淡江大學電機工程學系碩士班（電路組）

碩士論文

指導教授：饒建奇　博士

重分佈層之避障繞線演算法

Obstacle-Avoiding Routing Algorithm for Redistribution Layer

研究生：王偉丞　撰

中華民國　104 年 6 月

論文名稱：重分佈層之避障繞線演算法　　頁數：33

校系(所)組別：淡江大學 電機工程學系學系(研究所) 積體電路組

畢業時間及提要別： 104 學年度第二學期　　碩士學位論文提要

研究生：王偉丞　　指導教授：饒建奇

論文提要內容：

重分佈層(RDL,Redistribution Layer)目前多使用在覆晶技術(Flip-Chip)上，而覆晶技術是一種將 IC 與基板(substrate)相互連接，基於小尺寸晶片、高 I/O 密度的封裝(Packging)方法。在封裝的過程中，先將晶片的墊片(pad)長出凸塊(bump)，然後將其翻覆過來，以面朝下的方式讓晶片上的墊片透過金屬導體與基板的接合點相互連接的封裝技術。

然而，覆晶技術最初的 I/O 接點並不具有面陣列(area array)的設計，使得此技術在早期受到不小的阻礙,於是才出現了重分佈層這樣的技術來解決這個問題，重分佈層是在晶圓表面沉積金屬層和介質層並形成相應的金屬佈線，來對晶片的 I/O 接點進行重新佈局，將其以面陣列形式佈置到較寬鬆的區域。

雖然截至今日，覆晶技術已不算是陌生的新技術了，探討重分佈層繞線演算法的論文數量也不在話下，而本篇論文則是在探討，當重分佈層中的繞線工程遇到了不可抗力的障礙(obstacle)而影響了繞線路徑時,線路該如何規劃以避開障礙。本論文從近期論文裡所討論到的模組下去做改善，並運用更為簡易之演算法省去較為複雜的步驟。

繞線演算法之目的為，讓晶片四個邊緣的 I/O 接腳(I/O pad)重新分布到平面陣列的凸塊墊片(bump pad)上。

大致的步驟分為:全域繞線與細部繞線兩種,而全域繞線又分成四個步驟:1)區塊分割 2)區塊合併 3)建立路網圖 4)分配線路。其中所談到的「區塊模組」探討的是：當一個矩形區塊的四個周圍有數條線路須經過此區塊時，該如何正確的分配線路的空間與走位。

鑑於「區塊合併」的步驟中限制了區塊每邊不可超過兩個障礙物阻擋，在演算法上更加深了其複雜度，且此步驟雖有益處，但也有其弊端，故省略此步驟以達到更快速的要求；也因此，在「區塊模組」的演算法上收到了簡化之效果，前者演算法需考慮區塊的四個周圍是否有障礙物的包覆，但若省去「區塊合併」的步驟便不用考慮到這樣的問題。

最後的研究成果與前篇相較之下，若是在障礙物較少的狀況下，多數的測試結果都能以稍快的速度與更短的路徑達成繞線問題;而若是在障礙物較多的狀況下，雖繞線路徑相較前篇較不明顯，但在演算的速度上卻能快上非常多。

未來，重分佈層將有可能因三維晶片技術(3D-IC)的突破與發展，而大量仰賴重分佈層的輔助，所以重分佈層在未來還是有可觀的存在價值。

關鍵字：重分佈層、障礙迴避、繞線、演算法

Title of Thesis： **Obstacle-Avoiding Routing Algorithm for Redistribution Layer** Total pages:33

Key word: **Rouing,RDL,Obstacle-Avoiding**

Name of Institute: **Master's Program in Electrical Engineering, Tamkang University**

Graduate date: **June 2015** Degree conferred: **Master**

Name of student: **Wei-Cheng Wang** Advisor: **Jiann-Chyi Rau**

王 偉 丞 饒 建 奇

Abstract:

RDL, redistribution layer, mostly applies on Flip-Chip technology in recent years. Here is the mention of Flip-Chip technology, which connecting both integrated circuit and the substate together, based on small-size chip and high-IO-density packaging. In the package process, first, deposits solder balls on each of the pads. And then, flipped and positioned, so that the solder balls will facing the connectors on the external circuitry.

However, in the early, Flip-Chip's I/O ports doesn't have plane array designing. So it receives a big difficulty when developing. To solve this problem, the designing of redistribution layer is came out. Redistribution layer is a re-routing layer between deposited metal layer and medium layer. It redistributes I/O port into plane array at wider area.

Though, Flip-Chip technology is getting more mature nowadays, the amount of related papers are also getting much more. And this paper is focused on when the obstacles exist in the redistribution layer and affect the routing process, how we plan the new routing against it. There has a previous work "Obstacle-Avoiding Free-Assignment Routing for Flip-Chip Designs." Then we refer their model and improve it with simpler method to reduce its

complexity.

The purpose of the routing algorithm is to redistribute the route from the I/O pads which aroundding the origin chip to the bump pads which scattering in the plan area.

The method of previous work is divided into two parts: Global Routing and Detail Routing.

Then, Global Routing is divided into four steps: 1) tile partition, 2) tile merging, 3) flow-network, and 4) minimum-cost-flow solving.

Step one, the routing plane is partitioned into a number of local regions called "tiles". Step two, merge some tiles based on a dynamic programming algorithm to improve solution quality and reduce the problem size. Step three, connect all models together, producing a global flow network. Step four, apply the minimum-cost maximum-flow algorithm to the network. Finally, transform the network-flow result into global routing topology. Then, based on the routing topology, detailed routing determines the specific wiring locations and completes the routing procedure.

Comparing to the previous work, our method omitted the second part. Because this step restrict that the edge of each tile can only have one opening at most, which means that the edge cannot have more than two obstacle aside. In consequence, this step cannot guaranteed that the result will be better. So, if we omit this step, not also reduce the complexity of algorithm, but also avoid the case that even worse.

目錄

圖目錄

表目錄

Chapter 1 緒論

1.1 研究背景與動機

覆晶技術(Flip-Chip technology，或稱倒裝晶片)，源自於 1960 年代中後期，由 IBM 所開發之技術，最早應用於大型主機上。覆晶技術既不是一種特定的封裝方式(例如：DIP 或 SOIC)，亦非一種封裝類型(例如：PGA 或 BGA)，他是一種把晶片(die)連接到封裝載體(package carrier)且使其可導通的方法。而封裝載體所扮演的角色則是晶片與外部電路連接的溝通橋樑。在一般的打線封裝(wire bonding)方式中，晶片與載體之間是利用導線去做連接，晶片以面朝上的方式置放於載體上，後以導線與晶片周圍的墊片(pad)去做連接。概念與介紹參考自文章[11]。

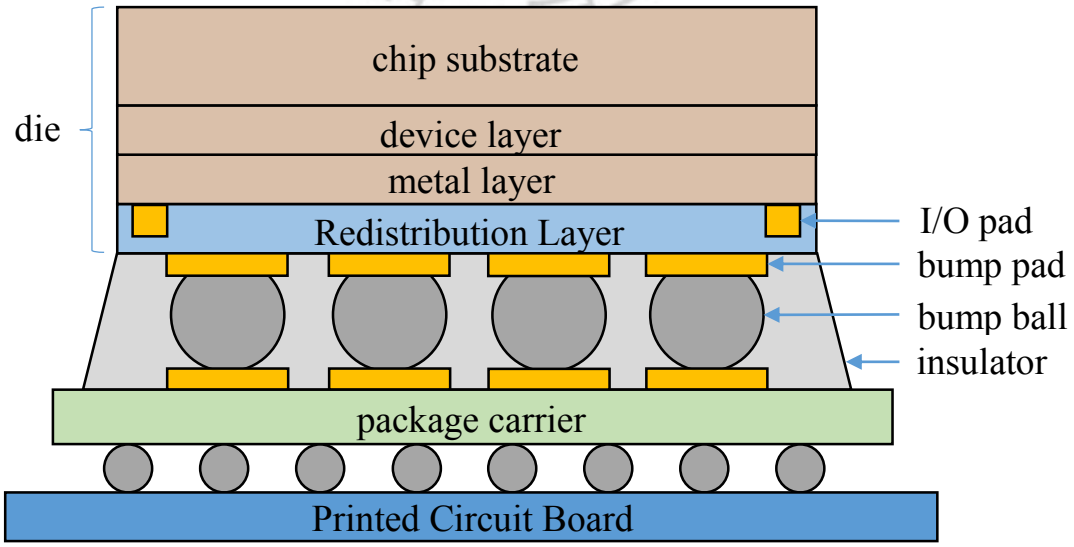

圖 1.1 覆晶技術架構圖

而覆晶所使用的方法則是，先在晶片上生成墊片(bump pad)，再於每個墊片上生成凸塊(bump ball)又稱焊球(solder ball)，然後翻轉晶片以面朝下的方式讓凸塊與載板的墊片接合，隨後重熔凸塊使其穩定，最後步驟則是使用絕緣膠(electrically-insulating adhesive, 或稱insulator)在接合處作填充，以降低晶片與載板不同的熱膨脹係數對凸塊所導致的應力，增加產品的壽命。

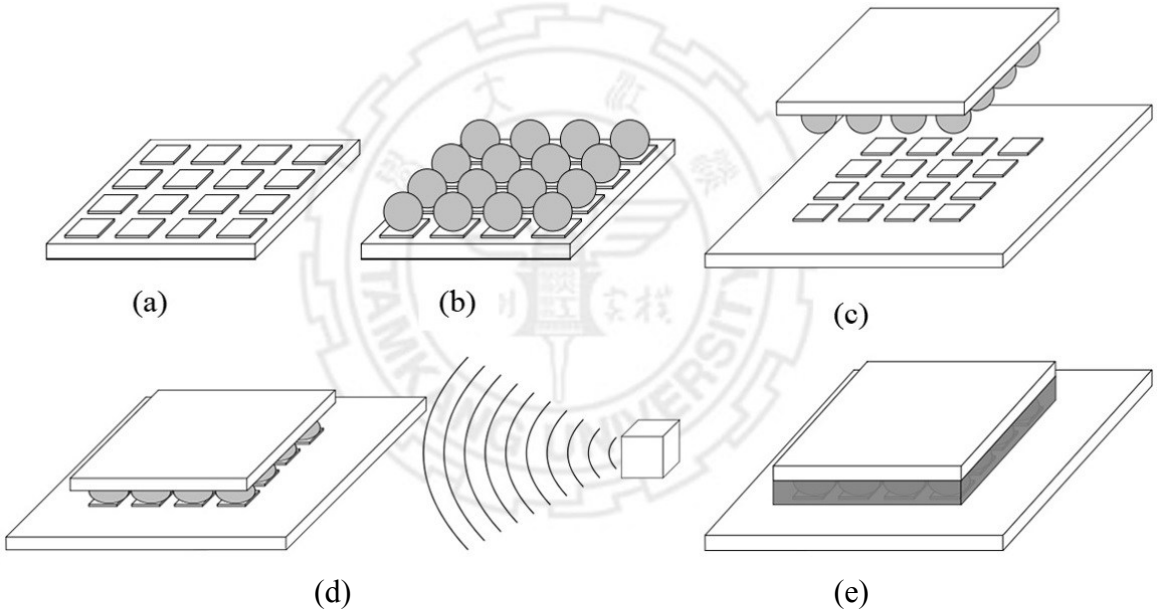

圖 1.2 覆晶技術封裝流程

然而，在最初的時期並不具有平面二維陣列(area array)的設計，所以在早期的應用上並不如預期般的推廣，直到後來才出現了重分佈層(Redistribution layer)的技術來改善這項方案，重分佈層被設計裝置在晶片的表層、並對 I/O 接口進行重新佈局，因為原本的 I/O 接口是被

分佈在晶片的四周圍，在避免需要重新設計晶片線路的情況下，就需要加裝一個重新佈局線路的裝置上去，將分散在四周的 I/O 接口重新分佈成平面二維陣列的分佈方式。

一個完整的覆晶封裝相較於打線封裝擁有更多的優點，例如：降低訊號電感、提升訊號密度與傳輸速度、更小的晶片尺寸與封裝體積等。

而本論文所要探討的問題是，當重分佈層中出現了數個大小不等的障礙物阻礙了繞線路徑時，該如何以演算法規劃新的繞線路徑以避之。本論文參考了現有著作[2]的模組設計與流程規劃後，觀察到若是把部分細節稍做修改與簡化，應可達到更快速、更佳的演算結果。

重分佈層的包含了以下數種類型可做分類：扇入式(fan-in)或扇出式(fan-out)，自由分配(free-assignment)或預分配(pre-assignment)，90°或 135°繞線角度(routing degree)。本論文則是以較為常見的前項做為研究參考。

1.2 論文總覽

本篇論文後續部分按照以下方式編列：

本論文共分為六個章節。本章節簡單介紹了研究的背景：覆晶技術，與研究的核心問題：重分佈層，並且簡單說明研究的動機。

在第二章節中將會提到一些基本概念，例如：全域規劃(global routing)、繞線限制(routing constraint)等，以及一些會使用到的理論，例如：A*搜尋演算法。

在第三章節中會簡要說明現有著作[2]所提出的內容，並分析這些方法與模組等等的特性與其優劣之處。

在第四章節中會詳細說明本論文所提出的新模組與改進過後的流程。

最後會在第五章節顯示出演算的繞線結果並與現有著作的結果去做比較，然後會在第六章節的部份去做總結。

Chapter 2 基本概念與理論

本章節會介紹一些繞線(routing)相關的基本概念，還有一些我們將會使用到的一些理論。在 2-1 裡會介紹繞線工程時會使用到規劃方法：全域規劃與細部規劃。在 2-2 裡會介紹到一些設計規則(design rule)與其限制。在 2-3 裡會介紹到尋徑時採用的 A*搜尋演算法。在 2-4 裡會介紹到最小成本流問題讓我們的路徑在可行的狀況下，將成本降至最低。多數內容參考自[10]一書。

2.1 全域規劃與細部規劃

一般在繞線規劃中會被劃分成兩個部分：全域(global)與細部規劃(detail routing)兩種。

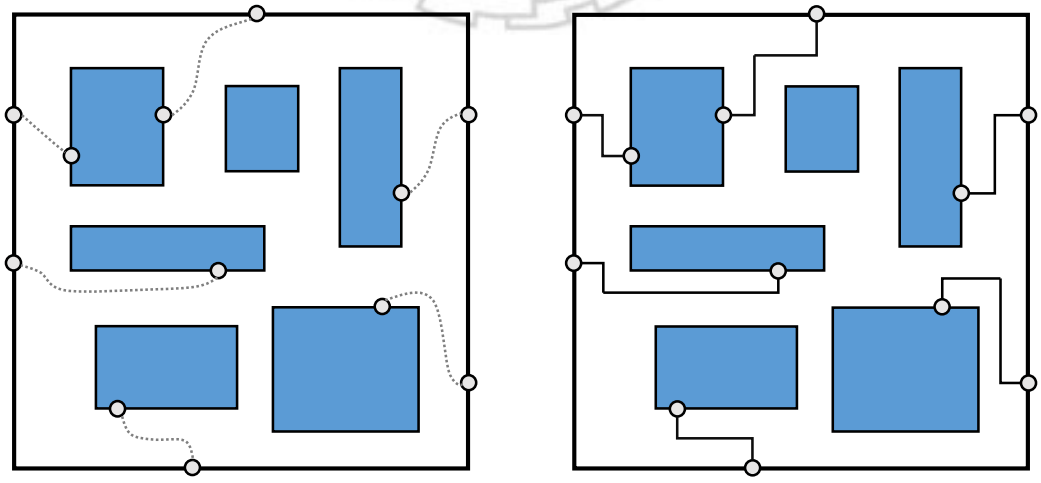

圖 2.1 全域規劃與細部規劃

2.1.1 全域規劃

首先，先把要繞線的區域(通常指的是整片區域)劃分成數個大小不一的矩形「區塊」(tiles)，最主要是利用凸塊墊片(bump pad)、障礙物的邊界去做延伸，而後分割出來形成區塊。當區塊生成後我們搭配模組下去運算，算出該區塊可以如何去做路線的規劃，例如：上下左右邊界可通過幾條線，然後再把每一個區塊去做連結型成流通路網(flow network)，最後再依據此路網把需要連接的兩個點去做連線的動作，若一條路線僅包含兩點，則可稱為雙腳線(two-pin net)。所以當全域規劃程序完成時，會形成一個鬆散的路線結構，如圖 2.1(a)。

2.1.2 細部規劃

在此步驟我們會把每一條路徑依循區塊間的連線來畫出精確的路線，而每一條精確路線所代表的是實際的幾何佈局，像是金屬佈線(metal wires)與通道(vias)，如圖 2.1(b)。

而細部規劃雖然步驟較為簡單，但較需要考慮到一些設計規則與實現上的限制，例如：線寬、線距、繞線角度等。

而在模組設定上，依據金屬層的方向性可分為「預留」(reserved)與「無預留」(unreserved)兩種；依據佈線方式可分為「網格化」(grid-based)與「無網格」(gridless)兩種。

2.2 設計規則與限制

通常在電路的設計上都需要遵循一些設計規則，例如導線(wires)與導通孔(vias)的最小寬度，導線與導線之間的間距等等；而還有前面所提到的網格化繞線方式就會包含一個網格之間的距離叫：「節距」(wire pitch)。其他的限制例如像是「時序限制」(timing constraint)是為了「高速」(high-speed)的設計而制。

2.3 A* 搜尋演算法

參考自演算法一書[9]，又稱A星搜尋演算法(A* search algorithm)，被發明於 1968 年，是一個被廣泛應用於「尋找路徑」(pathfinding)和「圖形遍歷」(graph traversal)的電腦演算法，其效能與準確程度使其被廣泛的運用，甚至被大量運用在網路遊戲的尋徑演算法中。前身為 Dijkstra 演算法(1959 年)，雖然 A*搜尋深度不如現有方法，但因為 A* 演算法採用了「啟發式評估」(heuristic estimate)的系統，排除了許多明顯為壞的參考，進而快速的計算出一條最短的路徑。

以下為 A*搜尋演算法之虛擬碼：

```
1.  Add START to OPEN list
2.  While OPEN not empty
3.      get node n from OPEN that has the lowest f(n)
4.      If n is GOAL then return path
5.      move n to CLOSED
6.      For Each n' = CanMove(n)
7.          If n' is in CLOSED then continue
8.          g(n') = g(n) + 1
9.          If n' is not in OPEN or n' is better
10.             add n as n's parent
11.             f(n') = g(n') + h(n')
12.             add n' to OPEN
13.         End if
14.     End for
15. End while
```

2.4 最小成本流問題

(Minimum-Cost Flow Problem, MCFP)

給定一個具有方向性的圖形$G = (V, E)$，節點集合 V、路線集合 E，包含源點 $s \in V$、匯點 $t \in V$，而每條路線 $(u, v) \in E$ 都有容量 $c(u, v) > 0$ 、流量 $f(u, v) \geq 0$ 與成本 $a(u, v)$，所以在這個邊緣上的總成本就會是 $f(u, v) \cdot a(u, v)$。而我們現在要做的即是將數條流 d 從 s 送至 t。

所以最後就整個有向圖形來看，我們的總成本就會是：

$$\sum_{(u,v) \in E} f(u, v) \cdot a(u, v)$$

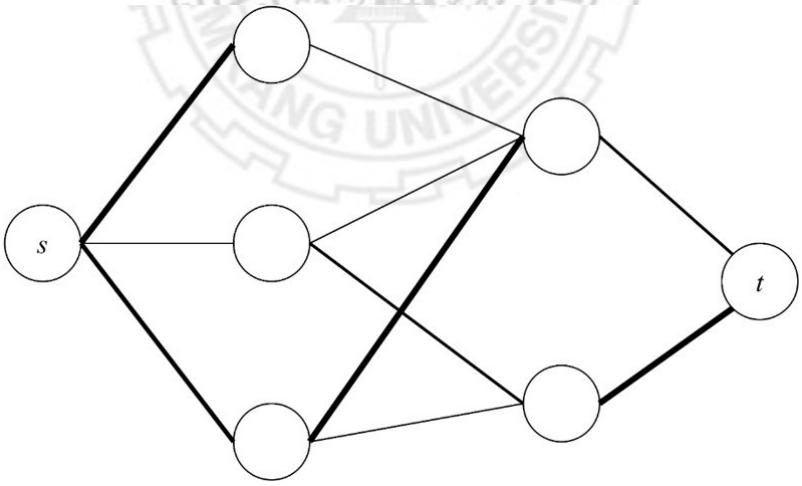

圖 2.2 最小成本流問題示例

像是我們所參考的現有著作之一[3]，就有使用到這樣的有向圖形來建立「繞線圖形結構」(Routing Graph Construction)。

Chapter 3 現有方法論

本章節會介紹到現有的著作，在 3.1 節裡會簡要介紹繞線工程的完整流程，在 3.2 節裡會介紹現有著作所改良的新模組，並提到此模組需要遵守的方程式與不等式，最後再以虛擬碼簡要描述此模組的演算流程。在 3.3 節會介紹到兩篇針對預分配方式去做重分布層繞線的現有方法。

3.1 完整繞線流程

誠如第二章所提及，一個完整的繞線流程包含了全域規劃以及細部規劃，而全域規劃在以往的著作中被規劃為以下四個部分。

3.1.1 區塊劃分

「同時」將凸塊墊片與障礙物的邊界去做延伸，延伸直到碰觸到其他物件的邊界時停止，隨後在這些交會的延伸線中去建立矩形「區塊」。

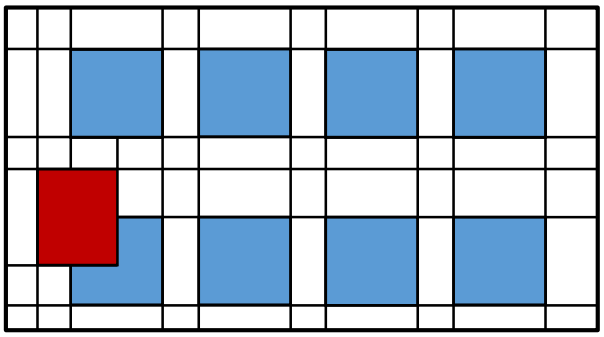

圖 3.1 區塊劃分

3.1.2 區塊合併

由於上一個步驟所建立出來的區塊可能會因為物件較多且複雜而會出現數量過多的情況，如圖 3.1 所示，由於障礙物的上邊界向外延伸時並未碰觸到其他物件，於是一直延伸到晶片邊界(chip boundary)才停止。

此外，還會有「最佳化缺失」(optimality loss)的狀況出現，如圖 3.2(a)，假設線寬為一單位，而這裡剛好有八塊寬度為一單位的小區塊堆疊在一起，由於寬度不夠讓一條線路通過所以該區域是無法通行的，但若我們把八個區塊去做合併的話便能使線路通過；反之，圖 3.2(b)，假設有兩塊高度為八單位、寬度為三單位的區塊併在一起，在如此的狀態下各區塊可以通過三條線路，如藍色線段所示，但若此時兩區塊去做合併時，便會出現如紅色線段所示的錯估狀況而形成衝突(violation)。

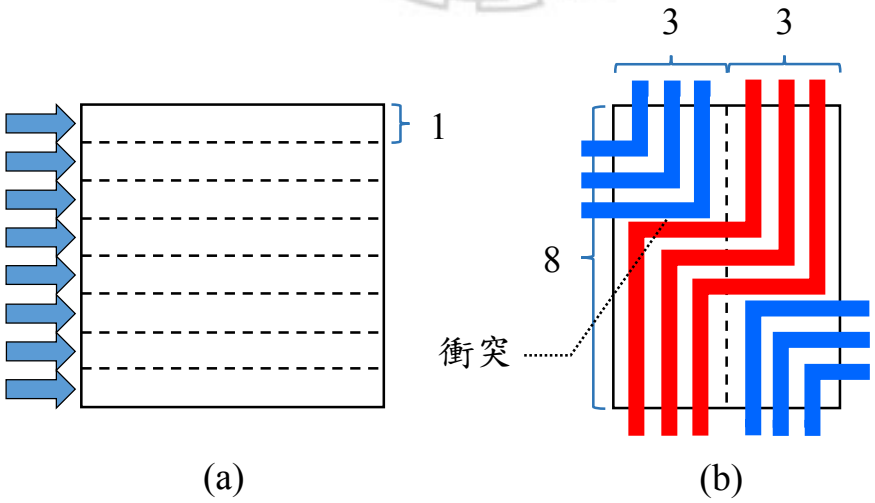

圖 3.2 最佳化缺失

3.1.3 流通路網結構

當完成了完整的區塊規劃後，接著就是將「障礙感知流通路網模組」(obstacle-aware network-flow model，以下簡稱OA-model)應用於各個區塊上，簡而言之即是計算每個區塊可行的組合，而詳細內容與演算法將在3.2小節做介紹。

當每個區塊的OA-model建立完成後，我們將會把每條可行的通行線段去做連結，每個OA-model的相連、OA-model與I/O接口或凸塊墊片的連結，進而形成一個涵蓋整片區域的流通路網結構(Flow-Network Construction)。

3.1.4 最小成本問題應用

透過上個步驟所建構的流通路網結構，我們會在此步驟運用到「最小成本最大流量問題」(minimum-cost maximum-flow problem)，由於每條路徑都有「通道限制」與「通行成本」，要如何在「完成每個流通」的條件下使成本達到最低，最後得到一個最佳解的路徑拓樸。

這個部分的線路源點與終點的配對(assignment)，我們將參考[1]與[6]的架構來完成，此方法首先將每個區塊的中心點(Tile Node)與墊片間的「中間點」(Intermediate Node)建立出來，使源點可藉由這些點繞路去找到更深入的終點，如圖3.3。

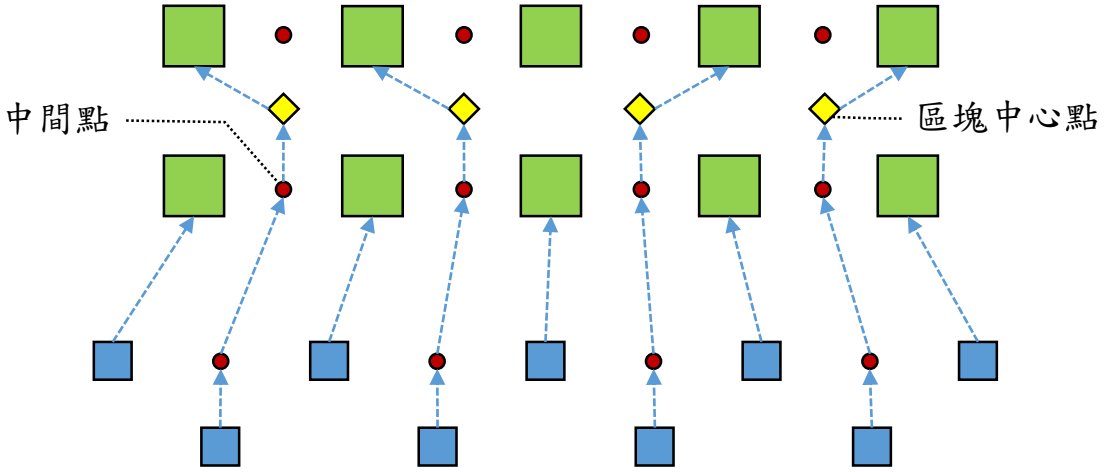

圖 3.3 區塊中心點與中間點

3.1.5 細部規劃

在這個部分我們依然前一個小節所參考[1]與[6]的方法來完成。

首先會利用「交叉點」(Cross Point)將被重複連線的中間點分散成多個交叉點，然後將源點與終點給定一組編號(Net Ordering Determination)，最後再利用橫向的「軌道」(Track)來分配每條路徑所要走的位置，如圖 3.4，在[7]中也有提到軌道分配的詳細部分。

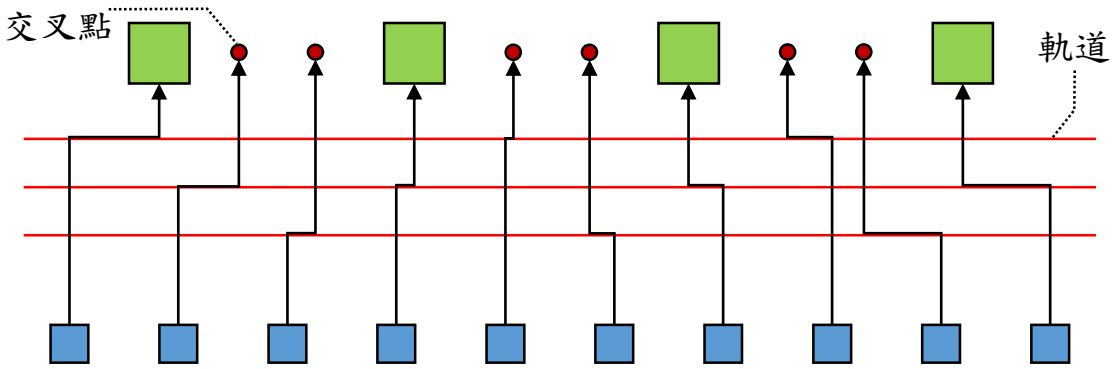

圖 3.4 交叉點與軌道

3.2 障礙感知流通路網模組

每一個矩形區塊的四個邊都能讓線路通過，但是通過的數量有一定的限制，且每個邊的數量也有著微妙的關聯，因為當一條線路從某個邊進入時就一定會從另一個邊離開，所以考慮了這些條件後，最早的模組出現在[8]裡，由於該模組演算條件僅考慮了方形區塊，且並未考慮當區塊外圍有被障礙物阻擋的情況，於是後來出現了[2]去做模組的改良，而我們將會從此篇的改良版本去做介紹。

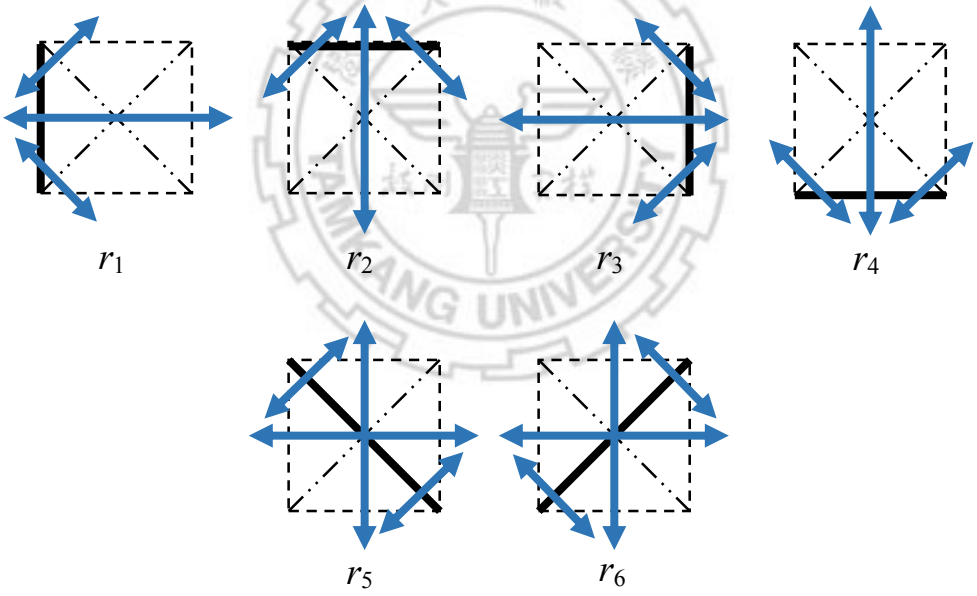

圖 3.5 含有六個變數的 r-vector

一個矩形區塊的四個邊都沒有因為障礙物而無法通行的基本情況下，我們設立六個變數 $r_1 \sim r_6$，$r_1 \sim r_4$ 依序代表左邊界、上邊界、右邊界、下邊界能通過的線路數量，而 r_5 與 r_6 則代表了下行對角線(descending

diagonal)與上行對角線(ascending diagonal)的通過數量，這六個變數集合起來則稱作 r-vector，如圖 3.3。

而一組合理的 r-vector 必須遵守以下不等式：

$$r_1 \leq r_2 + r_3 + r_4, \quad (1)$$

$$r_2 \leq r_1 + r_3 + r_4, \quad (2)$$

$$r_3 \leq r_1 + r_2 + r_4, \quad (3)$$

$$r_4 \leq r_1 + r_2 + r_3, \quad (4)$$

$$r_5 \leq \min\{r_1 + r_4,\ r_2 + r_3\}, \quad (5)$$

$$r_5 \geq \max\{\min\{r_1,\ r_2\},\ \min\{r_3,\ r_4\}\}, \quad (6)$$

$$r_6 \leq \min\{r_1 + r_2,\ r_3 + r_4\}, \quad (7)$$

$$r_6 \geq \max\{\min\{r_1,\ r_4\},\ \min\{r_2,\ r_3\}\}, \quad (8)$$

$$r_5 + r_6 \geq \max\{r_1 + r_3,\ r_2 + r_4\}, \quad (9)$$

不等式(1)~(4)確保單邊進入的線路數量不會超過其他三邊的總和；不等式(5)~(8)計算通過對角線的線路數量之上限與下限，不等式(9)則是要讓通過對角線的線路必須大於進入區塊的線路總數，如此才能確保區塊內有足夠空間讓線路通過。

基於以上的 r-vector 概念，而後衍生出了如圖 3.4 這樣的一個 OA-model，包含了九個容量(capacities)：上下左右單邊(各標記為 c_N、c_S、c_W、c_E)、西北(c_{WN})、東北(c_{NE})、東南(c_{ES})、西南(c_{SW})的斜邊方向線路，

還有橫跨整個矩形區塊(c_C)的垂直或水平線路。

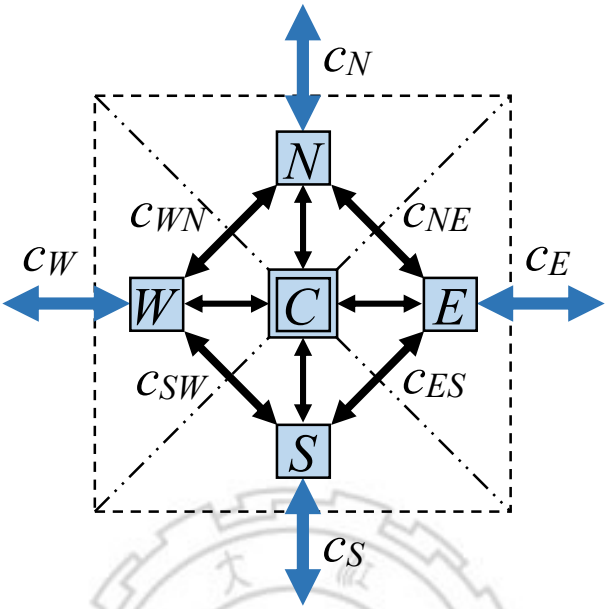

圖 3.6 OA-model 與九個容量變數

而這九個容量與前面所提的 r-vector 之間的關係如下方所列出之方程式與不等式：

$$c_W = r_1,\ c_N = r_2,\ c_E = r_3,\ c_S = r_4\,, \tag{10}$$

$$c_{SW} + c_{WN} + c_C \geq r_1\,, \tag{11}$$

$$c_{WN} + c_{NE} + c_C \geq r_2\,, \tag{12}$$

$$c_{NE} + c_{ES} + c_C \geq r_3\,, \tag{13}$$

$$c_{ES} + c_{SW} + c_C \geq r_4\,, \tag{14}$$

$$c_{WN} + c_{ES} + c_C = r_5\,, \tag{15}$$

$$c_{NE} + c_{SW} + c_C = r_6\,, \tag{16}$$

以下為 OA-model 演算法之虛擬碼：

1. Input r-vector
2. $C_W = r_1$, $C_N = r_2$, $C_E = r_3$, $C_S = r_4$;
3.
4. $m_0 = \min\{ r_1, r_2, r_3, r_4 \}$;
5. /* Compute upper-bounds of capacities */
6. $m_{WN} = \min\{r_5 - m_0, \lfloor 0.5(2r_5 + r_6 - m_0 - r_3 - r_4)\rfloor\}$;
7. $m_{NE} = \min\{r_6 - m_0, \lfloor 0.5(2r_6 + r_5 - m_0 - r_4 - r_1)\rfloor\}$;
8. $m_{ES} = \min\{r_5 - m_0, \lfloor 0.5(2r_5 + r_6 - m_0 - r_1 - r_2)\rfloor\}$;
9. $m_{SW} = \min\{r_6 - m_0, \lfloor 0.5(2r_6 + r_5 - m_0 - r_2 - r_3)\rfloor\}$;
10.
11. /* Compute excess value of an edge constraint */
12. $A_1 = (m_0 + m_{SW} + m_{WN}) - r_1$;
13. $A_2 = (m_0 + m_{WN} + m_{NE}) - r_2$;
14. $A_3 = (m_0 + m_{NE} + m_{ES}) - r_3$;
15. $A_4 = (m_0 + m_{ES} + m_{SW}) - r_4$;
16. $A_5 = (m_0 + m_{WN} + m_{ES}) - r_5$;
17. $A_6 = (m_0 + m_{NE} + m_{SW}) - r_6$;
18.
19. /* Eliminate excess values by decreasing m */
20. subtract $\min\{m_{WN}, A_1, A_2, A_5\}$ from m_{WN}, A_1, A_2, A_5;
21. subtract $\min\{m_{NE}, A_2, A_3, A_6\}$ from m_{NE}, A_2, A_3, A_6;
22. subtract $\min\{m_{ES}, A_3, A_4, A_5\}$ from m_{ES}, A_3, A_4, A_5;
23. subtract $\min\{m_{SW}, A_4, A_1, A_6\}$ from m_{SW}, A_4, A_1, A_6;
24.
25. $c_{WN} = \lfloor 0.5m_0 \rfloor + m_{WN}$;
26. $c_{WN} = \lfloor 0.5m_0 \rfloor + m_{WN}$;
27. $c_{WN} = \lfloor 0.5m_0 \rfloor + m_{WN}$;
28. $c_{WN} = \lfloor 0.5m_0 \rfloor + m_{WN}$;
29. $c_C = m_0 \bmod 2$;
30.
31. return $c = (c_W, c_N, c_E, c_S, c_{WN}, c_{NE}, c_{ES}, c_{SW}, c_C)$.
32. Ouput c (the capacities of edges)

3.3 現有預分配繞線方法

在現有的兩篇論文[4][5]中，提出了基本的重分佈層繞線方法，並考慮了「預分配」(Pre-assignment)的規定，所以在連接分配時期(Connection Assignment)就去處理了 I/O 輸入與凸塊的配對，不過由於我們的方法是針對「自由配對」(free assignment)，所以我們僅參考 RDL Routing 的部分，包含了「全域線路分配」(Global Wire Assignment)、「史坦納點分配」(Steiner-Point Assignment)、「交叉點分配」(Crossing-Point Assignment)，流程如圖 3.7。

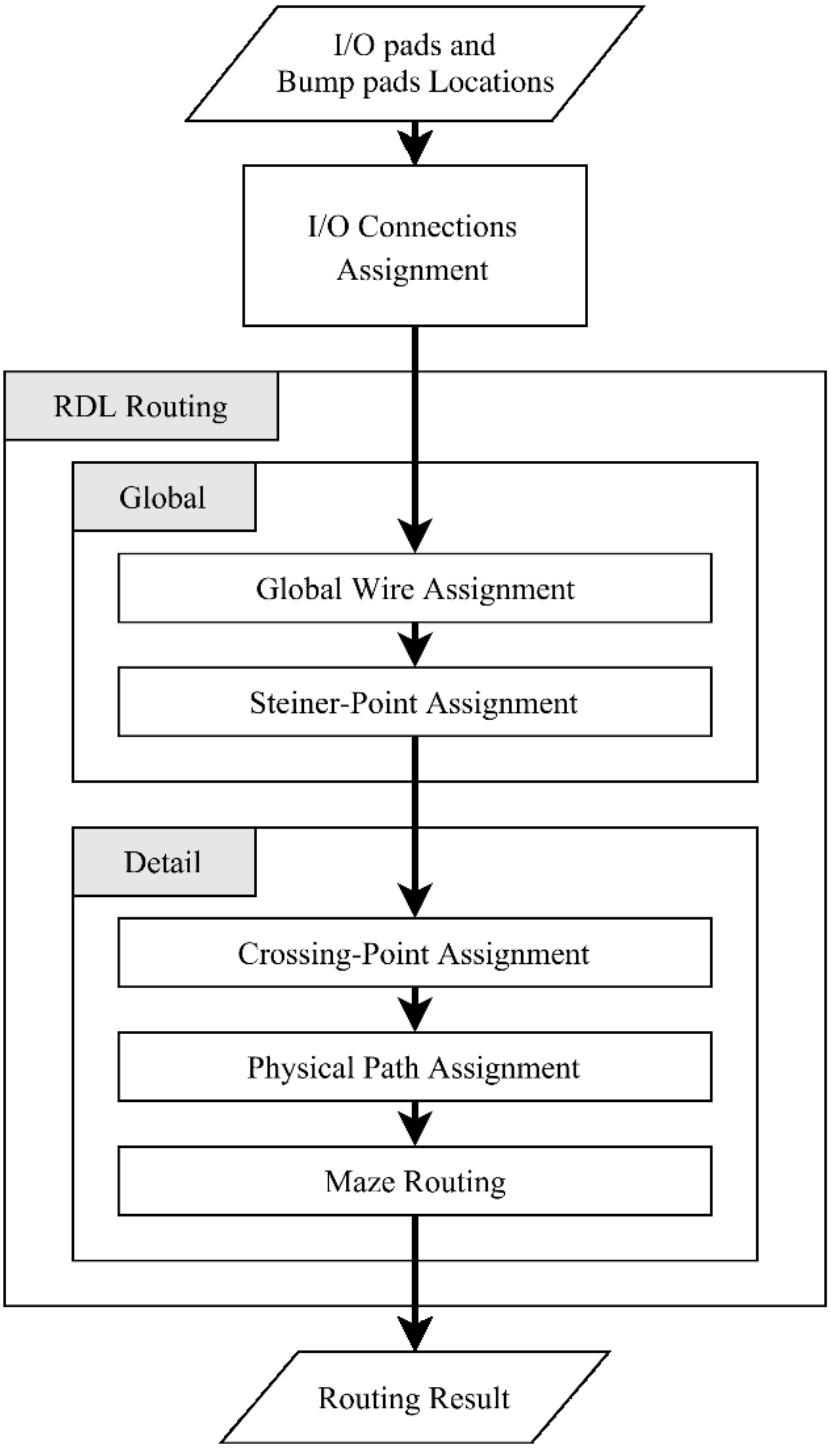

圖 3.7 現有預分配繞線方法流程

Chapter 4 提出的新方法

在本章節，我們首先在 4.1 提出問題的描述。在 4.2 提到現有方法的優劣之處，然後我們會以什麼樣的方式去做改進。在 4.3 詳細描述我們參考了現有方法後去做改進、簡化後的新模組，並使用虛擬碼簡要描述我們的演算流程。在 4.4 中我們會利用流程圖來詳細解釋每個步驟是如何去做演算的。

4.1 問題描述

首先我們會從待規劃的電路 input 檔案中，取得：繞線角度、晶片的邊界、設計規則(design rule)、線的寬度(width)、線的間距(spacing)，還有 I/O 輸入(driver)、凸塊墊片(bump pad)、障礙物的位置。

經過區塊劃分後，取得我們可以通行的空間，再將這些空間去做連線來建立出路網結構，然後再應用「最小成本問題」來配對來源端(source)與目的端(target)，最後進入到細部規劃取得精確的線路位置。

4.2 方法分析與改進

在前面 3.1.2 小節有提到「最佳化缺失」的問題，在區塊合併的步驟中可能會發生「低估可行狀況」(underestimated feasible solution)或

「超估不可行狀況」(overestimated infeasible solution)，再加上 OA-model 規定單邊不能超過兩個通道(openings)，所以在做合併時就必須考慮到這個限制還有最佳化的問題。

我們嘗試省去這樣的複雜步驟以節省運算時間與記憶體的消耗。但由於省略了區塊合併的步驟，以至於會造成區塊數量過多且複雜的問題，所以為了避免這樣的狀況，我們在前一個步驟：區塊分割上做了改進。在前面 3.1.1 時有提到，區塊分割是同時把所有延伸線都建立出來，而我們的方法則是依序先將凸塊墊片的延伸線建立完後，再去處理障礙物的延伸線，規定只要碰到其他線段隨即停止，藉此降低因為區塊被劃分而造成更多細碎且不必要的小區塊，如圖 4.1。

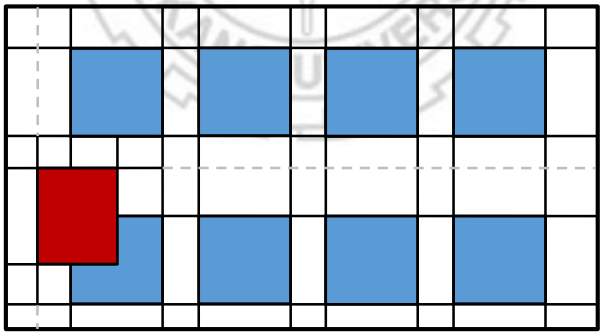

圖 4.1 優化後的區塊劃分

有了如此的規劃，每個區塊的四個邊與障礙物接觸的選項也變得簡單了，只須記錄該邊是否可通行，所以會形成五種區塊類型，如圖 4.2。

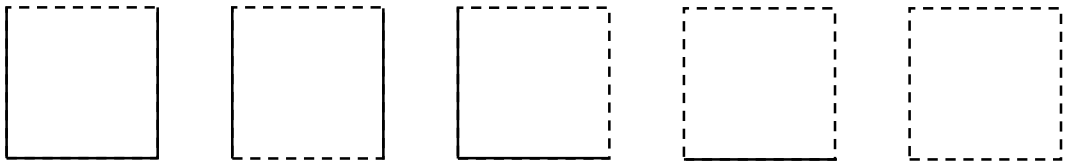

圖 4.2 五種區塊類型

圖 4.2，(a)單邊通行，由於此類型無法正常通行，故我們不建立此種區塊；(b)對邊通行，此種類型區塊只要考慮單邊(c_N、c_S、c_W、c_E)可通過的線路容量即可；(c)鄰邊通行，只需考慮斜邊(c_{WN}、c_{NE}、c_{ES}、c_{SW})方向的線路容量即可，會受到單邊容量的影響；(d)三邊通行，只有一邊無法通行，但我們可先將橫跨對邊的線路放入後，再計算兩個斜邊的線路容量即可；(e)全邊通行，四個邊都沒有阻礙可通行，配線方式較為自由但較為複雜，甚至還會出現橫跨對邊的線路會有彎角(bend)的現象。

而一個全邊通行的區塊類型中，我們大略的整理出三種可能會出現的佈線狀況，如圖 4.3，(a)僅有鄰邊線路通過、(b)含有垂直穿越區塊的線路、(c)含有水平橫跨區塊的線路。

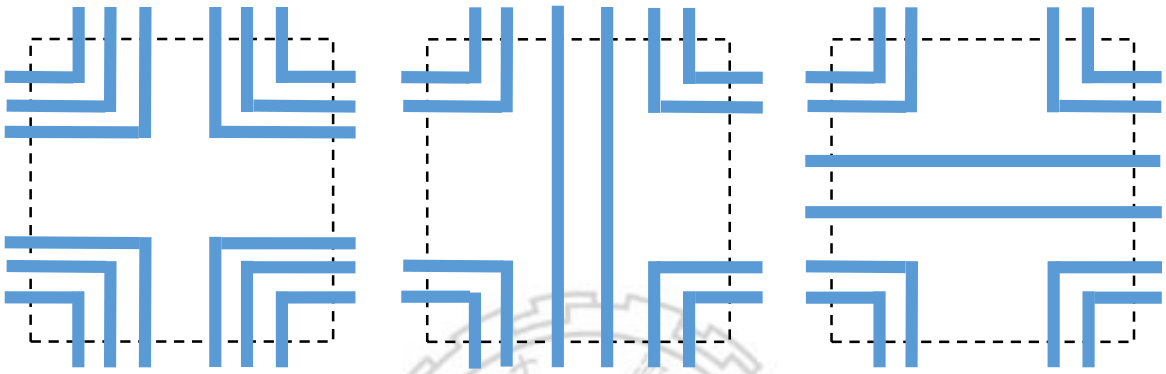

圖 4.3 全邊通行的三種佈線狀況

4.3 提出的新模組

承 4.1 所提，對邊、鄰邊、三邊的佈線方式較為簡單，故此章節僅針對較為常見且複雜的全邊通行來討論。

相較於現有著作的方法，此種類型的周圍是沒有阻礙物的，由於矩形的對邊相等，所以我們可以將單邊的容量(c_N、c_S、c_W、c_E)化簡為垂直與水平(c_V、c_H)兩種，如圖 4.4。

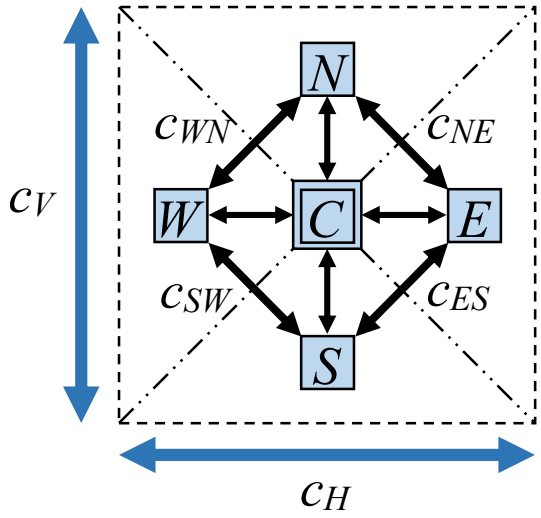

圖 4.4 簡化後的新模組

以下是此種模組類型需遵循的方程式與不等式：

$$c_V = r_2 = r_4\,,\ c_H = r_1 = r_3\,, \tag{17}$$

$$c_{WN} + c_{ES} + c_C = r_5\,, \tag{18}$$

$$c_{NE} + c_{SW} + c_C = r_6\,, \tag{19}$$

$$\max\{c_{WN} + c_{NE},\ c_{ES} + c_{SW}\} \geq r_2\,, \tag{20}$$

$$\max\{c_{WN} + c_{SW},\ c_{NE} + c_{ES}\} \geq r_1\,, \tag{21}$$

$$\min\{2(c_V - c_C) + c_C,\ \max\{c_V,\ c_H\}\} \geq \max\{r_5,\ r_6\}\,, \tag{22}$$

$$r_5 + r_6 = 2\min\{c_V,\ c_H\}\,, \tag{23}$$

4.4 程式碼規劃流程圖

4.4.1 區塊劃分

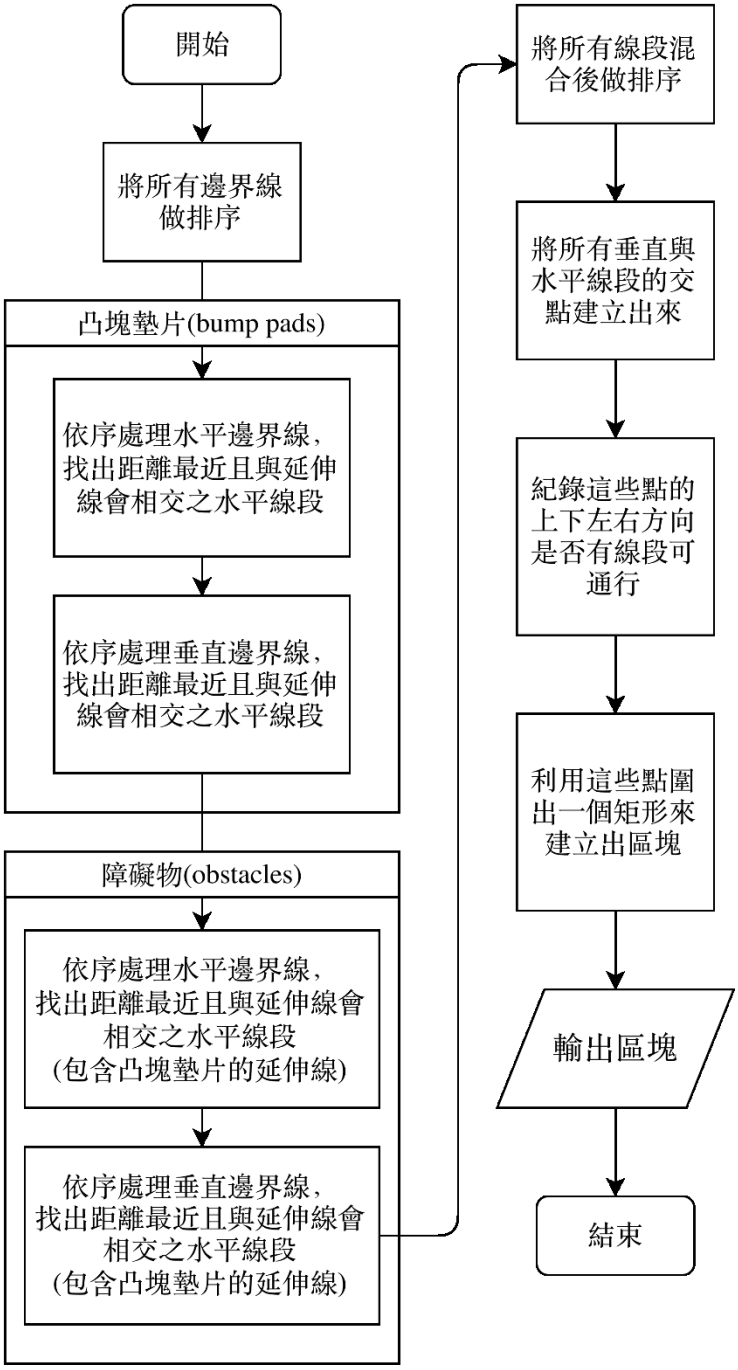

圖 4.5 區塊劃分流程圖

4.4.2 建立流通路網結構

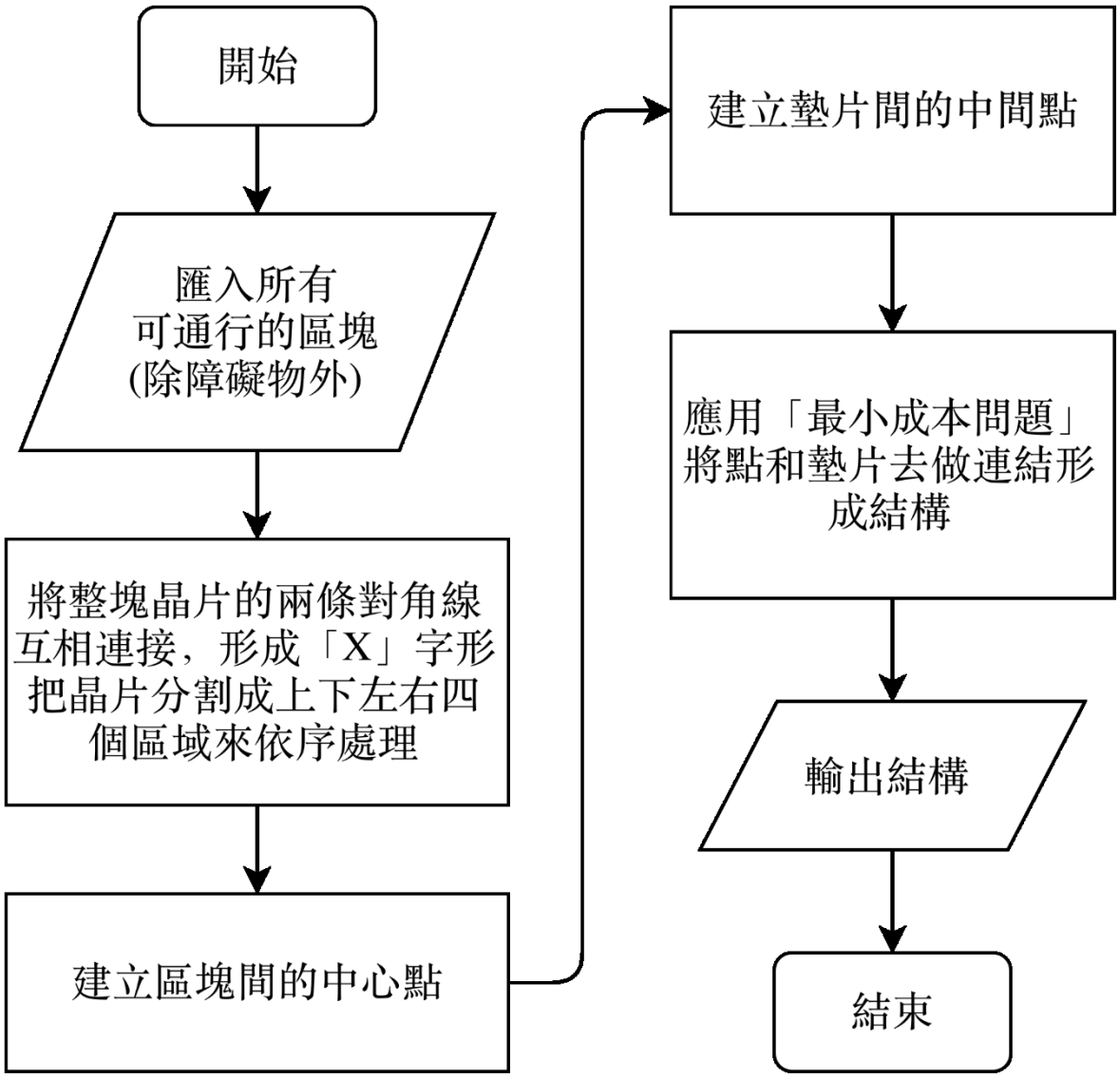

圖 4.6 流通路網結構的建立流程圖

4.4.3 細部規劃

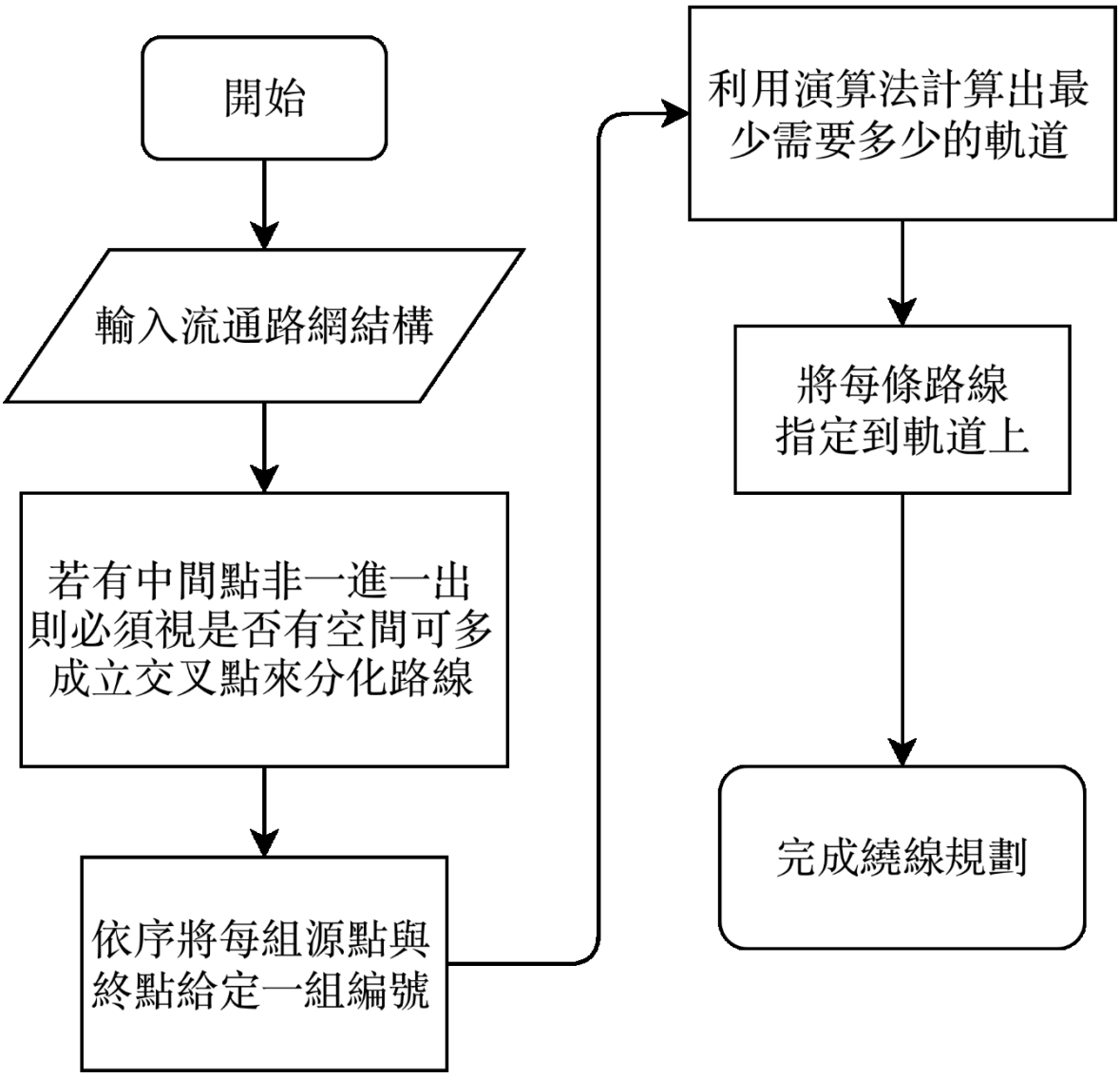

圖 4.7 細部規劃流程圖

Chapter 5 實驗結果

我們採用了一系列的工業電路(test19、test143、test143_2、test150、test200)，在 2.6GHz CPU 與 8GB 的記憶體的環境下做測試，雖然我們在運算期間花費了較多的演算時間(runtime)去做路徑的解析與尋徑，但計算後的繞線路徑長度卻有較明顯的縮減，在相同都是 100%繞線率(routability)的情況下卻可以降低較多的總線長(wire lenth)。

電路	前者方法			本論文方法		
	繞線率 (%)	總線長 (μm)	運算時間 (ms)	繞線率 (%)	總線長 (μm)	運算時間 (ms)
test19	100.0	461	40	100.0	358	5
test143	100.0	10,611	270	100.0	8,587	630
test143_2	100.0	10,967	350	100.0	8,691	870
test150	100.0	21,334	420	100.0	19,265	5,000
test200	100.0	26,645	580	100.0	23,815	7,460

表 5.1 結果比較表格

而在圖 5.1 的折線圖中以總線長的差異來做比較，可以較明顯的看出，若是物件越多的情況下，我們的方法所演算出來的結果會比現有著作的方法來得更好。

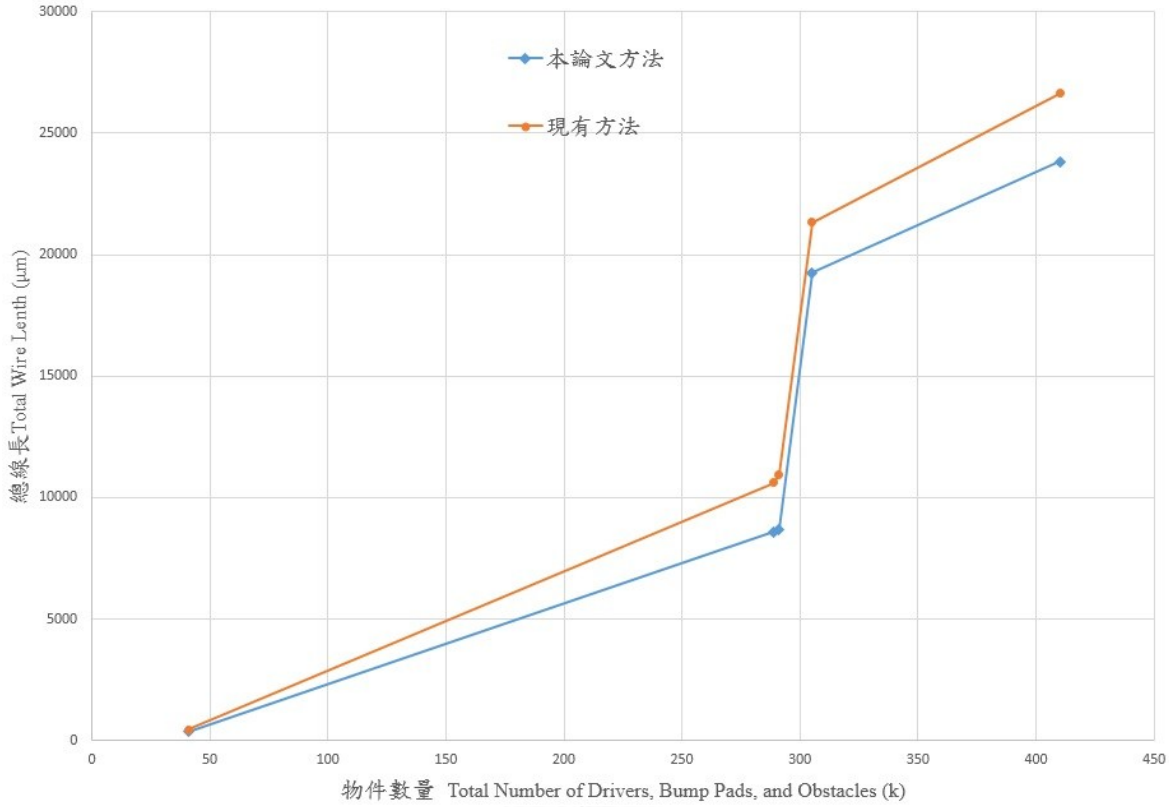

圖 5.1 結果比較折線圖

最後我們以 C#的程式來跑出我們的佈線結果圖，如圖 5.2，分布在邊界四周圍的是 I/O 輸入(drivers)，整齊分布在中央周圍的則是凸塊墊片(bump pads)，而四散在墊片中、含有數字的物件則是無法行進的障礙物(obstacles)。

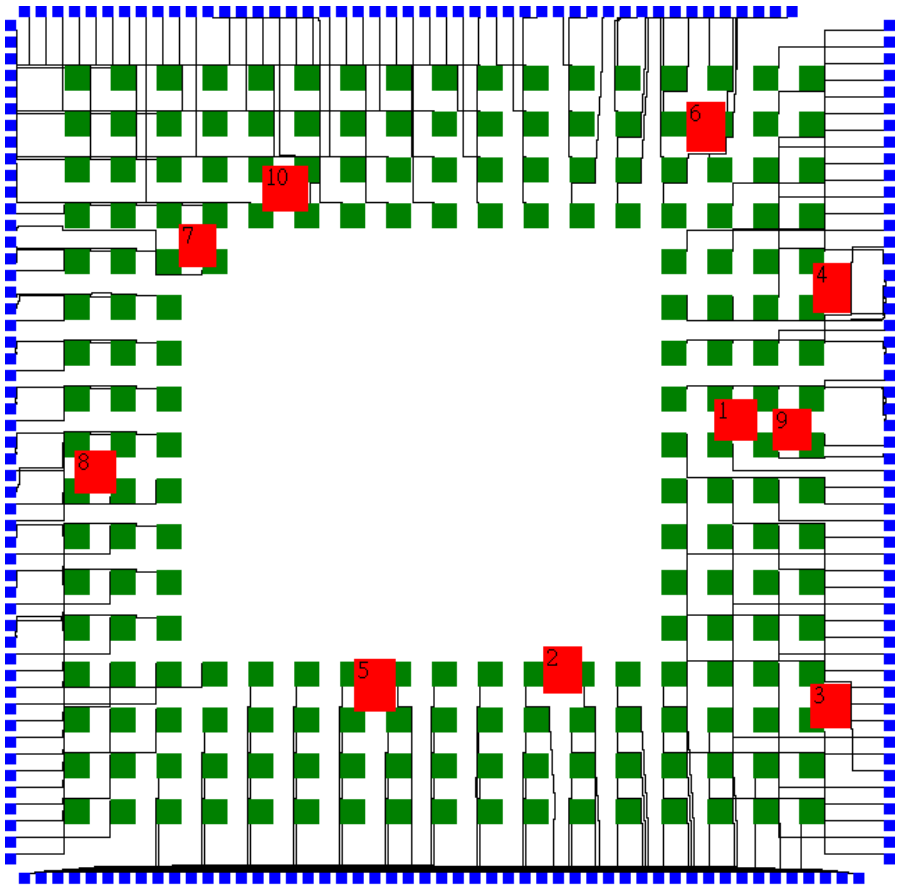

圖 5.2 佈線結果圖

Chapter 6 總結

在此篇論文中，我們首先介紹了整個繞線工程的概觀：全域規劃與細部規劃，並且介紹了 A*搜尋演算法與最小成本流問題，還有一些設計限制等等的繞線規則。

然後我們提到了現有著作的方法論，簡要的介紹了他們的演算方法與模組，由於在區塊合併的步驟中存在著超估或低估的可能，於是我們試著將此步驟省略來降低演算的時間與複雜度，但為了達到更好的效果，我們在區塊劃分的步驟去做了優化，藉以降低了劃分出來的區塊數量。

也由於省掉了區塊合併的步驟，我們的每個區塊也變得更為簡單，藉此我們改善了他們的模組，試著降低其複雜度。

最後我們再利用前面所提到的演算法來完成細部規劃的部分，進而演算出我們最後的精確路徑。

雖然我們的方法在運算時間上不甚理想，但所演算出來的總路徑長度較現有著作的方法更為簡短。

參考文獻

[1] J. W. Fang, I. J. Lin, Y. W. Chang, and J. H. Wang, "A network-flow based RDL routing algorithm for flip-chip design," IEEE Trans. Comput. Aided Design Integr. Circuits Syst., vol. 26, no. 8, pp. 1417–1429, Aug. 2007.

[2] Y. K. Ho, H. C. Lee, P. W. Lee, Y. W. Chang, C. F. Chang, I J. Lin, C. F. Shen, "Obstacle-Avoiding Free-Assignment Routing for Flip-Chip Designs," Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on , vol.33, no.2, pp.224,236, Feb. 2014.

[3] J. T. Yan, Z. W. Chen, "Pre-Assignment RDL Routing via Extraction of Maximal Net Sequence" Computer Design (ICCD), 2011 IEEE 29th International Conference on , vol., no., pp.65,70, 9-12 Oct. 2011.

[4] J. T. Yan, Z. W. Chen, "IO connection assignment and RDL routing for flip-chip designs," Asia and South Pacific Design Automation Conference, pp.588-593, 2009.

[5] J. T. Yan and Z. W. Chen, "RDL pre-assignment routing for flip-chip designs," ACM Great Lakes Symposium on VLSI, pp.401-404, 2009.

[6] J. W. Fang, I J. Lin, P. H. Yuh, Y. W. Chang, J. H. Wang, "A routing algorithm for flip-chip design," Computer-Aided Design, 2005. ICCAD-2005. IEEE/ACM International Conference on , vol., no., pp.753,758, 6-10 Nov. 2005.

[7] X. D. Liu, Y. F. Zhang, G.K. Yeap, C. L. Chu, J. Sun, X. Zeng, "Global routing and track assignment for flip-chip designs," Design Automation Conference (DAC), 2010 47th ACM/IEEE , vol., no., pp.90,93, 13-18 June 2010.

[8] T. Yan; Wong, M.D.F., "A correct network flow model for escape routing," Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE , vol., no., pp.332,335, 26-31 July 2009.

[9] A. Levitin, *Introduction to The Design and Analysis of Algorithms*, 2nd ed. Pearson, 2009.

[10] L. T. Wang, Y. W. Chang, K. T. Cheng, *Electronic Design Automation: Synthesis,*

Verification, and Test (Systems on Silicon), 1st ed. Morgan Kaufmann, 2009, ch.9~11.

[11] 鄭榮淇. 1999, June 10. *IC 載板產業展望* [Online]. Available: http://money.hinet.net/z/zd/zdc/zdcz/zdcz_C87A5222-EA76-4F94-9ADE-F500B3297AA9.djhtm