國立交通大學

電子工程學系 電子研究所碩士班

碩 士 論 文

考慮多層繞線與障礙物迴避之
直角史坦那樹建構法

Effective Multi-Layer Obstacle-Avoiding Rectilinear Steiner Tree Construction

研 究 生：林聖偉

指導教授：江蕙如 博士

中 華 民 國 九 十 六 年 十 二 月

考慮多層繞線與障礙物迴避之直角史坦那樹建構法
Effective Multi-Layer Obstacle-Avoiding Rectilinear Steiner Tree Construction

研究生：林聖偉　　Student: Shung-Wei Lin

指導教授：江蕙如 博士　　Advisor: Dr. Iris Hui-Ru Jiang

國 立 交 通 大 學

電子工程學系電子研究所碩士班

碩 士 論 文

A Thesis
Submitted to Department of Electronics Engineering & Institute of Electronics
College of Electrical and Computer Engineering
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of
Master
in
Electronics Engineering

December 2007

Hsinchu, Taiwan, Republic of China

中華民國 九十六年十二月

考慮多層繞線與障礙物迴避之直角史坦那樹建構法

學生：林聖偉　　指導教授：江蕙如 博士

國立交通大學

電子工程學系　電子研究所碩士班

摘　　要

已知散佈在各繞線層上的接點(pin)和障礙物(obstacle)的座標，我們的目的是找出一棵迴避障礙物之直角史坦那樹去連接所有的接點，而可以利用額外的點(即史坦那點)及 via，但是禁止穿越過障礙物，使得繞線總長與 via 的總合花費最少。符合這些條件的樹，即所謂考慮多層繞線與障礙物迴避之直角史坦那樹。

本篇論文定義了一個典型流程，根據此流程我們提出了以 construction-by-correction 為概念的演算法：第一步驟建立 Delaunay triangulation 圖，連接所有的接點；第二步驟對所建立的 Delaunay triangulation 圖進行障礙物加權的最小生成樹的建構；接著第三步驟將樹直角化且利用三維 U 型線段修正總線長。

本篇論文的特點在於：1.第一步驟不作完整圖，因為其時間複雜度為 O(n^2)。本篇論文利用 Delaunay triangulation 當作一開始的連接圖，其時間複雜度為 O($n$lg$n$)。此外，我們只對所有接點作圖，並加入額外的 edge 在 DT 中，使得結果更好。2.第三步驟提出了新的 U 型線段修正方法並擴展到層與層之間的立體 U 型線段修正。特別的是，我們是對所有可能的立體 U 型線段作完整的分類，此處可以得到最佳解。我們在多層繞線的實驗結果，當單顆 via 以 5 單位線長計算時，平均總線長比前人少了 1.99%。且在單層繞線的實驗結果平均也和目前最好的結果不相上下。

Effective Multi-Layer Obstacle-Avoiding Rectilinear Steiner Tree Construction

Student: Shung-Wei Lin Advisor: Dr. Iris Hui-Ru Jiang

Department of Electronics Engineering
Institute of Electronics
National Chiao Tung University

Abstract

Given a set of pins and a set of obstacles on multiple routing layers, a multi-layer obstacle-avoiding rectilinear Steiner minimal tree (ML-OARSMT) is a tree of minimal total wirelength that connects these pins, possibly through some extra points (called Steiner points) and/or vias, and bypasses all obstacles.

In this thesis, we define a typical flow and propose a construction-by-correction based algorithm. First of all, an initial connection graph is constructed over all pins based on Delaunay triangulation. Secondly, an obstacle-weighted minimum spanning tree is grown up over the initial connection graph. Finally the tree edges are rectilinearized and the total wirelength is further reduced by three-dimensional U-shape refinement.

Our algorithm has two features. One is that we perform Delaunay triangulation instead of using a complete graph in the first step. In addition, we only consider pins into Delaunay triangulation thus completing it in only O($n$lg$n$) time; we add some extra edges into Delaunay triangulation that possibly contribute to a better minimum spanning tree. The other is that we present optimal three-dimensional U-shape refinement to further reduce the total wirelength. Experimental results show that as one via costs five-unit wirelength, our results in ML-OARSMT outperforms previous work by 1.99%. On the other hand, our results in SL-OARSMT are as good as the best results in literature so far.

Acknowledgements

I would like to express my heartfelt gratitude to my advisor, Prof. Iris Hui-Ru Jiang, for her guidance and spur throughout my graduate course. I really learnt a lot of abilities about research attitudes and presentation from her careful guidance. Meanwhile, I deeply appreciate my lab members at NCTU and my classmates from NCU. For their kindly help, I could break through my research bottlenecks and share my interesting life. Finally, I display my warmest appreciation to my parents for their love and support.

Shung-Wei Lin

National Chiao Tung University

December 2007

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction

在這一章中，首先會介紹何謂考慮多層繞線與障礙物迴避之直角史坦那樹(Multi-Layer Obstacle-Avoiding Rectilinear Steiner Minimal Tree, ML-OARSMT)，接著介紹目前國內外針對多層繞線的 ML-OARSMT 及只考慮單層繞線(single layer)的 SL-OARSMT 問題的研究，最後是我們的方法新穎之處及其效能。

1.1 Obstacle-Avoiding Rectilinear Steiner Minimal Trees

已知散佈在各繞線層上的接點(pin)和障礙物(obstacle)的座標，我們的目的是找出一棵迴避障礙物之直角史坦那樹去連接所有的接點，而可以利用額外的點(即史坦那點)及 via，但是禁止穿越過障礙物，使得繞線總長與 via 的總合花費最少。符合這些條件的樹，即所謂 ML-OARSMT。

隨著製程的進步，繞線層數也越來越多，目前已到了九層的複雜度[1]。在現今系統晶片(SOC)設計中，常常存在了不少的障礙物，例如矽智財區塊(IP blocks)、電源供應網路(power networks)、與為了改善製程所加入的 feature pattern 或是已經被繞過的線段。因此，解決 ML-OARSMT 的問題是當務之急。

然而單單只有 RSMT 的問題，就已經被證明為 NP-complete [2]，如果再加入多繞線層與障礙物的因素，無疑地更增加了問題的複雜度。

1.2 Previous Works

目前大部分的研究都注重在 SL-OARSMT 的問題上，然而也逐漸開始有一些針對 ML-OARSMT 問題的研究，基本上都是延伸 SL-OARSMT 的方法，而應用在 ML-OARSMT 上。用來解決 OARSMT 的方法大致可以分成以下五種的類型：(1) maze routing based approach、(2) non-deterministic approach、(3) construction-by-correction approach、(4) connection graph based approach、(5) hybrid approach。

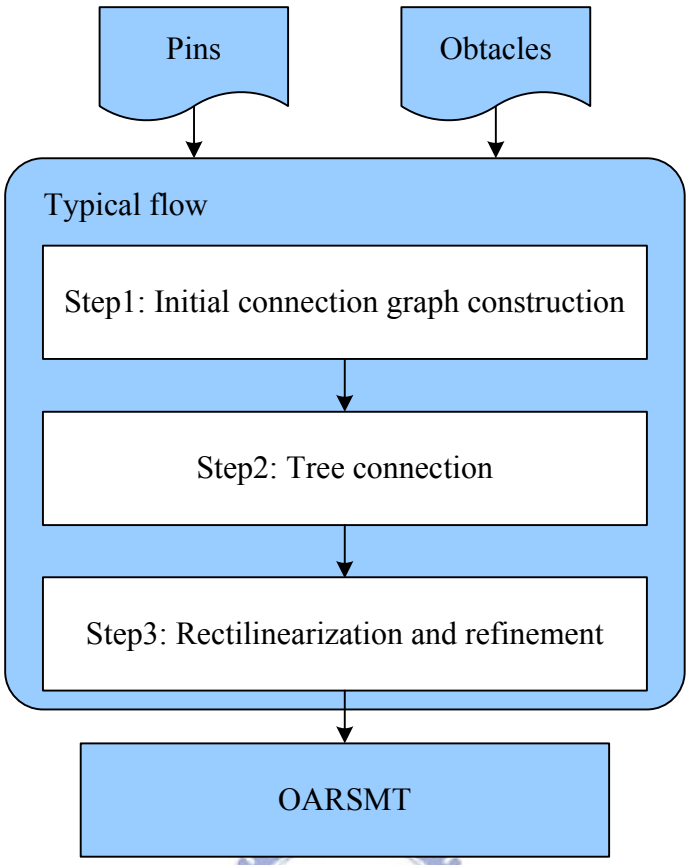


Figure 1.1: Typical flow of OARSMT.

若以步驟區分可以利用 Figure 1.1 所示的典型流程(typical flow)來完成。典型流程分成三個步驟：(1)建立 initial connection graph：將所有的接點連接起來，而形成一個初始圖。目前會用到的 initial connection graph 有完全圖(complete graph)，Delaunay triangulation (DT)，和 spanning graph 三類。(2)執行 tree connection：利用 initial connection graph，計算初始圖上邊(edge)的權重，可以建立對所有接點的最小生成樹(Minimum Spanning Tree, MST)。(3)直角化以及修正(rectilinearization and refinement)：基於 MST，我們可以將所有斜的 edge 轉換成直角轉彎的線段與一些史坦那點及 via 的組合，且可以進一步的縮短線長，一般都是利用 U 型線段修正。

綜合以上兩種劃分方式，整理了部分的相關研究如 Table 1-1。其中第一欄是方法的分類；第二欄表示考慮多層繞線(M)與否(S)；第三欄表示，第一步驟所建的初始圖種類及是否把障礙物的四個角落加入初始圖(Y/N)；第四和第五欄分別表示第二和第三步驟所用的方法。

Table 1-1: Comparison between recent researches on OARSMT.

Approach		Single/multi layer (S/M)	Initial graph construction w/ or w/o obstacles (Y/N)	Tree connection	Rectilinearization & refinement
Maze routing based		S	Wave propagation or line search		
Non-deterministic		S	Ant colony optimization		
Construction-by-correction	[6]	S	Delaunay triangulation (Y)	Obstacle-avoiding MST	Rectilinearization & refinement
	[4]	S	Complete graph (N)	Obstacle-weighted MST	Rectilinearization & refinement
	Ours	M	Delaunay triangulation (N)	Obstacle-weighted MST	Rectilinearization & 3D refinement
Connection graph based	[5]	S	Spanning graph (Y)	Obstacle-avoiding MST	Rectilinearization
	[3]	S	Improved spanning graph (Y)	Obstacle-avoiding MST	Rectilinearization & refinement
	[1]	M	3D improved spanning graph (Y)	Obstacle-avoiding MST	Rectilinearization & refinement
Hybrid [2]		S	Complete graph (N)	Mixed obstacle-avoiding & obstacle-weighted MST	Rectilinearization & refinement

1.2.1 Maze Routing Based Approach

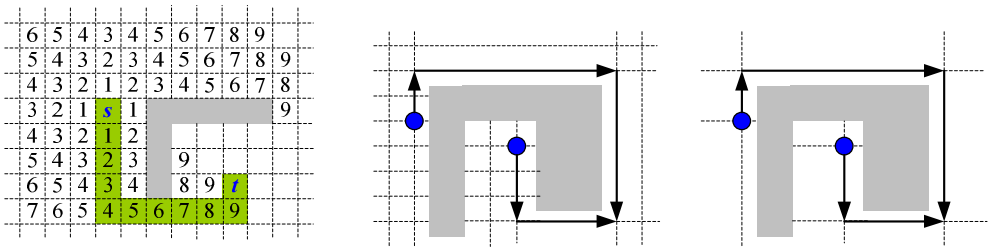


Figure 1.2: (a) Maze routing. (b) The first variant of line search. (c) The second variant of line search.

使用 maze routing 的方法來繞線一開始是由[10]所提出，目的在於找出可能兩接點連線(two-pin net)中，最短線長的結果。maze routing 的方法藉由 wave propagation 的方式來從起點(source) *s* 計算到終點(destination) *t*，如 Figure 1.2(a)。所以這種演算法在時間複雜度與記憶體的使用量都相當的驚人，一般來說，都不會使用這類的演算法來繞線。而改進 maze routing 而來的 line search 演算法使用 escape points 來使得繞線更有效率，如 Figure 1.2(b)和 Figure 1.2(c) [11][12]。但是延伸到多接點繞線上，依然沒辦法得到好的結果，因為畢竟是針對分段的 two-pin net 的各別繞線，所以始終沒有辦法得到整體更佳的結果。綜合以上的缺點來說，maze routing based 的演算法，並不常在現今的設計流程中使用。

1.2.2 Non-Deterministic Approach

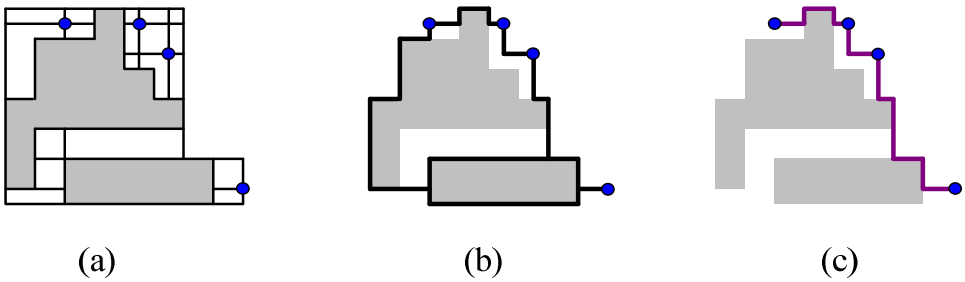


Figure 1.3: (a) A track graph. (b) The redundant edges are removed. (c) The SL-OARSMT.

Hu 等人在[8]提出了以螞蟻雄兵演算法(ant colony optimization)為基礎的一種 non-deterministic 局部搜尋法，可以彈性地用來處理較複雜的凸型(convex)或是凹型(concave)障礙物。這類的演算法首先建立了 track graph，如 Figure 1.3(a)。然後把一些不會用到的 edge 移除掉，如 Figure 1.3(b)。最後在每個接點上面放上一隻螞蟻，螞蟻會根據一些規則來決定下一個會走到的接點而形成 SL-OARSMT，結果如 Figure 1.3(c)。然而 non-deterministic 的方法比較適合處理接點與障礙物較少的問題，在處理大型輸入時會相當費時，所以不太適用於現今的設計流程。

1.2.3 Construction-by-Correction Approach

Construction-by-correction approach 的演算法是先建立一棵只連接所有接點的 MST 或是史坦那樹。然後再把所有會穿越障礙物的 edge，重新沿著障礙物的邊緣繞線。這類的演算法在業界非常常用，主要是因為方法很簡單且有效率。但是由於第一步驟所長出的最小生成樹或史坦那樹並沒有考慮到障礙物的問題，所以第二步驟對於穿越障礙物的邊，也只能局部的重繞，效能因此受到了影響。在[3]中指出，使用這類的方法來建構的 SL-OARSMT，實驗結果通常都會有所侷限。

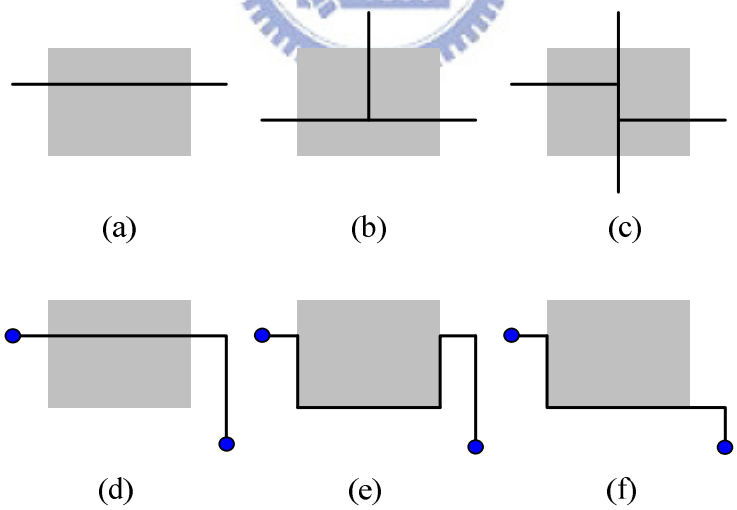


Figure 1.4: (a)-(c) Three cases of edges overlapping an obstacle. (d) An example of case (a). (e) The edge is routed along the obstacle. (f) The wirelength is further reduced.

Yang 等人在[14]中提出了一種演算法來移除穿越障礙物的 edge。首先把穿越障礙物的 edge 分成三種，如 Figure 1.4(a)-(c)。接著將穿越障礙物的邊移除，且沿著障礙物的周圍重繞，如 Figure 1.4(d)是 case (a)的例子，Figure 1.4(e)-(f)是重繞的結果。

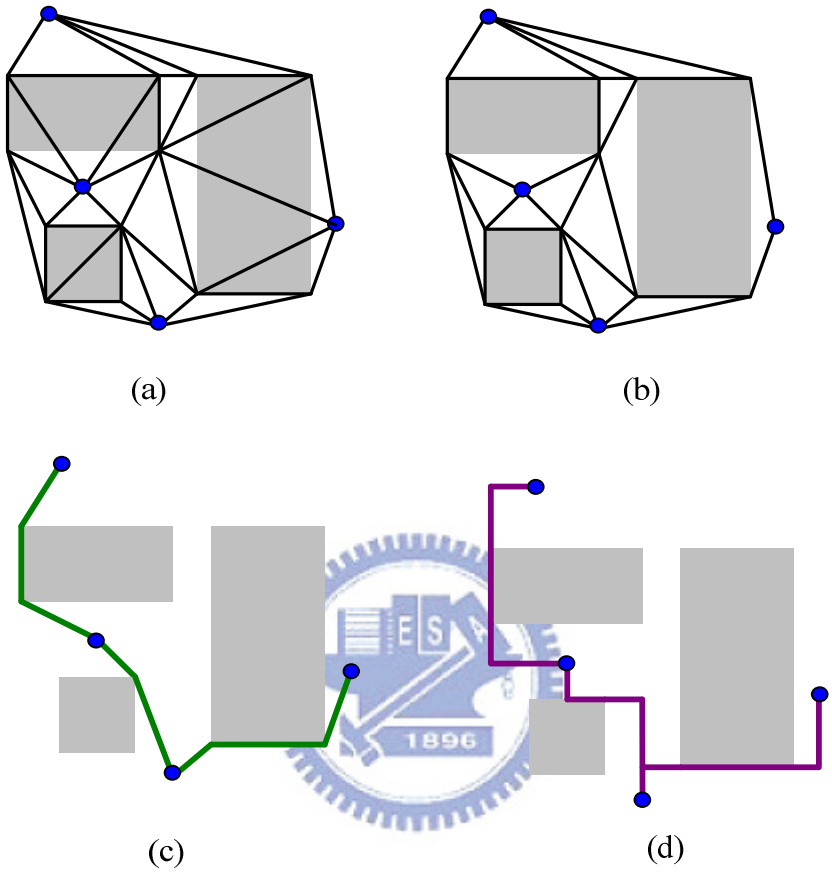
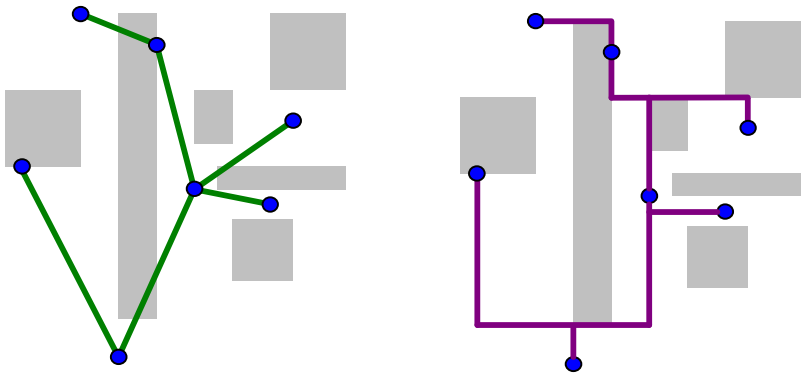


Figure 1.5: (a) Delaunay triangulation. (b) The edges overlapping obstacles are removed. (c) An obstacle-avoiding minimum spanning tree. (d) The SL-OARSMT.

Feng 等人在[6]中提出了利用 Delaunay triangulation (DT)演算法來實現多角幾何(λ–Geometry)的 SL-OARSMT。對照典型流程：第一步驟根據所有接點和障礙物的四個角落，建立 DT，如 Figure 1.5(a)，且移除所有和障礙物重疊的 edge，如 Figure 1.5(b)。第二步驟長出 MST 和移除不是接點的 leaves，如 Figure 1.5(c)。第三步驟將 MST 轉換為相對應的多角幾何的 SL-OARSMT，如 Figure 1.5(d)。[6]的特點在於將障礙物的四個角加入 DT 的建造。



(a) (b)

Figure 1.6: (a) An obstacle-weighted minimum spanning tree. (b) The SL-OARSMT.

Tsai 等人在[13]中，把障礙物的因素加到在兩點間 edge 的權重上，建立了障礙物加權(obstacle-weighted)之 MST，如 Figure 1.6(a)。最後針對樹中的每一條 two-pin net，根據 escape graph 來使用 Dijkstra 演算法，同時也會檢查已建好的史坦那樹是否有 U 型多餘線段產生，結果如 Figure 1.6(b)。

1.2.4 Connection Graph Based Approach

利用關連圖(connection graph)來完成 SL-OARSMT。首先會建立由接點和障礙物的四個邊界所連接起來的 connection graph，而可以保證至少會有一個 SL-OARSMT 的 solution 存在 connection graph 裡面。這類的方法跟 construction-by-correction approach 比較起來，可以整體性地考量接點和障礙物的相對關係。現有文獻中，這種演算法可以得到目前最好的結果。

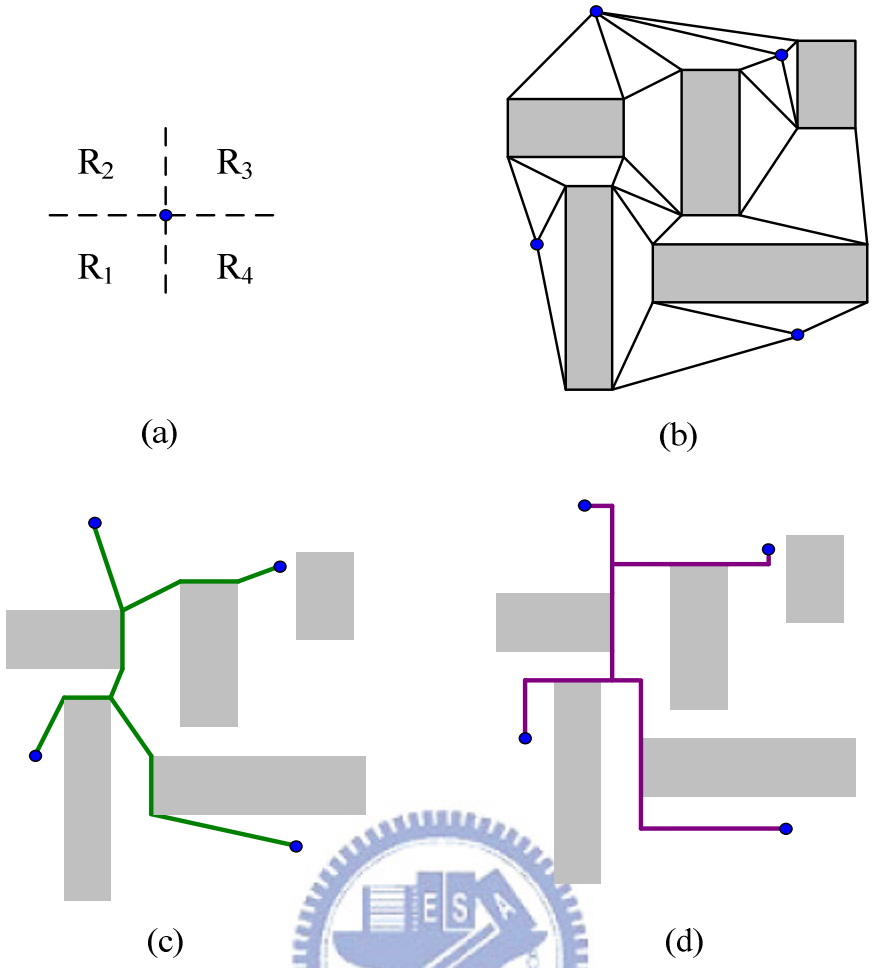


Figure 1.7: (a) The plane is divided into four regions with respect to a vertex. (b) An obstacle-avoiding spanning graph. (c) The minimum spanning tree. (d) A SL-OARSMT.

在[3]中，Shen 等人將每個端點(包含所有接點及障礙物的四個角落)對平面切成四個區域，如 Figure 1.7(a)。每個接點在四個區域中各找出最近的端點，而連接形成一條 edge。藉由這個方法，使得所有的接點和障礙物的四個角落，連接形成 spanning graph，如 Figure 1.7(b)。再轉化成 MST，如 Figure 1.7(c)。Lin 等人在[4]提出，在連接 spanning graph 時，加入更多的 essential edges，可以得到更完整的 spanning graph，進而得到更好的 SL-OARSMT，如 Figure 1.7(d)。

1.2.5 Hybrid Approach

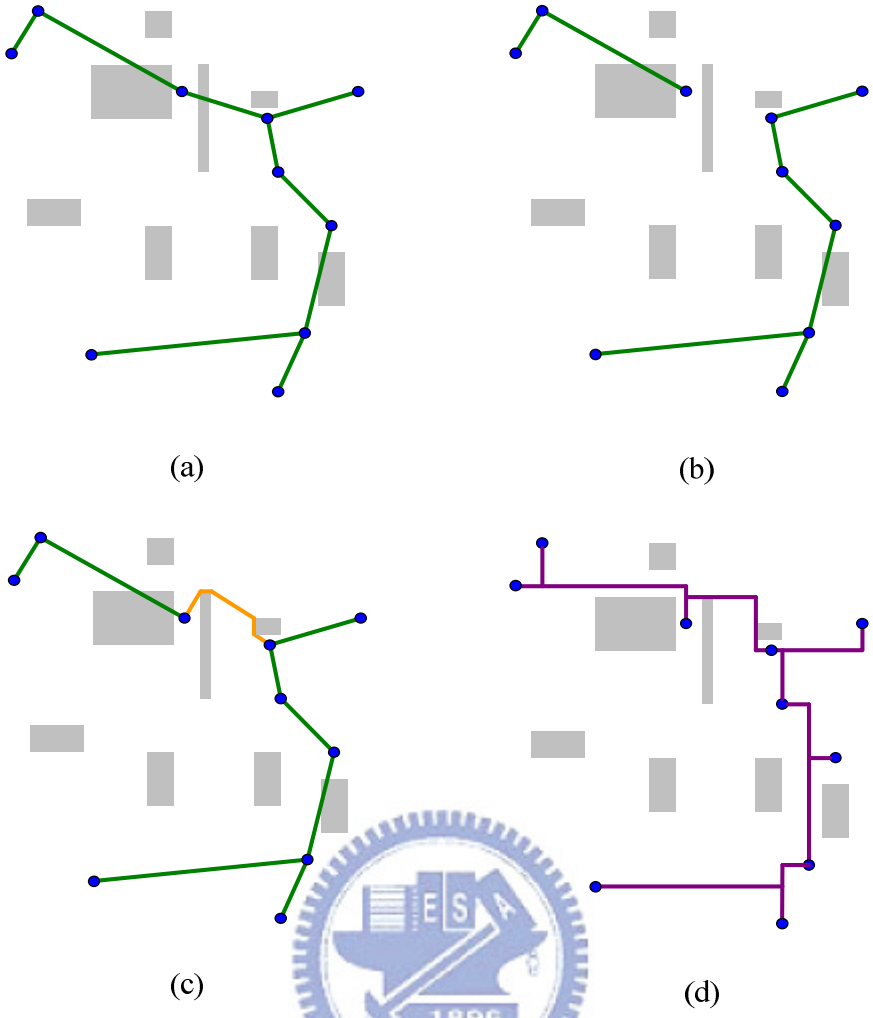


Figure 1.8: (a) An obstacle-weighted minimum spanning tree. (b) The edges overlapping the obstacles are removed. (c) The sub-trees are merged based on ant colony optimization. (d) The SL-OARSMT.

Wu 等人在[5]提出了與其他四種不太一樣的演算法。第一步驟對所有的接點找出障礙物加權之 MST，如 Figure 1.8(a)。在這邊所使用的邊權重(edge weight)並沒有[13]精準，只粗略估計上下兩個 L-型線段在最差情況下，要繞過所有的障礙物，加總起來的最長寬度。第二步驟移除所有跟障礙物重疊的 edge，而拆成多棵子樹(sub-trees)，如 Figure 1.8(b)。第三步驟利用螞蟻雄兵演算法將子樹連接起來，如 Figure 1.8(c)。然而這邊的螞蟻雄兵演算法跟[8]不同之處在於，[8]是在 track graph 上實現，但是[5]是在局部的 spanning graph 上實現以大幅減少執行的時間。最後是將重新連好的 tree，轉換成 SL-OARSMT，如 Figure 1.8(d)。

1.3 Contribution

本篇論文提出了以 construction-by-correction 為概念的演算法，可應用於解決 SL-OARSMT 和 ML-OARSMT 的問題。根據本篇論文所定義的典型流程(typical flow)：第一步驟對於所有接點建立 Delaunay triangulation 圖；第二步驟對所建立的 DT 圖進行障礙物加權之 MST 的建構；接著第三步驟將樹直角化且利用三維 U 型線段修正總線長。

本篇論文的特點在於：

1. 第一步驟不作完整圖(complete graph)，因為 complete graph 的時間複雜度是 $O(n^2)$，本篇論文利用 Delaunay triangulation，當作初始的連接圖，其時間複雜度是 $O(n\lg n)$。此外，此步驟中，我們只對所有接點作圖，並加入額外的 edge 在 DT 中，使得結果更好。
2. 第三步驟提出了新的 U 型線段修正方法並擴展到層與層之間的立體 U 型線段修正。特別的是，我們是對所有可能的立體 U 型線段作完整的分類，此處可以得到最佳解。

我們在 ML-OARSMT 的實驗結果顯示，當單顆 via 以 5 單位線長計算時，平均總線長比[15]少了 1.99%。且在 SL-OARSMT 的問題上，實驗結果平均也和目前最好的結果不相上下。

1.4 Thesis Organization

本篇論文在第二章將描述問題的定義和預備知識；第三章詳述本篇論文的演算法；第四章為本篇論文的實驗數據以及分析；第五章是結論以及未來研究的方向。

Chapter 2 Problem Formulation and Preliminaries

在本章首先會介紹本篇論文所研究的問題定義，以及將會使用到的相關研究知識。

2.1 Problem Formulation

首先定義幾個會用到的名詞與參數。

障礙物(obstacle)：障礙物皆是以矩形的形式出現，且一個障礙物只會出現在同一繞線層。兩兩障礙物不能互相重疊，只能互相接觸到，如 Figure 2.1。

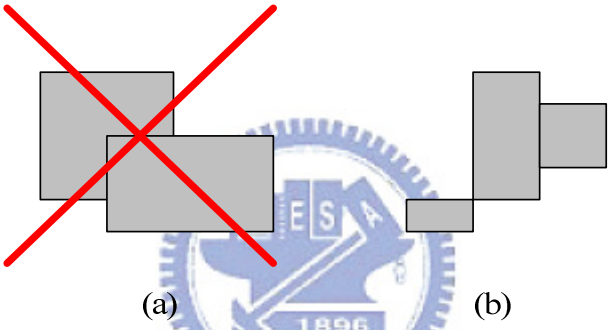


Figure 2.1: (a) Obstacles cannot overlap each other. (b) Obstacles can be point-touched at corners or line-touched at boundaries.

接點(pin)：一個接點只會出現在同一繞線層，且不能出現在障礙物的裡面，但可以在障礙物的邊界上，如 Figure 2.2。

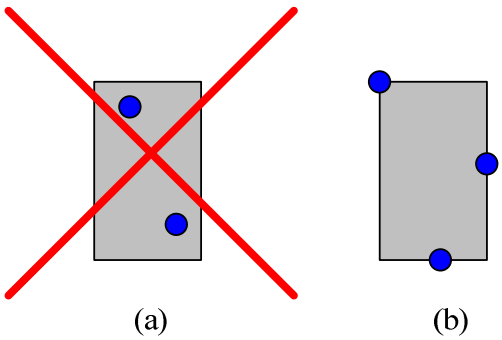


Figure 2.2: (a) A pin cannot locate inside any obstacle. (b) It can be at some corner or on some boundary of any obstacle.

via：一個 via 其座標為(x, y, z)，存在繞線層 z 和繞線層 z+1 間，且(x, y, z)和$(x, y, z+1)$皆不能存在障礙物的內部，不過可以在障礙物的邊界，如 Figure 2.3。

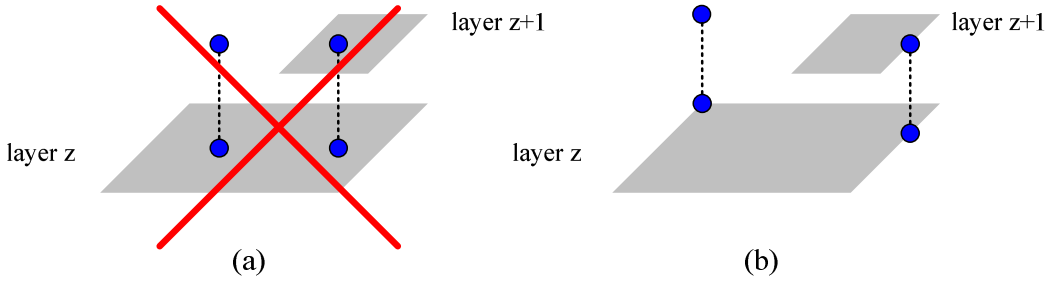


Figure 2.3: (a) A via cannot locate inside any obstacle. (b) It can be at some corner or on some boundary of any obstacle.

邊(edge)：每一條 edge 不能穿越過障礙物，只能沿著障礙物的邊界繞線，如 Figure 2.4。

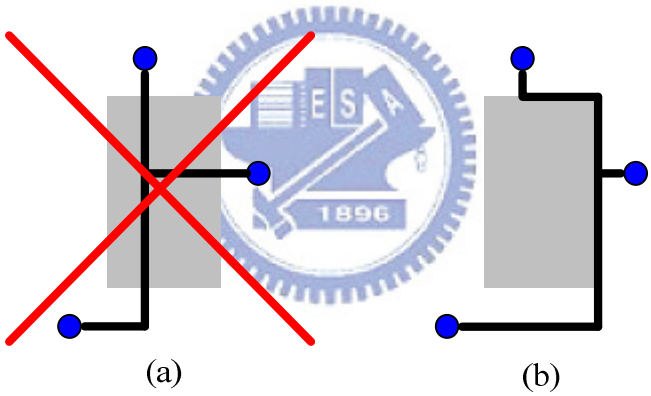


Figure 2.4: (a) An edge cannot intersect any obstacle. (b) It can be point-touched at some corner or line-touched on some boundary of any obstacle.

C_v：via 的 cost；一顆 via 對等於多少單位的線長。

N_l：提供的繞線層數。

P：所有 m 個接點座標的集合 $P = \{p_1, p_2, \ldots, p_m\}$。

O：所有 k 個障礙物座標的集合 $O = \{o_1, o_2, \ldots, o_k\}$。

n：所有點的個數，由 P 和 O 的四個角落座標貢獻。

本篇論文所要解決的問題有兩類：

1. 單層繞線下障礙物迴避之直角史坦那樹問題(SL-OARSMT)。
2. 考慮多層繞線與障礙物迴避之直角史坦那樹問題(ML-OARSMT)。

問題一：單層繞線下障礙物迴避之直角史坦那樹問題(SL-OARSMT)：

已知要連接的接點集合 P，障礙物集合 O，同在單一繞線層。目的為找出一棵迴避障礙物之直角史坦那樹去連接所有的接點，而可以利用額外的點(即史坦那點)，但是禁止穿越過障礙物，使得繞線總長最少。符合這些條件的樹，即所謂 SL-OARSMT。

問題二：考慮多層繞線與障礙物迴避之直角史坦那樹問題(ML-OARSMT)：

已知要連接的接點集合 P，障礙物集合 O，分佈在 N_l 個繞線層。目的為找出一棵迴避障礙物之直角史坦那樹去連接所有的接點，而可以利用額外的點(即史坦那點)及 via，但是禁止穿越過障礙物，使得繞線總長與 via 的總合花費最少。符合這些條件的樹，即所謂 ML-OARSMT。

2.2 Preliminaries

本篇論文將會簡單介紹所會用到的演算法，以及相關的知識：Delaunay triangulation [7]、Kruskal 最小生成樹演算法[9]、Dijkstra 最短路徑演算法[9]、非均勻間隔之繞線格(non-uniform routing grid)及 escape graph [13]。

2.2.1 Delaunay Triangulation

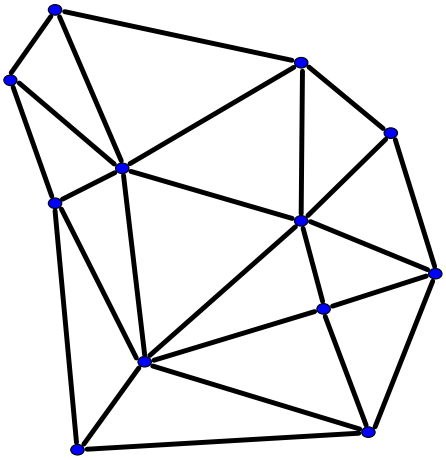


Figure 2.5: Delaunay triangulation.

Delaunay triangulation (DT) [7]是對接點集合 *P* 作每三個點三角化的演算法，且點不會存在任一三角形裡面，而每個三角形都不會互相重疊，如 Figure 2.5。在 DT 上每一個三角形的最小的角度都是越大越好，使得三角形的三個頂點中間的連接線，能夠越短。本篇論文所用到的 DT 演算法是[7]所實現的，其時間複雜度為 O(*n*lg*n*)。以下將簡述 DT 演算法。

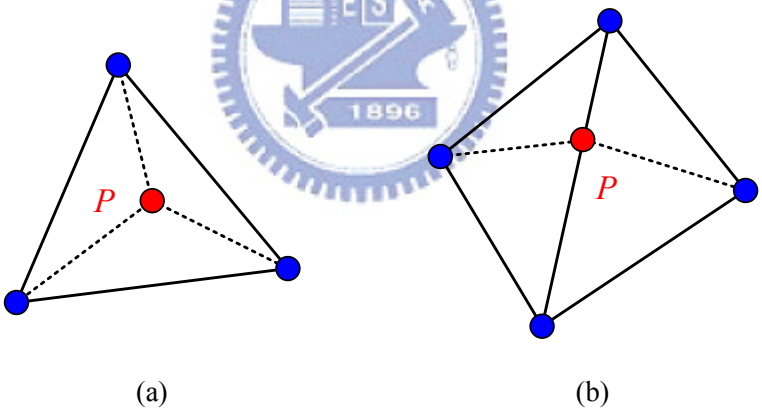


Figure 2.6: (a) A pin is inside a triangle. (b) A pin is onto an edge.

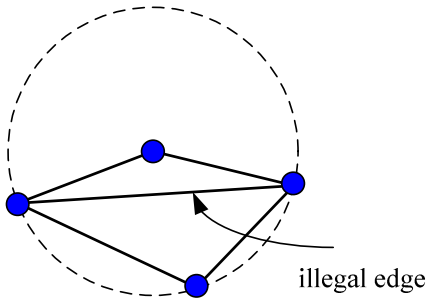
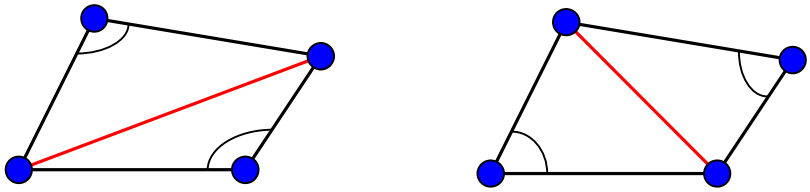


Figure 2.7: An illegal edge example.



(a) (b)

Figure 2.8: (a) An illegal edge. (b) It is transformed into a legal edge.

首先建立一個無窮大的三角形，使得所有的接點都位於三角形的內部。接著一次將一個接點 *p* 放入平面中，找出 *p* 是位於哪個三角形內(如 Figure 2.6(a))，或是哪一條 edge 的上面(如 Figure 2.6(b))。如果 *p* 位某個於三角形裡面，就連接 *p* 到三角形的三個頂點，形成三個三角形。檢查原本三角形的三個邊是否為 illegal edge，如果是，就將原本的 edge 進行 flip 的動作，如 Figure 2.8。illegal edge 的定義為包含 edge 的三角形的三個點形成一圓，如果圓內包含了其他頂點，即此 edge 為 illegal edge，如 Figure 2.7。如果 *p* 位於一條 edge 的上面，就連接 *p* 和其餘不在同一條線的頂點，如 Figure 2.6(b)。同時會檢查外圍的四個 edge 是否為 illegal edge，是就 flip。當所有的接點都放入平面中，然後移除無窮大三角形以及跟無窮大三角形的頂點有連接的 edge，即完成了最終的 DT。

2.2.2 Kruskal's Algorithm

用 Kruskal 演算法建立 MST [9]，是先將所有的 edge 移除並根據權重由小到大排序。由所有的點形成的 forest 開始，每個點都代表一棵子樹(sub-tree)。接著每次都選取連接二棵 sub-tree 而且權重最小的 edge 加入 forest 中，逐漸將 sub-tree 合併，最後只剩下一棵樹，即完成一棵 MST。

2.2.3 Dijkstra's Algorithm

Dijkstra 最短路徑演算法[9]，是用來找出單一起點的最短路徑(single source shortest path)。已知的圖 *G* 中，可以找出從起點(source) *s* 到圖 *G* 中任一終點(destination)的最短路徑。

用 d 值估算，任一點與起點 s 之間的最小路徑長度。先將起點 s 的 d 值設為 0 (即 $d[s]=0$)，同時把其他點 v 的 d 值設為無窮大($d[v]=\infty$)，且將所有點放入 priority queue。從 s 向外擴展至與 s 相鄰的點 v，同時將點 v 的 d 值改為 s 與 v 之間的 edge 權重 $w(s, v)$ ($d[v]=d[s]+w(s, v)= w(s, v)$)。接著將依序從 priority queue 中移除 d 值最小的點 v'，向與點 v'相鄰的點 v''作擴展，如果 $d[v'']$大於 $d[v'] + w(v', v'')$，則將 $d[v'']$的值更新為 $d[v'] + w(v', v'')$，且將 $d[v'']$的值放回 priority queue。直到 priority queue 空了為止，此時每個點的 d 值即為其與起點之間的最小路徑長度。

2.2.4 Non-Uniform Routing Grid and Escape Graph

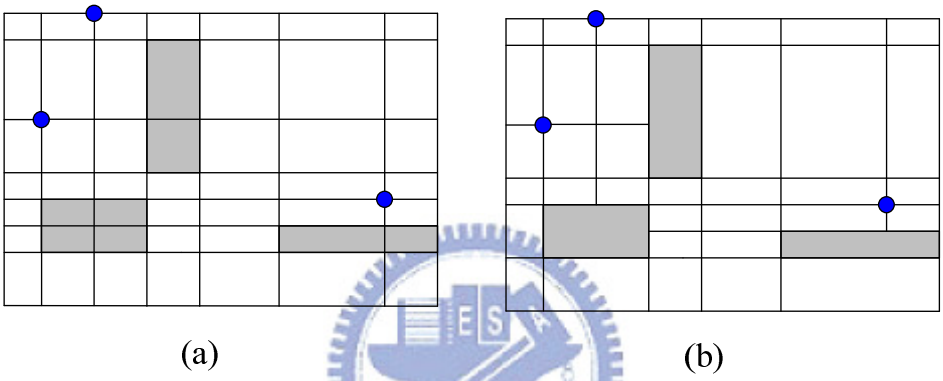


Figure 2.9: (a) An example of non-uniform routing grid. (b) The corresponding escape graph.

非均勻間隔之繞線格(non-uniform routing grid)是由所有的接點和障礙物的邊界，作水平及垂直方向的延伸線所組成，直到延伸到問題的邊界，如 Figure 2.9(a)。消除在非均勻間隔之繞線格中不必要的線段，即會形成 escape graph，如 Figure 2.9(b)。方法為對每一個接點和障礙物的四個角進行四個方向的擴展，如果遇到障礙物或邊界就停止形成線段。可以發現因為障礙物的阻擋 Figure 2.9(b) 比 Figure 2.9(a)少了四條線段。但由於 escape graph 的建構比非均勻間隔之繞線格複雜，並且由前人的實驗結果來看並無明顯的效果，所以本論文以非均勻間隔之繞線格來完成 SL-OARSMT 及 ML-OARSMT 的架構。並且[16]指出，如果存在一個障礙物迴避之直角史坦那樹最佳解的話，那麼最佳解一定會位於 escape graph 上，且 Steiner point 會位於 escape graph 中線段的交接點上。因此利用 escape graph 為基礎，在上面繞線，預期可以的得到最佳解。

Chapter 3 Our Algorithm

本章將介紹我們所提出的考慮多層繞線與障礙物迴避之直角史坦那樹(ML-OARSMT)演算法以及複雜度分析。Figure 3.1 是我們提出的流程，概述如下：

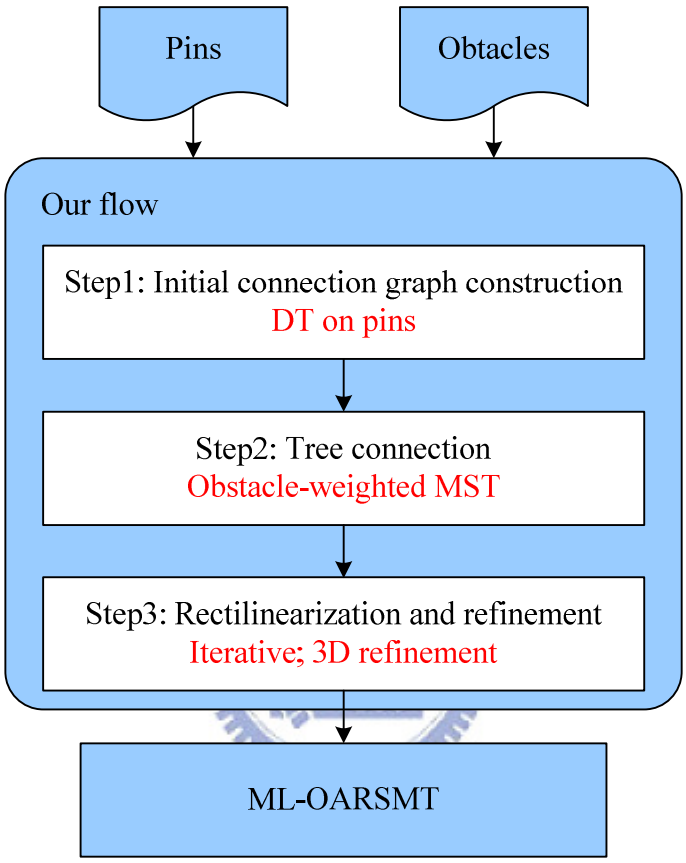


Figure 3.1: Our flow of ML-OARSMT.

(1) 建立 initial connection graph：將所有接點的位置投影到一個 pseudo plane，在此平面上，建立一個不含障礙物的 Delaunay triangulation 圖。

(2) 執行 tree connection：在 DT 中每一條 edge 權重的計算，我們將分兩種情況去考慮。如果一條 edge 的兩個接點原本分屬於不同層的話，我們暫不考慮障礙物的影響，而直接用曼哈頓距離(Manhattan distance)來當作 edge 的權重；否則，我們會將障礙物的影響加上曼哈頓距離當作 edge 的權重。在每一條 edge 的權重都計算完畢之後，接著利用 Kruskal 演算法，來建立障礙物加權(obstacle-weighted)之 MST。

(3) Rectilinearization and refinement：首先，建立根據接點和障礙物四個角落的座標，建立非均勻間隔之繞線格。接著利用 Dijkstra 最小路徑演算法在非均勻間隔之繞線格上面完成直角化繞線，並且利用我們所提的新的立體 U 型線段移除(removal)的方法，將線長縮短。特別的是，我們是對所有可能的立體 U 型線段作完整的分類，此處可以得到最佳解。

這三步驟結束即完成考慮多層繞線與障礙物迴避之史坦那樹(ML-OARSMT)。

3.1 Initial Connection Graph Construction

在第一步驟，我們只利用所有接點的座標(x, y, z)來建立 initial connection graph，其中 z 座標是所在的繞線層。

先將所有的接點投影到一個 pseudo plane，進行 Delaunay triangulation 的計算[7]。Delaunay triangulation 演算法是一次丟進一個接點到一個平面當中，所以會遇到三種情況

(1) 位於在某一個三角形裡：將會把這一個三角形分割成三個小三角形，如 Figure 2.6(a)，同時會對原來的三角形的三個邊檢查是否為 illegal edge。如果是的話，就翻轉這個 illegal edge，如 Figure 2.8；如果不是的話就不動作。

(2) 位於某個三角形的邊上：將所在的邊分成兩部分，而從原來的兩個三角形形成四個三角形，Figure 2.6(b)。同時會檢查外圍的四個 edge 是否為 illegal edge，是就 flip。

(3) 位於某個接點上：即一接點位於某一接點的正上方或是正下方。在此我們新增了一條 edge，連接了這兩個接點，而不破壞原本 DT 的架構。

當所有的接點都放入平面中，即完成了最終的 DT。此外，我們保留 illegal edge 在 DT 中，使得後續 MST 的建造可以更好。

3.2 Tree Connection

經由 initial connection graph 的建造，我們得到了一個 DT 圖，圖中將所有的接點都利用 DT edge 連接起來。接著在這一步驟的剛開始，我們計算所有在 DT 上的 edge 權重。根據 edge 所連接的兩接點的位置，我們有不同的計算方式。

若 edge 兩接點在同一繞線層時，我們利用了[13]中的方法，得到障礙物加權的 edge 權重。當 edge 兩接點在不同繞線層時，在此我們只用曼哈頓距離來估計 edge 的權重，實驗結果顯示，我們的方法仍能得到很好的表現。以下將介紹 edge 兩接點在同一平面的權重計算。

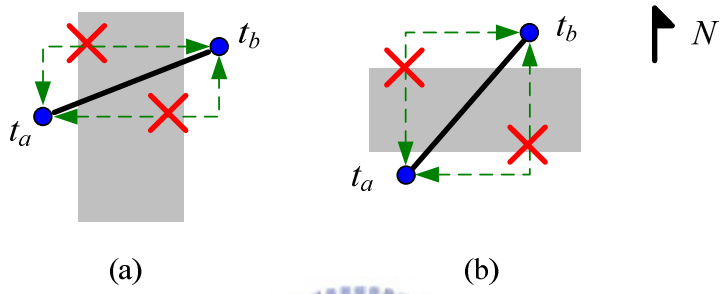


Figure 3.2: (a) An obstacle horizontally blocks an edge. (b) An obstacle vertically blocks an edge.

[13]定義若一連接 t_a 和 t_b 兩點的 edge 穿越障礙物 O 的話，則會是以下兩種情況之一：(1) 如圖 Figure 3.2(a)，障礙物東北角的 y 座標比 t_a 和 t_b 的 y 座標大，且障礙物西南角的 y 座標比 t_a 和 t_b 的 y 座標小，且障礙物東西邊界的 x 座標都在 t_a 和 t_b 的 x 座標之間。(2) 如圖 Figure 3.2(b)，障礙物東北角的 x 座標比 t_a 和 t_b 的 x 座標大，且障礙物西南角的 x 座標比 t_a 和 t_b 的 x 座標小，且障礙物南北邊界的 y 座標都在 t_a 和 t_b 的 y 座標之間。

這樣的定義可以確保障礙物一定會阻擋到 t_a 和 t_b 間的最小距離，也就是說 t_a 和 t_b 之間的南北兩個 L 型線段都會被此障礙物所檔到。Figure 3.3 中的三個例子，依此定義，皆不會被障礙物擋住，此時 edge 的權重為兩點間的曼哈頓距離 (Manhattan distance)。Figure 3.3(a)雖然南方 L 型線段被擋住，但是北方 L 型線段可以通過；Figure 3.3(b)雖北方 L 型線段被擋住，但是南方 L 型線段可以通過；Figure 3.3(c)雖然南北 L 型線段都被擋住，但還是可以經由中間通過，達到 t_a 和 t_b 間的最小距離。

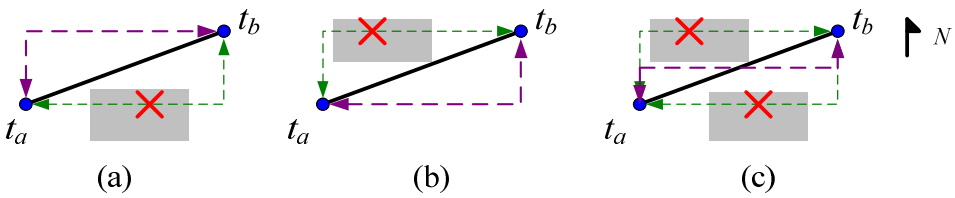


Figure 3.3: (a) Routing path passes the obstacle through upper L-shape segment, and (b) Routing path passes the obstacle through lower L-shape segment, and (c) Routing path passes the obstacles between obstacles.

經由以上的定義，我們將障礙物的權重加入在 DT 中每一條同一平面上會被阻擋的 edge，如 Figure 3.4。當一條 edge 無法經由南北方 L 型線段避開障礙物，就需沿著障礙物的邊緣而達成繞線，若從障礙物的北方所需要的距離為 D_{north} 而從南方繞過的距離為 D_{south}，即取這兩個距離中較小的值當作 edge 上的權重。

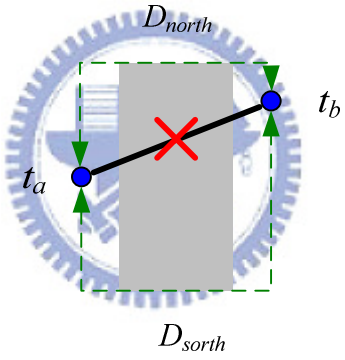


Figure 3.4: The edge weight computation on single layer.

當兩點間被多個障礙物所擋住的同時，先分別計算出每個障礙物的 D_{north} 和 D_{south}，且取其小值。接著從這些值中，選出最大的值當作是 edge 的權重。Figure 3.5 中，t_a 和 t_b 的連線被三個不同大小的障礙物所擋住。Figure 3.5(a) 中對 O_1 取出較小的 D_{north}，Figure 3.5(b) 中對 O_2 取出較小的 D_{south}，Figure 3.5(c) 中對 O_3 取出較小的 D_{south}。最後再比較這三個值的大小，得到最大的為 Figure 3.5(c) 中的 D_{north}。當障礙物由東西方向阻擋 DT 的 edge 時，亦可用類似方法求得 edge 的權重。

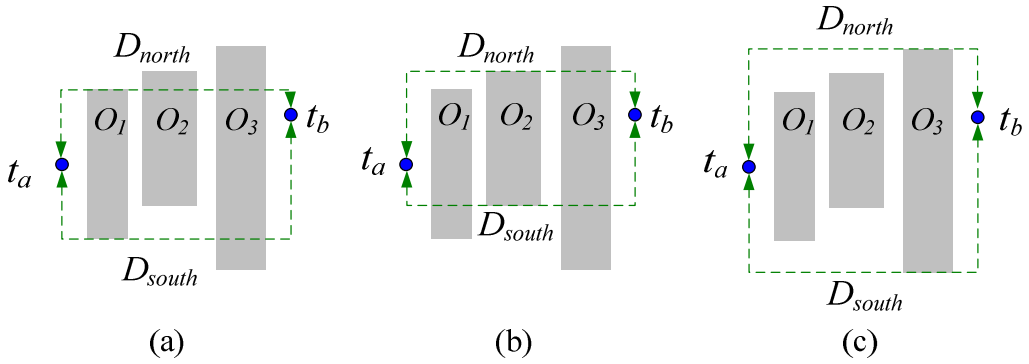


Figure 3.5: (a) (b) (c) The weights of L-shape segments dominated by Obstacle O_1, O_2, O_3, respectively.

當計算完所有在 DT 上 edge 的權重之後，我們利用 Kruskal 演算法來完成一棵連接所有接點的障礙物加權(obstacle-weighted)之 MST。

3.3 Rectilinearization and Refinement

在最後一步驟，首先會建立非均勻間隔之繞線格。然後利用非均勻間隔之繞線格上的直角 edge 來把原來的 MST 上斜的 edge 轉換成水平、垂直線段或經由 via 連接起來。在轉換的同時，會檢查是否與已繞好線段形成平面或立體的 U 型線段，進而使線長縮短。最後所有的 edge 都完成直角化及修正後，即得到一棵 ML-OARSMT。

3.3.1 Non-Uniform Routing Grid

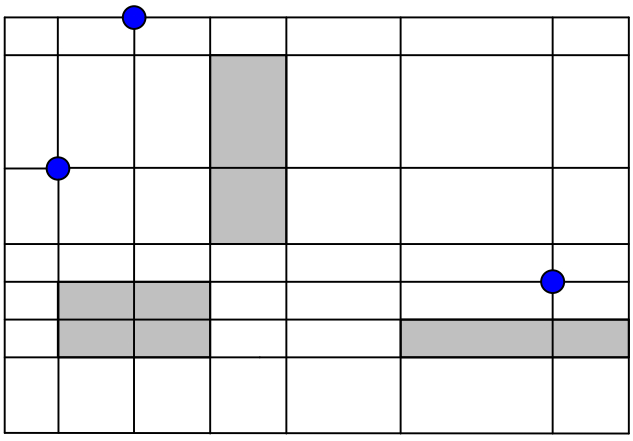


Figure 3.6: An example of non-uniform routing grid.

在讀進檔案之後，我們分別將輸入進來的 x、y、z 座標存在三個動態 array 中，並且對於所存的值加以排序。

根據三個 array 的大小，可以建立三維的非均勻間隔之繞線格。非均勻間隔之繞線格上的格子點(grid point)的 x 座標即為原本在 x array 中所存的值，y 座標和 z 座標也分別為 y array 和 z array 的中的值。Figure 3.6 中為二維的非均勻間隔之繞線格的例子。

對於每個格子點的六個方向，都標示能否通過的 flag。對於平面東南西北四個 flag，我們會檢查障礙物的四個邊，除了障礙物的四個角之外，其他在邊上的格子點都會分別標示不可通過，如 Figure 3.7 中紅色的標記。對於障礙物的正上方以及正下方兩個 flag 也會標示是否可以通過。當障礙物裡面有格子點，我們會在此格子點正上方的格子點標示不可往正下方通過；在正下方的格子點標示不可往正上方通過，如在 Figure 3.7 中西南角的障礙物，中間有格子點，因此會標示不可通過的方向，如 Figure 3.8。

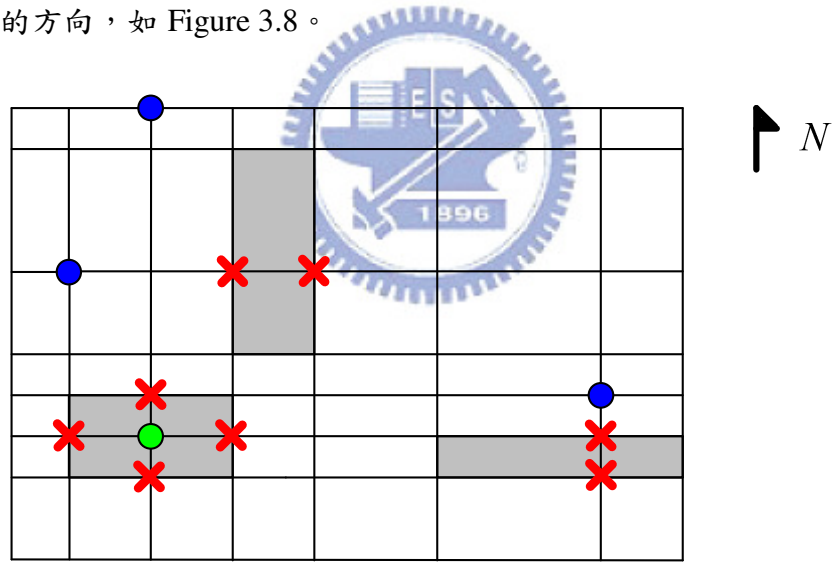


Figure 3.7: The forbidden directions of grid points at one layer.

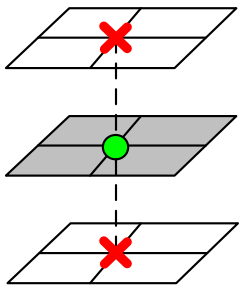


Figure 3.8: The forbidden directions of grid points at adjacent layers.

3.3.2 Dijkstra's Algorithm

在建造完非均勻間隔之繞線格之後，接著對步驟二所產生的 MST，來進行直角化。任選一接點當作 root 開始，不斷取出相鄰的 edge 進行直角化的繞線，並在每一條 edge 繞完線之後，使用下一小節的方法來檢查是否產生 U 型多餘線段並且修正。

對於 MST 中 edge 連接的兩個接點，必定會有一個在目前的直角史坦那樹上，我們視為終點(destination)；而另一個即將要加入史坦那樹的點則視為起點(source)。使用 Dijkstra 演算法找到了第一條從 root 連接出去最短路徑之後，對非均勻間隔之繞線格上面所經過的格子點，標示為已經過的線段。

接著對其他相鄰的 edge 從起點用 Dijkstra 演算法繞線，而 Dijkstra 演算法停止的條件為是否遇到其他線段所經過的格子點，不一定需要連接到目標點才會停止。以 Figure 3.9(a)為例，t_a 和 t_b 為已連接的線段，當要連接 t_c (source)和 t_b (destination)時，我們可以發現 t_c 並不會直接連接到 t_b，而是連接到 t_a 和 t_b 的中間線段，而形成 Steiner 點 S，如 Figure 3.9(b)。使得新的接點都和已繞好的史坦那樹成最短距離，以達到縮短總線長的目的。最後即可將 MST 轉成直角史坦那樹。

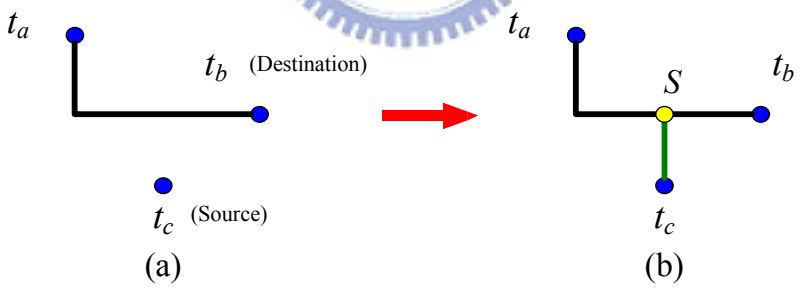


Figure 3.9: (a) The routed Steiner tree includes t_a and t_b, considered as destination. t_c is considered as source in Dijkstra's algorithm. (b) t_c is connected to the routed Steiner tree through a newly created Steiner point S.

3.3.3 3D U-Shape Refinement

為了使總線長可以更加縮短，在繞完每一條 edge 之後，都會檢查新繞好的線段，是否跟之前已建立好的直角史坦那樹產生 U 型多餘線段，而只要對此 U 型線段作修正就可以有效的縮短線長。

U 型多餘線段只有兩種可能：依照中間點的位置，第一種我們分類成 skewed U 型線段，如 Figure 3.10(a)，中間點位於 U 型線段的角落；第二種我們分類成 standard U 型線段如 Figure 3.10(c)，中間點位於 U 型線段的中間。如 Figure 3.10(a) 中(t_a, t_b)為已建立好的線段，且新連上的線段(t_c, S)是一條直線並在 S 處形成史坦那點，新線段和已建立好的史坦那樹產生了 U 型多餘線段(t_c, S, t_a)，最多可以縮短的線長如虛線箭頭所示。藉由 U 型多餘線段移除，使得線長縮短成 Figure 3.10(b)；Figure 3.10(c)中(t_a, t_b)為已建立好的線段，(t_c, t_b)為新建立的線段，新線段是 L 型線段並和舊線段形成了 U 型多餘線段(t_c, t_b, t_a)，最多可以縮短的線長如虛線箭頭所示，而如 Figure 3.10(d)顯示經由 U 型線段移除，可以得到更短的線長。

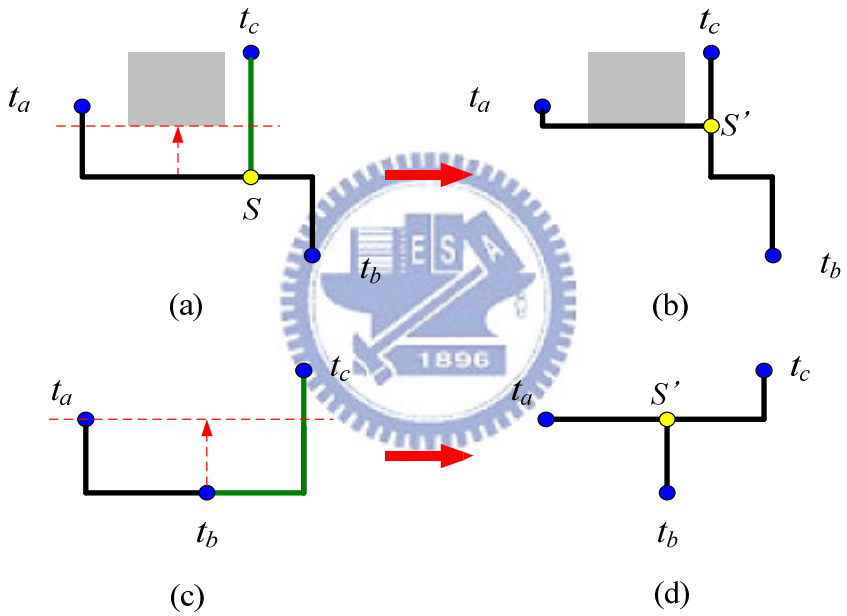


Figure 3.10: (a) A sample skewed U-shape segment. (b) The optimal solution after refinement. (c) A sample standard U-shape segment. (d) The optimal solution after refinement.

對於此二種 U 型線段，本論文進一步延伸至 3D 空間並提出個別的 U 型線段修正對策。特別的是，我們是對所有可能的立體 U 型線段作完整的分類，此處可以得到最佳解。

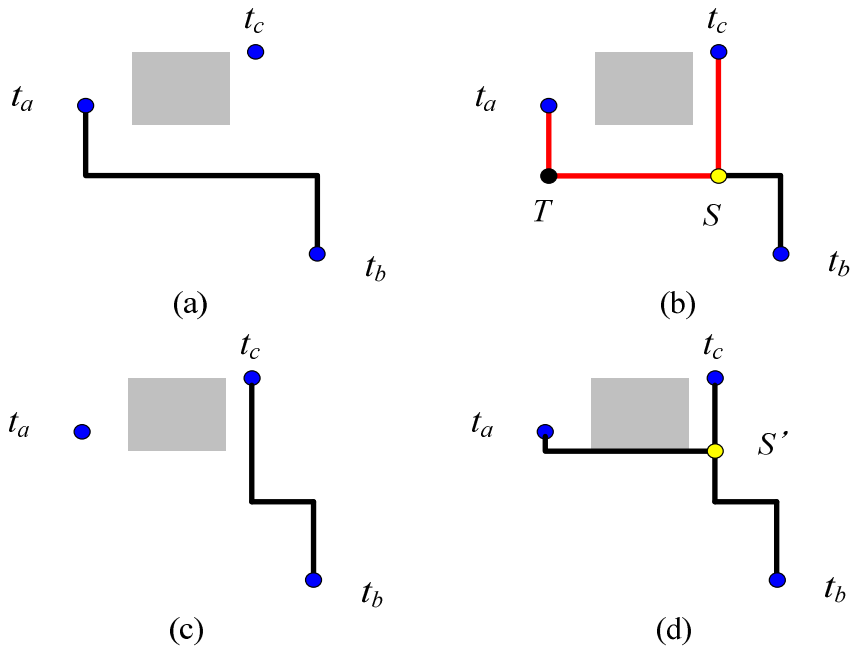


Figure 3.11: (a) A routed segment and a pin to be connect it. (b) A skewed U-shape segment. (c) Old segment is removed and rerouted. (d) The optimal solution.

(1) skewed U 型線段移除：因為這類的方法在多層繞線的問題上只會出現在同一平面，判斷的方式較為簡單，所以會先檢查是否為 skewed U 型線段，且如果不是此類型的話，必為 standard U 型線段。以 Figure 3.11 為例子來說明如何判斷：在 Figure 3.11(a)中(t_a, t_b)線段為已經連接好的線段，且 t_c 為新加入的點，經由 Dijkstra 演算法，t_c 跟原來的線段在 S 點形成 Steiner point，並且將原來的線段分割成兩條線段，如 Figure 3.11(b)。接著檢查新形成的線段(t_a, S)跟這兩條舊線段是否產生 U 型多餘線段。判斷的方法為分別計算新線段和舊線段的總線長和曼哈頓距離，如果總線長大於曼哈頓距離的話，代表可能有 U 型多餘線段的產生。如(t_c, S)和(t_a, S)的總線長大於(t_c, t_a)的曼哈頓距離，所以可能有U型多餘線段。接著判斷是哪一類型的 U 型線段，我們會先判斷從中間點到新線段的第一條直線方向(即 S 至 t_c 方向)，和從中間點到舊線段的第一個 L 型線段結尾的方向(即 T 至 t_a 方向)，如果兩邊的方向一樣的話，即形成了 skewed U 型線段。我們可以在 Figure 3.11(b)中發現新線段(t_c, S)和舊線段(t_a, S)產生了 skewed U 型線段，接著我們移除舊線段和 S 點，並把新線段和另一條舊線段合併起來，如 Figure 3.11(c)。重新對 t_a 到線段(t_c, t_b)進行 Dijkstra 演算法，如圖 Figure 3.11(d)所示，可以讓總線長可以縮短。

本論文跟[13]，不同之處在於，如果兩條舊線段都和新線段有 U 型多餘線段的產生，[13]只會找新線段跟舊線段差距最大的組合進行 U 型線段修正。本論文也是從差距最大的組合開始修正，但是如果修正後沒有改進線長的話，將會繼續對其他有 U 型線段的組合進行修正，直到有縮短到線長為止，或是沒有多餘線段的組合可以進行修正為止。

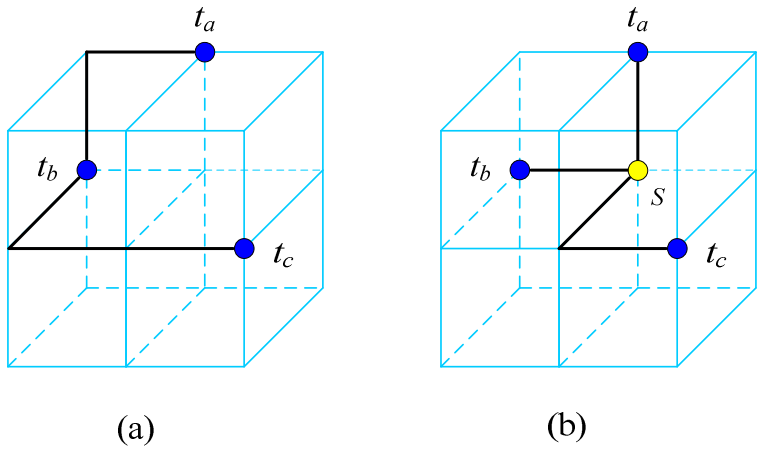


Figure 3.12: (a) Case I of standard U-shape segment. (b) The optimal solution of case I.

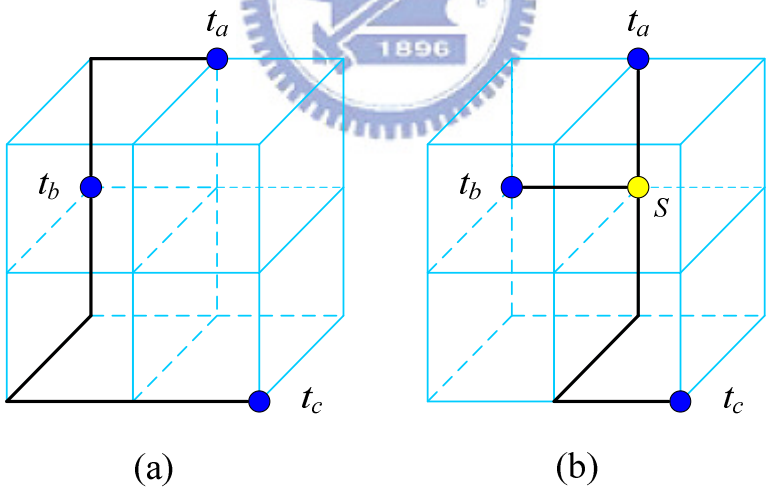


Figure 3.13: (a) Case II of standard U-shape segment. (b) The optimal solution of case II.

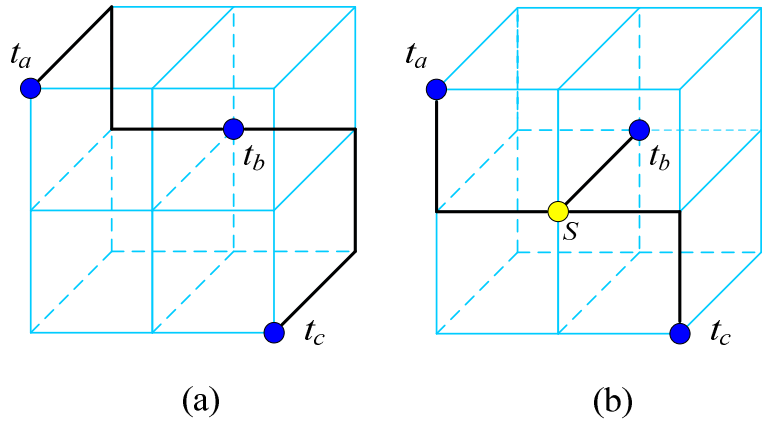


Figure 3.14: (a) Case III of standard U-shape segment. (b) The optimal solution of case III.

(2) standard U 型線段移除：如果不屬於 skewed U 型線段的話，即為 standard U 型線段，我們分成三類來考慮。

- 第一類：從中間點到新線段的第一個 L 型線段結尾的方向，和中間點到舊線段的第一個 L 型線段結尾的方向，如果一樣的話就為第一類，如圖 Figure 3.12(a)，而 Figure 3.12(b)為線長修正後史坦那樹。
- 第二類：從中間點到新線段(舊線段)的第一個 L 型線段結尾的方向，和中間點到舊線段(新線段)的第二個 L 型線段結尾的方向，如果一樣的話就為第二類，如圖 Figure 3.13(a)，而 Figure 3.13(b)為線長修正後的史坦那樹。
- 第三類：從中間點到新線段的第二個 L 型線段結尾的方向，和中間點到舊線段的第二個 L 型線段結尾的方向，如果一樣的話就為第三類，如 Figure 3.14(a)，而 Figure 3.14(b)為所線長縮短的史坦那樹。

關於細節的部分將在圖 Figure 3.15 中說明。Figure 3.15(a)中為(t_a, t_b)線段為已繞線的部分，而 t_c 為新加入的點。當(t_c, t_b)新線段和舊線段形成 standard U 型線段，如 Figure 3.15(b) (此為第一類)，在此找出 t_a、t_b、t_c 三點所形成的 Steiner point 的位置，如 Figure 3.15 (c)。接著從中間點 t_b 往 S 的方向形成一個線段，且此線段如果被障礙物檔到，而無法到達 S 點時，就會在障礙物的邊界中停止，並且將新線段和舊線段移除，如圖 Figure 3.15(d)-(e)。

再來 t_a 和 t_c 分別對 t_b 到 S 的線段作繞線，因而得到縮短線長的目的，如 Figure 3.15(f)。

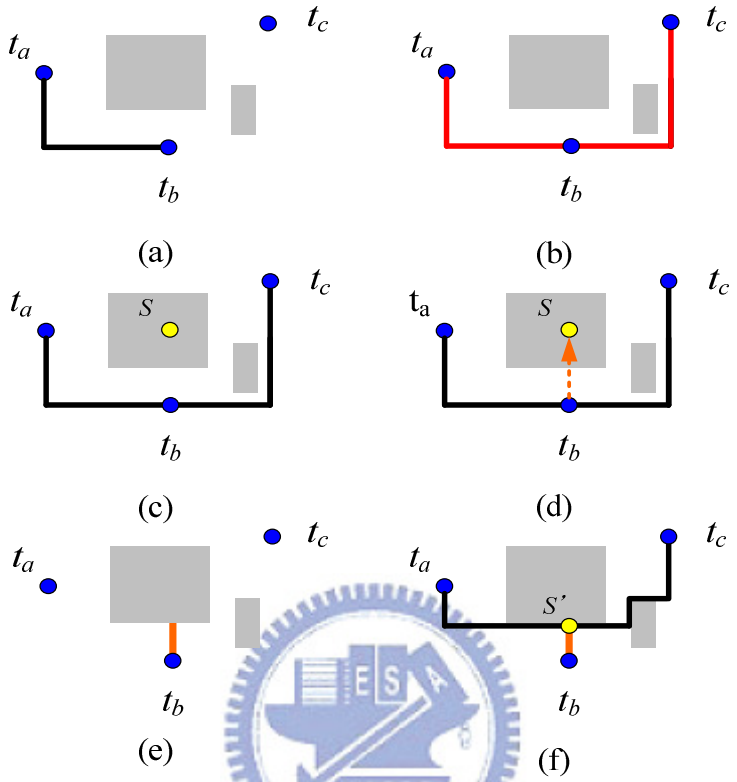


Figure 3.15: (a) (t_a, t_b) is routed, and t_c is to be connected. (b) (t_a, t_b, t_c) forms a standard U-shape segment. (c) The Steiner point S corresponds to t_a, t_b, t_c. (d) The segment from the middle point t_b to S is found. (e) The new segment is blocked by an obstacle, and old segments are removed. (f) After rerouting them, the optimal solution is found.

3.4 Complexity Analysis

空間複雜度分析：本論文使用了三維非均勻間隔之繞線格，是根據所有接點和障礙物的座標所建立的，最差情況(worst case)為所有接點和障礙物的座標都沒有重複到，因此對於每一個接點和障礙物都需要使用記憶體，以 n 個接點和 m 個障礙物來說，空間複雜度的 worst case 為 $O((n+2m)^3) = O((n+m)^3)$。

時間複雜度分析：建立對所有接點的 DT 所花的時間為 $O(n\lg n)$，利用 Kruskal 演算法所花時間為 $O(n^2)$，而建立三維非均勻間隔之繞線格所花的時間為

$O((n+m)^3)$。在此假設所有接點的位置平均分佈在整個三維的空間，所以每兩個接點間隔的格子數為$(\frac{n+2m}{n})$，所以在對每一 edge 作 Dijkstra 演算法繞線的複雜度為 $O((\frac{n+2m}{n})^3)= O(\frac{n+m}{n})^3$，而繞完全部的 edge 所花的時間為 $O(n(\frac{n+m}{n})^3)$。所以本論文的時間複雜度為 $O(n\lg n)+ O(n^2)+O((n+m)^3)+ O(n(\frac{n+m}{n})^3)$，化簡後為 $O((n+m)^3)$。

Chapter 4 Experimental Results

本論文以 C++來實作整個架構，使用 Pentium® 4 3.4GHz CPU，1GB 記憶體的平台測試。

4.1 The SL-OARSMT Problem

對於單層繞線 SL-OARSMT 的問題，本篇論文總共執行 14 個測試檔案，並且將實驗結果與三篇目前最好的結果做比較[5][4][13]。[5]的測試平台為 Sun Blade 2000，1.2GHz CPU，8GB 記憶體；[4]的測試平台為 2GHz AMD-64 machine，8GB 記憶體，作業系統為 Ubuntu 6.06。[13]的測試平台為 Sun Blade 2000，1.2GHz CPU，4GB 記憶體，作業系統為 Solaris 2.9 版。

Table 4-1 和 Table 4-2 中列出本論文所得到的實驗結果和[5][4][13]的比較。Table 4-1 是沒有加入 illegal edge 的結果；Table 4-2 是加入 illegal edge 的結果。第一欄代表的是測試檔案的名稱，第二欄第三欄分別代表接點和障礙物的個數，接著為各研究的總線長和我們的結果。粗體代表的是在[5][4][13]中，每一個測試檔案的最短線長。最後是我們的總線長相對於[5][4][13]的結果所改善的百分比，其計算方法為($\frac{x-Ours}{x}$)，其中 x 代表粗體的總線長。而 Best 欄為我們的總線長相對於[5][4][13]中每一筆測試檔案的最佳結果所改善的百分比。

Table 4-3 和 Table 4-4 則是考慮第二步驟和第三步驟的 total wirelength 和 CPU time。第五欄是第二步驟形成 MST 之後，我們對所有 edge 作權重的加總，且第六欄為完成第一第二步驟的執行時間。第七第八欄為第三步驟只對所有 edge 作直角化的結果。第九第十欄為第三步驟所有 edge 作直角化兼作 U 型線段修正。最右邊一欄代表的是樹有作 U 型線段修正與否，所得到的改善百分比。實驗結果發現我們的 Total cost 平均改善了 2.69%和 2.63%。另一方面，加入 illegal edge 的確如我們預期地產生了 Total wirelength 比較小的 MST，並且也幾乎都能產生比較好的 SL-OARSMT。

Figure 4.1 為測試檔案 Rc9 所實際繞出的障礙物迴避之直角化史坦那樹，其中包含了 100 個接點和 500 個障礙物，而灰色矩形代表障礙物，黑色小方塊代表接點。

Table 4-1: The comparison on total wirelength *without* illegal edges, where "-"means that the result is not available.

Test cases	Pins	Obs.	Total Wirelength				Improvement (%)			
			[5]	[4]	[13]	Ours	[5]	[4]	[13]	Best
Rc1	10	32	626	632	**614**	609	2.72	3.64	0.81	0.81
Rc2	74	625	1,640	-	**1,632**	1,621	1.16	-	0.67	0.67
Rc3	115	1204	2,872	-	**2,820**	2,832	1.39	-	-0.43	-0.43
Rc4	10	10	27,250	26,900	**26,120**	25,980	4.66	3.42	0.54	0.54
Rc5	30	10	43,220	**42,210**	42,320	43,040	0.42	-1.97	-1.70	-1.97
Rc6	50	10	56,500	55,750	**55,170**	54,690	3.20	1.90	0.54	0.54
Rc7	70	10	61,090	60,350	**59,670**	61,210	-0.20	-1.43	-2.58	-2.58
Rc8	100	10	76,870	76,330	**75,410**	75,680	1.55	0.85	-0.36	-0.36
Rc9	100	500	84,327	83,365	**81,904**	81,126	3.80	2.69	0.95	0.95
Rc10	200	500	115,461	113,260	**112,391**	112,318	2.72	0.83	0.06	0.06
Rc11	200	800	122,574	118,747	**117,602**	116,578	4.89	1.83	0.87	0.87
Rc12	200	1000	120,017	116,168	**115,448**	115,066	4.13	0.95	0.33	0.33
Rc13	500	100	172,490	170,690	**169,160**	169,040	2.00	0.97	0.07	0.07
Rc14	1000	100	238,377	**236,615**	237,475	236,570	0.76	0.02	0.38	0.02
Avg.							2.37	1.14	0.04	-0.03

Table 4-2: The comparison on total wirelength *with* illegal edges, where "-"means that the result is not available.

Test cases	Pins	Obs.	Total Wirelength				Improvement (%)			
			[5]	[4]	[13]	Ours	[5]	[4]	[13]	Best
Rc1	10	32	626	632	**614**	609	2.72	3.64	0.81	0.81
Rc2	74	625	1,640	-	**1,632**	1,622	1.10	-	0.61	0.61
Rc3	115	1204	2,872	-	**2,820**	2,814	2.02	-	0.21	0.21
Rc4	10	10	27,250	26,900	**26,120**	25,980	4.66	3.42	0.54	0.54
Rc5	30	10	43,220	**42,210**	42,320	43,040	0.42	-1.97	-1.70	-1.97
Rc6	50	10	56,500	55,750	**55,170**	54,690	3.20	1.90	0.54	0.54
Rc7	70	10	61,090	60,350	**59,670**	61,210	-0.20	-1.43	-2.58	-2.58
Rc8	100	10	76,870	76,330	**75,410**	75,560	1.70	1.01	-0.20	-0.20
Rc9	100	500	84,327	83,365	**81,904**	81,126	3.80	2.69	0.95	0.95
Rc10	200	500	115,461	113,260	**112,391**	112,318	2.72	0.83	0.06	0.06
Rc11	200	800	122,574	118,747	**117,602**	116,456	4.99	1.93	0.97	0.97
Rc12	200	1000	120,017	116,168	**115,448**	115,066	4.13	0.95	0.33	0.33
Rc13	500	100	172,490	170,690	**169,160**	168,770	2.16	1.12	0.23	0.23
Rc14	1000	100	238,377	**236,615**	237,475	236,570	0.76	0.02	0.38	0.02
Avg.							2.44	1.02	0.11	0.04

Table 4-3: The impact of refinement in SL-OARSMT *without* illegal edges.

Test cases	Pins	Obs.	Tree connection		Rectilinearization		Rectilinearization and 3D refinement		
			Total cost	CPU time (s)	Total cost	CPU time (s)	Total cost	CPU time (s)	Imp. (%)
Rc1	10	32	659	<0.01	622	0.015	609	0.016	2.09
Rc2	74	625	1,712	0.015	1,640	0.171	1,621	0.187	1.16
Rc3	115	1204	3,309	0.031	2,873	0.344	2,832	0.406	1.43
Rc4	10	10	29,775	<0.01	28,080	0.031	25,980	0.016	7.48
Rc5	30	10	45,450	<0.01	44,820	0.047	43,040	0.047	3.97
Rc6	50	10	59,540	0.015	57,370	0.062	54,690	0.093	4.67
Rc7	70	10	65,300	0.015	63,280	0.110	61,210	0.156	3.27
Rc8	100	10	84,080	0.016	77,580	0.156	75,680	0.297	2.45
Rc9	100	500	89,415	0.031	83,285	8.157	81,126	29.391	2.59
Rc10	200	500	121,545	0.063	113,992	9.516	112,318	40.515	1.47
Rc11	200	800	128,210	0.062	118,312	20.343	116,578	95.437	1.47
Rc12	200	1000	124,293	0.078	116,990	28.969	115,066	293.688	1.64
Rc13	500	100	185,950	0.266	173,080	2.734	169,040	19.75	2.33
Rc14	1000	100	260,846	0.875	243,518	11.234	236,570	158.328	2.85
Avg.									2.78

Table 4-4: The impact of refinement in SL-OARSMT *with* illegal edges.

Test cases	Pins	Obs.	Tree connection		Rectilinearization		Rectilinearization and 3D refinement		
			Total cost	CPU time (s)	Total cost	CPU time (s)	Total cost	CPU time (s)	Imp. (%)
Rc1	10	32	659	<0.01	622	0.031	609	0.016	2.09
Rc2	74	625	1,709	0.016	1,645	0.219	1,622	0.219	1.40
Rc3	115	1204	3,039	0.046	2,855	0.437	2,814	0.500	1.44
Rc4	10	10	29,770	<0.01	28,080	0.031	25,980	0.031	7.48
Rc5	30	10	45,450	<0.01	44,820	0.031	43,040	0.047	3.97
Rc6	50	10	59,540	0.015	57,370	0.063	54,690	0.109	4.67
Rc7	70	10	65,300	0.016	63,280	0.093	61,210	0.712	3.27
Rc8	100	10	83,990	0.015	77,570	0.156	75,560	0.297	2.59
Rc9	100	500	89,415	0.031	83,285	8.062	81,126	28.671	2.59
Rc10	200	500	121,545	0.062	113,992	10.375	112,318	41.250	1.47
Rc11	200	800	128,094	0.063	118,130	22.688	116,456	98.032	1.42
Rc12	200	1000	124,293	0.062	116,990	26.031	115,066	274.609	1.64
Rc13	500	100	185,910	0.265	172,920	2.813	168,770	20.547	2.40
Rc14	1000	100	260,846	0.782	243,518	11.360	236,570	158.984	2.85
Avg.									2.81

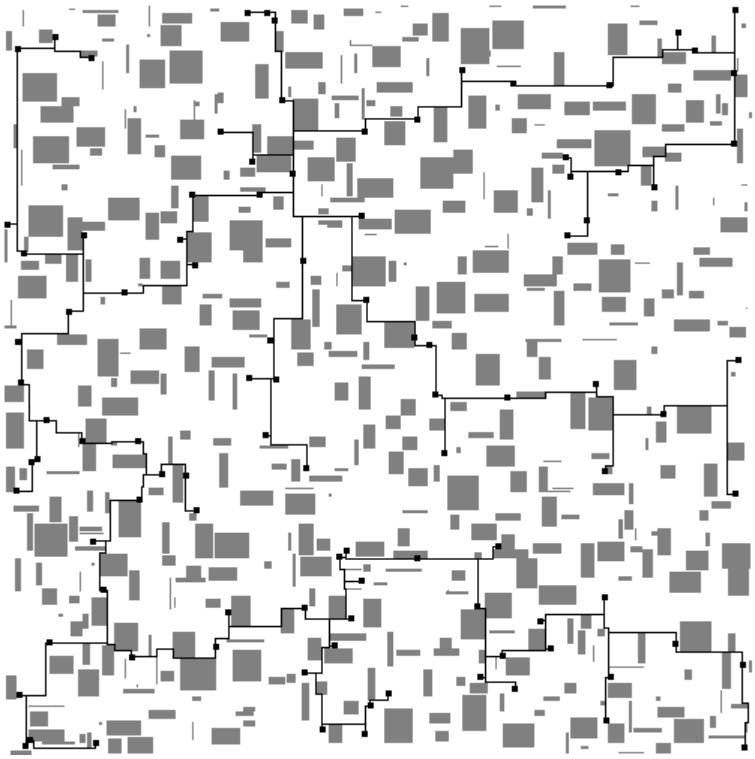


Figure 4.1: The final routing result of Rc9.

4.2 The ML-OARSMT Problem

對於多層繞線的 ML-OARSMT 問題，我們跟[15]作比較，[15]的測試平台為 2.8GHz AMD-64 machine，8GB 記憶體，作業系統為 Ubuntu 6.06。

多層繞線和單層繞線不同的地方在於 via 的出現。在製程中，via 越多所造成的成本相對來說也越多，[15]中將 via 的成本換算成線長來作繞線，via cost (C_v)為 3 和 5。Table 4-5 和 Table 4-6 是 C_v 等於 5 並考慮 illegal edge 與否時的總線長比較；Table 4-7 和 Table 4-8 是 Cv 等於 3 並考慮 illegal edge 與否時的總線長比較。在第一欄代表的是測試檔案的名稱，第二欄代表的是接點的個數，第三欄代表障礙物的個數，第四欄代表繞線層的層數。WL 代表的是所有繞線層上的總線長，#via 代表 via 的個數，Total cost 表示加上 via cost 後的總線長，實驗結果發現對於 C_v=5 的情況而言，我們的 Total cost 平均少了 1.99%和 1.97%；C_v=3 的情況而言，我們的 Total cost 平均少了 1.68%和 1.76%，且 WL 和 via 數目平均上都有不錯的結果。因為本論文大量的使用 Dijkstra 演算法，所以執行時間多出了許多，但仍在可以接受的範圍內。

Table 4-5: The comparison on total cost when C_v=5 *without* illegal edges. "WL" is the wirelength on all layers; "#via" is the number of vias between layers; "Total Cost" is "WL+ C_v *#via"; "Imp." is the improvement on WL, #via, and total cost ($\frac{[15]-Ours}{[15]}$), respectively.

Test Case	Pins	Obs.	N_l	WL			#via			Total cost			CPU time (s)	
				[15]	Ours	Imp. (%)	[15]	Ours	Imp. (%)	[15]	Ours	Imp. (%)	[15]	Ours
Ind1	50	6	4	55,390	55,180	0.38	49	50	-2.04	55,635	55,430	0.37	0.07	0.08
Ind2	200	85	6	11,859	11,604	2.15	208	214	-2.88	12,899	12,674	1.74	3.01	3.27
Ind3	250	13	9	9,983	9,805	1.78	343	343	0.00	11,698	11,520	1.52	3.35	20.89
Ind4	500	100	4	77,033	78,335	-1.69	0	0	0.00	77,033	78,335	-1.69	8.21	8.45
Ind5	1000	20	4	14,515,511	14,496,361	0.13	0	0	0.00	14,515,511	14,496,361	0.13	45.71	389.28
Rt1	25	10	10	4,106	3,846	6.33	76	67	11.84	4,486	4,181	6.80	0.06	0.33
Rt2	100	20	10	8,812	8,661	1.71	200	191	4.50	9,812	9,616	2.00	0.90	4.80
Rt3	250	50	10	14,239	13,888	2.47	429	412	3.96	16,384	15,948	2.66	6.84	38.48
Rt4	500	50	10	19,488	18,785	3.61	879	862	1.93	23,883	23,095	3.30	17.56	179.77
Rt5	1000	100	4	25,528	24,884	2.52	814	762	6.39	29,598	28,694	3.05	56.45	276.94
Avg.						1.94			2.37			1.99		

Table 4-6: The comparison on total cost when C_v=5 *with* illegal edges. "WL" is the wirelength on all layers; "#via" is the number of vias between layers; "Total Cost" is "WL+ Cv *#via"; "Imp." is the improvement on WL, #via, and total cost ($\frac{[15]-Ours}{[15]}$), respectively.

Test Case	Pins	Obs.	N_l	WL			#via			Total cost			CPU time (s)	
				[15]	Ours	Imp. (%)	[15]	Ours	Imp. (%)	[15]	Ours	Imp. (%)	[15]	Ours
Ind1	50	6	4	55,390	55,180	0.38	49	50	-2.04	55,635	55,430	0.37	0.07	0.08
Ind2	200	85	6	11,859	11,589	2.28	208	214	-2.88	12,899	12,659	1.86	3.01	3.28
Ind3	250	13	9	9,983	9,816	1.67	343	341	0.58	11,698	11,521	1.51	3.35	20.92
Ind4	500	100	4	77,033	78,335	-1.69	0	0	0	77,033	78,335	-1.69	8.21	8.55
Ind5	1000	20	4	14,515,511	14,496,361	0.13	0	0	0	14,515,511	14,496,361	0.13	45.71	394.28
Rt1	25	10	10	4,106	3,846	6.33	76	67	11.84	4,486	4,181	6.80	0.06	0.33
Rt2	100	20	10	8,812	8,661	1.71	200	192	4.00	9,812	9,621	1.95	0.90	4.84
Rt3	250	50	10	14,239	13,933	2.15	429	415	3.26	16,384	16,008	2.29	6.84	39.48
Rt4	500	50	10	19,488	18,729	3.89	879	861	2.05	23,883	23,034	3.55	17.56	182.77
Rt5	1000	100	4	25,528	24,908	2.43	814	763	6.27	29,598	28,723	2.96	56.45	279.94
Avg.						1.93			2.31			1.97		

Table 4-7: The comparison on total cost when C_v=3 *without* illegal edges. "WL" is the wirelength on all layers; "#via" is the number of vias between layers; "Total Cost" is "WL+ C_v *#via"; "Imp." is the improvement on WL, #via, and total cost ($\frac{[15]-Ours}{[15]}$), respectively.

Test Case	Pins	Obs.	N_l	WL			#via			Total cost			CPU time (s)	
				[15]	Ours	Imp. (%)	[15]	Ours	Imp. (%)	[15]	Ours	Imp. (%)	[15]	Ours
Ind1	50	6	4	55,390	55,180	0.38	49	50	-2.04	55,537	55,330	0.37	0.07	0.09
Ind2	200	85	6	11,840	11,613	1.91	224	216	3.57	12,512	12,261	2.01	3.05	2.64
Ind3	250	13	9	9,896	9,758	1.39	359	371	-3.34	10,973	10,871	0.93	3.32	18.70
Ind4	500	100	4	77,033	78,335	-1.69	0	0	0	77,033	78,335	-1.69	8.12	8.58
Ind5	1000	20	4	14,515,511	14,496,361	0.13	0	0	0	14,515,511	14,496,361	0.13	45.63	348.20
Rt1	25	10	10	4,106	3,912	4.72	76	67	11.84	4,334	4,113	5.10	0.06	0.31
Rt2	100	20	10	8,789	8,745	0.50	215	197	8.37	9,434	9,336	1.04	0.88	4.67
Rt3	250	50	10	14,099	13,807	2.07	490	460	6.12	15,569	15,187	2.45	6.86	34.25
Rt4	500	50	10	19,280	18,543	3.82	918	903	1.63	22,034	21,252	3.55	17.52	192.78
Rt5	1000	100	4	25,283	24,665	2.44	869	800	7.94	27,890	27,065	2.96	55.61	267.45
Avg.						1.57			3.41			1.68		

Table 4-8: The comparison on total cost when C_v=3 *with* illegal edges. "WL" is the wirelength on all layers; "#via" is the number of vias between layers; "Total Cost" is "WL+ C_v *#via"; "Imp." is the improvement on WL, #via, and total cost ($\frac{[15]-Ours}{[15]}$), respectively.

Test Case	Pins	Obs.	N_l	WL			#via			Total cost			CPU time (s)	
				[15]	Ours	Imp. (%)	[15]	Ours	Imp. (%)	[15]	Ours	Imp. (%)	[15]	Ours
Ind1	50	6	4	55,390	55,180	0.38	49	50	-2.04	55,537	55,330	0.37	0.07	0.09
Ind2	200	85	6	11,840	11,617	1.88	224	217	3.13	12,512	12,268	1.95	3.05	2.66
Ind3	250	13	9	9,896	9,744	1.54	359	364	-1.39	10,973	10,836	1.25	3.32	18.72
Ind4	500	100	4	77,033	78,335	-1.69	0	0	0	77,033	78,335	-1.69	8.12	8.66
Ind5	1000	20	4	14,515,511	14,496,361	0.13	0	0	0	14,515,511	14,496,361	0.13	45.63	350.20
Rt1	25	10	10	4,106	3,912	4.72	76	67	11.84	4,334	4,113	5.10	0.06	0.32
Rt2	100	20	10	8,789	8,745	0.50	215	197	8.37	9,434	9,336	1.04	0.88	4.69
Rt3	250	50	10	14,099	13,794	2.16	490	464	5.31	15,569	15,186	2.46	6.86	34.75
Rt4	500	50	10	19,280	18,464	4.23	918	902	1.74	22,034	21,170	3.92	17.52	199.78
Rt5	1000	100	4	25,283	24,635	2.56	869	802	7.71	27,890	27,041	3.04	55.61	269.45
Avg.						1.64			3.47			1.76		

另外，Table 4-9 和 Table 4-10 為 C_v=5 時比較了第二步驟和第三步驟的 total wirelength 和 CPU time；Table 4-11 和 Table 4-12 為 C_v=3 時比較了第二步驟和第三步驟的 total wirelength 和 CPU time。第五欄是第二步驟形成 MST 之後，我們對所有 edge 作權重的加總，且第六欄為完成第一第二步驟的執行時間。第七第八欄為第三步驟只對所有 edge 作直角化的結果。第九第十欄為第三步驟所有

edge 作直角化與作 U 型線段修正。最右邊一欄代表的是樹有作 U 型線段修正與否，所得到的改善百分比。實驗結果發現對於 C_v=5 的情況而言，我們的 Total cost 平均改善了 2.69%和 2.63%；C_v=3 的情況而言，我們的 Total cost 平均改善了 2.33%和 2.33%。另一方面，加入 illegal edge 的確如我們預期地產生了 Total cost 比較小的 MST，但是在直角化與修正時，會因為障礙物所在位置與疏密的不同而有不同的改善程度，所以在某些 case 下，反而沒有比較好的結果。

Table 4-9: The impact of 3D refinement in ML-OARSMT as C_v=5 *without* illegal edges.

Test cases	Pins	Obs.	N_l	Tree connection		Rectilinearization		Rectilinearization and 3D refinement		
				Total cost	CPU time (s)	Total cost	CPU time (s)	Total cost	CPU time (s)	Imp. (%)
Ind1	50	6	4	56,130	<0.01	56,945	0.05	55,430	0.08	2.66
Ind2	200	85	6	13,414	0.047	12,950	0.59	12,674	3.27	2.13
Ind3	250	13	9	12,006	0.078	11,856	3.19	11,520	20.89	2.83
Ind4	500	100	4	92,330	0.250	78,841	1.70	78,335	8.45	0.64
Ind5	1000	20	4	15,975,855	0.734	14,981,554	20.41	14,496,361	389.28	3.24
Rt1	25	10	10	4,652	<0.01	4,392	0.16	4,181	0.33	4.80
Rt2	100	20	10	10,183	0.016	9,861	1.31	9,616	4.80	2.48
Rt3	250	50	10	17,414	0.078	16,397	6.16	15,948	38.48	2.74
Rt4	500	50	10	25,942	0.250	23,741	15.22	23,095	179.77	2.72
Rt5	1000	100	4	32,467	0.797	29,470	15.61	28,694	276.94	2.63
Avg.										2.69

Table 4-10: The impact of 3D refinement in ML-OARSMT as C_v=5 *with* illegal edges.

Test cases	Pins	Obs.	N_l	Tree connection		Rectilinearization		Rectilinearization and 3D refinement		
				Total cost	CPU time (s)	Total cost	CPU time (s)	Total cost	CPU time (s)	Imp. (%)
Ind1	50	6	4	56,130	<0.01	56,945	0.06	55,430	0.08	2.66
Ind2	200	85	6	13,414	0.056	12,952	0.67	12,659	3.28	2.26
Ind3	250	13	9	11,962	0.089	11,814	3.45	11,521	20.92	2.48
Ind4	500	100	4	87,068	0.269	78,956	1.88	78,355	8.55	0.76
Ind5	1000	20	4	15,974,754	0.864	14,981,554	22.57	14,496,361	394.28	3.24
Rt1	25	10	10	4,652	<0.01	4,392	0.16	4,181	0.33	4.80
Rt2	100	20	10	10,177	0.017	9,855	1.53	9,621	4.84	2.37
Rt3	250	50	10	17,385	0.085	16,385	6.97	16,008	39.48	2.30
Rt4	500	50	10	25,858	0.287	23,722	16.87	23,034	182.77	2.90
Rt5	1000	100	4	32,429	0.813	29,484	18.64	28,723	279.94	2.58
Avg.										2.63

Table 4-11: The impact of 3D refinement in ML-OARSMT as C_v=3 *without* illegal edges.

Test cases	Pins	Obs.	N_l	Tree connection		Rectilinearization		Rectilinearization and 3D refinement		
				Total cost	CPU time (s)	Total cost	CPU time (s)	Total cost	CPU time (s)	Imp. (%)
Ind1	50	6	4	56,002	<0.01	56,839	0.05	55,330	0.09	2.65
Ind2	200	85	6	12,815	0.062	12,488	0.61	12,261	2.64	1.82
Ind3	250	13	9	11,079	0.078	11,086	3.24	10,871	18.70	1.94
Ind4	500	100	4	92,330	0.250	78,841	1.59	78,335	8.58	0.64
Ind5	1000	20	4	15,975,855	0.766	14,981,554	18.94	14,496,361	348.20	3.24
Rt1	25	10	10	4,462	<0.01	4,248	0.16	4,113	0.31	3.18
Rt2	100	20	10	9,632	0.016	9,552	1.41	9,336	4.67	2.26
Rt3	250	50	10	16,090	0.078	15,542	6.47	15,187	34.25	2.28
Rt4	500	50	10	23,548	0.250	21,849	14.83	21,252	192.78	2.73
Rt5	1000	100	4	30,325	0.797	27,770	14.88	27,065	267.45	2.54
Avg.										2.33

Table 4-12: The impact of 3D refinement in ML-OARSMT as C_v=3 *with* illegal edges.

Test cases	Pins	Obs.	N_l	Tree connection		Rectilinearization		Rectilinearization and 3D refinement		
				Total cost	CPU time (s)	Total cost	CPU time (s)	Total cost	CPU time (s)	Imp. (%)
Ind1	50	6	4	56,002	<0.01	56,839	0.06	55,330	0.09	2.65
Ind2	200	85	6	12,815	0.063	12,498	0.65	12,268	2.66	1.84
Ind3	250	13	9	11,061	0.083	11,046	3.33	10,836	18.72	1.90
Ind4	500	100	4	87,068	0.276	78,956	1.71	78,335	8.66	0.79
Ind5	1000	20	4	15,974,754	0.834	14,981,554	21.54	14,496,361	350.20	3.24
Rt1	25	10	10	4,462	<0.01	4,248	0.16	4,113	0.32	3.18
Rt2	100	20	10	9,632	0.017	9,552	1.44	9,336	4.69	2.26
Rt3	250	50	10	16,080	0.088	15,530	6.86	15,186	34.75	2.22
Rt4	500	50	10	23,525	0.270	21,772	15.69	21,170	199.78	2.77
Rt5	1000	100	4	30,317	0.819	27,733	15.82	27,041	269.45	2.50
Avg.										2.33

Chapter 5 Conclusion

5.1 Concluding Remarks

在本篇論文中，我們提出了有效的演算法來解決考慮多層繞線與障礙物迴避的問題，且定義了一個典型流程(typical flow)來比較前人的方法並依此改良。我們的方法如下：第一步驟只對所有接點建立 DT 圖當作初始的 connection graph，此外，我們保留 illegal edge 在 DT 中，使得後續 MST 的建造可以更好；第二步驟對所建立的 connection graph 進行障礙物加權的 MST 的建構；接著第三步驟將樹直角化且利用三維 U 型多餘線段修正總線長。特別的是，我們是對所有可能的立體 U 型線段作完整的分類，此處可以得到最佳解。

實驗的結果發現我們所提出的演算法在多層繞線的 ML-OARSMT 問題上，最好的狀況下，平均改善 1.99%的總線長和 3.47%的 via 數。對於單層繞線的 SL-OARSMT 問題，總線長平均也和目前最好的結果不相上下。此外 illegal edge 的加入的確產生比較好的 MST。

5.2 Future Work

因為在第三步驟所使用的非均勻間隔之繞線格在多層繞線上所佔的記憶體很大，相對來說使用 Dijkstra 演算法所面對的空間太大，使得本論文的執行時間並不理想。後續可以採取局部建立的方式來縮短執行時間。

Bibliography

[1] http://www.tsmc.com/.

[2] M. R. Garey and D. S. Johnson, "The rectilinear Steiner tree problem is NP-complete", in SIAM Journal on Applied Mathematics, pp.826-834, 1997.

[3] Z. Shen, C. C.N. Chu, and Y. Li, "Efficient Rectilinear Steiner Tree Construction with Rectilinear Blockings," in Proc. of International Conference on Computer Design, 2005.

[4] C.-W. Lin, S.-Y. Chen, C.-F. Li, Y.-W. Chang, and C.-L. Yang, "Efficient Obstacle-Avoiding Rectilinear Steiner Tree Construction," in Proc. of International Symposium on Physical Design, 2007.

[5] P.-C. Wu, J.-R Gao, and T.-C. Wang, "A Fast and Stable Algorithm for Obstacle-Avoiding Rectilinear Steiner Minimal Tree Construction," in Proc. of Asia and South Pacific Design Automation Conference, 2007.

[6] 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 λ–Geometry Plane," in Proc. of International Symposium on Physical Design, 2006.

[7] M. Berg, M. Kreveld, M. Overmars, and O. Schwarzkopf, Computational Geometry: Algorithms and Applications, 2nd Edition, Springer-Verlag, 2000.

[8] Y. Hu, T. Jing, X. Hong, Z. Feng, X. Hu, and G. Yan, "An-OARSMan: Obstacle-Avoiding Routing Tree Construction with Good Length Performance," in Proc. of Asia and South Pacific Design Automation Conference, 2005.

[9] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, Introduction to Algorithms, 2nd Edition, 2001.

[10] C.Y. Lee, "An Algorithm for Connections and Its Application," in IRE Transactions on Electronic Computer, pp. 346-365, 1961.

[11] D. W. Hightower, "A Solution to the Line Routing Problem on the Continuous Plane," in Proc. of the 6th Design Automation Workshop, pp.1-24, 1969.

[12] K. Mikami and K. Tabuchi, “A Computer Program for Optimal Routing of Printed Circuit Connectors,” in Proc. of International Federation for Information Processing, pp. 1475-1478, 1968, H47.

[13] Y.-W. Tsai, Y.-T. Chang, J.-C. Chi, and M.-C. Chi. “An Obstacle-Avoiding Rectilinear Steiner Minimal Tree Construction Algorithm,” in Proc. of 18th VLSI/CAD Symposium in Taiwan, Hualien, 2007.

[14] Y. Yang, Q. Zhu, T. Jing, X. Hong, and Y. Wang, “Rectilinear Steiner Minimal Tree among Obstacles,” in Proc. of IEEE International Conference on ASIC, pp. 348-351, 2003.

[15] C.-W. Lin, S.-Y. Chen, K.-C. Hsu, M.-X. Li, and Y.-W. Chang, “Efficient Multi-Layer Obstacle-Avoiding Rectilinear Steiner Tree Construction,” in Proc. of International Conference on Computer Aided Design, 2007.

[16] J. Ganley and J. P. Cohoon, "Routing A Multi-Terminal Critical Net: Steiner Tree Construction in The Presence of Obstacles," in Proc. of IEEE International Symposium on Circuits and Systems, pp. 113-116, 1994.