國　立　交　通　大　學

電子工程學系 電子研究所碩士班

碩　士　論　文

使用規則導向且考慮障礙物之多層直角史坦納樹的建造

Rule Based Multi-Layer Obstacle-Avoiding Rectilinear Steiner Tree Construction

研 究 生：洪禎徽

指導教授：陳宏明 博士

中 華 民 國 九 十 六 年 十 月

使用規則導向且考慮障礙物之多層直角史坦納樹的建造

Rule Based Multi-Layer Obstacle-Avoiding Recitlinear Steiner Tree Construction

研究生：洪禎徽　　　　　Student: Chen-Hui Hung

指導教授：陳宏明 博士　　　　Advisor: Prof. Hung-Ming Chen

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 Requirements
for the Degree of
Master
in
Electronics Engineering
October 2007
Hsinchu, Taiwan, Republic of China

中華民國九十六年十月

使用規則導向且考慮障礙物之多層直角史坦納樹的建造

研究生：洪禎徽　　　　　　　　　　　指導教授：陳宏明 博士

國立交通大學

電子工程學系 電子研究所碩士班

摘要

隨著超大型積體電路設計的大小增加，繞線的問題也越來越重要。繞線樹建立在繞線器的運作中對於最後的繞線結果影響尤其重大。在我們的論文當中，我們提出了一個單/多層考慮障礙物之直角史坦納樹的建造有效且可以快速的建立多層繞線的方法。

我們介紹了一個最小化總繞線長度的方法，可以稱之為共用邊。在同樣的拓墣結構中，這個方法比只有考慮U型修正來最小化總繞線長度好[20]。

在多層繞線部分，我們提出一個多層級近似的方法來處理這個問題，比起一次考慮整個問題的方法，可以大量的縮短運算時間。實驗數據顯示，當測試檔的大小增大時，我們的方法依然有很好的效能。

Rule Based Multi-Layer Obstacle-Avoiding Rectilinear Steiner Tree Construction

Student: Chen-Hui Hung Advisor: Prof. Hung-Ming Chen

Department of Electronics Engineering
& Institute of Electronics
National Chiao Tung University

Abstract

In very/ultra large scale design (VLSI/ULSI), routing is a very challenging work. Especially, the routing tree construction, as an extremely important step for routers, plays a crucial role for the routing results. In this thesis, we have proposed an algorithm to construct a single/multi-layer obstacle-avoiding rectilinear Steiner tree, which can get good solution at single layer and fast yet effective at multi layers. We use a concept called co-edge to minimize the total wirelength. It is better than just considering the U-Shaped refinement [20] under the same topology. In multi-layer, we proposed a hierarchical and heuristic approach to solve this problem. Experimental results have shown that our algorithm is still efficient in larger multi-layer cases, with slightly more wirelength.

誌謝

首先要特別感謝的人，是我的指導教授陳宏明老師，沒有老師的指導與包容，學生是不可能有能力完成這篇論文的。

此外，要感謝的是 VDA LAB 實驗室所有的成員，謝謝他們兩年來的砥勵、幫助及帶給我的歡樂，讓我兩年的生活充滿歡笑及淚水。

另外，在研究的過程中，要感謝清華大學王廷基教授以及團隊所提供的協助，以及台灣大學張耀文教授以及林忠緯同學的協助，讓我能順利完成這篇論文。

家人對我的支持、鼓勵更是我研究路上最大的依靠，對他們的感謝，更是筆墨難以形容。

最後由衷感謝所有我幫助關懷過我的人。

洪禎徽

民國九十六年十月 於新竹

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In very/ultra large scale design (VLSI/ULSI), routing is a very challenging work. To get accurate interconnect information, such as wirelength, congestion, and timing estimations, a good router is necessary. Especially, the routing tree construction, as an extremely important step for routers, plays a crucial role for the routing results. Rectilinear Steiner minimal tree (RSMT) construction is a fundamental problem in VLSI/ULSI design. In fact, today's design often contains rectilinear obstacle, marco cells, IP blocks, and pre-routed nets. Taking obstacles into consideration which is called obstacle - avoiding RSMT (OARSMT) becomes a very practical and complicated problem.

There are many works on OARSMT problem. The researches at single layer have received attention, such as [19] [20] [4] [10] [11]. They can be classified into four major categories according to connection graph construction : (1) the grid based connection graph approach, (2) the Hanan graph based connection graph approach, (3) the Escape graph based connection graph approach, and (4) the spanning graph based connection graph approach.

To route by spanning graph could save a lot of run time[20]. [14] improves the spanning graph to get better solution. But the spanning graph could not extend the relationship between pins and obstacles to multi-layers. [2] pops the nodes of pins and obstacles to other layers for spanning graph construction. It could increase the run time and the complexity by scaling nodes and obstacles.

However, [8] proves the RSMT routing problem is NP-Complete. If we extend the RSMT problem to OARSMT or multi-layers OARSMT problem, the problems are more complex than RSMT. Some approaches extend the problem to timing driven routing problem[13][1]. It would increase the complex of the problem too. Recently, there are some researches considering multi-layer OARSMT. [2] considered multi-layer pins and obstacles problem. But they popped too many of nodes to other layers, the run time would scale a lot in larger cases.

1.1 Contribution

For the OARSMT problem, we have the following distinguished features and theoretical findings:

- We propose some good rules to get the minimal total wirelength. Using the rules to get routing path is better than just using U-shaped refinement at the same topology.[20]
- We propose a very fast yet effective method to form the routing path at multi-layers. When the size of case grows larger, it is still effective.

1.2 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 introduces the previous works and basic terminology definitions first, and then formulates the OARSMT and multi-layer OARSMT problems. Chapter 3 presents our algorithm. Chapter 4 reports the experimental results. Finally, we conclude our work in Chapter 5.

Chapter 2

Preliminary

In this chapter, we introduce some previous works and problem formulation.

2.1 Previous Works

There are many works on obstacles-avoiding rectilinear Steiner minimal tree (OARSMT) problem. They can be classified into four major categories according to connection graph : (1) the grid based connection graph approach, (2) the Hanan graph based connection graph approach, (3) the Escape graph based connection graph approach, and (4) the spanning graph based connection graph approach.

The number of vertices and edges in spanning graph is smaller than others, but it means losing more solution space more than others. There are some approaches without using connection graph. [4][5][16] use lookup table to construct the routing tree, but the approach without considering obstacles. [18] is circuit simulation based obstacle-aware Steiner routing. [3] uses multi-levels approach to handle the problem on multi-layers, but their approach could only use vertical / horizontal at single layer. We can say almost all the approaches use some kind of connection graph, and then do some correction or improvement. We will talk about some approaches recently in this section.

2.1.1 Grid Based Connection Graph Approach

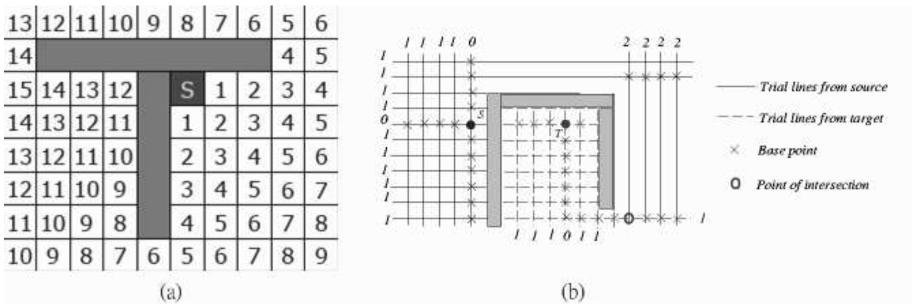


Figure 2.1: (a) The initial routing problem,Maze routing [12] (b) Line search is the variant of Maze routing [15])

These approaches are the ancestor of the routing. Maze routing, first proposed in [12], find a path from a source to a target on a layer by wave propagation as show in Figure 2.1 (a). It can get an optimal solution at two-pins net. However, the time complexity and memory usage grow prohibitively huge as the routing area becomes larger. Further, there are some variants [15][9]. They decide several "escape points" to make the computation more efficient as show in Figure 2.1(b), but they still incur unsuitable solution quality since they only handle the two-pins net.

2.1.2 Hanan Graph and Escape Graph Based Connection Graph Approach

The Hanan graph is similar with Escape graph, we can discuss them together. The Hanan graph is shown in Figure 2.2 (a) and the Escape graph is shown in Figure 2.2 (b). When an edge is blocked by obstacle, the edge could be marked the prohibited routing direction.

Ganley and Cohoon [7] introduces a strongly connected graph called the Escape Graph. They show that at least one optimal solution can be found. They also provid three approximation algorithms, namely G3S, B3S, and G4S, with time complexities $O(k^2n)$, $O(nklogk^2)$, and $O(k^3n^2)$, where k is the sum of terminals and obstacle

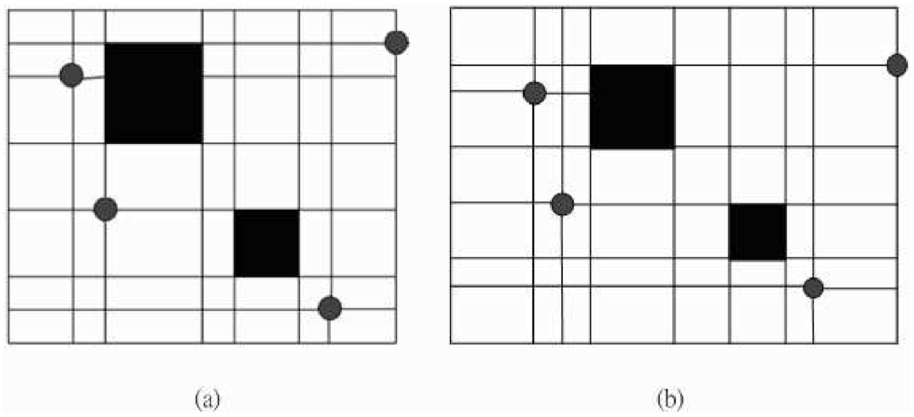


Figure 2.2: (a) Hanan graph consists of the pins and obstacle boundaries and the lines which extended by pins and obstacle boundaries. (b)Escape graph remove the extended edges which are blocked by obstacles.

boundaries and n is the number of intersections over the Escape graph.

Tsai *et al.* [19] presents an algorithm to construct a escape graph , obstacle-weighted minimum spanning tree, and applies Dijkstra's algorithm [6] to get the obstacle-avoiding rectilinear Steiner minimal tree.

2.1.3 Spanning Graph Based Connection Graph Approach

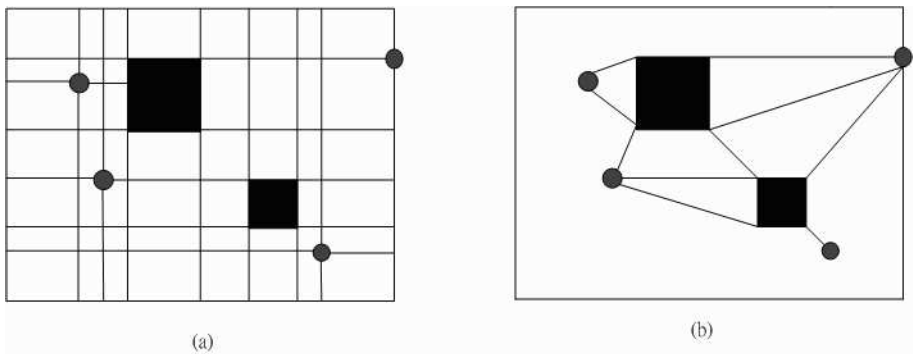


Figure 2.3: (a) Hanan graph consists of the pins and obstacle boundaries and the lines which extended by pins and obstacle boundaries. (b)The spanning graph is an undirected connected graph between the set of pins and the set of obstacles, where no edge intersects with an obstacle.

The spanning graph is an undirected connected graph between the set of pins and the set of obstacles, where no edge intersects with an obstacle, as shown in Figure 2.3. Shen. *et al.* [17] proposed a clever heuristic to construct an obstacle-avoiding rectilinear minimal tree. In this heuristic, the plane is divided into four region is chosen to construct and edge. Based on this method, a single-layer obstacle-avoiding spanning graph is first constructed as shown in Figure 2.3 and Figure 2.4(a). This work [17] is effective in general, but there are some edges are missed which lead to better solutions.

Wu *et al.* [20] presents an approach for OARSMT problem. Their first step, construct a minimal spanning tree for all pins as shown in Figure 2.4 (a) and partition the tree into sub trees by removing edges whose two L-shaped segments both intersct obstacle as shown in Figure 2.4 (b). The second step uses the ant colony optimization to connect the sub trees as shown in Figure 2.4 (c). Their ant colony optimization is performed on the spanning graph in [13] to reduce the runtime. In the last step, the tree constructed in the previous step is transformed into an OARSMT and further improved for its wirelength as shown in Figure 2.4(d).

Recently, Lin et al. [14] proposes an algorithm for an obstacle-avoiding rectilinear Steiner tree construction. It can achieve an optimal solution for any 2-pin net and nets with more pins in many cases. Experimental results have shown that it is very effective and efficient. They construct the spanning graph with ”essential” edges and prove the existence of a rectilinear shortest path between any two-pin net. They present an approach for multi-layer OARSMT problem at ICCAD 2007 [2]. They extend the spanning graph at [14] with some additional VIAS and edges. This approach can construct a good connection graph for searching the solution.

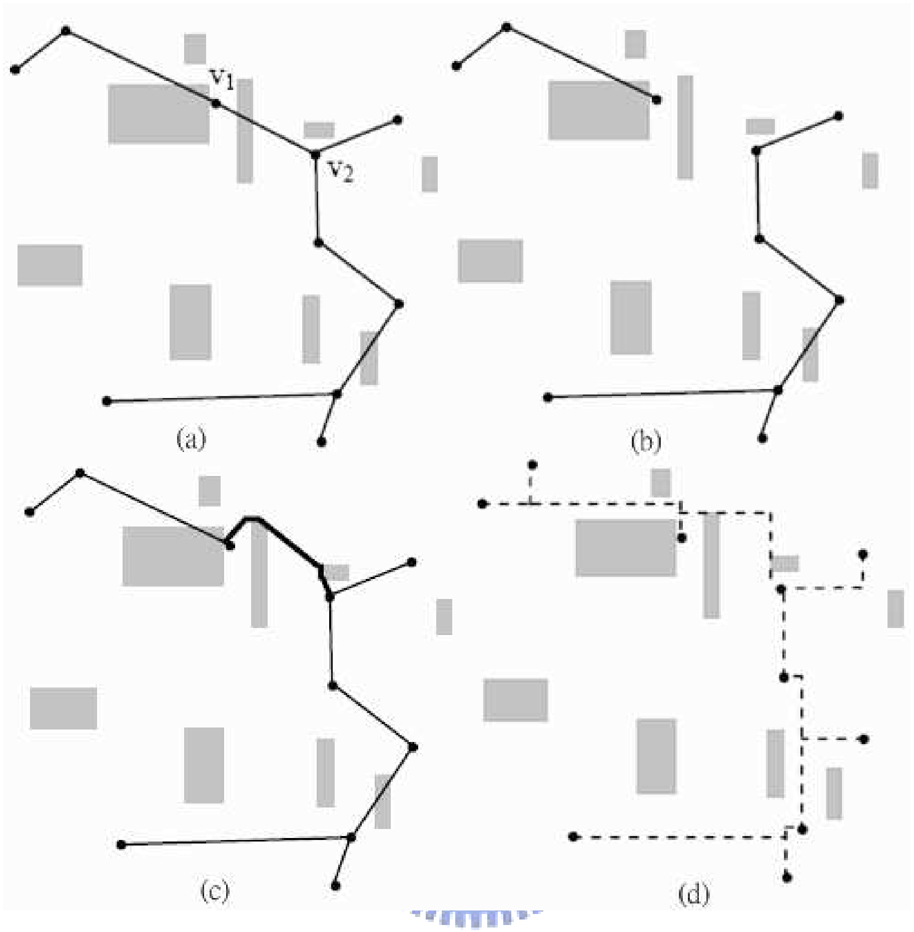


Figure 2.4: (a) A minimal spanning tree for all pins (b) partition the tree into sub trees by removing edges whose two L-shaped segments both intersect obstacle (c) use the ant colony optimization to connect the sub trees (d) transform the tree into an OARSMT.

2.2 Basic Terminology Definitions

An **obstacle** is a rectangle on a layer. No two obstacles overlap with each other, but two obstacles could point-touched at the corner or line-touched at the boundary. A **pin** is a vertex on a layer. A pin must not locate inside any obstacle, but it can be at the corner or on the boundary of an obstacle.

See Figure 2.6 (a) to show any two obstacles cannot overlap each other, but can point-touched or line-touched with each other. See Figure 2.6 (b) to show the illegal situation while the pin are inside the obstacle, but they can be at the corner or on

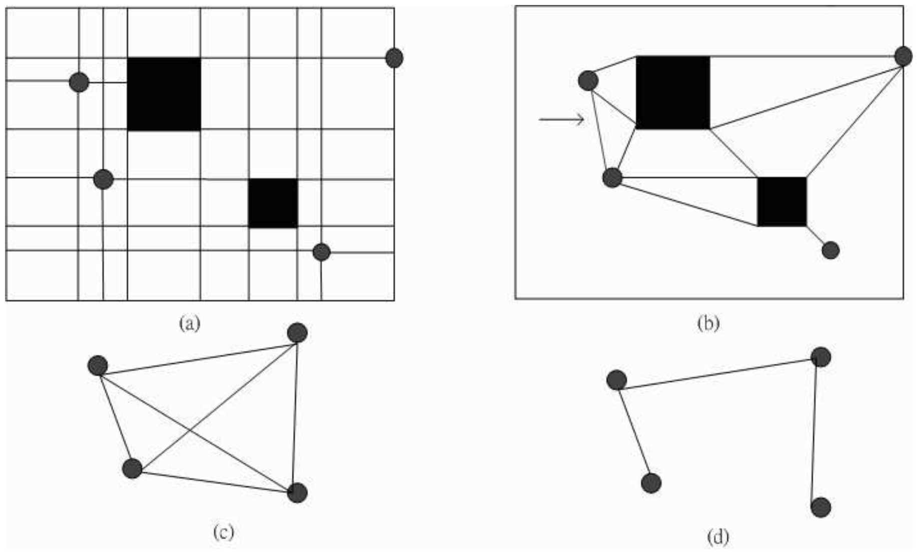


Figure 2.5: (a) The initial routing case (b) First, construct the spanning graph, and improve the graph with additional edges. (c) They construct the complete graph to represent the relationship between pins. (d) They search the minimal spanning tree at the complete graph, and then project the tree back to the spanning graph. Finally, they route the path by the spanning graph and refinement it by U-shaped.

the boundary of an obstacle.

An **via** on layer z is an edge between (x, y, z) and (x, y, z+1). (x, y, z) and (x, y, z+1) must not locate inside any obstacle, but can be at the corner or on the boundary of an obstacle. see Figure 2.7(a), the illegal via is the via locate inside any obstacle , but it can be at the corner or on the boundary of an obstacle(2.7(b)).

The **routable region** is the region without intersecting with any obstacle, but the edge could be point-touched at the corner or line-touched on the boundary of an obstacle. Figure 2.8 shows the tree edges intersecting an obstacle(a), and the tree edges are point-touched and line touch at the obstacle boundary(b).

Co-edge means the possible overlapping wirelength with other connection at one direction of search. **Co-wirelength** means $\sum_{i=1}^{n}(the$ co-edge on connection path). Figure 2.9 is an example of co-edge. When we check co-edge of t1, we can see e2 is the co-edge of the right direction. There is no co-edge at up and down direction ,but there is a path existed. At the left direction of t1, we never do any search. The same as the left direction of t2, e1 is the co-edge.

Simple path means a pair of pins which have a "L" path to connect each other. We call the connection of pins is "simple path "

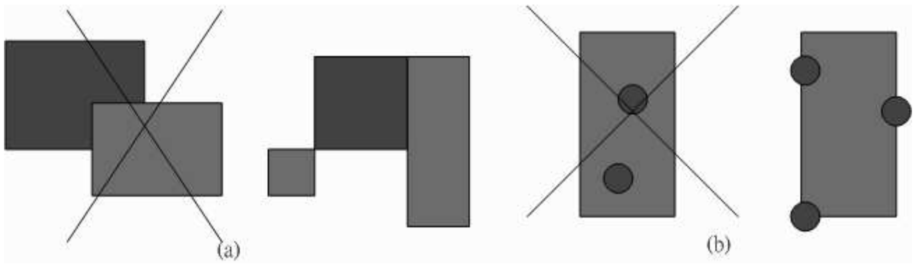


Figure 2.6: (a) Any two obstacle cannot overlap each other (left), but two obstacles can be point-touched at the corner or line-touched at the boundary (right). (b) A pin must not locate inside any obstacle (left), but it can be at the corner or on the boundary of an obstacle (right)

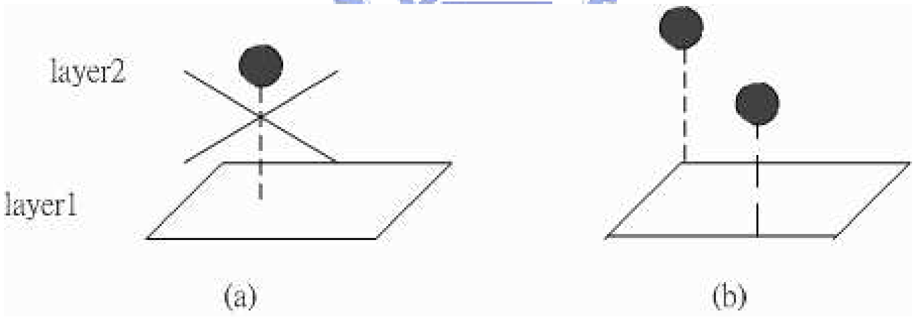


Figure 2.7: (a) A via on layer z is an edge between (x, y, z) and (x, y, z+1). (x, y, z) and (x, y, z+1) must not locate inside any obstacle. (b) It can be at the corner or on the boundary of an obstacle.

Let T=t1, t2,........, tn be the set of pins. Each t includes three information (x, y, z). The x and y represent pin's coordinate and z represents the layer which the pin locates. The set of O=o1, o2........, on are the obstacle set. Every o includes five information, (x1, y1 ,x2 ,y2 ,z), x1 and y1 represent the coordinate of an obstacle's left-down corner and x2 and y2 mean the coordinate of an obstacle's right-up corner. The z are the layer which the obstacle locate.

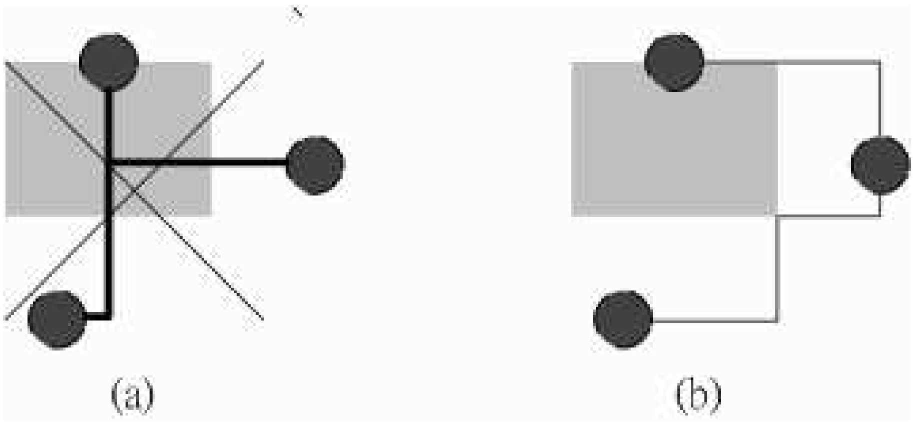


Figure 2.8: (a) the tree edges intersecting an obstacle (b) the tree edges are point-touched and line touch at the obstacle boundary.

2.3 Problem Formulation

We consider the rectilinear routes which use both vertical/horizontal edges in layers and vias between layers. The obstacle-avoiding rectilinear Steiner minimal tree (OARSMT) and multi-layer obstacle-avoiding rectilinear Steiner minimal tree (ML-OARSMT) problems as follows :

Problem 1 : Obstacle-Avoiding Rectilinear Minimal Steiner Tree : Given a set T of pins and a set O of obstacle on single layer. To construct a rectilinear Steiner tree which connects the pins without intersecting the obstacles is called OARSMT and the total wirelength of the tree is minimized.

Problem 2 : Multi-Layer Obstacle-Avoiding Rectilinear Minimal Steiner Tree: Given a set T of pins and a set O of obstacles, constructing a multi-layer rectilinear Steiner tree to connect pins in the set ,possibly through some additional points (called Steiner points), and no edge or via intersects any obstacle in set O. The total cost of the tree is minimized.

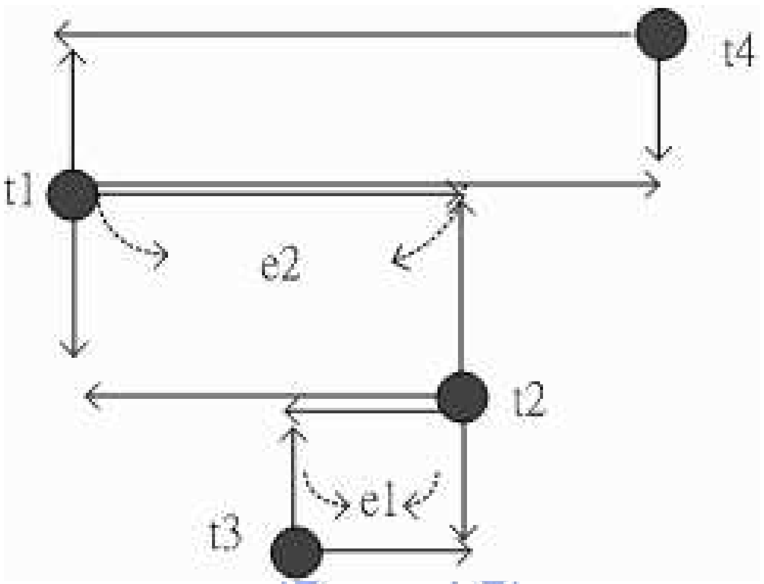


Figure 2.9: The co-edge at the left direction of t2 is e1 and the co-edge at the right direction of t1 is e2.

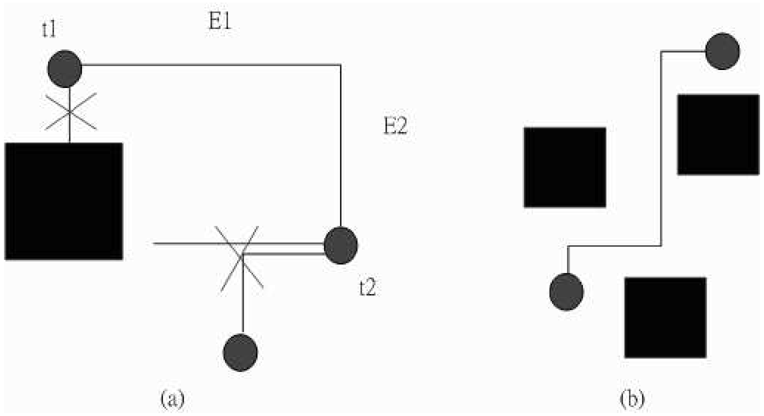


Figure 2.10: (a) The edge E1 and E2 are connected as simple path (b) the path is not a L path, so we called the path not a simple path.

Chapter 3

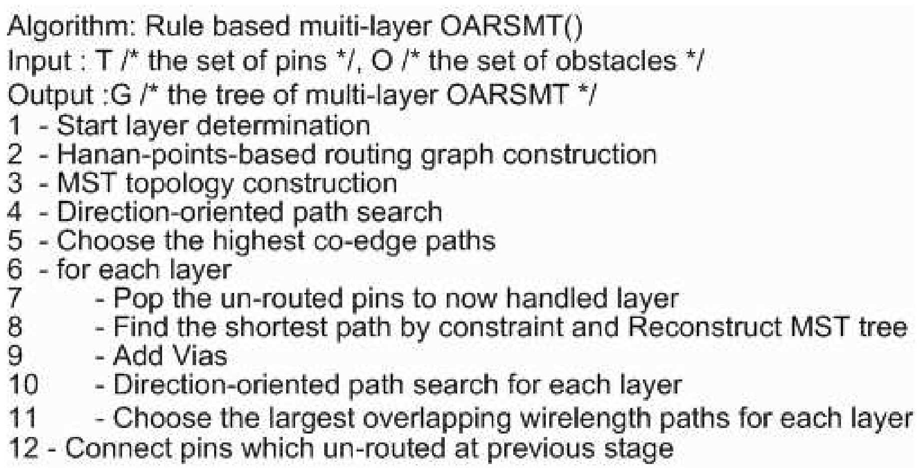
Algorithm

Our algorithm for multi-layer Steiner tree construction consists of the following steps : First, we choose a start layer which has maximal number of pins and minimal number of obstacle, and get a tree consist of the pins connected by simple path with each other at this layer. second, we pop the pins which the simple connection is not existed to next layer. Then, we repeat the work of previous step with some adding methods over next layer for finding shortest path. Third, some pins which can not pop to other layer or still not find the path would be considered in this step. This step will use spanning graph to get the routing path. If there is a path existed, it can be found in this step. The overall algorithm is shown in Figure 3.1.

When we handle single layer problem, we can modify the algorithm of multi-layers. The modified algorithm is shown in Figure 3.2. We remove the step 2, and then run step 3 before selecting highest co-wirelengths path. When we run the step first, we can increase the solution space at selecting highest co-wirelengths.

3.1 Starting Layer Determination

In this section, we construct a tree consist of the pins connected by simple path with each other at single layer. We choose a start layer by a rule R : (the number of pins at the layer) - (the number of blocks at the layer).



```
Algorithm: Rule based muiti-layer OARSMT()
Input : T /* the set of pins */, O /* the set of obstacles */
Output :G /* the tree of multi-layer OARSMT */
1  - Start layer determination
2  - Hanan-points-based routing graph construction
3  - MST topology construction
4  - Direction-oriented path search
5  - Choose the highest co-edge paths
6  - for each layer
7         - Pop the un-routed pins to now handled layer
8         - Find the shortest path by constraint and Reconstruct MST tree
9         - Add Vias
10        - Direction-oriented path search for each layer
11        - Choose the largest overlapping wirelength paths for each layer
12 - Connect pins which un-routed at previous stage
```

Figure 3.1: The algorithm of multi-layers. There are three stages for this algorithm. First, we select and handle a START layer, and then extend the solution to other layer. Finally, we handle the un-routed pin-pairs.

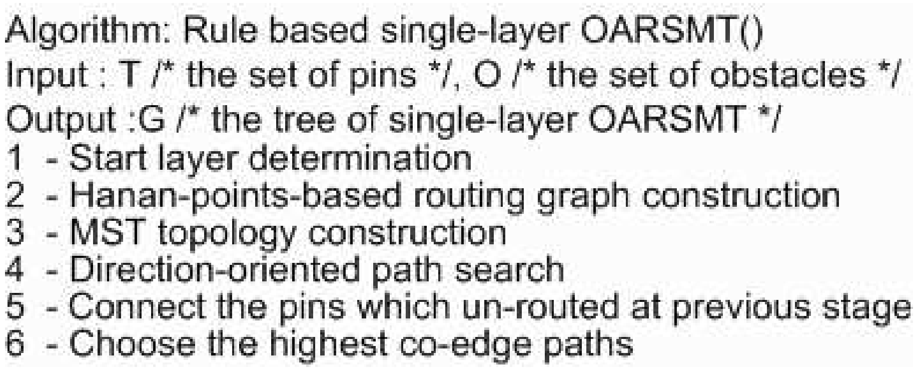
We choose the layer which has maximal R to be the start layer. The reason is when we consider the other layers, we will use completed layer to check the better solution existed or not. If choosing the layer having maximal R ,we usually have more chance to get better solution.

3.1.1 Hanan-Points-Based Routing Graph Construction

After choosing start layer, we construct Hanan points graph first. The Hanan points graph is based on Hanan graph, but the Hanan points graph only consists of vertices. The reason is the cost of constructing Hanan points graph is smaller than Hanan graph, and we don't need some information (like edges) in Hanan graph.

When we read the data of pins and obstacles, we will dynamic construct x and y coordinates table. The table of y coordinates is a reference table which only contains y coordinates. The table of x coordinates is the table for the search of path which contains vertices had constructed.

Figure 3.3 (a) shows the Hanan graph and its data structure. In (b), Y table means the y reference table. In x table, each x coordinates consist by list of y



```
Algorithm: Rule based single-layer OARSMT()
Input : T /* the set of pins */, O /* the set of obstacles */
Output :G /* the tree of single-layer OARSMT */
1  - Start layer determination
2  - Hanan-points-based routing graph construction
3  - MST topology construction
4  - Direction-oriented path search
5  - Connect the pins which un-routed at previous stage
6  - Choose the highest co-edge paths
```

Figure 3.2: When we handle single layer problem, we can modify the algorithm of multi-layers. We remove the step 2, and run step 3 before selecting highest co-wirelengths path.

coordinates. We construct the pins and obstacle vertices first. The other vertices will be constructed while we visit them on search step. If we construct the vertices only when we need them, we can save the run time. The reason is that the most vertices are not visited on searching step.

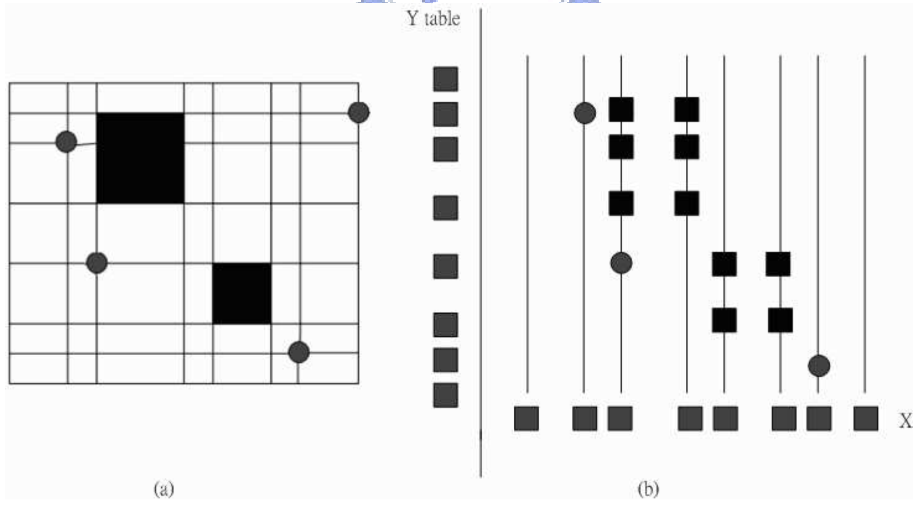


Figure 3.3: (a) Hanan graph (b) The data structure is constructed by (a).

3.1.2 Minimum Spanning Tree Topology Construction

We apply Kruskal's MST algorithm for finding MST topology with the distance constrain. When we construct minimum spanning tree (MST) ,the distance between pins is as follows : **Distance :** Manhattan distance + obstacle penalty. **Manhattan distance :** $|x_1$-$x_2|$ + $|y_1$-$y_2|$. **obstacle penalty :** min (the length of obstacle boundary edge).

In general, the minimum spanning tree uses the Manhattan distance between two-pin net as cost. However, it will not be accurate in estimating the real routing distance with the existence of obstacles. In order to have a better estimation of real routing distance, we use the obstacle penalty to construct a better topology. The method is used in [19][20] too.

First, we check any simple path existed or not in a two-pin net. There are at most two L-shaped segment in a two-pin net. If all of them intersect the obstacle, as show in Figure 3.4 (a), we say no simple path existed. Then, we assign a new cost to the path. We draw a rectangle whose boundaries pass through the two-pin net, and two non-intersecting boundaries of the obstacle, as show in Figure 3.4 (b). Then, we have two different path now (Edge 1 and Edge 2). We will choose the path which length is smaller than another one. To considering obstacle penalty means the new length larger than Manhattan distance.

We use an example in Figure 3.5 to illustrate the difference between the general MST and obstacle penalty MST. (a) is the MST routing result whose topology got by general distance, and then we can easily dig out the topology is bad than (b). The reason is the distance of (a) not consider the length from obstacle, and then the topology is impertinent.

3.1.3 Direction-Oriented Path Search

On this step, we divide MST topology net into a set of two-pin nets. The direction-oriented path search will work on these two pin nets. We handle a two-pin nets one time and pick up every pin in a net to do two direction search. The direction of one

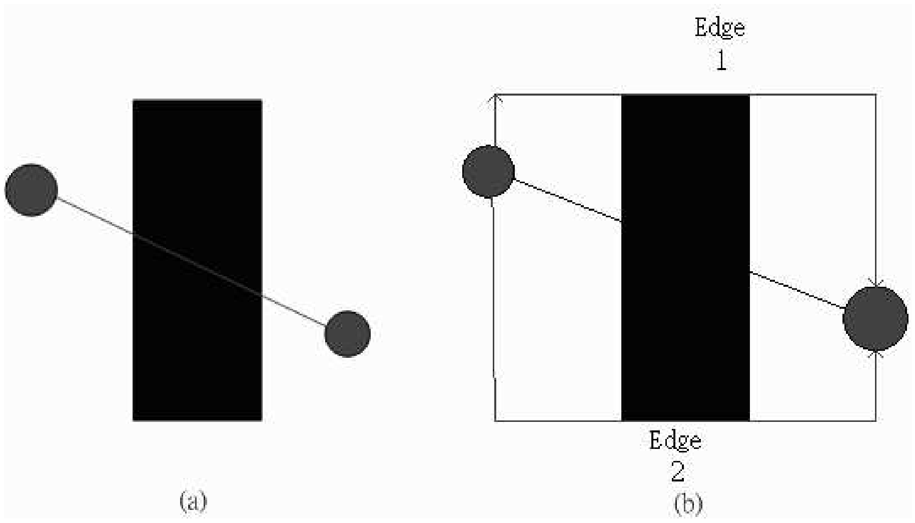


Figure 3.4: 16 (a) There are at most two L-shaped segment in a two-pin net. If all of them intersect the obstacle, we say no simple path existed. (b) a rectangle whose boundaries pass through the two-pin net, and two non-intersecting boundaries of the obstacle. Then we have two different path now.

search depends on the two-pin net topology.

We use Figure 3.6 to illustrate the direction-oriented path search. There are three two-pin net in this example and some searches cover with each other. (a) shows the MST topology of an example. (b) is the Direction-Oriented Path Search of the MST topology. We consider a two-pint net, t1-t3, the search direction of t1 is right and down and the direction of t3 is up and left. The search direction of a pin depends on the two-pin net topology.

In addition, when we consider the two-pin net ,t1-t2 , the right direction of t1 is blocked by an obstacle. It means the search of this direction is false. We can easily see that the searches of pins in a two-pin net meet at most twice in search region. When there is no touched point in a two-pin net search, we call no simple path between pins.

When we do the direction-oriented path search, we not only check the path existed or not, but also recode some information for path selection. The information

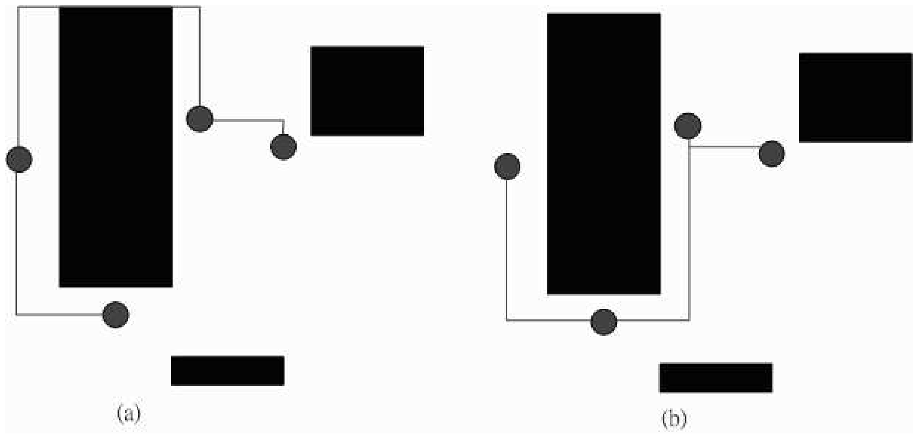


Figure 3.5: (a) The routing result whose MST topology got by general distance, (b) The routing result whose MST topology got by obstacle penalty . The total wirelength of (b) is smaller than (a).

are listed as follows :

1. The end position at every succeeded search.
2. The total search times of pin's at each direction.
3. The failed search times of pin's at each direction.
4. When any direction-oriented path search of two-pin net is failed in one direction, but the corresponding search is successful in the net. We will remark the successful search and the end position of this search.

We explain (4) with Figure 3.6. In the two-pin net, " t1-t2 ", t1's right direction search is failed, but the t2's opposite search ,down direction , is successful. We will special remark this kind of situation.

3.1.4 Path Selection

In general, if a routing path contains more co-wirelength, it means the total wirelength is smaller. The purpose of all of our rules is to get as more co-wirelength

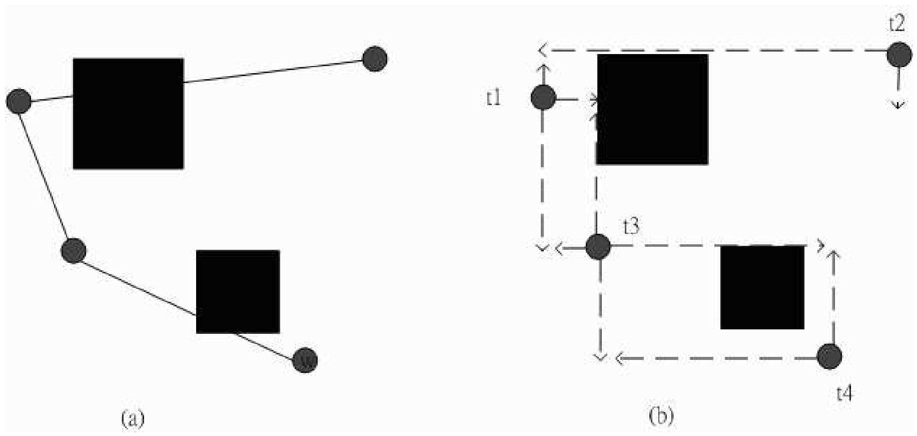


Figure 3.6: (a) the MST topology of an example (b) the Direction-Oriented Path Search on the MST topology .

as possible. Here, we define a kind of edge, co-edge, and co-wirelength as follows :

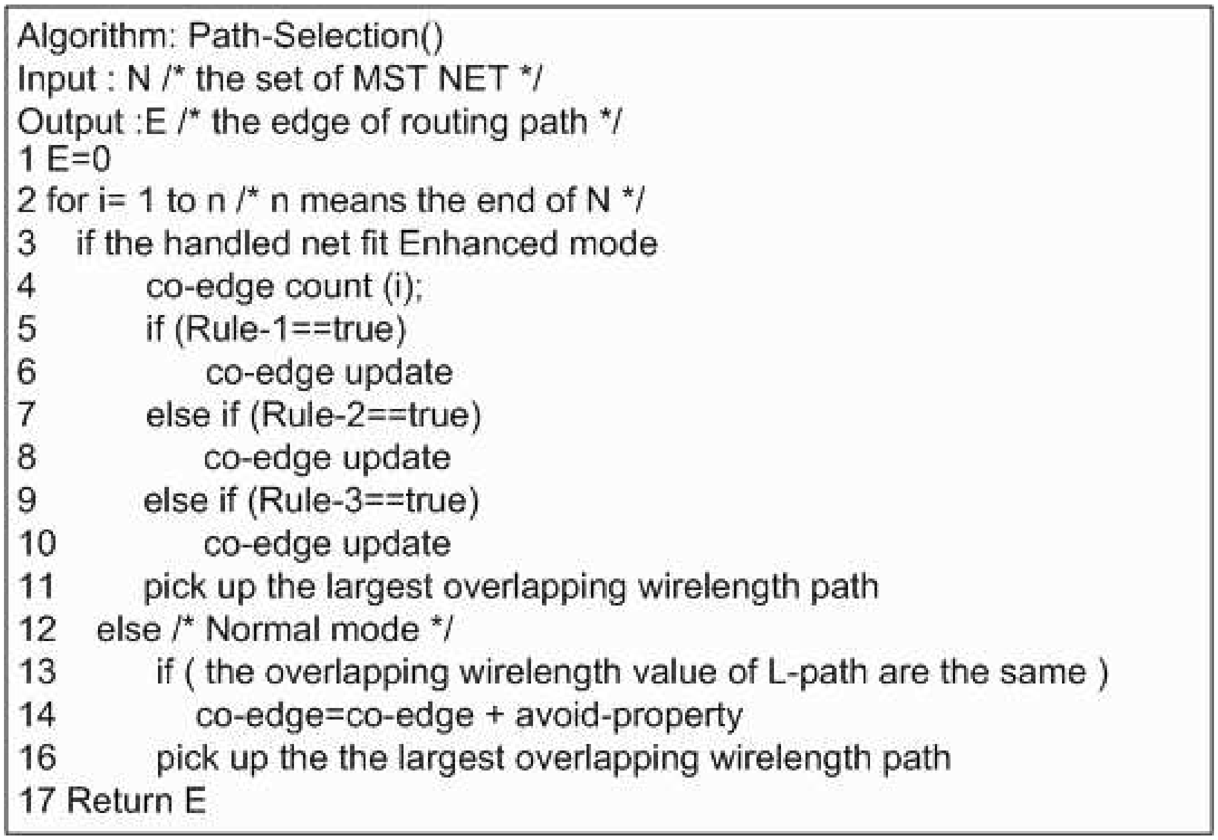
Co-edge : the possible co-wirelength with other connection at one direction search edge **Co-wirelength :** $\sum_{i=1}^{n}(the$ co-edge on connection path)

Figure 2.9 is an example of co-edge. When we check co-edge of t1, we can see e2 is the co-edge of the right direction. There is no co-edge at up and down direction of t1, but there is a path existed. At the left direction of t1, we never do any search. The same as the left direction of t2, e1 is the co-edge.

We say the possible co-wirelength is the total co-edge on the connection path. This is the reason why we always select the path which contains highest total co-edge. Then, how to estimate the co-edge accurately is the problem we should resolve. The solution we proposed will be described at the rest of this section.

We propose a way about rule based path selection. Here, we use the information from direction-oriented path search to give a better path selection. After observing routing example we think all two-pins nets can be divide into two kinds of situation. The different situation will be handled by difference modes, **Enhanced mode** and **Normal mode**. The Enhanced mode means we should consider additional routing path case (it will form Z-shape), because the additional solution might be better than L-shape. The Normal mode means there is no chance to improve the solution in Z-shape , and then we will add a property called " Avoid property " to help making a decision at L-shape.

On this step, we still base on MST topology, and we only handle a two-pins net one time. When we select a two-pins net for path selection, we will decide which mode of this net should be. Then, we determine the routing path between these two-pin nets by the rules. The algorithm of path selection is shown in Figure 3.7.



```
Algorithm: Path-Selection()
Input : N /* the set of MST NET */
Output :E /* the edge of routing path */
1 E=0
2 for i= 1 to n /* n means the end of N */
3    if the handled net fit Enhanced mode
4          co-edge count (i);
5          if (Rule-1==true)
6               co-edge update
7          else if (Rule-2==true)
8               co-edge update
9          else if (Rule-3==true)
10              co-edge update
11         pick up the largest overlapping wirelength path
12   else /* Normal mode */
13         if ( the overlapping wirelength value of L-path are the same )
14             co-edge=co-edge + avoid-property
16         pick up the the largest overlapping wirelength path
17 Return E
```

Figure 3.7: The algorithm of path selection. We divided the path into two kinds of mode, and then selecting the path which had highest total number of co-edges

First, we show how to select which kind of mode the two-pins net belonged to. When any search direction of two-pins net fit the condition shown in Figure 3.8, the net will be led into **Enhanced mode**.

The two conditions are shown in Figure 3.8 (a) and (b). In Figure 3.8, the two-pins net to be handled are " t1-t2 " in (a) and " t3-t4 " in (b). (a)shows one direction of the pin, t1, is more than once and longest at handled two-pins net (like e1). (b)shows one direction of the pin, t4, is a failed search, but it is succeed in another two-pins net (like e2).

It is easy to show the reason why we only consider the Z-shape case when the net fits the Enhanced condition. If the net does not fit the condition, it means every direction of pins must be one of the situation: no path, only be searched one time, or not the longest path. As shown in Figure 3.11 (b), when we consider t1-t2 net, the down direction of t2 is not a longest path at this direction. IF we want to get maximal co-edge, the meaning is to use whole e2 to be the path. At this kind of case we only consider the L-shape path.

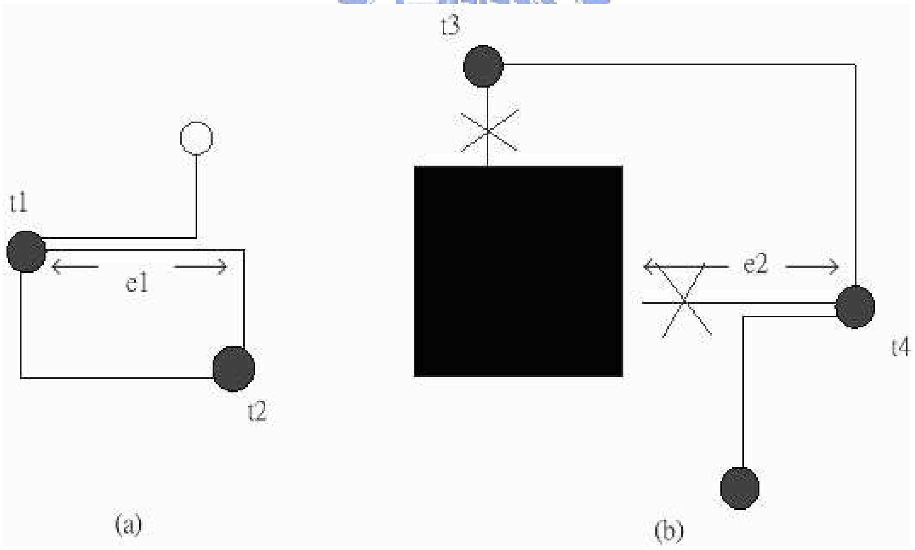


Figure 3.8: The conditions of enhanced mode: (a) one of the pin's direction search is more than once and longest at handled two-pins net (like e1) (b) one of the pin's direction is a failed search, but it is succeed in another two-pins net. (like e2).

- **Enhanced Mode :**

The difference of Enhanced mode and normal mode are not only the additional routing case, but also some additional co-edge estimation mode. If there is no co-edge and path existed at one of any pin's direction, we will check any hidden co-edge existed or not. We give an example of hidden co-edge at Figure 3.9.

In Figure 3.9, (a) is the direction-oriented path search. When we consider the two-pins net, t1-t3, shown in (b). There is only one path existed, if we just consider L-shape and the all of the co-edge is 0. In fact, there are 2 hidden co-edge existed. One is t1's right direction. Although, the t1's right direction is failed in t1-t3 net, the t1's right direction is successful in t1-t2 net. So, there is hidden co-edge, e1, at t1's right direction. The same as e2 is an hidden co-edge. (c) is the net of t1-t2. When we consider this net, there are two hidden co-edge too.

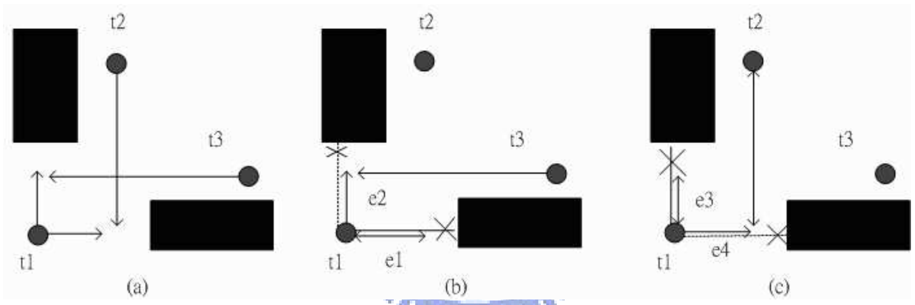


Figure 3.9: (a) the direction-oriented path search (b) the t1-t3 net have two hidden co-edge e1 and e2. (c) the t1-t2 net has two hidden co-edge e3 and e4. When we consider the hidden co-edge, we can have more solution space.

We list the hidden co-edge rules at table 3.1 If the co-edge is not existed, we will use the rule to check hidden co-edge. We show the hidden co-edge checking flow at Figure 3.10.

The Rule-1 is the same as e1 in Figure 3.9 (b) and e4 in (c). The search at the direction is successful in handling net, but is failed in other net.

The Rule-2 means the information of search time getting form direction- oriented path search. When any two-pin net's direction-oriented path search is failed in one direction, but the opposite search is successful in the net. We will remark the successful search and the end position of this search. At this step, we measure the hidden co-edge by these information. The Rule-2 is shown in Figure 3.11(a). In Figure 3.11 (a), there is no co-edge at t1's right direction, but we can easily to see that there is one co-edge at the t2's down direction. The e1 in 3.11(a) not fits Rule-1, because the down direction search of t2 is failed in handling net.

The Rule-3 means the total numbers of search time more than one time. The priority of these rules means the chance of co-edge really existed at these three kinds of condition. We discover the results of using these priority on experiment are better than others. As shown in Figure 3.9 (b) and (c), e2 and e3 fit Rule-3. There is no successful search at that direction, but the times of search are more than once.

Table 3.1: hidden co-edge rules. The success-fail search time means there is at least one of the search in other two-pin net is failed in this direction, but the the corresponding search direction is successful in the net.

	Priority	Formula
Rule-1	High	(Failed search time * length)
Rule-2	Med.	(Success-Fail search time * length)
Rule-3	Low	((All search time -1)* length)

- **Normal Mode :**

At this mode, we don't need to consider hidden co-edge, but we should consider the net which is the neighbor of handled net. When the total co-edge at two L-shape are the same, we add a new parameter called " avoid property ". If the avoid property of one L-shape is higher than another one. We say the L-shape is not a good choice.

We define the property should be avoided and avoid direction as follows : **Property :** the co-edge of avoid direction. **Avoid direction :** the direction which will increase total wirelength.

The property is the co-edge of avoid direction. The avoid direction means the

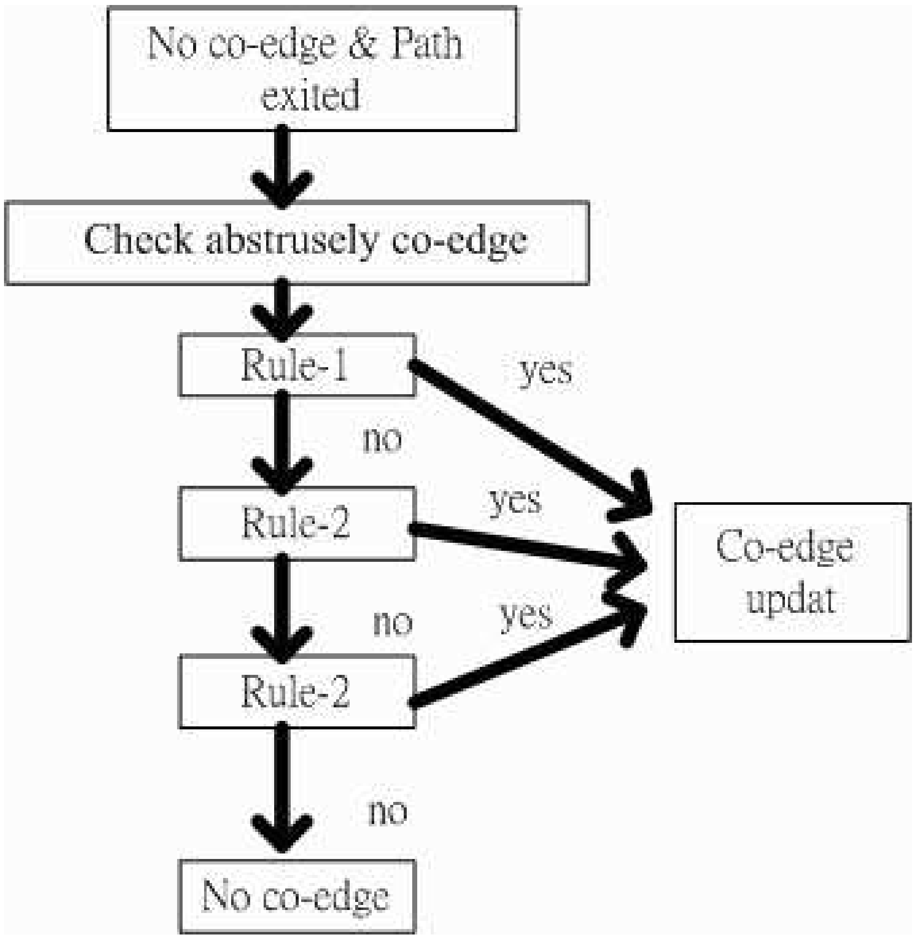


Figure 3.10: When we determine the value of hidden co-edge in enhanced mode, our algorithm will run as this flow.

direction avoid now direction. Figure 3.12 (a) shows the avoid direction. The handling net is t1-t2 and the co-edge value of that net is the same at up L-shape and down L-shape. Let's us check the property of down L-shape. The avoid direction of t1's right direction is down direction. The reason is at t1-t3 net, if we select t1's right direction to be the path, t1's down direction is not be selected. All the other avoid directions are selected by the same way. In this example, the total number of property at down L-shape is E1, and then the total number of property at up L-shape is 0. We should select up L-shape at t1-t2 net. (b) is the final result at this case by our algorithm.

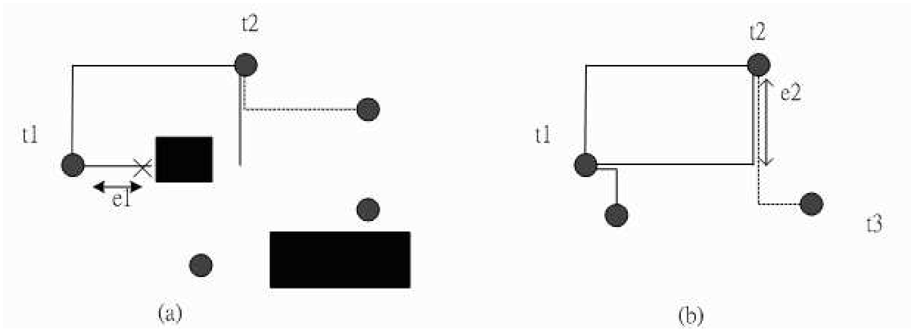


Figure 3.11: (a) The right direction of the pin,t1 , is a failed search, but it is successful in another two-pins net. (e1)(b) The down direction search of the pin, t2, is more than once and not longest at handled two-pins net. (e2)

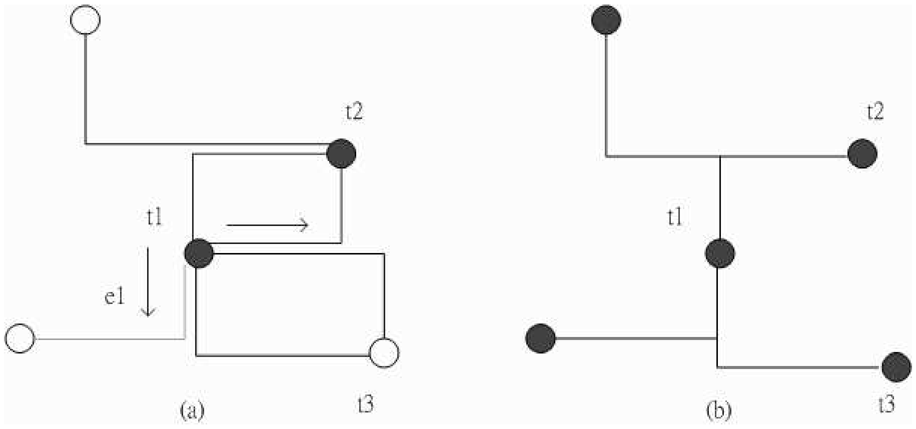


Figure 3.12: (a)an example of avoid direction. When we consider the two-pin net t1-t2, t1's down direction is the avoid direction of t1's right direction, and E1 is the co-edge of down direction. The property of t1's right direction is E1. (b) The result of considering the property.

- **Path Selection :**

After counting the co-edge in each direction of the two-pins net, we will pick up any two of them which have maximal overlapping wirelength. When the net lead in Enhanced mode, we should especially notice that the total co-edge of Z-shape be bounded by the length of one direction.

When we consider the Z-shape, we should notice that the additional path is available or not in the Z-shape. IF the path is not available, we will drop the solution and choose other solution. In addition, our approach can apply to preferred direction problem too. The reason is that we consider one direction one time in our search and path selection step.

To see Figure 3.13, in (a), the total co-edge of up L-shape is e1+e2. In (b), the total co-edge of the Z-shape is e3+e4, but be bounded by the length of x direction. The total co-edge is | x_t3 - x_t4 |.

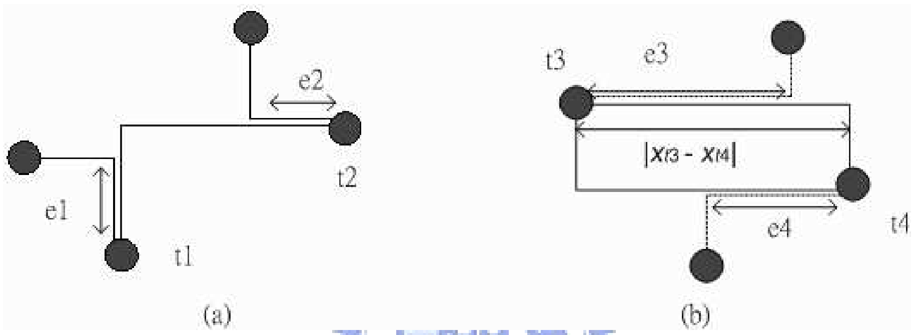


Figure 3.13: (a) In the up L-shape,the total co-edge is e1+e2 (b) In the Z-shape,the total co-edge is bounded. The total co-edge of this example is | x_{t3} - x_{t4} |.

3.2 Solution Propagation to Other Layers

After graph construction, we have handled the two-pins nets which contain simple path. We propagate the pins which not contain simple path. If we propagate the pins which there is no simple path existed between them to other layers, it may be better than finding the solution in that layer. The reason is when the simple path is blocked by obstacle, the total wirelength may be worse than we propagate to other layers.

3.2.1 Search for the Shortest Path

First, we propagate the uncompleted pins at last layer to next layer. Then, we complete Hanan points graph construction and MST topology construction with these pins.

Before doing direction-oriented path search at the MST topology, we try to search the shortest path with completed layers. We use Figure 3.14 to show the idea. In Figure 3.14, there are three pins at last layer and two-pin at now layer. IF we connect the pins of now layer with VIA to last layer, we can get smaller wirelength.

We propose a constrain to control the number of VIAs. When we cross a Via, we add the user defined cost C to the wirelength. If the wirelength at layer N is W and the wirelength at layer M is Z, layer N and M are connected by one via crossed from N to M. The total wirelength should be W+Z+|N-M|.

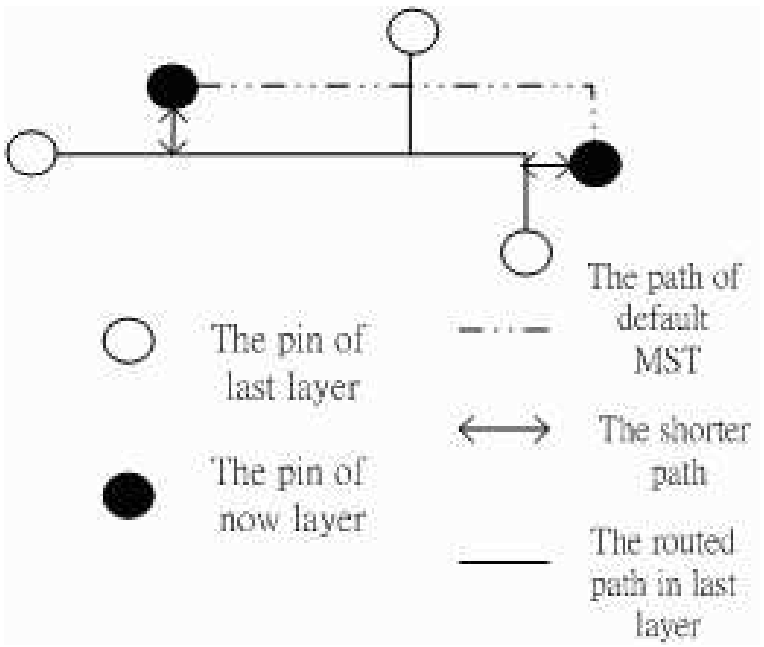


Figure 3.14: When we connect the pins of now layer with VIA to last layer, we can get smaller wirelength without violating constrain. We can get best better solution with adding these Vias.

3.2.2 MST Reconstruction

After finding the shortest path, the MST topology should reconstruct. We use an example to show the reason. Figure 3.15 (a) is the default MST topology. After finding the shortest path, the result is (b). The new topology is not a MST anymore. We must reconstruct the MST for the direction-oriented path search step. (c) is the new tree after reconstruction.

To see Figure 3.15 (c), we do not reconnect the path between t1 and t2 for no cycled. For no cycled and the keeping the connection of net, We will show that at next sub-section.

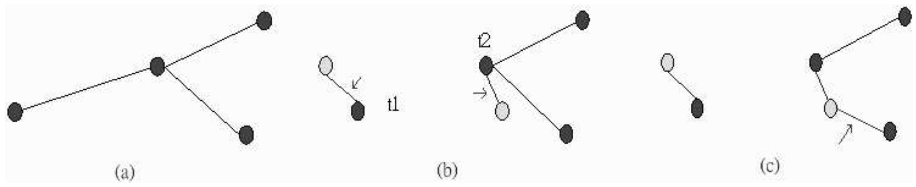


Figure 3.15: (a) the default MST topology (b) After finding the shortest path, we got some new paths and broke some paths. The MST should be reconstructed to keep the property of MST (c) After reconstructing MST, the net should be still no cycled and connect with each other (through Vias or lines).

3.2.3 The Connection Relationship of Net between Layers

As shown in Figure 3.16, when we break the connection between nets at n layer, we add an via to the other net at n+1 layer. At Figure 3.16, all of the sub-net are connected and there is no cycle in the net. The figure shows two information for us. First is the pins of uncompleted MST, second is about finding of shortest path in multi-layers.

For no cycled and the keeping the connection of net, we define some constrain to simplify the problem and guarantee the connections.

1. If all of the pins in one un-routed two-pin net are popped successfully, we will pop two of them, or else we not consider them until the step which is described at next section.

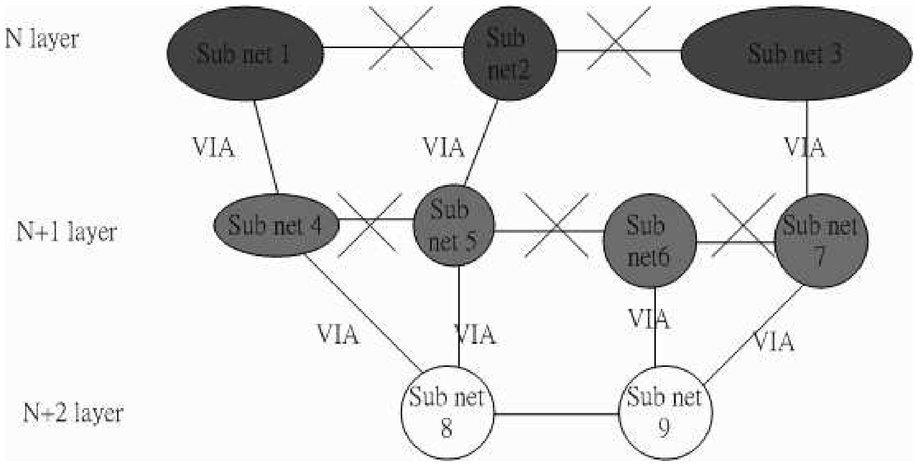


Figure 3.16: The connection relationship of net between layers. We can connect the sub nets through the other layers sub nets. (connect sub net 4 and 5 through sub net 8)

2. After popping pins from last layer, the pins are the nodes at this layer considering in MST step. Then, we should keep the connection of the popped two-pin pair.(The MST between them should be broke at searching shortest path.)

3.3 Connect the Pins Which Are Not Connected at Previous Steps

At this section, we will complete all routing path which not be connected at previous stage.

3.3.1 Sub-Net Construction

First, we list all uncompleted pins (popped falsely or still un-routed pins), and then find all pins which had connected to every uncompleted pin. For example, there are 4 pins, t1,t2 ,t3 and t4. The uncompleted pins are t3 and t4. The nets which had be completed are t1-t2 and t2-t3. We can say t1, t2 ,t3 are connected with each other. We group t1, t2 , t3 to the same group, and t4 is another group with only one pin. As shown in Figure 3.17.

This is because we need these groups and pins to make decision what the start pin/net and end pin/net should be, when we find pins connection.

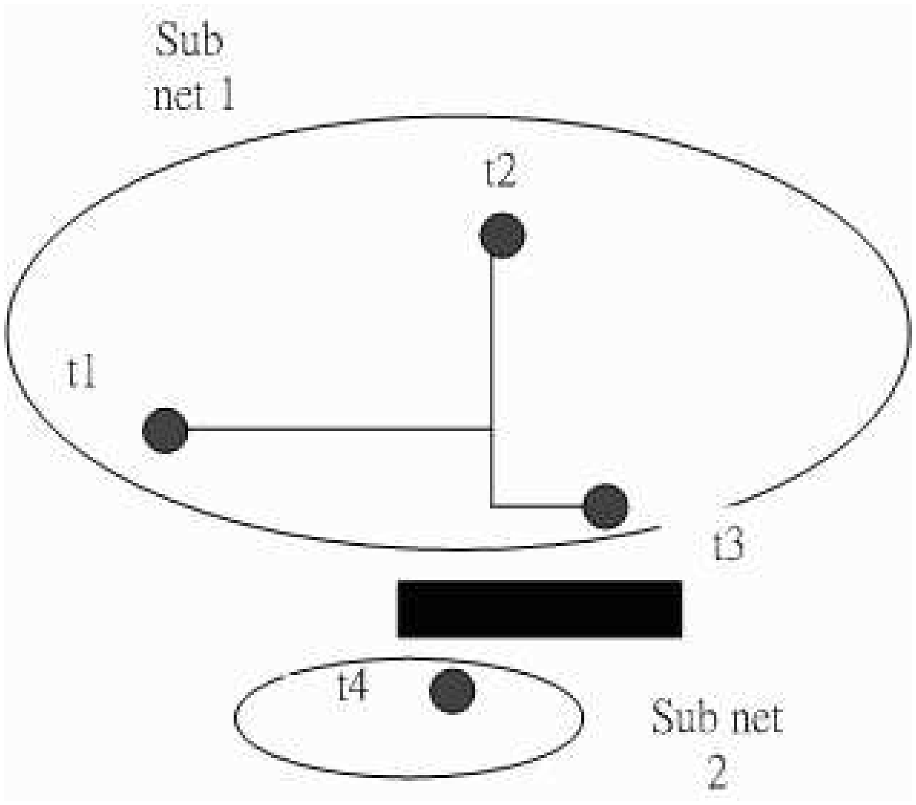


Figure 3.17: There are 4 pins, t1,t2 ,t3 and t4. The uncompleted pins are t3 and t4. The nets which had be completed are t1-t2 and t2-t3. We can say t1, t2 ,t3 are connected with each other. We group t1, t2 , t3 to the same group, and t4 is another group with only one pin.

3.3.2 Spanning Graph Construction

In this step, we construct an obstacle-avoiding spanning graph which is defined as follows :

- **Obstacle-avoiding spanning graph :** The graph is an undirected connected graph between the set of pins and the set of obstacles, where no edge intersects with an obstacle.

We implement the algorithm proposed in [17] to construct the spanning graph. The algorithm use an efficient algorithm to construct a spanning graph. The algorithm holds two dynamic lists to decrease the solution space. Every obstacle can be divided into 8 search region and every pin can be divided into 4 search region, as show as Figure 3.18.

The algorithm is listed at Figure 3.19. For every pins and obstacle corner v, we only connect at most one visible point v‘ in each neighboring search region, where v‘ is the closest point to v in the corresponding neighboring search region. In algorithm, all points are sorted by coordinates in non-decreasing order, and then keep an active set A of v such that all points in A are visible to v. The detail of this algorithm can be found in [17].

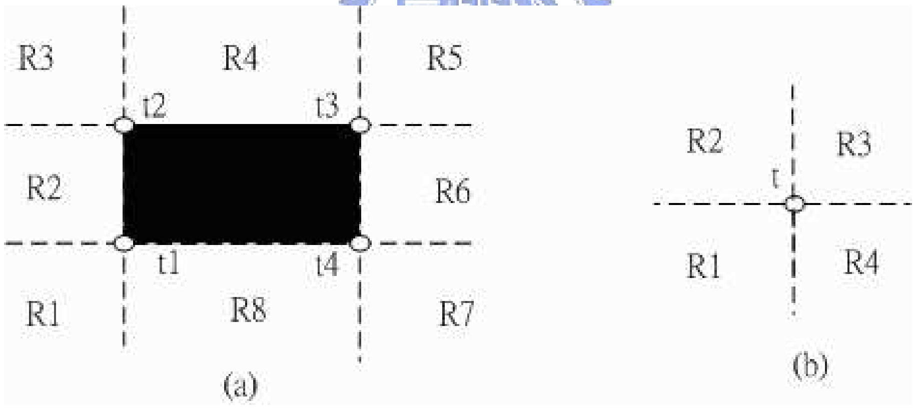
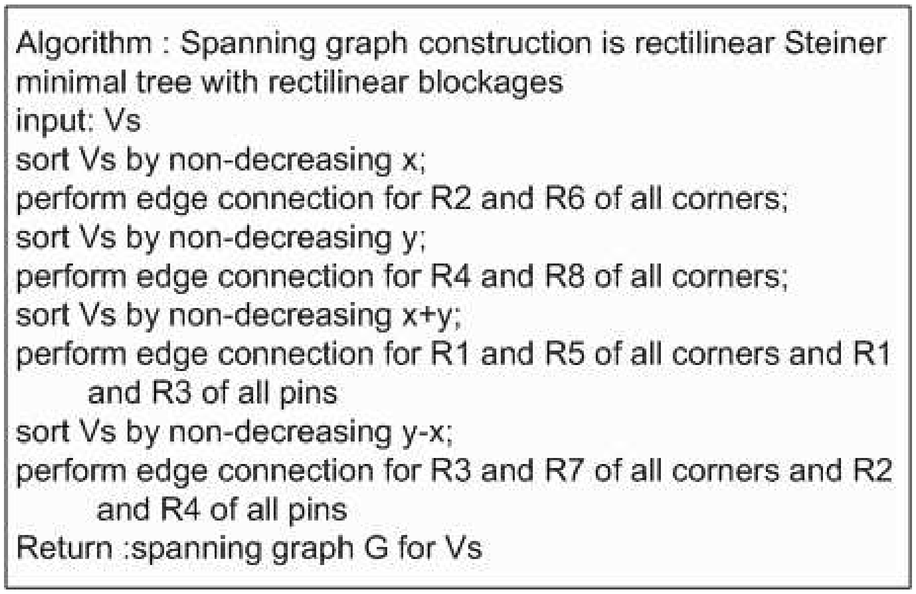


Figure 3.18: (a) The search region of the obstacle. There are eight regions of one obstacle. (b) The search region of pin. There are four regions of one pin.

3.3.3 Shortest Path Search on Spanning Graph

At this step, we pick up one uncompleted pin one time, and we know the target point in the MST topology too. Then, we find out the nets which these two-pin pair belong to. We can pick up one of the pins to be the START point, and the other one is the END point; the sub net of START point which we got at previous step is START net, and the sub net of END point is the END net. When the search meet



```
Algorithm : Spanning graph construction is rectilinear Steiner
minimal tree with rectilinear blockages
input: Vs
sort Vs by non-decreasing x;
perform edge connection for R2 and R6 of all corners;
sort Vs by non-decreasing y;
perform edge connection for R4 and R8 of all corners;
sort Vs by non-decreasing x+y;
perform edge connection for R1 and R5 of all corners and R1
        and R3 of all pins
sort Vs by non-decreasing y-x;
perform edge connection for R3 and R7 of all corners and R2
        and R4 of all pins
Return :spanning graph G for Vs
```

Figure 3.19: The spanning graph construction algorithm. The detail of these algorithm can be found at [17])

the END net (it means any pin belong the sub net), the search stops and then picks up the END point to be the START point, the START point to be the END point. The pins of two-pin net will be the START point one by one. We choose the better solution which be smaller total wirelength from these two calculations.

When we do the search, we not only consider the total wirelength, but also the co-edge. Because of the start and end points in the connection path might have co-edge to be used. The total wirelength should be estimated as the length minus co-edge.

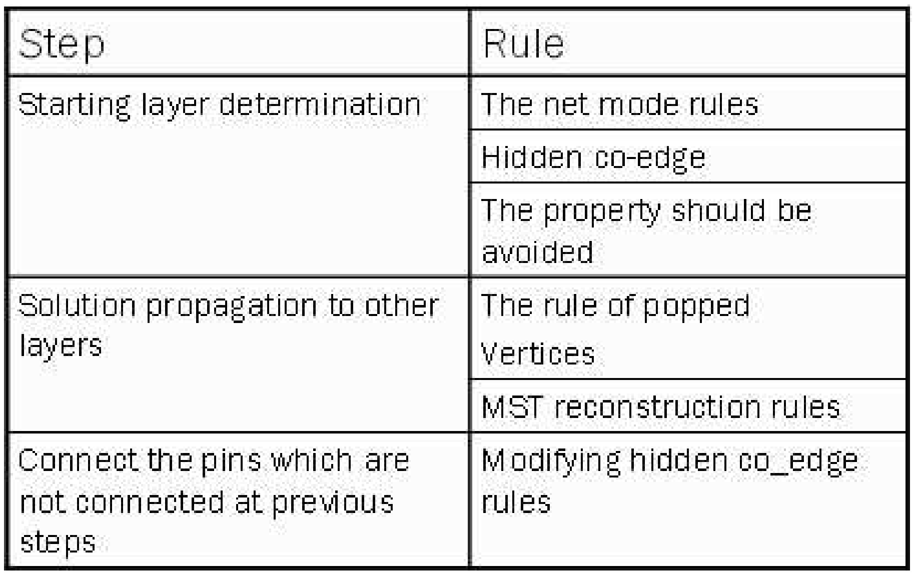
When we are doing a search, we can dynamically keep the best solution. If the total wirelength until now is larger than the best solution, we can drop this search for saving counting time. Then, there is a new problem how to get the initial solution efficiently. The way to get an initial solution is to calculate the points which have more chances of exiting a path first. The points which have more chances of exiting a path mean the points nearest the END points in Manhattan distance.

We implement and modify the Dijkstra's algorithm [6] for shortest path search. We modify the lightest vertex as the point nearest the target not nearest the source. After completing one of two-pins net, the two sub nets of pins are combined together and the vertices of the path are added into the sub net. It also increases efficient.

After shortest path search on spanning graph, we get the path for each uncompleted two-pin net. We run the path selection step for each solution, but there is one difference from the section 3.1.4. The enhanced rule should be modified because the false search should be subtracted. The modified rule is listed as table 3.2. The Rule-2 has higher priority than Rule-1 now. If the value of Rule-2 is not zero, it means there is at least one successful two-pin net routing from previous step at this direction. The successful two-pin net represents a co-edge, and the failed search time means might be a existed co-edge. Finally, we list all rules in 3.20.

Table 3.2: The modified hidden co-edge rules for the final step which is described at section 3.3. They are modified from 3.1.

	Priority	Formula
Rule-1	Med.	((Failed search time -1) * length)
Rule-2	High	(Success Fail time * length)
Rule-3	Low	((All search time -2)* length)



Step	Rule
Starting layer determination	The net mode rules
	Hidden co-edge
	The property should be avoided
Solution propagation to other layers	The rule of popped Vertices
	MST reconstruction rules
Connect the pins which are not connected at previous steps	Modifying hidden co_edge rules

Figure 3.20: The Rule list of our algorithm.)

Chapter 4

Experimental Results

We implemented our algorithm in the C++ language on a 2.8 Ghz AMD-64 machine with 2 GB memory under Linux version 2.4.21-51.ELsmp system. There are totally 23 benchmark circuits, fourteen test cases used in [14](rc1-rc14) in single layer, five industrial test cases (ind1-ind5)and five random cases in multi layers(rt1-rt5). The cases in single layer got from [20]. The cases in multi-layers got from Synopsys and [2].

We compared our algorithm with those presented in [20], [14], [19] at one layer, and [2] at multi layers. The paper [2] is based on [14] to extend the algorithm. The results of these papers are quoted from the paper, where the algorithm [20] was performed on a Sun Blade 2000 workstation with 1200 MHz CPU and 8GM memory. The [19] was performed on a Sun Blade 2000 workstation with 1200 MHz CPU and 4GM memory. The [14] is performed on a 2 GHz AMD-64 machine with 8GB memory. The [2] is performed on a 2.8 GHz AMD-64 machine with 8 GB memory under Ubuntu 6.606 operating system.

4.1 Single Layer Routing Problem

Table 4.1 lists the total wirelength of these algorithms the comparison means to compare the total wierlength of our algorithms with best solution. We discover the solution of our algorithm at most worst than others 1%. This is because the topology getting by our method are different from [14], [19], but the same as [20]. The different topology would cause different solution.

Table 4.1: The comparison on the total wirelength in single layer cases. We compare the total wierlength of our algorithms with best solutions in comparison column. The best solutions are mark as boldface type in the table.

	Pin	Obs.	[20]	[14]	[19]	Ours	Comparison(%)
Rc1	10	32	626	632	614	**614**	0
Rc2	74	625	1640	X	1632	**1608**	0
Rc3	115	1204	2872	X	2820	**2796**	0
Rc4	10	10	27250	26900	26120	**26040**	0
Rc5	30	10	43220	42210	42320	**42170**	0
Rc6	50	10	56500	55750	55170	**54800**	0
Rc7	70	10	61090	60350	**59670**	60050	-0.63
Rc8	100	10	76870	76330	**75410**	75910	-0.65
Rc9	100	500	84327	83365	**81904**	82778	-1.05
Rc10	200	500	115461	113260	**112391**	113046	-0.57
Rc11	200	800	122574	118747	**117602**	119092	-1.25
Rc12	200	1000	120017	116168	**115448**	117244	-1.53
Rc13	500	100	172490	170690	**169160**	171240	-1.21
Rc14	1000	100	238377	**236615**	237475	237834	-0.51

4.2 Multi-Layers Routing Problem

The table 4.3 lists the information of test bench which we use in multi-layer routing problem. The table 4.4 list the results of our algorithm and [2], and we compare the cost with [2]. The algorithm of CC means the simple algorithm, based on the construction by correction approach in [2]. It first constructs a minimum spanning tree for all pins, then transforms slants edges into vertical / horizontal edges to form an initial Steiner tree. Finally, it replaces the edges overlapping obstacles with edges around the obstacles with a smaller cost.

The table 4.4 list the cost and the number of via. The cost means wirelength +

Table 4.2: The comparison on the CPU time in single layer cases. Our run time is worse than other approaches in a lot of cases. The diminution of the performance comes from the difference of connection graph and topology construction.

	Pin	Obs.	[20]	[14]	[19]	Ours
Rc1	10	32	<0.01	<0.01	<0.01	<0.01
Rc2	74	625	0.1	X	0.07	0.21
Rc3	115	1024	0.21	X	0.14	0.73
Rc4	10	10	<0.01	<0.01	0.02	0.01
Rc5	30	10	<0.01	<0.01	0.02	0.01
Rc6	50	10	<0.01	<0.01	0.06	0.02
Rc7	70	10	<0.01	<0.01	0.09	0.03
Rc8	100	10	<0.01	0.01	0.14	0.05
Rc9	100	500	0.31	0.24	0.81	1.29
Rc10	200	500	0.36	0.43	1.16	1.96
Rc11	200	800	1.53	0.83	2.02	3.74
Rc12	200	1000	1.8	0.91	2.72	4.87
Rc13	500	100	0.27	0.61	1.73	6.13
Rc14	1000	100	0.81	3.15	10.05	14.72

C * number of vias. We compared our cost with the algorithm in [2] and then shown in the cost reduction column. Here, we set C=3 to be the constant in these cases. The table 4.5 list our cup time and we compared the cpu time with the algorithm in [2] and then shown in the run time reduction column. We can easily see the CPU time is faster than [2]. When the case size become larger, the saving of CPU time is fabulous. The algorithm in [2] is propagated the pins to other layers. The solution space at the algorithm is fabulous at large case, even they not propagate pins to every layer at big case.

In addition, the rc14 in single layer and the rt5 in multi-layer are both containing 1000 pins and 100 obstacle. However, we spend more CPU time in rc14 than in rt5. The reason is all the pins and obstacles locate at the same layer in rc14, and the pins and obstacles locate at different layers in rt5. The cpu time of constructing connection graph depends on the number of pins and obstacles in one layer. Our algorithm will have higher performance in multi-layer.

Figure 4.1 and 4.2 are shown one of single layer rouging result and one of multi layers routing result projected to a plane without showing the obstacles.

Table 4.3: We list the information of test bench which we use in multi layer problem. The information include the number of pins, obstacles and layers.

	Pin	Obs.	Layer
ind1	50	6	5
ind2	200	85	6
ind3	250	13	10
ind4	500	100	5
ind5	1000	20	5
rt1	25	10	10
rt2	100	20	10
rt3	250	50	10
rt4	500	50	10
rt5	1000	100	5

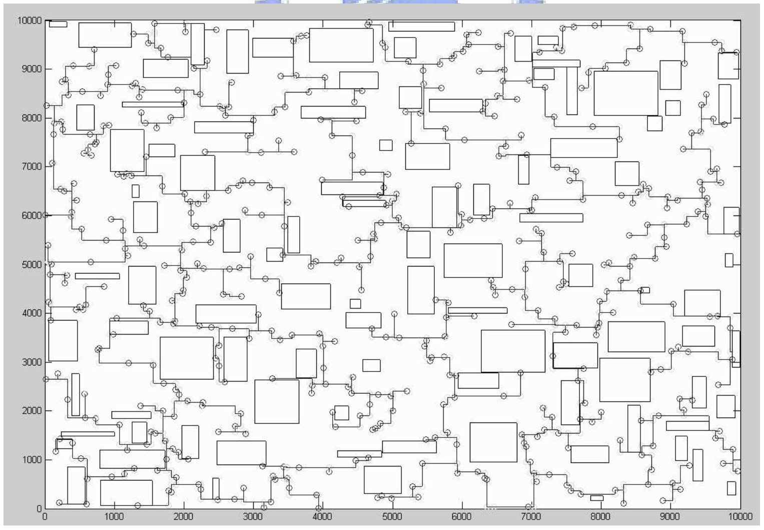


Figure 4.1: The single layer routing result of Rc13. The number of pins is 500, and the number of obstacles is 100.

Table 4.4: The cost and the number of vias of the algorithm in [2] and ours in multi-layer cases. The cost is wirelength + C * number of vias and the cost reduction means the results of the algorithm in [2] compared with ours. We set C=3 to be the constant in these cases.

	Ours		CC[2]		Cost	ML-OASG[2]		Cost
	Cost	Via	Cost	Via	reduction	Cost	Via	reduction
ind1	62809	33	82556	59	23.92%	56177	49	-11.81%
ind2	14296	150	17568	293	18.62%	12689	223	-12.66%
ind3	13218	252	17837	529	25.89%	11047	359	-19.65%
ind4	76738	0	273235	0	71.92%	77509	0	0.99%
ind5	14473173	0	23314944	0	37.92%	14656729	0	1.25%
rt1	4781	13	5095	91	6.16%	4379	76	-9.18%
rt2	11292	134	12885	290	12.36%	9623	215	-17.34%
rt3	18221	361	23233	705	21.57%	15801	490	-15.31%
rt4	25984	632	29464	1282	11.81%	22355	922	-16.23%
rt5	31310	719	38702	1102	19.1%	28213	863	-10.98%

Table 4.5: The list of the CPU time by the algorithm in [2]. We compared our run time with the ML-OASG algorithm in [2] and list in "run time reduction" column. [2] is performed on a 2.8 GHz AMD-64 machine with 8 GB memory and ours is performed on a 2.8 GHz AMD-64 machine with 2 GB memory

	CC[2]	ML-OASG[2]	Ours	Run time
	Run time	Run time	Run time	reduction
ind1	0.02	0.06	0.01	91.67%
ind2	0.21	2.96	0.40	83.11%
ind3	0.32	3.28	1.27	57.62%
ind4	0.97	7.86	2.43	68.70%
ind5	3.28	47.00	3.57	91.67%
rt1	0.01	0.05	0.01	80.00%
rt2	0.06	0.82	0.17	76.82%
rt3	0.41	6.60	1.22	76.52%
rt4	1.45	18.11	3.10	85.69%
rt5	5.25	27.73	5.50	80.60%

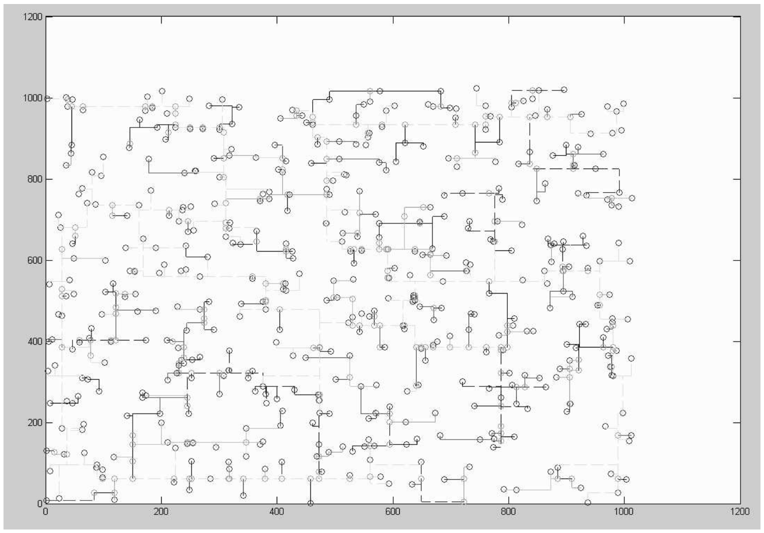


Figure 4.2: The routing result of rt4. All pins and path are projected to a plane, without showing the obstacles. The number of pins is 500, and the number of obstacles is 50.

Chapter 5

Conclusions and Future Works

In this thesis, we have proposed an algorithm, which can get good solution at single layer and fast yet effective at multi layers. The solution is limited by topology, but previous routing result at completed layers can provide an effective approach in multi layers. Experimental results have shown that our algorithm is still effective in larger case. The idea to estimate the co-edge is good for wirelength saving. It is better than just considering the U-Shaped refinement.[20]. Our algorithm and [20] have the same topology, but our results are better than [20]. In multi-layer cases, we proposed a hierarchical and heuristic approach to solve this problem. Experimental results have shown that our algorithm is still efficient in larger multi-layer cases, with slightly more wirelength.

The results of our algorithm lost in some cases in single layer. The reason is the different of topology. In the multi-layer cases, the orders of layers handling will affect the results. If we improve the method to get efficient MST topology and the order of layers, we should get higher performance. The run time of our algorithm in single layer is a little large. There are two reasons for higher run time. One is the construct of the penalty MST and the other is the construction of Hanan points graph. We spent a lot of time at obstacle points construction in the construction of Hanan points graph step.The time on the construction is large, but the time on the search is small. Because of the graph information we can decide the routing path easily in multi-layers. And then, we can add the buffer insertion step into the routing flow. The routing problem with timing driven might be the good target.

Bibliography

[1] C. Bartoschek, S. Held, D. Rautenbach, and J. Vygen. " Efficient Generation of Short and Fast Repeater Tree Topologies ". In *Proceedings International Symposium on Physical Design*, pages 120–127, 2006.

[2] M. X. Lee C. W. Lin. S. L. Huang, K. C. Hsu and Y. W. Chang. "Efficient Multi-Layer Obstacle-Avoiding Rectilinear Steiner Tree Construction". In *Proceedings IEEE/ACM International Conference on Computer-Aided Design*, 2007.

[3] Y. W. Chang and S.P. Lin. "MR: A New Framework for Multilevel Full-Chip Routing ". In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 23, pages 793– 800, 2004.

[4] C. Chu. "FLUTE: Fast Lookup Table Based Wirelength Estimation sTechnique". In *Proceedings IEEE/ACM International Conference on Computer-Aided Design*, pages 696–701, 2004.

[5] C. Chu. "Fast and Accurate Rectilinear Steiner Minimal Tree Algorithm for VLSI Design". In *Proceedings International Symposium on Physical Design*, pages 28–35, 2005.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stin. "Dijkstra's Algorithm". In *INTRODUCTION TO ALGORITHMS*, pages 595–599, 2001.

[7] J. L. Ganley and J. P. Cohoon. “Routing a Multi-Terminal Critical Net: Steiner Tree Construction in Presence of Obstacles”. In *Proceedings Internationl Symposium on Circuits and Systems*, pages 113–116, 1994.

[8] M.R. Garey and D. S. Joheson. “The Rectilinear Steiner Tree Problem is NP-Complete”. *SIAM Journal on Applied Mathematics*, 32(4):826–834, 1977.

[9] D. W. Hightower. “A Solution to The Line Routing Problem on The Continous Plane”. In *Proceedings of CAN Design Automation Workshop*, pages 1–24, 1969.

[10] Y. Hu, T. Jing, X. Hong, W. Chang Z. Feng, and G. Yan. “An-OARSMan: Obstacle-Avoiding Routing Tree Construction with Good Length Performance ”. In *Proceedings IEEE Asia and South Pacific Design Automation Conference*, pages 7– 12, 2005.

[11] R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh. “ Pattern routing: use and theory for increasing predictability andavoiding coupling ”. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 777–790, 2002.

[12] C. Y. Lee. “An Algorithm for Connections and It’s Application”. In *IRE Transactions on Electronic Compute*, pages 346–365, 1961.

[13] J. Lillis, C. K. Cheng, T. T. Y. Lin, and C. Y. Ho. “ New Performance Driven Routing Techniques with Explicit Area/Delaytradeoff and Simultaneous Wire Sizing ”. In *Proceedings IEEE/ACM Design Automation Conference*, pages 395–400, 1996.

[14] 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 International Symposium on Physical Design*, pages 127–134, 2007.

[15] K. Mikami and K. Tabuchi. "A Computer Program for Optimal Routing of Printed Circuit Conductors". In *Proceedings of IFIP Congress,*, volume 2, pages 1475–1478, 1968.

[16] M. Pan and C. Chu. "A Novel Performance-Driven Topology Design Algorithm". In *Proceedings IEEE Asia and South Pacific Design Automation Conference*, pages 244–249, 2007.

[17] Z. C. Shen, C. C. N. Chu, and Y. N. Li. "Efficient Rectilinear Steiner Tree Construction with Rectilinear Blockages". In *Proceedings IEEE International Conference on Computer Design*, pages 38–44, 2005.

[18] Y. shi, P. Mesa, H. Yu, and L. He. "Circuit Simulation Based Obstacle-aware Steiner Routing". In *Proceedings IEEE/ACM Design Automation Conference*, pages 385–388, 2006.

[19] Y. W. Tsai, Y. T. Chang, J. C. Chi, and M. C. Chi. "An Obstacle-Avoiding Rectilinear Steiner Minimal Tree Construction Algorithm". In *18th VLSI/CAD Symposium in Taiwan, Haulim*, 2007.

[20] P. C. Wu, J. R. Gao, and T. C. Wang. "A Fast and Stable Algorithm for Obstacle-Avoiding Rectilinear Steiner Minimal Tree Construction". In *Proceedings IEEE Asia and South Pacific Design Automation Conference*, pages 262–267, 2007.

作者簡歷

洪禎徽，民國七十年二月出生於台中市。民國九十三年六月畢業於國立中央大學電機工程學系，並於九十四年九月進入國立交通大學電子研究所就讀，從事 VLSI 實體設計方面相關研究。民國九十六年十月取得碩士學位，碩士論文題目為『使用規則導向且考慮障礙物之多層直角史坦納樹的建造』。