國立臺灣科技大學資訊工程系

110 學年度第一學期專題研究

總報告

Macro Legalization 巨集元件合法化

研究組員

B10732040 李宇哲

指導教授 :

中 華 民 國 101 年 01 月 22 日

國立臺灣科技大學資訊工程系

專題報告書

巨集電路合法化

Macro Legalization

指導教授: 劉一宇 教授

組員：李宇哲、林宇恩、林哲旭

中華民國一百一十年九月

目錄

摘要

隨著半導體製程技術不斷演進，現今的混合尺寸電路設計（mixed-size circuit design）使用大量可重複使用的巨集電路（macro，例如矽智產模組和嵌入式記憶體）於晶片中。相較於標準元件，巨集電路的尺寸龐大，加上有眾多的設計規則（design rule）需要納入設計考量，大幅提升混合尺寸電路擺置的複雜度。

本專題提出了一個巨集電路擺置演算法，我們的演算法是以降低對初始巨集電路擺置的擾動（perturbation）和提高後續標準元件的擺置空間（free space）為最佳化目標，自動化地將巨集電路擺置在晶片邊框內，擺置結果除了保證巨集電路之間沒有重疊外，還保留了最小通道距離以及周圍的緩衝元件區域以便後續時序（timing）修正。為了降低擾動程度，我們結合了遞移封閉圖（transitive closure graph, TCG）和線性規劃（linear programming），使巨集電路能以最小的位移來進行合法化，並且用 corner-stitching 資料結構有效表示晶片上的可用剩餘空間，來幫助位在密度過高區域的巨集元件找到鄰近適合的位置擺放。另外，我們利用 sweep-line 演算法來完成複雜的計算幾何問題，例如：處理非矩形的晶片外框、避開障礙物(預先擺放的巨集電路)和找出晶片中的可擺放標準元件空間(free space)等。

實驗結果顯示，面對複雜，甚至是巨集元件密度高於 90%的測試資料，我們的演算法皆能在短時間內，以微小的擾動量完成合法化及空間優化，並且獲得很好的結果。此外，我們的演算法在 2021 國際積體電路電腦輔助設計（CAD）軟體製作競賽中獲得第三名的佳績。

關鍵詞：超大型積體電路設計、實體設計、巨集電路、模組擺置、模組合法化

一、緒論

1.1 混合尺寸電路擺置介紹

隨著超大型積體電路不斷進步，系統級晶片的電晶體數量急劇上升，大型晶片除了包含數億個標準元件外，亦使用數千個預先定義、驗證，且可以重複使用的巨集元件，例如：矽智產模組和嵌入式記憶體，這種包含標準元件和巨集元件的電路稱為混合尺寸電路。

實體設計（physical design）流程中，混合尺寸電路擺置階段負責決定標準元件和巨集元件在晶片上的實體位置，擺置方式分為兩種，第一種是同時擺放巨集元件和標準元件，這種方法的缺點是標準元件容易占用過多的連續空間，導致大型的巨集元件找不到足夠的空間擺放，無法保證能找出沒有重疊的合法解。因此較有效的方法是第二種，將混合尺寸電路擺置分成三個階段完成，分別是混合尺寸電路雛型擺置（mixed-size prototyping）、巨集元件擺置（macro placement）和標準元件擺置（standard-cell placement），三個階段的介紹如下。

1. 混合尺寸電路雛形擺置

先根據線長（wirelength）、可繞度（routability）或功耗（power）等考量來決定巨集元件和標準元件在晶片上的初始散佈位置，此階段常見的演算法種類包含：基於分割演算法（partitioning-based algorithm）、數學解析法（analytical approach）和模擬退火（simulating annealing）。由於此階段尚未將擺置限制納入考量，所以初始擺置結果將違反元件不可重疊或超出晶片邊框等限制。

2. 巨集元件擺置

繼承前一步的初始擺置，此階段負責巨集元件的合法化工作，其中包括：巨集元件間不可重疊、不可超出晶片邊框、預留緩衝元件區域等，確保巨集元件沒有違反擺置限制，並且根據特定需求，往往是最小化原始擺置擾動量及最大化標準元件可擺放空間（free space），進一步優化巨集元件的擺放。由於巨集元件的尺寸龐大，加上晶片中經常存在預先擺置的巨集元件視同障礙物，使晶片不再是單純的矩形，加深了巨集元件擺置的複雜度。多篇論文針對

巨集元件擺置提出相關的資料結構及演算法，詳細描述請見 1.2 節，本文的演算法亦著重在這個階段。

3. 標準元件擺置

當巨集元件擺置位置固定後，此階段將數個等高度的標準元件擺入晶片的剩餘空間，大多的演算法是根據線長和擁擠度（congestion），安排標準元件到各個不同的列（row）上，再來將每列中的標準元件合法化，使單元元件間彼此不重疊且不可和巨集元件重疊。

三階段擺置刻意將巨集元件和標準元件的擺置問題分開，除了能更有效的優化各個階段的擺置，也能保證找到一組沒有重疊的解，因此這種分成三階段擺置的方法被業界所廣用。

1.2 文獻探討

在整個 IC 設計流程中，巨集元件的擺置結果對晶片效能有很大的影響，使巨集元件擺置成為一個重要課題，近年來有許著名的表示方法被提出，例如：B*-tree[1]、Multipacking-tree（MP-tree）[2]、Circular-packing trees（CP-trees）[3]和 Constraint Graph（CG）[4]等。B*-tree 使用二元樹（binary tree）結構來表示元件之間的位置關係，透過簡單的樹操作，如搜尋、插入和刪除可以改變元件之間的位置關係，B*-tree 的特色是將巨集元件不斷的往晶片左下角堆積（packing）擺置，使得晶片面積大幅的縮小，如圖 1(a)為一棵 B*-tree，n_i表示晶片上的巨集元件，令l_i、r_i分別表示n_i的左、右子節點，圖 1(b)為 B*-tree 轉換成實體擺置的結果，轉換方法如下：(1)根節點（root）n_0須擺放在最左下角的位置； (1)l_i的擺放位置在相鄰於n_i右邊的最低處；(2)r_i的擺放位置須緊鄰n_i上方且左下角的 x 座標須對齊，按照上述方法轉換完的結果將會往左下角壓實。

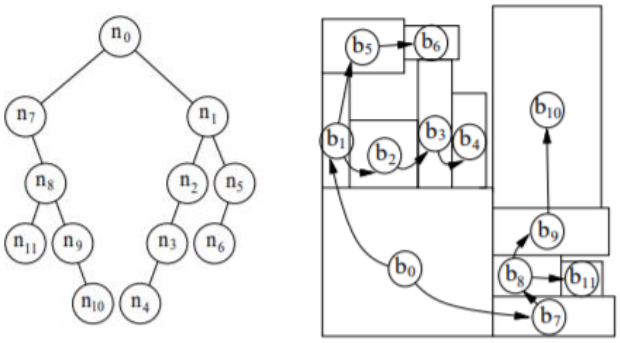


圖 1：B*-tree 轉換成實體擺置範例。(a) B*-tree。(b) B*-tree 轉換成實體擺置。

MP-tree 延續 B*-tree 往角落堆積擺置的概念，為晶片的四個角落（左上、左下、右上和右下角）分別建立一棵堆積子樹（packing subtree），圖 2 為一棵 MP-tree，由 BL、TL、TR 和 BR-packing subtree 構成（其中，BL-packing subtree 即為上一段所述的 B*-tree），讓巨集元件往四個的角落堆積擺置，目的在於最大化晶片的中間區域給後續的標準元件擺置和繞線，然而，MP-tree 並沒有把障礙物（如預先擺置巨集元件）表示成節點放入子樹中，因此每擺入一個巨集元件都要額外檢查它是否和障礙物重疊，如果有，則需水平或垂直的移動巨集元件到最近的位置來移除重疊，圖 3 為一範例，淺藍色為巨集元件，紅色為障礙物，當巨集元件 a 擺入後發現與障礙物重疊，則向右移到障礙物的右邊來消除重疊。

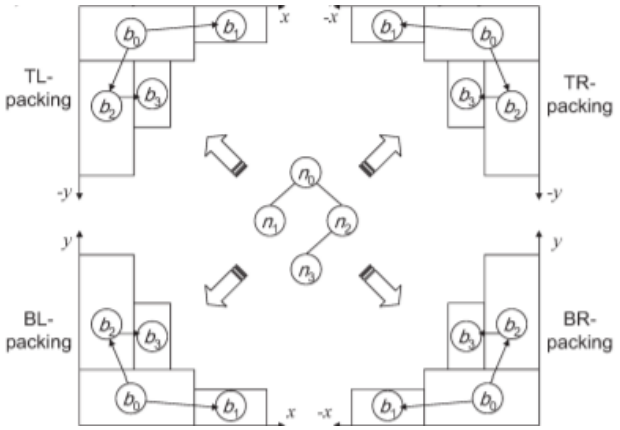


圖 2 MP-tree

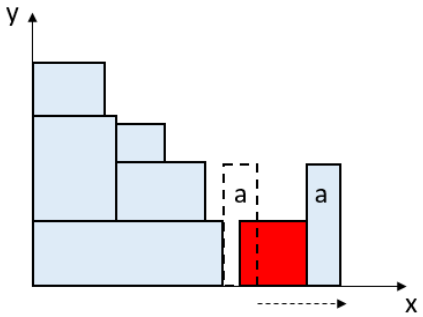
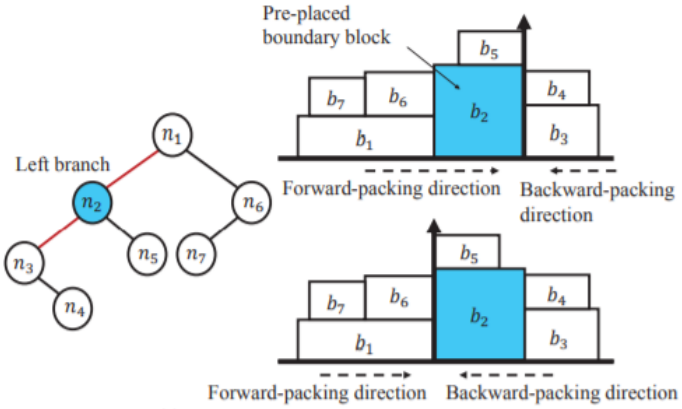


圖 3 MP-tree 解決障礙物重疊問題的範例。

CP-tree 為了更有效率的處理障礙物，將障礙物表示成節點納入堆疊子樹中，如此一來，便可以讓巨集元件往角落或者障礙物的四周擺置，提高擺置彈性，圖 4(a)中，CP-tree 左子樹的n_2節點表示障礙物b_2，根據深度優先搜尋，會先擺放左子樹中的n_1、n_2和n_3，針對左子樹中每個節點的壓實方向分為 forward 和 backward，在障礙物之前被搜尋到的為 forward，反之則為 backward，障礙物本身可以是 forward 或 backward，剩餘節點的壓實方向則跟所屬的父點一樣如圖 4(b)，因此，b_1、b_6和b_7的壓實方向為 forward，b_3和b_4的壓實方向為 backward，圖 4(b)上圖表示b_2和它的右子樹的壓實方向為 forward，圖 4(b)下圖表示b_2和它的右子樹的壓實方向為 backward。



(a) (b)

圖 4 CP-tree 處理障礙物（預先擺置巨集元件）問題的方法。(a) CP-tree。(b) CP-tree 對應的兩種實體擺置，上圖為 Forward-packing direction，下圖為 Backward-packing direction。

有別於前述 B*-tree、MP-tree 和 CP-tree 沿著角落邊界壓實（compacted）擺置的方式，CG 基於遞移封閉圖清楚地描述兩兩巨集元件之間的約束關係，並結合線性規劃決定出巨集元件

非壓實（non-compacted）的擺置結果，大幅減少對初始擺置的擾動程度，且能夠有效避開障礙物。

1.3 研究動機與研究概要

在巨集元件尚未被廣泛使用的年代，工程師主要以人工的方式來決定巨集元件在晶片上的擺放位置，然而，隨著積體電路規模的擴大，混合尺寸電路中的巨集元件數目已高達數千個，且為了因應後續實體設計（physical design）流程和製程的需求，有越來越多的設計規則（design rule）需要遵守，使得巨集元件擺置複雜度早已高出人力所及的上限。為了解決上述問題、加快設計流程和更精確的優化晶片擺置，工程師必須依賴電子設計自動化（electronic design automation, EDA）工具來完成擺置工作。

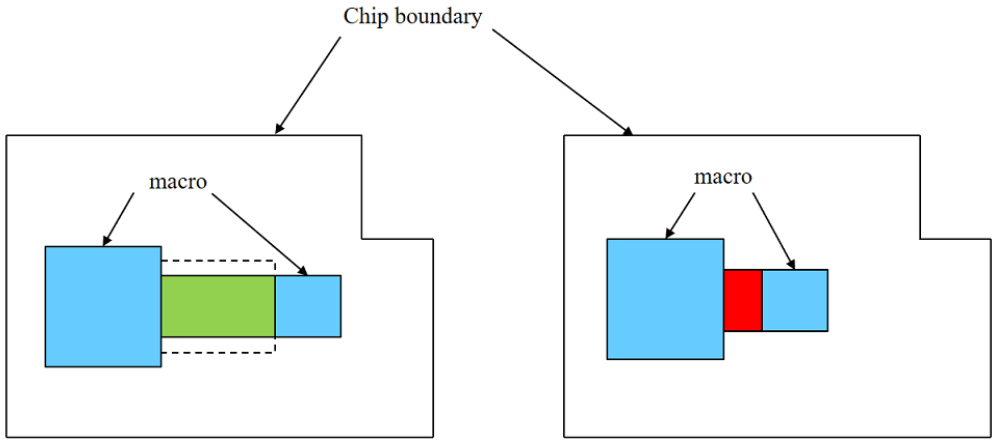
本文演算法專門處理巨集元件的合法化及優化問題，利用線性規劃技術使巨集元件僅用微小的擾動量來移除重疊部分，然而，傳統上將不重疊條件式（non-overlapping constraint）代入線性規劃求解的過程相當費時，為了提高效率，我們利用遞移封閉圖限制住巨集元件間的相對位置，使線性規劃的可行解區域（solution space）縮小，加快求解時間，且遞移封閉圖表示法能夠輕易處理特定的擺置需求，例如將巨集元件貼齊晶片邊界擺放或是避開預先擺置之巨集元件。我們亦使用隅角編織（corner stitching）資料結構來處理密度較高的晶片擺置，其特性是將晶片的剩餘空間（多邊形）切割成多塊矩形，並記錄矩形之間的實體位置關係，利用相關的幾何運算，能夠有效找出巨集元件周圍鄰近的可擺放區域。另外，我們利用掃描線（sweep line）演算法來偵測不規則的晶片外框，並用預先擺置之巨集元件將外框填補成矩形，降低設計複雜度。

二、專有名詞介紹

此章節將文中所提到的專有名詞（terminology）作概括性的介紹：

（1） Powerplan width：電源網路寬度，單位為 micron。

（2） Dead space / Free space（不可/可擺放 standard cell 空間）：在晶片內，巨集元件以外的任一空間，如果無法找到一個長與寬皆大於等於電源網路寬度限制的矩形涵蓋此空間，則此空間即為 dead space，如圖 5(a)中，兩個相鄰 Macros 之間的通道空間（綠色區塊）找的到一個長與寬皆大於等於 powerplan width 的矩形（虛線）涵蓋之，所以綠色空間為 free space，反之則為 free space，如圖 5(b)中，兩個相鄰 Macros 之間的通道空間（紅色區塊）找不到一個長與寬皆大於等於 powerplan width 的矩形涵蓋之，所以紅色空間為 dead space。



（a） （b）

圖 5（a）Free space（b）Dead space。

（3） 緩衝元件區域保留（buffer area reservation）限制：為了後續時序（timing）修正需要，每一個巨集元件的外擴固定距離範圍內，必須至少有一點與可擺置標準元件的空間重疊。

（5） 最小通道距離（minimum spacing）限制：左右（或上下）兩兩相鄰的巨集元件（包含可移動及不可移動的巨集元件），其在 X 方向（或 Y 方向）的通道必須保留的最小距離。

三、 演算法步驟及方法

3.1 演算法步驟

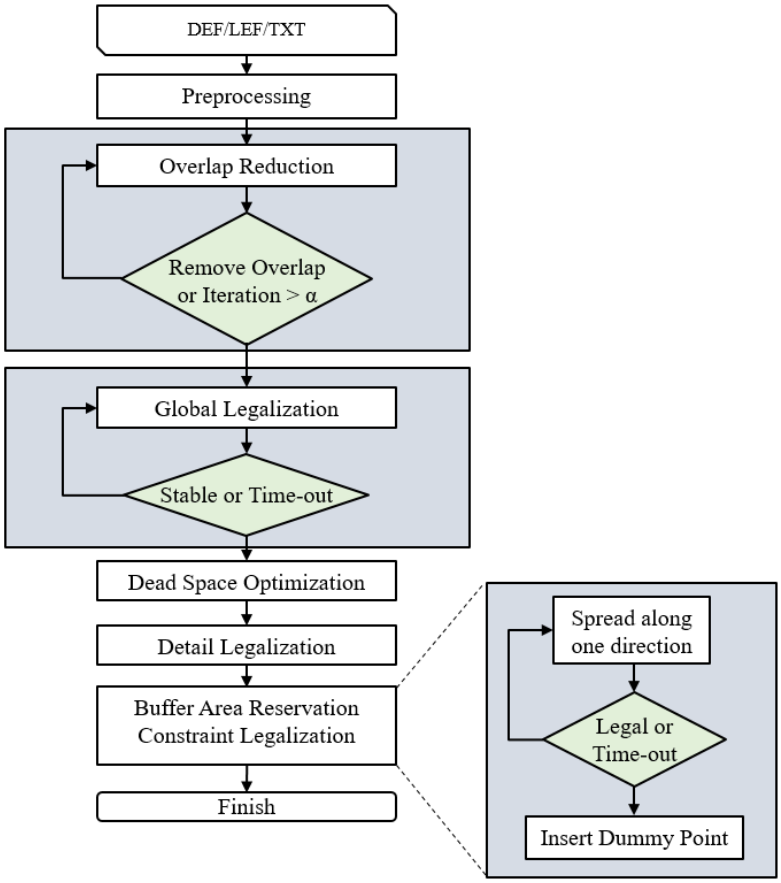


圖 6 演算法流程圖

圖 6 是我們的演算法流程，一開始讀入 DEF、LEF、TXT 檔，得到初始擺放（initial placement）位置和限制（constraint）資訊。Preprocessing 階段會利用 sweep-line 演算法把超出晶片邊框的巨集元件移至最近的邊框內，並且額外用不可移動的巨集元件（fixed macro）把邊框填補成矩形，以降低問題的複雜度。Overlap Reduction 是使用類似 Force-directed 的演算法以宏觀（global）的角度將重疊的巨集元件彼此推開，會重複執行直到沒有重疊或迭代次數超過一個門檻值為止，此時，仍可能有部分巨集元件違反重疊限制。Global Legalization 階段會根據前一步的擺置結果建出水平和垂直的約束圖（constraint graph），並且結合線性規劃，盡量找出不重疊（non-overlapping）且位移量總和（total displacement）最小的解，此時，仍可能有部分巨集元件違

反重疊限制。Dead Space Optimization 階段同上一個步驟，額外將某些巨集元件之間的距離拉大，試圖減少不可擺置標準件的通道空間。Detail Legalization 階段利用 corner stitching 資料結構和相關演算法將之前違反重疊限制的巨集元件一一擺進合法位置。在這之前，都尚未考慮緩衝元件區域保留（buffer area reservation）限制，因此最後 Buffer Area Reservation Constraint Legalization 階段利用 sweep-line 演算法找出所有違反此限制的巨集元件，針對這些巨集元件，往密度最小的方向擴散試圖將它們合法化，如果依然不行，就利用 corner stitching 結合插 dummy buffer 的方式來合法化。

3.2 研究方法

1. Preprocessing

此步驟是為了處理初始擺放階段會遇到的兩種情況：

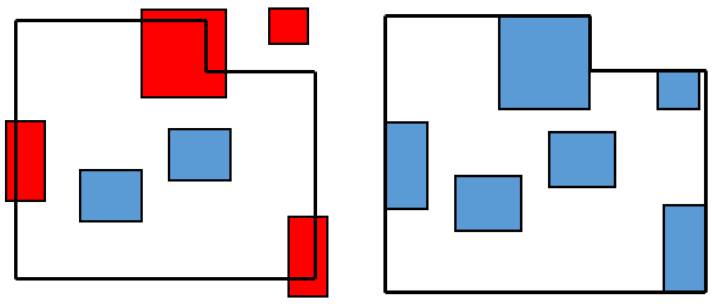
（1） 有巨集元件超出晶片外框

（2） 晶片外框不是矩形，而是正交多邊形

遇到的話，會分別用以下方式解決:

（1） 將巨集元件移至最近的邊框內

初始擺放中，可能會有巨集元件超出晶片邊框，而違反 fixed-outline 規則，如圖 7（a）的紅色元件，所以我們利用 sweep-line 演算法[5]找出這些巨集元件，並將它們移至距離最近的邊框範圍內，如圖 7（b）。

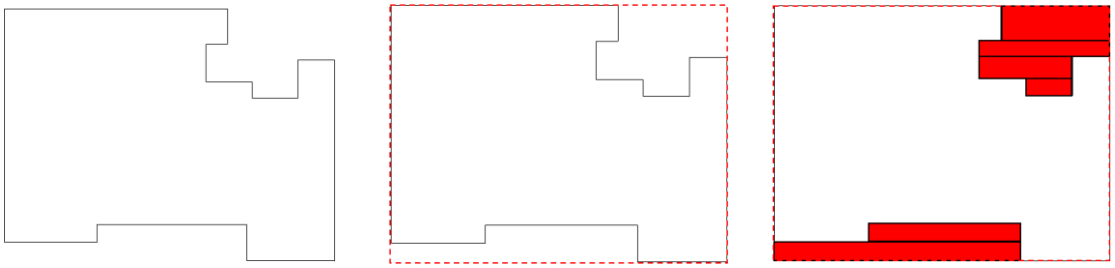


(a)　　（b）

圖 7 (a)違反 fixed-outline 規則擺置。(b)符合 fixed-outline 規則擺置

（2） 將晶片外框填補成矩形

當晶片外框不是矩形，而是正交多邊形時如圖 8（a），會大幅增加問題的複雜度，因此我們會以正交多邊形的最小 Bounding box 最為新的晶片邊框，如圖 8（b）的紅色虛線，並且用 sweep-line 演算法找出 Bounding box 之內，原邊框之外的區域，用不可移動巨集元件（fixed macro）將其填滿，我們稱這些不可移動巨集元件為 Boundary macro，如圖 8（c）的紅色實心矩形。



（a） （b） （c）

圖 8（a）晶片外框（b）晶片的 bounding box（c）插入 fixed macro

2. Overlap Reduction

此步驟運用類似 Force-Directed 演算法，以宏觀（Global view）的方式計算出巨集元件之間的排斥力，使得重疊的巨集元件得以根據排斥力擴散開來，減少重疊比例，進而讓巨集元件之間的方向關係變得更加明確，以便後續演算法在優化時有更好的表現。

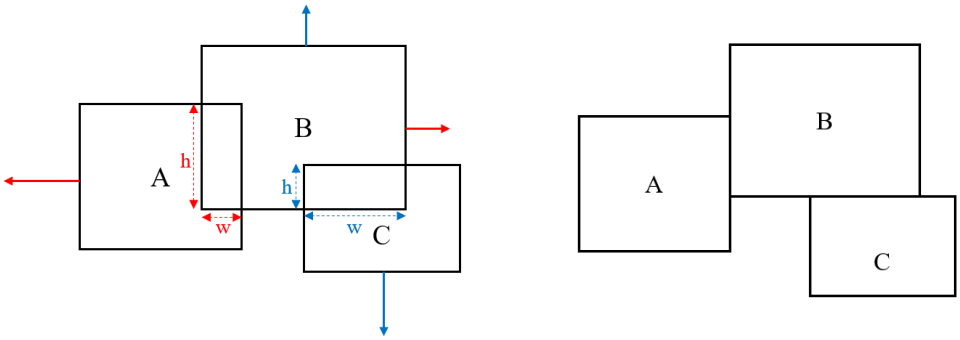
我們修改了 Force-Directed 演算法，只用到了自己定義的排斥力，令 M_i 表示巨集元件，$Overlap_i$ 為所有和 M_i 重疊的巨集元件集合，則 M_i 受到的排斥力合力 F_i 計算如下:

$$O_{ij} = min\left(O_{ij}^{W}, O_{ij}^{H}\right) \quad (1)$$

$$F_i = \sum_j \frac{Area_j}{Area_i + Area_j} \times O_{ij} \quad \forall M_j \in Overlap_i \quad (2)$$

（1）式中，O_{ij}^{W}和O_{ij}^{H}分別表示 M_i、M_j 重疊區塊的長度和寬度，如果$O_{ij} = O_{ij}^{W}$，則排斥力為水平方向，相反地，則為垂直方向。（2）式中，$Area_i$（$Area_j$）表示 M_i（M_j）的面積大小。任意兩個巨集元件可能有重疊或沒有重疊，如果沒有重疊，則它們之間不會產生排斥力，相反

的，則會產生排斥力，而且當面積一大一小的巨集元件在互相推動時，根據公式，面積小的會移動較大，面積大的則移動較小，以圖 9 為例，A、B 和 C 巨集元件的面積分別為 10、14 和 6，圖 9(a)中，A 和 B 重疊區域的寬度（紅色 w）小於高度（紅色 h），因此 AB 之間的推力方向為水平（紅色箭頭），A 受到的推力大小為$\frac{14}{10+14}$，B 受到的推力大小為$\frac{10}{10+14}$； B 和 C 的重疊區域中，高度（藍色 h）小於寬度（藍色 w），因此 BC 之間的推力方向為垂直（藍色箭頭），B 受到的推力大小為$\frac{6}{14+16}$，C 受到的推力大小為$\frac{14}{14+16}$，最後巨集元件受到推力移動後的結果如圖 9(b)。



(a)　　（b）

圖 9（a）執行 Overlap Reduction 前的擺置（b）執行 Overlap Reduction 後的結果。

傳統上想要求得 *Overlap* 集合是使用兩兩比較的方式，需要 $O(N^2)$ 的時間複雜度，相當費時，因此本文提出了改良方法，利用 Sweep line algorithm 取代傳統方法，由於 Sweep line algorithm 背後的資料結構是 Interval tree(用 AVL tree 實作)，因此時間複雜度只要 *O(NlogN)*，能有效降低 run time。

3. Global Legalization

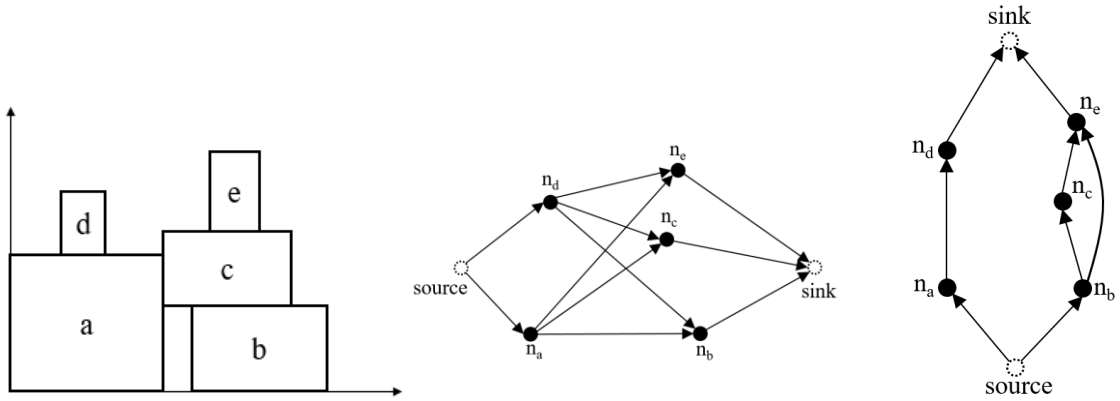
前一步做完的結果只是將重疊巨集元件以力推開，但並不保證完全沒有重疊，所以這步運用 TCG （Transitive Closure Graph）[6]和線性規劃技巧，盡量讓所有巨集元件不重疊，並且最小化可移動巨集元件的位移長度總合。之所以要先建立 TCG 是因為單純用不重疊限制

（non-overlapping constraint）來解線性問題的話非常耗時，所以我們先利用 TCG 將巨集元件之間的關係限制住，達到縮小可行解區域（solution space）的目的，有效降低執行時間。

(1) TCG （Transitive Closure Graph）

將 Global Macro Placement 完的擺置結果轉成水平和垂直的 TCG，分別表示成 G^H 和 G^V，如圖 10，擺置轉成 TCG 的方法請參考[6]，TCG 的頂點 n_i 代表巨集元件，邊的權重 e_{ij} 代表邊的兩個端點不重疊至少要有的距離長度（non-overlapping distance）（要注意的是，這個距離必需額外加上最小通道距離，以符合最小通道距離限制），所以本文的 TCG 同時也是一個 CG（Constraint Graph），每一條邊都對應一組 Non-overlapping constraint。

另外，為了防止巨集元件按照上下左右關係展開來後會超出晶片邊框，我們利用拓樸排序（topology sort）找出 TCG 的最長路徑（critical path）長度，當最長路徑長度大於邊框時，會從最長路徑中挑出面積最小的巨集元件自 TCG 中刪除，直到最長路徑長度小於等於晶片邊框為止，剩下的頂點會進到下一個步驟算出實際的座標位置，剛剛被刪除的頂點則是會在 E 步驟被決定座標位置。



（a） （b） （c）

圖 10 （a）擺置結果 （b）G^H （c）G^V

(2) LP （Linear Programming）

將前一步 TCG 的邊轉成 LP 問題的約束條件（Constraint）（3），以最小位移（Displacement）總和作為 LP 問題的目標函數（Objective function）（4），有了約束條件和目標函數後，便可將它們代入 LP solver[7]來決定巨集元件的實際座標位置。

$$\min \quad \sum_{i=1}^{n} d_{x_i} + d_{y_i} \tag{3}$$

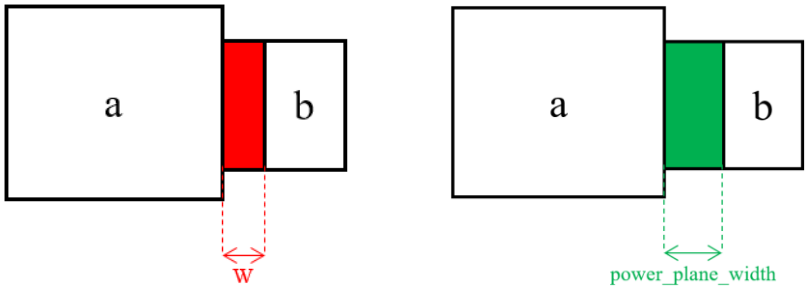
$$\begin{aligned} \text{s.t.} \quad & x'_j - x'_i \geq e_{ij} \quad if \ \exists e_{ij} \in G^H \\ & y'_j - y'_i \geq e_{ij} \quad if \ \exists e_{ij} \in G^V \\ & x'_i = x_i \qquad if \ n_i \ is \ FIXED \\ & y'_i = y_i \qquad if \ n_i \ is \ FIXED \end{aligned} \tag{4}$$

完成以上兩個步驟後，（1）步驟中沒有被刪除的頂點即可被決定確切位置，被刪除的頂點位置則維持在最後的位置（global legalization 完後的位置）。

4. Dead Space Optimization

確定巨集元件的座標位置並且優化最小位移（displacement）總和後，接著要來優化的是不可擺放標準元件空間（dead space）。當兩個相鄰巨集元件 x 軸或 y 軸距離小於電源網路寬度（powerplan width）時，它們之間的通道空間必為不可擺放標準元件空間，我們利用這個特性，適當的將這些巨集元件之間的距離拉開到電源網路寬度的距離，有機會減少很多 dead space，如圖 11(a)兩個聚集元件 a 和 b 的 x 軸距離為 w，小於電源網路寬度，因此中間通道（紅色區塊）為不可擺放標準元件空間，為了優化這塊空間，如果 *w* 大於 $\alpha \times$ *powerplan width*，就將這兩個聚集元件的距離拉開成 *powerplan width*，有很高的機會讓中間通道變成可擺放標準元件空間（綠色區塊），如圖 11(b)。

Dead space optimization 的作法非常類似 global legalization，唯一不同的是 Placement 結果轉成 TCG 時，如果邊的兩個端點的距離大於 $\alpha \times$ *powerplan width*（α 為參數 0~1，我們設 0.6），則該條邊的權重 e_{ij} *設為 powerplan width*，否則，e_{ij} 設為 non-overlapping distance，實際用 CAD 競賽提供的 10 筆測資做測試，結果顯示 dead space 面積皆減少 15%以上。

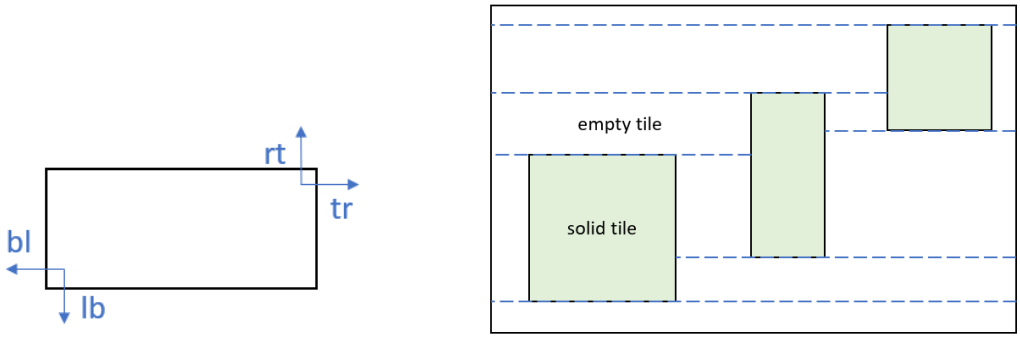


（a） （b）

圖 11（a）執行 Dead space opt.前。（b）執行 Dead space opt.後。

5. Detail Legalization

在提出如何解決 overlap constraint 前，先說明使用到的資料結構 corner stitching[8]，corner stitching 是表示一群矩形(也稱作 tile) 在 2D 平面上實體位置關係的資料結構，最初的設計目的是用在 interactive VLSI layout tool 的內部資料結構，用以回應使用者對物件的操作(像是使用者點擊物件時，軟體如何快速找到物件且做相對應事情)。corner stitching 有兩大特色：(1)空白區域(未被物件占用區域)會被明確的表示成空白 tile；(2)所有 tile 藉由自身的 corner 來連結其他 tile，如圖 12(a)所示每一個 tile 有四個指標代表其四個鄰居：rt(最右邊的上方 tile),tr(最上面的右方 tile)，bl(最下面的左方 tile，lb(最左邊的下方 tile)。前面提及的空白 tile 需要滿足 horizontally maximal，指任意兩個空白 tiles 不會共享同一垂直邊，如圖 12(b)的例子 corner stitching 表示擺放三個 solid tile 的結果，而剩餘空間則用多塊 empty tile 表示。根據這樣的資料結構一些 geometric operations 能夠很有效率地完成，我們使用的 operations 有(1)point finding：給定一個座標點找到包含此點的 tile；(2)area searches：給定一個矩形範圍，找到與此範圍有交集的 tiles；(3)tile creation：插入一個 solid tile，依情況更新周圍的空白 tiles(切割或合併)；(4)area fill：給定一個矩形範圍，把範圍內的空白 tile 利用 tile creation 把空白 tile 與範圍交集區域更改為 solid tile。



(a) (b)

圖 12（a）tile 的四個指標。（b）corner stitching 結果圖。

Detail legalization 階段要把在 global legalization 中未考慮的 macro 找到合法位置，tile 的類型分成五種：invalid, space, macro, boundary, buffer，invalid 代表此區域是在晶片外面；space 代表此區域是合法位被占用區域；macro 代表此區域是屬於某一個 macro；boundary 代表 preprocessing 階段產生的邊框矩形；buffer 則代表此區域是可擺置標準元件空間，此階段的流程圖如圖 13，(1)tile plane construction：首先創建一個 tile plane，使用 tile creation 擺放跟晶片 bounding box 大小的 tile(類型為 space)；(2)chip boundary construction：使用 tile creation 擺放所有在 preprocessing 階段產生的邊框矩形(類型為 boundary)；(3)pre-placed macro placement：使用 tile creation 或是 area fill 擺放所有的 pre-placed macro 和前面步驟已經合法的 macro；(4)macro legalization：未合法的 macro 找到合法位置。

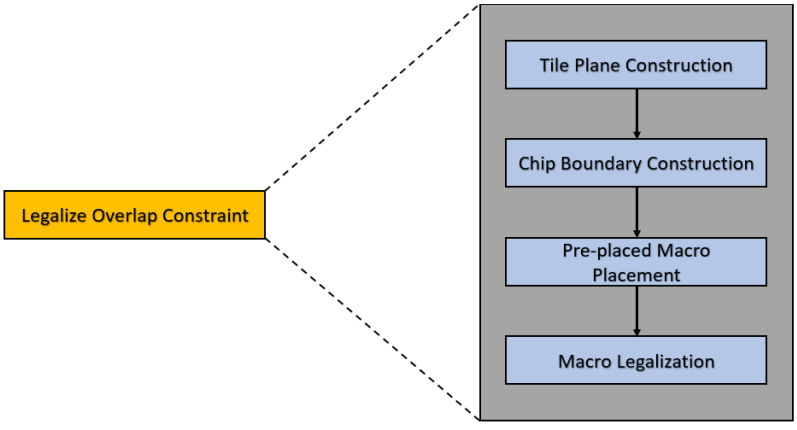
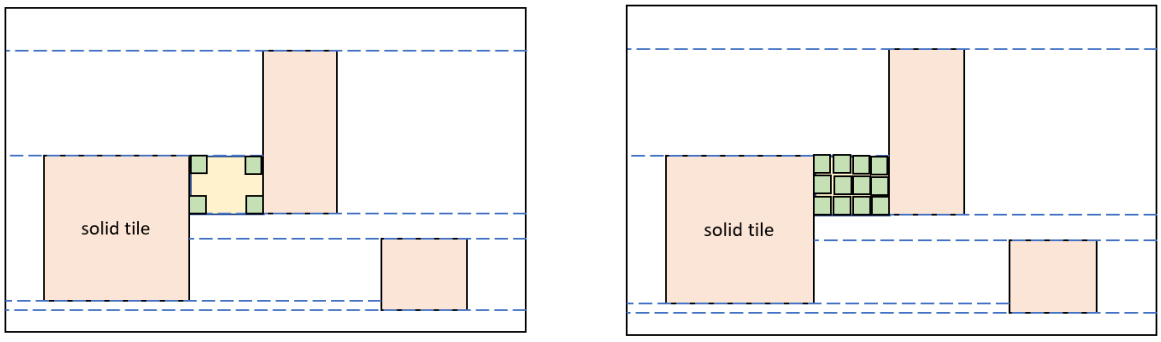


圖 13 Detail legalization 分成四個步驟：tile plane construction, chip boundary construction, pre-placed macro placement, macro legalization。

第四步驟的 macro legalization，macro 會先經過排序依照排序一個接著一個擺放，排序的依據是 macro 的面積大小以及靠近邊界的距離，我們的論點是面積越大的 macro 優先擺放，原因是大面積的 macro 較晚擺放時，剩餘的擺放空間沒有足夠大的連續空間做擺放，導致需要 rip up and replace；而越靠近邊界的 macro，應優先擺放的依據是讓 macro 由外往內擺放，以減少 dead space 的產生。決定好 macro 的擺放順序後，要找到合法的位置擺放，作法如下：(1)根據 tile plane 找到所有的空白 tiles；(2)利用 tiles 的四個角落當作錨點，用 area search 檢查 macro 大小範圍內是否全部都是空白 tiles，如果是則要進一步檢查外擴巨集元件間的最小通道距離(一樣使用 area search)內是否沒有其他 macro，兩個條件都符合下則是合法位置，如果有一個以上的合法位置，則會挑選距離初始位置最接近的點。在搜尋可能的位置時可以依照需求，選擇要考慮 runtime 還是 solution quality，兩種做法如圖 10 所示，以黃色空白 tile 為例，圖 14(a)在考慮 runtime 下，只會搜尋 tile 的四個角落(綠色方塊)；圖 14(b)考慮 solution quality 下，用 macro 寬或高的距離往上下左右搜尋直到超出空白 tile 的範圍(所有的綠色方塊都是合法位置)。當所有 macro 都找到合法位置後此階段結束，圖 15 為 corner stitching 的擺放結果，其中淡紅色為 macro，灰色為空白區域，黃色為邊界區域。



(a) (b)

圖 14（a）考慮 runtime 下的搜尋位子。（b）考慮 solution quality 下的搜尋位子。

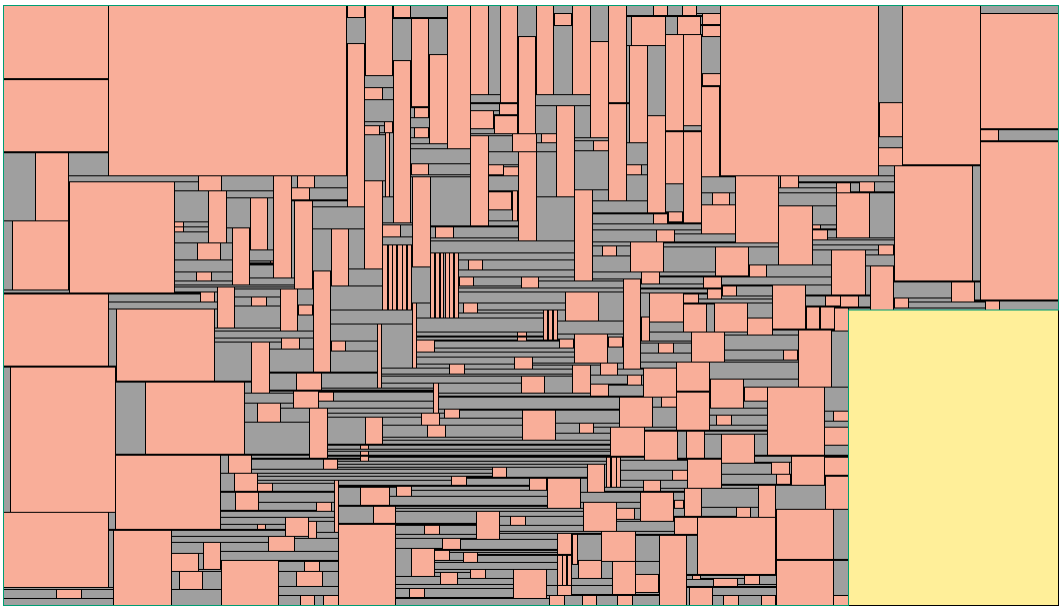


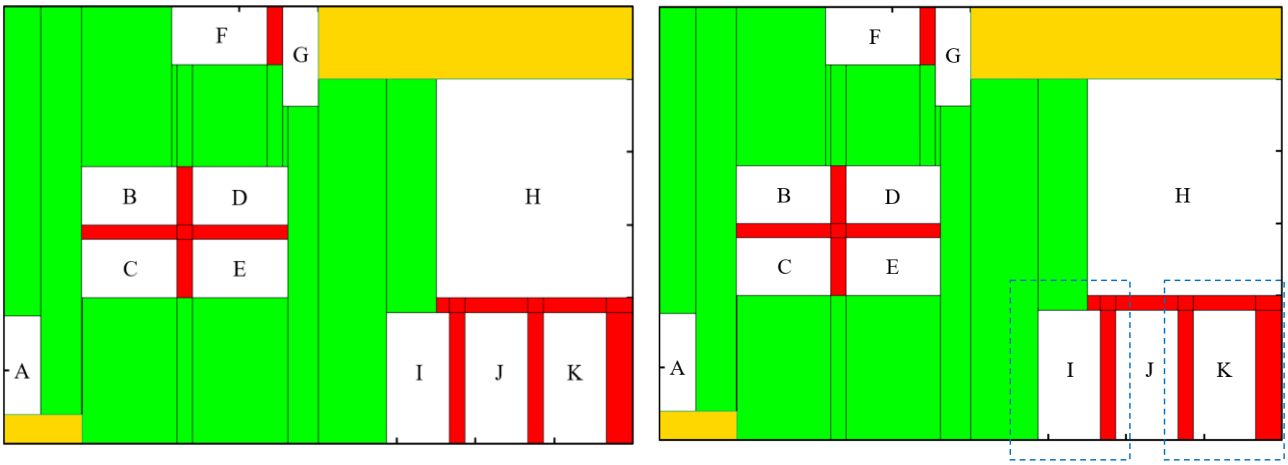
圖 15 為 corner stitching 結果。

6. Buffer Area Reservation Constraint Legalization

在這之前，都尚未考慮緩衝元件區域保留限制，因此，這個階段會利用以下兩個步驟使擺放滿足此限制，(1)Spread along one direction：希望以最小的位移變化量來合法化該限制;(2)Insert Dummy Point：若前一步無法將所有巨集元件合法化，則會在這一步解決，缺點是對原擺置的變動較大。

（1） Spread along one direction

首先，利用 sweep-line 演算法掃過整個晶片找出所有的 free space，如圖 16(a)所示，白色為 macro，黃色為邊界區域，綠色為 free space，紅色為 dead space，再來依序檢查每一個 macro 的外擴電源網路寬度範圍內是否有和任何一個 free space 重疊，如果沒有，則該 macro 即違反緩衝元件區域保留限制，我們稱這些違反限制的巨集元件為 illegal macro，如圖 16(b)中 macro I 的外擴電源網路寬度範圍為左邊藍色虛線區域，並且有和 free space 重疊，故 macro I 遵守緩衝元件區域保留限制; macro K 的外擴電源網路寬度範圍為右邊藍色虛線區域，沒有和任何一個 free space 重疊，故 macro K 違反緩衝元件區域保留限制。

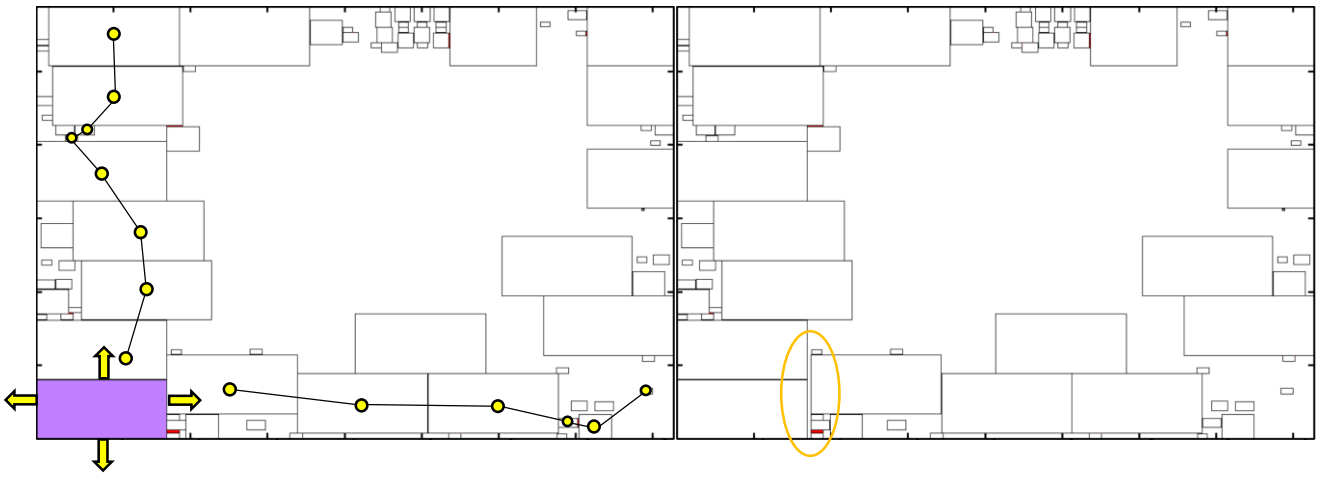


(a) (b)

圖 16 (a)sweep-line algorithm 結果圖。(b) 緩衝元件區域保留限制合法判斷。

接下來，高機率讓這些 illegal macro 變成合法的方法流程如下：(1)找出 illegal macro 的上、下、左、右方向中最不擁擠的方向，以圖 17(a)為例，紫色為 illegal macro，紅色為 dead space，以紫色 illegal macro 作為起點，往四個方向（黃色箭頭）執行拓樸排序找出四個方向的 critical path（黑色線段）之長度，並且分別算出他們和四邊邊框之間的距離差，越大則代表該方向越不擁擠，該圖顯示 illegal macro 的右方最不擁擠；(2)找到 illegal macro 四周最不擁擠的方向後，接下來如圖 17(b)示意圖，推開右方所有和 illegal macro 相鄰的巨集元件至少電源網路寬度的距離，使 illegal macro 與這些相鄰 macro 之間的通道空間有很高的機會轉成 free space(黃色圈起處)，則 illegal macro 的外擴範圍內必定和 free space 重疊，達到合法化目的。

然而，完成上述的步驟後會移動到其他的 macro，可能導致原本的 legal macro 變成 illegal macro，因此需重複執行 Spread out one direction，直到沒有 illegal macro 或是執行時間超過自訂門檻值為止，如果離開該函數後仍有 illegal macro 的話，就進到下一個步驟 Insert Dummy Point 去解決。



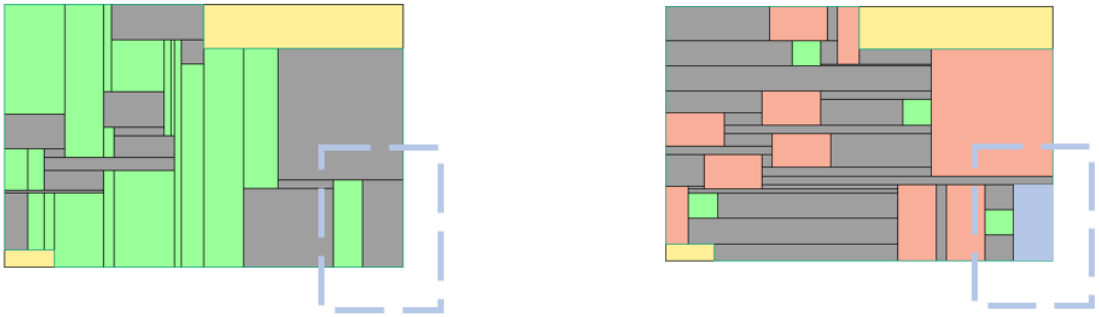
(a) (b)

圖 17（a）illegal macro 四周的 critical path。（b）Spread along one direction 示意圖。

（2） Insert dummy point

當前面的 spread along one direction 結束後，還有一個以上的 illegal macro 時，我們再度利用前面提及的 corner stitching，去找到滿足緩衝元件區域保留限制的合法位置，根據緩衝元件區域保留限制的定義，每一個 macro 在緩衝元件區域保留外擴距離範圍內至少有一點與可擺置標準元件的空間重疊要求，轉換成要求所有 macro 在緩衝元件區域保留外擴距離範圍內有 buffer area (tile 的屬性為 buffer，並且此位置無法被其他屬性的 tile 覆蓋)，面積為電源網路寬度的平方(滿足此限制的最小面積)，所以擺放任一個 macro 到 tile plane 之前，需要先檢查緩衝元件區域保留外擴距離內是否有 buffer area，如果有則滿足此限制；反之則近一步檢查此範圍內是否有可以擺放 buffer area 的位置，找的方法有兩種：(1)和 detail legalization 的第四步驟相同利用空白區域四個角落。(2)利用前一步驟產生的 free space(圖 16)創建另一個 tile plane(稱之為 buffer plane)來輔助找到合法插入 buffer area 位置，詳細說明以圖 18 為例，綠色區域為可擺置標準元件區域，圖 18 (a)根據前一步驟產生 buffer plane，接下來如圖 18(b)，藍色框線為藍色巨集元件的外擴範圍，當需要插入 buffer area 時，利用 area searches 在 buffer plane 的相同範圍找到所有的類型是 buffer 的 tiles，與 legalize overlap constraint 的第四步驟相同，使用 buffer tile 的四個角落尋找合法位置，而 cost function 則改成離 macro 擺放位置中心越近越好，最後在 macro 位置的中左邊插入 buffer area(綠色矩形)。

第一個方法的缺點是比較後面擺放的 macro 會被越多的 buffer area 影響，導致原本合法的 macro 需要另外找到別的合法位置，讓整體結果變差，所以我們採用第二種方法，只在 free space 尋找合法的 buffer area 位置，好處在於不會去影響原本已經是合法 macro 的擺放位置，當所有 macro 的外擴範圍內都有 buffer area 時，我們就完全解決緩衝元件區域保留限制。



(a) (b)

圖 18 (a) buffer plane。(b)插入 buffer area。

四、 實驗結果

我們的演算法使用 C/C++撰寫，測試平台的硬體規格為 Intel Core i7-8750H 2.20GHz 處理器和 16G 記憶體，測試資料共有 12 筆如表 1，其中 case1~case10 測試資料是由 *Maxeda Technology Inc.*公司提供，case11 和 case12 是我們特別生成的高密度測試資料。

我們的演算法輸入為一組可能違法設計規則的巨集元件雛型擺置（如圖 19~30 左側），透過我們的演算法，可以快速輸出合法且經優化的巨集元件擺置（如圖 19~30 右側）。實驗結果數據包含「可移動巨集元件的位移長度總合(displacement)」、「不可擺置標準元件的通道空間總和(dead space)」以及「目標值(target)」，如表 2，其中，目標值的計算方式為$\alpha \times displacement + \beta \times free\ \ space$，$\alpha$和$\beta$權重介於 0~-100000。

表 1 測試資料

測試資料	可移動聚集元件數量	不可移動聚集元件數量	聚集元件總數量
case1	84	0	84
case2	84	0	84
case3	294	0	294
case4	48	0	48
case5	81	0	81
case6	187	0	187
case7	101	0	101
case8	106	0	106
case9	253	0	253
case10	195	0	195
case11 (非官方)	898	6	904
case12 (非官方)	1015	0	1015

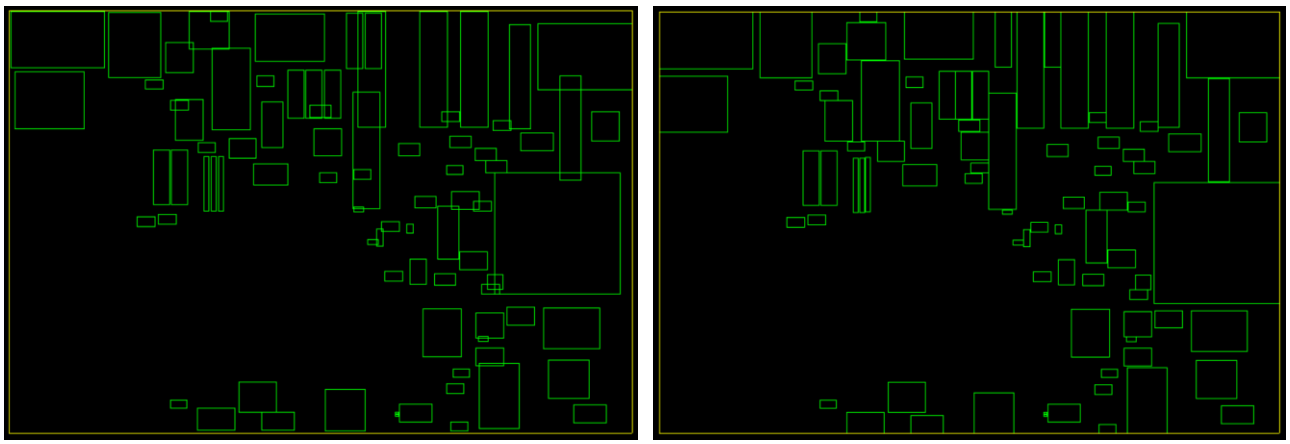


圖 19 case1

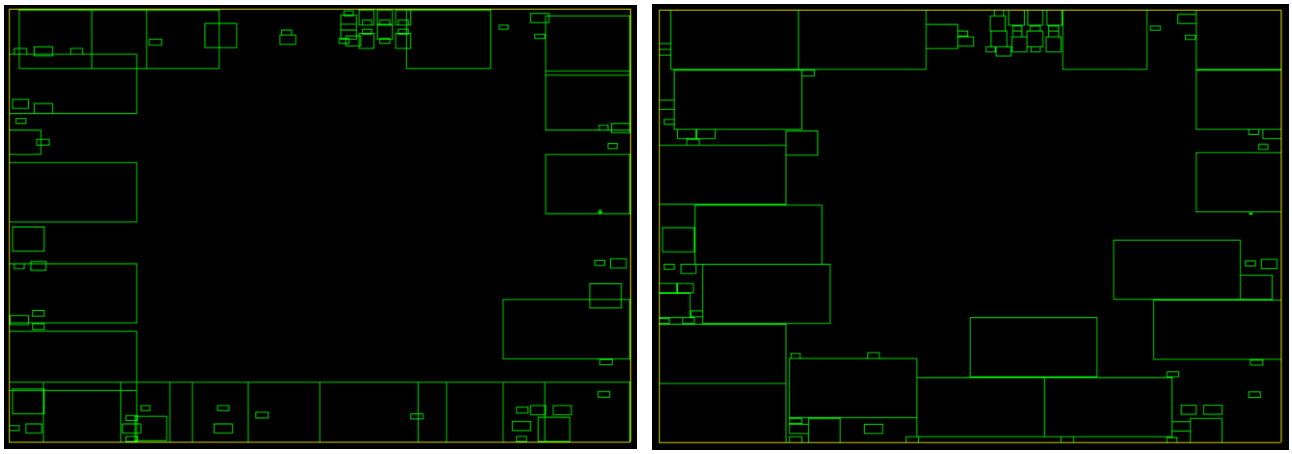


圖 20 case2

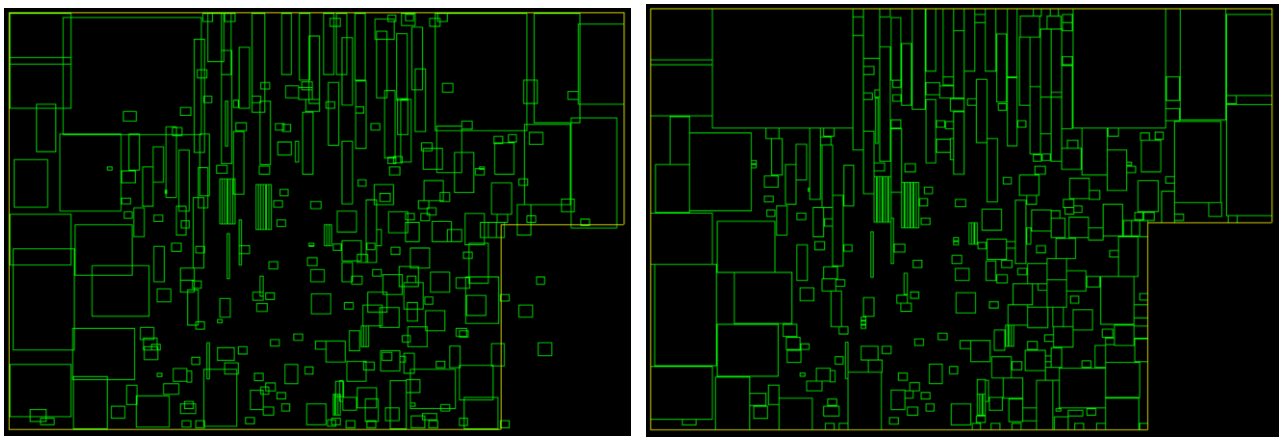


圖 21 case3

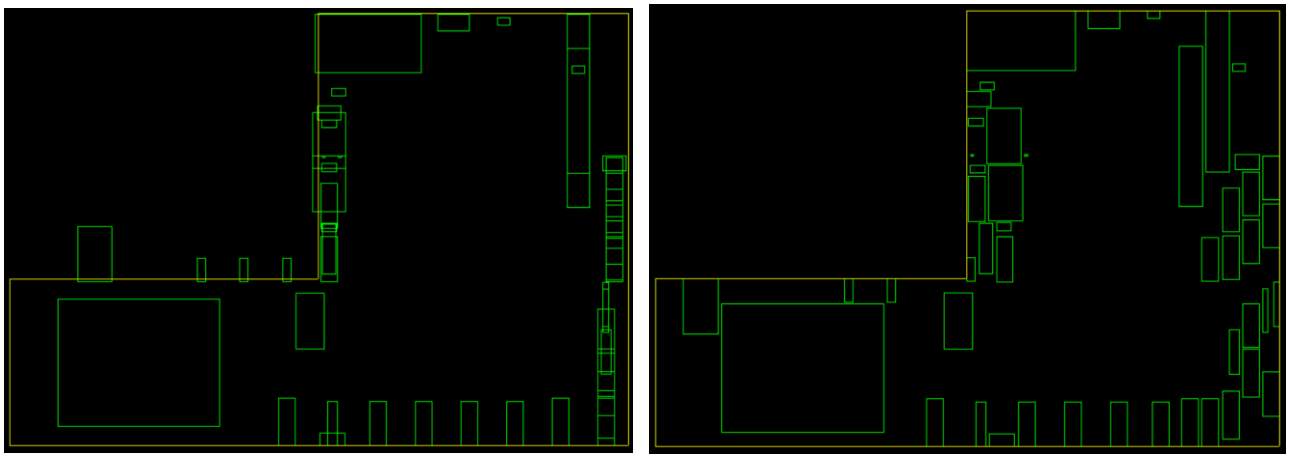


圖 22 case4

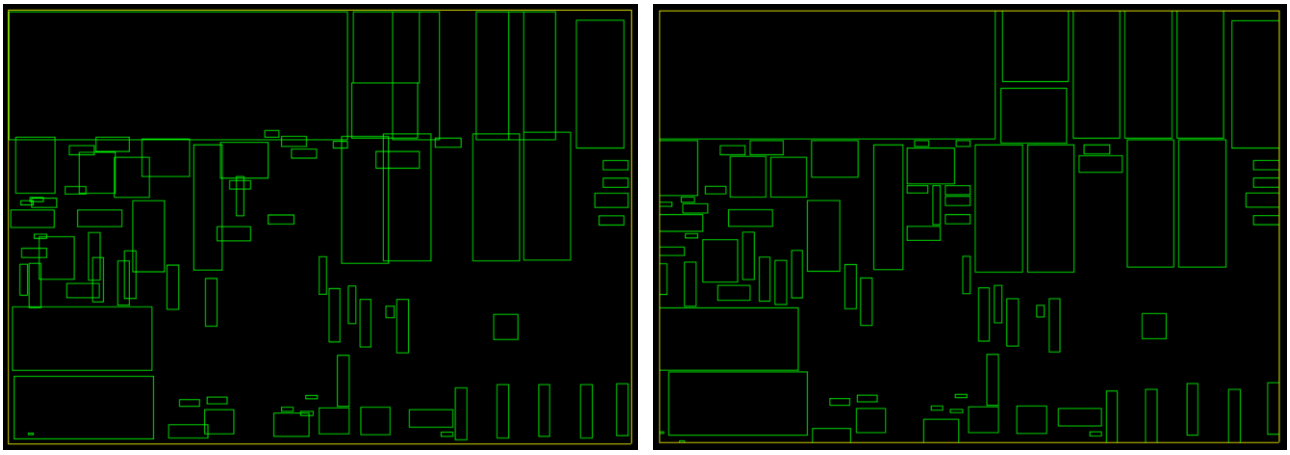


圖 23 case5

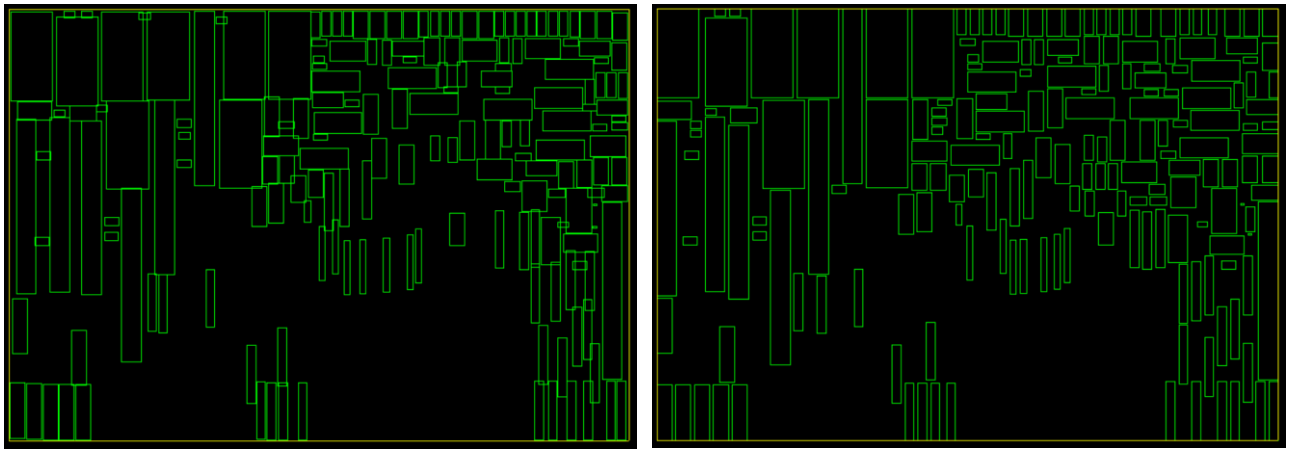


圖 24 case6

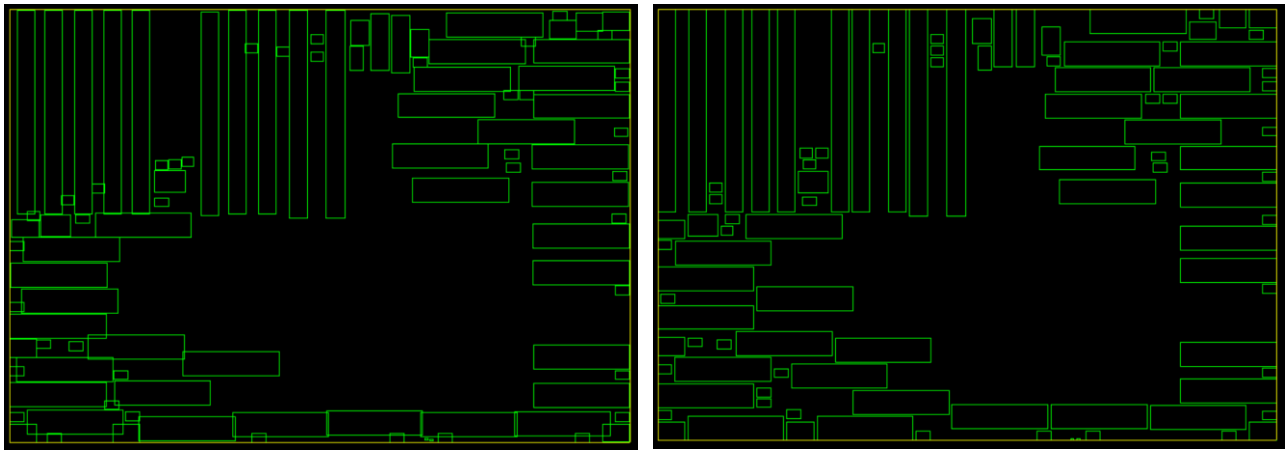


圖 25 case7

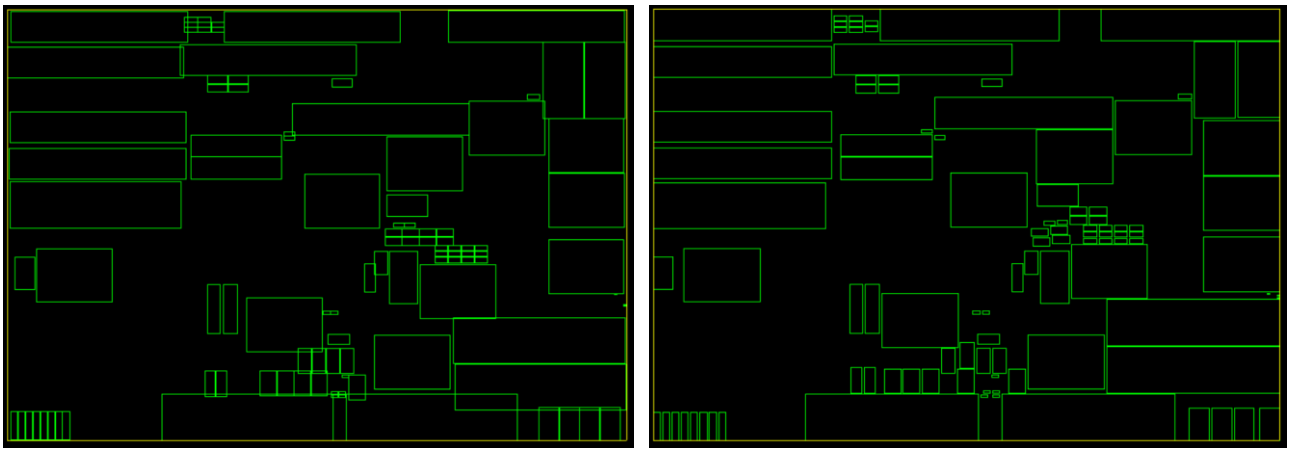


圖 26 case8

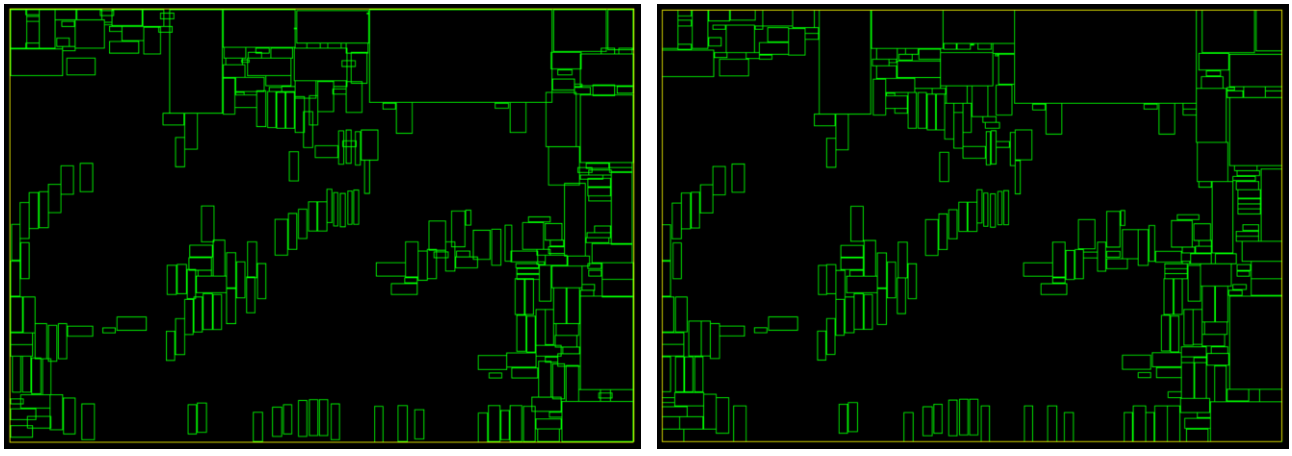


圖 27 case9

圖 28 case10

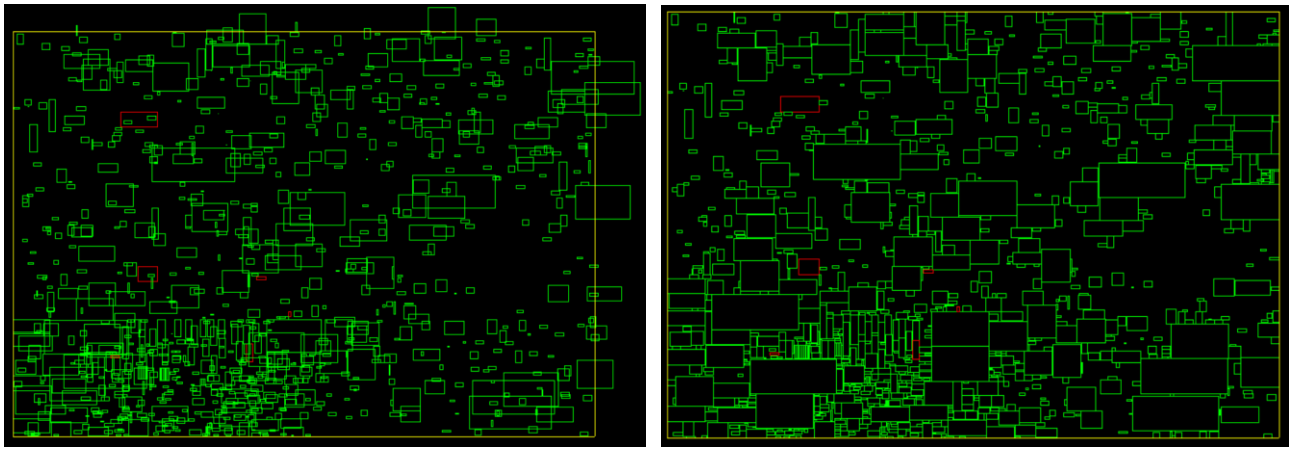


圖 29 case11

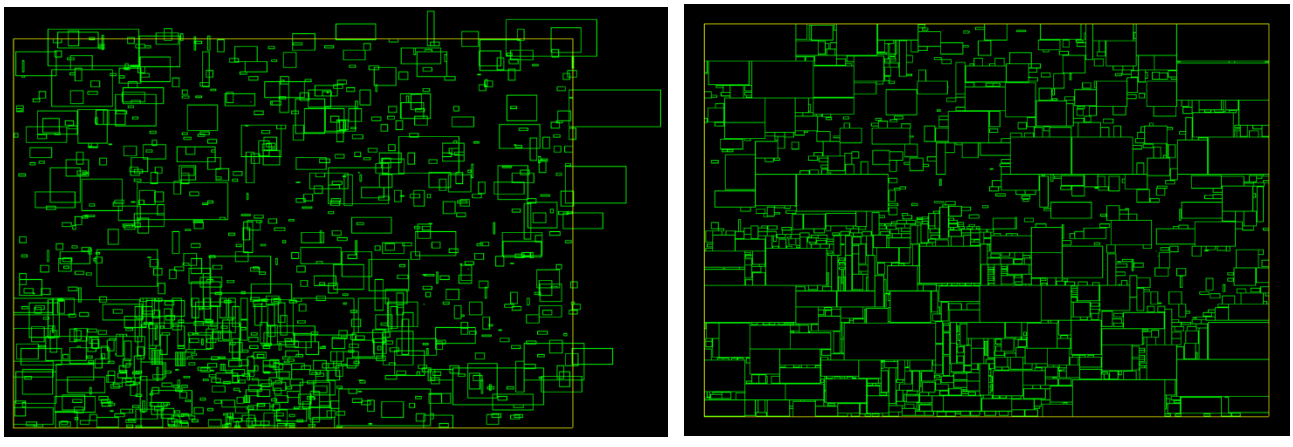


圖 30 case12

圖 19~30 為實驗結果圖，左半部為巨集元件的雛型擺放位置圖，右半部為執行完我們的演算法所得到的結果圖，使用 gnuplot 工具繪製。綠色為可移動之巨集元件，紅色為不可移動之巨集元件，黃色為晶片外框。

表 1 實驗結果數據

測試資料	可移動巨集元件的位移長度總合（micron）	不可擺置標準單元元件的通道空間總和（micron^2）	目標值
case1	2130	66593	3162.22
case2	28093	114488	38920.54
case3	21521	383400	100778
case4	2303	43287	73904.06
case5	2381	92627	41337.40
case6	12632	245548	28488.90
case7	3295	106870	45139.45
case8	5001	111967	5043670
case9	15214	557947	122458.96
case10	11362	408946	364223.49
Case11 (非官方)	388789	1556890	548501
Case12 (非官方)	1135630	2310090	1330170

五、 參考文獻

[1] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, and S.-W. Wu, "B*-trees: A new representation for non-slicing floorplans," in *Proceedings of the 37th Annual Design Automation Conference*, 2000, pp. 458-463.

[2] T.-C. Chen, P.-H. Yuh, Y.-W. Chang, F.-J. Huang, and T.-Y. Liu, "MP-trees: A packing-based macro placement algorithm for modern mixed-size designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 27, no. 9, pp. 1621-1634, 2008.

[3] Y.-F. Chen, C.-C. Huang, C.-H. Chiou, Y.-W. Chang, and C.-J. Wang, "Routability-driven blockage-aware macro placement," in *Proceedings of the 51st Annual Design Automation Conference*, 2014, pp. 1-6.

[4] H.-C. Chen, Y.-L. Chuang, Y.-W. Chang, and Y.-C. Chang, "Constraint graph-based macro placement for modern mixed-size circuit designs," in *2008 IEEE/ACM International Conference on Computer-Aided Design*, 2008: IEEE, pp. 218-223.

[5] U. Bartuschka, K. Mehlhorn, and S. Näher, "A robust and efficient implementation of a sweep line algorithm for the straight line segment intersection problem," in *IN PROC. WORKSHOP ON ALGORITHM ENGINEERING*, 1997: Citeseer.

[6] J.-M. Lin and Y.-W. Chang, "TCG: A transitive closure graph-based representation for non-slicing floorplans," in *Proceedings of the 38th annual Design Automation Conference*, 2001, pp. 764-769.

[7] Michel Berkelaar, Kjell Eikland, and Peter Notebaert. "lp_solve." http://lpsolve.sourceforge.net/5.5/ (accessed July 20, 2021.

[8] J. K. Ousterhout, "Corner stitching: A data-structuring technique for VLSI layout tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 3, no. 1, pp. 87-100, 1984.