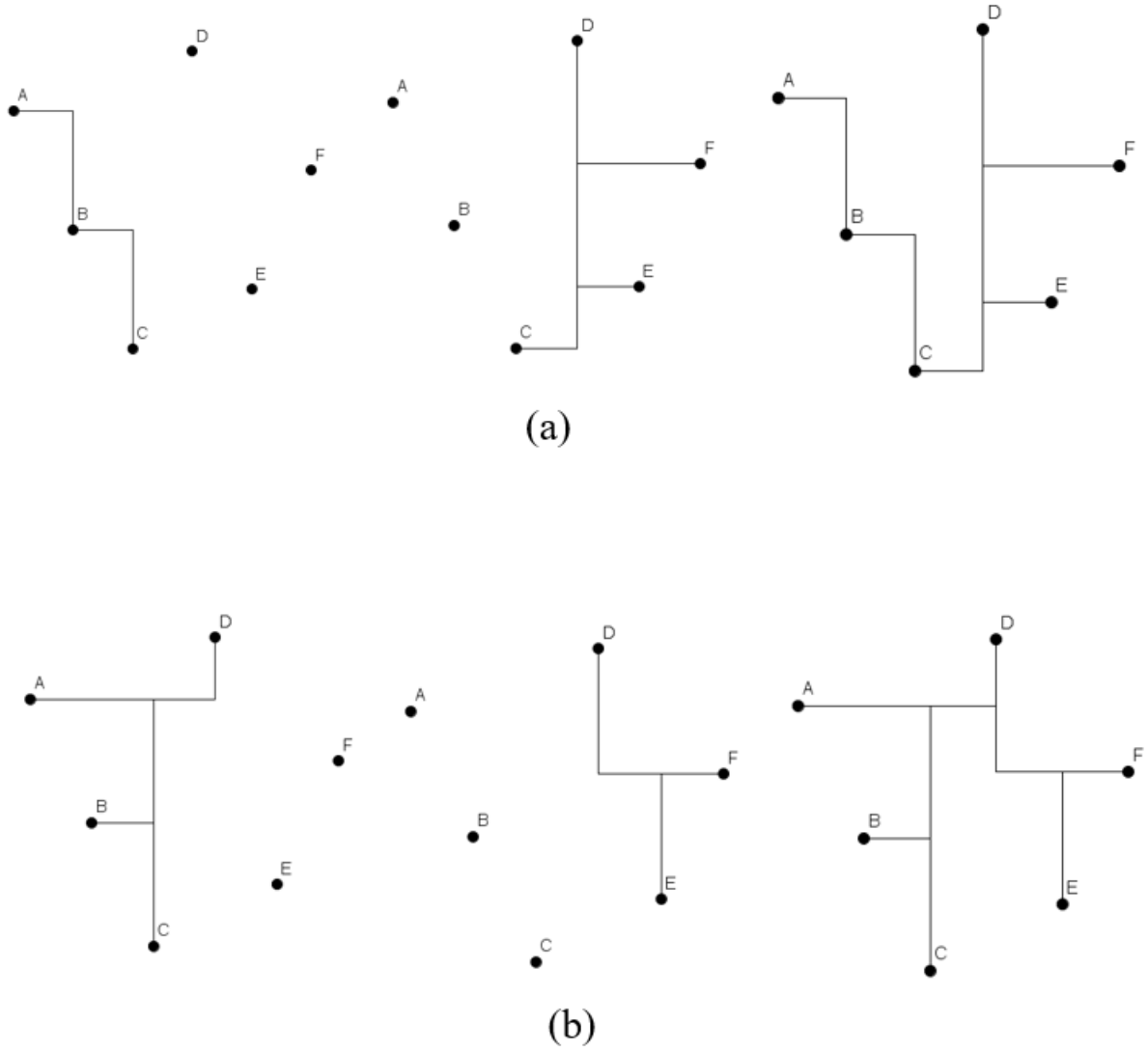

Figure 4.7: (a) Selection of partitioning point; and (b) Alternative selection of partitioning point.

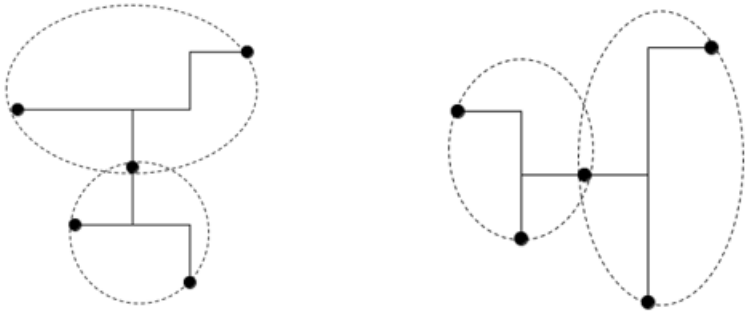

Figure 4.8: Group partitioning with different directions

Figure 4.7(b) illustrates the process above. At this point, we surprisingly find that RST' is better than RST, because RST' is optimal but RST is not. Therefore, the

accuracy of computing an RST for a $G(\omega)$ is significantly related to the common terminal of two subgraphs $G(\sigma_1)$ and $G(\sigma_2)$ we select. However, the perfect junction point of two subtrees is not predictable. Hence we need to try different combinations. For instance, for a $G(\omega)$ with $|G(\omega)| = 10$, there are five ways to split: (1) $|G(\sigma_1)| = 7$, $|G(\sigma_2)| = 3$; (2) $|G(\sigma_1)| = 6$, $|G(\sigma_2)| = 4$; (3) $|G(\sigma_1)| = 5$, $|G(\sigma_2)| = 5$; (4) $|G(\sigma_1)| = 4$, $|G(\sigma_2)| = 6$; and (5) $|G(\sigma_1)| = 3$, $|G(\sigma_2)| = 7$. Each way may yield a different accuracy of constructing a subtree of $G(\omega)$, and we need to try all of them to decide an appropriate separating point to split the $G(\omega)$.

For each step of partitioning a $G(\lambda)$ or $G(\omega)$, it can be implemented by horizontally or vertically. Therefore, for a particular graph, partitioning at a different direction may yield a different rectilinear Steiner tree. Figure 4.8 shows that two different $RSTs$ are generated by partitioning horizontally and vertically.

However enumerating all the possible combinations will significantly increase the runtime, therefore a reduced way is sought. For example, let $|G(\omega)| = 10$, $|G(\sigma_1)| = 7$ and $|G(\sigma_2)| = |G(\omega)| - |G(\sigma_1)|$, and horizontally divide a $G(\lambda)$ and vertically divide a $G(\omega)$.

4.1.2 Optimization

For a subgraph, when two or more subtrees are generated, the RST of the entire subgraph is also constructed, because any two subtrees share a common terminal between them. However, this may bring the extra cost of the total length for merging two subtrees together.

Suppose $G(\lambda) = \{G, F, D, E, C, B, A\}$, we split $G(\lambda)$ into $G(\sigma_1) = \{D, C, A\}$, $G(\sigma_2) = \{G, F, E, B, A\}$, and $G(\sigma_1) \cap G(\sigma_2) = \{A\}$. In Figure 4.9, (a) and (b) show that two optRSTs are generated and then an RST is constructed in (c). As we can see, there appear redundant edges in the RST. If we flip edge $e(A, C)$ and edge $e(A, B)$, a new RST' can be formed. Finally, as (d) indicates, segment j and segment p are overlapped. By removing the segment j, a better RST with less total length can be achieved in (e).

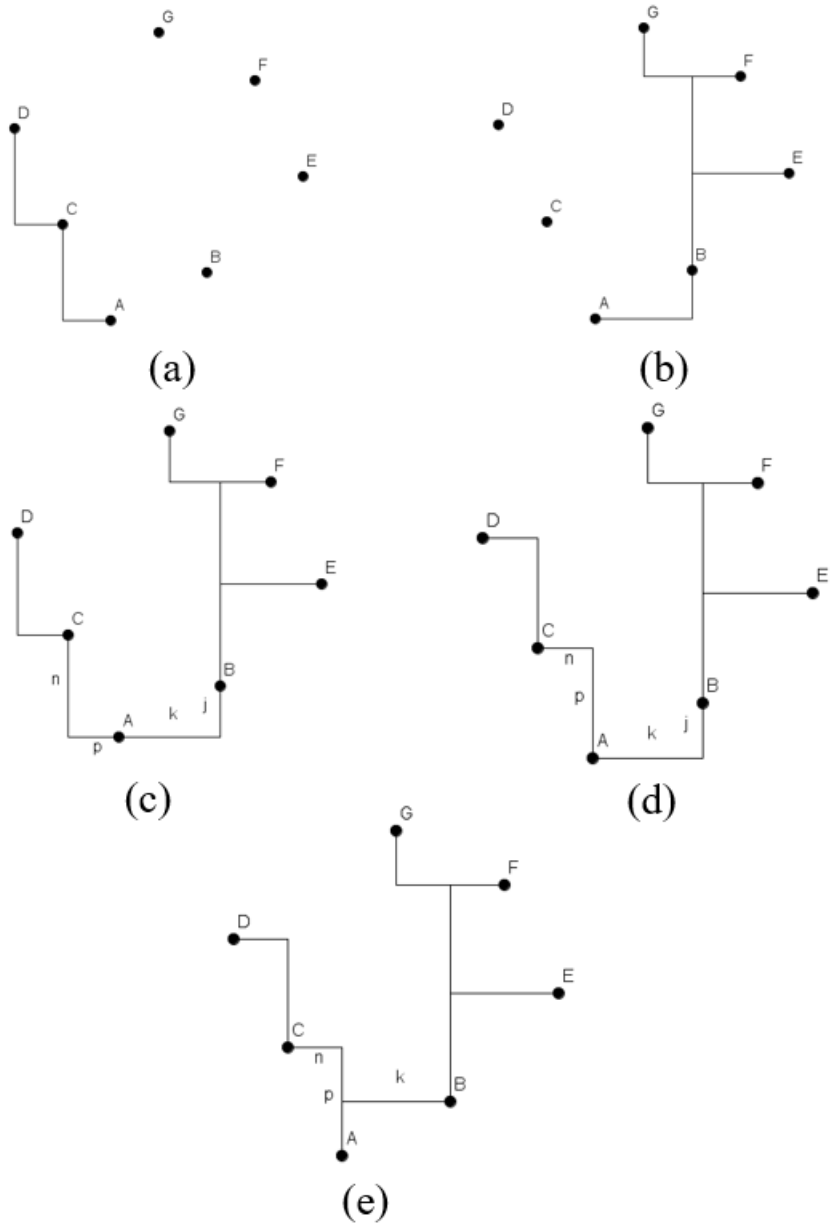

Figure 4.9: Optimization of total length

4.2 Heuristic Algorithm

In this section, we propose an efficient heuristic for constructing any given n terminals in the plane. We first map n terminals into a permutation, and then sequentially partition the permutation into a set of districts. By generating the optimal subtree of each district, and a suboptimal rectilinear Steiner tree can be finally constructed since every pair of subtree have been connected per se.

First of all, the mapping of n terminals to a permutation is all about sorting n terminals

according to their x-coordinates or y-coordinates, and it can be done by typical sorting algorithms in $(n \log n)$ time.

Algorithm 4.1: $RSTC$

Input : Given n terminals in the plane

Output: The suboptimal MRST

1 sort n terminals and generate the corresponding permutation
2 transform the permutation to a $G(P)$ with a set of permutation points
3 divide $G(P)$ into a set of subgraphs and generate permutation for each subgraph
4 retrieve the $optRST$ structures of each permutation
5 compute the $RSTs$ of all permutations
6 return the minimal RST and its structure

In line 3, we first divide $G(P)$ into a set of subgraphs which contain no more than 7 points. Suppose the permutation of $G(P)$ is $P = \{7, 4, 6, 2, 5, 1, 3, 8\}$ as shown in (a), and assume the size of each subgraph is 3, then P is partitioned into 4 districts as (b) illustrates. And then by counting-sort, the four districts of subgraphs is sorted as shown in (c). Finally, by classifying each district, we can attain each permutation of each district. For instance, the permutation of district 3 is 312. Therefore, permutations of all subgraphs are generated. The time and space complexity of line 3 are both $O(n)$.

In the algorithm $RSTC$, we compute the minimal length and return the RST construction of $G(P)$. We first load all the topology structure of $optRSTs$ of $G(P)$ with $|G(P)| \leq 7$ into memory with trivial time compared to the entire runtime of $RSTC$. After sorting the terminals and attaining the permutation, we transform the permutation to a $G(P)$ with a set of permutation points and then compute the $optRSTs$ of $G(P)$, and finally, accumulate all the $RSTs$ together to achieve the total minimal length and the topology of $G(P)$. The entire complexity depends on line 1 because line 2 line 6 are all in $O(n)$. Since line 1 is in $O(n \log n)$ time, therefore the complexity of algorithm $RSTC$ is $O(n \log n)$ with $O(n)$ space.

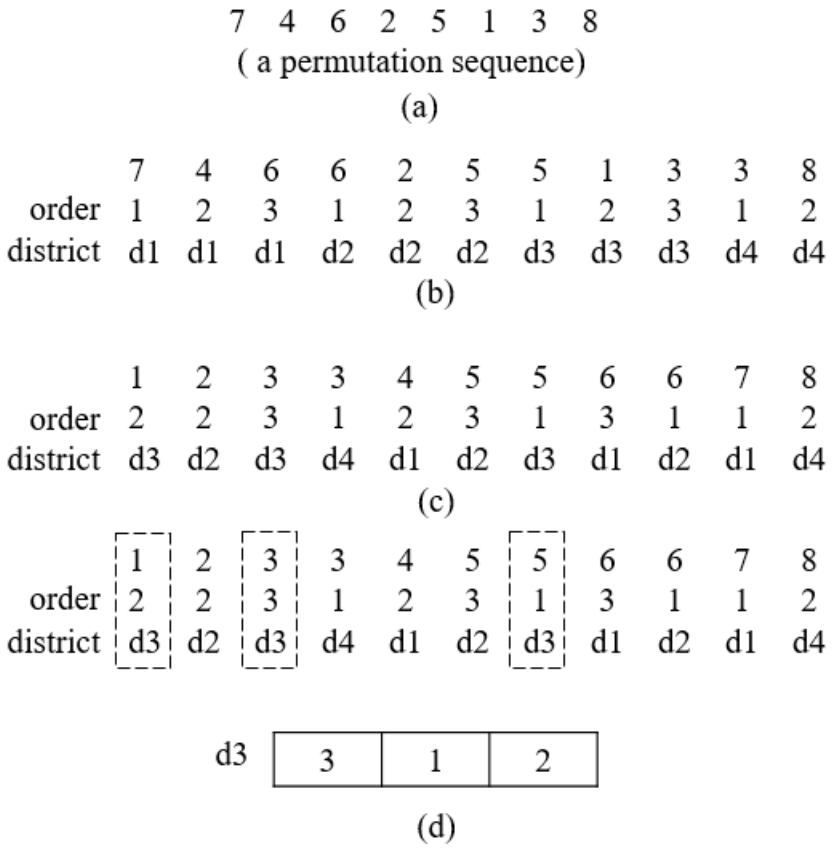

Figure 4.10: Permutation of each district

4.3 Results and Discussion

In this section, we describe the experimental setup, evaluate and compare the performance of RSTC with RMST [32] which is a very efficient Prim's algorithm for computing minimal rectilinear spanning trees. We report and compare performance in terms of CPU runtime, accuracy in minimal length, and the Steiner ratio. The experiments we conducted are based on sequential partitioning which is elaborated in Section 4.1. We denote Per# to be the number of permutations, $\rho = \frac{total\,length\,of\,RMST}{total\,length\,of\,RSTC}$ to be the Steiner ratio, $\chi = \frac{total\,length\,of\,RMST - total\,length\,of\,RSTC}{total\,length\,of\,RMST}$ to be the improvement of total length compared to RMST, $\eta = \frac{runtime\,of\,RMST}{runtime\,of\,RSTC} - 1$ to be the CPU-time comparison between RMST and RSTC, Ω to be the number of permutation points in each partitioned district, and Δ to be the number of partitioning for a large graph divided by subgraphs.

We do not generate random points for the input of RSTC algorithm, but enumerate

permutation points instead, because different groups of random points yield different total lengths in each time. And all experiments were conducted on an Intel(R) Core(TM) i5-3330 CPU @ 3.00GHz processor with 4GB memory. In addition, we use low-level Unix interval timers to perform timing for CPU runtime.

Table 4.1: Comparison of total length for $8 \sim 10$ terminals ($\Delta = 2$)

Terminal	**Per#**	**RMST**	**RSTC**	ρ	$\chi(\%)$
8	40,320	924110	833280	1.10	9.83 %
9	362,880	9915402	8902656	1.11	10.21 %
10	3,628,800	115908944	104068992	1.11	10.21 %

Table 4.2: Comparison of runtime for $8 \sim 10$ terminals ($\Delta = 2$)

Terminal	**Per#**	**RMST**	**RSTC**	η
8	40,320	$4.31s$	$0.96s$	3.45
9	362,880	$39.31s$	$9.27s$	3.23
10	3,628,800	$385.37s$	$115.39s$	2.33

In Table 4.1, we can observe that with $\Delta = 2$, the Steiner ratio ρ of $8 \sim 10$ terminals varies from 1.1090 to 1.1138, and the improvement of total length χ varies from 9.83 % to 10.21 %. In Table 4.2, we can observe that with $\Delta = 2$, RSTC is 3.45, 3.23 and 2.33 times faster than RMST for 8, 9 and 10 terminals respectively.

Table 4.3: Comparison of total length for $15 \sim 100$ terminals (Per# $= 10,000$, $\Omega = 6$)

Terminal	**Per#**	**RMST**	**RSTC**	ρ	$\chi(\%)$
15	10,000	3983363	3708724	1.07	6.89 %
20	10,000	4983363	4708724	1.05	5.51 %
25	10,000	5983363	5708724	1.04	4.59 %
30	10,000	6983363	6708724	1.04	3.93 %
35	10,000	7983363	7708724	1.03	3.44 %
40	10,000	8983363	8708724	1.03	3.06 %
45	10,000	9983363	9708724	1.02	2.75 %
50	10,000	10983363	10708724	1.02	2.50 %
100	10,000	20983363	20708724	1.01	1.31 %

In Table 4.3, we randomly selected 10,000 permutations for conducting the experiment. We can observe that with $\Omega = 6$, the Steiner ratio of $15 \sim 100$ terminals varies from 1.01 to 1.07, and the improvement from 1.31 % to 6.89 %. In Table 4.4, we can observe that with $\Omega = 6$, RSTC is 2.58, 2.00, 1.91, 1.74, 1.59, 1.50, 1.41, 1.38 and 1.06 times faster than RMST for 15, 20, 25, 30, 35, 40, 45, 50 and 100 terminals respectively.

Table 4.4: Comparison of runtime for $15 \sim 100$ terminals (Per# $= 10,000$, $\Omega = 6$)

Terminal	**Per#**	**RMST**	**RSTC**	η
15	10,000	11.89s	3.31s	2.58
20	10,000	11.81s	3.92s	2.00
25	10,000	13.34s	4.58s	1.91
30	10,000	14.17s	5.16s	1.74
35	10,000	14.87s	5.72s	1.59
40	10,000	15.85s	6.33s	1.50
45	10,000	16.61s	6.87s	1.41
50	10,000	17.18s	7.21s	1.38
100	10,000	25.48s	12.36s	1.06

From the experiment results, we can clearly see that RSTC is excellent both in the accuracy of total length and CPU runtime. Compared to RMST, the improvement of total length varies from 3.93 $\%$ to 10.21 $\%$ and it is 1.74 to 3.45 times faster for the size of 30 terminals. In addition, the improvement of total length is 1.31 $\%$ and 1.06 times faster for the size of 100 terminals.

Chapter 5

Conclusion and Future Work

In this chapter, we conclude the thesis and summarize our future directions for research.

5.1 Conclusion

This thesis introduces a new heuristic to efficiently and accurately construct minimum rectilinear Steiner trees which are the shortest interconnections of a set of points in the plane. Previous research work about this topic shows that exact solutions for this problem only exist in exponential time complexity, and approximation solutions with good accuracy also have a long running time. Therefore this thesis seeks a new way to address this problem with good performance both in worst-case running time and accuracy. To achieve this goal, we propose an exact solution for a few points. We first split a plane into a set of sub-planes which only contain a few points, and then employ the exact algorithm to grow optimal rectilinear Steiner trees for all the sub-planes. By connecting the optimal rectilinear Steiner tree of each sub-plane, we finally achieve an approximated minimum rectilinear Steiner tree.

To reduce the runtime, we compute all the topologies of rectilinear Steiner trees in which $n \leq 7$ points into a data file and load it into memory in hashing fashion. Therefore for constructing the rectilinear Steiner tree of a given plane, we only need to retrieve its optimal rectilinear Steiner trees of all sub-planes, and then multiply the real distances by the topology units. We conducted our experiments which show that the heuristic demon-

strates a good performance both in runtime and accuracy for up to 100 terminals.

5.2 Future Work

Below we identify two directions for further research:

1. The optimal algorithm $const\_optRST$ addresses the rectilinear Steiner problem with up to 7 points, and it may be expanded to 12 points to construct rectilinear Steiner trees for larger instances.

2. More sophisticated partitioning approaches are needed for the heuristic algorithm $RSTC$. With better ways for dividing a plane, more accuracy in total length can be achieved. However, it may also bring overhead in time complexity.

Bibliography

[1] Andrew Chi-Chih Yao, *On constructing minimum spanning trees in k-dimensional spaces and related problems*, SIAM Journal on Computing, 11(4):721-736, November 1982.

[2] Hua Lo-Keng, *Application of mathematical methods to wheat harvesting*, Chinese Math., pp. 77-91, 1962.

[3] R. Courant and H. Robbins, *What is Mathematics?*, London, New York and Toronto, Oxford University Press, 1941.

[4] M. Garey and D. S. Johnson, *The rectilinear Steiner problem is NP-complete*, SIAM J. Applied Math., 32(4):826834, 1977.

[5] M. Hanan, *On Steiners problem with rectilinear distance*, SIAM J. Applied Math., 14:255265, 1966.

[6] F. K. Hwang, *On Steiner minimal trees with rectilinear distance*, SIAM J. Applied Math., 30(1): 10414, 1976.

[7] F. K. Hwang, D. S. Richards, and P. Winter, *The Steiner tree problem*, Annals of Discrete Mathematics, 1992.

[8] E. N. Gilbert and H. O. Pollak, *Steiner minimal trees*, SIAM J. Applied Math., 16:129, 1968.

[9] Z. A. Melzak, *On the problem of Steiner*, Canad. Math. Bull., pp. 143-148, 1961.

[10] J. B. Kruskal, *On the shortest spanning subtree of a graph*, Proc. Amer. Math. Soc., pp. 48, 1956.

[11] R. C. Prim, *Shortest connecting networks*, Bell System Tech. J., pp. 1398- 1401, 1952.

[12] F. P. Palermo, *A network minimization problem*, IBM J. Res. Develop., pp. 335-337, 1961.

[13] Joseph L Ganley, *Computing optimal rectilinear Steiner trees: A survey and experimental evaluation*, Discrete Applied Mathematics, Pages 161-171, Volume 90, Issues 13, 15 January 1999.

[14] M. Borah, R. M. Owens, and M. J. Irwin, *A fast and simple Steiner routing heuristic*, Discrete and Applied Mathematics, 90(1-3):5167, 1999.

[15] U. Fomeier, M. Kaufmann, and A. Zelikovsky, *Faster approximation algorithms for the rectilinear Steiner tree problem*, Discrete and Computational Geometry, 18:93109, 1997.

[16] G. Georgakopoulos and C. H. Papadimitriou, *The 1-Steiner tree problem*, J. Algorithms, 8:122130, 1987.

[17] Y.Y. Yang and O. Wing, *Optimal and suboptimal solution algorithms for the wiring problem*, In Proceedings of the International Symposium on Circuit Theory, pages 154-158, 1972.

[18] G. Robins and J. S. Salowe, *Low-degree minimum spanning trees*, Discrete and Computational Geometry, 14:151165, September 1995.

[19] A. Z. Zelikovsky, *An 11/6 approximation algorithm for the network Steiner problem*, Algorithmica, 9:463470, 1993.

[20] P. Berman, U. Fomeier, M. Karpinski, M. Kaufmann, and A. Z. Zelikovsky, *Approaching the 5/4 - approximation for rectilinear Steiner trees*, In Proc. European Symposium on Algorithms, pages 533542, 1994.

[21] A.F. Sidorenko, *On minimal rectilinear Steiner trees*, Diskretnaya Matematika, 1:28-37, 1989.

[22] M. Karpinski and A. Zelikovsky, *New approximation algorithms for the Steiner tree problems*, Journal of Combinatorial Optimization, 1(1):4765, March 1997.

[23] L. Kou, G. Markowsky, and L. Berman, *A fast algorithm for Steiner trees*, Acta Informatica, 15:141145, 1981.

[24] G. Robins and A. Zelikovsky, *Improved Steiner tree approximation in graphs*, In Proc. ACM/SIAM Symp. Discrete Algorithms, pages 770779, San Francisco, CA, January 2000.

[25] M. Zachariasen, *Rectilinear full Steiner tree generation*, Networks, vol. 33, issue 2, pages 125143, March 1999.

[26] G. Robins and A. Zelikovsky, *Minimum Steiner Tree Construction*, in The Handbook of Algorithms for VLSI Physical Design Automation, CRC Press, 2009, Chapter 24, pp. 487-508.

[27] David Warme, Pawel Winter, and Martin Zachariasen, *GeoSteiner - software for computing Steiner trees*, http://www.diku.dk/geoSteiner/.

[28] Hai Zhou, *Efficient Steiner tree construction based on spanning graphs*, In Proc. Intl. Symp. On Physical Design, pages 152157, 2003.

[29] C.D. Thomborson, B. Alpern, and L.Carter, *Rectilinear Steiner tree minimization on a workstation*, Discrete Mathematics and Theoretical Computer Science, pages 119-136, 1994.

[30] A. B. Kahng and G. Robins, *A new class of iterative Steiner tree heuristics with good performance*, IEEE Transactions Computer-Aided Design, 11(7):893902, July 1992.

[31] M. Borah, R. M. Owens, and M. J. Irwin, *An edge-based heuristic for Steiner routing*, IEEE Transactions Computer-Aided Design, 13:15631568, 1994.

[32] Andrew B. Kahng and Ion Mandoiu, *RMST - Pack: Rectilinear minimum spanning tree algorithms*, http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/RSMT/RMST/.

[33] A.E.F. Clementi and L. Trevisan, *Improved non-approximability results for minimum vertex cover with density constraints*, Theor. Comput. Sci., 225:113-128, 1999.

[34] J.L. Ganley and J.P. Cohoon, *A faster dynamic programming algorithm for exact rectilinear Steiner minimal trees*, In proceedings of the Fourth Great Lakes Symposium on VLSI, pages 238-241, 1994.

[35] P. K. Agarwal and M. T. Shing, *Algorithms for special cases of rectilinear Steiner trees: Points on the boundary of a rectilinear rectangle*, Networks, 20(4):45385, 1990.

[36] T. H. Chao and Y. C. Hsu, *Rectilinear Steiner tree construction by local and global refinement*, IEEE Transactions Computer-Aided Design, 13(3):303309, March 1994.

[37] Chris Chu, *FLUTE: Fast lookup table based wirelength estimation technique*, In Proc. IEEE/ACM Intl. Conf. on Computer-Aided Design, pages 696701, 2004.

[38] J. Griffith, G. Robins, J. S. Salowe, and T. Zhang, *Closing the gap: Near-optimal Steiner trees in polynomial time*, IEEE Transactions Computer-Aided Design, 13(11):13511365, November 1994.

[39] N. Hasan, G. Vijayan, and C. K. Wong, *A neighborhood improvement algorithm for rectilinear Steiner trees*, In Proc. IEEE International Symp. Circuits and Systems, New Orleans, LA, 1990.

[40] J. M. Ho, G. Vijayan, and C. K. Wong, *New algorithms for the rectilinear Steiner tree problem*, IEEE Transactions Computer-Aided Design, 9(2):185193, 1990.

[41] A. O. Ivanov and A. A. Tuzhilin, *Minimal Networks: The Steiner Problem and Its Generalizations*, CRC Press, Boca Raton, Florida, 1994.

[42] A. B. Kahng and G. Robins, *A new family of Steiner tree heuristics with good performance: The iterated 1-Steiner approach*, In Proc. IEEE International Conf. Computer-Aided Design, pages 428431, Santa Clara, CA, November 1990.

[43] A. B. Kahng and G. Robins, *On performance bounds for a class of rectilinear Steiner tree heuristics in arbitrary dimension*, IEEE Transactions Computer-Aided Design, 11(11):14621465, November 1992.

[44] I. I. Mandoiu, V. V. Vazirani, and J. L. Ganley, *A new heuristic for rectilinear Steiner trees*, IEEE Transactions Computer-Aided Design, 19:11291139, October 2000.

[45] J. S. Salowe and D. M. Warme, *An exact rectilinear Steiner tree algorithm*, In Proc. IEEE International Conf. Computer Design, pages 472475, Cambridge, MA, October 1993.

[46] A. Z. Zelikovsky, *An 11/8-approximation algorithm for the Steiner problem on networks with rectilinear distance*, In Janos Bolyai Mathematica Societatis Conf.: Sets, Graphs, and Numbers, pages 733745, January 1992.

[47] A. Z. Zelikovsky, *A faster approximation algorithm for the Steiner tree problem in graphs*, Information Processing Letters, 46(2):7983, May 1993.

[48] J.S. Salowe and D.M. Warme, *An exact rectilinear Steiner tree algorithm*, In Proceedings of the International Conference on Computer Design, pages 472-475, 1993.

[49] F. K. Hwang, *An O (n log n) Algorithm for Rectilinear Minimal Spanning Trees*, Journal of the ACM (JACM) JACM, vol. 26, issue 2, pages 177-182, April 1979.

[50] Gabriel Robins and Alexander Zelikovsky, *Tighter Bounds for Graph Steiner Tree Approximation*, SIAM Journal on Discrete Mathematics, vol. 19, issue 1, pages 122 134, 2005.

[51] P. Berman and V. Ramaiyer, *Improved approximations for the Steiner tree problem*, In Proc. ACM/SIAM Symp. Discrete Algorithms, pages 325334, San Francisco, CA, January 1992.

[52] Great Lakes Symp, *Polynomial time approximation scheme for the rectilinear Steiner arborescence problem*, Journal of Combinatorial Optimization, 4(3):357363, September 2000.

[53] S. Hougardy and H. J. Promel, *A 1.598 approximation algorithm for the Steiner problem in graphs*, In Proc. ACM/SIAM Symp. Discrete Algorithms, pages 448453, January 1999.

[54] S. Arora, *Polynomial time approximation schemes for Euclidean TSP and other geometric problems*, J. ACM, pp. 753782, 1998.