CS6135 VLSI Physical Design Automation

Homework 3:

Fixed-outline Floorplanning with Fixed and Soft Module

112062590 劉俊圻

1. How to compile and execute program, and give an execution example.

(1) Compile

進入 src/目錄，輸入 make 即可完成所有檔案的編譯，並在 bin/目錄底下產生 hw3 執行檔

e.g.

$ cd src

$ make

(2) Execute

直接使用 command 找到執行檔和相對應的輸入 testcase 來執行程式

<execute file> <testcase> <output file>

e.g.

(in src directory)

$ ../bin/hw3 ../testcase/public1.txt ../output/public1.floorplan

$ ../bin/hw3 ../testcase/public2.txt ../output/public2.floorplan

…

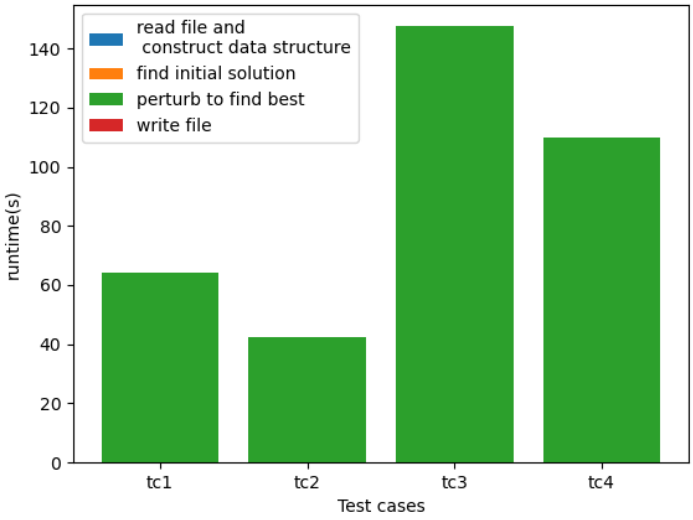
$ ../bin/hw3 ../testcase/public4.txt ../output/public4.floorplan

2. wirelength and runtime

(1) wirelength

Test case	wirelength
public1	228573890
public2	33017687
public3	3439441
public4	117511075

(2) runtime



3. How did you determine the shapes of soft modules?

我在 read file 的過程中就會先找出每個 soft module 符合 aspect ratio 的所有 width 和 height，並且將這些 width 和 height 儲存起來。當我開始建構 initial solution 時，會先從最大的 module 開始擺放，決定形狀的方式也相當直覺：如果遇到 soft module 發生 overlapped 或超出邊界的情況，就從儲存起來的 width 和 height 中尋找可行的 shape，若是在這個位置上都沒有合法的形狀，就將 soft module 的座標位置進行移動，再重複一樣的動作。後續進行 perturb 時也是一樣的概念，進行 soft module 搬移或交換時如果發生不合法的情況，那就先遍歷所有已經儲存過的長寬來找尋是否有合法的形狀，若沒有，則放棄這次 perturb。

我覺得這個方法好處是在進行 initial floorplan 時，決定形狀的方法相當直覺，也一定能找到合法的形狀，因為每個 testcase 都保證能放得下所有 module，我只要從最大的 module 開始擺放(避免小 module 和其他 module 之間隨意擺放造成的空隙)，遍歷所有可行的位置、可行的形狀就一定能找到合法的 initial floorplan。合法的 initial floorplan 對後續的 perturb 的合法保證相當重要，若是 initial floorplan 就不合法，則有可能最後的結果也不合法。

4. The details of your floorplanning algorithm

我的整個 floorplanning 流程可分為尋找 initial floorplan 和 perturb 兩個部分，在進行 initial floorplan 前，我已經先將所有 soft module 符合 aspect ratio 的可能形狀都先找出來並儲存。

(1) initial floorplan Pseudocode

Input: softModules[], fixedModules[]
sort softModules[] in descending order in aspect of their area
placedModules[] = fixedModules[]
sort placedModules in ascending order in aspect of their y coordinate
for each softModule in softModules[]:
  place softModule in coordinate (0, 0)

```
  while (overlap with any other module in placedModules[])
    if (have tried every shape)
      reposition this softModule x coordinate to the right bound of overlapped module
    else
      try another shape
    end if
  end while
  if (left bound of softModule >= right bound of chip)
    reposition this softModule to (0, current y+1)
  end if
  else if (right bound of softModule >= right bound of chip)
    if(have tried every shape)
      replace this softModule to (0, current y+1)
  end if
  add this softModule to placedModules
  sort placedModules in ascending order in aspect of their y coordinate
end for
```

在 initial floorplan 中，從面積最大的 module 開始，每一個 soft module 都會先從(0, 0)座標開始擺放，若是與其他 module 發生重疊，則選擇當前 module 的另一組長寬，一直到找到合法的長寬或所有的長寬都遍歷過。若還是與其他 placedModule 發生重疊並且找不到合法的長寬，則將當前 module 移動至被重疊 module 的右邊界，再重新第一段所敘述的流程一次

執行完上述內容後，當前的 module 必不與任何 placedModule 重疊，但若是當前 module 的右邊界已經大於等於 chip 右邊界，則再需要重新遍歷一次所有的長寬，直到找到不超過 chip bound 的長寬；若都找不到合法不超過 chip 邊界的長寬，則將當前 module 的 x 座標設為 0，y 座標為原來 y 座標+1，再重新所有上述流程

(2) perturb

```
Input: isPlacedMoudle[], softModules[], perturbChance
for i = 0; i = perturbChance; i++
  pick a move randomly: reshape, slide, move, swap
  move:
    pick a softModule randomly
    move this softModule to a random place within chip bound
  end move
  reshape:
    pick a softModule randomly
    pick a random shape of this softModule
  end move
```

```
slide:
   ori = pick x or y axis randomly
   delta =    pick a random number from -5 to 5
   shift this soft module along ori with delta
end slide
swap:
    pick two different softModules randomly and exchange their coordinate
end swap
if (softModules after perturbation is illegal)
   if(HPWL is better then last result)
      use this result as next input
   end if
end if
end for
```

在 perturb 之中，會根據輸入的 perturbChance 變數，來決定可以進行 perturb 的總時間，一次的 perturb 中可以隨機選擇四種動作:move、swap、slide、reshape。在 move 中，隨機選擇一個 module，並且再隨機選擇一個座標，將選擇到的 module 移動至這個座標上；swap 為隨機選擇兩個 module，並交換兩個 module 的座標；slide 為先隨機選擇 x 方向或 y 方向，再隨機選擇[5, -5]的其中一個非 0 整數，再隨機選擇一個 module 根據前述的選擇結果進行位移，以上動作若是有更短的 HPWL，則將當前的 best 結果替換為 perturb 結果，再將這個結果做為下一次的 perturb 基礎。但若是 perturb 之後的結果違法，不論好與壞，直接放棄這次的 perturb。

5. Try your best to enhance solution

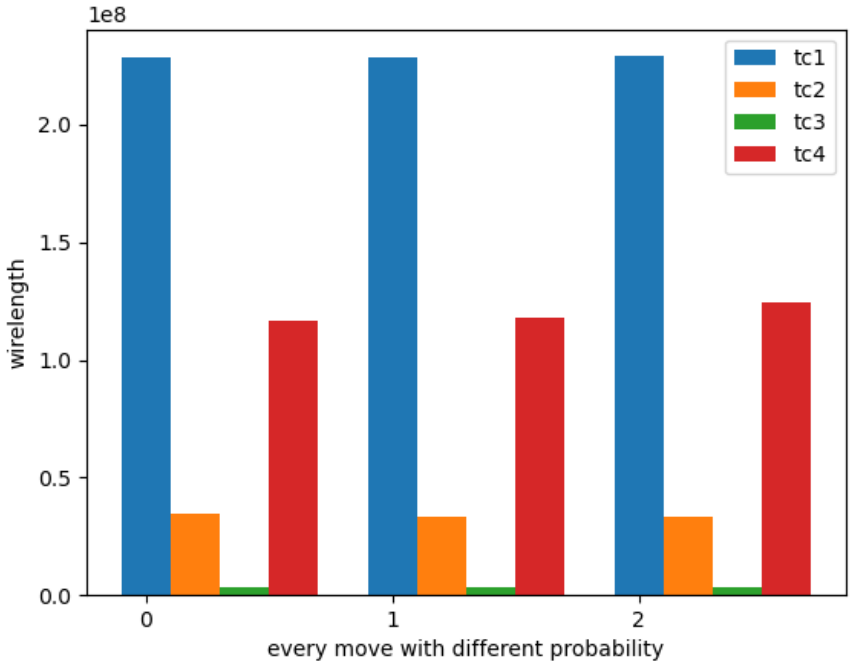
由於 perturb 的動作是隨機選擇進行的，因此我嘗試了幾個不同的機率組合，以下是幾組結果比較好的組合，我認為若是真的實作 simulation annealing 並找到適合的參數的話可能能有更好的結果，但時間不夠了，因此只嘗試了以下幾種不同組合。

0.rotate:0.25 swap:0.25 slide:0.25 move:0.25

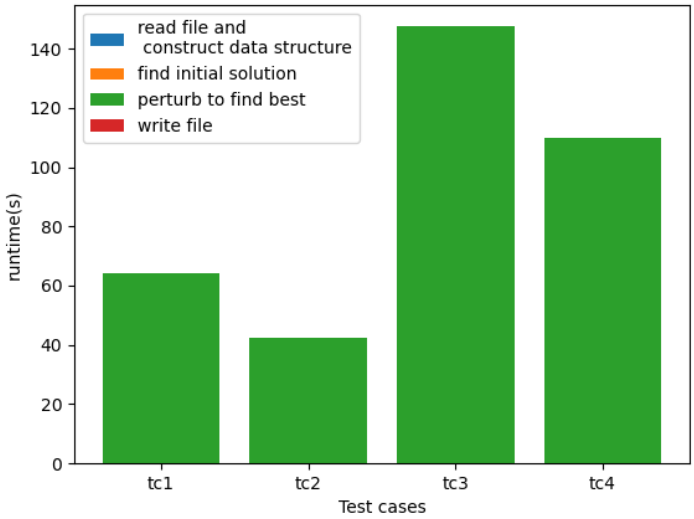
1.rotate:0.3 swap:0.2 slide:0.3 move:0.2

2.rotate:0.3 swap:0.2 slide:0.4 move:0.1

結果並沒有相差很多，我最後選擇了第二種組合



Runtime 則是和前面 2.中的結果差不多，



6. Parallelization

我沒有實作平行化

7. What have you learned from this homework? What problem(s) have you encountered in this homework?

這次的作業我原本實作的是 B*tree，也參考了不少 B*tree 的論文，甚至還找到老師之前寫的

一篇針對 preplaced module 的 B*tree 調整方法，但一直到最後在測試 simulation annealing 時，我發現我不管怎麼調整 cost function 和 simulation annealing schedule 都找不到合法的解，我認為是我的 initial solution 實在太差，但這時已經花了許多時間在 B*tree 上，無法再嘗試可能找不到合法解的辦法，因此我後來我選用的方法偏向暴力法，但至少可以保證 initial solution 會是合法的，後續的 perturb 也只採用合法且 HPWL 比前一次更小的解。在選擇新方法後，因為資料結構重改，我也因此學到了動態多形、C++ shallow copy 和 deep copy 的相關知識。我認為我一開始 B*tree 的 initial solution 若再進行調整應該會有機會找到合法的 solution，後續的方法應該也還有改進空間，奈何時間不夠，希望將來還有機會能繼續未完成的方法。