CS6135 VLSI PDA HW3 Report

108062537 魏聖修

(2)

compile : make

execute : ./hw3 <hardblocks> <nets> <pl> <output_name> <white_space_ratio>

```
[zscaxd5651@ic22 src]$ make
g++ -g -std=c++11 -O3 main.cpp -o  hw3
[zscaxd5651@ic22 src]$ time ./hw3 ../testcase/n100.hardblocks ../testcase/n100.nets ../testcase/n100.pl ../output/n100.floorplan 0.2
```

(3)

	0.2 wirelength	0.2 runtime	0.15 wirelength	0.15 runtime
N100	225833	29.12	223287	60.45
N200	412315	274.46	399099	1073.98
N300	549259	625.11	560770	1143.23

(4)

N100: 0.1

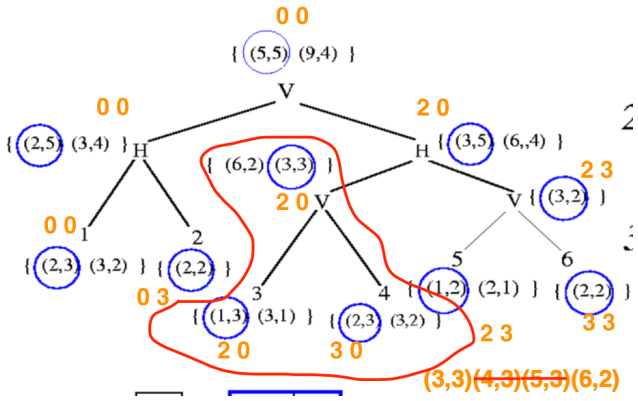
N200: 0.15

N300: 0.15

(5)

基本上這次 simulating annealing 是依照那篇 paper 上的 polish expression 實作，我主要把這個程式分成兩個部分。

（一） basic function



基本上就是寫那些能夠將一個 polish expression 表示成 binary tree 的 function。共分成四個 function: MergeVCut、MergeHCut、MinAreaAlgorithm、SetBlocks。

MergeVCut & MergeHCut:

這兩個做的事基本上相同，只是針對不同的 cut。做的事情就是如講義上範例，能將兩個 hardblocks 依照他們的 cut 以及不同的 orientation 合併，並且刪除那些不理想的長寬。

MinAreaAlgorithm:

這個部分的難處是要依照 postorder traversal 的順序來合併，才不會出錯。例如上圖 block3 和 block4 合併 Vcut，block5 和 block6 合併 Vcut，再將兩個 Vcut 合併 Hcut。實作時，把合併完的 cut 視為一個新的 block 是我自己用來簡化這個問題的一個方法。這個 function 的目的主要就是把上面例子的那些括號全部找出來，以及是如何由他們的 child 建造的，並且記錄相對應的 index。當初實作時有遇到一個問題就是，其實任兩個 Block 合併並不一定只有兩種長寬的可能，基本上是有很多種組合的。當初並沒有想到這方面的問題，我只設定找最長的和最寬的。造成最後塞不進去，就是因為在這一步並沒有找到這個 polish expression 的 min-area。

SetBlocks:

基本上上面兩個 function 都把 tree 做出來了，polish expression 的最小面積也都找出來了。這個 function 就是依照剛剛紀錄的資訊去設定每個 block 的 xy 座標以及 rotation。這部分我是使用 preorder traversal 來實作。由上面的例子來解釋的話，首先先確定好要 root 的哪一個面積，再來把相應的 child node 用藍色圈起來，這樣就能決定好 rotation。最後用 preorder 的方式來決定橘色寫的座標，至於決定的方式就是看這兩個 node 是依照什麼 cut 合併的。依此，left child 放在目前已經擺放完成的座標的最右下角，right child 放在 left child 的上面或者右邊。這樣就能決定好 xy 座標。

（二）simulating annealing

這次我用的 annealing schedule 和 paper 上的實作有些許不同。但主要一樣分成四個 function: init_floorplan、perturb、cost_function、simulated_annealing。

Init_floorplan:

initial 方式是依照所以 block 的高度去做排列再去依序擺放，並且擺放不能超過限制的寬度，超過就往上疊。

Perturb:

是照 paper 一樣分成三種 Move。SwapOperand、ComplementOperator、SwapOperandOperator。我是都會去記錄這個 polish expression index 哪些是 operand，哪些是 operator。依此就能簡單的實作上面三個 move，只要依照紀錄的 index 去做 swap 以及 complement 即可。

Cost_function:

cost = 1 * (max_x * max_y / initial_area)
+ 1 *(wirelength / initial_wirelength)
+ 1 * (x_diff + y_diff + xy_diff)

前兩項都是使用當前的去除以 initial 作標準化。然後把 x_diff 和 y_diff 以及 xy_diff 當作 cost function 的主要項，最後一項基本上會 dominate 整個 cost function，目的就是要先以塞進去限制範圍為主。

Simulated_annealing:

先創建 initial flooplan，依照他的 parameter 當作 cost function 的標準化基準以及溫度的初始值。之後我會去進行 3000 次的 perturb，這 3000 次中，如果是 down hill 直接接受，如果是 up hill 依照講義上定義的機率去接受。之後不同的是，我會進行一個回溫，並且回復到目前最好的 polish expression 重新開始 perturb。所以也因此終止條件必須調整，我調整為塞入限制的 outline 後，連續三次 cost 不再下降，即可結束。或者是超過限制時間 1200s 也會結束。所以有時候可能運行很久是因為雖然早就塞進去了，但 cost function 一直都有在下降，所以這樣可以漸漸找到更小的 wirelength。

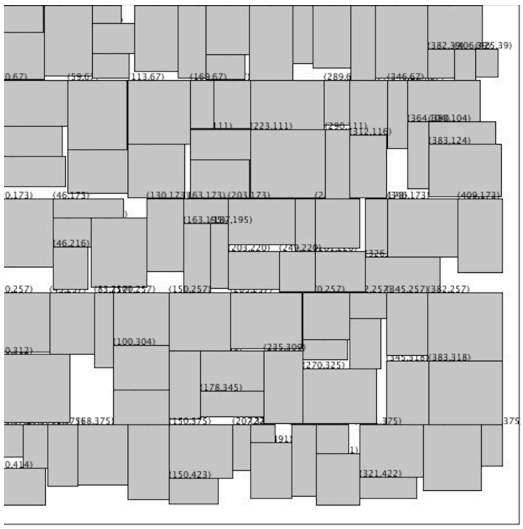
(6)
這次我做的優化，主要是使用變形的 simulating annealing 來實作，不再是只用溫度來控制，基本上這樣是比較容易跳出 local optimal。除此之外就是參數的調控了。那個 5000 次也是實驗之後覺得是最好的，次數太低可能永遠也找不到比目前 best 更好的結果，卡在 local optimal。次數太高就可能在一些根本沒希望的 polish expression 上一直 perturb 浪費時間，結果也找不到更好的答案。
還有在 perturb ComplementOperator 的方面，我也有做一點修改。在講義上推薦的方式是一次 complement 整個 chain，然而我的方法只會 complement 一個 operator，原因是因為我做過實驗，一次 complement 整個 chain 很容易造成每 3000 次重複去運算到相同的 polish expression，因為 chain 的組合相較之下比較少。

(7)
這次的實作和上次比較，有一個很大的不同點是之前的都是使用 b* tree 來實作，並沒有要求 slicing structure，所以在比較上我們應該算是比較吃虧，必須侷限於一個比較不 general 的結構。不過好處是 solution space 應該會比較少，照理說應該能夠較快找到好的解，不過我的結果看起來並不是這麼回事。我覺得有一個很大的原因是，我並沒有在 perturb 的時候去直接對建出來的 tree 做 partial change。而是再從頭重建整個 tree，時間上來說浪費了不少。若能做出這一個部分，想必速度上會提升許多。

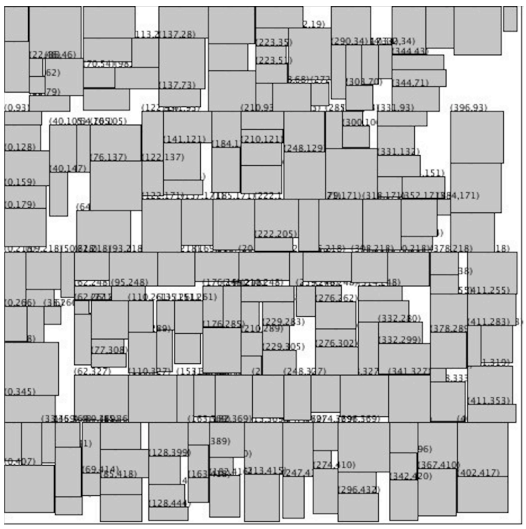
(8)
基本上就是把資料結構的 tree 全部重新複習了一次，當初要決定 preorder postorder 來實作 MinAreaAlgorithm 以及 SetBlocks，每個 order 都有實作過一次，最後是選擇了比較好做的 order。再來基於一篇 paper 來實作出程式真的不太簡單，因為其中的內容很多都不太清楚，或者是本身能力不足，不能充分達到想要的效果，只能照著自己的想法去改善或妥協，也許就是這些妥協造成程式的效果不太佳，有點可惜。

(9) Bouns:

N100



N200



N300