國立臺灣大學電機資訊學院電子工程學研究所

博士論文

Graduate Institute of Electronics Engineering

College of Electrical Engineering & Computer Science

National Taiwan University

Doctoral Dissertation

超大型積體電路實體設計中 考慮障礙物之直角史坦納樹建構

Obstacle-Avoiding Rectilinear Steiner Tree Construction in VLSI Physical Design

劉智弘

Chih-Hung Liu

指導教授：郭斯彥 博士

Advisor: Sy-Yen Kuo, Ph.D.

中華民國 98 年 7 月

July, 2009

國立臺灣大學博士學位論文
口試委員會審定書

超大型積體電路實體設計中
考慮障礙物之直角史坦納樹建構
Obstacle-Avoiding Rectilinear Steiner Tree Construction in VLSI Physical Design

本論文係劉智弘君（F94943076）在國立臺灣大學電子工程學研究所完成之博士學位論文，於民國 98 年 6 月 2 日承下列考試委員審查通過及口試及格，特此證明

口試委員：

郭斯彥
（指導教授）

[illegible]　　袁世一

王勝德　　[illegible]

[illegible]　　雷欽隆

[illegible]　　蔡一鳴

系主任、所長 [illegible]

Obstacle-Avoiding Rectilinear Steiner Tree Construction in VLSI Physical Design

By

Chih-Hung Liu

Dissertation

Submitted in partial fulfillment of the requirement
for the degree of Doctor of Philosophy
in Electronics Engineering
at National Taiwan University
Taipei, Taiwan, R.O.C.

June 2009

Approved by：

[illegible]	[illegible]	Shih-Yi Yuan
[illegible]	[illegible]	[illegible]
[illegible]	Shyue-Kung Lu	[illegible]

Advised by： [illegible]

Approved by Director： [illegible]

To my parents, Li-Mei Hsu and Shih-Jen Liu
and my sister, Chia-Yen Liu.

Acknowledgement

I would like to express my gratitude to my advisor, Prof. Sy-Yen Kuo, for his tremendous support, encouragement, and guidance throughout my graduate study. On the whole, I learned what a researcher should be and how to be from him. Without his guidance and assistance, I would not have completed the dissertation successfully.

I am very much indebted to Prof. Shih-Yi Yuan, Prof. Yao-Wen Chang, and Prof. Der-Tsai Lee. Prof. Shin-Yi Yuan participated in all my research works, and always gave me reasonable suggestions. Prof. Yao-Wen Chang taught me the fundamental knowledge in Physical Design and his precious experiences. Prof. Der-Tsai Lee brought me basic techniques in Computational Geometry, and led me to the essence of Algorithms.

I greatly appreciate the members of my dissertation committee, Prof. Chin-Laung Lei, Prof. Hsu-Chun Yen, Prof. Sheng-De Wang, Prof. Kuo-Chen Wang, Prof. Shyue-Kung Lu, Prof. Shih-Yi Yuan, Prof. I-Ming Tsai, and Prof. Chin-Chou Chen, for invaluable comments and suggestions, and for their kind assistance in many occasions.

Thanks also to wonderful friends who made my life during my doctoral period so pleasant and unforgettable. Especially, I would like to express my appreciation to my college senior classmates, Shu-Min Yang, Yi-Sheng Fu, and Yen-Nien Wu, my college classmates, Min-Hsieh Tsai, Yung-Huan Hsieh, Chin-Hsiung Hsu, Yen-Ting Liu, Chen-Kai Hsieh, Wei-Shun Chuang, and Gwo-Shu Huang, my high school classmate, Hsien-Hsiang Chiu. They always encouraged me when I was disappointed, and gave me the most honest suggestions when I was confused or had difficulties. Without their warm-hearted help, I would not have overcome all the obstacles I met to receive my Ph.D. degree.

I acknowledge all members of the Dependable Distributed Systems and Networks Laboratory. I especially appreciate Szu-Chi Wang and Yao-Hsin Chou for their collaboration, Chia-Ling Ma, Yen-Ting Liu, Chun-Yen Kuo, Chi-Yuan Chen, Chia-Mu Yu, Hong-Wei Huang, Ping-Hsun Hsieh, and Wei-Ting Tu for the beneficial discussions, and Kuan-Lin Chen, Che-Yang Shen, Wen-Ti Cheng, and Tsung-Han Wu for the life sharing.

Thanks also to my friends in other laboratories, Dr. Tung-Chieh Chen, Dr. Po-Hung Lin, Chung-Wei Lin, and Chin-Hsiung Hsu in Electronic Design Automation Laboratory, Tien-Ching Lin and Teng-Kai Yu in Algorithmic Theory and Applications Laboratory. They provided me a lot of useful comments on technical matters and research life.

My deepest appreciation goes to my family, especially for my mother, Li-Mei Hsu, my father, Shih-Jen Liu, and my sister, Chia-Yen Liu. Without their constant support and unending love, the completion of this dissertation would not have been possible. I also indeed appreciate my girlfriend, Yun-Hsiu Cheng, for her wholehearted company. Her patience and tenderness helped me to stand firm and erect in any frustration. I also want to thank my grandmothers, Yueh-Yun WU and Hsiu-Ling Shih, my uncles, Li-Jen Hsu, Li-Kuang Hsu, and Li-Chi Hsu, and my aunt, Shi-Hui Liu, for their endless and honest concerns since I was born. Especially, Uncle Li-Jen Hsu always gave me advice to guide me in a correct way. At last, I want to dedicate this dissertation to my late grandfathers, Yi-Hao Liu and Ching-Wen Hsu, and my late grandmother, Feng-O Chang. Although they could not see I receive the Ph.D. degree, I hope they would be proud of me.

Chih-Hung Liu

National Taiwan University

July 2009

摘要

直角史坦納(Steiner)樹問題長久以來，都是一個非常重要的研究課題，並且已經在積體電路設計領域提供了大量的應用。在積體電路設計中，繞線工具通常用直角史坦納樹去繞訊號線，而且很多連接最佳化問題先建構直角史坦納樹，然後再藉由改變線路大小或是插入緩衝器去滿足實際要求。此外，在早期實體設計階段，直角史坦納樹建構也常被使用去估計連接的線長。

然而，隨著科技的進步，現在的積體電路設計包含了越來越多的障礙物。這些障礙物主要是硬式矽智財、巨型區塊、和已繞的繞線。因此，在積體電路設計中，「考慮障礙物的直角史坦納樹問題」已經變成相當重要的實際問題，並且也受快速增加的注視。此外，因為現在的繞線環境包含了許多繞線層，積體電路設計必須分層處理。對於多重繞線層，至少有兩個重要限制必須被考慮。它們是特定繞線方向和不同繞線資源。首先，特定繞線方向指的是在不同繞線層的繞線方向。考量到訊號完整度和積體電路製程，同一繞線層的繞線方向傾向不是水平就是鉛直。其次，因為在較低繞線層通常需要較短的繞線，繞線工具能藉由給於較低繞線層較重的成本，去避免繞出較長的線長。這就是不同繞線資源。所以，「考慮障礙物的特定方向史坦納樹」問題需要有系統地規劃去考慮上述的限制。

在此篇博士論文中，我們企圖在「考慮障礙物的直角史坦納樹問題」和「考慮障礙物的特定方向史坦納樹問題」的研究上取得進展。此外，「考慮障礙物的特定方向史坦納樹問題」也在這篇博士論文中第一次被有系統地規劃。

我們對於「考慮障礙物的直角史坦納樹問題」的研究，企圖在早期實體電路階段，譬如置放(Placement)和平面規劃(Floorplanning)，提供快速且精確的連接線長估計。我們循序漸進地發展出許多新穎的策略，並且藉由這些策略完成三個高效能的演算法。這些演算法成功地同時在時間和線長上，達到了實際最好的效能。

首先，我們提出考慮障礙物繞線圖。與現存的繞線圖相比，這個考慮障礙物繞線圖不是包含較好的解答就是有較高的效能。接著，不同於先前的研究，我們提出一個根據路徑的架構。這個架構直接產生關鍵路徑當作解答元件，而沒有建構一個繞線圖或是產生一個錯誤解答。我們的根據路徑演算法保證能對一些特定測資，在線性對數時間內產生最佳解，這樣的保證在過去的研究要花立方的時間。

最後，我們提出「根據史坦納點的架構」和「史坦納點位置」的概念。這些架構和概念深刻地反映了考慮障礙物的直角史坦納樹問題的本質。這個方法能在實際的線性對數時間內產生目前最好的解答，這樣的解答過去要使用迷宮(maze)繞線在平方空間的繞線圖上才能得到。更重要的是，這兩個新想法可以很自然地給予未來的「考慮障礙物的直角史坦納樹研究」貢獻，以及相關的重要問題，譬如多層考慮障礙物的直角史坦納樹問題以及考慮障礙物的特定方向史坦納樹問題。

我們對於「考慮障礙物的特定方向史坦納樹問題」的研究，企圖在繞線階段，爲接下來的連接繞線最佳化提供較好的訊號線拓撲（topology）。作爲第一個研究，我們循序漸進地建立基本的理論基礎，以幫助未來相關演算法的發展。

首先，我們根本地分析最佳解的結構，並且提供分析解答品質的方法。在演算法的設計方面，解答品質的分析通常提供非常大的幫助，尤其是近似演算法的設計。第二，我們證明「考慮障礙物的特定方向最小延伸樹」是「考慮障礙物的特定方向史坦納樹問題」的二倍近似解，因此提供相當重要的特色，去支持強力的探索式方法（heuristic）。第三，我們證明「考慮障礙物的特定方向最小延伸樹」至少需要平方的空間，因此提供了較強的動機，去發展其他演算法。最後，我們分析了「局部最小的保證」以及「最小節點延伸樹（Minimal Terminal Spanning Tree）演算法的本質」，去設計了一個花費線性對數平方時間的演算法。

關鍵字：演算法、實體設計、繞線、史坦納樹、延伸樹、障礙物、多層。

Abstract

The *rectilinear Steiner minimal tree* (RSMT) problem has been a very important research issue for a long time, and already provided a lot of applications in very large scale integration (VLSI) design. In VLSI design, routing tools usually use *rectilinear Steiner trees* (RSTs) to route signal nets, and many interconnect optimization approaches begin with RST constructions, and then apply wire sizing, driver sizing, and buffer insertion to meet the performance requirements. Furthermore, at early physical design stages, the RST construction is also employed to make interconnect estimation.

However, as the technology advances, the modern *integrated circuit* (IC) design includes more and more routing obstacles incurred from hard intellectual property (IP) cores, macro blocks, and pre-routed nets. Therefore, the *obstacle-avoiding RSMT* (OA-RSMT) problem has arisen as an important practical problem in the VLSI physical design, and received dramatically increasing attention recently. Besides, since the modern routing environment includes a number of layers, IC designs are processed layer and layer. To deal with multiple layers, at least two important constraints should be considered: preferred directions and different routing resources. First, preferred directions are the routing orientations on those multiple layers. Considering signal integrity and IC manufacturing, the orientation of routing in a single layer tends to be either horizontal or vertical but not both. Second, since the lengths of wires tend to be shorter in lower layers, the router can weight the routing cost in lower layers higher to avoid routing long wires in lower layers, i.e., different routing resources. As a result, the *obstacle-avoiding preferred direction Steiner tree* (OAPD-ST) problem needs to be formulated to catch all the mentioned constraints.

This dissertation attempts to make progress in the study of the OARSMT problem and the OAPD-ST problem. This is also the first attempt to formulate the OAPD-ST problem.

The study for the OARSMT problem attempts to provide quick and accurate interconnect estimation at early physical design stages such as floorplanning and placement. We progressively develop novel strategies to bring about three excellent algorithms, and successfully achieve the best practical performance in both wirelength and run time.

Firstly, we propose an *obstacle-avoiding routing graph* (OARG). Compared with existing routing graphs, the OARG either contains better solutions or has higher efficiency. Secondly, unlike previous works, we propose a path-based framework to directly generate critical paths as solution components instead of constructing a routing graph or generating an invalid solution. Our path-based algorithm guarantees to provide optimal solutions for a number of specific cases, which requires $O(n^3)$ time in previous works.

Thirdly, we propose the Steiner-point based framework and the concept of Steiner point locations to give critical insights into the OARSMT problem. This approach achieves the best solution quality in empirical $\Theta(n \log n)$ time, which was originally obtained by applying maze routing on an $\Omega(n^2)$-space graph. More importantly, the two ideas naturally contribute to the future researches on the OARSMT problem and its important generations such the multi-layer OARSMT problem and the OAPD-ST problem.

The study for the OAPD-ST problem attempts to provide better signal net topologies at the routing stage for succeeding interconnect optimization. As the first study, we progressively build theoretical foundations for the development of future algorithms.

Firstly, we analyze the structure of the optimal solution, and provide a way to analyze the solution quality, which significantly helps the development of algorithms, especially for approximation ones. Secondly, we prove that an *obstacle-avoiding preferred direction minimum spanning tree* (OAPD-MST) is a factor 2 approximation solution for the OAPD-ST problem, and thus provides important features to support strong heuristics. Thirdly, we prove the space complexity of an OAPD-MST is $\Omega(n^2)$, and give a strong motivation to develop more efficient algorithms instead of OAPD-MST construction. Fourthly, we analyze local minimal guarantees and the essentials of an *minimal terminal spanning tree* (MTST) based algorithm to develop an $O(n \log^2 n)$-time strong heuristic.

Keywords: Algorithm, Physical Design, Routing, Steiner tree, Spanning Tree, Obstacles, Multi-Layer.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

VLSI routing has been a very critical design stage for a long time since it provides important interconnect information such as wirelength, congestion, and timing estimation, all of which dominate the correctness and the performance of the final design. In particular, as an intensely important step for routing, the routing tree construction plays a decisive role for the routing results. However, as the technology advances into the nanometer era, there are many new design challenges to make the routing tree construction much harder. In this dissertation, we present a series of strategies to deal with the routing tree construction considering rectilinear interconnect, a huge number of obstacles, multiple routing layers, preferred routing directions, and different routing resources.

1.1 Design Flow

Physical design of an *integrated circuit* (IC) is a process to convert a circuit description into a geometric mapping [39]. The physical design process succeeds circuit design and precedes fabrication. As shown in Figure 1.1, the physical design cycle is divided into

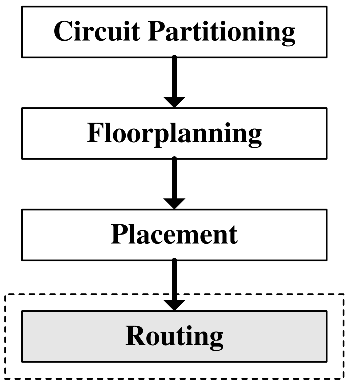

Figure 1.1: Physical design flow.

four major tasks: circuit partitioning, floorplanning, placement, and routing.

Circuit partitioning divides a circuit design into smaller parts such that each components falls within a prescribed range and the number of connections among those components is minimized. Floorplanning fills each component into one room such that those non-overlapping rooms are enveloped by a rectangle and the area of the rectangle is minimized. Placement further assigns exact locations for various components on a chip and/or determines pins locations according to appropriate cost functions. Routing routes all the signal nets completely to meet those given constraints such as timing issues, and to minimize the total costs for interconnect optimization.

Among all tasks in physical design, as the final step, routing has the most direct impact on the final design performance since it determines the timing delay, timing skew, and crosstalk of all the signal nets. Furthermore, a wrong choice made in early stages probably makes the resulting solution very hard to be refined, and thus appropriate routing-related schemes should be employed at the floorplanning or placement stage to estimate the interconnect wirelength. In other words, routing-related operations need to function across all the physical design flow to prevent the design failure. In particular, the routing tree construction is an essential step of routing and plays a decisive role for the final routing results. As a result, it is desired to study the various routing tree construction to address different requirements for difference purposes.

1.2 Rectilinear Steiner Tree Construction

The *rectilinear Steiner minimal tree* (RSMT) problem has been an important research issue for a long time, and its practical implementations have provided a lot of applications in VLSI design. Given a set of pins, an RSMT connects all the pins possibly through some additional points (called Steiner points) using rectilinear edges with minimum total wirelength. Many interconnect optimization approaches begin with constructing rectilinear Steiner trees and then apply wire sizing, driver sizing, and buffer insertion to meet the performance requirements. Furthermore, the *rectilinear Steiner tree* (RST) construction is usually invoked at the floorplanning and placement stages to estimate the interconnect wirelength.

The RSMT problem has been proved to be NP-complete [10], indicating that it seems intractable to find the RSMT in polynomial time. Hwang et al. [18] provided a comprehensive study of various RSMT-related works. For exact RSMT algorithms, the GeoSteiner package [42, 43] is currently the fastest implementation. Griffith et al [11] and Mandoiu et al. [30] also proposed near-optimal algorithms. For fast heuristics, Borah et al. [2] proposed an $O(n^2)$-time edge-based approximation algorithm; Zhou [50] used the spanning graph [51] to improve [2] to $O(n \log n)$ time, and achieved better solution quality. Recently, Chu and Wong [4] developed a very fast and accurate lookup table based RSMT algorithm called *fast lookup table estimation* (FLUTE). For the RSMT problem, FLUTE can efficiently construct the optimal solution for signal nets with at most 9 pins, and thus has been widely used at the floorplanning and placement stages as well as the global and detailed routing process.

However, as the technology advances, there are new challenges for the RST construction. First, to cope with the increasing design complexity, IP modules are widely reused for large-scale designs, and form a huge number of obstacles. Therefore, the RST construction, even optimal ones, will cause inaccurate wirelength estimation at the floorplanning and placement stages, which may lead to the design failure at the routing stage. Second, aside from the obstacles, there are also quite a few routing constraints in modern IC design such as multiple routing layers, different routing resources, preferred routing directions, and via costs. As a result, the RST construction may provide wrong routing topologies which cannot be refined to meets practical requirements. To conclude, it is necessary to propose appropriate mechanics, frameworks, and algorithmic skills to meet the various requirements of the routing tree construction at each stage of the physical design flow.

1.3 Obstacle-Avoiding Rectilinear Steiner Tree Routing

To cope with the significantly increasing design complexity, IP modules are widely reused for large-scale designs, and the interconnect complexity is also significantly increased. As a results, there are more and more obstacles including *macro cells, IP blocks, pre-routed nets, heavily congested regions,* etc.. Under these circumstances, the RST construction without considering obstacles will make inaccurate wirelength estimation, and even leads to the design failure. To remedy the deficiency, the *obstacle-avoiding RSMT* (OARSMT) problem has become more important than ever, and received dramatically increasing attentions [8, 14, 15, 22, 28, 36–38, 44, 47]. Since the RSMT problem, even without considering the obstacles, has been proved to be NP-Complete [10], the presence of obstacles further increases the difficulty of the *obstacle-avoiding RST* (OARST) construction, and takes much more run time to obtain satisfactory solutions.

Aside from the difficulty, the large and increasing number of obstacles also raises the requirement of efficiency (run time). For the RSMT problem, since the number of pin-vertices is quire small in practice, an $O(n^3)$-time algorithm is acceptable. However, according to ITRS [19], the hard IP count per chip will be more than one thousand in the near future, indicating that the number of obstacles will dominate the run time instead of that of pins. In other words, an $O(n^2)$-time algorithm will not be sufficiently efficient for the OARST construction. Furthermore, to estimate the interconnect wirelength in early stages, Steiner tree construction will be invoked millions of times in floorplanning and placement phases [35]. Therefore, it is necessary to develop an $O(n \log n)$-time approach to handle a huge number of obstacles.

On the other hand, the effectiveness (wirelength) could not be sacrificed to increase the efficiency. First of all, by 90nm, wiring delay dominates 75% of the overall delay [39], and the high overall delay will cause the design failure. Second, in a modern system-on-chip (SOC) design, there are a million number of signal nets. To increase the routability, the wiring congestion in routing area should be minimized, and thus the wirelength should be minimized for each net.

To conclude, as the technology advances, the OARST construction has become much more important. Considering a huge number of obstacles, millions of signal nets, overall timing delay, and routing congestion, the OARST construction requires both the high efficiency (small run time) and the high effectiveness (small wirelength).

Most existing works targeting on the OARSMT problem can be categorized into two major categories: (1) construction-by-correction and (2) connected-graph based. The first class of algorithms generates an initial solution without considering the obstacles, and then legalizes the edges intersecting obstacles. The second class constructs a connected graph embedding at least one desirable solution, and applies graph algorithms to obtain a solution. On the whole, connected-graph based approaches usually have the global view of the obstacles, and generate better solutions. According to the routing graph, the second class can be further partitioned into two sub-classes: (1) rectilinear graph and (2) spanning graph. It will be shown latter that there exists a trade-off between rectilinear graphs and spanning graphs in effectiveness and efficiency. Besides, there also exist hybrid approaches integrating the two categories. In the following, we introduce major existing approaches according to the above classification.

1.3.1 Construction-by-Correction Approach

The construction-by-correction approach first constructs a Steiner tree or a spanning tree without considering the obstacles, and then replaces the edges intersecting obstacles with edges around the obstacles. Due to its simplicity, this approach is very efficient, and was widely used in industry. However, since the first step does not consider the obstacles, the resulting solution may be very hard to refined to an satisfactory one. Furthermore, with the significant increasing of obstacles, the situation will probably become much worse. In other words, the construction-by-correction approach may not have the global view of the obstacles, and thus the solution quality would be limited. Similar arguments were also

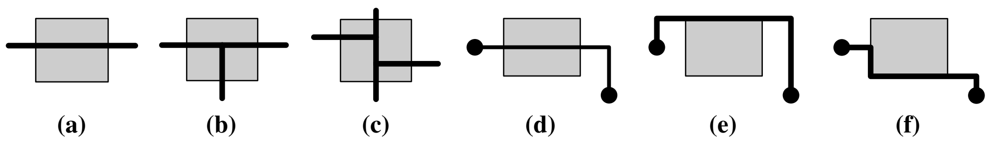

Figure 1.2: Yang et al. [47]. (a)–(c) three proposed cases of edge overlapping. (d) an example for edge overlapping. (e) the possible overlapping removal for (d) without the preprocessing. (f) the overlapping removal for (d) with the preprocessing.

pointed out in [22,28,36]. Below, we introduce one example work.

Yang et al. [47] developed a four-step heuristic to remove the overlapping edges. They proposed three cases of edge overlapping as shown in Figure 1.2(a), (b), and (c). In accordance with the three cases, their heuristic uses a preprocessing to construct new edges and to remove the edge overlapping. Figure 1.2(e) shows the possible result without their preprocessing, and Figure 1.2(e) shows the result with their preprocessing, which is much better.

1.3.2 Rectilinear-Graph Based Approach

A rectilinear graph uses rectilinear edges to connect pin-vertices, obstacle corners, and more other desirable Steiner point candidates. Therefore, a rectilinear graph usually contains better solutions and even the optimal solution, e.g., Escape graph [9] and extended Hanan grid [21]. There are quite a few novel mechanics to derive excellent solutions from rectilinear graphs. For example, to the best of our knowledge, the maze-routing approach in [21] achieves the best solution quality compared with all the state-of-the-art works.

However, most rectilinear graph has $\Omega(n^2)$ space, and thus significantly lower the efficiency. For example, the approach in [21] performs much slower than those in [8,28]. In short, compared to a spanning-graph based approach, a rectilinear-graph based approach

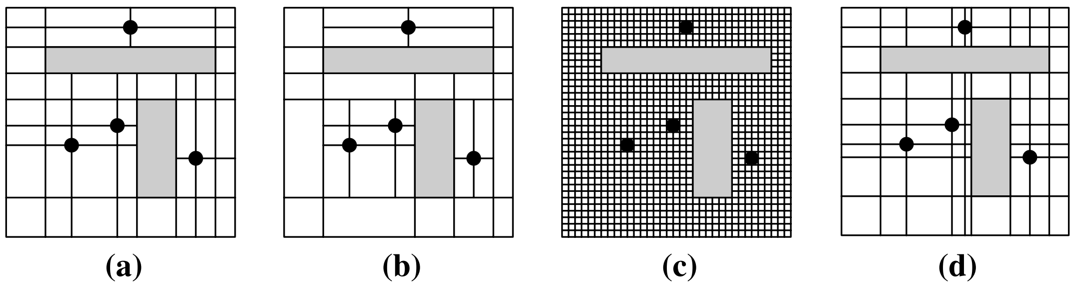

Figure 1.3: Example rectilinear graph. (a) Escape graph [9]. (b) Track graph [15,45]. (c) global routing graph [38]. (d) extended Hanan grid [21].

has higher effectiveness (solution quality) but lower efficiency (speed performance). Four examples of rectilinear graphs are illustrated in Figure 1.3, and the corresponding approaches are discussed below.

Ganley and Cohoon introduced a routing graph called *Escape graph* [shown in Figure 1.3(a)], and proved that at least one optimal solution exists in Escape graph. Based on Escape graph, they also provided three approximation algorithms with time complexity of $O(n^3 \log n)$, $O(n^4)$, and $O(n^7)$ respectively. Their algorithms perform well for smaller cases. Hu et al. proposed a nondeterministic local search heuristic, called An-OARSMan, which mainly applies the ant colony optimization [7] on Track graph [45] [shown in Figure 1.3(b)]. Their approach can handle small cases with complex obstacles of both concave and convex shapes.

Shi et al [38] proposed an *global routing graph* (GRG) [shown in Figure 1.3(c)], which is a uniform graph generated from Escape graph, and applied circuit simulation on it. Their approach also performs well for obstacle-free cases. Very recently, Li and Young developed a maze-routing scheme, and applied it on the extended Hanan grid [shown in Figure 1.3(d)]. Li's approach uses the maze-routing to generate desirable paths as solution components, and then constructs a *minimum spanning tree* (MST) based on those

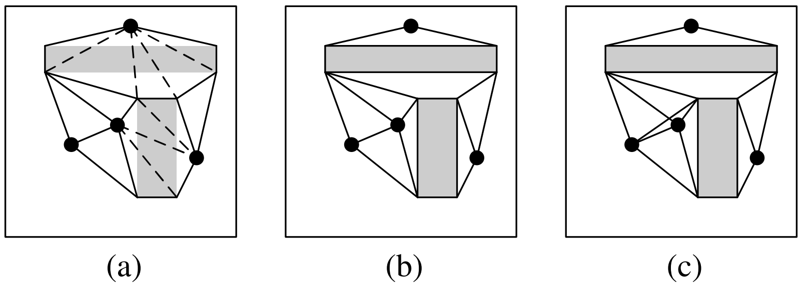
Figure 1.4: Example spanning graph. (a) obstacle-avoiding constrained Delaunay triangulation (OACDT) [8]. (b) Shen's and Long's obstacle-avoiding spanning graph (OASG) [28, 36]. (c) Lin's OASG [22].

components. The solution quality of Li's approach outperforms all the state-of-the-art works, but the speed performance is rather low.

1.3.3 Spanning-Graph Based Approach

A spanning graph often has sparse edges, and only consists of pin-vertices and obstacle corners. Therefore, a spanning graph significantly increases the efficiency. There are a number of efficient approaches based on spanning graphs. For example, to the best of our knowledge, the approaches in [8, 28] achieve the best speed performance.

However, since a good Steiner point is not necessarily restricted on obstacle corners, the corresponding solution quality may be limited compared with rectilinear-graph based ones. For example, the solution quality of the approaches in [8, 22, 28, 36] is still worse than that in [21]. In short, compared to a rectilinear-graph based approach, a spanning-graph based approach has higher efficiency (speed performance) but lower effectiveness (solution quality). Three examples of spanning graphs are illustrated in Figure 1.3; the corresponding approaches are discussed below according to their frameworks.

Feng et al. [8] proposed an *obstacle-avoiding constrained Delaunay triangulation*

(OACDT) [shown in Figure 1.4(a)], and developed a 3-step algorithm based on the OACDT. To the best of our knowledge, Feng's method is the first and the only one algorithm for the *obstacle-avoiding Steiner minimal tree* (OASMT) problem in **lambda-geometry**, and has $O(n \log n)$-time complexity. Although Feng's method can be applied to the OARSMT problem, experimental results in [22] show that Feng's method generates much larger wirelength compared with [22, 36].

Three recent works [22, 28, 36] share a very common framework. Shen et. al [36] proposed the common structure as follows: (1) graph construction, (2) *minimal terminal spanning tree* (MTST) construction, and (3) RST construction. In this dissertation, we denote this framework as *an MTST-based framework*, and an algorithm following this framework as an MTST-based algorithm. Among existing works, the integration of an spanning graph and an MTST-based algorithm has higher balance in efficiency and effectiveness.

Zhou et al. [51] divided a plane into eight octal regions for each pin-vertex, connected each pin-vertex to the closest vertex in each its octal region, and thus constructed a spanning graph which guarantees the existence of a *rectilinear minimum spanning tree* (RSMT) on an obstacle-free plane. Shen et al. [36] simplify Zhou's eight octal regions into four quadrant regions, and constructed an *obstacle-avoiding spanning graph* (OASG) [shown in Figure 1.4(b)], which also has only $O(n)$ edges.

Lin et al. [22] proposed a new OASG [shown in Figure 1.4(c)] which includes more essential edges than Shen's OASG. Lin's OASG guarantees a rectilinear shortest path for any two pins, and thus Lin's method provides the optimal solution for any two-pin nets and a number of specific multiple-pin nets. Experimental results have shown that the specific theoretical guarantee greatly supports better solutions. However, since Lin's OASG has $O(n^2)$ edges, the time complexity is $O(n^3)$ in the worst case. Long et al. [28] proposed a faster MTST construction algorithm with time complexity of $O(|E| \log |V|)$. Besides, they also constructed an OASG which is almost equivalent to Shen's OASG in another way. Since Long's OASG has only $O(n)$ edges, Long's method has $O(n \log n)$ time complexity, and achieves the best speed performance. Nevertheless, Long's method cannot guarantee the specific theoretical optimality in [22], and the solution quality is still worse than that of [22].

1.3.4 Hybrid Approach

Wu et al. [44] presented a hybrid approach for the OARSMT problem. First, similar to a construction-by-correction approach, their approach constructs a *minimum spanning tree* (MST) for all the pins without considering the obstacles. Second, their approach integrates their constructed MST, the ant-colony optimization in [15], and the OASG in [36] to construct an *obstacle-avoiding Steiner tree* (OAST). Finally, their approach transforms the constructed OAST into an OARST, and performs refinements to reduce the redundant segments. Experimental results show that the solution quality and the speed performance of their approach are between those in other existing approaches.

1.3.5 Challenge of the OARSMT problem

The OARSMT problem is becoming more important as the technology advances. In modern VLSI design, there are more and more obstacles, such as macro cells, IP blocks, pre-routed nets, etc., and thus the original RST construction would provide inaccurate interconnect estimation. Since a wrong choice made in early stages potentially leads to the design failure in the current top-down flow, the OARST construction should be employed in early stages such as floorplanning and placement.

Under these circumstances, the OARST construction requires both the efficiency (small run time) and the effectiveness (small wirelength). For the former, the huge number of obstacles dominates the run time instead of pins, and the Steiner tree construction will be invoked millions of times at the floorplanning and placement stages. For the latter, wiring delay dominates 75% of the overall delay by 90 nm, and the wiring congestion should be minimized to handle millions of signal nets in an SOC design. However, most existing approaches are limited in either solution quality or speed performance. Therefore, it is desired to develop new mechanics, frameworks, and algorithmic skills to meet the performance requirements of the OARST construction.

1.4 Obstacle-Avoiding Preferred Direction Steiner Tree Routing

Aside from the obstacles, with the progress of IC technology, there are other new practical constraints in the routing process. In particular, for the routing tree construction, *multiple routing layers*, *preferred routing directions*, and *different routing resources* need to be considered.

Multiple routing layers are layered metal material, and the metal material will partially be the wires of signal nets. To utilize the metal material, the modern IC design are processed layer by layer. Under these circumstances, pins are connected by rectilinear edges within layers and vias between layers. Moreover, most pins of standard cells are located in lower layers, while many pins of macro cells are located in higher layers. Therefore, the routing tree construction should be able to connect all the pins of a signal net, no matter on which layers those pins are.

Preferred directions are the orientations of routing among multiple routing layers. Considering signal integrity and IC manufacturing, the orientation of routing in a single layer tends to be either *vertical* or *horizontal*.

Different routing resources are the different weight of routing resources in different layers. In general, the widths of wires in lower layers are thinner than those in upper layers. Therefore, to balance the resistance and timing delay, wires in lower layers tend to be shorter than those in upper layers. Toward this end, the router needs to weight the routing costs in lower layers significantly higher such that the router would route shorter wires in lower layers.

To the best of our knowledge, none of the existing works entirely catches all the mentioned processing constraints at the same time. it is desired to formulate the *obstacle-avoiding preferred direction Steiner tree* (OAPD-ST) problem to simultaneously deal with the five constraints: (1) multiple routing layers, (2) obstacles, (3) preferred directions, (4) different routing resources, and (5) via costs.

In the following subsections, we introduce two highly related works.

1.4.1 The Preferred Direction Steiner Tree Problem

Yildiz and Madden [48] discussed the *preferred direction Steiner tree* (PDST) problem which reflects the current multi-layer design (allows preferred directions). Their PDST

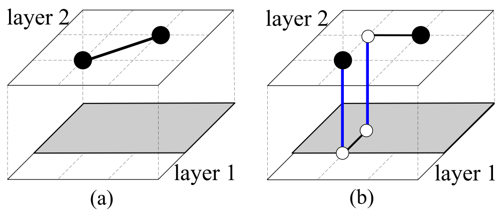

Figure 1.5: An instance about the invalid transformation for a slant edge into preferred direction edges. (a) a slant edge. (b) illegal transformed edges.

problem also consider the routability by the following two strategies:

- **Different routing resources**: The wires are assigned different unit cost in different layers to minimize congestion. For instance, in a more congested layer, the wires have a higher unit cost such that the router would avoid routing nets through this congested layer.

- **Via costs**: Generally speaking, vias can enhance the transmission of signal, but too many vias will cause a waste of area, which increases the congestion. Hence, via costs should be employed to balance the benefit and loss.

However, their PDST model does not consider the obstacles, which cannot be ignored in modern IC design

1.4.2 The Multi-Layer OARSMT Problem

Lin et al. [24] formulate the *multi-layer* OARSMT (ML-OARSMT) problem to deal with the obstacles and multiple routing layers. They extended their original method in [22] to construct a new routing graph called *multi-layer obstacle-avoiding spanning graph* (ML-OASG).

However, ML-OASG could not be suitable for the OAPD-ST problem due to its edges, that is, ML-OASG not only allows horizontal and vertical edges in the same layer but also uses slant edges. In ML-OASG, slant edges are used to connect points and to reduce the wirelength by their heuristic. In other words, slant edges are the backbone of ML-OASG. Unfortunately, slant edges have no information about the PD constraints such that the transformations from slant edges into preferred direction edges will cause an infeasible solution. Figure 1.5(a) shows a slant edge, and Figure 1.5(b) shows the corresponding invalid transformed preferred direction edges. However, if removing the slant edges and invalid rectilinear edges, ML-OASG may not be connected such that no feasible solution for the OAPD-ST problem can be generated.

1.4.3 Challenge of the OAPD-ST Problem

At the current routing stage, there are many new processing constraints such as obstacles, multiple routing layers, preferred directions, and different routing resources, all of which are important for the interconnect optimization. To prevent the design failure, the routing tree construction should address those constraints. However, no existing works entirely catches all the mentioned constraints since each of those constraints is hard to handle. It is desired to formulate the OAPD-ST problem, and to develop novel techniques to attack it. Since there is no study on the OAPD-ST problem, it is necessary to build essential theoretical foundations such as the structure of the optimal solution and the approximation guarantee. Those theoretical foundations usually give critical insight into an NP-complete problem and provide a potential way to develop future algorithms.

1.5 Overview of the Dissertation

In this dissertation, we study two increasingly important routing tree issues: the OARSMT problem and the OAPD-ST problem. Routing is a critical step in physical design, and the routing tree construction plays a crucial role for the routing results. As the technology advances into nanometer era, there are much more routing constraints, such as obstacles, multiple routing layers, preferred directions, and different routing resources. Therefore, the original routing tree construction will make inaccurate wirelength estimation at the floorplanning or placement stage, and may generate wrong topologies at the routing stage which cannot be refined to meet practical interconnect constraints. All these defects probably lead to the design failure. As a result, it is necessary to address those practical constraints, and propose new various techniques to meet different requirements at different stages of the physical design flow.

The study for the OARSMT problem attempts to provide quick and accurate interconnect estimation at early physical design stages such as floorplanning and placement. Although more than ten recent works provide diverse approaches, none of them is able to achieve the best efficiency and the best effectiveness at the same time. Therefore, we progressively develop a series of strategies to attack the OARSMT problem. Based on these strategies, we develop three excellent algorithms, and successfully achieve our purposes, i.e., the the best practical performance in both wirelength and run time.

First, since the integration of a spanning graph and an MTST-based algorithm has better balance in efficiency and effectiveness as discussed in Section 1.3.3, we propose an advanced OASG as well as efficient wirelength reduction approaches. Second, since the specific theoretical optimality guarantee in [22] greatly supports better solutions, we propose a path-based framework to obtain the specific theoretical optimality in $O(n \log n)$ time instead of the original $O(n^3)$. Third, since there exists a tradeoff between rectilinear graphs and spanning graphs in the effectiveness and the efficiency as discussed in Section 1.3, we analyze the essence of the OARSMT problem, and propose the idea of Steiner point selection to integrate advantages of rectilinear graphs and spanning graphs.

The study for the OAPD-ST problem attempts to provide better tree topologies at the routing stage for the succeeding interconnect optimization. As a first study of the OAPD-ST problem, we want to build essential theoretical foundations for the development of future algorithms such as the structure of the optimal solution, the approximation algorithm, the bottleneck of complexity, and the local minimal heuristic. All of these are critical in the study of Steiner tree problem [18], and significantly help the research and the development of future algorithms. Based on these theoretical foundations, we also develop one approximation algorithm, and one efficient heuristic.

We divide this dissertation into two parts. Part 1 studies the OARSMT problem, and includes Chapter 3, Chapter 4, and Chapter 5. Part 2 studies the OAPD-ST problem and includes Chapter 6 and Chapter 7. Chapter 3, Chapter 4, and Chapter 6 have been published in [27], [26], and [25] respectively. We introduce these chapters below.

1.5.1 Advanced Obstacle-Avoiding Spanning Graph

We propose an advanced OASG called *obstacle-avoiding routing graph* (OARG) to develop an $O(n \log n)$-time MTST-based algorithm as well as an efficient local refinement. Compared with the OASG in [28, 36], the OARG contains more local OARMSTs and thus would contains better solutions. Compared with the OASG in [22], the OARG has fewer edge and will increase the efficiency. First of all, we utilize the rectangle-avoidance property (Definition 3.2) to construct the OARG in $O(n \log n)$ time. Besides, based the rectangle-avoidance property, we also integrate existing approaches reduce the wirelength during the OARST construction. Finally, we present an $O(n \log n)$-time local refinement scheme to reduce the redundant segments. Extensive experimental results show that our approach outperforms [22, 28, 36] in both run and wirelength.

1.5.2 Path-Based Framework

We presents a path-based framework to bring about an $O(n \log n)$-time algorithm with theoretical optimality guarantees on a number of specific cases, which required $O(n^3)$ time in previous works. Unlike previous frameworks, the new framework directly generates critical paths as essential solution components instead of generating invalid initial solutions or constructing connected routing graphs. We integrate computational geometry skills to generate $O(n)$ critical paths, and prove that those paths guarantee the existence of the optimal for specific cases and only take $O(n)$ space. Therefore, the new framework provides a new way to deal with the OARSMT problem. Experimental results show that our algorithm achieves the best speed performance, while the average wirelength of the resulting solutions is only 1.1% longer than that of the best existing solutions.

1.5.3 Steiner Point Selection

We restudy the essence of the OARSMT problem, and propose the idea of Steiner point selection to integrate the advantages of rectilinear graphs and spanning graphs. The idea consists of two major components, the Steiner-point based framework and the concept of Steiner point locations. Unlike many previous works, the Steiner-based framework is more focused on the usage of Steiner points instead of the handling of obstacles, and seems closer to the essence of the OARSMT problem. The concept of Steiner point locations reflects the nature of Steiner points from another viewpoint, and thus provides an effective as well as efficient way to generate desirable Steiner point candidates.

We also give an informal but intuitive analysis to show that the average-case time complexity of our algorithm seems $O(n \log n)$. Experimental results show that this algorithm achieves the best solution quality in $\Theta(n \log n)$ empirical time, which was originally generated by applying the maze routing on an $\Omega(n^2)$-space graph. More importantly, the idea of Steiner point selection can be applied to the future researches on the OARSMT problem and its generations, such as the ML-OARSMT problem and the OA-PDST problem.

1.5.4 Preferred Direction Evading Graph and Approximation Guarantee

We analyze the structure of the optimal solution and propose an approximation algorithm. For the structure of the optimal solution, we propose *preferred direction evading graph* (PDEG), and prove that at least one optimal solution exists in PDEG. The theoretical optimality proof provides a way to analyze the solution quality, which significantly help the development of algorithms, especially for approximation ones. For the approximation algorithm, based on PDEG, we prove that the approximation factor of an *obstacle-avoiding preferred direction minimum spanning tree* (OAPD-MST) to the OAPD-ST problem is 2, and thus the OAPD-MST construction is a factor 2 approximation algorithm for the OAPD-ST problem. The approximation guarantee of an OAPD-MST gives important features to support the development of strong heuristics, especially for MST-like heuristics.

1.5.5 Time Complexity Bottleneck and Local Minimal Heuristic

We first analyze the worst-case complexity of the OAPD-MST construction, and then develop a local minimal heuristic based on a local minimal guarantee and an MTST-based concept. We first prove that the space complexity of an OAPD-MST is $\Omega(n^2)$, which gives a strong motivation to develop more efficient algorithms for the OAPD-ST problem instead of the OAPD-MST construction. Then, we analyze the properties of MTST-based algorithm and make critical inference. Based on the inference and other computational geometry skills, we propose a routing graph, *preferred direction visibility graph* (PDVG) with a local minimal guarantee, and develop an $O(n \log^2 n)$-time MTST-based heuristic. Experimental results shows that our algorithm performs more efficiently than OAPD-MST construction and can generate comparable solutions. Experimental results also show the high competitiveness of PDVG and justify all our claims about PDVG and our algorithm.

1.6 Organization of the Dissertation

The reminder of this dissertation is organized as follows. Chapter 2 formulates the two addressed problems, and gives overall definitions, assumptions, and abbreviations. Part 1 studies the OARSMT problem and consists of Chapter 3, Chapter 4, and Chapter 5. Chapter 3 presents an $O(n \log n)$-time advanced spanning-graph based algorithm. Chapter 4 develops a path-based framework to bring about an $O(n \log n)$-time algorithm with a specific theoretical optimality guarantee, which took $O(n^3)$ in previous works. Chapter 5 proposes an idea of Steiner-point selection including the Steiner-point based framework and the concept of Steiner point locations, which lead to an excellent algorithm with the best practical performance in both wirelength and run time. Part 2 studies the OAPD-ST problem and includes Chapter 6 and Chapter 7. Chapter 6 proves the optimality guarantee of PDEG, and proposes a factor 2 approximation algorithm for the OAPD-ST problem. Chapter 7 analyzes the lower bound of time complexity to construct an OAPD-MST, and develops a more efficient heuristic. Chapter 8 makes concluding remarks for this dissertation.

Chapter 2

Preliminary

This chapter gives basic definitions, symbol notations and fundamental assumptions in this dissertation. Based those definitions, notations, and assumptions, the OARSMT problem and the OAPD-ST problem are clearly formulated. This chapter also gives extended definitions, which are widely used in this dissertation.

2.1 Basic Definitions

Definition 2.1 *An* ***obstacle*** *is a rectilinear polygon on a plane (layer). Any two obstacles cannot intersect with each other, but could be line touched at the corner and point-touched at the boundary.*

Definition 2.2 *An* ***pin-vertex*** *is a point on a plane (layer) and should be connected by a signal net. No pin-vertex can be inside any obstacle, but a pin-vertex could be located at the corner or on the boundary of an obstacle.*

Figure 2.1(a) shows two intersecting obstacles in layer 1, and three line-touched or point-touched obstacles in layer 2. Figure 2.1(b) shows two invalid pin-vertices inside an

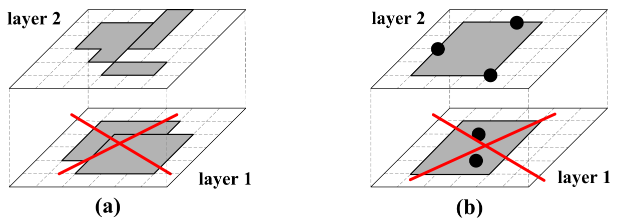

Figure 2.1: Obstacles and pins. (a) Any two obstacles cannot intersect with each other (layer 1), but two obstacles could be line-touched at the boundary or point-touched at the boundary (layer 2). corners of an obstacle. (b) a pin-vertex cannot be inside any obstacles (layer 1), but could be located at the corner or on the boundary of an obstacle (layer 2).

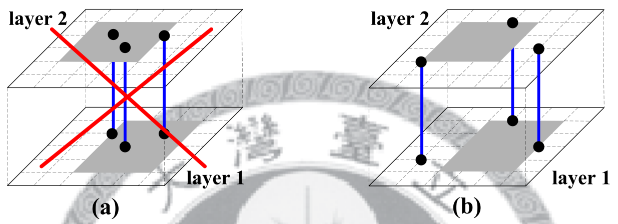

Figure 2.2: Vias. (a) The endpoints of a via cannot locate inside an obstacle. (b) The endpoints of a via could be on the boundary or at the corner of an obstacle.

obstacle in layer 1, and three pin-vertices at the corner or on the boundary of an obstacle in layer 2.

Definition 2.3 *A* ***via*** *between layer z and $z+1$ is an edge between (x, y, z) and $(x, y, z+1)$. Neither of the two endpoints can locate inside an obstacle, but they could be on the boundary or at the corner of an obstacle.*

Figure 2.2(a) shows invalid vias whose endpoints locate inside an obstacle, and Figure 2.2(b) shows valid vias whose endpoints are on the boundary or at the corner of an obstacle.

Definition 2.4 *For a tree, a Seiner point is a no-pin vertex with degree of 3 or 4.*

2.2 Fundamental Assumptions

Assume that the costs of vias are the same, and the unit costs of wires in the same layer are identical. Without loss of generality, we assume the *preferred direction* (PD) constraints as follows:

- *Odd* layers only allow *vertical* edges.
- *Even* ones only allow *horizontal* edges.

All edges/paths/distances are measured by rectilinear distance (L_1 metric), e.g., the length of an edge (v_i,v_j), denoted as $|(v_i, v_j)|$, is $|x_i - x_j| + |y_i - y_j|$. For simplification, we assume that neither an edge nor a path runs over any obstacles except their boundaries. In preferred direction model, a path is further assumed to meet the preferred direction constraints. For two vertices v and u, $\boldsymbol{SP(v, u)}$ represents a shortest path between v and u, and the length of $SP(v, u)$ is $|SP(v, u)|$.

2.3 Symbol Notations

Let $\boldsymbol{N_l}$ be the number of routing layers, $\boldsymbol{P = \{P_1, P_2, \ldots, P_m\}}$ be a set of pin-vertices for an m-pin net, $\boldsymbol{O = \{O_1, O_2, \ldots, O_k\}}$ be a set of $\boldsymbol{k}$ obstacles, $\boldsymbol{N_c}$ be the number of obstacle corners in O, and $\boldsymbol{n}$ be the size of $P \bigcup \{\text{obstacle corners in } O\}$. Thus, we have $\boldsymbol{n \leq m + N_c}$. Since N_l is a small constant in practice, let $\boldsymbol{n}$ be the input size for this problem. Let $\boldsymbol{C_v}$ be the cost of a via, and $\boldsymbol{UC_i}$ be the unit cost of wires in layer $\boldsymbol{i}$. Table 2.1 summarize the main symbols in this dissertation.

Table 2.1: Symbols.

$\boldsymbol{P}$	the set of pin-vertices	$\boldsymbol{O}$	the set of obstacles
$\boldsymbol{U_i}$	the unit cost of wire in layer i	$\boldsymbol{C_v}$	the cost of a via
$\boldsymbol{m}$	the number of pin-vertices	$\boldsymbol{k}$	the number of obstacles
$\boldsymbol{N_c}$	the number of obstacle corners	$\boldsymbol{N_l}$	the number of layers
$\boldsymbol{n}$	the size of $P \bigcup \{$obstacle corners in $O\}$. $n \leq m + N_c$. (**the input size**)		

2.4 Problem Formulation

- **The Obstacle-Avoiding Rectilinear Steiner Minimal Tree Problem:**

  Given a set P of pins and a set O of obstacles on a plane, construct a tree connecting all the pins in P possibly through some additional points (called Steiner points) using vertical or horizontal edges, such that no tree edges intersect the interior of any obstacles in O and the total wirelength is minimized.

Hereafter, we denote an *obstacle-avoiding rectilinear Steiner tree* (OARST) as a solution and an OARSMT as an optimal solution.

- **The Obstacle-Avoiding Preferred Direction Steiner Tree Problem:**

  Given a constant C_v, a set P of pin-vertices, a set O of obstacles, N_l routing layers with their specific unit costs of wires (UC_i, $1 \leq i \leq N_l$), and PD constraints, construct a routing tree connecting all the pin-vertices in P possibly through some additional points (called Steiner points) such that no tree edges intersect any obstacles in O or violate the PD constraints.

Throughout this paper, we denote a solution to this problem as an *OAPD-ST* and an optimal solution whose cost is minimum as an *OAPD-SMT* or an *optimal OAPD-ST*.

Table 2.2 shows the main abbreviations in this dissertation. Most of those abbreviations will be introduced in the remaining parts of this dissertation.

Table 2.2: Abbreviations.

OARSMT	obstacle-avoiding rectilinear Steiner minimal tree
OARST	obstacle-avoiding rectilinear Steiner tree
OAPD-ST	obstacle-avoiding preferred direction Steiner tree
OAPD-SMT	obstacle-avoiding preferred direction Steiner minimal tree
MTST	minimum terminal Steiner tree
OARMST	obstacle-avoiding rectilinear minimum spanning tree
OAPD-MST	obstacle-avoiding preferred direction minimum spanning tree

2.5 Extended Definitions

We define a series of MST-related terms, and these terms will be widely used in this dissertation.

Definition 2.5 *Given a graph $G(V, E)$, a* ***minimum spanning tree*** *(MST) is a tree connecting all the vertices in V with the minimum wirelength.*

Below, we define a minimum terminal spanning tree in two ways. The first way is based on virtual edges (Definition 2.6) and the same as a *generalized minimum spanning tree* in [46]. The second way is based on terminal paths and was used in [28]. In fact, the two definitions are equivalent to each other but applied in different aspects. Chapter 3 uses Definition 2.7 and Chapter 4 uses Definition 2.9

Definition 2.6 *Given a graph $G(V, E)$, a* ***virtual edge*** *is a shortest path between two vertices. The length of a* ***virtual edge*** *is the length of the corresponding shortest path.*

Definition 2.7 *Given a graph $G(V, E)$ and a terminal set $S \subseteq V$, an* ***minimum terminal spanning tree*** *(MTST) is an MST of a new graph whose vertex set is S and whose edge set is the corresponding virtual edges among all the vertices in S.*

Definition 2.8 *Given a graph $G(V, E)$ and a terminal set $S \subseteq V$, a* ***terminal path*** *is a path between two vertices in S without other internal vertices in S [28].*

Definition 2.9 *For a graph $G(V, E)$ and a terminal set $S \subseteq V$, a* ***minimum terminal spanning tree*** *(MTST) connects all vertices in S using a set of terminal paths with the minimum sum of the lengths of those terminal paths.*

Based on Definition 2.5, we define an OARMST and OAPD-MST. Since there is no edge in geometry domain, a shortest path between two pin-vertices is naturally viewed as a edge connecting them.

Definition 2.10 *Given an OARSMT problem instance, an* ***obstacle-avoiding rectilinear minimum spanning tree*** *(OARMST) connects all the pin-vertices using a set of shortest paths among those pin-vertices such that the total wirelength of those paths is minimized.*

Definition 2.11 *Given an OAPD-ST problem instance, an* ***obstacle-avoiding preferred direction minimum spanning tree*** *(OAPD-MST) connects all the pin-vertices using a set of obstacle-avoiding preferred direction shortest paths (OAPD-SPs) among those pin-vertices such that the sum of costs of those OAPD-SPs is minimized.*

Chapter 3

Advanced Obstacle-Avoiding Spanning Graph

We propose a new spanning graph called *obstacle-avoiding routing graph* (OARG) for the OARSMT problem, and develop an $O(n \log n)$-time MTST-based algorithm as well as an efficient local refinement. Compared with the OASG in [28,36], the OARG contains more local OARMST leading to better solution, and compared with the OASG in [22], the OARG has fewer edge heightening the efficiency. At the first step, we utilize the rectangle-avoidance property (Definition 3.2) to construct the OARG in $O(n \log n)$ time. At the second step, we develop a more efficient MTST construction to generate an initial Steiner tree. At the third step, we integrate existing approaches and the rectangle-avoidance property to construct an OARST. Finally, we present a $O(n \log n)$-time local refinement scheme to reduce the redundant segments. Extensive experimental results show that our method outperforms [22, 28, 36] in both wirelength and run time.

3.1 Motivation

As discussed in Section1.3.3, a spanning graph has the global view of obstacles and spare edges, which leads desirable OARSMT algorithms. Also, MTST-based algorithms on spanning graphs [22,28,36] has better balance in wirelength and run time. However, there still exists a significant trade-off between existing spanning graphs in space and solution quality. Shen's OASG in [28, 36] is mainly extended from Zhou's octal regions [51], and Zhou's spanning graph guarantees a rectilinear minimum spanning tree in an obstacle-free plane. However, Shen's OASG only considers four quadrants, and thus it may lose some desirable edges leading to better solutions. On the other hand, Lin's OASG [22] has more essential edges to include better solutions, while it has $O(n^2)$ edges, which significantly lower the efficiency. Therefore, we attempt to develop algorithmic skills to apply Zhou's octal regions in the presence of obstacles and to construct an $O(n)$-space graph with desirable solutions. We also want to extend existing obstacle-free heuristics such as [2,50] to further improve the solution. In this chapter, the most critical algorithmic technique is the utilization of the rectangle avoidance property in Section 3.2.1 and Section 3.2.3.

3.2 Algorithm

Our three-step algorithm can be summarized as follows:

1. *OARG* Construction: In this step, an *obstacle-avoiding routing graph* (OARG) shown in Figure 3.1(b) is constructed for the following *obstacle-avoiding rectilinear Steiner tree* (OARST) construction. We will prove that the time and space complexities of OARG are $O(n \log n)$ and $O(n)$ respectively.

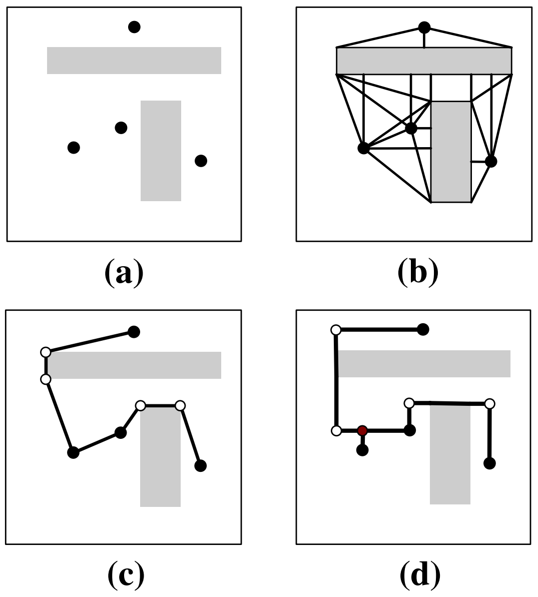

Figure 3.1: Procedure of our 3-step algorithm . (a) a problem instance. (b) OARG construction. (c) MTST-OARG construction. (d) OARST transformation.

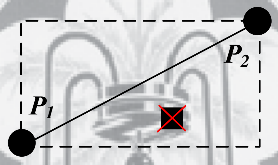

Figure 3.2: No obstacle corners exist in the rectangle area of edge (P_1, P_2).

2. *MTST-OARG* Construction: An MTST of OARG (denoted as *MTST-OARG*) is constructed by our new MTST generation scheme (MTST-H) as shown in Figure 3.1(c). We will prove that the corresponding time complexity is $O(n \log n)$.

3. *OARST* Transformation: An OARST is transformed from an MTST-OARG by a scheme called *MTST-OARG-Reduction* as shown in Figure 3.1(d).

3.2.1 Step 1: OARG Construction

A full description of OARG is given below, as well as the related definitions and proofs. In brief, OARG has three types of edges, namely k_1, k_2, and k_3.

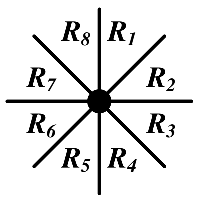

Figure 3.3: The octal regions of point P.

Definition 3.1 *Given an edge* e*, we correlate* e *with a rectangle whose diagonal line is* e *and boundaries are perpendicular to the x- or y-coordinate as shown in Figure 3.2. We define the area formed with dashed lines as the* ***rectangle area*** *of* e*.*

Definition 3.2 *For an edge* ***e****,* ***e*** *holding the* ***rectangle avoidance property*** *means that there is no obstacle corner in the rectangle area of* ***e****. An example is shown in Figure 3.2. Note that since a rectilinear edge has no rectangle area, a rectilinear edge holds the* ***rectangle avoidance property*** *if it does not intersect any obstacles.*

Definition 3.3 *For a graph* ***G****,* ***G*** *holding the* ***rectangle avoidance property*** *means that all edges of* ***G*** *hold the* ***rectangle avoidance property****.*

The k_1 edges take advantage of Zhou's spanning graph [51] and Lin's OASG [22]. Zhou et al. [51] divided the plane into eight octal regions for each point as shown in Figure 3.3; an edge connecting a point to its closest point in each region is a **possible candidate edge** of Zhou's spanning graph. By this selection, Zhou's spanning graph guarantees at least one RMST, and has linear space and loglinear construction time. However, Zhou's spanning graph does not consider obstacles such that edges of Zhou's spanning graph may intersect obstacles. On the other hand, Lin's OASG construction select an edge which holds the rectangle avoidance property and does not intersect any obstacles. By their selection, Lin's OASG contains all rectilinear shortest paths for all pairs of pin-vertices.

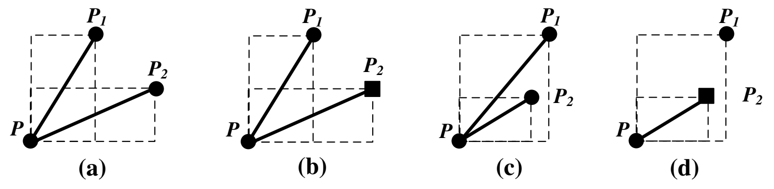

Figure 3.4: Illustrations of the quarter-based decision method. Circles represent terminals, and squares represent obstacle corners.

However, for each vertex, Lin's OASG contains $O(n)$ edges in the worst case such that Lin's OASG has $O(n^2)$ edges in the worst case. As a result, Lin's OASG construction takes at least $O(n^2)$ run time in the worst case, which loses the efficiency.

By our edge selection strategy, our k_1 edge construction applies Zhou's spanning graph [51] and considers obstacles at the same time. The edge selection strategy has three conditions as follows.

1. A possible candidate edge is an edge connecting a point with its closest point in one of its eight regions (the same as Zhou's spanning graph [51]).
2. A possible candidate edge will not be connected if the edge does not hold the rectangle avoidance property.
3. A possible candidate edge will not be connected if the edge intersects at least one obstacle.

First, the condition 1 can be satisfied by using a sweep-line algorithm to find all possible candidate edges similar to [51]. Second, we achieve the condition 2 by applying **a quadrant-based decision method** to all points and the corresponding possible candidate edges. Take the first **quarter** for example. As shown in Figure 3.4, we assume the closest points in R_1 and R_2 regions of a point P are P_1 and P_2 respectively. If neither rectangle

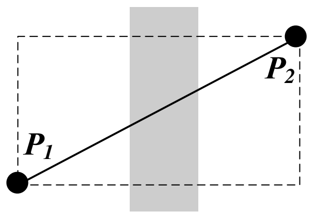

Figure 3.5: An obstacle intersects an edge (P_1, P_2) which holds the rectangle avoidance property, with no obstacle corners lying in the rectangle area of (P_1, P_2).

area of P_1 nor that of P_2 contains each other, both (P,P_1) and (P,P_2) are connected as shown in Figure 3.4(a) and Figure 3.4(b). Otherwise, the edge connection depends on whether the point contained by rectangle area of another is an obstacle corner or not. We assume that P_2 is contained by the rectangle area of P_1 as shown in Figure 3.4 (c) and (d). If P_2 is not an obstacle corner, both (P,P_1) and (P,P_2) are connected as shown in Figure 3.4 (c). Otherwise, we only connect (P,P_2) to achieve the condition 2 as shown in Figure 3.4(d). Third, for each possible candidate edge satisfying the condition 2, we can determine whether the edge satisfies the condition 3 in $O(1)$ time, which will be proved later by Lemma 3.1.

The k_2 edges consist of all obstacle boundaries. The k_3 edges are generated by connecting each terminal with the closest obstacle boundaries in four directions via perpendicular (to the x- or y-coordinate) lines. Take Figure 3.1(b) for an instance, there exists one k_3 edge starting from the topmost terminal in the down direction.

Lemma 3.1 *If the closest obstacles in the four directions (up, down, right and left) for all points are known, for an edge **e** which holds the rectangle avoidance property, it takes* $O(1)$ *time to determine whether **e** intersects obstacles or not.*

Proof: We assume that *e* is (P_1, P_2), and the x-coordinate of P_1 is not larger than that of P_2. We also assume that *e* has positive slope and *e* intersects at least one obstacle denoted as O_i. By the definition for the rectangle avoidance property, no obstacle corners of O_i lie in the rectangle area of *e* as shown in Fig 3.5. If O_i is on the right side of P_1, the closest obstacle O_r to P_1 in the right direction must intersect *e* as well. Otherwise, O_i is closer to P_1 than O_r such that there exists a contradiction. Similarly, if O_i is on the up side of P_1, the closest obstacle O_u to P_1 in the up direction must intersect *e* as well. As a result, we can determine whether *e* intersects obstacles or not in $O(1)$ time since the closest obstacles in the four directions (up, down, right and left) for all points are known.

□

Theorem 3.2 *The time complexity of OARG-construction scheme is* $O(n \log n)$.

Proof: First, we consider the k_1 edges construction. [51] proved that finding the closest neighbors of all octal regions for all points takes $O(n \log n)$ run time. For the selection strategy of the k_1 edges, it is sufficient to consider the determination of edges which intersects obstacles. From Lemma 3.1, for each edge, the process takes merely $O(1)$ run time, once the nearest four-direction obstacles for all points are known. Shen et al. [36] presented a scheme to figure out the nearest four-direction obstacles for all points with $O(n \log n)$ run time complexity. Therefore, the run time complexity to generate the k_1 edges is $O(n \log n)$. It is easy to verify that generating the k_2 edges costs $O(n)$ time. As for the k_3 edges, by exploiting Shen's method to find the nearest four-direction obstacles for all points, the time complexity is limited to $O(n \log n)$. From the discussion, we conclude that the time complexity of OARG-construction scheme is $O(n \log n)$. □

According to k_1, k_2, k_3 edges, we directly have the following two lemmas.

Lemma 3.3 *Both the sizes of edges and vertices of OARG are* $O(n)$.

Lemma 3.4 *OARG holds the rectangle avoidance property*

Aside from the time and space complexity, OARG has high solution quality and thus is very suitable for an MTST-based algorithm. In [51], the authors claimed that their spanning graph guarantees at least one RMST among input vertices. Therefore, our application of Zhou's spanning graph will generate local OARMSTs. Since OARG contains many local OARMSTs, an MTST of OARG will be a good initial solution for the OARSMT problem. Noticeably, Zhou's spanning graph cannot handle obstacles; the correctness and the efficiency of the application fully depends on our proposed rectangle avoidance property and quadrant-based decision method.

Overall, OARG substantially helps the development of an efficient and effective MTST-based algorithm for the OARSMT problem. Compared to Lin's OASG [22], which has $O(n^2)$ edges, the linear space of OARG significantly increases the efficiency. Although OARG cannot guarantee an OARMST (Lin's OASG [22] can), the solution quality of OARG is still good since OARG contains local OARMSTs. Furthermore, using the linear space and the rectangle avoidance property, the solution quality of OARG can be very significantly improved (Section 3.2.3 and Section 3.3) and then will be comparable to that of [22]. Compared to the Shen's OASG [36], which also has $O(n)$ edges, OARG likely contains more local OARMSTs such that the solution quality of OARG could be much better. This is because the OARG construction applies Zhou's Spanning Graph to divide a plane into eight octal regions instead of four quadrant regions in [36].

3.2.2 Step 2: MTST-OARG Cosntruction

In this step, we construct an MTST over our OARG (denoted as MTST-OARG) by our new MTST generation scheme, namely MTST-H. The time complexity of MTST-H is $O(|E|\log|E|)$. Since both the vertex size and edge size of OARG are $O(n)$ (by Lemma 3.3), our MTST-OARG construction takes $O(n \log n)$ run time by directly applying MTST-H to OARG. Mehlhorn [31] also developed the KM algorithm to construct an MTST with $O(|E| + |V|\log|V|)$ time complexity. Although the KM algorithm has better time complexity than the MTST-H algorithm in general cases, since $|E|$ and $|V|$ are both of $O(n)$ in OARG, the KM algorithm also takes $O(n \log n)$ time complexity for the MTST-OARG construction. More importantly, in practice, the MTST-H algorithm probably performs more efficiently than the KM algorithm for the MTST-OARG construction. This is because the MTST-H algorithm retrieves only possible candidates of an MTST instead of all vertices. In general, those possible candidates of an MTST could be much fewer than all vertices.

An example for the procedure of MTST-H is shown in Figure 3.6. The problem instance is on the upper-left part of each step in Figure 3.6, where circles denote terminals and squares denote nonterminals. Basically, MTST-H consists of the operations of two specific heaps. Brief explanations of these heaps are as follows.

The first heap is called *nonterminal reachable heap* (n-heap) (the upper-right part of each step in Figure 3.6), and it is used to find the closest terminal for each nonterminal, by which we can generate the possible virtual edges for an MTST. For each nonterminal, an element of n-heap (called an n-element) exists to represent the latest information about the closest terminal of this nonterminal and the path between them. In Table 3.1, some

Table 3.1: Notations For MTST-H

$t(w)$	the closest terminal of w.
$d(w)$	the length of a shortest path between w and $t(w)$.
$f(w)$	the vertex directly connected to w on the shortest path between w and $t(w)$.

functions are outlined to indicate the current information mentioned above. For each nonterminal w, the n-element is a 4-tuple $< w, f(w), t(w), d(w) >$, and the key of n-heap is $d(w)$. For example, in Figure 3.6(d), $< g, f, b, 5 >$ indicates, at that time, b is the closest terminal to g through a path of length 5, and g is directly connected to vertex f in this path.

The second heap is called *virtual edge heap* (v-heap) (the lower-right part of each step in Figure 3.6), and it is used to generate a minimum spanning tree over the virtual edges. Each element of the v-heap (called a v-element) is a 5-tuple <*from_terminal, to_terminal, generated_from_vertex, generated_to_vertex, length*>, where length serves as the key. In Figure 3.6(e), $< a, b, e, f, 16 >$ represents that there exists a path between terminals a and b; this path is generated from edge $< e, f >$, and is of length 16.

Algorithm 1 MTST-H

```
1:  for each nontermianl $w \in$ V do
2:    set ($f(w), t(w), d(w)$) as ($\Phi$, $\Phi$, $\infty$),
3:      and insert $< w, f(w), t(w), d(w) >$ into n-heap
4:  for each terminal $v \in$ V do
5:    for each $(u, v) \in$ E do
6:      if $u$ is terminal then
7:        insert $< u, v, u, v, |(u, v)| >$ to v-heap
8:      else if $|u, v| < d(u)$ then
9:        $f(u) \leftarrow v$
10:       $t(u) \leftarrow v$
11:       $d(u) \leftarrow |(u, v)|$
12:       update n-heap
13: repeat
14:   if $MIN$(v-heap) $\leq 2\times$ $MIN$(n-heap) then
15:     do V-Heap-EXTRACT-MIN
16:   else
17:     do N-Heap-EXTRACT-MIN
18: until An MTST has been constructed or the two heaps are empty
```

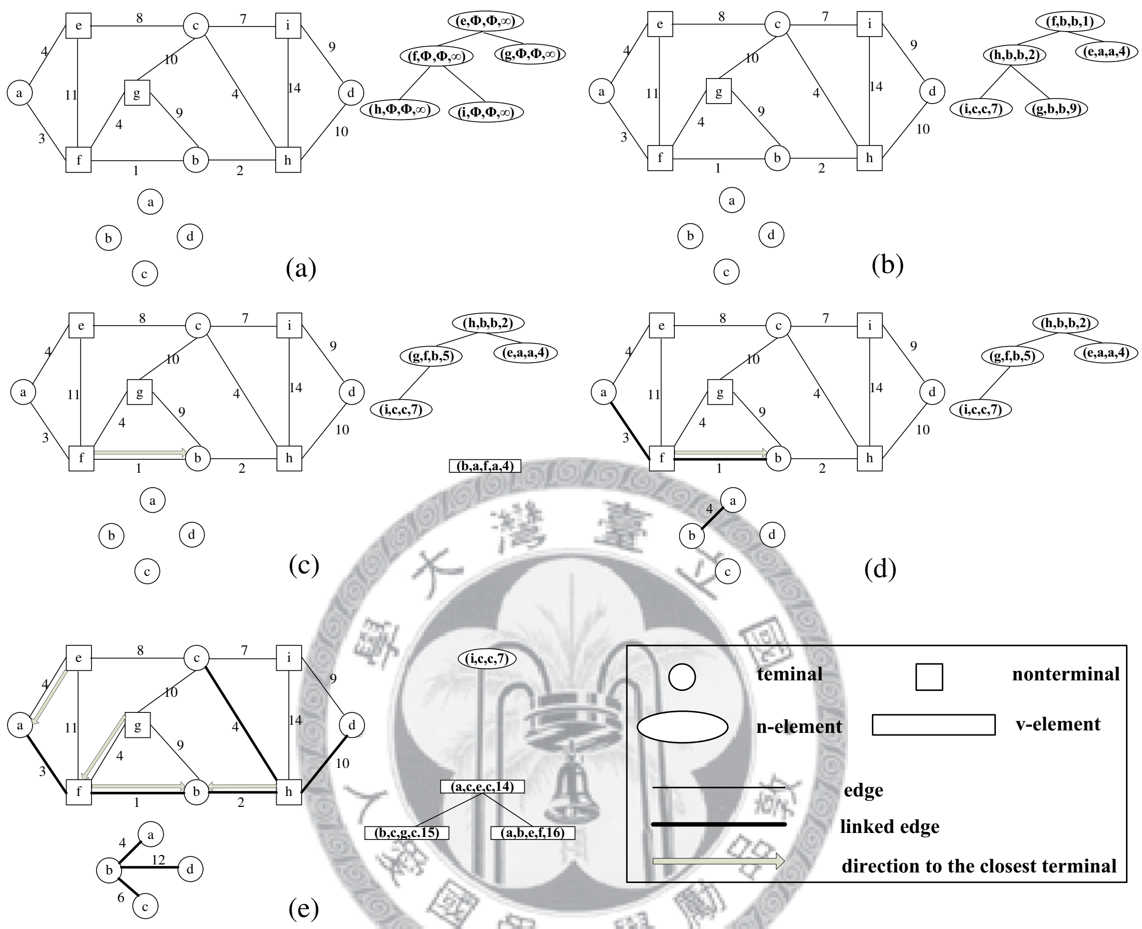

Figure 3.6: A process sketch of MTST-H.

Algorithm 2 V-Heap-Extract-Min

```
1: extract the min element $< f\_t, t\_t, f\_v, t\_v, l >$ from v-heap
2: **if** $f\_t$ and $t\_t$ are in different components **then**
3:     Link all edges in the virtual edge
4:     union the two components which contain $f\_t$ and $t\_t$
```

Algorithm 3 N-Heap-Extract-Min

```
1: extract the min element $< w, f(w), t(w), d(w) >$ from n-heap
2: mark $w$
3: for each $(u, w) \in$ E do
4:    if $u$ is a terminal then
5:       insert $< t(w), u, w, u, |(u, w)| + d(w) >$ to v-heap
6:    else
7:       if $u$ is marked and $t(u) \neq t(w)$ then
8:          set $l = d(w) + d(u) + |(u, w)|$
9:          insert $< t(w), t(u), w, u, l >$ to v-heap
10:      else if $u$ is unmarked and $d(w) + |(u, w)| < d(u)$ then
11:         $f(u) \leftarrow w$,
12:         $t(u) \leftarrow t(w)$
13:         $d(u) \leftarrow d(w) + |(u, w)|$
14:         update n-heap
```

We use pseudo-codes, namely Algorithm 1, 2, and 3, to clarify the process of MTST-H. Algorithm 1 operates n-heap and v-heap to generate an MTST. Lines 1–13 initialize n-heap and v-heap. In details, lines 6–7 generate direct virtual edges, and lines 9–12 updates n-elements. Lines 14–16 operate N-Heap-EXTRACT-MIN and V-Heap-EXTRACT-MIN to generate virtual edges and connect virtual edges respectively. Algorithm 2 connects a virtual edge when the connection will not generate a cycle. Algorithm 3 updates n-elements and generates virtual edges. In details, line 5 and lines 8–9 generate virtual edges; lines 11–14 update n-elements.

Line 14 of Algorithm 1, "if MIN*(v-heap)* $\leq 2\times MIN$*(n-heap)*", is a very important constraint of MTST-H algorithm. The basic idea is that assuming the length of the longest virtual edge of an MTST is $\boldsymbol{l}$, a possible candidate of an MTST must have a path leading to a terminal with length smaller or equal to $\lceil \frac{l}{2} \rceil$; otherwise, the length of any virtual edge including this vertex between terminals is larger than $\boldsymbol{l}$, i.e., not a virtual edge of an MTST. To implement this idea, the MTST-H algorithm never retrieves a vertex which has a path leading to a terminal with length of l' if there exists a virtual edge with length smaller or equal to 2*l', i.e., the constraint, "MIN*(v-heap)* $\leq 2\times MIN$*(n-heap)*". The correctness will be supported by Lemma 3.6. As a result, the MTST-H algorithm only retrieves possible candidates of an MTST, and thus it may only retrieve a smaller subset of all vertices, which will significantly increase the efficiency.

As illustrated in Figure 3.6(a), lines 1-3 of Algorithm 1 first initialize n-elements of nonterminals and insert them into n-heap. As shown in Figure 3.6(b), lines 4-12 of Algorithm 1 extend all edges of terminals to update elements of n-heap and add direct virtual edges into v-heap. After that, the lines 13-18 of Algorithm 1 manipulate v-heap and n-heap repeatedly until an MTST is generated. As shown in Figure 3.6(c), N-Heap-Extract-Min has been executed such that lines 11-14 of Algorithm 3 update (g,$f(g)$,$t(g)$,$d(g)$); lines 8-9 of Algorithm 3 insert a virtual edge $< b, a, f, a, 4 >$ into v-heap. After that, Algorithm 2 will be executed. As shown in Figure 3.6(d), $< b, a, f, a, 4 >$ is popped: two terminals a and b are connected by the virtual edge (a, b) in the lower-left part, and the corresponding path $a \leftrightarrow f \leftrightarrow b$ is constructed in the upper-left part. By repeating the above operations, an MTST forms eventually as shown in Figure 3.6(e).

Below, we will prove the time complexity and correctness of MTST-H algorithm.

Lemma 3.5 *Whenever an n-element representing a nonterminal w is extracted, $t(w)$ corresponds to the closest terminal to w.*

Proof: If the path between w and the closest terminal to w is the edge that directly connects them, this Lemma can be easily verified by lines 4-12 of Algorithm 1. Thus we only need to discuss the condition in which the path between w and the closest terminal to w includes at least one other nonterminal. Because there is no nonterminal u in an n-element such that $d(u) < d(w)$ when w is extracted by MTST-H. By definition there exist no negative-weight edges, the remaining nonterminals in the n-heap cannot generate any paths shorter than the path w belonging to afterward. Therefore, $t(w)$ is the closest terminal to w while a nonterminal w is extracted. □

Lemma 3.6 *The N-Heap-Extract-Min cannot generate a virtual edge whose length is smaller than 2*MIN(n-heap).*

Proof: Suppose the N-Heap-Extract-Min generates a virtual edge (a, b) whose length $\boldsymbol{l}$ is smaller than 2*MIN*(n-heap)*. Without loss of generality, assume that (a, b) is generated by extracting an n-element $< w, f(w), t(w), d(w) >$, and $t(w)$ is a. By Lemma 3.5, it is clear that distance(w,b) must be at least $d(w)$; otherwise, $t(w)$ will be b. As a result, we have l = distance(a,w) + distance(w,b) $\geq$ 2*$d(w)$ = 2*MIN*(n-heap)*. There exists a contradiction. □

Lemma 3.7 *MTST-H adds a virtual edge e to construct an MTST only if n-heap cannot generate any virtual edges shorter than e afterward.*

Proof: Directly from line 14 of Algorithm 1, Lemma 3.6, and Algorithm 2. □

Lemma 3.8 *A MTST can be constructed by MTST-H.*

Proof: First, the sequence order of all virtual edges connected by MTST-H are labeled $e_i, i = 1, 2, \ldots, |T| - 1$ ($|T|$ denotes the terminal size). By Algorithm 2 and Lemma 3.7, we have $\forall i \leq j, |e_i| \leq |e_j|$. Second, all virtual edges of the MTST are labeled e'_i, $i = 1, 2, \ldots, |T| - 1$ in increasing order such that $\forall i \leq j, |e'_i| \leq |e'_j|$. Assume there exists a label j such that $j = min(k \mid |e_k| \neq |e'_k|, k \in 1, 2, \ldots, |T| - 1)$. Because of the assumption for e'_j (e'_j is a virtual edge of MTST in increasing order), the inequality $|e_j| >$

$|e'_j|$ must hold. By Lemma 3.7, a virtual edge with length of $|e'_j|$ will not be generated by MTST-H. Without loss of generality, we assume that e'_j is $(1, 2)$. Since $(1, 2)$ can not be generated by MTST-H, there exists at least one nonterminal in the corresponding path belonging to neither terminal 1 nor 2. We assume that the nonterminal is labeled 4 and belongs to terminal 3. As a result, neither path (1, 4) nor path (2, 4) is shorter than path (3, 4) by Lemma 3.5, which implies that both the virtual edges (1, 3) and (2, 3) are smaller than (1, 2). Based on the cycle property of a minimum spanning tree [6], (1, 2) cannot be a virtual edge of the MTST. There exists a contradiction. □

Theorem 3.9 *The time complexity of MTST-H is* $O(|E| \log |E|)$.

Proof: The size of virtual edges, generated by a terminal or an extracted nonterminal, is at most the same as its degree. Thus the number of virtual edges generated by MTST-H is $O(E)$. Consequently, the run time complexity of v-heap operations is $O(|E| \log |E|)$. Because there are $|E|$ edges, the number of nonterminal updates is at most $O(E)$. Thus the number of n-heap operations is $O(|E| \log |V|)$. We can conclude that the time complexity of MTST-H is $O(|E| \log |E|)$. □

Combining Lemma 3.3 and Theorem 3.9, constructing an MTST-OARG takes $O(n \log n)$ time by using MTST-H. On the other hand, Lemma 3.8 and Algorithm 1 guarantee that MTST-H will terminate whenever an MTST is generated, which implies that MTST-H does not always go through the entire graph. It should also be noted that MTST-H use operations of heaps, which is well-known more efficient than operations of Fibonacci-Heaps for practical usage. Hence, in practice, MTST-H could be more efficient than KM algorithm for the MTST-OARG construction.

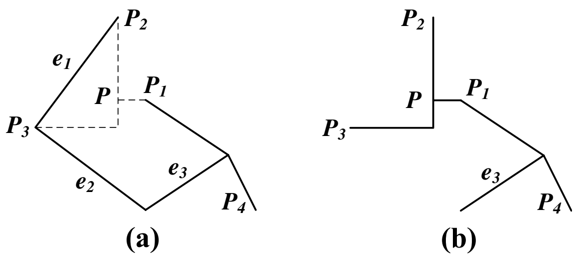

Figure 3.7: An example of the edge substitution technique in [2]

3.2.3 Step 3: OARST Transformation

In this step, we transform an MTST-OARG into an OARST by the scheme called MTST-OARG-Reduction. This scheme can transform an MTST-OARG to an OARST in $O(n \log n)$ run time and reduce total wirelength at the same time. In brief, MTST-OARG-Reduction is based on Borah's edge-based algorithm [2], Zhou's observations about Borah's method [50], and the rectangle avoidance property of OARG. Note that neither Borah's edge-based algorithm nor Zhou's observations consider obstacles such that the direct extension of their method may result in infeasible solution for the OARST construction. As a result, the correctness and efficiency of MTST-OARG-Reduction is based on the rectangle avoidance property and loglinear space of OARG.

Borah et al. [2] proposed a method for finding a rectilinear Steiner tree from a rectilinear minimum spanning tree via an edge substitution technique. As shown in Figure 3.7(a), node P_1 will be connected to the nearest point on the rectangle area of edge e_1. If we remove e_1 and connect edges (P_1, P), (P_2, P), (P_1, P_3), a circuit will form. Suppose the longest edge in this circuit is e_2, then we remove e_2 as shown in Figure 3.7(b). The difference between (a) and (b) is called the gain of pair $< e_1, P_1 >$. Zhou [50] observed some problems in Borah's scheme and provided a method to compute gains of $<$ *edge,point* $>$ pairs efficiently. Zhou represented a minimum spanning tree (as shown in Figure 3.8 (a))

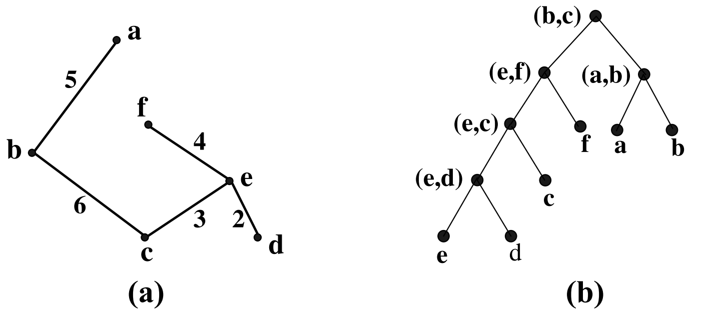

Figure 3.8: Zhou's observations [50]. (a) a minimum spanning tree. (b) a binary tree derived from (a). For each two leaf nodes of (b), their least common ancestor is the longest edge between them in the minimum spanning tree

by a binary tree (as shown in Figure 3.8(b)). The binary tree is constructed by Kruskal Algorithm [6] which can find a minimum spanning tree efficiently, and Zhou proved that the least common ancestor (LCA) of two leaf nodes is the longest edge between them in the circuit which is formed by connecting these two leaf nodes.

Definition 3.4 *Given a graph $G(V, E)$, for two vertices v_1 and $v_2 \in V$, if $(v_1, v_2) \in E$, v_2 is a **neighbor** of v_1, and vice versus.*

Our scheme can handle obstacles based on the rectangle avoidance property of OARG; the correctness of our scheme will be proved in Theorem 3.11. Furthermore, for the efficiency consideration, we only consider the neighbors of each edge instead of the visible-points defined in [2]. Theorem 3.12 will verify the efficiency of MTST-OARG-Reduction.

Basically, MTST-OARG-Reduction computes gains by finding the longest edge for each $<$ *edge, neighbor-point* $>$ pair, where the edge belongs to MTST-OARG and the neighbor-point is a neighbor of one endpoint of this edge on OARG. As a result, we need to construct the binary tree representation for the MTST-OARG. We first remove the non-terminals not in the MTST-OARG, as well as the edges connected to them, from OARG.

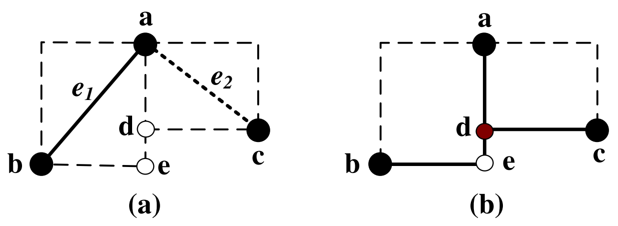

Figure 3.9: An example about the Steiner point when applying edge-substitution.

After that, we re-construct a minimum spanning tree over the reduced graph via Kruskal's algorithm and represent this tree by Zhou's binary tree (as shown in Figure 3.8 (b)). Then we recursively remove nonterminals with edge degree being one. It is clear that the cost of the resulting MTST is no larger than the cost of the MTST-OARG, the later correlates with the worst case. Subsequently, we apply Tarjan's off-line least-common-ancestor algorithm [6] to this binary tree in search for the corresponding longest edge of each $<$ *edge, neighbor-point* $>$ pair. By the results, the gains of all $<$ *edge, neighbor-point* $>$ pairs can be computed. Finally, we sort all $<$ *edge, neighbor-point* $>$ pairs by their gains in decreasing order, do edge-substitutions to the pairs with positive gain subsequently, and complete the OARST transformation.

Lemma 3.10 *An oblique edge* **e** *of OARG can be transformed to rectilinear edges without intersecting any obstacles.*

Proof: Since all edges of OARG hold the rectangle avoidance property, there is no obstacle corner in the rectangle area of e. As a result. e can be replaced by two boundaries of its rectangle area which are perpendicular to each other. $\square$

Theorem 3.11 *MTST-OARG-Reduction can transform a given MTST-OARG into an obstacle-avoiding rectilinear Steiner tree.*

Proof: There are only two cases: 1) Rectilinear edges are generated by directly transforming the original edge; 2) Edge substitution is applied. First, since OARG has no edges intersecting the obstacles, by Lemma 3.10, the rectilinear edges in Case 1 will not intersect any obstacles. Second, for each edge e, edge substitution is applied to e and a neighbor of e, thus the Steiner points must fall in the boundaries of the rectangle areas, which belong to e and the edge connecting e with the neighbor in OARG, as shown in Figure 3.9. Since all edges of OARG hold the rectangle area property, there is no obstacle corner in those rectangle areas, which implies that the rectilinear edges in Case 2 will not intersect any obstacles. To conclude, an MTST-OARG can be transformed into an obstacle-avoiding rectilinear Steiner tree via MTST-OARG-Reduction. □

Theorem 3.12 *The time complexity of applying MTST-OARG- Reduction to MTST-OARG is* $O(n \log n)$.

Proof: Because we merely consider the $<$ *edge, neighbor* $>$ pairs and the degree of each point is constant, the total number of queries for least common ancestors is $O(n)$. Since the size of the binary tree and the number of queries are both $O(n)$, the run time complexity of Tarjan's off-line least-common-ancestor algorithm [6] for our queries is $O(n\alpha(n))$. Moreover, $\alpha(n)$ is the inverse of the Ackermann's function and grows very slowly, thus we can consider it as constant for all practical cases (see [2] for similar arguments). Thus the sorting part dominates the run time, and the time complexity is $O(n \log n)$. □

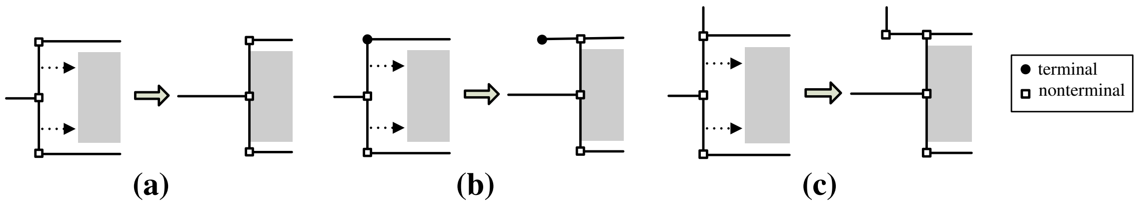

Figure 3.10: Examples for moving a CLS. (a) three nonterminals. (b) one terminal and two nonterminals. (c) three nonterminals and one connected to other line segment in the same direction.

3.3 Refinement

In this section, we propose a local refinement scheme for reducing total wirelength of an OARST. Unlike previous U-shaped pattern refinement method in [22], our refinement scheme considers more general cases. Furthermore, we also use sorting to make our scheme more greedy such that the wirelength may be reduced more. By using the essence of our local refinement scheme, the local redundant wirelength of the OARST constructed by our 3-step algorithm can be significantly reduced in $O(n \log n)$ run time. Note that we use our refinement scheme after our 3-step algorithm.

Definition 3.5 *For an OARST, a **simple line segment** is a line segment crossing no vertices except the two endpoints, and an **i-CLS** is a collection of **i** simple line segments whose directions are all the same.*

Our scheme is based on our observation which will be described below. For a CLS of an OARST, if the numbers of line segments connected to this CLS in opposite sides are different, it is possible to move the CLS to reduce the total wirelength. For example, for the vertical CLS in the left part of Figure 3.10(a), the number of left side line segments is 1, but the number of right side line segments is 2. Hence, we can move this CLS right to reduce the wirelength. However, there are two notable constraints for considerations. First, terminals of a CLS may affect the refinement. As shown in Figure 3.10(b), since terminals can not be moved, the movement will not reduce any wirelength. Second, line segments connected to the two endpoints of a CLS in the same direction may affect the result, too. As shown in Figure 3.10(c), we cannot reduce the wirelength by moving the vertical CLS.

Definition 3.6 *For an OARST, the* ***gain*** *of a CLS is the possible reduced wirelength by moving this CLS.*

Our scheme is generated intuitively from our observation by following steps:

1. Compute gains for all CLSs of the OARST and sorts them in decreasing order.
2. Do movements in the sorting sequence and remove the related gains (whose CLSs connected to the moved CLSs) from the sorting sequence.
3. Construct the reduced OARST finally.

Since the number of *i*-CLSs is at most *e*-*i* (*e* is the edge number of the target OARST), the number of CLSs is $\sum_{i=1}^{e-1}(e-i) = O(e^2)$. For an *i*-CLS, the time to compute the gain is $O(i)$. As a result, the run time for computing the gains for all CLSs is $\sum_{i=1}^{e-1}(e-i)O(i) = O(e^3)$. Then, the sorting for $O(e^2)$ CLSs takes $O(e^2 \log e)$ run time. Thus, the direct scheme costs $O(e^3)$ run time in the worst case.

However, our main concept is to construct an $O(n \log n)$ method for the OARSMT problem such that our original scheme is not suitable. As a result, we will not consider all CLSs for efficiency consideration.

Table 3.2: Number of CLSs for Our OARST.

TERM #	OBS #	$\frac{2-CLS}{1-CLS}$	$\frac{3-CLS}{1-CLS}$
50	250	20.9%	2.7%
100	500	21.2%	2.7%
500	2500	21.7%	2.7%
1000	5000	2.8%	2.9%
Average		21.4%	2.75%

In order to reduce the number of gain computations, we take experiments about the number of CLSs. Each kind of testcases has 20 samples; terminals and obstacles are randomly generated. In Table 3.2, experimental results show the number of 3-CLSs is 2.75 % of the number of 1-CLSs. As a result, it may be sufficient to consider only 1-CLSs , 2-CLSs and 3-CLSs for efficiency issue. Since this scheme doesn't consider all kinds of CLSs, the time complexity is reduced drastically. The number of CLSs decreases to $O(n)$, and the computations of gains for CLSs scale as $O(n)$, too. Furthermore, sorting dominates the run time complexity and costs $O(n \log n)$.

To conclude, in this paper, we use our 3-step algorithm and our local refinement to construct an OARST in $O(n \log n)$ run time.

3.4 Experimental Results

We implemented our algorithm in C language on a PC with 3GHz Pentium processor and 2GB memory under Linux operation system. There are 22 benchmark circuits given from [22] (five industrial testcases (ind1-ind5) from Synopsys, twelve testcases from [8] (rc1-rc12), and five random testcases (rt1-rt5) from [22]).

We compared our algorithm with those presented in [8,22,28,36,44]. All the results are directly quoted from [8,22,28,44]. The results of [8] are generated by the algorithm performed on a Unix workstation with 2.66GHz CPU and 1GB memory. The results of [44] are generated by the algorithm performed on a Sun Blade2000 workstation with 1200MHz CPU and 8GB memory. The results of [28] are generated by the algorithm performed on a Redhat Linux workstation with 2.1 Dual Core CPU and 2GB memory. The results of [36] and [22] are generated by the algorithms performed on a 2GHz AMD-64 machine with 8GB memory under Ubuntu 6.06 operation system. It is also well-known that the memory access performance of workstation is much better than normal PC-based computer even the processor of PC outperforms workstation processor.

Table 3.3 shows the solution quality (wirelength) comparison of these algorithms. "HPBB" means the half-perimeter of the bounding box of all terminals and is the lower bound of optimal solution. "-" means that the result is not available. Considering the differences from the HPBB, the respective average improvements on the total wirelength are 24.39%, 4.30 % , 7.01%, 6.10% and 1.47%, compared with the algorithms in [8], [44], [36], [28], and [22]. Since HPBB is the lower bound of optimal solution for the OARSMT problem, if we consider the difference from an optimal solution, the improvement will be larger. Considering the percentages of the reduced wirelength (solution quality improvement), our algorithm is very competitive.

Table 3.4 lists the execution time of each algorithm. Although the execution environments of those algorithm are different, the effect on the execution time under each environment is constant, which implies that the execution environment does not affect the time complexity. As a result, we can use "time ratio," the ratio of different works versus our work on the same testcase, to analyze the difference of the time complexity.

To the best of our knowledge, [8] is the fastest method for the OARSMT problem. Since the time ratios (between [8] and ours) in Table 3.4 does not increase with input

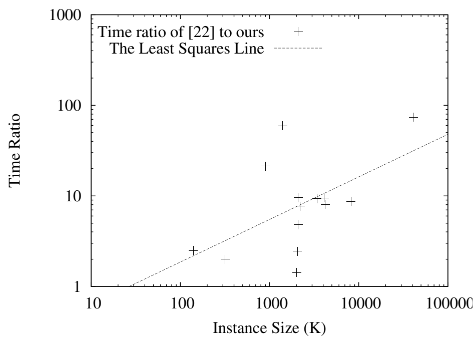

Figure 3.11: The time ratio of [22] to ours is plotted as a function of n.

size, the empirical time complexity of our algorithm is similar to that of [8]. Furthermore, our wirelength improvement over [8] is 24.39%/29.48% in average and can be up to 52.92%/55.01% (rc12), which is very significant.

To the best of our knowledge, [22] is the best work with regard to wirelength according to the literal experimental results. The theoretical worst case run time complexity of [22] is $O(n^3)$ while our algorithm to be $O(n \log n)$. Table 3.4 shows that the time ratios of [22] to ours seem to increase with the input size. As shown in Figure 3.11, the time ratios of [22] to ours are plotted as a function of the input size k. By the least squares fitting on the log-log-axes, the respective slope of the fitting line is 0.47, which implies that the empirical complexity of time ratios (the empirical time complexity of [22] to that of ours) is $O(n^{0.47})$. When the input size increases, the $O(n^{0.47})$ difference in empirical time complexity will make our algorithm more competitive. Despite of the empirical time complexity, it should be noted that our solution quality is also improved by 0.86%/1.47% on average compared with [22].

To conclude, according to the analysis of experimental results in Table 3.3 and Table 3.4, our algorithm outperforms all state-of-the-art works in wirelength and has similar speed to the fastest work.

Table 3.3: The Comparison On Wirelength.

Test	HPBB	Total Wirelength						Improvement (%) ($\frac{X-G}{X} / \frac{X-G}{X-A}$)				
Cases	(A)	(B) [8]	(C) [44]	(D) [36]	(E) [28]	(F) [22]	Ours (G)	X=B	X=C	X=D	X=E	X=F
ind1	501	-	-	646	649	632	636	-	-	1.55 / 6.90	2.00 / 8.78	-0.63 / -3.05
ind2	8200	-	-	10100	10100	9600	9600	-	-	4.95 / 26.32	4.95 / 26.32	0.00 / 0.00
ind3	498	-	-	623	623	613	613	-	-	1.61 / 8.00	1.61 / 8.00	0.00 / 0.00
ind4	705	-	-	1121	1131	1121	1116	-	-	0.45 / 1.20	1.33 / 3.52	0.45 / 1.20
ind5	732	-	-	1392	1379	1364	1364	-	-	2.01 / 4.24	1.09 / 2.32	0.00 / 0.00
Average								-	-	2.11 / 9.33	2.20 / 9.79	-0.04 / -0.37
rc01	17890	30410	27250	27730	27540	26900	25980	14.57 / 35.38	4.66 / 13.57	6.31 / 17.78	5.66 / 16.17	3.42 / 10.21
rc02	19470	45640	43320	42840	42030	42210	42070	7.82 / 13.64	2.89 / 5.24	1.80 / 3.29	-0.10 / -0.18	0.33 / 0.62
rc03	19380	58570	56500	56440	56070	55750	54630	6.73 / 10.05	3.31 / 5.04	3.21 / 4.88	2.57 / 3.92	2.01 / 3.08
rc04	19850	63340	61090	60840	59550	60350	59270	6.43 / 9.36	2.98 / 4.41	2.58 / 3.83	0.47 / 0.71	1.79 / 2.67
rc05	19600	83150	76870	76970	76320	76330	75410	9.31 / 12.18	1.90 / 2.55	2.03 / 2.72	1.19 / 1.60	1.21 / 1.62
rc06	19593	149750	84327	86403	87432	83365	82300	45.04 / 51.82	2.40 / 3.13	4.75 / 6.14	5.87 / 7.56	1.28 / 1.67
rc07	19882	181470	115461	117427	117855	113260	111752	38.42 / 43.15	3.21 / 3.88	4.83 / 5.82	5.18 / 6.23	1.33 / 1.61
rc08	19803	202741	122574	123366	124852	118747	116646	42.47 / 47.06	4.84 / 5.77	5.45 / 6.49	6.57 / 7.81	1.77 / 2.12
rc09	19964	214850	120017	119744	120554	116168	113781	47.04 / 51.86	5.20 / 6.23	4.98 / 5.98	5.62 / 6.73	2.05 / 2.48
rc10	19900	198010	172490	171450	168859	170690	167540	15.39 / 17.11	2.87 / 3.24	2.28 / 2.58	0.78 / 0.89	1.85 / 2.09
rc11	19984	250570	238377	238111	235795	236615	234097	6.57 / 7.14	1.80 / 1.96	1.69 / 1.84	0.72 / 0.79	1.06 / 1.16
rc12	65422	1723990	786731	843529	852401	789097	811620	52.92 / 55.01	-3.16 / -3.45	3.78 / 4.10	4.78 / 5.18	-2.85 / -3.11
Average								24.39 / 29.48	2.74 / 4.30	3.64 / 5.45	3.28 / 4.78	1.27 / 2.19
rt01	1363	-	-	2438	-	2267	2259	-	-	7.34 / 16.65	-	0.35 / 0.88
rt02	16280	-	-	51981	-	48441	48377	-	-	6.93 / 10.09	-	0.13 / 0.20
rt03	1996	-	-	8783	-	8368	8323	-	-	5.24 / 6.78	-	0.54 / 0.71
rt04	1985	-	-	10619	-	10306	10221	-	-	3.75 / 4.61	-	0.82 / 1.02
rt05	8097	-	-	55557	-	53993	53745	-	-	3.26 / 3.82	-	0.46 / 0.54
Average								-	-	5.30 / 8.39	-	0.46 / 0.67
Total Average								24.39 / 29.48	2.74 / 4.30	3.77 / 7.01	3.02 / 6.10	0.86 / 1.47

Table 3.4: The Comparison On CPU Time.

Test Cases	Term #	Obs #	input size (k)	CPU Time (second)						Time Ratio ($\frac{X}{G}$)				
				(B) [8]	(C) [44]	(D) [36]	(E) [28]	(F) [22]	Ours (G)	(X=B)	(X=C)	(X=D)	(X=E)	(X=F)
ind1	10	32	138	-	-	< 0.01	0.01	< 0.01	0.001	-	-	-	10	-
ind2	10	43	182	-	-	< 0.01	0.01	< 0.01	0.001	-	-	-	5	-
ind3	10	50	210	-	-	< 0.01	0.01	< 0.01	0.002	-	-	-	5	-
ind4	25	79	341	-	-	< 0.01	0.02	< 0.01	0.004	-	-	-	5	-
ind5	33	71	317	-	-	< 0.01	0.02	0.01	0.004	-	-	-	5	2.5
rc1	10	10	50	0.002	<0.01	0.01	< 0.01	<0.01	0.001	2	-	-	10	-
rc2	30	10	70	0.003	<0.01	0.01	< 0.01	< 0.01	0.001	3	-	-	10	-
rc3	50	10	90	0.004	<0.01	0.01	< 0.01	<0.01	0.001	4	-	-	10	-
rc4	70	10	110	0.004	<0.01	0.02	< 0.01	< 0.01	0.002	2	-	-	10	-
rc5	100	10	140	0.004	<0.01	0.02	0.01	0.01	0.003	1.33	-	3.33	6.67	3.33
rc6	100	500	2100	0.057	0.31	0.17	0.14	0.24	0.038	1.5	8.16	4.47	3.68	6.32
rc7	200	500	2200	0.062	0.36	0.3	0.15	0.43	0.042	1.48	8.57	7.14	3.57	10.24
rc8	200	800	3400	0.095	0.53	0.48	0.27	0.83	0.065	1.46	23.54	7.38	4.15	12.77
rc9	200	1000	4200	0.129	1.8	0.64	0.36	0.91	0.085	1.52	21.18	7.53	4.24	10.71
rc10	500	100	900	0.026	0.27	0.27	0.08	0.62	0.021	1.24	12.86	12.86	3.81	29.52
rc11	1000	100	1400	0.037	0.81	0.95	0.15	3.15	0.039	0.95	20.77	24.36	3.84	80.77
rc12	1000	10000	41000	2.823	4.2	65.73	5.93	118.52	1.292	2.18	3.25	50.87	4.59	91.73
rt1	10	500	2010	-	-	0.06	-	0.06	0.032	-	-	1.88	-	1.88
rt2	50	500	2050	-	-	0.11	-	0.11	0.034	-	-	3.24	-	3.24
rt3	100	500	2100	-	-	0.16	-	0.47	0.036	-	-	4.44	-	13.06
rt4	100	1000	4100	-	-	0.34	-	0.95	0.077	-	-	4.42	-	12.34
rt5	200	2000	8200	-	-	1.42	-	2.06	0.181	-	-	7.85	-	11.38

Chapter 4

Path-Based Framework

For the OARSMT problem, this chapter proposes a new framework to develop an $O(n \log n)$-time algorithm with the same theoretical optimality guarantee with [22], which originally takes $O(n^3)$ time. Unlike most existing approaches discussed in Section 1.3, the new framework directly generates essential solution components without constructing a routing graph or generating invalid initial solutions. Through our analysis of the geometry mapping on previous works [22, 28], the algorithm generates only $O(n)$ critical paths, and those critical paths still guarantee the existence of an optimal solution for a number of specific cases. The algorithm also increases the overlapping between different paths for improving the wirelength, and performs an $O(n \log n)$-time dynamic local refinement scheme. Experimental results show that compared with [21], which achieves the best solution quality, our algorithm achieves 50.1 times speedup on average, while the resulting wirelengths are only 1.1% longer on average.

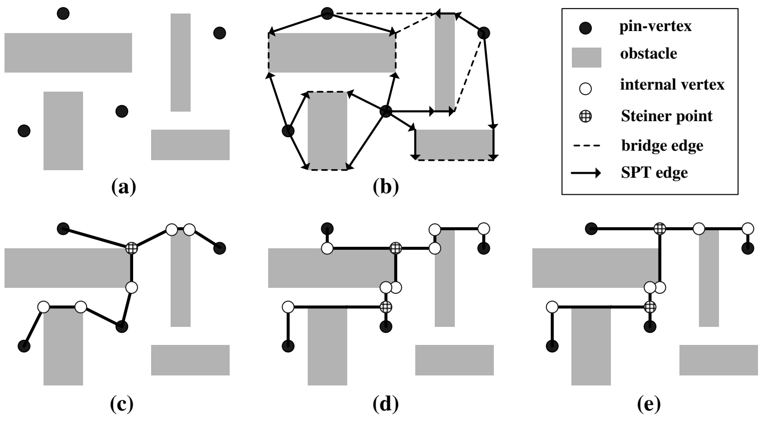

Figure 4.1: (b)–(e) four phases of our algorithm.

4.1 Motivation

As discussed in Section 1.3.3, Lin's OASG guarantees the existence of a shortest path for every two pin-vertices, and thus their algorithm provides an optimal solution for any two-pin net or every multi-pin net whose topology consists of simple paths among pin-vertices. Experimental results in [22] have shown that the the specific optimality guarantee significantly helps the solution quality. However, under the original MTST framework, an $O(n \log n)$-time algorithm with the specific optimality guarantee will require a routing graph which guarantees the existence of a shortest path for any two pin-vertices and has only $O(n)$ space. It is unknown if there exists such a graph and probably not. Therefore, a new framework should be employed to resolve the bottleneck.

4.2 Algorithm

Our path-based algorithm consists of the following four phases as shown in Figure 4.1. The first two phases generate a solution using critical paths without constructing a routing graph or generating an invalid solution.

1. Critical Path Generation: In this phase, critical paths are generated as solution components [See Figure 4.1(b)]. Those critical paths guarantee the existence of desirable solutions.

2. *Obstacle-Avoiding Steiner Tree* (OAST) Construction: An OAST connecting all pin-vertices is constructed by selecting those critical paths in Phase 1 [See Figure 4.1(c)]. A greedy path-based method is employed to reduce the wirelength.

3. OARST Construction: An OARST is constructed from the OAST in Phase 2 by transforming slant edges into rectilinear ones [See Figure 4.1(d)].

4. Local Refinement: The wirelength of the OARST in Phase 3 is be reduce by an $O(n \log n)$-time dynamic refinement scheme [See Figure 4.1(e)].

4.2.1 Critical Path Generation

We generate $O(n)$ critical paths in $O(n \log n)$ time, and claim that those critical paths guarantee the existence of optimal solutions on a number of specific cases. Section 4.2.1.1 and Section 4.2.1.2 analyze Lin's OASG construction [22] and Long's MTST algorithm [28] from our viewpoints. Section 4.2.1.3 simulates Long's MTST algorithm on Lin's OASG to analyze the corresponding geometry mapping and conclude the bottleneck of the time complexity. Section 4.2.1.4 applies shortest path map to generate critical paths, and justifies our claim using geometry information and graph theory.

4.2.1.1 Lin's OASG Construction

Definition 4.1 *For two vertices u and v, u is* ***immediately visible*** *from v, if (1) there is a path between u and v in their bounding box and (2) no other vertex in $P \cup C$ locates*

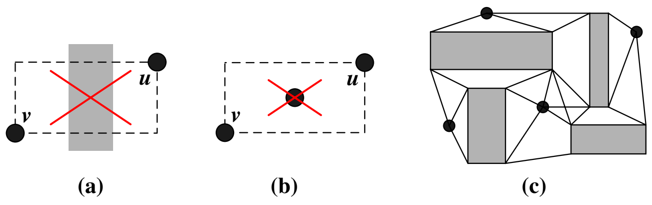

Figure 4.2: Example Definition 4.1 and Lin's OASG. (a) no path between u and v exists in the bounding box. (b) one vertex locates inside the bounding box. (c) Lin's OASG for Figure 4.1(a).

*inside or on the boundary of the bounding box [33]. If u is immediately visible from v, u is a **neighbor** of v [22]. For example, u is not a neighbor of v in Figure 4.2 (a)–(b). An edge connecting two neighbors is a **visible** edge.*

Lin et al. [22] constructed an OASG by connecting each vertex in $P \cup C$ to all its neighbors in $P \cup C$. Figure 4.2(c) shows Lin's OASG for Figure 4.1(a). They proved that their OASG guarantees the existence of a shortest path for any two vertices in $P \cup C$, implying that an MTST (Definition 2.9) of Lin's OASG is an OARSMT for any two-pin net or multiple-pin nets where an OARSMT contains only simple paths between pin-vertices. However, since a vertex has $O(n)$ neighbors, Lin's OASG has $O(n^2)$ edges.

4.2.1.2 Long's MTST algorithm

According to Definition 2.9, Long et al [28] proposed a two-step MTST construction with time complexity of $O(|E| \log |V|)$. Figure 4.3(c) shows an MTST of Figure 4.3(a) which consists of two terminal paths, $(p_1 \leftrightarrow v_1 \leftrightarrow v_2 \leftrightarrow v_4 \leftrightarrow p_3)$ and $(p_2 \leftrightarrow v_6 \leftrightarrow p_3)$.

Definition 4.2 *For a graph $G(V, E)$ and a terminal set $S \subseteq V$, a **terminal forest** F consists of $|S|$ disjoint shortest path trees where each tree T of F is rooted at a terminal $s \in S$ and for each vertex v of T, s is the nearest terminal of v in G (s is called the **root***

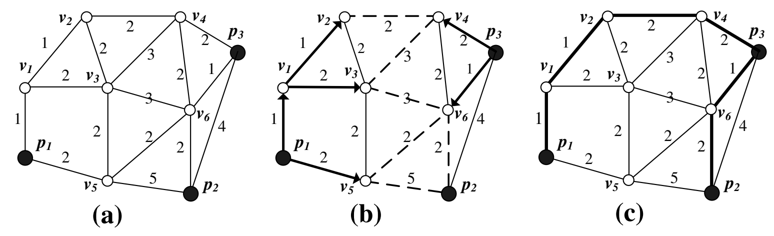

Figure 4.3: Example Definition 4.2. (a) an instance, where $S = \{p_1, p_2, p_3\}$. (b) a terminal forest and bridge edges of (a), where arrow segments are forest edges and dash segments are bridge edges. (c) an MTST of (a).

***terminal** of v) [28]. An edge $(v, u) \in E$ is a **bridge edge** if v and u belong to different trees in F.*

Figure 4.3(b) shows a terminal forest and bridge edges of Figure 4.3(a). v_1 belongs to the shortest path tree rooted at p_1 since p_1 is the nearest terminal of v_1; the dash segment connecting v_2 and v_4 is a bridge edge since v_2 and v_4 belong to different trees in the forest.

At the first step, they extended Dijkstra algorithm [6] to construct a terminal forest. At the second step, they first generated terminal paths (Definition 2.8 via all the bridge edges, e.g., a terminal path via a bridge edge (v_2, v_4) consists of

$SP(p_1, v_2)$, (v_2, v_4), and $SP(v_4, p_3)$, i.e., $(p_1 \leftrightarrow v_1 \leftrightarrow v_2 \leftrightarrow v_4 \leftrightarrow p_3)$. In other words, a bridge edge represents a terminal path. Then, they constructed an MTST by applying Kruskal algorithm [6] on a graph whose vertices are terminals and whose edges are those terminal paths generated via bridge edges.

4.2.1.3 Shortest Path Trees

Since an MTST of Lin's OASG is an OARSMT in many cases [22], we attempt to construct an equivalent solution. However, since Lin's OASG has $O(n^2)$ edges, for $O(n \log n)$ time complexity, we should avoid constructing Lin's OASG. Toward this end,

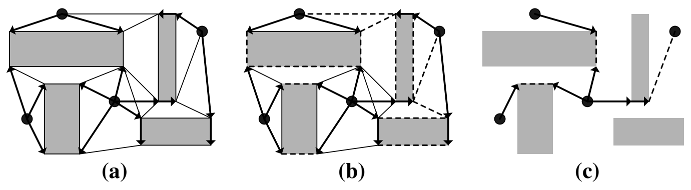

Figure 4.4: Long's MTST algorithm on Lin's OASG. (a) a terminal forest of Figure 4.2(c). (b) terminal paths via bridge edges of (a). (c) an MTST generated by those terminal paths in (b).

we simulate Long's MTST algorithm on Lin's OASG as shown in Figure 4.4 to analyze the corresponding geometry mapping. Since Lin's OASG [22] guarantees a shortest path for any two vertices in $P \cup C$, we study the literature about L_1 shortest paths, and conclude that it is feasible to map a terminal forest of Lin's OASG to Definition 4.3.

Definition 4.3 *Given a set P of pin-vertices and a set O of obstacles with a set C of corners,* ***multi-source shortest path trees*** *(multi-source SPTs) connect all the vertices in $P \cup C$ such that (1) for each vertex v in $P \cup C$, v belongs to a tree rooted a pin-vertex p in P, (2) p is the nearest pin-vertex of v, and (3) $SP(v, p)$ of the tree is also $SP(v, p)$ in the plane.*

Mitchell [33] proposed a wavefront-based method to construct multi-source SPTs in $O(n \log^2 n)$ time, and later improved the time complexity to $O(n \log n)$ in [32]. For each edge (v,u) of multi-source SPTs in [33], v and u are immediately visible to each other as the same as that of Lin's OASG, implying that the multi-source SPTs are equivalent to a terminal forest of Lin's OASG. Therefore, a terminal forest of Lin's OASG can be constructed in $O(n \log n)$ time without constructing Lin's OASG, and the bottleneck of the time complexity has become the handling of those bridge edges in Figure 4.4(b), each of which represents a terminal path.

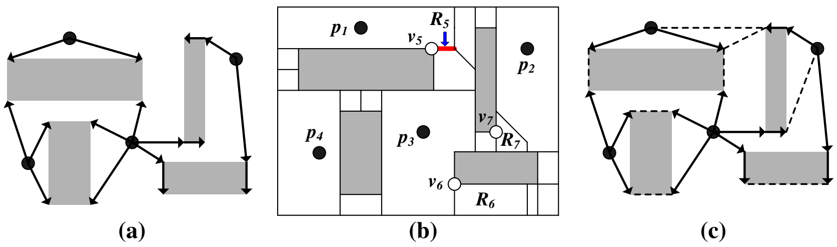

Figure 4.5: The critical path generation for the instance in Figure 4.1(a). (a) multi-source shortest path trees. (b) a shortest path map. (c) critical paths represented by bridge edges (dash segments) and multi-source SPTs.

Although the number of those bridge edges could be $\Omega(n^2)$, in Figure 4.4, we observe many redundant bridge edges which must not be in an MTST of Lin's OASG. Therefore, we conjecture that to obtain the equivalent solution, only $O(n)$ bridge edges need to be considered and can be constructed in $O(n \log n)$ time.

4.2.1.4 Shortest Path Map

To resolve the bottleneck of the time complexity and prove our conjecture, we should reduce redundant bridge edges. In [1], for constructing a *minimum spanning tree* (MST) in a plane, a Voronoi diagram can be applied to divide a plane and thus reduce the redundant edges. Similarly, we use a shortest path map to divide a plane, consider the obstacles, and reduce the redundant bridge edges.

Definition 4.4 *Given a set P of pin-vertices and a set O of obstacles with a set C of corners, a **shortest path map** (SPM) is a subdivision of plane where (1) each region belongs to a vertex $v \in P \cup C$ (called the site of region), (2) all points in the region of v share the same nearest pin-vertex $p \in P$, and (3) those points have the same predecessor v (the site of the region) along their shortest path to p. Each vertex in $P \cup C$ has only one* *region.*

Figure 4.5(b) shows an SPM for Figure 4.1(a). For example, all points in the region R_6, whose site is v_6, share the same nearest pin-vertex p_3, and all those points have at least one shortest path to p_3 passing through v_6. Noticeably, a region may have no area, e.g., the region R_5, which is represented by a red bold line segment and referred by a blue arrow, has no area. The slopes of those slant boundaries are 1 or -1 since in L_1 metric, the slope of a bisector must be one of $\{-1,\ 1,\ 0\ (horizontal),\ \infty\ (vertical)\}$

We propose the critical path generation as follows:

1. Multi-source SPTs are constructed using the wavefront-based method in [32, 33] as shown in Figure 4.5(a).

2. An SPM is constructed using the information of the multi-source SPTs as shown in Figure 4.5(b).

3. For each two adjacent regions, if their sites have different nearest pin-vertices and are immediately visible to each other, a bridge edge between the two sites is constructed to generate a critical path. Figure 4.5(c) shows those generated critical paths and represents them using bridges edges and multi-source SPTs. For a bridge edge (v_1, v_2), assuming the nearest pin-vertices of v_1 and v_2 to be p_1 and p_2 respectively, the critical path is $SP(p_1, v_1) \leftrightarrow (v_1, v_2) \leftrightarrow SP(v_2, p_2)$.

We will prove that those critical paths guarantee the existence of an equivalent solution to an MTST of Lin's OASG. Since a terminal forest of Lin's OASG is multi-source SPTs (discussed in Section 4.2.1.3), according to Long's MTST algorithm, we only need to discuss those bridge edges. If the critical path generation fails the guarantee, at least one essential bridge edge is not constructed. Without loss of generality, we assume the essential bridge edge to be (v_1, v_2). Since (v_1, v_2) is not constructed by the critical path generation, (v_1, v_2) should be broken by a region belonging to other vertex v_3. Assume the nearest pin-vertices of v_1, v_2, and v_3 to be p_1, p_2, and p_3 respectively, and v_4 to be a point on (v_1, v_2) in the region of v_3. According to Definition 4.4, $|SP(p_3, v_3)|+|(v_3, v_4)|$ is smaller than $|SP(p_1, v_1)|+|(v_1, v_4)|$ and $|SP(p_2, v_2)|+|(v_2, v_4)|$, implying that $|SP(p_3, v_3)|$+$|(v_3, v_4)|$+$|SP(p_1, v_1)|$+$|(v_1, v_4)|$ and $|SP(p_3, v_3)|$+$|(v_3, v_4)|$ + $|SP(p_2, v_2)|$ + $|(v_2, v_4)|$ is smaller than $|SP(p_1, v_1)|$+$|(v_1, v_4)|$+$|SP(p_2, v_2)|$+$|(v_2, v_4)|$, which is the length of the terminal path via (v_1, v_2). That is, at least two terminal paths for (p_1, p_3) and (p_2, p_3) respectively are shorter than the terminal path for (p_1, p_2) via (v_1, v_2). According to the cycle property of an MST [6], the terminal path via (v_1, v_2) can be safely deleted, i.e., (v_1, v_2) is not essential. There exists a contradiction.

Since an MTST of Lin's OASG is an OARSMT for any two-pin nets and multiple-pin nets where an OARSMT consists of only simple paths between pin-vertices, we conclude Theorem 4.1.

Theorem 4.1 *Our critical path generation guarantees the existence of an OARSMT for any two-pin net or multiple-pin nets where an OARSMT consists of only simple paths between pin-vertices.*

We analyze the time complexity of the critical path generation and the number of those generated critical paths below. Since the multi-source SPTs construction takes $O(n \log n)$ time in [32, 33], we only analyze step 2–3. In [33], Mitchell also claimed that an SPM, i.e. our step 2, can be extended from multi-source SPTs in $O(n \log n)$ time. By Euler's formula, the number of boundaries must be linear to the number of faces for a planar

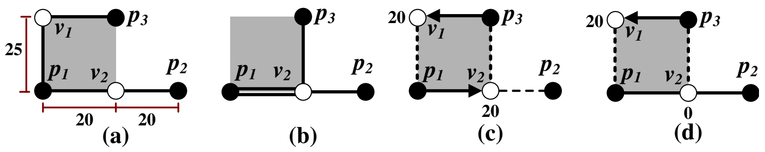

Figure 4.6: The overlapping of different paths. (a) no overlapping between the two paths. (b) overlapping at (v_2, p_3). (c) the critical paths generated by Section 4.2.1. (d) $|SP(p_1, p_3)|$ via (v_2, p_3) has become 20 after update operation.

subdivision. Since there are $O(n)$ regions in an SPM (by Definition 4.4), there are $O(n)$ pairs of adjacent regions, implying that our critical path generation constructs $O(n)$ bridge edges (critical paths). To check if a bridge edge is visible can be done by searching other vertices locating inside the bounding box of the two endpoints. The search operation takes $O(\log n)$ time after constructing a range search tree with fractional cascading [1] among $P \cup C$, which takes $O(n \log n)$ time. Therefore, we can conclude Theorem 4.2. Theorem 4.1 and Theorem 4.2 strongly support our conjecture in Section 4.2.1.3.

Theorem 4.2 *Our critical path generation provides $O(n)$ critical paths in $O(n \log n)$ time.*

4.2.2 OAST Construction

In this Section, we propose a greedy method to construct an *obstacle-avoiding Steiner tree* (OAST) based on those critical paths in Section 4.2.1. Unlike most previous works, for better solution quality, our greedy method attempts to increase the overlapping between different paths, which is critical but often neglected.

Most MTST-based algorithms [22,28,36] select paths (edges) according to their lengths without considering the overlapping between different paths, and thus may lose much potential improvement of wirelength. For example, as shown in Figure 4.6, $|SP(p_1, p_2)|$, $|SP(p_2, p_3)|$, and $|SP(p_1, p_3)|$ are 40, 45, and 45 respectively, implying that an MTST can consist of $SP(p_1, p_2)$ and $SP(p_1, p_3)$. However, there are two possibilities to select $SP(p_1, p_3)$, $p_1 \leftrightarrow v_1 \leftrightarrow p_3$ and $p_1 \leftrightarrow v_2 \leftrightarrow p_3$, as shown in Figure 4.6(a) and Figure 4.6(b) respectively. In the former case, $SP(p_1, p_2)$ and $SP(p_2, p_3)$ do not overlap, while in the latter case, $SP(p_1, p_2)$ and $SP(p_2, p_3)$ overlap at (p_1, v_2) and result in an obviously redundant edge, which will be removed easily. After removing the redundant edge (p_1, v_2), the wirelength of Figure 4.6(b) is 65, while that of Figure 4.6(a) is 85. Thus, from our viewpoint, Figure 4.6(b) contains more potential improvement of wirelength than Figure 4.6(a). In short, increasing the overlapping between different paths will lower the wirelength.

Our path-based framework provides a potential way to increase the overlapping of different paths. For example, Figure 4.6(c) shows those critical paths generated by Section 4.2.1 and represents them using three shortest path trees and three bridge edges (dash segments). In detail, bridge edges, (p_1, v_1), (v_2, p_2), and (v_2, p_3), represent $p_1 \leftrightarrow v_1 \leftrightarrow p_3$, $p_1 \leftrightarrow v_2 \leftrightarrow p_2$, and $p_1 \leftrightarrow v_2 \leftrightarrow p_3$, whose lengths are 45, 40, and 45, respectively. The values adjacent to v_1 and v_2 are the distances from them to their nearest pin-vertices, e.g., $|SP(p_1, v_2)|$ is 20. Since $p_1 \leftrightarrow v_2 \leftrightarrow p_2$ has the lightest length, we will first select this critical path. After the selection, as shown in Figure 4.6(d), the distance between v_2 and p_1 can be viewed as 0 since any path to p_1 through v_2 can share the path between p_1 and v_2. Then, we update bridge edges connected to v_2, i.e. (v_2, p_3), to reduce $|p_1 \leftrightarrow v_2 \leftrightarrow p_3|$ to be 25. Therefore, we will connect $p_1 \leftrightarrow v_2 \leftrightarrow p_3$ instead of $p_1 \leftrightarrow v_1 \leftrightarrow p_3$ to obtain the same result with Figure 4.6(b). To conclude, by updating bridge edges connected to

```
**Algorithm:** $Greedy\_OAST\_Construction(P, C, BE, TE, E)$
**Input:** $P$ /* the set of pin-vertices */
       $C$ /* the set of obstacle corners */
       $BE$ /* the set of bridge edges */
       $TE$ /* the set of directed tree edges
           of multi-source SPTs*/
**Output:** $E$ /* the set of edges of the OAST */
1  **for** each directed tree edge $e = (v, u) \in TE$
2    $p(u) \leftarrow v$
3  **for** each vertex $v$ in $P \cup C$
4    $w(v) \leftarrow |SP(v, np(v))|$
   /* $np(v)$ is the nearest pin-vertex of $v$ */
5  Heap $H_{cp} \leftarrow \phi$
6  **for** each bridge edge $e = (v, u) \in BE$
7    $w(cp(e)) \leftarrow w(v) + |(v, u)| + w(u)$
   /* $cp(e)$ denotes the critical path via $e$ */
8    $H_{cp}.insert(e, w(cp(e))$
9  **while** $H_{cp}$ is nonempty
10   $e(v, u) \leftarrow H_{cp}.extractMin()$
11   **if** *Find-Set*$(np(v)) \neq$ *Find-Set*$(np(u))$
12     *Set-Union*( *Find-Set*$(np(v))$ , *Find-Set*$(np(u))$ )
13     $E \leftarrow E \cup \{(v, u)\}$
14     **for** each vertex $v' \in \{v, u\}$
15       **while** $v'.makred$ = false and $p(v') \neq null$
16         $w(v') \leftarrow 0$
17         **for** each bridge edge $e' = (v', u') \in BE$
18           $w(cp(e')) \leftarrow w(v') + |(v', u')| + w(u')$
19           $H_{cp}.decreaseKey(e', w(cp(e')))$
20         $v'.marked \leftarrow$ true
21         $E \leftarrow E \cup \{(v', p(v'))\}$
22         $v' \leftarrow p(v')$
```

Figure 4.7: The greedy OAST construction algorithm.

selected critical paths, our path-based framework shares the edges of selected paths with unselected ones, and thus increases the overlapping between different paths.

The greedy OAST construction algorithm is summarized in Figure 4.7. Lines 1–8 initialize the information representing those critical paths. Lines 1–2 assign the predecessor to each vertex, Lines 3–4 assign weight to each vertex, and Lines 5–8 initial H_{cp} by assigning weight to each critical path and inserting the corresponding bridge edge to H_{cp}. Lines 9–22 construct an OAST, and increase the overlapping of different paths by updating bridge edges. Line 11 determines if a critical path results in a cycle in P; Lines 13–22 recursively connect edges in the critical path and update the corresponding bridge edges.

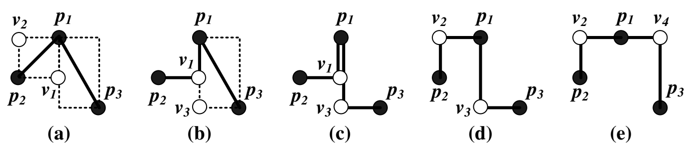

Figure 4.8: The rectilinear transformation for slant edges. (a) an instance. (b)–(c) transformation with edge overlapping. (d)–(e) other cases without edge overlapping.

In details, Lines 16–19 update the corresponding bridge edges; Lines 20–22 connect the edges leading to the nearest pin-vertex. It is clear that the recursive updates of bridge edges will further take benefit of the path overlapping. In fact, the OAST construction does not connect those obviously redundant edges.

We prove the time complexity to be $O(n \log n)$. Since multi-source SPTs are a forest connecting all vertices in $P \cup C$, $|TE|$ is $O(n)$, and thus Lines 1–2 take $O(n)$ time. Lines 3–4 can be done by traversing tree edges in TE from each root, which takes $O(n)$ time. Lines 5–8 take time loglinear to the size of BE, which is $O(n)$ by Theorem 4.2. Since Lines 14–22 trace a vertex at most once and update a bridge edge at most twice, Lines 9–22 perform $O(n)$ times find-set, trace-vertex, decrease-key operations. Since each operation takes $O(\log n)$ time, Lines 9–22 take $O(n \log n)$ time.

In not considering the overlapping between different paths, we will obtain an OAST as shown in Figure 4.4(c), while our greedy OAST construction obtains a better OAST as shown in Figure 4.1(c).

4.2.3 OARST Construction

We construct an *obstacle-avoiding rectilinear Steiner tree* (OARST) from the OAST in Section 4.2.2 by transforming all slant edges into rectilinear ones. Since all edges of the OAST are visible we can directly transform an edge of the OAST into L-shaped rectilinear edges. For example, as shown in Figure 4.8(a), since no obstacle intersects the bounding box of p_1 and p_2 (otherwise, (p_1,p_2) must not be visible), (p_1, p_2) can be transformed to L-shaped rectilinear edges, (p_1, v_1) and (v_1, p_2) (or (p_1, v_2) and (v_2, p_2)).

The OARST construction also considers the overlapping between different edges. For example, if (p_1, p_2) has been transformed to (p_1, v_1) and (v_1, p_2) as shown in Figure 4.8(b), (p_1, p_3) will be transformed to (p_1, v_3) and (v_3, p_3) to result in the edge overlapping $(p1, v1)$ as shown in Figure 4.8(c), which can be directly removed to improve the wirelength. On the other hand, if (p_1, p_2) has been transformed to (p_1, v_2) and (v_2, p_2), Figure 4.8(d) or Figure 4.8(e) will occur. Nevertheless, since no obstacle intersects the bounding boxes of (p_1, p_2) and (p_1, p_3), Figure 4.8(d) and Figure 4.8(e) can be refined to Figure 4.8(c) by the local refinement in Section 4.2.4. It is clear that the whole OARST construction takes $O(n \log n)$ time.

4.2.4 Efficient Local Refinement

Definition 4.5 *A* ***segment*** *is a connected graph whose edges are arranged in the same line, e.g.,* $\{e_1, e_2\}$ *in Figure 4.9(a) is a segment. For a segment* s*, an* ***adjacent*** *segment is a segment connected to and perpendicular to* s*, e.g.,* $\{e_1, e_2\}$ *in the left part of Figure 4.9(a) has three adjacent segments.*

We attempt to perform dynamic local refinements to reduce the wirelength in $O(n \log n)$ time. Lin et al. [22] introduced a U-shaped refinement handling the two cases in Figure 4.9(a)–(b). Based on the U-shaped refinement, we conclude that if a segment has more adjacent segments on one side than the other side, moving this segment to the former side may reduce the wirelength. We called a segment as *movable* if it can be moved

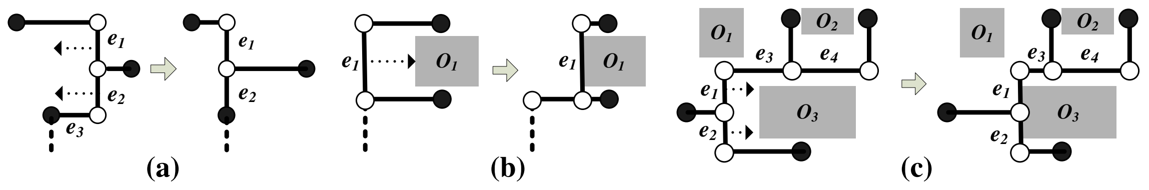

Figure 4.9: Refinement Patterns. (a)-(b) two cases of U-shaped patterns. (c) the up nearest obstacle of a segment, $\{e_3, e_4\}$, has been changed to O_2 from O_1 after moving $\{e_1, e_2\}$ right.

to reduce the wirelength.

As shown in Figure 4.9(b), the moving offset of a segment may depend on the nearest obstacle. However, the nearest obstacle of a segment cannot be pre-computed before dynamic local refinements since the nearest obstacle may be changed. For example, in Figure 4.9(c), the up nearest obstacle of $\{e_3, e_4\}$ originally is O_1, while the up nearest obstacle has become O_2 after moving $\{e_1, e_2\}$ right. For $O(n \log n)$ time complexity, we should find a way to compute the nearest obstacle for a movable segment in $O(\log n)$ time.

The right nearest obstacle of a segment is also the first touched obstacle when moving the segment right. As shown in Figure 4.10(a)–(b), moving a segment to touch an obstacle can be divided into two cases, (1) touch the obstacle corners and (2) just touch the obstacle boundary. In the latter case, if the nearest obstacle influences the moving offset, the movable segment should have an adjacent segment with one endpoint locating at the boundary of the obstacle. Therefore, the moving offset in the latter case can be computed using those adjacent segments, and we only discuss the first case.

Chazelle [3] studied *the segment dragging query problem*: Given a set of n points, pick a horizontal (vertical) segment and answer the first hit point when dragging the seg-

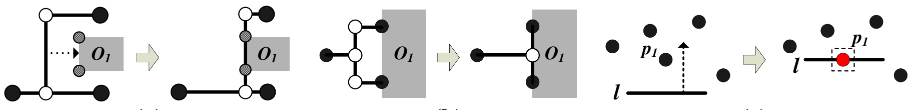

Figure 4.10: Computation of the nearest obstacles. (a) touching the corners of an obstacle. (b) just touching the boundary of an obstacle. (c) the segment dragging problem.

ment vertically (horizontally). As shown in Figure 4.10(c), dragging l up will first hit p_1, and p_1 is the answer of this segment dragging query. Chazelle proposed a method to answer the segment dragging query in $O(\log n)$ time after an $O(n \log n)$-time preprocessing. Therefore, by applying Chazelle's method on the corner set $\boldsymbol{C}$, we can compute the nearest obstacle for the first case in $O(\log n)$ time.

Considering the tradeoff between efficiency and solution quality, our refinement scheme operates each movable segment with at most 8 adjacent segments until no such segment exists. Moving a movable segment may generate a new movable segment (e.g. merging two segments), but in the case, the movement will remove at least one edge such as e_3 in Figure 4.9(a). Since the OARST has $O(n)$ edges, the refinement scheme will operate $O(n)$ movable segments. For each segment s, since s has at most 8 adjacent segments and the nearest obstacle is computed in $O(\log n)$ time, the moving offset can be computed in $O(\log n)$ time. Therefore, the refinement scheme takes $O(n \log n)$ time.

4.2.5 Discussion

Since each phase takes $O(n \log n)$ time, the time complexity of the whole construction is still $O(n \log n)$. By Theorem 4.1 and the details in Section 4.2.2 to 4.2.4, our algorithm guarantees all the optimality in Section 4.2.5 of [22]. We can conclude the following.

Theorem 4.3 *Our algorithm constructs an OARST in $O(n \log n)$ time, and guarantees to* *provide an OARSMT for any two-pin nets and multiple-pin nets where the topology of an OARSMT contains only simple paths between pin-vertices.*

The output of Phase 1–2 is similar to [22] and [28], while [22] takes $O(n^3)$ time and [28] cannot guarantee the same optimality. Long et al. [28] reduced the redundant terminal paths in a graph, but for $O(n \log n)$ time, they omitted some essential edges to construct an $O(n)$-space graph. Our framework directly reduces the redundant terminal paths in geometry domain without constructing a graph. This is why our algorithm can satisfy Theorem 4.1.

To conclude, compared with recent works [36]– [28], our path-based framework has a global view of obstacles, guarantees the existence of desirable solutions, and keeps only $O(n)$-space solution components. It really gives a new efficient and effective way to deal with the OARSMT problem.

4.3 Experimental Results

We have implemented our algorithm in C. There are 22 benchmark circuits, five industrial test cases (ind01–ind05) from Synopsys, 12 test cases used in [8] (rc01–rc12), and five random test cases used in [22] (rt01–rt05). Our experiments were conducted on a Linux server with two 2.4-GHz Intel processors and 8-GB memory.

We compare our algorithm with those presented in [22], [28] and [21]. To the best of our knowledge, the approach in [28] is the most effective $O(n \log n)$-time method, and the method in [21] achieves the best solution quality. The results of [28] are quoted from the paper, where the algorithm was conducted on a Redhat Linux server with two 2.1-GHz AMD processors and 2-GB memory. The results of [22] are quoted from the paper, where the algorithm was conducted on a Ubuntu 6.06 server with one 2-GHz AMD-64 CPU and 8-GB memory. We have obtained the program in [21], and execute their program on our platform.

Table 4.1 lists the total wirelengths and the CPU times of these algorithms. “speedup” compares the run time of [21] and ours.

Compared with [28], the most effective $O(n \log n)$-time method, our algorithm achieves the same speed performance and further improves by 2.7% in wirelength on average, which is significant considering the routability for millions of signal nets. The wirelength of our algorithm is at most 7.4 % shorter than that of [28] (rc12), but at most only 0.7% longer (rc03). Besides, our algorithm guarantees the same theoretical optimality as that in [22], but [28] cannot. Therefore, the solution quality of our algorithm could be more stable.

Compared with [21], which achieves the best solution quality, our algorithm runs much faster, while the wirelengths of the resulting solutions are only 1.1% longer on average. As shown in Table 4.1, across all the 22 benchmarks, our algorithm achieves 50.1 times speedup on average compared with [21]. For rc12, which contains 10 000 obstacles, our algorithm terminates within only one second, while that in [21] takes 192 seconds. Considering the large and increasing number of obstacles and signal nets in a modern IC design, the advantage in the run time is very significant.

To conclude, the above analysis shows that our algorithm achieves the best speed performance, and the solution quality is still comparable to the best previous work [21]. Our algorithm meets the requirements of the OARST construction in a modern IC design.

Table 4.1: Comparison on the Total Wirelength and the CPU Time.

Test		Total Wirelength				Imp.(%) ($\frac{X-D}{X}$)			Time (sec.)				speedup ($\frac{E}{F}$)
Cases	m / k	[28] (A)	[22] (B)	[21](C)	ours (D)	(X=A)	(X=B)	(X=C)	[28]	[22]	[21] (E)	ours (F)	
ind01	10 / 32	639	632	619	626	2.0	0.9	-1.1	0.01	$<$ 0.01	0.007	0.001	7.00 x
ind02	10 / 43	10,000	9,600	9,500	9,700	3.0	-1.0	-2.1	0.01	$<$ 0.01	0.042	0.001	42.00 x
ind03	10 / 50	623	613	600	600	3.7	2.1	0.0	0.01	$<$ 0.01	0.009	0.001	9.00 x
ind04	25 / 79	1,126	1,121	1,096	1,095	2.8	2.3	0.1	0.02	$<$ 0.01	0.033	0.002	16.50 x
ind05	33 / 71	1,379	1,364	1,360	1,364	1.1	0.0	-0.3	0.02	0.01	0.008	0.002	4.00 x
rc01	10 / 10	27,540	26,900	25,980	26,740	2.9	0.6	-2.9	0.01	$<$ 0.01	0.044	$<$ 0.001	-
rc02	30 / 10	41,930	42,210	42,010	42,070	-0.3	0.3	-0.1	0.01	$<$ 0.01	0.103	0.001	103.00 x
rc03	50 / 10	54,180	55,750	54,390	54,550	-0.7	2.2	-0.3	0.01	$<$ 0.01	0.097	0.001	97.00 x
rc04	70 / 10	59,050	60,350	59,740	59,390	-0.6	1.6	0.6	0.02	$<$ 0.01	0.121	0.001	121.00 x
rc05	100 / 10	75,630	76,330	74,650	75,430	0.3	1.2	-1.0	0.02	0.01	0.092	0.001	92.00 x
rc06	100 / 500	86,381	83,365	81,607	81,903	5.2	1.8	-0.4	0.13	0.24	0.577	0.017	33.94 x
rc07	200 / 500	117,093	113,260	111,542	111,752	4.6	1.3	-0.2	0.15	0.43	0.714	0.026	27.46 x
rc08	200 / 800	122,306	118,747	115,931	118,349	3.2	0.3	-2.1	0.27	0.83	1.194	0.043	27.77 x
rc09	200 / 1,000	119,308	116,168	113,460	114,928	3.7	1.1	-1.3	0.36	0.91	1.502	0.049	30.65 x
rc10	500 / 100	167,978	170,690	167,620	167,540	0.3	1.8	0.0	0.08	0.62	0.291	0.016	18.19 x
rc11	1,000 / 100	232,381	236,615	235,283	234,097	-0.7	1.1	0.5	0.14	3.15	0.844	0.021	40.19 x
rc12	1,000 / 10,000	842,689	789,097	761,606	780,528	7.4	1.1	-2.5	5.88	118.52	192.920	0.681	283.29 x
rt01	10 / 500	2,362	2,267	2,231	2,259	4.4	0.4	-1.3	0.12	0.06	0.171	0.017	10.06 x
rt02	50 / 500	52,218	48,441	47,297	48,684	6.8	-0.5	-2.9	0.11	0.11	0.612	0.018	34.00 x
rt03	100 / 500	8,645	8,368	8,187	8,347	3.4	0.3	-2.0	0.13	0.47	0.211	0.020	10.55 x
rt04	100 / 1,000	10,580	10,306	9,914	10,221	3.4	0.8	-3.1	0.42	0.95	0.364	0.040	9.10 x
rt05	200 / 2,000	55,286	53,993	52,473	53,745	2.8	0.5	-2.4	1.43	2.06	2.759	0.078	35.37 x
Average						2.7	0.9	-1.1	Average				50.10 x

Chapter 5

Steiner Point Selection

For the OARSMT problem, this chapter presents a Steiner-point based algorithm in order to achieve the best practical performance in both wirelength and run time. Unlike most existing works, the Steiner-based framework is more focused on the usage of Steiner points instead of the handling of obstacles. We also proposes a new concept of Steiner point locations, which provides an effective as well as efficient way to generate desirable Steiner point candidates. Experimental results show that this algorithm achieves the best solution quality in $\Theta(n \log n)$ empirical time, which was originally generated by applying the maze routing on an $\Omega(n^2)$-space graph. In other words, this algorithm achieves the best practical performance in both wirelength and run time. More importantly, the Steiner-point based framework and the new concept of Steiner point locations give critical insights into the OARSMT problem, and can be naturally applied to the future researches on the OARSMT problem and its generations, such as the ML-OARSMT problem [24] and the OA-PDST problem [25].

5.1 Motivation

As discussed in Section 1.3, there exists a significant trade-off between rectilinear graphs and spanning graphs. In detail, a rectilinear graph usually has better solution quality, and a spanning graph usually has higher efficiency. Therefore, it is desirable to balance the trade-off to achieve the best practical performance in both the wirelength and the run time. Toward this end, it is interesting to find a way to integrate the advantages of spanning graphs and rectilinear graphs. Such a way very probably helps the development of more desirable algorithms. On the other hand, from the viewpoint of practice, the $O(n \log n)$ worst-case time complexity is not necessary, and the $\Theta(n \log n)$ empirical time complexity is sufficient. As a result, we attempt to comprehensively study the essence of the OARSMT problem, and make essential theoretical foundations.

5.2 Algorithm

We propose a Steiner-point based algorithm for the OARSMT problem. Section 5.2.1 claims the importance of Steiner points and gives a basic idea. Section 5.2.2 proposes a new concept of Steiner point locations to be the foundation of our algorithm. Section 5.2.3 shows the procedure of our algorithm in detail. Section 5.2.4 analyzes the time complexity of our algorithm. Section 5.2.5 gives a discussion and makes comparisons.

5.2.1 Basic Idea

The selection of Steiner points is the most important step to solve the OARSMT problem although many previous works mainly focus on the handling of obstacles. This is because the NP-completeness of the OARSMT problem probably comes from the selection of Steiner points instead of the handling of obstacles. For example, without selecting Steiner points, the OARSMT problem will become the *obstacle-avoiding rectilinear minimum spanning tree* (OARMST) problem, which is solvable in polynomial-time [45]. From the viewpoint of Steiner point selection, we make another definition for an OARSMT below, which is also used in [38].

Definition 5.1 *An* ***OARSMT*** *is an OARMST connecting all the pin-vertices and some selected Steiner points where the total wirelength of the OARMST is minimum.*

Based on Definition 5.1, the OARST construction naturally first selects desirable Steiner points, and then constructs an OARMST to connect all the pin-vertices and those selected Steiner points. In general, a routing graph is an efficient way to deal with obstacles, and the OARMST construction has become the MTST construction (Definition 2.5) on the routing graph. Besides, to reduce the redundant wirelength, refinement schemes are usually employed after the initial OARST construction. As a result, our Steiner-point based algorithm is divided into four steps: (1) graph construction, (2) Steiner point selection, (3) MTST construction, and (4) refinement. More details will be shown in Section 5.2.3.

As mention in Section 5.1, there is a trade-off between solution quality and speed performance in the selection of rectilinear graphs and spanning graphs. Rectilinear graphs [9, 21, 38] usually contain more desirable Steiner points and even include the optimal solution, while the number of vertices is usually $\Omega(n^2)$, indicating that the Steiner point selection in rectilinear graphs is very time-consuming. Spanning graphs [22, 28, 36] mainly use obstacle corners to contain valid solutions, while a good Steiner point is not necessarily restricted on obstacle corners, indicating that the solution quality in spanning graphs is limited.

Since the Steiner point selection is the most critical step, we attempt to analyze the Steiner point locations (Section 5.2.2) to develop (1) a new graph with both $O(n)$ space and desirable Steiner point candidates, and (2) a new Steiner point selection to find good Steiner points. Both the new graph and the new Steiner point selection lead our algorithm to achieve both of the best solution quality and the best speed performance.

5.2.2 Concept of Steiner Point Locations

For Steiner point locations, Hanan grid [12] (Figure 5.1(c)) and Escape graph [9] (Figure 5.2(c)) should be discussed since they contain at least one optimal solution for the RSMT problem and the OARSMT problem, respectively. Both Hanan grid and Escape graph generate vertices on the intersections of line segments extended from pin-vertices or obstacle corners. However, n segments will generate up to $\frac{n*(n-1)}{2}$ intersections, which are too many for the Steiner point selection. Although $O(n)$ intersections can be generated by $O(\sqrt{n})$ line segments, $O(\sqrt{n})$ line segments cannot reflect all the n vertices of $P \cup C$. In other words, the intersection concept of Hanan grid and Escape graph will generate too many Steiner point candidates and lower the efficiency of Steiner point selection. It is necessary to propose a new concept of Steiner point locations instead of line segment intersections, and this new concept potentially provides a more efficient way to generate desirable Steiner point candidates.

We first consider the RSMT problem for two pin-vertices. Since an RSMT for two pin-vertices is directly a shortest path between them, we define a *shortest path region*

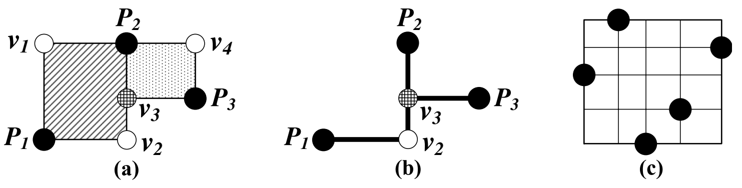

Figure 5.1: Steiner points for the RSMT problem. (a) a three-pin instance. (b) an RSMT of (a). (c) Hanan grid.

(SPR) below.

Definition 5.2 *For two vertices p and q on a plane, their **shortest path region** (SPR), denoted as SPR(p,q), is the union of all the shortest paths between p and q.* In Figure 5.1,(a) the bounding box of P_1 and P_2 is their shortest path region.

When a two-pin instance is extended to a three-pin instance, the location of the new pin-vertex should be first determined. If the new pin-vertex is located inside the SPR of the two original pin-vertices, the new pin-vertex is directly the best Steiner point of the three-pin instance, i.e., there is no Steiner point for this instance. Otherwise, the best Steiner point must be a boundary corner of one SPR among the three pin-vertices. For example, as shown in Figure 5.1(b), the Steiner point of the RSMT is v_3, which is a boundary corner of SPR(P_2,P_3) in Figure 5.1(a). With the similar reasoning, we conclude that a Steiner point of an RSMT in Hanan grid must be a boundary corner of the SPR of two pin-vertices. In the new concept, Hanan grid is the collection of boundary corners of SPRs among all the pin-vertices. We have the following theorem.

Theorem 5.1 *For the RSMT problem, there exists at least one RSMT such that each Steiner point of which is a boundary corner of the shortest path region of two pin-vertices.*

For the OARSMT problem, obstacles will affect Steiner point locations, and the shape of an SPR has become different. For example, in Figure 5.2(a), SPR(P_1,P_2) has become

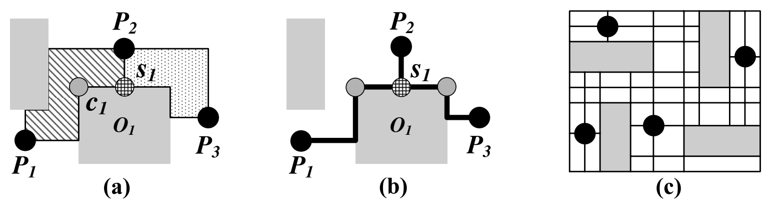

Figure 5.2: Steiner points for the OARSMT problem. (a) a three-pin instance. (b) an ORSMT of (a). (c) Escape graph.

a rectilinear polygon, and in Figure 5.2(b), the best Steiner point has become s_1. We return to the RSMT problem, and find that the boundary of an SPR consists of two L-shaped paths, and a Steiner point in Hanan grid is located at one corner of L-shaped paths between two pin-vertices. For example, in Figure 5.1(a), the boundary of SPR(P_2,P_3) consists of two L-shaped paths, $P_2 \leftrightarrow v_3 \leftrightarrow P_3$ and $P_2 \leftrightarrow v_4 \leftrightarrow P_3$, and in Figure 5.1(b), v_3 is the best Steiner point.

From the same viewpoint, in Figure 5.2(a)–(b), s_1, the Steiner point of the OARSMT, is the corner of L-shaped path $c_1 \leftrightarrow s_1 \leftrightarrow P_2$. Similarly, Escape graph can be viewed as the collection of corners of L-shape paths among all the pin-vertices and all the obstacle corners. We also have the following theorem.

Theorem 5.2 *For the OARSMT problem, there exists at least one OARSMT such that each Steiner point of which is a corner of the L-shaped path between one pair of vertices in* $P \cup C$.

Based on the new concept (Theorem 5.2), Escape graph construction is equivalent to considering at most C_2^n pairs of pin-vertices and obstacle corners, i.e., $O(n^2)$ L-shaped paths, and then generating $O(n^2)$ L-shaped path corner (vertices). If we only consider $O(n)$ pairs of vertices, we will have only $O(n)$ Steiner point candidates, which is more efficient for the Steiner point selection. Therefore, the bottleneck has become how to

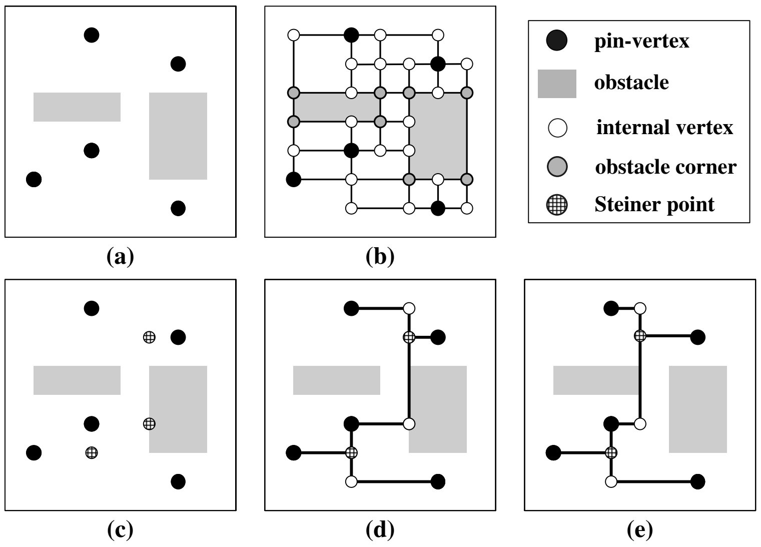

Figure 5.3: (b)–(e) four steps of our algorithms.

choose $O(n)$ pairs of vertices to generate desirable Steiner point candidates, which is our OAVG construction in Section 5.2.3.1.

In summary, the new concept of Steiner point locations will lead to our OAVG construction, which constructs a routing graph with $O(n)$ space and desirable Steiner point candidates, i.e., the OAVG integrates the effectiveness and the efficiency of routing graphs. Besides, the shortest path region also contributes to our Steiner point selection in Section 5.2.3.2. Note that it is impossible to select $O(n)$ Steiner point candidates in $\omega(n^2)$ time to guarantee the existence of an RSMT unless Hanan grid can be reduced to $O(n)$ space in $\omega(n^2)$ time. In fact, although Hanan grid has been proposed for about 40 years, there is still no such method to reduce it.

5.2.3 Procedure of Algorithm

Based on the basic idea in Section 5.2.1, our Steiner-point based algorithm consists of following four steps.

1. *Obstacle-avoiding Voronoi graph* (OAVG) construction

   (Figure 5.3(b)): An OAVG is constructed to connect pin-vertices, obstacle corners, and $O(n)$ desirable Steiner point candidates in $O(n \log n)$ time. The effectiveness and efficiency is supported by the new concept of Steiner point locations and novel computational geometry techniques.

2. Steiner point selection (Figure 5.3(c)): Good Steiner points are efficiently selected from the OAVG in step 1 based on the concept of Prim's algorithm and our defined SPRs.

3. Initial *Obstacle-avoiding rectilinear Steiner tree* (OARST) construction (Figure 5.3(d)): An OARST is constructed in

   $O(n \log n)$ time by applying an integration of Long's MTST algorithm [28] and Liu's path-overlapping scheme [26] on the OAVG where all the pin-vertices and those selected Steiner points in step 2 are specified as terminals.

4. Refinement (Figure 5.3(e)): A refinement scheme is applied on the OARST construced in step 3 to reduce the redundant segments in $O(n \log n)$ time.

5.2.3.1 OAVG Construction

We attempt to construct a routing graph called *obstacle-avoiding Voronoi graph* (OAVG), which contains desirable Steiner point candidates and uses only $O(n)$ space. According to Section 5.2.2 and Theorem 5.2, we need to choose $O(n)$ desirable pairs of pin-vertices and obstacle corners, and then collect L-shaped path corners of those pairs as Steiner point candidates.

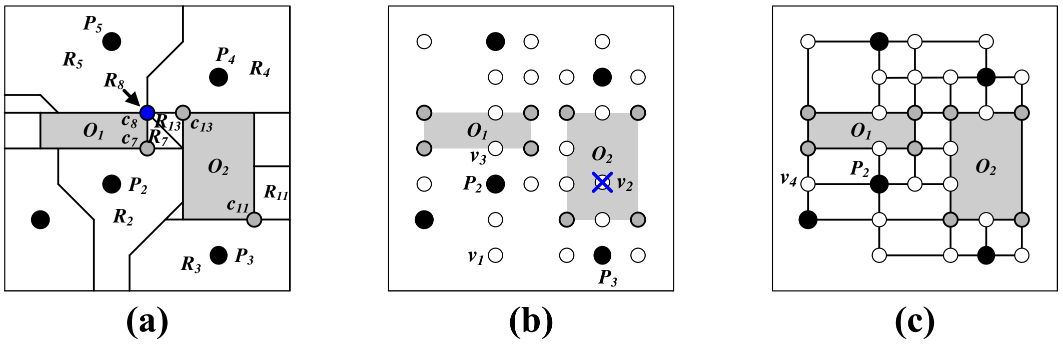

Figure 5.4: OAVG construction. (a) SPM construction. (b) vertex generation. (c) edge connection.

Since a *rectilinear minimum spanning tree* (RMST) is usually used to approximate an RSMT, we study the RMST problem. Hwang [17] use an L_1 Voronoi diagram to construct an RMST in $O(n \log n)$ time since the dual graph of the L_1 Voronoi diagram contains at least one RMST and has only $O(n)$ edges. Therefore, it seems a good idea to choose all pairs of adjacent vertices in the dual graph to generate Steiner point candidates.

In the presence of obstacles, Mitchell [33] proposed a wavefront based method to construct an L_1 Voronoi diagram in $O(n \log^2 n)$ time, and later improved it to $O(n \log n)$-time in [32]. The L_1 Voronoi diagram can be partitioned into finer regions by a shortest path map, which is defined below.

Definition 5.3 *For a set P of pin-vertices and a set O of obstacles with a set C of corners, a **shortest path map** (SPM) is a subdivision of plane where (1) each region belongs to a vertex $v \in P \cup C$ (called the site of the region), (2) all points in the region of v share the same nearest pin-vertex $p \in P$, and (3) those points have the same predecessor v (the site of the region) along their shortest path to p [26]. Noticeably, each vertex in $P \cup C$ has only one region.*

Figure 5.4(a) shows an SPM for Figure 5.3(a). For example, all points in the region

R_{11}, whose site is c_{11}, share the same nearest pin-vertex P_3, and all those points have at least one shortest path to P_3 passing through c_{11}. Noticeably, a region may have no area, i.e., is a line segment or a point. For example, R_8, which is the region of c_8 and represented by a blue solid circle, has no area but is adjacent to R_4, R_5, R_7, and R_{13}. The slopes of those slant boundaries are 1 or -1 since in L_1 metric, the slope of a bisector must be one of $\{-1, 1, 0(horizontal), \infty(vertical)\}$.

Our OAVG construction is summarized in Figure 5.4:

1. SPM construction (Figure 5.4(a)).
2. Vertex generation (Figure 5.4(b)): For each two adjacent regions R_i and R_j in the SPM, generate Steiner point candidates for the two sites according to Theorem 5.2. For example, v_1 will be generated in Figure 5.4(b) since R_2 is adjacent to R_3 in Figure 5.4(a) and v_1 is the corner of the L-shaped path, $P_2 \leftrightarrow v_1 \leftrightarrow P_3$. Besides, for the connectivity of the OAVG, we also project all the vertices to their closest obstacles in the four directions. For example, in Figure 5.4(b), v_3 is a projection from P_2 to its up closest obstacle O_1.
3. Edge connection (Figure 5.4(c)): Connect all the vertices in Figure 5.4(b) using rectilinear edges.

By Euler's formula, for a planar subdivision, the number of boundary edges must be linear to the number of faces. As a result, since an SPM has n regions (by Definition 5.3), there are $O(n)$ pairs of adjacent regions, implying the number of generated Steiner point candidates is $O(n)$. Since a vertex has at most 4 closest obstacles, there are $O(n)$ projection vertices. As a result, an OAVG has both $O(n)$ vertices and edges.

In [32,33], an SPM can be constructed in $O(n \log n)$ time. Since there are $O(n)$ pairs of adjacent regions, the time to generate Steiner point candidates depends on the time to determine the existences of those L-shaped paths. For two vertices, their closest obstacles can be used to determine if there exists an L-shaped path between them in $O(1)$ time. For example, in Figure 5.4(b), $P_2 \leftrightarrow v_2 \leftrightarrow P_3$ is invalid since the down boundary of the up closest obstacle of P_3 (i.e. O_2) is lower than P_2. Besides, for a vertex, if its closet obstacles have been known, the projection time is $O(1)$. Shen et al. [36] proposed an $O(n \log n)$-time method to determine the closest obstacles for n vertices. As a result, the vertex generation takes $O(n \log n)$ time.

For two vertically or horizontally adjacent vertices, if their closest obstacles have been known, it takes $O(1)$ time to determine if they should be connected. For example, in Figure 5.4(c), since v_4 and P_2 have the same right closest obstacle, (v_4, P_2) should be connected. Therefore, after the obstacle determination, the edge connection can be done by sorting those vertices according to (x-coor,y-coor) and (y-coor, x-coor), respectively. To conclude, we have the following theorem.

Theorem 5.3 *An OAVG can be constructed in $O(n \log n)$ time, and has both $O(n)$ vertices and edges.*

5.2.3.2 Steiner Point Selection

We attempt to select good Steiner points from the OAVG. To minimize the wirelength, it is natural to find the closest pin-vertex from an existing connected component. This is also the concept of the well-known *minimum spanning tree* (MST) algorithm, Prim's algorithm [6]. In [21], Li's maze routing also takes the similar concept. Therefore, there are some

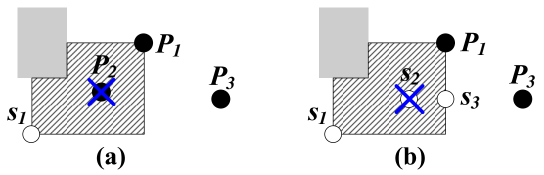

Figure 5.5: P_1 has been reached by source s_1. The stripe region is SPR(s_1,P_1). (a) P_2 must not be inside SPR(s_1,P_1). (b) P_3 must not be reached from s_2.

similar behaviors between Li's maze routing and our Steiner point selection, while the motivations and foundations are quite different. We will discuss the main differences in Section 5.2.5.

For the MST problem, Prim's algorithm can start from any pin-vertex to achieve the same wirelength. However, since the OARSMT problem is NP-complete, a starting vertex will affect the resulting wirelength. In general, the longest connection often significantly affects the final result. Therefore, we select the pin-vertex which is the farthest from the other pin-vertices as a starting vertex since connecting such a pin-vertex more possibly causes the longest connection than the others. Below, we define the farthest pin-vertex formally.

Definition 5.4 *For a set of pin-vertices, a pin-vertex* p *is* ***the farthest pin-verrex*** *if* p *has the longest distance to its* ***neighbor pin-vertex****, which is the closest pin-vertex to* p.

Similar to the reasoning in the Voronoi diagram in [17], a pin-vertex p_1 has at least one neighbor pin-vertex p_2 such that an SPM has two adjacent regions belonging to v_1 and v_2, respectively, and the nearest pin-vertices of v_1 and v_2 are p_1 and p_2, respectively. Therefore, since an SPM has $O(n)$ pairs of adjacent regions, finding the farthest pin-vertex takes $O(n)$ time after the SPM construction . Experimental results in Section 5.3 will show the effectiveness of starting at the farthest pin-vertex.

Based on Prim's concept, our Steiner point selection first sets the farthest pin-vertex as a source, and then repeats the following steps to include Steiner points:

1. Find the closest pin-vertex $\boldsymbol{p}$ from those existing sources.
2. Include the corresponding source $\boldsymbol{s}$ as a Steiner point if $s \notin P$, and construct SPR($\boldsymbol{s}$,$\boldsymbol{p}$).
3. Set the boundary vertices of SPR($\boldsymbol{s}$,$\boldsymbol{p}$) as new sources.

The first step is to find the closest pin-vertex from an existing connected component, and the last two steps are to update the connected component. Unlike the MST problem, there are a lot of non-pin-vertices in the OAVG, i.e., s and p are probably not adjacent to each other. Under this circumstance, we construct SPR(s,p) as a new part of the connected component, instead.

The usage of SPRs is the foundation of our Steiner point selection and very efficient since all vertices inside a constructed SPR must not be a pin-vertex or a source. Take Figure 5.5 for example. In Figure 5.5(a), P_2 must not be inside SPR(s_1,P_1); otherwise, P_2 should be first reached from s_1. In Figure 5.5(b), P_3 must not be reached from s_2 since $|(s_3, P_3)|$ is smaller than $|(s_2, P_3)|$.

The Steiner point selection algorithm is summarized in Figure 5.6, and mainly extends Dijkstra's algorithm [6] to propagate vertices to find the current closest pin-vertex. Figure 5.7 shows the procedure of Figure 5.6 on Figure 5.3(b), where P_3 is the farthest pin-vertex in Figure 5.3(a). Below, we first introduce the pseudo code in Figure 5.6, and then use Figure 5.7 to trace the corresponding procedure.

For a vertex, "sr", "dis", and "pi" represent its source, the distance to its source, and its predecessor on the path to its source, respectively. Lines 1–4 are the initialization,

```
Algorithm: $Steiner\_Point\_Selection(P, V, E, fp, S)$
Input: $P$ /* the set of pin-vertices */
       $V$ /* the set of vertices in OAVG */
       $E$ /* the set of edges in OAVG */
       $fp$ /* the farthest pin-vertiex */
Output: $S$ /* the set of Steiner points */
1  for each vertex $v \in V$
2    $(v.dis, v.pi, v.sr) \leftarrow (\infty, \emptyset, \emptyset)$
3  $(fp.dis, fp.pi, fp.sr) \leftarrow (0, \emptyset, fp)$
4  Insert $fp$ into Min-Heap $H$ whose key is $dis$
5  while $\exists p \in P$, $p.dis \neq 0$
6    $v \leftarrow$ Extra-Min($\boldsymbol{H}$)
7    if $v \notin P$ or $v.dis = 0$
8      for each $(v, u) \in E$
9        if $(v.dis + |(v, u)|,\ v.sr.id) < (u.dis,\ u.sr.id)$
   /* The first element is the primary key.*/
10         $(u.dis, u.pi, u.sr) \leftarrow (v.dis + |(v, u)|, v, v.sr)$
11         if $u$ is not in $H$
12           Insert $u$ into $H$
13   else
14     if $v.sr \notin P$
15       $S \leftarrow S \cup \{v.sr\}$
16     $Q \leftarrow \{v\}$
17     while $Q$ is nonempty
18       $u_1$=Pop($Q$)
19       for each $(u_1, u_2) \in E$
20         if $u_1.sr = u_2.sr$ and $u_2.dis + |(u_1, u_2)| = u_1.dis$
21           $Q \leftarrow Q \cup \{u_2\}$
22         $(u_1.dis, u_1.pi, u_1.sr) \leftarrow (0, \emptyset, u_1)$
23         Insert $u_1$ into $H$
```

Figure 5.6: The Steiner point selection Algorithm.

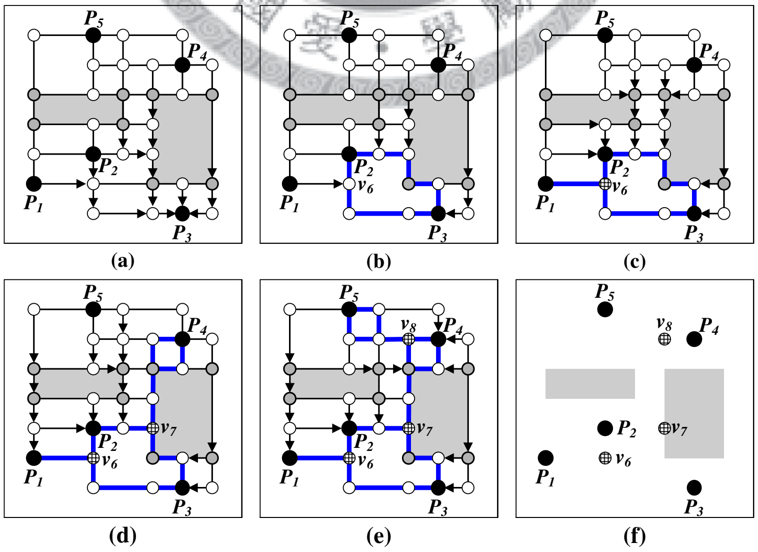

Figure 5.7: The Steiner point selection (Figure 5.6) on Figure 5.3(b). Arrow segments are from vertices to their “pi”. Blue segments are boundaries of constructed SPRs.

"$\exists p \in P$, $p.dis \neq 0$" (Line 5) means that at least one pin-vertex has not been reached, and "$v \notin P$ or $v.dis = 0$" (Line 7) means that v is not an unreached pin-vertex. Lines 8–12 propagate vertices to reach pin-vertices. Lines 14–23 select Steiner points, construct SPRs, and set new sources. In detail, Lines 14–15 select Steiner points, Lines 16–21 build a SPR by backtracking, and Lines 22–23 set new sources. Since there could be more than one source, we use "sr" to record the current source of a vertex.

Since more than one source could have the same distance to a vertex, we compare both distance and source ID in Line 9 to ensure the correctness of the backtracking SPR construction in Lines 16–21. Although Lines 22–23 may insert internal vertices of a constructed SPR into H, they will not be propagated (Lines 10–12 won't perform) since the distances of their adjacent vertices, i.e. vertices on the boundary of the SPR, will also be 0. In other words, only vertices on the boundary of a constructed SPR will be real sources.

As shown in Figure 5.7(a), in the first 11 iterations, Lines 5–12 (Figure 5.6) are performed to propagate P_3 to find the closest pin-vertex P_2. In Figure 5.7(b), P_2 has been reached, SPR(P_2,P_3) has been constructed (Lines 16–21), and vertices of SPR(P_2,P_3) have been inserted into H (Lines 22–23). Noticeably, only vertices on the boundary of an constructed SPR, e.g. v_6 in Figure 5.7(b), will be propagated (Lines 10–12). In Figure 5.7(c), P_1 has been reached from v_6, and v_6 has been selected as a Steiner point (Lines 14–15). By repeating the above process, in Figure 5.7(d) and Figure 5.7(e), P_4 and P_5 have been reached from v_7 and v_8 respectively, which have also been selected as Steiner points. Figure 5.7(f) shows pin-vertices and those selected Steiner points.

5.2.3.3 Initial OARST Construction

An OARST is constructed based on those selected Steiner points in Section 5.2.3.2. According to the idea in Section 5.2.1, we should construct an MTST on the OAVG by specifying both all the pin-vertices and all the selected Steiner points as terminals.

Long et al. [28] proposed an $O(n \log n)$-time 2-step MTST algorithm for an $O(n)$-space graph: (1) terminal path generation and (2) MST construction based on those generated terminal paths. Liu et al. [26] claimed that the second step of Long's MTST construction does not consider the overlapping between different paths, which can lead to better solutions, and proposed an $O(n \log n)$-time path-overlapping scheme. Noticeably, Liu's scheme does not generate terminal paths from a graph since Liu's OARST construction directly generates critical paths.

Therefore, we integrate Long's MTST construction and Liu's path-overlapping scheme into an improved MTST construction. Since some Steiner points may be leafs of the constructed OARST, after the improved MTST construction, non-pin-vertex leafs will be recursively removed . By Theorem 5.3, [28], and [26], our initial OARST construction takes $O(n \log n)$ time.

5.2.3.4 Refinement

A refinement scheme is performed to reduce the redundant segments of the constructed OARST in Section 5.2.3.4. Liu et al. [26] proposed an $O(n \log n)$-time effective refinement scheme. We directly use Liu's refinement scheme.

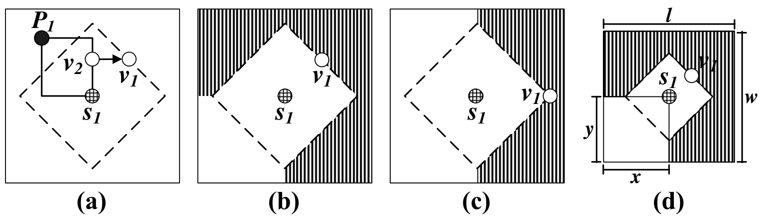

Figure 5.8: The probability of an update operation of v_1 after its first extraction, where s_1 is the original source of v_1. (a) v_1 is updated from v_2 of SPR(P_1,s_1). (b)–(c) P_1 must be in the stripe region. (d) the area of the stripe region.

5.2.4 Time Complexity Analysis

As discussed in Section 5.1, from practical viewpoint, for the OARST construction, the $\Theta(n \log n)$ empirical time complexity is sufficient. To achieve $\Theta(n \log n)$ empirical time complexity, the average-case time complexity of an algorithm tends to be $O(n \log n)$. Therefore, we analyze the average-case time complexity of our algorithm

Since the OAVG construction, the initial OARST construction, and the refinement scheme have been proved to take

$O(n \log n)$ time in the worst case, it is sufficient to analyze the Steiner point selection. The Steiner point selection is extended from Dijkstra's algorithm, and Dijkstra's algorithm takes

$O(|E| \log |V|)$ time using heap structure [6]. This is because in Dijkstra's algorithm, a vertex is extracted from the heap at most once. However, in the Steiner point selection, a vertex can be extracted more than once. Below, we give an intuitive but informal analysis to show that the average number of extract operations for a vertex seems $O(1)$, which implies that the average-case time complexity of our algorithm seems $O(n \log n)$.

If a vertex has been extracted i times, it must have also been inserted into the heap i times. According to the pseudo-code in Figure 5.6 (Lines 10–12 and Lines 22–23), if a vertex has be inserted $i + 1$ times, its distance must be updated after its i-th extract operation. It is clear that those update operations result from the construction of SPRs since the boundary vertices of constructed SPRs will become new sources (Line 22–23). Supposing that s_1 is the source of v_1 at the first extraction of v_1, if v_1 will be updated from SPR(s_1, P_1), the distance between v_1 and SPR(s_1, P_1) must be shorter than that between s_1 and v_1 (Lines 9–10). For example, as shown in Figure 5.8(a), when pin-vertex P_1 has been reached by s_1, v_1 will be updated from v_2 on the boundary of SPR(P_1, s_1) since $|(v_1, v_2)|$ is smaller than $|(v_1, s_1)|$. Since P_1 is reached after the first extraction of v_1, the distance between s_1 and v_1 is at most that between s_1 and p_1.

We first assume there is no obstacle. Considering rectilinear distance (L_1 metric), a concentric circle has become a square whose boundary slopes are 1 or -1, e.g. the dash square in Figure 5.8(a). Since $|(s_1, P_1)|$ is not smaller than $|(v_1, s_1)|$, P_1 must be outside the dash square except the boundaries in Figure 5.8. Furthermore, if v_1 will be updated from SPR(s_1, P_1), the distance between them must be smaller than $|(v_1, s_1)|$. As a result, if v_1 is at the middle of the right-upper boundary of that dash square, P_1 must be located at the stripe region in Figure 5.8(b); if v_1 is at the right corner of the dash square, P_1 must be located at the stripe region in Figure 5.8(c). Since the stripe region in Figure 5.8(c) is smaller than that in Figure 5.8(b), we use Figure 5.8(b) to compute the expected area of the stripe region.

Supposing the whole routing region to be an $l \times w$ rectangle, if s_1 is located at (x,y) according to the left-lower corner of the rectangle, as shown in Figure 5.8(d), the area of the stripe region is smaller than $(l * w - x * y)$. Therefore, the expected area of the stripe region is smaller than $((\int_0^l \int_0^w (l * w - x * y) * dy * dx)/(l * w))$ = $((l * w)^2 - \frac{1}{4} * (l *$

Table 5.1: Comparison between Figure 5.6 and [21].

	Our Steiner point selection	Liang's maze routing
motivation	Select Steiner points	Generate paths
start pin	The farthest one	Nearest to the boundary
propagation	Boundaries of SPRs	Generated paths

$w)^2)/(l * w)$=$\frac{3}{4} * l * w$, implying that the probability of the second insertion of v_1 is $\frac{3}{4}$. With the similar reasoning, supposing Pr_i to be the probability of the i-th insertion, we have $Pr_{i+1} \leq \frac{3}{4}Pr_i$. Since the SPR construction performs at most n-1 times, the expected number of insertions for a vertex is $1 + \frac{3}{4} + \frac{3}{4^2} + \cdots \frac{3}{4^{n-2}} \leq 4$.

In presence of obstacles, the number of valid paths among vertices will be reduced. Therefore, we believe that the probability of the i-th insertion for a vertex will on average be not more than that without considering obstacles although there are still a number of exceptions. To conclude, the above analysis shows that the expected number of insertions (extractions) for a vertex seems $O(1)$, and thus the average-case time complexity of our algorithm seems $O(n \log n)$.

5.2.5 Discussion

In summary, our algorithm is mainly based on the Steiner-point based framework (Section 5.2.1 and the new concept of Steiner point locations (Section 5.2.2). The Steiner-point based framework makes our algorithm close to the essence of the OARSMT problem (Definition 5.1). The integration of Prim's concept [6] and SPRs (Definition 5.2) in Section 5.2.3.2 select good Steiner points to support the framework.

The new concept of Steiner point locations raises the proposal of an OAVG, which contains desirable Steiner points and has both only $O(n)$ vertices and edges. All these help our algorithm to achieve the best practical performance in both wirelength and run time, which will be supported by experimental results later. More importantly, duo to their simplicity, both the Steiner-point based framework and the new concept of Steiner point locations can be naturally extended to the ML-OARSMT problem, the OA-PDST problem, and other RSMT generations

Below, we compare our algorithm with a rectilinear-graph method [21] and spanning-graph methods [22, 26, 28] respectively on (1) routing graph and (2) Steiner tree generation.

Li's extended Hanan grid [21] contains the OARSMT, but has both $O(n^2)$ edges and vertices, which significantly lowers the efficiency. Although an OAVG cannot guarantee the existence of an OARSMT, based on the concept of Steiner point locations, the solution quality of an OAVG will be close to that of Li's extended Hanan grid, which will also be supported by experimental results. Furthermore, an OAVG has only $O(n)$ vertices and edges, which significantly increases the efficiency.

Our Steiner tree construction (steps 2–3) first selects Steiner points and then constructs an OARST based on those Steiner points, while Li's first use maze routing to generate paths as solution components and then constructs an OARST based on those paths, i.e., their solution is limited on those paths. Although Li's maze routing also generates Steiner points, it does not generate all the paths between the pin-vertices and the Steiner points, and there probably exist some other ungenerated paths leading to better solutions. Therefore, since their solution is limited on their generated paths, the solution quality of our Steiner tree construction would be better.

Furthermore, the differences between our Steiner point selection (Figure 5.6) and Li's maze routing are summarized in Table 5.1. In fact, for two vertices, Li's maze routing may generate more than two paths. Backtracking those paths is a critical step of Li's maze routing and very time-consuming. However, according to Figure 5.5, it is sufficient to only propagate from the boundaries of constructed SPRs. In other words, our Steiner point selection is more efficient since it only backtracks the vertices in a constructed SPR instead of multiple paths.

[22, 26, 28] construct a routing graph or generate paths only considering pin-vertices and obstacle corners, while our OAVG further includes desirable Steiner points. Therefore, although all [28], [26], and the OAVG use $O(n)$ edges, the OAVG would contain better solutions. Furthermore, [22, 26, 28] construct an MTST only based on paths between pin-vertices, while our Steiner tree construction considers paths among pin-vertices and selected Steiner points. As a results, the solution quality of our algorithm will be better. Although the difference in the solution quality will probably be reduced after refinement process, the solution quality of our algorithm is still much closer to the optimal. This is very important and meaningful for an NP-complete problem, especially for evaluating a fast heuristic.

5.3 Experimental Results

We have implemented our algorithm in C language, and conducted all the experiments on a Linux server with 8 2.5-GHz AMD processors and 16-GB memory. There are 22 commonly used benchmark circuits, 5 industrial test cases (ind1–ind5) from Synopsys, 12 test cases used in [8] (rc01–rc12), and 5 randomly generated test cases from [22] (rt01–rt05).

We compare our algorithm with those presented in [22], [28], [21] and [26]. To the best of our knowledge, the approaches in [26,28] achieve the best speed performance, and the approach in [21] achieves the best solution quality. We have obtained the executable file in [21], and run their program on our platform. The results of [22,26,28] are directly quoted from those papers, and those algorithms were conducted on a Redhat Linux server with two 2.1-GHz AMD processors and 2-GB memory, a Ubuntu 6.06 server with one 2-GHz AMD-64 CPU and 8-GB memory, a Debian Linux 2.6.18 server with two 2.4-GHz Dual Core Intel processors and 8-GB memory, respectively.

Table 5.2 lists the total wirelengths of these algorithms, and Table 5.3 lists the CPU times, where "speedup" compares the run time of [21] and our algorithm. Since those results might have already been close to the optimal, for an NP-complete problem, it is very meaningful to compare the differences from the optimal solution. Since the RSMT is definitely a lower bound of an OARSMT instance, we list the wirelengths of the RSMTs without considering the obstacles in the second column of Table 5.2, which were generated by GeoSteiner [43].

Compared with [21], which achieves the best solution quality, our algorithm obtains the same solution quality, but runs much faster. As shown in Table 5.3, across all the 22 benchmarks, our algorithm achieves 26.67 times speedup on average over that in [21]. For rc12, which contains up to 10 000 obstacles, our algorithm takes only 1.565 seconds, while that in [21] takes about 142 seconds. The improvement in run time is very significant considering the large and increasing number of obstacles and signal nets in a modern IC design.

Furthermore, since the extended Hanan grid [21] contains the optimal solution, those wirelength results show that the solution quality of our proposed OAVG is very close to the

Table 5.2: Comparison on the Total Wirelength. “our*(H)”: our algorithm starts at the best pin-vertex.

Test	GeoSteiner	Total Wirelength						Imp.(%)($\frac{X-F}{X}$ / $\frac{X-F}{X-A}$)				$\frac{F-H}{F}$
Cases	[43] (A)	[22] (B)	[28] (C)	[26] (D)	[21] (E)	ours (F)	ours*(H)	(X=B)	(X=C)	(X=D)	(X=E)	
ind1	604	632	639	626	619	604	604	4.43 / 100.00	5.48 / 100.00	3.51 / 100.00	2.42 / 100.00	0.00
ind2	9,100	9,600	10,000	9,700	9,500	9,600	9,600	0.00 / 0.00	4.00 / 44.44	1.03 / 16.67	-1.05 / -25.00	0.00
ind3	587	613	623	600	600	600	600	2.12 / 50.00	3.69 / 63.89	0.00 / 0.00	0.00 / 0.00	0.00
ind4	1,078	1,121	1,126	1,095	1,096	1,092	1,092	2.59 / 67.44	3.02 / 70.83	0.27 / 17.65	0.36 / 22.22	0.00
ind5	1,295	1,364	1,379	1,364	1,360	1,374	1,353	-0.73 / -14.49	0.36 / 5.95	-0.73 / -14.49	-1.03 / -21.54	1.53
rc01	25,290	26,900	27,540	26,740	25,980	26,040	25,980	3.20 / 53.42	5.45 / 66.67	2.62 / 48.28	-0.23 / -8.70	0.23
rc02	39,170	42,210	41,930	42,070	42,010	41,570	41,350	1.52 / 21.05	0.86 / 13.04	1.19 / 17.24	1.05 / 15.49	0.53
rc03	51,900	55,750	54,180	54,550	54,390	54,620	54,360	2.03 / 29.35	-0.81 / -19.30	-0.13 / -2.64	-0.42 / -9.24	0.48
rc04	54,910	60,350	59,050	59,390	59,740	59,860	59,530	0.81 / 9.01	-1.37 / -19.57	-0.79 / -10.49	-0.20 / -2.48	0.55
rc05	71,260	76,330	75,630	75,430	74,650	74,770	74,720	2.04 / 30.77	1.14 / 19.68	0.87 / 15.83	-0.16 / -3.54	0.07
rc06	76,356	83,365	86,381	81,903	81,607	81,854	81,290	1.81 / 21.56	5.24 / 45.16	0.06 / 0.88	-0.30 / -4.70	0.69
rc07	105,003	113,260	117,093	111,752	111,542	111,211	110,991	1.81 / 24.82	5.02 / 48.65	0.48 / 8.02	0.30 / 5.06	0.20
rc08	107,416	118,747	122,306	118,349	115,931	116,132	115,516	2.20 / 23.08	5.05 / 41.46	1.87 / 20.28	-0.17 / -2.36	0.53
rc09	105,698	116,168	119,308	114,928	113,460	113,559	113,254	2.25 / 24.92	4.82 / 42.24	1.19 / 14.83	-0.09 / -1.28	0.27
rc10	161,790	170,690	167,978	167,540	167,620	167,460	166,970	1.89 / 36.29	0.31 / 8.37	0.05 / 1.39	0.10 / 2.74	0.29
rc11	-	236,615	232,381	234,097	235,283	236,018	234,875	0.25 / -	-1.57 / -	-0.82 / -	-0.31 / -	0.48
rc12	-	789,097	842,689	780,528	761,606	762,435	758,717	3.38 / -	9.52 / -	2.32 / -	-0.11 / -	0.49
rt01	1,817	2,267	2,362	2,259	2,231	2,193	2,193	3.26 / 16.44	7.15 / 31.01	2.92 / 14.93	1.70 / 9.18	0.00
rt02	44,214	48,441	52,218	48,684	47,297	47,488	46,965	1.97 / 22.55	9.06 / 59.10	2.46 / 26.76	-0.40 / -6.20	1.10
rt03	7,579	8,368	8,645	8,347	8,187	8,231	8,136	1.64 / 17.36	4.79 / 38.84	1.39 / 15.10	-0.54 / -7.24	1.15
rt04	7,634	10,306	10,580	10,221	9,914	9,893	9,832	4.01 / 15.46	6.49 / 23.32	3.21 / 12.68	0.21 / 0.92	0.62
rt05	42,608	53,993	55,286	53,745	52,473	52,509	52,318	2.75 / 13.03	5.02 / 21.90	2.30 / 11.10	-0.07 / -0.36	0.36
Average								2.06 / 28.10	3.76 / 35.28	1.15 / 15.70	0.05 / 3.15	0.44

optimal. From another viewpoint, since the OAVG cannot guarantee the existence of the optimal solution, those wirelength results reflect that our Steiner-point based framework seems more effective. In particular, our algorithm achieves the lower bound of "ind1," i.e., obtains the optimal solution. This also gives another strong evidence to figure out the high effectiveness of our algorithm.

Compared with [26, 28], which achieve the best speed performance, our algorithm achieves the same speed performance since the ratios of their CPU times to ours do not increase with the problem size, and even improves 3.76% and 1.15% in wirelength on average, respectively. Furthermore, considering the differences from the lower bound of the optimal solution, the respective average improvements are up to 35.28% and 15.70% compared with [28] and [26], respectively These improvements are very important and meaningful for an NP-complete problem, especially for evaluating a fast heuristic. To conclude, all the above results support the discussion in Section 5.2.5.

As shown in Figure 5.9, the CPU time of our algorithm is plotted as a function of the input size n. By the least squares fitting on the log-log-axes, the respective slope of the fitting line is 1.17, implying that the empirical time complexity of our algorithm is $\Theta(n^{1.17})$. To obtain the empirical time complexity of a $\Theta(n \log n)$ algorithm, we generates 22 vertices whose x-coordinates are n of those 22 test cases and whose y-coordinates are $n \log n$. Therefore, using the least squares fitting on the log-log-axes of those vertices, we conclude that the empirical time complexity of an $\Theta(n \log n)$ algorithm is $\Theta(n^{1.15})$, implying that the empirical time complexity of our algorithm can be viewed as $\Theta(n \log n)$. This analysis strongly supports our time complexity analysis in Section 5.2.4 that the average-case time complexity of our algorithm seems $O(n \log n)$.

Table 5.3: Comparison on the CPU time.

Test Cases	m / k	n	CPU Time (sec.)					speedup
			[22]	[28]	[26]	[21] (E)	ours (F)	($\frac{E}{F}$)
ind1	10/32	138	<0.01	0.01	0.001	0.008	0.001	8.00x
ind2	10/43	182	<0.01	0.01	0.001	0.029	0.001	29.00x
ind3	10/50	210	<0.01	0.01	0.001	0.008	0.001	8.00x
ind4	25/79	341	<0.01	0.02	0.002	0.009	0.002	4.50x
ind5	33/71	317	0.01	0.02	0.002	0.010	0.003	3.33x
rc01	10/10	50	<0.01	0.01	<0.001	0.038	0.001	38.00x
rc02	30/10	70	<0.01	0.01	0.001	0.089	0.001	89.00x
rc03	50/10	90	<0.01	0.01	0.001	0.077	0.001	77.00x
rc04	70/10	110	<0.01	0.02	0.001	0.098	0.002	49.00x
rc05	100/10	140	0.01	0.02	0.001	0.080	0.002	40.00x
rc06	100/500	2,100	0.24	0.13	0.017	0.521	0.032	16.28x
rc07	200/500	2,200	0.43	0.15	0.026	0.667	0.039	17.10x
rc08	200/800	3,400	0.83	0.27	0.043	1.175	0.063	18.65x
rc09	200/1,000	4,200	0.91	0.36	0.049	1.445	0.084	17.20x
rc10	500/100	900	0.62	0.08	0.016	0.258	0.021	12.29x
rc11	1,000/100	1,400	3.15	0.14	0.021	0.760	0.038	20.00x
rc12	1,000/10,000	41,000	118.52	5.88	0.681	142.393	1.565	90.99x
rt01	10/500	2,010	0.06	0.12	0.017	0.190	0.028	6.79x
rt02	50/500	2,050	0.11	0.11	0.018	0.559	0.035	15.97x
rt03	100/500	2,100	0.47	0.13	0.020	0.180	0.039	4.62x
rt04	100/1,000	4,100	0.95	0.42	0.040	0.322	0.062	5.19x
rt05	200/2,000	8,200	2.06	1.43	0.078	2.603	0.164	15.87x
Average								26.67x

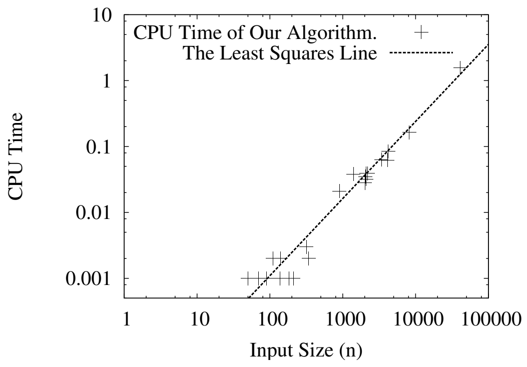

Figure 5.9: The CPU time is plotted as a function of n.

To show the effectiveness of starting at the farthest pin-vertex, we start our Steiner point selection at each pin-vertex for each test case, and list the best results in the 8th column, “ours*(H)”, of Table 5.2. As shown in the 13th column of Table 5.2, starting at the farthest pin-vertex causes only 0.44% more wirelength on average than starting at the best one, which supports the discussion in Section 5.2.3.2. Besides, compared with [21,22,26,28], the results of “ours*(H)” are the best for 17 benchmarks. Since those 22 test cases are commonly used benchmarks, these extensive experimental results can help the future researches for the OARSMT problem.

Chapter 6

Preferred Direction Evading Graph and Approximation Guarantee

As the first work for the OAPD-ST problem, this chapter attempts to analyze the structure of the optimal solution, and proposes an approximation guarantee, both of which very probably help the development of future algorithms. For the structure of the optimal solution, we propose *preferred direction evading graph* (PDEG), and prove that at least one optimal solution exists in PDEG. The theoretical optimality proof provides a way to analyze the solution quality, which significantly help the development of algorithms, especially for approximation ones. For the approximation guarantee, we analyze the Steiner ratio of an OAPD-MST (Definition 2.11) based on PDEG. The analysis results in a factor 2 approximation algorithm for the OAPD-ST problem, and gives important features to support strong heuristics. Experimental results show that our algorithm improves 43.53% on average in total cost compared with a construction-and-correction method when the cost of routing resource doubles between layers, and performs more stably.

6.1 Motivation

As the technology advances into nanometer era, there are more and more constraints at the routing stage such as multiple routing layers, obstacles, preferred directions, different routing resources, and via costs. However, none of existing approaches catches all the mentioned constraints since each of those constraints is hard to handle. Therefore, it is desired to give a comprehensive study of the OAPD-ST problem such as the structure of the optimal solution and the approximation guarantee. The structure of the optimal solution can provides a way to analyze the solution quality, which is critical for the development of algorithms, especially for approximation ones. The approximation guarantee can give important features to support the development of strong heuristics. For example, since an OARSMT can approximate an OARSMT, MTST-based algorithms perform well for the OARSMT problem.

6.2 Preferred Direction Evading Graph

In this section, we propose a new routing graph, called *preferred direction evading graph* (PDEG), for the OAPD-ST problem. Then, we prove that PDEG contains at least one optimal solution, and discuss the contributions of our proofs.

6.2.1 Procedure of PDEG Construction

Definition 6.1 *Given an instance of the OAPD-ST problem,* **PDEG** *is an undirected connected graph connecting all the pin-vertices and some other vertices, where no edges intersect any obstacle or violate the PD constraints.*

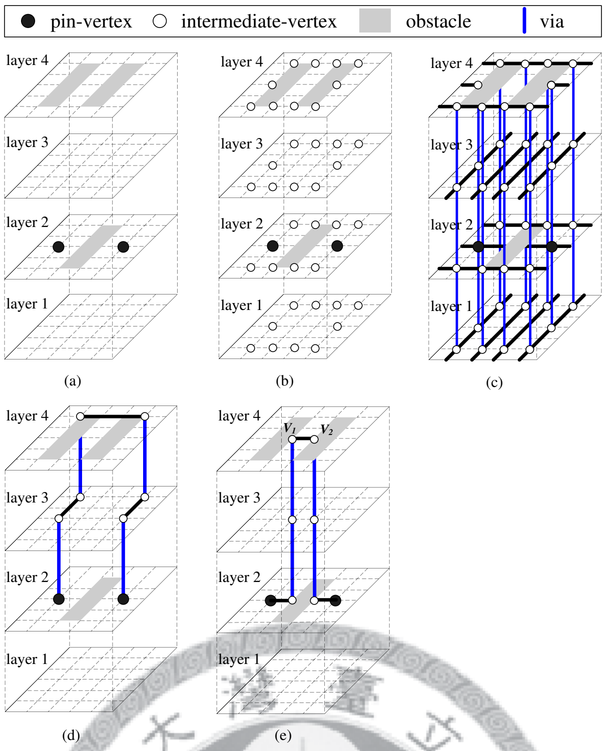

Figure 6.1: Observations for the OAPD-ST problem. (a) an OAPD-ST problem instance. (b) vertex projections. (c) edge construction (primitive PDEG). (d) the optimal solution of primitive PDEG. (e) the optimal solution of (a). ($C_v = 1$ and $UC_i = 1, 1 \leq i \leq N_l$.)

6.2.1.1 Observation

To attack the OAPD-ST problem, we typically generate a routing graph. First, we recursively project pin-vertices and obstacle corners to adjacent layers until the projection is blocked by an obstacle as shown in Figure 6.1(b). Second, we extend line segments from pin-vertices, obstacle corners and those projection vertices to meet the PD constraints. Finally, we construct vias on intersections of the extending line segments between layers, and a routing graph, called *primitive PDEG*, is constructed in Figure 6.1(c).

Although primitive PDEG is applicable to the OAPD-ST problem, it cannot guarantee the optimality. For example, assuming C_v and UC_i ($1 \leq i \leq N_l$) to be 1, Figure 6.1(d)

```
**Algorithm:** Vertex-Generation($P$, $O$, $N_l$, $G$)
**Input:** $P$ /* the set of pin-vertices */
       $O$ /* the set of obstacles */
       $N_l$ /* the number of routing layers */
       $G = (V, E)$ /* the PDEG */
1  $C$ = the set of the corners of $O$
2  $V' = P \bigcup C$
3  $X$ = the set of distinct x-coordinates of $V'$
4  $Y$ = the set of distinct y-coordinates of $V'$
5  $U = \emptyset$
6  **for** $i$ = 1 to $N_l$
7     **for** each $x \in X$ and each $y \in Y$
8        $U \leftarrow (x, y, i)$
9  $V = V' \bigcup U$
10 **for** each obstacle $o \in O$
11    remove all vertices inside $o$ from $V$
```

Figure 6.2: The Vertex-Generation algorithm.

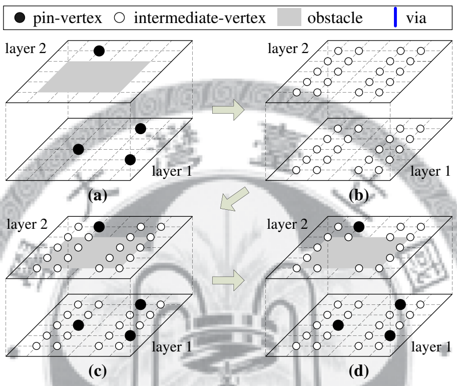

Figure 6.3: Vertex generation. (a) a problem instance. (b) possible candidates of essential vertices. (c) the union of those vertices. (d) the vertices of PDEG.

shows the optimal solution of primitive PDEG in Figure 6.1(c) with cost of 11. However, the optimal solution to the instance in Figure 6.1(a) is shown in Figure 6.1(e) and has a cost of 7, which implies that primitive PDEG cannot guarantee the existence of an optimal solution. This is because the primitive PDEG loses some essential vertices, such as V_1 and V_2 in Figure 6.1(e).

From the above analysis, we observe that vertices with the same x-coordinate and y-coordinate as pin-vertices and obstacle corners are possible candidates of essential vertices of an optimal solution. To guarantee an optimal solution, PDEG construction should contains all the possible candidates.

```
Algorithm:PDEG-Construction($P$, $O$, $N_l$, $G$)
Input: $P$ /* the set of pin-vertices */
       $O$ /* the set of obstacles */
       $N_l$ /* the number of routing layers */
       $G = (V, E)$ /* the PDEG */
1  $V = \emptyset$
2  $E = \emptyset$
3  Vertex-Generation($P$, $O$, $N_l$, $G$)
/* $V$ can be represented by a matrix, and*/
/* $V[i][j][k]$ denotes a vertex of PDEG having*/
/* the $i/j$-th $x/y$-coordinate in $k$ layer.*/
/* In Figure 6.5(b), $z$ represents $V[4][5][1]$.*/
4  for $k$=1 to $N_l$
5    for $i$=1 to the number of distinct x-coordinates of PDEG
6      for $j$=1 to the number of distinct y-coordinates of PDEG
7        if layer $k$ only allows horizontal line
8          if $(V[i][j][k], V[i+1][j][k])$ is valid
9            $E \leftarrow (V[i][j][k], V[i+1][j][k])$
10       else
11         if $(V[i][j][k], V[i][j+1][k])$ is valid
12           $E \leftarrow (V[i][j][k], V[i][j+1][k])$
13 for $k$=1 to $N_l - 1$
14   for $i$=1 to the number of distinct x-coordinates of PDEG
15     for $j$=1 to the number of distinct y-coordinates of PDEG
16       if $(V[i][j][k], V[i][j][k+1])$ is valid
17         $E \leftarrow (V[i][j][k], V[i][j][k+1])$
```

Figure 6.4: The PDEG-Construction algorithm.

6.2.1.2 Vertex Generation

The vertex generation algorithm of PDEG is summarized in Figure 6.2, and Figure 6.3 shows an example. Since PDEG should contain all possible candidates of essential vertices, x-coordinates and y-coordinates from pin-vertices and obstacle corners are collected (lines 1–4); then all possible candidates are generated from those coordinates (lines 5–8) as shown in Figure 6.3(b). After that, pin-vertices, obstacle corners, and possible candidates are combined to be a vertex set (line 9) as shown in Figure 6.3(c). Finally, invalid vertices of the vertex set, which locate inside obstacles, are removed (lines 10–11), and then the vertices of PDEG have been generated as shown in Figure 6.3(d).

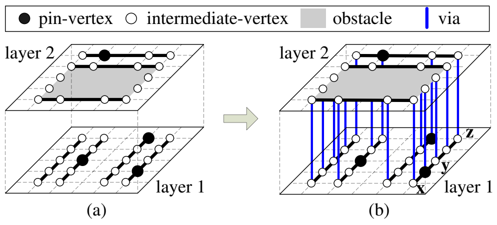

Figure 6.5: PDEG construction. (a) line segment construction. (b) via connection construction (the resulting PDEG).

6.2.1.3 PDEG Construction

The whole process of PDEG construction is summarized in Figure 6.4, and Figure 6.5 is an example of the remaining parts, which succeeds Figure 6.3. To connect these vertices completely after vertex generation (lines 1–3), there are two steps for constructing edges: one is within a layer, and the other is between layers. Figure 6.5(a) shows the result for connecting edges within a layer considering the PD constraints (lines 4–12). Then, edges (vias) between layers have been connected (lines 13–17) as shown in Figure 6.5(b). To conclude, Figure 6.3 and Figure 6.5 show the whole process of PDEG construction for a 2-layer example, and Figure 6.5(b) shows the resulting PDEG.

Since N_l is a small number in practice, we conclude the following complexity from the algorithms in Figure 6.2 and Figure 6.4.

Theorem 6.1 *The space complexity of PDEG is $O(n^2)$, and the time complexity of the PDEG construction is $O(n^2)$.*

We make definitions to formally describe PDEG. A *line segment* (*via connection*) is a rectilinear edge connecting two vertices in the same layer (in different layers). A line segment (via connection) can span more than two vertices. See layer 1 of Figure 6.5(b), each of $\{(x, y), (y, z), (x, z)\}$ is a line segment.

Definition 6.2 *An* ***evading line segment (ELS)*** *is a* ***maximal*** *line segment of PDEG, and an* ***evading via connection (EVC)*** *is a* ***maximal*** *via connection of PDEG.* In Figure 6.5(b), (x, y) is not an ELS since (x, y) is *not maximal*, while (x, z) is one.

By Definition 6.2, PDEG is a union of evading line segments, evading via connections, and the intersections of them.

6.2.2 The Optimality of PDEG

We will prove *the optimality of PDEG*, i.e., if there exists an OAPD-SMT for the OAPD-ST problem, at least one must exist in PDEG. The key idea of our proof is to move an OAPD-SMT to PDEG without increasing the cost. Since PDEG is a union of ELSs and EVCs, it is equivalent to moving all the line segments and via connections of an OAPD-SMT to ELSs and EVCs respectively without increasing the cost.

We develop two mathematic skills for our proof: (1) a two-stage movement: move line segments first and then move via connections (supported by Lemma 6.2) and (2) an informative structure: preferred direction component (defined in Definition 6.4). Since we want to move all line segments and all via connections to ELSs and EVCs respectively, we define the states of a line segment and a via connection as follows.

Definition 6.3 *A line segment (via connection) is* ***evaded*** *if and only if an ELS (EVC) contains it.*

When moving an OAPD-SMT, the first question is the order of movement. Figure 6.6 shows an invalid movement for a via connection. In Figure 6.6(a), line segments connected to the unevaded maximal via connection $\boldsymbol{s}$, union of v_1 and v_2, have different

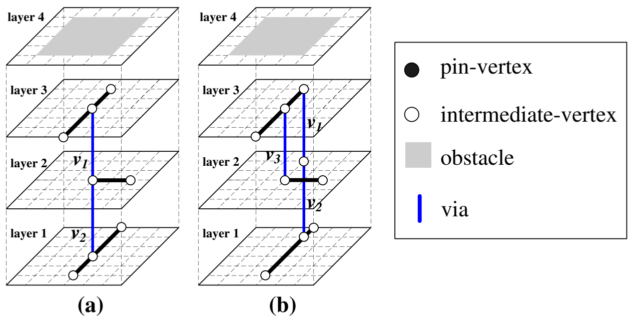

Figure 6.6: An invalid movement for a via connection. (a) a maximal via connection, i.e. $v_1 \cup v_2$, connected to line segments with different directions. (b) movement of (a) will increase the cost.

directions. As shown in Figure 6.6(b), moving s will increase the cost because another via connection, v_3, must be inserted to connect all the elements. In other words, for an OAPD-SMT, if line segments connected to a via connection have different directions, this via connection cannot be moved without increasing the cost. However, it will be clear in Lemma 6.2 that if all the line segments are evaded, i.e., all the line segments have been moved to ELSs, line segments connected to an unevaded via connection have the same direction. Therefore, our two-stage movement is applicable.

Lemma 6.2 *For an unevaded maximal via connection s, if all the line segments connected to s are evaded, those line segments have the same direction (horizontal or vertical).*

Proof: Assume the line segments connected to s have different directions. Since all the line segments connected to s are evaded, s is intersected by ELSs with different directions. According to PDEG construction in Figure 6.4, s must be contained by an EVC, contradicting that s is unevaded. Therefore, all the line segments connected to s have the same direction. $\square$

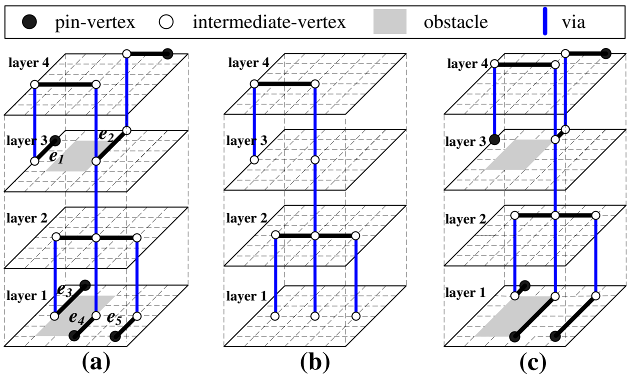

Figure 6.7: An example for the definition and movement of a PDC. (a) an OAPD-ST. (b) a horizontal PDC of (a). (c) an OAPD-ST which is generated from (a) by moving (b) up. ($UC_1 = 1$ and $UC_3 = 2$.)

Furthermore, the OAPD-ST problem deals with multiple routing layers, preferred directions, and different routing resources such that it is more complicated than the OA-RSMT problem. As a result, we need to propose a novel and informative structure to simultaneously move line segments both within a layer and between layers, and via connections.

Definition 6.4 *A **preferred direction component** (PDC) is a **maximal connected subgraph** whose elements have the same either x-coordinate or y-coordinate. A PDC with the same x-coordinate is **vertical**; otherwise, one with the same y-coordinate is **horizontal**.*

Figure 6.7(a) shows an OAPD-ST (not an OAPD-SMT), and Figure 6.7(b) shows a horizontal PDC of (a). All elements of the PDC in Figure 6.7(b) have the same x-coordainte. Lemma 6.3 will clarify that PDCs are useful for moving an OAPD-SMT.

Definition 6.5 *For a PDC s, $U_{cost}(s)$ is the total unit cost of line segments which are connected to s in the **up** side. Similarly, we can define $D_{cost}(s)$, $L_{cost}(s)$, $R_{cost}(s)$.*

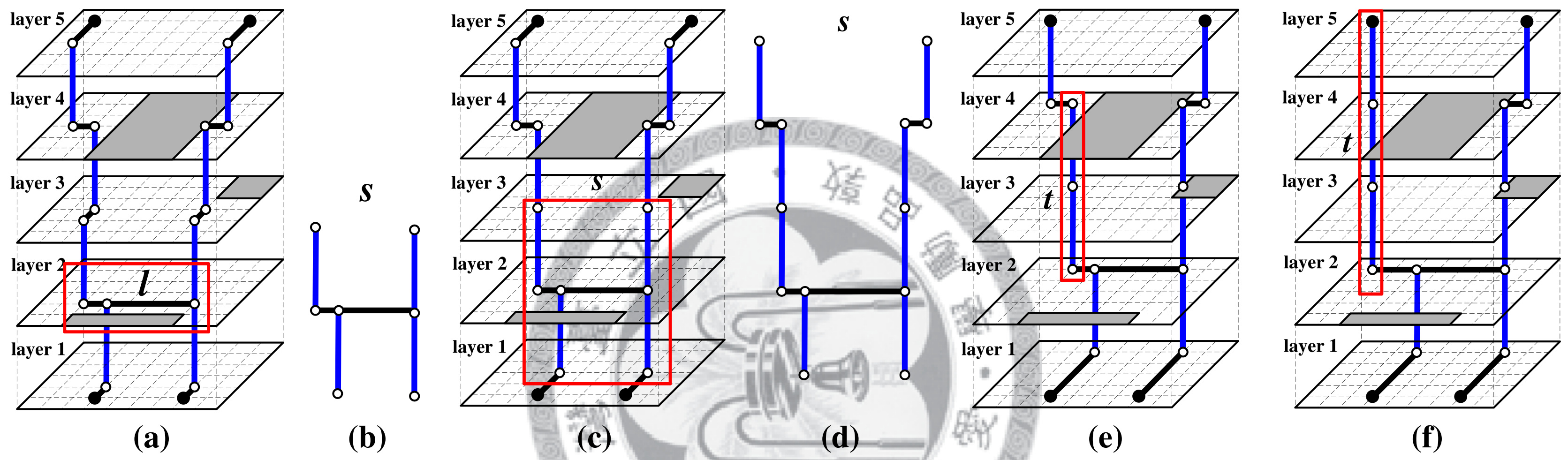

Figure 6.8: Example for the proof of Theorem 6.4. (a) an OAPD-SMT and an unevaded line segment $\boldsymbol{l}$. (b) a PDC of (a) contain $\boldsymbol{l}$. (c) another OAPD-SMT generated from (a) by moving (b) up. (d) a PDC by extending (b) in (c). (e) another OAPD-SMT generated from (c) by moving (d) up and an unevaded maximal via connection $\boldsymbol{t}$ enclosed by a rectangular frame. (f) another OAPD-SMT generated from (e) by moving $\boldsymbol{t}$ left.

We use Figure 6.7 to explain Definition 6.5 and illustrate a movement. For the horizontal PDC $\boldsymbol{s}$ in Figure 6.7(b), as shown in Figure 6.7(a), $\{e_1, e_2, e_3\}$ are line segments connected to $\boldsymbol{s}$ in the up side, and $\{e_4, e_5\}$ are in the down side. We assume that $UC_1 = 1$ and $UC_3 = 2$; hence we have $\boldsymbol{U_{cost}(s)}$ = 5 and $\boldsymbol{D_{cost}(s)}$ = 2. Since $\boldsymbol{s}$ contains no pin-vertices and leans against no obstacles, $\boldsymbol{s}$ can be freely moved. Moreover, $U_{cost}(s)$ is larger than $D_{cost}(s)$, implying that we can move $\boldsymbol{s}$ up 2 units to reduce the cost as shown in Figure 6.7(c), and the cost reduction is 6 (by $2 * (U_{cost}(s) - D_{cost}(s))$). Since one of $U_{cost}(s) \leq D_{cost}(s)$ and $U_{cost}(s) \geq D_{cost}(s)$ ($R_{cost}(s) \leq L_{cost}(s)$ and $L_{cost}(s) \geq R_{cost}(s)$) must be true for a horizontal (vertical) PDC, we have the following lemma.

Lemma 6.3 *For a PDC $\boldsymbol{s}$ of an OAPD-ST, if $\boldsymbol{s}$ contains no pin-vertices and leans against no obstacles, $\boldsymbol{s}$ can be moved without increasing the cost.*

By Lemma 6.2 and Lemma 6.3, in the proof of Theorem6.4, we will first use PDCS to move all the line segments to PDEG, and then move via connections.

Theorem 6.4 *If there exists an OAPD-SMT for an OAPD-ST problem instance, at least one exists in PDEG.*

Proof: Suppose there exists an OAPD-SMT $\boldsymbol{T}$ such that T contains a minimal total number of unevaded line segments and unevaded via connections among all OAPD-SMTs.

Without loss of generality, suppose $\boldsymbol{l}$ is an unevaded horizontal line segment of T and contained by a horizontal PDC $\boldsymbol{s}$ of T. For example, Figure 6.8(a) shows an OAPD-SMT T and an unevaded line segment l; Figure 6.8(b) shows a PDC s of T which contains l in Figure 6.8(a). According to PDEG construction in Figure 6.4, if a PDC contains a pin-vertex or leans gainst an obstacle, all line segments of this PDC must be evaded. Hence, in order to move l to be evaded, we attempt to move s to contain a pin-vertex or lean against an obstacle.

By Lemma 6.3, s can be moved without increasing the cost; otherwise, s contains a pin-vertex or leans against an obstacle, implying that the line segments of s including l are evaded. Since our movement would change the topology of T, during the movement, s would not be a maximal connected subgraph with the same x-coordinate, i.e., a horizontal PDC. For example, Figure 6.8(c) shows another OAPD-ST generated from Figure 6.8(a) by moving Figure 6.8(b) up. In Figure 6.8(c), Figure 6.8(b) is not a PDC since Figure 6.8(d) contains Figure 6.8(b), i.e., Figure 6.8(b) is a maximal connected subgraph with the same x-coordinate.

Since Lemma 6.3 holds only for a PDC, if s has become not a PDC, moving s could increase the cost. At that time, we extend s to be a PDC again and then continue the movement. For example, we extend Figure 6.8(b) of Figure 6.8(c) to Figure 6.8(d) and then move Figure 6.8(d) up to obtain Figure 6.8(e) without increasing the cost. At last, we have moved s to contain a pin-vertex or lean against an obstacle without increasing the cost. In other words, we can move line segments of s including l to be evaded without increasing the cost.

For the same reason, we can move all the unevaded line segments of T to be evaded using PDCs without increasing the cost. Now, we only need to move all the unevaded via connections of T to be evaded without increasing the cost.

Let $\boldsymbol{t}$ be an unevaded maximal via connection of T. By Lemma 6.2, all the line segments connected to $\boldsymbol{t}$ have the same direction. Without loss of generality, assume that the line segments connected to $\boldsymbol{t}$ are horizontal. For example, Figure 6.8(e) shows an unevaded maximal via connection $\boldsymbol{t}$, and all the line segments connected to t are horizontal.

It is clear that t contains no pin-vertex or horizontally leans against no obstacles; otherwise, t must be evaded according to the PDEG construction in Figure 6.4. Hence, we try to move t to contain a pin-vertex or horizontally lean against an obstacle. Similar to the reasoning in Lemma 6.3, we can move t without increasing the cost and then extend t when t is not maximal. For example, we can move t in Figure 6.8(e) left and obtain Figure 6.8(f). Finally, we have moved t to be evaded without increasing the cost.

Similarly, we can move all the unevaded maximal via connections of T to be evaded without increasing the cost. The resulting OAPD-ST has all the line segments and all the via connections to be evaded, and has the same cost with the original OAPD-SMT. To conclude, if there exists an OAPD-SMT, at least one exists in PDEG.

$\square$

6.2.3 Theoretical Contribution

Theorem 6.4 proves that even if the OAPD-ST problem has many constraints, the optimal solution can be restricted in a simple graph. By Theorem 6.4, the OAPD-ST problem can be reduced to the Steiner tree problem in PDEG. In other words, Theorem 6.4 reduces the infinite geometry solution space of the OAPD-ST problem to $O(n^2)$ graph. Hence, using PDEG as the solution space, more efficient methods could be generated.

Although PDEG could be used intuitively, Theorem 6.4 still has theoretical contributions. The direct theoretical contribution is to make solutions generated from PDEG more convincing. In general, brute-force solutions are not convincing. Since Theorem 6.4 supports that PDEG contains an OAPD-SMT, solutions generated from PDEG seem more convincing.

The main theoretical contribution of Theorem 6.4 is to help the design of future algorithms for the OAPD-ST problem, especially for approximation algorithms. Solution quality is the most important element of an algorithm. Comparison with the optimal solution is a general method to evaluate solution quality. For an approximation algorithm, the key step is to find the upper (lower) bound of the approximation solution to the optimal solution. Since PDEG contains an optimal solution, PDEG provides a way to analyze solution quality. In other words, through PDEG, we can evaluate the solution quality to help the design of future algorithms, especially for approximation algorithms. Our approximation algorithm in Section 6.3 is one example.

6.3 Approximation Algorithm

In this section, we propose a factor 2 approximation algorithm for the OAPD-ST problem, which constructs an OAPD-MST in $O(n^2logn)$ time. The critical step to design an approximation algorithm is to compare the approximation solution and the optimal solution. To compare an OAPD-MST and an OAPD-SMT, we first prove that at least one OAPD-MST exists in PDEG in Section 6.3.1. Then, we describe the approximation algorithm and derive the approximation guarantee in Section 6.3.2. Finally, we discuss the contributions of the approximation guarantee in Section 6.3.3.

6.3.1 Existence of OAPD-MST in PDEG

Theorem 6.5 *If there exists an OAPD-MST for an instance of the OAPD-ST problem, at least one must exist in PDEG.*

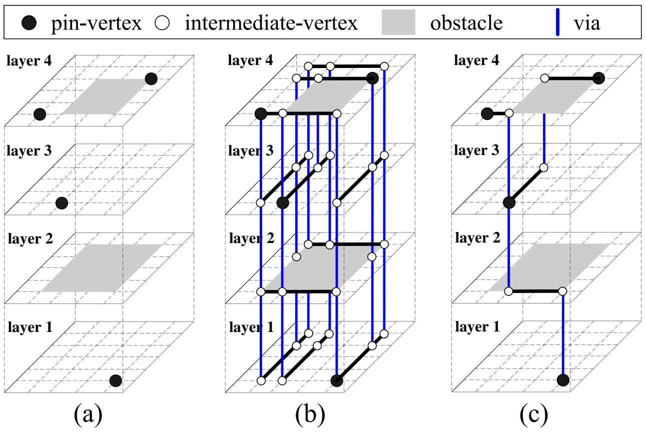

Figure 6.9: Procedure of our approximation algorithm. (a) an instance. (b) PDEG construction. (c) OAPD-MST construction. (UC_i=1, 1$\leq$ i$\leq N_l$ and C_v=1.)

Proof: By Definition 2.11, a graph including all OAPD-SPs among pin-vertices contains at least one OAPD-MST. Therefore, we only need to prove that for any two pin-vertices, at least one OAPD-SP between them exists in PDEG. Assume s is an OAPD-SP between two pin-vertices. Using the method of proving the optimality of PDEG in Theorem 6.4, we can move all the line segments and all the via connections of s to be evaded without increasing the cost. That is, we can construct an OAPD-SP between the two pin-vertices in PDEG with the same cost as s.

Therefore, for any two pin-vertices, at least one OAPD-SP between them exists in PDEG. To conclude, if there exists an OAPD-MST, at least one must exist in PDEG. □

6.3.2 Approximation Guarantee

Our approximation algorithm consists of two steps: (1) PDEG construction. (2) OAPD-MST construction.

PDEG construction has been shown in Section 6.2.1. See Figure 6.9(b) for instance.

After PDEG construction, we weight edges as follows to transform practical costs to PDEG:

1. An edge e within a layer i: $Cost(e) = length(e) * UC_i$.
2. An edge e between two adjacent layers: $Cost(e) = C_v$.

By the weight assignment, Theorem 6.4, and Theorem 6.5, an SMT and an MTST (Definition 2.9) in PDEG are an OAPD-SMT and an OAPD-MST respectively.

As a result, OAPD-MST construction can be done by applying an MTST algorithm on PDEG. In [31], Mehlhorn proposed an algorithm to find an MTST in $O(|E|+|V|\log|V|)$ time. By Theorem 6.1, finding an MTST on PDEG takes $O(n^2 \log n)$ time, implying the following.

Theorem 6.6 *The time complexity of our approximation algorithm is* $O(n^2 \log n)$.

Finally, we prove the approximation guarantee of our approximation algorithm.

Theorem 6.7 *For an OAPD-ST problem instance, the cost of an OAPD-MST is no more than twice the cost of an OAPD-SMT.*

Proof: The above discussion directly infers the two equations:

$$\textit{Cost(OAPD-MST) = Cost(MTST-PDEG).} \tag{1}$$

$$\textit{Cost(OAPD-SMT) = Cost(SMT-PDEG).} \tag{2}$$

In [41], for a graph with terminals and nonterminals,

$$\textit{Cost(MTST)} \leq \textit{2*Cost(SMT)} \tag{3}$$

By (1)–(3), we conclude the following inequality equations:

Cost(OAPD-MST) = Cost(MTST-PDEG)

$$\leq$$

*2*Cost(SMT-PDEG) = 2*Cost(OAPD-SMT).*

Therefore, *Cost(OAPD-MST)* $\leq$ *2*Cost(OAPD-SMT).* □

6.3.3 Discussion

It is clear that applying an MTST algorithm on PDEG is directly a factor 2 approximation algorithm for the OAPD-ST problem. The proof for the approximation guarantee of an OAPD-MST (Theorem 6.7) has further contributions.

First, Theorem 6.7 gives a strong motivation to develop more efficient OAPD-MST algorithms. Our algorithm generates an OAPD-MST from PDEG, but generating an OAPD-MST does not necessarily use PDEG. That is, there could be other more efficient OAPD-MST algorithms. If we develop an $O(n \log n)$-time OAPD-MST algorithm, we will have an $O(n \log n)$-time factor 2 approximation algorithm for the OAPD-ST problem.

Second, Theorem 6.7 gives critical features to support strong heuristic methods, e.g., MST-based methods. That is, a solution with similar properties or topology to an OAPD-MST could be a good solution for the OAPD-ST problem.

6.4 Experimental Results

We implemented our algorithm in C language on a PC-based machine with 3GHz Pentium processor and 2GB memory under Linux operation system. All testcases are randomly generated; obstacles are rectangles. Each kind of testcases has **100** samples; the reported results (wirelength, number of vias, and time) are average results.

Since there is no existing work targeting on the OAPD-ST problem, we compared our algorithm with another method, which is extended from the construction-by-correction approach. The construction-by-correction approach is one of major methods for the OARSMT problem. Generally, the construction-by-correction approach consists of following two major steps:

1. **Construction:** Construct a minimum spanning tree as an initial Steiner tree without considering obstacles.

2. **Correction:** Replace edges which intersect obstacles with edges around the obstacles.

However, the construction-by-correction approach can not be directly extended to the OAPD-ST problem. First, for the OARSMT or ML-OARSMT problem, the cost of the rectilinear shortest path without considering obstacles between two pin-vertices can be measured directly by their coordinates. But, for the OAPD-ST problem, the PD constraints and different routing resources among layers must be considered, which increases the difficulty. Second, replacing invalid edges is not easy due to the PD constraints. That is, the invalid edges can not be replaced with edges around the obstacles since each layer only allows either horizontal or vertical edges. In order to make a fair comparison, we extended the construction-by-correction approach to match the OAPD-ST problem by more complicated modifications, and denote this approach as ***CC***.

Table 6.1 lists the total cost and running time of these algorithm under the conditions where routing resource is the same ($UC_i = 1$ for $1 \leq i \leq N_l$, $N_l = 6$, and $C_v = 3$). Considering the total cost, the average improvement over CC is 15.65 %. Furthermore, our method also has comparable speed with *CC* method. Our algorithm achieves such

Table 6.1: The Comparison under the same routing resource where $UC_i = 1$ for $1 \leq i \leq N_l$, $C_v = 3$, and N_l=6.

m	k	Total Cost			Time (sec.)	
		CC	our	imp(%)	CC	our
20	4	4540672	3905891	13.98	0.005	0.005
40	8	6380169	5489595	13.96	0.023	0.025
60	12	7982803	6741823	15.55	0.059	0.063
80	16	9281025	7768205	16.30	0.106	0.122
100	20	10157322	8617818	15.16	0.164	0.203
200	40	14259141	12083500	15.26	0.682	1.009
400	80	19939542	16927531	15.11	2.869	4.978
600	120	24595604	20683857	15.90	7.117	12.772
800	160	29017940	23726589	18.23	13.236	18.253
1000	200	31723317	26301914	17.09	20.923	30.977
Average				15.65	-	

improvements because it constructs an OAPD-MST which can approximate an OAPD-SMT by our proof. The difference in wirelength also results from that the construction step of *CC* method does not consider obstacles.

We also perform experiments under different routing resources, which is an important part of our new proposed routing model, OAPD-ST model. Table 6.2 shows the wirelenth, number of vias, and total cost. Table 6.3 compares the CPU time of these algorithms. To make a fair comparison, we use the weighted model from experiments in [48] to model the difference in routing resource between layers. The weighted model is based on a multiple-factor mf such that $UC_i = mf^{N_l - i}$. In this paper, we set mf= 1.1 or 2 the same as [48].

When $mf = 1.1$, our algorithm achieves 33.33% average improvement in total cost compared with *CC* method. Then, when mf increases to 2, the average improvement significantly increases to 43.53%, which is very major. Obviously, our method is more suitable for the OAPD-ST problem than *CC* method, especially for difference routing resources. This is also another evidence to figure out that *CC* method lacks of the global view of obstacles. On the other hand, the running time of our method keeps stable when

Table 6.2: Experiments for the effects of different routing resources. C_v=3 and N_l=6.

m	k	multiple-factor= 1.1					multiple-factor= 2				
		CC		our		Imp.	CC		our		Imp.
		WL/#via	cost	WL/#via	cost	(%)	WL/#via	cost	WL/#via	cost	(%)
20	4	5890485/106	5890803	4254534/62	4254719	27.77	14864763/106	14865082	9221732/61	9221914	37.96
40	8	8628723/227	8629405	5995387/129	5995774	30.52	22486025/228	22486709	13300849/126	13301225	40.85
60	12	10902089/343	10903119	7405735/192	7406311	32.07	31773440/345	31774473	18021464/188	18022028	43.28
80	16	12828484/464	12829877	8530323/257	8531095	33.51	37771155/469	37772561	21145568/252	21146323	44.02
100	20	14203266/579	14205003	9490671/321	9491634	33.18	41632192/582	41633938	23596286/314	23597230	43.32
200	40	20243722/1163	20247211	13281974/645	13283910	34.39	58657761/1169	58661268	32894898/631	32896792	43.92
400	80	28396782/2334	28403784	18609646/1293	18613525	34.47	82727843/2347	82734886	46320929/1268	46324734	44.01
600	120	34814762/3521	34825326	22749753/1955	22755618	34.66	102323514/3538	102334128	57440217/1916	57445966	43.86
800	160	39409929/4648	39423871	25618166/2676	25626193	35.00	124008755/4622	124022621	67345733/2621	67353596	45.69
1000	200	47927969/6327	47946950	29826588/3395	29836773	37.77	150339544/6344	150358576	77566875/3383	77566875	48.41
Average						33.33	Average				43.53

Table 6.3: The comparison on CPU time in second for Table 6.2.

m	k	multi-factor= 1.1		multi-factor= 2	
		CC	our	CC	our
20	4	0.007	0.005	0.009	0.005
40	8	0.032	0.027	0.045	0.028
60	12	0.083	0.069	0.128	0.07
80	16	0.155	0.134	0.266	0.134
100	20	0.239	0.223	0.424	0.223
200	40	1.066	1.100	2.261	1.098
400	80	5.117	5.436	12.78	5.416
600	120	13.034	14.048	35.264	13.816
800	160	17.692	20.401	44.263	19.839
1000	200	31.991	33.993	75.823	32.24

the difference in routing resource between layers increases, but that of *CC* method significantly increases. We observe that the correction step of *CC* method takes more expensive running time to find feasible replacements for invalid edges when the difference in routing resource between layers increases. It also shows that our method can work more efficient, effective, and stable than *CC* method for the OAPD-ST problem.

Chapter 7

Time Complexity Bottleneck and Local Minimal Heuristic

For the OAPD-ST problem, this chapter attempts to analyze the time complexity bottleneck and to develop an efficient local minimal heuristic. We first prove that the time complexity of OAPD-MST construction is $\Omega(n^2)$. Considering the efficiency, the proof gives a strong motivation to develop more efficient algorithms instead of OAPD-MST construction. Therefore, we attempts develop a local minimal heuristic to approximate an OAPD-MST. Since MTST-based algorithms are usually feasible for the Steiner tree related problems, we analyze the essence of an MTST-based algorithm. Based on the analysis, we propose an *preferred direction visibility graph* (PDVG), and develop an $O(n \log^2 n)$ MTST-based algorithm. For MTST-based algorithms, PDVG has three advantages: (1) high efficiency, (2) high possibility of path overlapping (Section 7.3.2), and (3) high solution quality (Section 7.3.2). Experimental results show that the algorithm is more efficient than the OAPD-MST construction and has comparable solution quality.

7.1 Motivation

As mentioned in Section 1.3, there are a huge number of obstacles in modern IC design. According to ITRS [19], the hard IP count per chip will be more than one thousand in the future. Therefore, it is necessary to develop close to $O(n \log n)$ algorithm for the OAPD-ST construction. Besides, from practical viewpoints, the input size also changes the requirements of an algorithm. In detail, large test cases require the higher efficiency, while small test cases require the higher solution quality. Hence, it is also interesting if there exists an approach handling those variable requirements.

According to Chapter 6, since an OAPD-MST is an approximation solution to the OAPD-ST problem, it is desirable to develop faster OAPD-ST algorithms. From theoretical aspects, we should first analyze the time complexity of the OAPD-MST construction. If the worst-case lower bound is $\Omega(n^2)$, we should find another way to approximate an OAPD-MST. It is natural to study some local minimal guarantees, which probably help to obtain close to global minimal solutions. Since MTST-based algorithms perform well for the OARSMT problem, we may analyze the essence of an MTST algorithm.

7.2 Space Complexity of An OAPD-MST

In this section, we prove that the space complexity of an OAPD-MST is $\Omega(n^2)$. Liu et al. [25] proved *Cost(OAPD-MST)* ≤ 2**Cost(OA-PDSMT)*, which implies that an OAPD-MST is a good approximation solution for the OAPD-ST problem. However, since the space complexity of an OAPD-MST is $\Omega(n^2)$, the running time complexity of OAPD-MST construction is at least $\Omega(n^2)$. Considering efficiency, our proof provides a strong

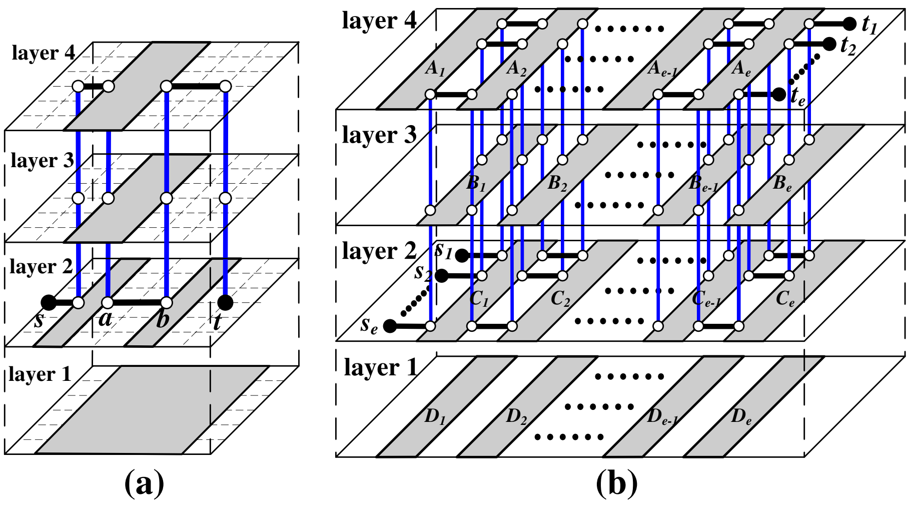

Figure 7.1: Example for failure of the alignment property in preferred direction model. (a) an OAPDSP between s and t must include (a, b). (b) an example for the lower bound of space complexity of an OAPD-MST.

motivation to develop more efficient algorithms for the OAPD-ST problem instead of OAPD-MST construction. Please first recall Definition 2.11.

Since the definition of an OAPD-MST is based on OAPDSPs, we discuss the space complexity of an OAPD-MST from the viewpoint of OAPDSPs.

Definition 7.1 *A path optimization problem is said to have the* ***alignment property*** *if there exists an optimal path consisting of line segments (edges) that are aligned with either one of the two endpoints or boundaries of the obstacles [20].*

In [20], Lee et al. studied rectilinear path problems in a two-layer preferred direction model and claimed that those problems have the alignment property. Thus, through their method, an OAPDSP in a two-layer preferred direction model can be computed in $O(k \log k)$ running time. However, in preferred direction model, the OAPDSP problem does not have the alignment property even when $N_l \geq 4$. For instance, Figure 7.1 (a) shows an OAPDSP between s and t when $N_l \geq 4$. It is clear that each OAPDSP between

s and t must include the line segment (a,b), but (a,b) is aligned neither one of the two endpoints nor edges of the obstacles. Hence, for finding an OAPDSP, we should consider many possibilities such as a and b, which significantly increases the complexity. We use this observation to make an example for proving the lower bound of space complexity of an OAPD-MST.

In Figure 7.1(b), there are $2e$ pin-vertices ($\{s_1,\cdots,s_e,t_1, \cdots,t_e\}$) and $4e$ rectangles ($\{A_1,\cdots, A_e,B_1,\cdots,B_e,C_1,\cdots,C_e,D_1,\cdots,D_e\}$) such that m=2e, k=16e and n=18e. For $1 \leq i \leq e$, there exists only one OAPDSP between s_i and t_i, and this path consists at least e line segments (edges). Furthermore, it is clear that those OAPDSPs between s_i and t_i where $1 \leq i \leq e$ are pairwise disjoint. Suppose that $|s_is_j|_{distance}$ ($|t_it_j|_{distance}$) for $1 \leq i < j \leq e$ is larger than $|s_lt_l|_{distance}$ for $1 \leq l \leq e$, which also implies that $|s_it_j|_{distance}$ for $1 \leq i,j \leq e$ and $i \neq j$ is larger than $|s_lt_l|_{distance}$ for $1 \leq l \leq e$. That is, $|s_it_i|_{distance}$ for $1 \leq i \leq e$ is smaller than the distance of any other pin-vertex pairs. Since an OAPDSP between two pin-vertices is regarded as an edge between them, we regard OAPDSPs, (s_i, t_j), (s_i, s_j) and (t_i, t_j) where $1 \leq i,j \leq e$, as edges between those pin-vertices with weight being their distances.

According to the cut property of minimum spanning trees [6], a light edge crossing a cut with minimum cost must be included in a minimum spanning tree. Therefore, in Figure 7.1(b), (s_i,t_i) where $1 \leq i \leq e$ must be included in an OAPD-MST connecting those pin-vertices. Since all line segments (edges) of (s_i,t_i) where $1 \leq i \leq e$ are pairwise disjoint and there are at least e^2 line segments, the space complexity of (s_i,t_i) where $1 \leq i \leq e$ is $\Omega(e^2)$, which implies that the space complexity of an OAPD-MST in Figure 7.1(b) is $\Omega(e^2)$. Since e is $\Omega(n)$, we have the following important theorem.

Theorem 7.1 *The space complexity of an OAPD-MST is* $\Omega(n^2)$.

Since the running time complexity of OAPD-MST construction is at least the space complexity of an OAPD-MST, we have the following corollary:

Corollary 7.1.1 *The running time complexity of OAPD-MST construction is* $\Omega(n^2)$.

Since an OAPD-MST consist of a set of OAPDSPs among pin-vertices, a routing graph preserving all OAPDSPs between pin-vertices contains at least one OAPD-MST, which implies the following corollary:

Corollary 7.1.2 *The space complexity of a routing graph preserving all OAPDSPs among pin-vertices is* $\Omega(n^2)$.

In the following of this paper, a lot of words are related to shortest paths and multi-layer models. To simplify descriptions, we make some assumptions as follow:

1. In 2 dimensions (2D), a shortest path means an obstacle-avoiding rectilinear shortest path.
2. In preferred direction model, a shortest path means an obstacle-avoiding preferred direction shortest path.
3. A multi-layer model means a preferred direction model even when $N_l = 2$ (**e.g. a two-layer model**).

7.3 Algorithm

Since OAPD-MST construction takes $\Omega(n^2)$ time, considering efficiency, we propose an $O(nlog^2n)$ MTST-based algorithm for the OAPD-ST problem instead of OAPD-MST construction. We develop a routing graph, called *preferred direction visibility graph* (PDVG), with both $O(nlogn)$ vertices and edges. Based on PDEG, we generate an MTST as our solution in $O(nlog^2n)$ running time. In the following subsections, we analyze properties of MTST-based algorithms in Section 7.3.1, and describe the basic ideas of PDVG in Section 7.3.2. Then, we show the details of PDVG construction in Section 7.3.3 and Section 7.3.4. Finally, we discuss our MTST-based algorithm in Section 7.3.5.

7.3.1 MTST-based Algorithms

In this section, we analyze properties of MTST-based algorithms and infer that it is possible to construct a smaller graph with high solution quality such that an MTST-based algorithm could perform more efficiently than OAPD-MST construction and generate comparable solutions. This is the backbone of our algorithm.

MTST-based algorithms are widely used in routing for Steiner tree problems [8, 22, 24, 25, 44]. An MTST-based algorithm is typically divided into two steps:

1. Construct a routing graph for a Steiner tree instance
2. Generate an MTST of the routing graph as a solution.

Routing graphs for Steiner tree problems often contain two kinds of vertices, terminal (pin-vertices) and nonterminals, such that the definition of an MTST is different from a general one. Similar to preferred direction model, a shortest path between two pin-vertices is regarded as an edge between them, and thus an MTST of a routing graph is defined as follows:

For simplifying descriptions, we denote the set of shortest paths which composes an MTST as ***SPs-MTST***, and the sum of costs of those shortest paths as ***Cost(SPs-MTST)***. In

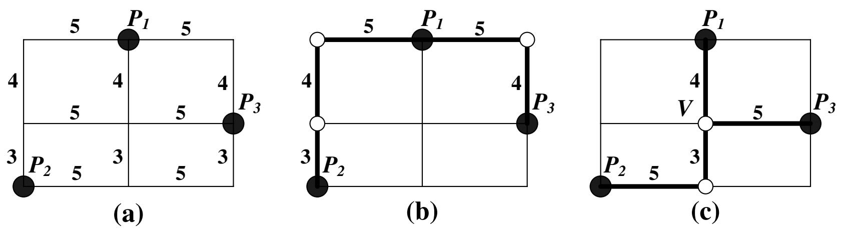

Figure 7.2: Example for the path selection problem. (a) Hanan Graph for three pin-vertices. (b) a minimum spanning tree with cost being 21. (c) another minimum spanning tree with lower cost.

addition, in a routing graph, the lower bound of cost of a shortest path between two pin-vertices is cost of an OAPDSP between them. *Since PDEG [25] preserves all OAPDSPs and an OAPD-MST, Cost(SPs-MTST) in PDEG is minimum.*

MTST-based algorithms often confront with *the path selection problem*. Take Hanan Graph [12] for instance. Fig 7.2(a) shows Hanan Graph for three pin-vertices; distances of (P_1, P_2), (P_1, P_3) and (P_2, P_3) are 12, 9 and 13 respectively. Hence, as shown in Figure 7.2(b), an MTST connecting the three pin-vertices consists of (P_1, P_2), (P_1, P_3) and has 21 total cost. That is, SPs-MTST are (P_1, P_2) and (P_1, P_3); *Cost(SPs-MTST)* is 21. However, since shortest paths in Hanan graph are not unique, there is another MTST with lower cost shown in Fig 7.2(c). In the MTST of Fig 7.2(c), *Cost(SPs-MTST)* is still 21 the same as the MTST of Figure 7.2(b), but *Cost(MTST)* decreases to 17. This is because that (P_1, P_2) and (P_1, P_3) in the MTST of Fig 7.2(c) overlap with each other, and cost of the overlap is 4. Thus, we have the following equation.

Equation 7.2 *For a routing graph,* ***Cost(MTST) = Cost(SPs-MTST) - Cost(overlap of SPs-MTST)****.*

Therefore, overlap of SPs-MTST could help *Cost(MTST)*. However, since the number of combinations of shortest paths is exponential, to maximize overlap of SPs-MTST is intractable, resulting that MTST construction usually does not consider overlap of SPs-MTST. Hence, most MTST-based algorithms do not take account of overlap of SPs-MTST. But, *we believe that overlap of SPs-MTST is crucial for MTST-based algorithm in some situations, e.g, a small instance.*

To conclude, *Cost(MTST)* depends on both *Cost(SPs-MTST)* and *Cost(overlap of SPs-MTST)*. Hence, minimum *Cost(SPs-MTST)* does not guarantee minimum *Cost(MTST)*, and higher *Cost(SPs-MTST)* possibly leads to lower *Cost(MTST)*. **Through the above analysis, we naturally define *Cost(SPs-MTST)* as the solution quality of a routing graph and *Cost(MTST)* as the solution quality of an MTST-based algorithm**. Therefore, the solution quality (*Cost(MTST)*) of an MTST-based algorithm on a routing graph with the best solution quality (*Cost(SPs-MTST)*) does not guarantee to be the best.

Now, we review the MTST-based algorithm in [25] and its routing graph, PDEG from the viewpoints of MTST-based algorithms. Although PDEG has minimum *Cost(SPs-MTST)*, the MTST-based algorithm in [25] has the following two drawbacks due to the large space complexity.

The first drawback is low efficiency. Since the space complexity of PDEG is $O(n^2)$, the running time complexity of the MTST-based algorithm is at least $O(n^2)$. Considering increasing obstacles and signal nets, the MTST-based algorithm in [25] is not sufficiently efficient.

The second drawback is low possibility of paths overlapping. Since the space complexity of PDEG is $O(n^2)$, the number of combinations of shortest paths significantly increases. Therefore, the possibility of paths overlapping is lower, and thus the overlap of

SPs-MTST would be relatively low.

As a result, considering efficiency, the MTST-based algorithm in [25] is not entirely suitable to the OAPD-ST construction. There could and should be another MTST-based algorithm on another routing graph for practical usage.

Through all discussion in this subsection, we infer that it is possible to construct a routing with *smaller size (high efficiency)*, *high solution quality (low Cost(SPs-MTST))* and *high possibility of paths overlapping (high Cost(overlap of SPs-MTST))*. Based on this graph, an MTST-based algorithm will perform more efficiently than OAPD-MST construction (recall Corollary 7.1.1) and generate comparable solutions.

Possible ideas are listed below. We can use a smaller graph to improve efficiency. Through delicate design of graph structure, the solution quality of a graph could be higher. The possibility of paths overlapping in a smaller graph will be higher since the number of combinations of shortest paths is lower. The simple topology of a graph would help the overlap of paths.

Finally, we make the following strong claim to be the backbone of our MTST-based algorithm.

Claim 7.3 *We can develop a routing graph with smaller size, high possibility of paths overlapping, and high solution quality. Based on those advantages, an MTST-based algorithm will preform more efficiently than OAPD-MST construction and generate comparable solutions.*

7.3.2 Basic Ideas

In this subsection, we will show the basic ideas of our routing graph, PDVG. For achieving Claim 7.3, PDVG should have high efficiency, high solution quality and high possibility of paths overlapping. Below, we classify our ideas into two parts, ***graph topology*** and ***solution quality***, to consider high possibility of paths overlapping and high solution quality respectively. Both the two parts are correlated with efficiency and related to each other.

7.3.2.1 Graph Topology

Considering efficiency and high possibility of paths overlapping, a routing graph should have the following properties:

1. **Small Size**: A small graph increase efficiency of MTST construction and has high possibility of paths overlapping.
2. **Simple Topology**: Aside from graph size, the topology of a routing graph also influences overlap of SPs-MTST. A simple topology could have higher overlap of SPs-MTST.

For those properties, we introduce an idea of graph construction called *Steiner points* which has been widely used for shortest problems among obstacles in computational geometry area. Clarkson et al. [5] introduced Steiner points to construct a routing graph called *visibility graph* in 2D within $O(n \log n)$ running time. Their visibility graph preserves all shortest paths and has both $O(n \log n)$ edges and vertices. Thus, a shortest path can be computed in $O(n \log^2 n)$ time.

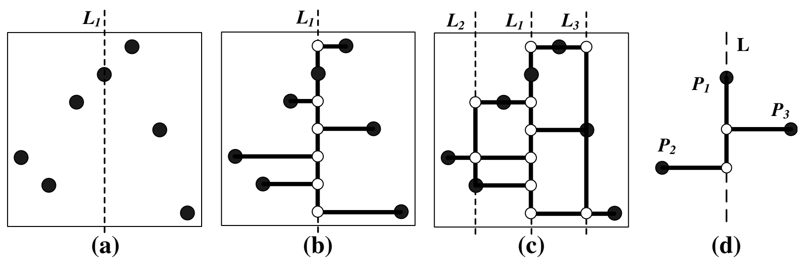

Figure 7.3: Example for Steiner points. (a) an instance. (b) projecting pin-vertices to L. (c) the resulting visibility graph for (a). (d) a visibility graph for three pin-vertices which has high overlap of SPs-MTST.

The idea of Steiner points is to use additional vertices to reduce graph size and is typically viewed as a 3-step procedure. We take Fig 7.3 for example.

1. Select a cutting line passing through the median vertex of all vertices according to their x-coordinates as shown in Figure 7.3(a).

2. Project vertices to the cutting line, connect vertices with their projections, and connect vertices in the cutting line as shown in Figure 7.3(b). Those projections are called *Steiner points*.

3. Recursively perform 1–2 on left and right parts. Figure 7.3(c) shows the resulting visibility graph for Figure 7.3(a).

Clearly, the depth of recursion is $O(\log n)$, and each vertex generates at most one Steiner point in each recursive level. Hence, space complexity of such a visibility graph is $O(n \log n)$ which is much smaller than $O(n^2)$.

Furthermore, the topology of a graph constructed by the idea of Steiner points could help overlap of SPs-MTST. For instance, Figure 7.3(d) shows a visibility graph for three pin-vertices. There is no path selection problem in this instance because the overlap of SPs-MTST is highest. Hence, **using the idea of Steiner points, the topology of a routing graph could probably help overlap of SPs-MTST in local, especially for small instances**.

As a result, we select the idea of Steiner points to be the basis of our graph construction. In Section 7.3.4, we develop algorithmic techniques to apply the idea of Steiner points to preferred direction model.

7.3.2.2 Solution Quality

For efficiency and solution quality, we correlate PDVG with OAPDSPs which are possible components of an OAPD-MST. Recall that *Cost(OAPD-MST)* $\leq$ 2**Cost(OAPD-SMT)* [25]; thus, an OAPD-MST is a good solution for the OAPD-ST construction. Since an OAPD-MST consists of OAPDSPs, a routing graph preserving all OAPDSPs will have best solution quality (the lowest *Cost(SPs-MTST)*). However, by Theorem 7.1 and Corollary 7.1.2, the space complexity of a graph preserving an OAPD-MST or all OAPD-STs is $\Omega(n^2)$, which is too expensive. Hence, we generate below concepts to collate PDVG with OAPDSPs:

- Preserve partial OAPDSPs instead of all OAPDSPs.
- Enhance the possibility of existence of OAPDSPs.
- Lower costs of shortest paths among pin-vertices.

For those concepts, we further develop two features for PDVG:

- ***Local Minimum Guarantee***: PDVG preserves partial OAPDSPs, which are possible components of an OAPD-MST.

- ***Delicate Vertices Inclusion*** PDVG construction includes possible candidates of vertices of OAPDSPs to improve the solution quality.

According to our ideas for graph topology and solution quality, *preferred direction visibility graph* (PDVG) construction is divided into two steps:

1. **Vertex Generation:** We add delicate vertices to build relations between layers; those vertices are possible candidates of vertices of OAPDSPs. Through the delicate vertices inclusion, the solution quality in PDVG could be higher. For efficiency considerations, number of those delicate vertices is limited to $O(n)$.
2. **Graph Construction:** We develop delicate algorithmic techniques to apply the idea of Steiner points to preferred direction model. Using those algorithmic techniques, PDVG is constructed in $O(n \log^2 n)$ running time, and has both $O(n \log n)$ vertices and edges. In addition, PDVG preserves OAPD-STs in each two adjacent layers considering wirelength, which is the local minimal guarantee of PDVG.

7.3.3 Vertex Generation

Below, we first discuss those delicate vertices which are possible candidates of vertices of OAPDSPs.

Definition 7.2 *For a vertex v, a vertex u in an adjacent layer to v is called* ***sibling*** *of v if u has the same coordinates with v and locates inside no obstacle. Since a layer has at most two adjacent layers, a vertex has at most two siblings.*

If not considering obstacles, a shortest path from a vertex in layer i to layer $i+1$ is an edge connecting this vertex with its sibling in layer $i+1$. As a result, we must consider

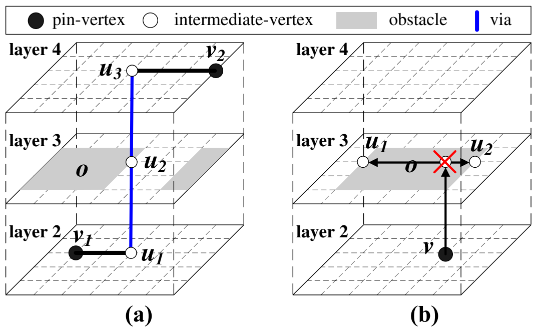

Figure 7.4: Examples for crossings. (a) a shortest path between v_1 and v_2 through a crossing u_2. (b) two crossings between v and o.

siblings of pin-vertices and obstacle corners as well as their siblings. Aside from siblings, due to the presence of obstacles, we also consider other two kinds of vertices.

We first define a relation between a vertex and an obstacle.

Definition 7.3 *For a vertex v, if a rectilinear projecting line from v to another layer is blocked by an obstacle o, o is a* ***blocked obstacle*** *of v.* As shown in Figure 7.4(a), o is a blocked obstacle of v_1. Clearly, a vertex has at most two blocked obstacles.

In Figure 7.4(a), the shortest path between v_1 and v_2 will go through u_2. We define u_2 as a crossing between v_1 and o.

Definition 7.4 *For a vertex v locates in a vertical (horizontal) layer, if o is a blocked obstacle of v,* ***crossings*** *between v and o are two vertices on the boundaries of o with their coordinates being $(x_v, min\_y(o))$ and $(x_v, max\_y(o))$ $((min\_x(o), y_v)$ and $(max\_x(o), y_v))$.* In Figure 7.4(b), the two crossings between v and o are u_1 and u_2.

Since a vertex has at most two blocked obstacles (Definition 7.3), a vertex has at most four crossings. Besides, if a vertex has crossings in layer i, this vertex has no sibling in layer i, and vice versus.

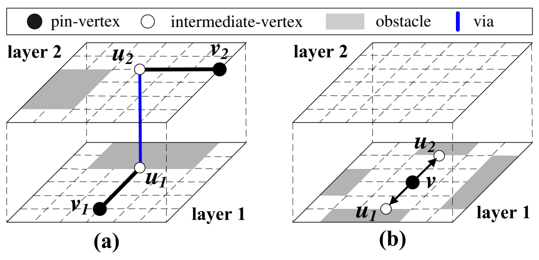

Figure 7.5: Examples for neighbors. (a) a shortest path between v_1 and v_2 through a neighbor u_1. (b) two neighbors of v.

In Figure 7.5(a), the shortest path between v_1 and v_2 will go through u_1. We define u_1 as a neighbor of v_1.

Definition 7.5 *For a vertex* $\mathbf{v}$ *in a vertical (horizontal) layer, a* ***neighbor*** *is a projection from* $\mathbf{v}$ *to one of adjacent obstacles in its up and down (left and right) directions.* In Figure 7.5(b), the neighbors of v is u_1 and u_2.

The algorithm for vertex generation is summarized in Figure 7.6. Lines 2–7 repeatedly generate siblings for pin-vertices and obstacle-corners by bottom-up and top-down procedures. This siblings generation is the same as recursively projecting each vertex to adjacent layers until blocked by obstacles. Since the number of pin-vertices and obstacle corners is n, the size of V at Line 9 is $O(n)$. Lines 9–11 generate crossings and neighbors to U for each v in V. Since each vertex has at most 2 neighbors and 4 crossings, $|U|$ is $O(n)$, implying that $|V|$ at Line 12 is $O(n)$. Lines 13–18 generate vertices for V similar to Lines 2–7. Thus, we directly have the following two lemmas.

Lemma 7.4 *The size of vertices generated by the Vertex-Generation algorithm is* $O(n)$.

Lemma 7.5 *The running time complexity of the Vertex-Generation algorithm is* $O(n \log n)$.

```
Algorithm: Vertex-Generation($P$, $O$, $N_l$)
Input: $P$ /* the set of pin-vertices */
       $O$ /* the set of obstacles */
       $N_l$ /* the number of routing layers */
Output: $V$ /* the set of generated vertices*/
1  $V \leftarrow P \bigcup \{corners\ of\ O\}$
2  for $i = 1\ to\ N_l - 1$
3      for each vertex $v \in V$ in layer $i$
4          add the sibling of $v$ in layer $i + 1$ to $V$
5  for $i = N_l\ to\ 2$
6      for each vertex $v \in V$ in layer $i$
7          add the sibling of $v$ in layer $i - 1$ to $V$
8  $U \leftarrow \emptyset$
9  for each vertex $v \in V$
10     add crossings of $v$ to $U$
11     add neighbors of $v$ to $U$
12 $V \leftarrow V \bigcup U$
13 for $i = 1\ to\ N_l - 1$
14     for each vertex $v \in V$ in layer $i$
15         add the sibling of $v$ in layer $i + 1$ to $V$
16 for $i = N_l\ to\ 2$
17     for each vertex $v \in V$ in layer $i$
18         add the sibling of $v$ in layer $i - 1$ to $V$
19 return $V$
```

Figure 7.6: The Vertex-Generation algorithm

7.3.4 Graph Construction

Here, we use the idea of Steiner points to complete PDVG construction on vertices generated by the Vertex-Generation algorithm.

The idea of Steiner points has been used for shortest path problems in 2D [5] and a two-layer preferred direction model [20]. As mentioned in Section 7.2, the shortest path problem in a two-layer preferred direction model has alignment property. Hence, a two-layer preferred is viewed as a special 2D instance [20], and the idea of Steiner points is applied similar to [5]. However, Figure 7.1(a) shows that the shortest problem in preferred direction model has no alignment property, implying that the methods in [5,20] cannot be directly applied. As a result, we develop delicate algorithmic techniques to apply the idea of Steiner points to preferred direction model.

For applying the idea of Steiner points to preferred direction model, we have three perspectives:

1. Since a preferred direction instance cannot be viewed as a 2D instance, a cutting line should be replaced with a cutting plane.

2. Due to preferred direction constraints, operations for vertices in vertical and horizontal layers should be different.

3. Since there are multiple layers, operations on a cutting plane should make relations between layers.

Based on the three perspectives, we develop PDVG construction.

Definition 7.6 *For two vertices v_1 and v_2, v_1 and v_2 are* ***adjacent*** *if there exists a rectilinear edge e connecting v_1 and v_2 such that e intersects no obstacle, violates no PD*

```
Algorithm: PDVG-Construction($P$, $O$, $N_l$)
Input: $P$ /* the set of pin-vertices */
       $O$ /* the set of obstacles */
       $N_l$ /* the number of routing layers*/
Output: $G(V, E)$ /* PDVG */
       $V$ /* the set of vertices */
       $E$ /* the set of edges*/
1  $E \leftarrow \emptyset$
2  $V$=Vertex-Generation($P$,$O$,$N_l$)
3  $U \leftarrow \emptyset$
4  for each vertex $v \in V$ in vertical layers
5     add siblings of $v$ to $U$
6     add crossings of $v$ in its two adjacent layers to $U$
7  $V \leftarrow V \bigcup U$
8  $S \leftarrow$ Steiner-Point-Generation($V$, $E$, $O$, $\emptyset$, $N_l$)
9  $V \leftarrow V \bigcup S$
10 for each two adjacent vertices $v_1$ and $v_2 \in V$
     in horizontal layers
11    $E \leftarrow E \bigcup \{(v_1, v_2)\}$
12 return $G(V, E)$
```

Figure 7.7: The PDVG-Construction algorithm.

constraints and crosses no other vertices.

The procedure of PDVG construction is summarized in Figure 7.7, and the main part is Steiner point generation. Figure 7.8 shows the Steiner-Point-Generation algorithm, and Figure 7.9 gives an example for one recursive routine. For clearly introducing the algorithm in Figure 7.8, the instance in Figure 7.9(a) only includes pin-vertices and obstacle corners. (i.e, Lines 2–7 in Figure 7.7 is not performed.)

We set cutting planes to be vertical, and PDVG construction is divided in four steps:

1. The Vertex-Generation algorithm is performed to build a set V (lines 1–2 in Figure 7.7), and the details have been discussed in Section 7.3.3.

2. Since cutting planes are vertical, only vertices in horizontal layers will be projected. Hence, considering vertices in vertical layers, we generate siblings and crossings of those vertices to adjacent layers (lines 4–7 in Figure 7.7). As shown in Figure 7.9

```
Algorithm: Steiner-Point-Generation($V$, $E$, $O$, $S_B$, $N_l$)
Input: $V$ /* the set of vertices */
       $E$ /* the set of edges */
       $O$ /* the set of obstacles */
       $S_B$ /* the boundary Steiner points */
Output: $S$ /* the set of Steiner points */
1  $x_{median} \leftarrow$ the median of x-coordinates in $V$
2  $Plane \leftarrow$ a plane perpendicular to routing layers
      with x-coordinate being $x_{median}$
3  $S \leftarrow \emptyset$
4  for each vertex $v \in V$
5     if a projecting line from $v$ to $Plane$ intersects no obstacle
6        $s \leftarrow$ the projection from $v$ to $Plane$
7        $S = S \bigcup \{s\}$
8  $U \leftarrow$ vertices in $V \bigcup S \bigcup S_B$ with x-coordinate $= X_{median}$
9  for each vertex $u \in U$
10    projecting $u$ to other layers to create siblings
        until projecting lines are blocked by obstacles
11    add those siblings to $U$
12    for each blocked obstacle $o$ of $u$
13       add crossings between $u$ and $o$ to $S_B$
14 for each two adjacent vertices $u_1$ and $u_2 \in U$
15    $E \leftarrow E \bigcup \{(u_1, u_2)\}$
16 $V_l \leftarrow$ vertices in $V$ with x-coordinate $< X_{median}$
17 $S_{B_l} \leftarrow$ vertices in $S_B$ with x-coordinate $< X_{median}$
18 if $|V_l| + |S_{B_l}| \geq 0$
19    $S = S \bigcup$Steiner-Point-Generation($V_l$, $E$, $O$, $S_{B_l}$, $N_l$)
20 $V_r \leftarrow$ vertices in $V$ with x-coordinate $> X_{median}$
21 $S_{B_r} \leftarrow$ vertices in $S_B$ with x-coordinate $> X_{median}$
22 if $|V_r| + |S_{B_r}| \geq 0$
23    $S = S \bigcup$Steiner-Point-Generation($V_r$, $E$, $O$, $S_{B_r}$, $N_l$)
24 return $S \bigcup S_B \bigcup U$
```

Figure 7.8: The Steiner-Point-Generation algorithm.

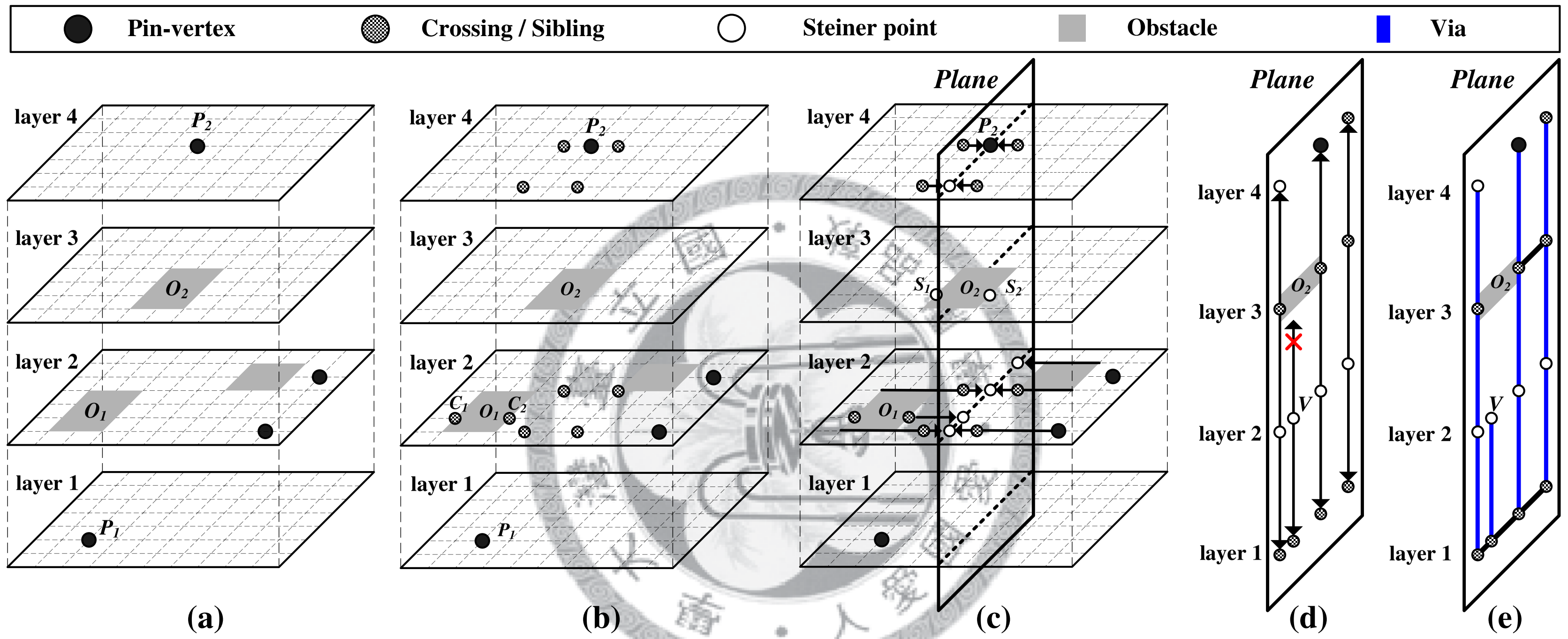

Figure 7.9: Example for Steiner point generation in preferred direction model. (a) an instance. (b) generating siblings and crossings for vertices in vertical layers. (c) projecting vertices in horizontal layers to a cutting plane. (d) projecting vertices in the cutting plane to other layers. (e) connecting vertices in the cutting plane.

(b), two crossings between P_1 and O_1 (i.e. C_1 and C_2) are generated, and eight siblings of four corners of O_2 are also generated.

3. As shown in Figure 7.8, the Steiner-Point-Generation algorithm is performed in the following steps:

   (a) Lines 1–2 select a vertical cutting plane passing through the median vertex of all vertices according to their x-coordinates. As shown in Figure 7.9(c), a cutting plane passes through P_2.

   (b) Lines 4–7 project vertices to the cutting plane to generate Steiner points also shown in Figure 7.9(c).

   (c) Lines 9–15 make connections on the cutting plane. Lines 10–11 project vertex to other layers to generate siblings as shown in Figure 7.9(d). If a vertex on the cutting plane is blocked by an obstacle, lines 12–13 generate crossings between them and call those crossings ***boundary Steiner points***. For instances, as shown in Figure 7.9(d), a vertex in layer 2, v, is blocked by O_2; thus, S_1 and S_2, crossings between v and O_2, are generated in layer 3 shown in Figure 7.9(c). Lines 14-15 connect adjacent vertices on the cutting plane as shown in Figure 7.9(e).

   (d) Lines 16–23 **recursively** perform steps a)–c) on the two parts divided by the cutting plane.

4. Lines 10–11 connect each adjacent vertices in horizontal layers.

Now, we analyze the space complexity of PDVG. Clearly, in PDVG, the number of edges is linear to that of vertices; thus, we only discuss the number of vertices. By Lemma

7.4, lines 4–6 in Figure 7.7 generate siblings and crossings for $O(n)$ vertices, implying that $|U|$ is $O(n)$. Hence, the Steiner-Point-Generation algorithm is perform on $O(n)$ vertices. Since there are $O(n)$ vertices, depth of the recursion is $O(\log n)$. In each recursive level, each vertex is projected to the cutting plane once such that the projection generates $O(n)$ Steiner points (lines 4–7 in Figure 7.8). Hence, in each recursive level, there are $O(n)$ vertices on those cutting planes. Since N_l is a small constant, there are both $O(n)$ siblings and boundary Steiner points generated (lines 9–13 in Figure 7.8) in each recursive level. Hence, in each recursive level, there are $O(n)$ vertices to be generated. To conclude, since there are $O(\log n)$ recursive levels, the number of generated vertices is $O(n \log n)$. Therefore, we have the following lemma about the space complexity of PDVG.

Lemma 7.6 *PDVG has both $O(n \log n)$ vertices and edges.*

Lemma 7.7 *The running time of PDVG construction is $O(n \log^2 n)$.*

Proof: By Lemma 7.5, the running time of the Vertex-Generation algorithm is $O(n \log n)$. Then, there are $O(\log n)$ recursive levels in the Steiner-Point-Generation algorithm. In each recursive level, we can use a sweep-line algorithm to project vertices to cutting planes. Similarly, siblings and boundary Steiner points could also be generated by sweep-line algorithms. The running time of a sweep algorithm is typically limited to sorting and the number of events (here, generating a vertex is an event). Since there are $O(n)$ vertices to be operated by sweep-line algorithms and $O(n)$ generated vertices, the running time complexity in each recursive level is $O(n \log n)$. Therefore, due to $O(logn)$ recursive levels, the running time of the Steiner-Point-Generation algorithm is $O(n \log^2 n)$. Finally, lines 10–11 in Figure 7.7 can also be performed by sweep-line algorithms. Since

$|V| = O(n \log n)$ in lines 10-11 (by Lemma 7.6, lines 10–11 take $O(n \log^2 n)$ running time. To conclude, the running time of PDVG construction is $O(n \log^2 n)$. ∎

Theorem 7.8 *PDVG preserves OAPDSPs between two adjacent layers considering wirelength.*

Proof: Lee et al. [20] proposed a routing graph in a two-layer preferred direction model, and their graph preserves OAPDSPs considering wirelength. In any two adjacent layers of PDVG, the subgraph contains the properties of the routing graph in [20]. Therefore, PDVG preserves OAPDSPs between two adjacent layers considering wirelength. ∎

Theorem 7.8 strongly supports the local minimal guarantee of PDVG. By the local minimal guarantee and delicate vertices inclusion, the solution quality (*Cost(SPs-MTST)*) of PDVG is very close to that of PDEG (*recall that Cost(SPs-MTST) in PDEG is minimum*). Experimental results will justify our claims.

7.3.5 MTST Construction and Discussion

After PDVG construction, we perform MTST construction on PDVG to generate an MTST as an OAPD-ST to be our solution. Mehlhorn [31] proposed an MTST construction method to generate an approximation solution for the Steiner tree prolbem. The running time complexity of his MTST construction method is $O(E + V \log V)$. We apply his method on PDVG and obtain the following time complexity from Lemma 7.6.

Lemma 7.9 *MTST construction on PDVG takes $O(n \log^2 n)$ time.*

By Lemma 7.7 and Lemma 7.9, we conclude the running time complexity of our OAPD-ST construction.

Theorem 7.10 *The running time complexity of our OAPD-ST construction is* $O(n^2 \log n)$

.

To conclude, based on advantages of PDVG, our OAPD-ST construction performs more efficiently than OAPD-MST construction and generates an OAPD-ST with comparable cost. First, the space complexity of PDVG is much smaller than PDEG [25]; thus MTST construction on PDVG performs more efficiently than the OAPD-MST construction [25]. Second, the local minimal guarantee and delicate vertices inclusion of PDVG make the solution quality of PDVG is very close to that of PDEG. In addition, the small size and simple topology of PDVG help the overlap of SPs-MTST and thus lower cost of the OAPD-ST. All advantages of PDVG originate from novel ideas in Section 7.3.2, and those novel ideas are inspired from the penetrating analysis of MTST-based algorithms in Section 7.3.1.

7.4 Experimental Results

We implemented our algorithms in the C/C++ language on a 3.0 Ghz Intel Pentium 4 PC with 2 GB memory under Linux 2.6 operating system. We obtained the program and the testcase generator from [25] to perform experiments. All testcases are randomly generated, and obstacles are rectangles. Each kind of testcases has **100** samples; the reported results are average results. As same as [25], we use a multi-factor mf to control different routing resources, i.e, $\boldsymbol{UC_i = mf^{N_l - i}}$. Experimental results in [25] show that their method completely outperforms an extension of traditional methods when the routing cost doubles between layers. Hence, we set mf to be 2 in our experiments; the related parameters (m, k, C_v and N_l) are also as same as [25]. Note that all experiments were performed in the same machine.

Table 7.1 shows comparison between our algorithm and [25] on wirelength (WL), number of vias (# of vias), total cost, time and *Cost(overlap of SPs-MTST)*. Note that [25] is an OAPD-MST construction algorithm. Since our algorithm and [25] are both MTST-based algorithms, the total cost is *Cost(MTST)*. By Equation 7.2, ***Cost(MTST)= Cost(SPs-MTST)-Cost(overlap of SPs-MTST)***. As shown in Table 7.1, our algorithm performs more efficiently than [25] and generates comparable solutions, i.e, our algorithm improves 1.46 % total cost on average over [25]. In details, when testcases are small, the solution quality is relatively important. At that time, our algorithm outperforms [25] in total cost. For instance, when testcases consist of 20 terminals and 4 obstacles, our algorithm improves 6.01 % total cost over [25], which is crucial. This is because that when testcases are small, possibility of paths overlapping in PDVG has become more higher due to the simple topology and small graph size of PDVG. Therefore, *Cost(overlap of SPs-MTST)* will be much higher such that *Cost(MTST)* would be lower. This also supports our belief that *Cost(overlap of SPs-MTST)* is crucial for a small instance. Besides, when testcases are large, the efficiency has become more important. At that time, our algorithm performs much faster than [25]. For instances, when testcases consists of 1000 terminals and 200 obstacles, our algorithm takes 1.338 seconds, but [25] takes 41.98 seconds. The above results indicate that our algorithm meets performance requirements for both small and large instances.

Table 7.2 shows comparison between PDVG and PDEG [25] on *Cost(SPs-MTST)*, number of vertices and number of edges. Since PDEG preserves OAPDSPs [25], *Cost(SPs-MTST)* is minimum. Table 7.1 shows that the space complexity of PDVG is much smaller

Table 7.1: Comparison in total cost and time where C_v=3, N_l=6, and $mf = 2$. "Total cost" is *Cost(MTST)* for an MTST-based algorithm.

m	k	OAPDMST Construction [1]			Ours			Imp. (%)	*Cost(overlap of SPs-MST)*	
		WL/#via	Total Cost (A)	Time (sec.)	WL/#via	Total Cost (B)	Time (sec.)	$\frac{A-B}{A}$	[1]	Ours
20	4	9197713 / 62	9197899	0.005	8645148 / 72	8645363	0.002	6.01	437615	1023715
40	8	13336491 / 128	13336875	0.023	12844161 / 151	12844613	0.006	3.69	631417	1158055
60	12	17913815 / 193	17914394	0.060	17453019 / 228	17453702	0.013	2.57	773576	1270025
80	16	21199418 / 258	21200192	0.114	20977214 / 307	20978134	0.020	1.05	902841	1179736
100	20	23401542 / 324	23402514	0.189	23008131 / 383	23009279	0.030	1.68	956147	1400461
200	40	32665350 / 647	32667291	0.951	32622273 / 778	32624607	0.096	0.13	1344600	1463057
400	80	46362334 / 1301	46366237	4.786	46396853 / 1574	46401576	0.305	-0.08	1884580	1950340
600	120	57351616 / 1962	57357502	12.280	57420367 / 2388	57427530	0.590	-0.12	2255025	2294329
800	160	64966432 / 2607	64974253	24.346	65068929 / 3182	65078476	0.938	-0.16	2603465	2644487
1000	200	74841986 / 3272	74851802	41.980	74947890 / 3999	74959888	1.338	-0.14	2843064	2901143
Average Improvement								1.46	-	

Table 7.2: Comparison between PDVG and PDEG [25] where C_v=3, N_l=6, and $mf = 2$.

m	k	*Cost(SPs-MST)*		Increase (%)	Number of Vertex		Number of Edge	
		PDEG (A)	PDVG (B)	$\frac{B-A}{A}$	PDEG	PDVG	PDEG	PDVG
20	4	9635514	9669078	0.35	4316	1306	7401	2131
40	8	13968292	14002668	0.25	16926	3424	29146	5639
60	12	18687970	18723727	0.19	37502	5985	64575	9901
80	16	22103033	22157870	0.25	66302	8878	114020	14708
100	20	24358661	24409740	0.21	103116	12149	177542	20217
200	40	34011891	34087664	0.22	411224	31218	708379	52188
400	80	48250817	48351916	0.21	1633001	79484	2811558	133276
600	120	59612527	59721859	0.18	3666070	135229	6305959	226768
800	160	67577718	67722963	0.21	6510892	197141	11204196	330703
1000	200	77694866	77861031	0.21	10162367	265075	17478127	445027
Average Increase				0.23	-			

than PDEG, but *Cost(SPs-MTST)* of PDVG is very close to that in PDEG (only increases 0.23 % on average). For instance, when testcases consists of 1000 terminals and 200 obstacles, PDVG has 265075 vertices and 445027 edges, but PDEG has 10162367 vertices and 17478127 edges. Those results indicate that PDVG is very competitive. Those results also support high solution quality of PDVG, i.e., by the local minimal guarantee and delicate vertices inclusion, the solution quality (*Cost(SPs-MTST)*) of PDVG is very close to that of PDEG.

To conclude, Table 7.1 and Table 7.2 justify Claim 7.3: we can develop a routing graph (PDVG) with smaller size, high possibility of paths overlapping, and high solution quality. Based on those advantages, an MTST-based algorithm will preform more efficiently than OAPD-MST construction and generate comparable solutions.

Chapter 8

Concluding Remarks

As the technology advances into nanometer era, routing has been much more important since it has the most direct impact on the final design performance. Under these circumstances, routing-related operations need to function across all the physical design flow to handle the timing delay, timing skew, and crosstalk of all the signal nets. In particular, the routing tree construction is an essential step of routing and plays a crucial role for the routing results. However, in modern IC design, there are much more routing constraints such as multiple routing layers, obstacles, preferred directions, different routing resources, and via costs. To cope with these constraints, we comprehensively study the OARSMT problem and the OAPD-ST problem in this dissertation.

8.1 The OARSMT Problem

The study of the OARSMT problem attempts to give more accurate interconnect estimation at the floorplanning and placement stages. However, there exists a big trade-off between all the-state-of-art works in the efficiency (run time) and the effectiveness (wirelength). Therefore, to resolve the bottleneck, we develop a series of strategies including appropriate mechanics, frameworks, algorithm skills. Overall, in this dissertation, we develop three excellent algorithms, and successfully achieve the best practical performance in both wirelength and run time. Behind those approaches, we also build many theoretical foundations, which will contribute to the future researches for the OARSMT problem as well as its important generations such as the ML-OARSMT problem [24] and the OAPD-ST problem [25]. Below, we give concluding remarks on those three approaches.

In Chapter 3, we propose an advanced spanning graph, called OARG, and develop a 3-step MTST-based algorithm as well as an efficient local refinement scheme. Compared to those spanning graphs in [22, 28, 36], the OARG either contains better solutions or has higher efficiency. Experimental results show that our MTST-based algorithms outperform other spanning-graph based algorithm [22,28,36]. Although the OARG cannot guarantees the existence of an OARMST, the solution quality is very close to that of Lin's OASG [22], which guarantees the existence of an OARMST. In other words, the local OARMSTs of the OARG can approximate an OARMST.

In Chapter 4, we propose a path-based framework to guarantee a specific theoretical optimality in $O(n \log n)$ time, which original required $O(n^3)$ time in [22]. Compared with previous works, the framework directly generates critical paths as essential solution components instead of generating invalid initial solutions or constructing connected routing graphs. Overall, the framework has a global view of obstacles, guarantees the existence of desired solutions, and keeps only $O(n)$ solution components in $O(n)$ space. Therefore, the framework provides an efficient as well as effective way to develop more desirable OARSMT algorithms. Experimental results have shown that the path-based algorithm achieves the best speed performance, while the average wirelength of the resulting solutions is only 1.1% longer than that of the best existing solutions.

In Chapter 5, we analyze the essence of the OARSMT problem, and propose the idea of Steiner point selection to achieve the best practical performance in both wirelength and run time. The idea consists of two major components, the Steiner-point based framework and and the concept of Steiner point locations. Unlike many previous works, the Steiner-based framework is more focused on the usage of Steiner points instead of the handling of obstacles, and seems closer to the essence of the OARSMT problem, which leads to better solutions. The concept of Steiner point locations reflects the nature of Steiner points from another viewpoint, and thus provides an effective as well as efficient way to generate desirable Steiner point candidates. Experimental results show that our algorithm achieves the best solution quality in $\Theta(n \log n)$ empirical time, which was originally generated by applying a maze-routing based method on an $\Omega(n^2)$-space extended Hanan grid [21]. More importantly, both the Steiner-point based framework and the new concept of Steiner point locations gives critical insight into the OARSMT problem, and probably contribute to the development of future algorithms for the OARSMT problem, and can be naturally extended to its important generations, such as the ML-OARSMT problem [24] and the OA-PDST problem [25].

8.2 The OAPD-ST Problem

The study for the OAPD-ST problem attempts to provide better tree topologies at the routing stage for the succeeding interconnect optimization. However, none of existing works catches all the mentioned constraints including multiple routing layers, obstacles, preferred directions, different routing resources, and via costs since each of those constraints is difficult to handle. Therefore, we formulate the OAPD-ST problem to catch all the mentioned constraints, and give a comprehensive study on it. As a first study on the OAPD-ST problem, for development of future algorithms, we build essential theoretical foundations such as the structure of the optimal solution, the approximation algorithm, the bottleneck of complexity, and the local minimal heuristic, all of which are critical in the study of Steiner tree problem [18].

In Chapter 6 we analyze the structure of the optimal solution and propose an approximation algorithm. For the structure of the optimal solution, we propose *preferred direction evading graph* (PDEG), and prove that at least one optimal solution exists in PDEG. The theoretical optimality proof provides a way to analyze the solution quality, which significantly help the development of algorithms, especially for approximation ones. For the approximation algorithm, based on PDEG, we prove that the approximation factor of an *obstacle-avoiding preferred direction minimum spanning tree* (OAPD-MST) to the OAPD-ST problem is 2, and thus the OAPD-MST construction is a factor 2 approximation algorithm for the OAPD-ST problem. The approximation guarantee of an OAPD-MST gives important features to support the development of strong heuristics, especially for MST-like heuristics.

In Chapter 7, We first analyze the worst-case complexity of the OAPD-MST construction, and then develop a local minimal heuristic based on a local minimal guarantee and an MTST-based concept. We first prove that the space complexity of an OAPD-MST is $\Omega(n^2)$, which gives a strong motivation to develop more efficient algorithms for the OAPD-ST problem instead of the OAPD-MST construction. Then, we analyze the properties of MTST-based algorithm and make critical inference. Based on the inference and other computational geometry skills, we propose a routing graph, *preferred direction visibility graph* (PDVG) with a local minimal guarantee, and develop an $O(n \log^2 n)$-time MTST-based heuristic. Experimental results shows that our algorithm performs more efficiently than OAPD-MST construction and can generate comparable solutions. Experimental results also show the high competitiveness of PDVG and justify all our claims about PDVG and our algorithm.

8.3 Future Works

In this dissertation, for the OARSMT problem and the OAPD-ST problem, we progressively propose desirable strategies and build basic theoretical foundations. Based on those excellent strategies and essential theoretical foundations, we want to give future contributions to the routing tree construction and its practical applications in diverse aspects. We mainly consider the two important topics: (1) multiple-pin net, and (2) multi-criteria.

First, in modern IC design, there millions of signal nets, and the current methodology is mainly the global and detailed routing framework [13,34,49]. In general, global routing assigns routing regions and layers to each net, and detailed routing takes solutions from global routing to complete the whole routing process. Since the global-to-detailed routing flow attempts to completely route millions of signal nets, it may not focus on the performance of routing specific nets. However, there are more and more constraints in current routing process. Therefore, under the global-to-detailed routing flow, several critical nets may not meet the performance requirements, cause overhead in routing resources, or take more time for correction behind the flow. Under this circumstance, since the OAPD-ST problem considers many constraints across the global and detailed routing, we attempt to apply the OAPD-ST construction to deal with critical nets beyond the global-to-detailed routing. We also want to study the multiple-nt OAPD-ST problem, and build a new flow for critical signal nets or integrate the study into existing methodologies.

Second, aside from wirelength, there are other objectives for routing a signal net such as timing skew. There have already existed a number of works to cover the multi-criteria objective such as the integration of wirelength and timing skew. However, in the presence of obstacles, there still exist no significant work. Very recently, Synopsys in Taiwan raised a multi-criteria OPAD-ST problem including wirelength and timing skew objectives for no more than 10,000 nets [40]. Since the optimization on multi-criteria objectives gives a strong potential to achieve the best performance for signal nets, we attempt to apply our strategies and theoretical foundations on multi-criteria routing topics, and want to give further contributions for interconnect optimizations.

Bibliography

[1] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, 2^{nd} edition, Springer, 2000.

[2] M. Borah, R. M. Owens, and M. J. Irwin, "An edge-based heuristic for Steiner routing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 12, pp. 1563–1568, 1994.

[3] B. Chazelle, "An algorithm for segment dragging and its implementation," *Algorithmica*, vol. 3,pp. 205–221, 1988.

[4] C. Chu and Y.-C. Wong, "FLUTE: Fast lookup table based rectilinear Steiner minimal tree algorithm for VLSI design," journal = "IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems", vol. 27, no. 1, pp. 70–83, 2008.

[5] K. Clarkson, S. Kapoor, and P. M. Vaidya, "Rectilinear shortest paths through polygonal obstacles in $O(n \log^2 n)$ time," in Proceedings of the ACM Symposium on Computational Geometry, pp. 251–257, 1987.

[6] T. H. Cormen, C. E. Leiserson, and R. H. Rivest, *Introduction to Algorithms 2nd Edition*, MIT Press, Cambridge, Massachusetts, 2001.

[7] M. Dorigo, V. Maniezzo, and A. Colorni, “The Ant System: optimization by a colony of cooperating agents,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 26, no. 1, pp. 29–41, 1996.

[8] Z. Feng, Y. Hu, T. Jing, X. Hong, X. Hu, and G. Yan, “An $O(n \log n)$ algorithm for obstacle-avoiding routing tree construction in the lambda-geometry plane,” in *Proceedings of the ACM International Symposium on Physical Design (ISPD)*, pp. 48–55, 2006.

[9] J. L. Ganley and J. P. Cohoon, “Routing a multi-terminal critical net: Steiner tree construction inthe presence of obstacles,” in Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), pp. 113–116, 1994.

[10] M. Garey and D. Johnson, “The Steiner tree problem is NP-complete,” *SIAM Journal on Applied Mathematics*, vol. 32, no. 4, pp. 826–834, 1977.

[11] J. Griffith, G. Robins, J. S. Salowe, and T. Zhang, “Closing the gap: Near-optimal Steiner trees in polynomial time,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 11, pp. 1351–1365, 1994.

[12] M. Hanan, “On Steiner’s problem with rectilinear distance,” *SIAM Journal on Applied Mathematics*, vol. 14, pp. 255–265, 1966.

[13] C.-H. Hsu, H.-Y. Chen, and Y.-W. Chang, “Multi-layer global routing considering via and wire capacities,” in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 350–355, 2008.

[14] Y. Hu, Z. Feng, T. Jing, X. Hong, Y. Yang, G. Yu, X. Hu, and G. Yan, "FORst: a 3-step heuristic for obstacle-avoiding rectilinear Steiner minimal tree construction," *Journal of Information and Computational Science*, vol. 1., no 3. pp. 107–116, 2004.

[15] Y. Hu, T. Jing, X. Hong, Z. Feng, X. Hu, and G. Yan, "An-OARSMan: obstacle-avoiding routing tree construction with good length," in *Proceedings of ACM/IEEE Asia and South Pacific Design Automation Conference*, pp. 7–12, 2005.

[16] F. K. Hwang, "On Steiner minimal trees with rectilinear distance," *SIAM Journal on Applied Mathematics*, vol. 30, no. 1, pp. 104–114, 1976.

[17] F.-K. Hwang, "An $O(n \log n)$ algorithm for rectilinear minimal spanning trees," *Journal of ACM*, vol. 26, no. 2, pp. 177–182, 1979.

[18] F.-K. Hwang, D. S. Richards, and P. Winter, "The Steiner tree problem," in *Annals of Discrete Mathematics*, Elsevier, Amsterdam, The Netherlands, 1992.

[19] ITRS, *International Technology Roadmap for Semiconductors*, 2007, http://www.itrs.net.

[20] D.-T. Lee, C.-D. Yang, and C.-K. Wong, "Finding rectilinear paths among obstacles in a two-layer interconnection model," *International journal of computational geometry & application*, vol. 7, no. 6, pp. 581–598, 1997.

[21] L. Li and Evangeline F.-Y. Young, "Obstacle-avoiding rectilinear Steiner tree construction," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 523–528, 2008.

[22] C.-W. Lin, S.-Y. Chen, C.-F. Li, Y.-W. Chang, and C.-L. Yang, "Efficient obstacle-avoiding rectilinear Steiner tree construction," in *Proceedings of the ACM International Symposium on Physical Design (ISPD)*, pp. 127–134, 2007.

[23] C.-W. Lin, S.-Y. Chen, C.-F. Li, Y.-W. Chang, and C.-L. Yang, "Obstacle-avoiding rectilinear Steiner Tree construction based on spanning graphs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* vol. 27, no. 4, pp. 643–653, 2008.

[24] C.-W. Lin, S.-L. Huang, K.-C. Hsu, M.-X. Lee, and Y.-W. Chang, "Efficient multi-layer obstacle-avoiding rectilinear Steiner tree construction," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 380–385, 2007.

[25] C.-H. Liu, Y.-H. Chou, S.-Yi Yuan, and Sy-Yen Kuo, "Efficient Multilayer Routing Based on Obstacle-Avoiding Preferred Direction Steiner Tree," in *Proceedings of the ACM International Symposium on Physical Design (ISPD)*, pp. 118–125, 2008.

[26] C.-H. Liu, S.-Y. Yuan, S-Y. Kuo, and Y.-H. Chou, "An $O(n \log n)$ path-based obstacle-avoiding algorithm for rectilinear Steiner tree Construction," to appear *in proceedings of the IEEE/ACM Design Automation Conference (DAC)*, 2009.

[27] C-H Liu, S-Y Yuan, S-Y Kuo, and S-C Wang, "High-performance obstacle-avoiding rectilinear Steiner tree construction," *ACM Transactions on Design Automation of Electronic Systems*, vol. 14, no. 3, Article 45, 2009.

[28] J. Long, H. Zhou, and S. O. Memik, "An $O(n \log n)$ Edge-Based Algorithm for Obstacle-Avoiding Rectilinear Steiner Tree Construction," in *Proceedings of the ACM International Symposium on Physical Design (ISPD)*, pp. 126–133, 2008.

[29] J. Long, H. Zhou, and S. Memik, "EBOARST: an efficient edge-based obstacle-avoiding rectilinear Steiner tree construction algorithm," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 12, pp. 2169–2182, 2008.

[30] I. I. Mandoiu and V. V. Vazirani and J. L. Ganley, "A new heuristic for rectilinear Steiner trees," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 157–162, 1999.

[31] K. Mehlhorn, "A faster approximation algorithm for the Steiner problem in graphs," *Information Processing Letters*, vol. 27, no. 3, pp. 125–128, 1988.

[32] J. S. B. Mitchell, "An optimal algorithm for shortest rectilinear paths among obstacles," in *Proceedings of the Canadian Conference on Computational Geometry (CCCG)*, 1989.

[33] J. S. B. Mitchell, "L_1 shortest paths among polygonal obstacles in the plane," *Algorithmica*, vol. 8, pp. 55–88, 1992.

[34] G.-J. Nam, C. Sze, and M. Yildiz, "The ISPD global routing benchmark suite," in *Proceedings of the ACM International Symposium on Physical Design (ISPD)*, pp. 156–159, 2008.

[35] M. Pan and C. Chu, FastRoute: a step to integrate global routing into placement, in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 464–471, 2006.

[36] Z. Shen, C. Chu, and Y. Li, “Efficient rectilinear Steiner tree construction with rectilinear blockages,” in *Proceedings of the IEEE International Conference on Computer Design (ICCD)*, pp. 38–44, 2005.

[37] Y. Shi, T. Jing, L. He, and Z. Feng, “CDCTree: novel obstacle-avoiding routing tree construction based on current driven circuit model,” in *Proceedings of the ACM/IEEE Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 630–635, 2006.

[38] Y. Shi, P. Mesa, H. Yu, and L. He, “Circuit simulation based obstacle-aware Steiner routing,” in *Proceedings of the IEEE/ACM Design Automation Conference (DAC)*, pp. 385–388, 2006.

[39] S. M. Siat and H. Youssef, *VLSI Physical Design Automation: Theory and Practice*, McGraw-Hill, 1995.

[40] Synopsys Taiwan, “Obstacle-Avoiding Rectilinear Clock Routing with Preferred Directions,” *Annual IC/CAD Contest (Ministry of Education)*, 2009, Taiwan. [On-Line] Available: http://cad_contest.ee.ntu.edu.tw/cad09/index.ht.

[41] M. Takahashi and A. Matsuyama, “An approximate solution for the Steiner problem in graph,” *Math Japonica*, vol. 24, pp. 573–577, 1980.

[42] D. M. Warme, P. Winter, and M. Zacharisen, “Exact algorithms for plane Steiner tree problems: A computational study,” *Technical Report DIKU-TR-98/11*, Dept. of Computer Science, University of Copenhagen, 1998.

[43] D. M. Warme, P. Winter, and M. Zacharisen, “GeoSteiner – Software for Computing Steiner Trees,” [Online]. Available: http://www.diku.dk/geosteiner.

[44] P.-C. Wu, J.-R. Gao, and T.-W. Wang, “A fast and stable algorithm for obstacle-avoiding rectilinear Steiner minimal tree,” in *Proceedings of the ACM/IEEE Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 262–267, 2007.

[45] Y.-F. Wu, P. Widmayer, M. D. F. Schlag, and C.-K. Wong, “Rectilinear shortest paths and minimum spanning trees in the presence of rectilinear obstacles,” *IEEE Transactions on Computers*, vol. 36, no. 3, pp. 321–331, 1987.

[46] Y.-F. Wu, P. Widmayer, and C.-K. Wong, “A faster appromixation algorithm for the Steiner problem in graphs,” *Acta Informatica*, vol. 23, pp. 223–229, 1986.

[47] Y. Yang, Q. Zhu, T. Jing, X. Hong, and Y. Wang, “Rectilinear Steiner Minimal Tree Among Obstacles,” in *Proceedings of the IEEE International Application Specific Integrated Circuits Conference (ASICCON)*, pp. 348–351, 2003.

[48] M. C. Yildiz and P. H. Madden, “Preferred direction Steiner trees,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 11, pp. 1368–1372, 2002.

[49] Y. Zhang, Y. Xu, and C. Chu, “FastRoute3.0: A fast and high quality global router based on virtual capacity,” in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 344–349, 2008.

[50] H. Zhou, “Efficient Steiner tree construction based on spanning graphs,” in *Proceedings of the ACM International Symposium on Physical Design (ISPD)*, pp. 152–157, 2003.

[51] H. Zhou, N. Shenoy, and W.Nicholls, “Efficient spanning tree construction without Delaney triangulation,” *Information Processing Letters*, vol. 81, no. 5, pp. 271–276, 2002.

Vita

Chih-Hung Liu was born in Taipei, Taiwan on September 3, 1982, the son of Li-Mei Hsu and Shih-Jen Liu. After completing his work from Chien-Kuo Senior High School, Taipei, Taiwan in 2001, he entered National Taiwan University. In 2005, he received the B.S. degree in the department of Computer Science and Information Engineering. In the same year, he entered the master program of the Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, Taiwan. In 2006, he applied to enter the Ph.D. program of the same institute.