應用於可變模組之零閒置平面規劃
Zero-deadspace Floorplanning for Soft Modules

王建延
國立臺北大學
電機工程研究所
s79682303
@webmail.ntpu.edu.tw

詹景裕
國立臺北大學
電機工程研究所
gejan@mail.ntpu.edu.tw

周典慶
國立臺北大學
電機工程研究所
s79782303
@webmail.ntpu.edu.tw

黃元欣
國立台灣科技大學
資訊工程研究所
shin@csie.ntust.edu.tw

摘要

平面規劃(floorplanning)為超大型積體電路(VLSI)的實體設計(physical design)階段中重要的步驟，本文針對可變模組之零閒置平面規劃(floorplanning)，提出全新且有效率的啟發式(heuristic)演算法，用於解決平面規劃面積最佳化的問題，此演算法基本的構想來自於俄羅斯方塊遊戲中。LS (L-Shaped packing)演算法之基本想法為將數個模組以並排的方式集結成模組區塊，並利用 L 形狀的方式持續地由限制邊界(fixed-outline)之左下至右上的方式擺置這些模組區塊，並藉由可變模組的高彈性度，使得最終的平面規劃可達到零閒置空間的最佳化效能。

在實驗部份，採用 MCNC 及 GSRC 標準測試電路，並且在 Intel® 2.6 GB 個人電腦上執行所有測試電路，結果顯示出所提出之 LS 演算法比較現今多種的平面規劃演算法，在面積最佳化效能與執行時間上，皆可獲得明顯的改進。LS 演算法於所有的標準測試電路中，以予限制一個嚴格的可變模組限制範圍下如：[0.5, 2]，皆可在小於1秒的執行時間內達到零閒置空間的平面規劃面積最佳化效能。

關鍵詞：面積最佳化、平面規劃、啟發式演算法、俄羅斯方塊、零閒置空間

Abstract

Floorplanning is an important phase in physical design of VLSI. In this paper, a new efficient heuristic algorithm is presented to handle soft modules: LS (L-Shaped packing) , which is inspired by the game, Tetris®. The LS algorithm generates a zero- deadspace floorplan for a set of soft modules with a very strict aspect ratio constraint. The soft modules are clustered into a zone side by side and placed the zone to the partial floorplan. While attempting to grow from lower-left to upper-right, all the zone are packed to the floorplan. The packing seems to be an iterative L-Shape packing for each pair of zones.

The experimental results are base on the standard MCNC and GSRC benchmarks. The resules demonstrate that the proposed algorithm can obtain a significant improvement both in the area and runtime, All the experiments are carried out on an Intel® 2.6 GB machine with 1 GB memory. Particularly, our methodology provides greater improvement over the other approaches. The results of LS on all benchmarks obtain zero-deadspace floorplans, and were completed within 1 second, under a strict aspect ratio of modules such as [0.5, 2]. The area utilization of the proposed model shows the superiority for solution quality, scalability and robustness. Consequently, it is also suitable to handle large scale floorplanning problems.

Keywords: Area optimization, Floorplanning, Heuristic approach, Tetris®, Scalability, Zero deadspace.

一、緒論

隨著超大型積體電路製程技術日新月異的發展使得晶片內電晶體數目持續地擴增導致晶片的設計愈趨複雜與龐大，相對地設計上也較為耗費時間，這對現今講求時效的電子產業來說是很大的挑戰，因此需藉由許多不同種類的設計流程與電腦輔助設計自動化工具(EDA tools)來做整合，並且隨著單一系統晶片(System-On-Chip, SoC)的盛行、矽智財(Intellectual Property, IP)被廣泛使用，使得模組化的設計概念逐漸普及，將

可有效的縮短系統晶片設計、驗證與佈局的時間。平面規劃(floorplanning)為超大型積體電路(VLSI)的實體設計(physical design)階段中重要的步驟，此步驟在於如何決定晶片內每個模組的形狀與位置，且在各模組間不允許任意兩模組相互重疊的情況下考慮各種的限制因素，期望能達到各種最佳化之目標，平面規劃的結果將會影響整體晶片的效能與成本。因此平面規劃之問題對於現今超大型積體電路實體電路設計上的影響顯得格外重要。

近年來針對解決平面規劃問題之方法已有許多不同的研究方法被提出，而大多數的方法專注於設計一個表示各模組間的拓樸關係(topology)的表示法(representation) ，並利用隨機搜尋與改進之演算法如：模擬退火演算法(Simulated Annealing, SA) [11]、演進式演算法(Evolutionary Algorithm, EA) [8,12]，期望最終能達到最佳解，但這些方法中，有些需要耗費較多的執行時間(SA)；有些只適合處理較小的電路，以現今註明的GSRC標準測試電路(benchmark)為例，它所包含模組數量最大之測試電路為n300(包含300個模組) ，這對現今複雜的電路設計而言是較為不足的，因此當在處理模組數量較多且較複雜的電路時(large scale)，這些方法會需要花費較大量的時間，且所得到的結果會隨者模組數量的增加呈現遞減的效能。

於一般平面規劃問題中主要包括處理模組長寬比例固定的固定模組(hard module)與模組長寬比例較彈性的可變模組(soft module)之面積最佳化問題，通常利用隨機搜尋與改進之演算法(SA、EA)較容易處理固定模組，但在處理可變模組上較為困難，且需耗費更多的時間，因此採用分析最佳化的方法將可有效地來處理可變模組的問題[15,16,17]，很不幸地；這些方法還是需要耗費較多的時間才能得到一個較佳的平面規劃結果。

本文之研究目標，利用創新而獨特的啟發式(heuristics)演算法來快速的解決平面規劃面積最佳化之問題，針對處理可變模組提出了LS (L-Shaped packing)演算法，此演算法的基本運算原理來自於俄羅斯方塊的遊戲(Tetris®)，我們將遊戲中的方塊比喻為模組，並如同以貪婪的方式(greedy)持續地堆積模組，當模組數量越多越有機會得到較佳的效能，因此隨著模組數量的增加會呈現出遞增的效能特性，改善先前研究以隨機搜尋最佳解之演算法來解決平面規定問題無法有效的處理大量模組且複雜之電路的缺點。而實驗部份將針對處理可變模組類型的問題進行實驗，並利用MCNC與GSRC標準測試電路(benchmark) [19]與現今多個平面規劃演算法進行比較來驗證所提出之演算法之效能。在比較多個平面規劃演算法中分成實行相同與不同平台之比較結果兩部份。在相同實驗平台中比較的是現今熱門的兩種平面規劃演算法：B*-Tree程式以快速模擬退火演算法(Fast-SA)為基礎[5]及CompaSS[4]演算法。而在不同實驗平台中與之比較的是與我們所提出之演算法有類似概念的MBS [12]演算法。

由於平面規劃問題需同時考慮多種的因素，使得此問題變的相當複雜，所以平面規劃之問題已被歸類為非多項式時間內能找出最佳解之問題(NP-Hard) ，因此執行時間(run time)將是個重大的挑戰，而本文針對處理可變模組所提出的演算法能有效率地在快速的時間內得到不錯的效能且包含了一些不錯的特性，非常地適合應用於現今的超大型積體電路實體電路設計中。

二、問題描述與定義

假設輸入為 $\mathbf{B} = \{b_1, b_2, \ldots, b_n\}$ 為 n 個矩形模組之集合，每個模組 b_i 有各別的長度(h_i)與寬度(w_i)以及面積 $a_i = h_i \times w_i$，如果輸入的模組為固定模組，則模組的長度於寬度為固定且允許做旋轉90度之動作；如果為可變模組則各模組可變的寬度與長度會限制於一個給定的限制範圍內(aspect ratio of modules)如(1)式所示：

$$AR_{min} \leq h_i / w_i \leq AR_{max} \qquad (1)$$

其中 $[AR_{min}, AR_{max}]$ 為給定的限制範圍，因此我們可得知模組 i 可變的寬度與長度的範圍為 $[L_i, U_i]$ 如(2)式所示：

$$L_i = \sqrt{\frac{a_i}{AR_{\max}}} \text{ and } U_i = \sqrt{\frac{a_i}{AR_{\max}}} \qquad (2)$$

其中 $p_i = (x_i, y_i)$ 表示模組 i 的左下角座標

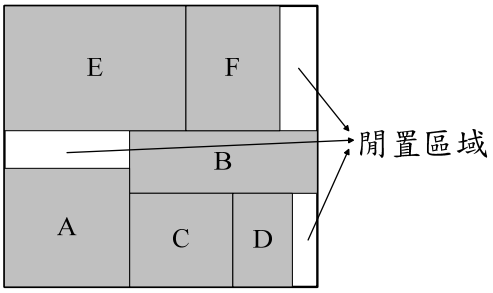

圖 1 閒置區域

(coordinates)位置。給定一個平面規劃 **P** 為平面空間且所有模組間彼此不允許有重疊的情況發生，並將所有模組分配於 **P** 中，即 **P**=$\{p_i \mid 1 \leqq i \leqq n\}$。本文研究主要針對固定模組與可變模組，並期望能在較短的執行時間內達到面積最佳化的目標為主要研究方向。

平面規劃問題之面積最佳化的目標，其評估的方式是針對所有模組擺置完成後所圍繞的矩形面積越小效能越佳，所以我們可以藉由計算整體晶片的閒置區域(dead space)作為面積最佳化效能之參考的根據，其閒置區域的定義如圖 1 所示:

其計算方式如(3)式，其中n代表模組數、$area_i$代表第i個模組之面積、γ代表模組擺置完成後所圍繞的矩形面積。

$$Dead\ space = A^{fp} - \sum_{i=1}^{n} a_i \qquad (3)$$

另外我們可計算其空間利用率(Area usage) ，通常我們以百分比來評估整體面積利用率之效能，當空間利用率越高代表其效能越佳，計算方式如(4)式:

$$Area\ usage(\%) = \frac{\sum_{i=1}^{n} a_i}{A^{fp}} \times 100 \qquad (4)$$

在現今的平面規劃問題可分類為典型的平面規劃(Classical floorplanning)問題與限制邊界的平面規劃問題(fixed-outline floorplanning)。典型的平面規劃問題為不需預先限制模組擺置的範圍(variable-die)，並盡可能的去找出最佳的平面規劃面積，因此面積最佳化之效能為此類平面規劃問題主要的目標。限制邊界的平面規劃問題，於 VLSI 實體設計中需將電路規劃於固定大小的晶片內(fixed-die)，使得限制邊界的平面規劃問題變得相對地重要[10]，近年已有許多文獻提出針對解決限制邊界的平面規劃問題之方法[1, 2, 3, 5, 6, 7, 13, 14]，此限制邊界的平面規劃問題於模組實行規劃前會先制定合理的矩形邊界，其此矩形邊界的長度(H)與寬度(W)的比例(aspect ratio of outline)為 $R=H/W$，並根據給定的總面積A與最大的閒置空間比例(maximum percentage of dead space) σ，可得到晶片的總面積為 $H*W=(1+\sigma)A$ 並根據(5)式[1]，可得到此限制矩形邊界合理的長度與寬度。

$$W = \sqrt{(1+\sigma)AR} \ \text{ and } \ H = \sqrt{(1+\sigma)A/R} \qquad (5)$$

三、LS (L-Shaped packing)演算法

本文所提出之有效處理可變模組(soft module)之演算法，稱為 LS (L-Shaped packing)，所提出之處理可變模組的方法將可額外地實行於一個限制邊界的平面規劃(fixed-outline floorplanning)問題中，並適用於多種限制邊界的比例，而 LS 演算法基本概念是挑選部份的模組以並排(side by side)的方式集結成模組區塊(zone)，各模組區塊內由數個模組以並排的方式所組成，所以此模組區塊看似一塊矩形區域，其模組區塊定義如(6)式：

$$z_j \ where \ j \in \{1,2,3,...,M\} \qquad (6)$$

其中 M 為最終平面規劃結果的模組區塊總數，並根據(5)式，給定的矩形限制邊界，持續地將模組區塊沿者矩形限制邊界的底邊(bottom of boundary)或左側邊(left of boundary)上擺置，所以當 j 為奇數(odd)時，模組區塊會以水平方式(row-orientation)擺置，而當 j 為偶數(even)時則以垂直方式(column-orientation)擺置，因此模組區塊會一個接者一個地(zone by zone) 由左下角往右上角呈現對角線(diagonal)方向持續地擺置，這種動作會呈現出一個 L 形狀的擺置方式。同時在分配模組給予每一個模組區塊時，會紀錄此模組區塊允許擺置的模組序列組合，這是為了讓在持續分配模組於模組區塊內時，於較後段的模組區塊較容易發生不存在任何一組模組序列組合可有效集結成一模組區塊的情況(例：可變模組之可變範圍超過所限制的比例)，因此紀錄每個模

Algorithm: LS Floorplanning
Input: SB－A set of n soft modules with U_i, L_i and $area_i$; a fixed-outline region for packing
Output: SB－A zero-deadspace packing including coordinates (p_i is non-null), height (h_i) and width (w_i) of the given modules
Begin
01: **SB**[n] → Sorting by area of all modules;
02: **for** (*index* from 1 to n)
03: z_j = **BasicPacking(SB**[*index*]);
04: **if** (The z_j does not assign successfully)
05: Perform the **Refinement()** subroutine;
06: **else if** (Exist a smaller module)
07: Perform the **Wheel()** subroutine;
08: **else** z_{j++} ; // assigning next region
09: **End for**
End

組區塊z_j的允許的模組序列組合可有利於後續實行重新擺置程序。並藉由可變模組的高彈性度(flexibility)，使得最終的平面規劃結果可達到零閒置空間(zero-deadspace)的面積最佳化效能。

所提出之 LS 主演算法如上所示，演算法初始化會先給定一個根據所有模組的面積由大到小排序的陣列，如演算法第一行所示，之後第二至九行會根據排序結果持續的實行基本的模組擺置程序，其中第三行會持續的分配模組給每一模組區塊(**BasicPacking**)，而當此模組區塊內發生沒有任何一組允許的模組序列組合的情況時，則執行第五行重新擺置副程式(**Refinement**)，或者；尚未擺置的模組中存在一個非常小的模組之特殊情況，則執行第七行解一個車輪形狀之模組區塊副程式(**Wheel**)。而所提出之 LS 演算法中主要分為三個副程式分別為: **BasicPacking**、**Refinement**、**Wheel**，分別於接下來的 3.1、3.2、3.3 小節詳細說明。

3.1 BasicPacking 副程式

此為一種全新的模組擺置(packing)方法，它持續地將已分配模組完成的模組區塊z_j利用類似 L 型的方式，持續地將模組區塊往矩形限制邊界的底邊和左側邊上擺置，當在持續擺置模組區塊時會需要持續計算剩餘矩形擺置空間的寬度與長度分別為 Hj 和 Wj，其定義如(7)式所示：

$$0 \leqq W_j \leqq W$$
$$0 \leqq H_j \leqq H \qquad (7)$$

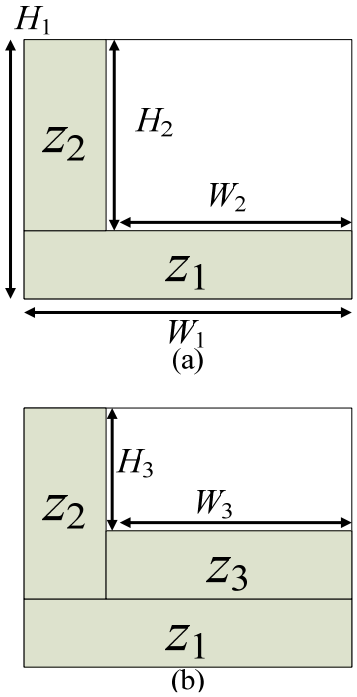

圖2 模組區塊的擺置方式：(a)模組區塊z_1根據W_1先以水平擺置，模組區塊z_2再根據H_2以垂直擺置; (b)模組區塊z_3根據W_3，並以水平擺置。

其中W_1=W、H_1=H為初始值，代表一開始所給定的矩形限制邊界的寬度與長度，而計算剩餘的矩形擺置空間是為了提供下一個z_j分配模組的依據，如圖2 (a)所示，當在實行分配模組至模組區塊z_1時會根據W_1來分配模組，並將模組區塊以水平方向擺置；接者當實行模組區塊z_2時則會根據H_2來分配模組，並以垂直方向擺置。緊接著如圖2 (b)所示，實行模組區塊z_3時則根據W_3來分配模組，並將模組區塊z_3以水平方向擺置依此類推，持續地由左下角往右上角的方向擺置，此種類似L形狀的擺置模組方式，是為了讓剩餘的矩形擺置空間之長度(H_j)與寬度(W_j)比例持續地維持在接近正方形，使得有較高的機會成功規劃剩下的模組區塊，以達到能處理多種嚴格的可變模組限制範圍(aspect ratio of modules)之比例。

每一模組區塊z_j中挑選模組的方法，是依照模組排序過後的順序依序地找出可以允許置入此模組區塊z_j之最多模組的可能情況，並以並排的方式擺置於此模組區塊中，且紀錄z_j中允許的模組序列組我們定義為fit_j，代表此模組區塊z_j允許的模組序列之數量，由於我們只需利用一個一維陣列來儲存各模組的擺置先後順序(priority)且

並不會去改變這它的順序，因此只需紀錄每個模組區塊z_j中的允許模組序列數量，即可得知此模組區塊內可移動的模組有哪些，以方便我們做後續實行重新擺置之動作(於3.2節詳細說明)。

此副程式之主要動作為：我們利用佇列(queue)將排序過後的模組一個一個持續加入，每加入一個模組將會計算佇列中的模組總面積除以H_j或W_j，根據(8)式決定：

$$\begin{cases} \text{if } j=\text{odd}, hz_j = \dfrac{\text{total area of modules in queue}}{W_j} \\ \text{if } j=\text{even}, hz_j = \dfrac{\text{total area of modules in queue}}{H_j} \end{cases} \quad (8)$$

計算所得的值即為此模組區塊的高度定義為hz_j，反之；H_j與W_j即為此的z_j寬度，而每當加入一模組至佇列並計算hz_j後，需檢查hz_j是否有超過佇列中各模組的可變範圍，此檢查方式只需找出佇列中所有模組的可變範圍最大值U_i^{Max}與最小值L_i^{Min}進行檢查即可，這是確保最終的平面規劃結果不會存在有不合法的模組(abnormal module)之情況(例：超過可變模組限制範圍)，而此**BasicPacking**副程式之詳細的動作流程如下之步驟：

Function: BasicPacking

Begin

Initial: 給定排序過後的陣列 **SB**[*index*]與空佇列 *queue*，以及 fit_j 來紀錄各模組區塊之允許的模組序列數量，宣告一個變數為 hz_j 來計算每一模組區塊的高度，宣告 U_i^{Max} 與 L_i^{Min} 分別表示目前佇列中所有模組可變範圍的最大值與最小值

Step 1. 從 **SB**[*index*]中移動一模組至佇列中

Step 2. 根據(8)式，計算hz_j值

Step 3. 檢查是否允許，如果滿足 $L_i^{Min} \leqq hz_j \leqq U_i^{Max}$ 則表示允許，且將fit_j++以及**SB**[*index*++]，並跳至Step 1，反之；則為不允許執行Step 4

Step 4. 當不允許的情況發生時，則分為下列三種情況討論:

Case 1. 如果當 $hz_j \geqq U_i^{Max}$ 則清空*queue*執行下一個模組區塊的配置z_j++

Case 2. 如果當 $hz_j \leqq L_i^{Min}$ 則將 **SB**[*index*++]，並跳至Step 1繼續移動模組至佇列中

Case 3. 如果當 $hz_j \geqq U_i^{Max}$ 且 $fit_j = 0$時，則表示剩下尚未擺置的模組無法允許的分配給此模組區塊，所以必須執行**Refinement**或**Wheel**副程式(於3.2、3.3詳細說明)

End

3.2 Refinement副程式

此副程式之主要動作為，當持續地分配模組給模組區塊時，可能會發生剩餘的模組無法合法的分配給較後分配的模組區塊，因此可根據fit_j所紀錄下來各模組區塊允許的模組序列，利用返回追蹤(back tracking)的方式持續地往前一個模組區塊找尋可允許移動的模組，並以一次移動一個模組至尚未允許擺置的模組區塊內做重新擺置的動作，這是因為起初在分配模組給各模組區塊時，是以此模組區塊最多可擺置的模組數量來決定此模組區塊內該分配多少模組，因此在較晚才分配模組的模組區塊內會僅存零星的模組，使得這些模組的可變範圍無法滿足所限制的範圍，所以必須從之前已分配的模組區塊內逐步地移動模組至目前尚未允許擺置的模組區塊內，我們用一個簡單的例子說明此副程式的動作如圖3(a)所示，圖中z_3為一個不允許的模組區塊，假如z_2中並無允許的模組序列(fit_2=0)，而z_1中存在

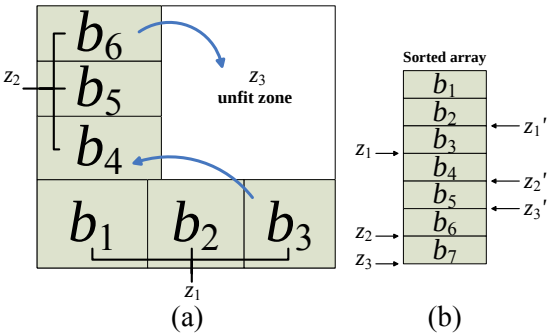

圖3 重新擺置副程式之動作 (a)移動一模組至尚未允許擺置之模組區塊內; (b)所對應之陣列內指標更新的方式

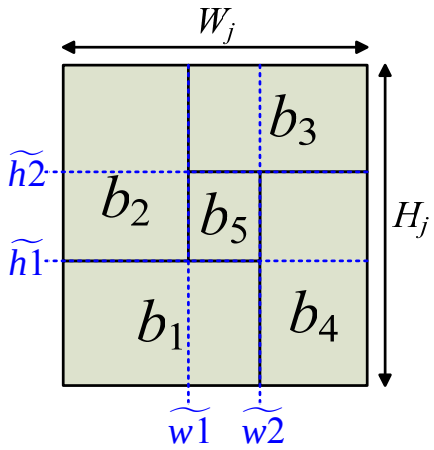

圖4 Modeling the wheel order five

一筆允許的模組序列(fit_3=1)，代表可以移動一模組至z_3，因此可以將模組b_3往前一個模組區塊z_2移動，在將z_2中的最後一個模組b_6給予z_3，這很類似連續擠壓的動作，一次只會加入一個模組給尚未擺置區塊實行重新擺置動作，由於我們只利用一個一維陣列來表示一個平面規劃中模組之間的關係，所以只需簡單的利用多個指標(point)，並將每個模組區塊的指標指向該模組區塊中最後一個模組的位址，將可有效地表示每個模組區塊內有哪些模組如圖3 (b)所示，由此可知當實行重新擺置動作時只需要去移動這些指標所指的陣列位址即可，一個簡單的例子如圖3 (b)所示為對應到圖3 (a)中模組擺置情形，其中z_1、z_2、z_3分為尚未實行重新擺置動作時，各模組區塊的指標所指向的位址，而z_1'、z_2'、z_3'分別表示移動一塊模組後所更新的指標位址。因此在大多數的情況下藉此動作將可在各種可變模組限制範圍以及矩形限制邊界下有效地將所有模組分配於模組區塊中，快速地達到零閒置空間的平面規劃結果。不過當存在者某些特殊的例外情況時，當實行完此副程式仍然無法有效地分配模組於尚未擺置的模組區塊時，則需執行**Wheel**副程式將於3.3詳細說明。

3.3 Wheel副程式

當分配模組區塊時存在者一個模組面積特別小(可變範圍與其他模組可變範圍無交集)或執行Refinement副程式仍無法有效滿足目前的模組區塊時，則需要透過此副程式來實行一個車輪形狀的模組擺置方式，其基本運算動作一樣會以返回追蹤的方式地持續地加入一模組至尚未允許擺置的模組區塊內，直到此模組區塊內恰好分配到五個模組之情況，並將這些模組中面積最小的模組表示為車輪狀模組擺置方式中的中間模組(seed module)，如圖4所示，由此我們可以利用解一元二次方程式來有效的決定這五個模組在此模組區塊內的位置，圖中W_j、H_j為已知數，$\widetilde{h1}$、$\widetilde{h2}$、$\widetilde{w1}$、$\widetilde{w2}$為方程式所需解的變數，假定a_1、a_2、a_3、a_4分別為模組b_1、b_2、b_3、b_4的已知面積，我們可建構出(9)式：

$$\begin{cases} a_1 = \widetilde{w2} \times \widetilde{h1} \\ a_2 = \widetilde{w1} \times (W_j - \widetilde{h1}) \\ a_3 = (W_j - \widetilde{w1}) \times (H_j - \widetilde{h2}) \\ a_4 = (W_j - \widetilde{w2}) \times \widetilde{h2} \end{cases} \quad (9)$$

假定我們任選$\widetilde{h1}$變數來解一元二次方程式如(10)式所示，再根據(9)式，重新歸類整理後得到(11)式中各常數項如(10)式：

$$\widetilde{h1}^2 \alpha + \widetilde{h1}\beta + \gamma = 0 \quad (10)$$

$$\begin{cases} \alpha = H_j W_j^{\,2} - W_j a_3 - W_j a_4 \\ \beta = H_j W_j a_3 + a_3 a_1 - H_j^{\,2} W_j^{\,2} + H_j W_j a_1 \\ \qquad + H_j W_j a_4 - a_2 a_4 - H_j W_j a_1 \\ \gamma = H_j^{\,2} W_j a_1 - H_j a_3 a_1 - H_j a_1 a_2 \end{cases} \quad (11)$$

根據(10)式與(11)式，可以利用(12)式解一元二次方程式如下：

$$root_1 = \frac{\beta^2 + 4\alpha\gamma}{2\beta} \text{ and } root_2 = \frac{\beta^2 - 4\alpha\gamma}{2\beta} \quad (12)$$

所得之兩根$root_1$、$root_2$只有一個根是合法的根，其檢查方法為將此兩根各別代回(9)式，將可解出剩下的變數$\widetilde{h2}$、$\widetilde{w1}$、$\widetilde{w2}$，藉由檢查$\widetilde{h2} > \widetilde{h1}$是否成立，如果成立代表為一合法根，反之；為一不合法之根。藉由利用一元二次方程式解車輪狀模組擺置方式，將可有效地應用於LS演算法中並解決一些較特殊之例外情況，並且可在快速執行時間內產生出一個零閒置空間的平面規劃結果。

四、效能分析

(一)空間複雜度分析

關於 LS 演算法的空間複雜度分析，演算法中一開始須將所有模組排序，其使用排序法，只需一個額外空間 $O(n)$。而在 **BasicPacking** function 中，當在實行分配模組至模組區塊時利用一個一維陣列來儲存在模組區塊中各模組的擺置先後順序 $O(n)$，且紀錄 z_j 中允許的模組序列組使用 fit_j 變數，代表此模組區塊 z_j 允許的模組序列之數量，空間複雜度為 $O(\sqrt{n})$。而在 Wheel 副程式中會使用五個變數代表其所使用的五個模組，因此其空間複雜度為 $O(1)$，所以本演算法的空間複雜度(Space Complexity) SC_{Total} 為：

$$SC_{Total} = SC_{data} + SC_{sort} + SC_{queue} + SC_{fit} + SC_{wheel}$$
$$= O(n) + O(1) + O(n) + O(\sqrt{n}) + O(1)$$
$$= O(n)$$

(二)時間複雜度分析

關於 LS 演算法的時間複雜度分析，演算法中一開始須將所有模組排序，其使用排序法，平均時間需花 $TC_{sort} = O(n\log n)$。接下來在 for 迴圈中有三個副程式，分別探討。

在 **BasicPacking** function 中，依照模組排序過後的順序依序地找出可以允許置入此模組區塊 z_j 之最多模組的可能情況，以平均時間的觀點來看其時間複雜度 TC_{bk}^{Avg} ：

$$TC_{bk}^{Avg} = |Z_i|^{Avg}$$
$$= O(\sqrt{n})$$

在 **Refinement** function 中，實行重新擺置動作時，一次只會加入一個模組給尚未擺置區塊，需要去移動這些指標所指的陣列位址，其時間複雜度為 TC_{rf} ：

$$TC_{rf} = O(\sqrt{n})$$

在 **Wheel** function 中，主要部份是在求一元二次方程式的值，因此其時間複雜度為 TC_{wh}：

$$TC_{wh} = O(1)$$

For 迴圈中平均執行 n 次，每一次執行 **BasicPacking**、**Refinement**、**Wheel** 三個副程式，因此綜合 LS 演算法中所有步驟的時間複雜度可得知道該演算法的時間複雜度的最大為 $O(n\log n)$，因此該演算法的時間複雜度(Time Complexity)為 $O(n\log n)$，即：

$$TC_{Total}^{Avg} = TC_{sort} + TC_{loop} + TC_{bk}^{Avg}\,(TC_{rf}^{Avg} + TC_{wh})$$
$$= O(n\log n) + O(n) + O(\sqrt{n})\left(O(\sqrt{n}) + O(1)\right)$$
$$= O(n\log n)$$

五、實驗結果

本文所提出之演算法以 PHP (Hypertext Preprocessor)程式語言來實現，實驗平台為Intel® 2.6 GHz CPU with 1 GB RAM之個人電腦。

本文以處理可變模組提出了LS演算法有效地去實行平面規劃面積最佳化之問題，因此本文之實驗部份將針對處理此模組類型的問題進行實驗，並利用 MCNC 與 GSRC 標準測試電路(benchmark) [19]與現今多個平面規劃演算法進行比較來驗證所提出之演算法之效能。本章節之架構如下：LS演算法用於處理可變模組之實驗於5.1中詳細說明、5.2為比較與我們所提出之演算法其基本概念類似的MBS演算法，並實行於不同的實驗平台與LS演算法比較處理可變模組之平面規劃面積最佳化效能。

5.1實行於相同實驗平台之比較結果

本文所提出之LS演算法，實驗所採用的測試電路為MCNC及GSRC標準測試電路(benchmark)進行實驗，所有標準測試電路皆為可變模組版本，其模組數量從9至300，我們比較現今熱門的兩種平面規劃演算法進行實驗，以予證明LS演算法能在快速的執行時間內達到零閒置空間的效能，並且可適用於各種的矩形限制邊界與極小模組限制範圍內，我們將所有的標準測試電路皆根據(5)式設定矩形限制邊界比例為1且允許最大的閒置空間比例為0 (R=1、σ=0)以予實行所提出之LS演算法。

比較現今熱門的兩種平面規劃演算法：B*-Tree 的原始程式版本為 btree_win_v1.1 來源至[18]此 B*-Tree 程式是以快速模擬退火演算法(Fast-SA)為基礎[5]，以及 CompaSS[4]之原始程式版本為 Version_1.2 來源至[20]，此 CompaSS

表 1 與 B*-Tree、CompaSS 於相同實驗平台上比較空間利用率(Area usage)與執行時間(Runtime)之結果。

circuit	B*-tree		CompaSS		LS(Ours)	
	Area Usage (%)	Runtime (s)	Area Usage (%)	Runtime (s)	Area Usage (%)	Runtime (s)
apte	99.11	0.443	99.25	0.06	100.00	0.018
xerox	99.09	0.466	100.00	0.05	100.00	0.014
hp	99.38	0.532	100.00	0.06	100.00	0.014
ami33	98.45	3.247	100.00	0.52	100.00	0.042
ami49	98.56	10.14	99.99	0.94	100.00	0.052
n10	99.49	0.716	100.00	0.05	100.00	0.019
n30	98.55	3.252	99.95	0.36	100.00	0.036
n50	98.30	8.704	100.00	0.41	100.00	0.054
n100	97.65	24.19	100.00	1.11	100.00	0.087
n200	96.94	78.33	100.00	5.11	100.00	0.176
n300	96.88	217.4	100.00	15.17	100.00	0.279

為一種用於處理可變模組的平面規劃面積最佳化之演算法，它可在部份的測試電路中快速地達到零閒置空間的效能。並將此兩種演算法與 LS 演算法實行在同樣的實驗平台上作比較，其中 B*-Tree、CompaSS 之原始程式碼是由 C++程式語言實行；而 LS 演算法一樣採用 PHP 程式語言實行，因此在比較執行時間上會趨於劣勢，但 LS 演算法在執行時間上並不會因此得到較差的結果，如表 1 所示。LS 演算法與 B*-Tree、CompaSS 使用相同的標準測試電路作比較，於第一行所示模組數量為 9 至 300 的 MCNC 與 GSRC 的 11 個標準測試電路，所有的標準測試電路所設定的模組可變範圍為[0.5, 2]，並在 B*-Tree 只考慮面積最佳化的情況下，各自執行十次(CompaSS 以及 LS 演算法只需執行一次)所得到之數據為平均值列於第二、三行分別為空間利用率與執行時間上之結果，第四、五行則為 LS 演算法所得之實驗結果，由表中可顯示 LS 演算法各個標準測試電路中，所測得的空間利用率之效能皆比 B*-Tree 來的好，且在執行時間上快 25~779 倍的時間；相較於 CompaSS 雖然它能在大部份的標準測試電路中達到 100%的空間利用

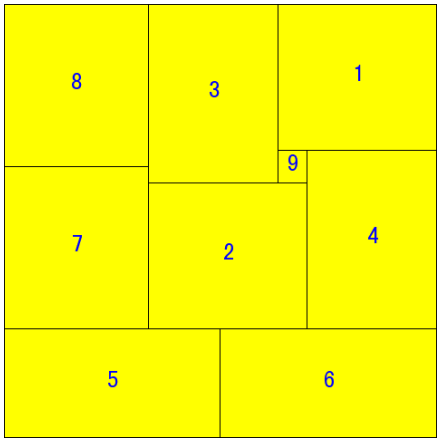

圖 5 apte 可變模組佈局圖

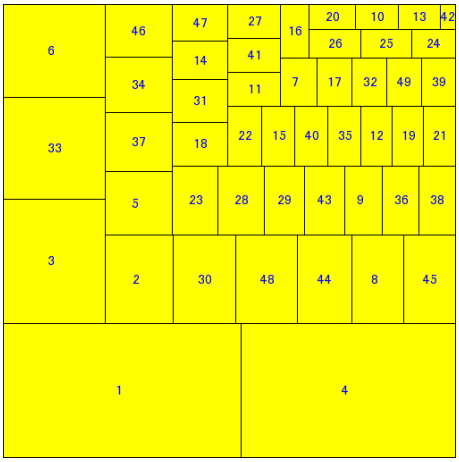

圖 6 ami49 可變模組佈局圖

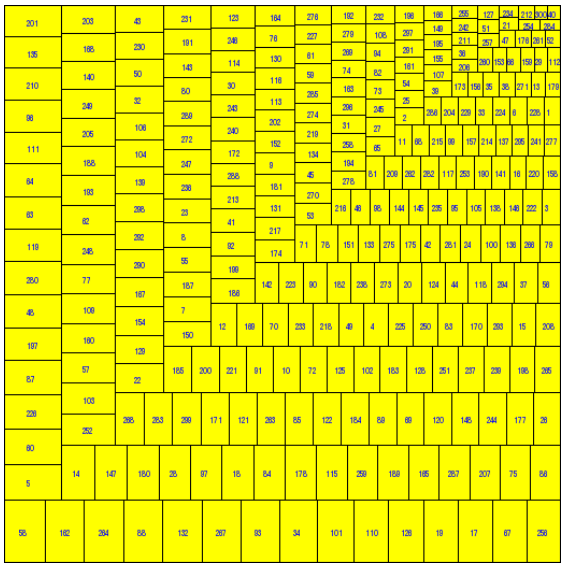

圖 7 n300 可變模組佈局圖

率，但其中還是有三個測試電路無法達到(apte、ami49、n30)，並且在執行時間上 LS 演算法可快 3~54 倍的時間。由此結果更精確的驗證了 LS 演

算法優秀的效能。圖 5、6、7 分別為此部份實驗的標準測試電路之平面規劃佈局圖(layout)，圖中可清楚顯示 LS 演算法持續地以 L 型方式往右上角擺置模組之特性。

5.2 實行於不同實驗平台之比較結果

此部份實驗利用與我們所提出之演算法有類似概念的 MBS [12]演算法實行於不同的實驗平台作比較，其基本的想法皆是來至俄羅斯方塊遊戲，差別在於 MBS 演算法採用演進式演算法(EA)為基礎來實行平面規劃面積最佳化，因此它並不是一個決定性的演算法，而我們所提出之演算法皆為決定性的演算法，其 MBS 實驗平台為 IBM IntelliStation Z Pro Type 6221 with 1 GB memory，其中根據 MBS 論文中的實驗結果所有標準測試電路的所設定的模組可變範圍(aspect ratio of modules)為所有 MCNC 標準測試電路皆設定為[0.5, 2] 而 GSRC 標準測試電路則設為[0.3, 3]，其實驗數據為執行 10 次所得到的空間利用率(Area Usage)與執行時間(Runtime)之平均值，注意到因為所有實驗皆實行於不同的實驗平台，所以於執行時間上僅為參考。

表 2 所有標準測試電路皆為可變模組，其中 MBS 實驗數據來至[12]。

circuit	MBS		LS(Ours)	
	Area Usage(%)	Runtime(s)	Area Usage(%)	Runtime(s)
apte	99.98	18.5	100.00	0.018
xerox	99.91	21.2	100.00	0.014
hp	99.96	26.1	100.00	0.014
ami33	99.23	361.4	100.00	0.042
ami49	98.68	890.3	100.00	0.052
n10	99.95	23.1	100.00	0.019
n10b	100.00	17.1	100.00	0.014
n10c	99.94	22.0	100.00	0.014
n30	99.18	300.7	100.00	0.036
n30b	99.22	296.7	100.00	0.038
n30c	99.21	284.7	100.00	0.031
n50	98.23	949.7	100.00	0.054
n50b	98.15	1077.3	100.00	0.050
n50c	98.13	1022.3	100.00	0.053
n100	94.11	6440.4	100.00	0.087
n100b	94.31	7123.4	100.00	0.087
n100c	94.47	7146.8	100.00	0.086

而所提出之LS演算法與MBS實行於可變模組版本之標準測試電路，採用模組數量從 9 至 300 的十七個標準測試電路比較之結果如表 2 所示，可明顯的比較出 LS 演算法在所有標準測試電路中不論是空間利用率與執行時間皆好過 MBS，同樣地再次證明在每個標準測試電路中 LS演算法皆可於非常快的時間內達到100%空間利用率。

六、結論與未來展望

本文提出LS演算法之方法如同將數個模組群聚成一個矩形模組區塊，並利用L形狀的方式持續地擺置這些矩形模組區塊，使得最終的平面規劃結果可達到零閒置空間的最佳效能，且適用於多種的可變模組限制範圍(aspect ratio of modules)之比例以及可有效地應用於多種比例的限制邊界(aspect ratio of outline)平面規劃問題中。

在處理可變模組上所提出的 LS 演算法之實驗結果證明，LS 演算法在所有的標準測試電路中，利用嚴格的可變模組限制範圍之比例[0.5, 2]以及多種比例的限制邊界平面規問題中，皆可有效地達到零閒置空間的最佳化效能。

在未來的研究方向上，由於本文現階段是以處理可變模組的平面規劃面積最佳化問題為主，因此於未來的研究方向上有很多種必然的可行目標，分別列舉如下：

- 同時達到多種最佳化目標：制定一套有效的成本函式，可有效地將所提出之演算法同時考慮面積最佳化、繞線長度最佳化、繞線訊號延遲(delay)等多種最佳化目標。
- 處理任意形狀模組：本文所提出之方法，基本的概念來自於俄羅斯方塊遊戲中，由於處理任意形狀模組的平面規劃問題就如

同現實的俄羅斯方塊遊戲中多種形狀的方塊，因此我們的方法更有利於處理此類型的模組。尤其是可應用於人工智慧(Artificial Intelligence, AI)領域中，有效地預測與規劃出下一個方塊的最佳位置、方向等等。

- 多層次與階層式平面規劃技術：隨者現今超大型積體電路日益龐大、計算複雜度的提升，使得我們需將平面規劃問題分割成多個子問題，再針對這些子問題各別實行平面規劃，此技術能有效地降低計算複雜度與實行較龐大的平面規劃問題。現今利用此類技術有效地實行較大的平面規劃問題之方法如：使用多層次的平面規劃技術、階層式方法。而本文之方法在單一平面規劃問題下處理大量模組的能力，有較為有力且顯著的效能(如：快速的執行時間) ，因此對於利用多層次技術或階層式方法，能將分割成較大的子問題並有效地處理它，所以對於處理更加龐大的平面規劃問題而言會顯得更有優勢。

參考文獻

[1] S. N. Adya and I. Markov, "Fixed-outline floorplanning: Enabling hierarchical design," *IEEE Trans. on Very Large Scale Integration System*, Vol. 11, No. 6, pp. 1120–1135, 2003.

[2] S. N. Adya and I. L. Markov, "Fixed-outline Floorplanning Through Better Local Search," *Int. Conf. on Computer Design*, pp. 328-333, 2001.

[3] S. N. Adya and I. L. Markov, "Consistent Placement of Macro-Blocks using Floorplanning and Standard-Cell Placement," *Proc. of ACM Intl. Symposium on Physical Design*, pp. 12-17, 2002.

[4] H. H. Chan and I. L. Markov, "Practical Slicing and Non-slicing Block-Packing without Simulated Annealing," *Technical Report CSE-TR-487-04*, The University of Michigan, 2004.

[5] T.-C. Chen and Y.-W. Chang, "Modern floorplanning based on B*- tree and fast simulated annealing," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 25, No. 4, pp. 637-650, 2006.

[6] T.-C. Chen, Y.-W. Chang and S.-C. Lin "IMF: interconnect-driven multilevel floorplanning for large-scale building-module designs," *Proc. IEEE/ACM Int. Conf. on Computer Aided Design*, pp. 159-164, 2005.

[7] J. Cong, M. Romesis and J. R. Shinnerl, "Fast floorplanning by look-ahead enabled recursive bipartitioning," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, No. 9, pp. 1719–1732, 2006.

[8] L. Jiao, J. Jiu and W. Zhong, "An organizational coevolutionary algorithm for classification," *IEEE Trans. Evol. Comput.*, vol. 10, No. 4, pp.67–80, 2006.

[9] S. Kang, "Linear Ordering and Application to Placement," *Proc. ACM/IEEE Design Automation Conf.*, pp. 457–464, 1983.

[10] A. B. Kahng, "Classical floorplanning harmful?," *Proc. of ACM Intl. Symposium on Physical Design*, pp. 207–213, 2000.

[11] S. Kirpatrick, C. D. Gelatt and M. P. Vecchi, "Optimization by simulated annealing," *Science*, 220(4598), pp. 671–680, 1983.

[12] J. Liu, W. Zhong, L. Jiao and X. Li, "Moving Block Sequence and Organizational Evolutionary Algorithm for General Floorplanning With Arbitrarily Shaped Rectilinear Blocks," *IEEE Trans. Evol. Comput.*, vol. 12, no. 5, pp. 630– 646, 2004.

[13] C. Luo, M. F. Anjos and A. Vannelli, "Large-scale fixed-outline floorplanning design using convex optimization techniques," *Proc. of ACM/IEEE Asia South Pacific Design Automation Conf.*, pp. 198–203, 2008.

[14] C. Luo, "Novel Convex Optimization Approaches for VLSI Floorplanning," *Ph.D. Dissertations*, The University of Waterloo, Canada, 2008.

[15] S. Sutanthavibul, E. Shragowitz and J. B. Rosen, "An analytical approach to floorplan design and optimization," *IEEE Trans. on*

Computer-Aided Design of Integrated Circuits an Systems, vol. 10, no. 6, pp. 761–769, 1991.

[16] F.-Y. Young, C. C.-N. Chu, W.-S. Luk and Y.-C. Wong, "Floorplan area minimization using Lagrangian relaxation," *Proc. of ACM Intl. Symposium on Physical Design*, pp. 174–179, 2000.

[17] F. Y. Young, C. C. N. Chu, W. S. Luk and Y. C. Wong. "Handling soft modules in general non-slicing foorplan using lagrangian relaxation," *IEEE Trans. on Computer-Aided Design*, vol. 20, no. 5, pp. 687–692, 2001.

[18] http://cc.ee.ntu.edu.tw/~ywchang/

[19] http://vlsicad.eecs.umich.edu/BK/parquet/

[20] http://vlsicad.eecs.umich.edu/BK/CompaSS/

[21] http://cc.ee.ntu.edu.tw/~ywchang/Courses/PD/Floorplanning_20080410.pdf