

VLSI Physical Design Automation Final Project

Group 1

1. Team member
王領崧 107062107
廖恩宇 110062603
周伯宇 110062552

2. How to compile and execute your program

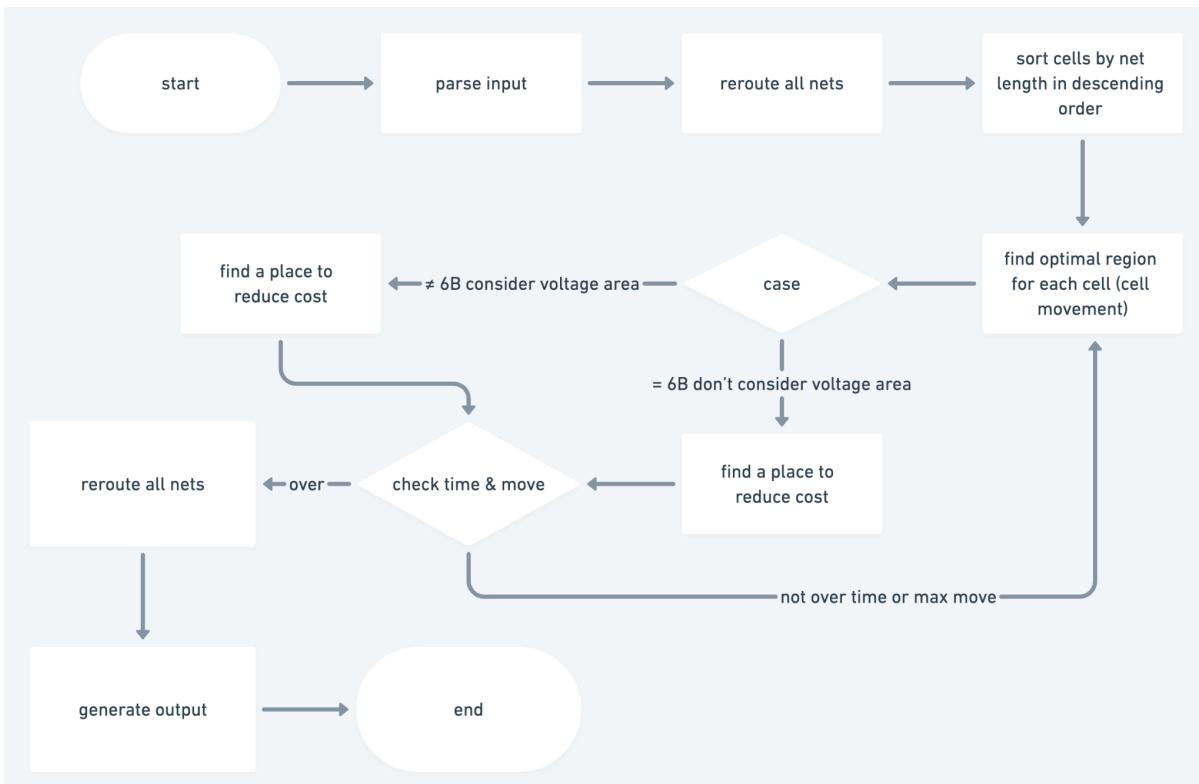
- Compile: cd Final_Project/src -> make
- Execute: cd ..bin -> ./cell_move_router ./testcases/case1.txt
..../output/case1.txt

3. score/runtime

	1	2	3	3B
final score	49.9	13.34	7868.6	8174
runtime	0.00247	0.00284	2.7591	2.9044
	4	4B	5	5B
final score	453465	470379	992850	993484
runtime	306.069	313.935	2057.53	2112.27
	6	6B		
final score	583251	622250		
runtime	3590.43	3591.28		

total score: 4131784.84

4. The details of your implementation and algorithm



架構: 基本上就是 flowchart 所示。我們的演算法主要可以分為 3 部分 : initial-rerouting -> moveCell -> post-rerouting

在 load & parse input file 後，我們利用 steiner-tree (Third-Party) 進行 initial rerouting 和 reduceRoutingSegments。

接下來是 moveCell 的部分，首先將 cell 由 netLength 大到小做排列，依序拿出 cell 進行移動，我們會計算出 cell 可以移動的 candidatePos (optimal region +1)，然後將 candidatePos 進行 shuffle，從第一個 pos 開始計算，如果 cost 會降低，則將 cell 移動到新的位置，換下一個 cell；反之則換下一個 pos 計算，直到 cost 降低 or candidatePos 都跑過了。moveCell 會持續到移動的 cell 數量 \geq maxCellMove or overtime。有一點比較特別的是，針對 6B 這個 testcase，我們經由跑實驗發現加入 voltageAreaConstraint 的 cell 如果加入 moveCell 中，會使得結果變差，而其他的 case 都有明顯的變好，因此，在選擇 moveCell 時，如果這個 testcase 為 6B，就不會將 voltageAreaConstraint 加入。

完成 moveCell 後，會進入 post-rerouting，針對已經 moveCell 後的 routing 進行最後的優化，然後輸出結果。

5. What tricks did you do to speed up your program or to enhance your solution quality?

1) 增加 voltageAreaConstraints cell 的移動

原本的 sample code 僅會嘗試移動完全沒有 constraint 的 movable cells，我們將具有 voltageAreaConstraint 的 cell 也加入考慮，然後面計算 candidatePos 時，只考慮那些在 optimal region 裡面的 voltageAreas。

2) 增加 moveCell 的 round

原本的 sample code 的 Mover::move() 僅會將所有 movable cells 移動一輪，但經由跑實驗後發現，最後移動的 cell 數量 $<$ maxMoveCount，代表仍有進步的空間，因此我們多增加一輪 round 來檢視。不過有一點要注意的是，因為會進行很多輪的檢視，同一個 cell 可能會被移動數次，但 moveCnt 應該只能被累加一次（因為都是同一個 cell），所以我們先建立一個 table，紀錄 cell 的是否有被移動過，而 moveCnt 只會在 table 從 false -> true 才會做累加。

3) 預先計算 optimal region 的 cost reduction

原本的 sample code 在選擇 optimal region 的 position 做 cell 移動時，是將所有 position 做 shuffle，然後挑第一個來進行移動，但是這樣不保證移動後的 cost 降低，我們也從實驗中驗證了此點 (6, 6B testcases 在 moveCell 後 cost 反而會升高)，因此為了兼顧 runtime & performance，我們新增“試 move”的步驟，在 move cell 到新的 position 前，會先模擬移過去的 cost gain，如果是增加就嘗試另外一個 position，直到第一個 cost 減少的 position 被找到。

4) 調整 candidatePos region 的大小

將 candidatePos region 的大小，從 optimal region 的 +5，調整為 optimal region 的 +1 (此改動由實驗數據得知)。

6. What have you learned from this project? What problem(s) have you encountered in this project?

1) starfish paper 的閱讀

起初 trace 完 sample code 後，雖然大致理解 code 的運作模式，但是卻不知從何優化起，因此先去閱讀助教提供的 paper - starfish。我們從中了解到 starfish 是如何從 initial rerouting -> movecell -> A* partial rerouting -> post-processing 來解決問題。此次的作業也是參考預先計算 cellMovement 的 cost，以及 greedy cell

put-back process 的概念進行優化。

2) net ordering

Sample code 在 GraphApproxRouter::rerouteAll() 的地方，是將 net 根據 netLength 由小到大的順序 reroute，與老師上課講的相符（小的先繞比較不會干擾，結果比較容易成功也比較好），然而，反覆實驗後，發現如果從 netLength 最大的開始 reroute，結果會變好，我們認為可能是測資方面，如果先繞 netLength 小的 net，雖然會降低 netLength，但同時也會把一些重要的 gridResource 佔掉，導致 netLength 大的 net 無法透過那些 gridResource 的 routing，把 netLength 降低更多。

3) 關於 steiner-tree 的座標轉換

由於 steiner-tree 是 Third-party，因此在使用上需要按照它的作法，這邊 sample code 所展示的方式十分厲害，值得學習。原本我們以為會把 3D 的 net pin 壓成 2D，類似 nthu route 的概念，然後再去建 edge & graph。結果 sample code 是先建出 R^*C^*L 的 vertex 數量，然後將 3D coord encode 為 1D 的 coord，再把彼此有相連的 pin 建出對應的 edge，經由 steiner-tree 的處理後，再將 1D coord decode 回 3D coord，把 edge 所對應的 routeSegment 接回來。