CS6135 VLSI Physical Design Automation

Final Project: Routing with Cell Movement Advanced

Team 3
Member: 109062526 楊皓崴 / 109062556 李濬安

- ## How to compile and execute your program ?

可以根據以下資料來去 compile & Run. (src/README)

src > $ README

```
1   --  How to Compile
2       In this directory, enter the following command:
3       $ make
4       It will generate the executable file "cell_move_router" in "Final_Project/bin".
5
6       if you want to remove it, please enter the following command:
7       $ make clean
8
9   --  How to Run
10      In this directory, enter the following command:
11      Usage: ../bin/<exe> <testcase file> <output file>
12      e.g.:
13      $ ../bin/cell_move_router testcases/testcases/case1.txt output/case1.txt
14
15      In "Final_Project/bin/", enter the following command:
16      Usage: ./<exe> <testcase file> <output file>
17      e.g.:
18      $ ./cell_move_router testcases/testcases/case1.txt output/case1.txt
```

- ## The score and the runtime of each testcase.

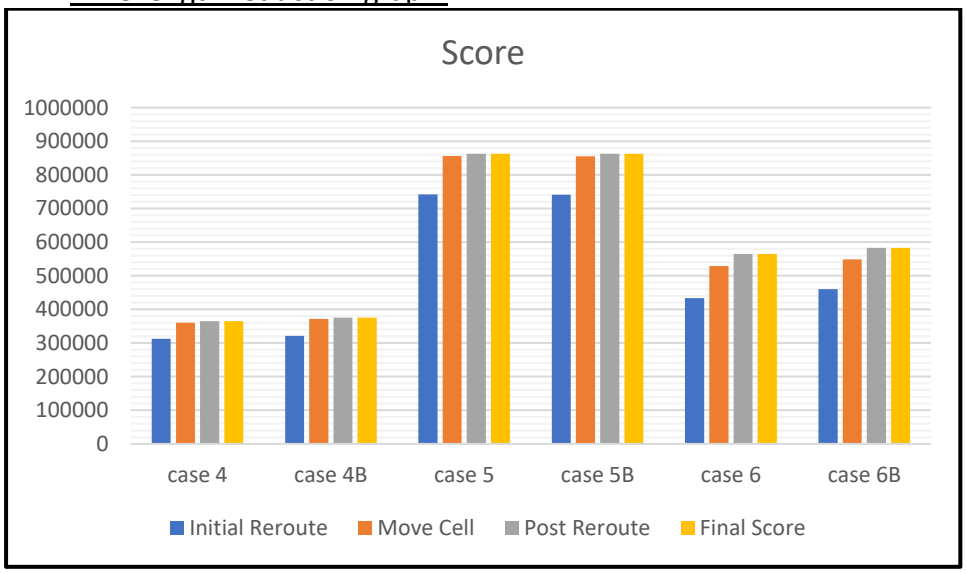
➢ Runtime:

Runtime (s)					
Testcase	I/O	Initial Reroute	Move Cell	Post Reroute	Total Time
case 1	0.001266539	0.000562332	1.33245	0.000261129	1.33454
case 2	0.001074282	0.00046328	1.33453	0.000202438	1.33627
case 3	0.043699	0.748877	8.62089	0.629034	10.0425
case 3B	0.041454	0.872607	6.31492	0.764479	7.99346
case 4	7.203	54.9454	207.248	43.7536	313.15
case 4B	9.0409	54.9818	231.697	43.5853	339.305
case 5	118.164	122.354	1126.9	139.112	1506.53
case 5B	120.405	119.884	1139.95	133.531	1513.77
case 6	100.254	285.786	2783.96	341.59	3511.59
case 6B	144.274	268.238	2369.9	277.138	3059.55

➢ Score:

Score				
Testcase	Initial Reroute	Move Cell	Post Reroute	Final Score
case 1	17.3	26	0	43.3
case 2	0	8.54	0	8.54
case 3	4967.2	1142.4	25.6	6135.2
case 3B	5055	1175.8	18.2	6249
case 4	312682	47666.8	4212	364560.8
case 4B	321686	49939.7	4172.7	375798.4
case 5	742264	113746	7054.5	863064.5
case 5B	740899	114269	7153	862321
case 6	433887	94597.1	36764.9	565249
case 6B	460202	88873.4	33958.5	583033.9

➢ Wire length reduction graph:



可以發現到我們 Score 成長幅度主要是在 Initial Route 和 Move cell 這兩步驟中，而後面的 Post Reroute 則相較之下成長幅度較小。

➢ Grading result:

```
[g109062526@ic22 ~/Final_Project_grading]$ bash Final_Project_grading.sh
|-------------------------------------------------------------|
|                                                             |
|    This script is used for PDA Final_Project grading.       |
|                                                             |
|-------------------------------------------------------------|
grading on group1:
  testcase |           score |     runtime | status
     case1 |            43.3 |        0.82 | success
     case2 |            8.54 |        0.84 | success
     case3 |          6135.2 |        7.73 | success
     case4 |          364560 |      325.93 | success
     case5 |          863064 |     1553.55 | success
     case6 |          565249 |     3394.41 | success
    case3B |            6249 |        7.95 | success
    case4B |          375799 |      356.80 | success
    case5B |          862322 |     1523.67 | success
    case6B |          583034 |     3393.76 | success
|-------------------------------------------------------------|
|                                                             |
|    Successfully generate grades to Final_Project_grade.csv  |
|                                                             |
|-------------------------------------------------------------|
```

- # The details of your algorithm

1. Data parsing:

   讀取測資的部分我們完全按照 [1] 的做法。

2. Initial Reroute:

   首先，我們在 reroute 前，先建了 Congestion Aware Net Order。做法是先考慮 net weight，net weight 較大的優先進行 reroute，而 net weight 相同者則是用每條 net 上的 route getRouteCost()的總和再乘以該 net 上 pin 的數量來遞增排列。其中，在對一條 net 進行 reroute 時，我們對 net cost 上的計算進行改動，同時考慮到 LayerFactor 以及 route 兩端點連接的 grid 目前的 supply，如果這兩個 grid 其中有 overflow 的話則會對這段 route 進行懲罰，給予 op_v 的 cost。而另一方面，我們還給予每一線段 cap_v1+cap_v2 的 cost，也就是剩餘的 capacity 多則會有較多的 cost，因為剩餘 capacity 多的線段可能是冷區反而會增加整體的 net length。接著，再去做 Steiner-Tree-2-Approximation Routing，得到新的 Refine 後的 routing solution。

   - $Cost_{initial\_route} = LayerFactor * \alpha + op\_v + (cap\_v1 + cap\_v2)/2$

   另外，這步驟中，Steiner-Tree-2-Approximation Routing 所使用的 reroute area 是原先 routing 中的範圍再往外擴增一個固定的單位長度（e.g., MinR - k1, MaxR + k1, MinC - k1, MaxC + k1, MinL - k2, MaxL + k2），其中，MinL 還需要考慮到 min routing layer constraint。

3. Movable Region Determinaiton

   首先，我們先對 cell 進行排序，排序方式是依照連接此 cell 的 all net length 降序移動，計算方式是將此 cell 連接到的 net 的 cost（LayerFactor * net weight * net length）再乘 net weight。

   - $Cost_{cell\_order} = LayerFactor * net\ weight^2 * net\ length$

   接著，我們嘗試將每個 cell 移動到 optimal region 中，所謂的 optimal region 計算方式如下：對連接到此 cell 的所有 net 的 bounding box（不考慮此移動中的 cell）取 x 軸的兩個中位數（x1, x2）以及 y 軸的中位數（y1, y2），而（x1, y1）是 optimal region 的左下角，（x2, y2）是 optimal region 的右上角。得到 optimal region 後，我們將對 optimal region 中可能擺放的位置進行排序，排序方式是先用 FLUTE 來模擬將 cell 擺放上去後的 net length 再乘以(num net pin)^2，由大到小排列，也就是說我們會優先將 cell 擺到線長較短的 grid 上。

   - $Cost_{candidate\_regio\_order} = Flute\_Length * Net\_Weight * Net\_NumPins \wedge 2$

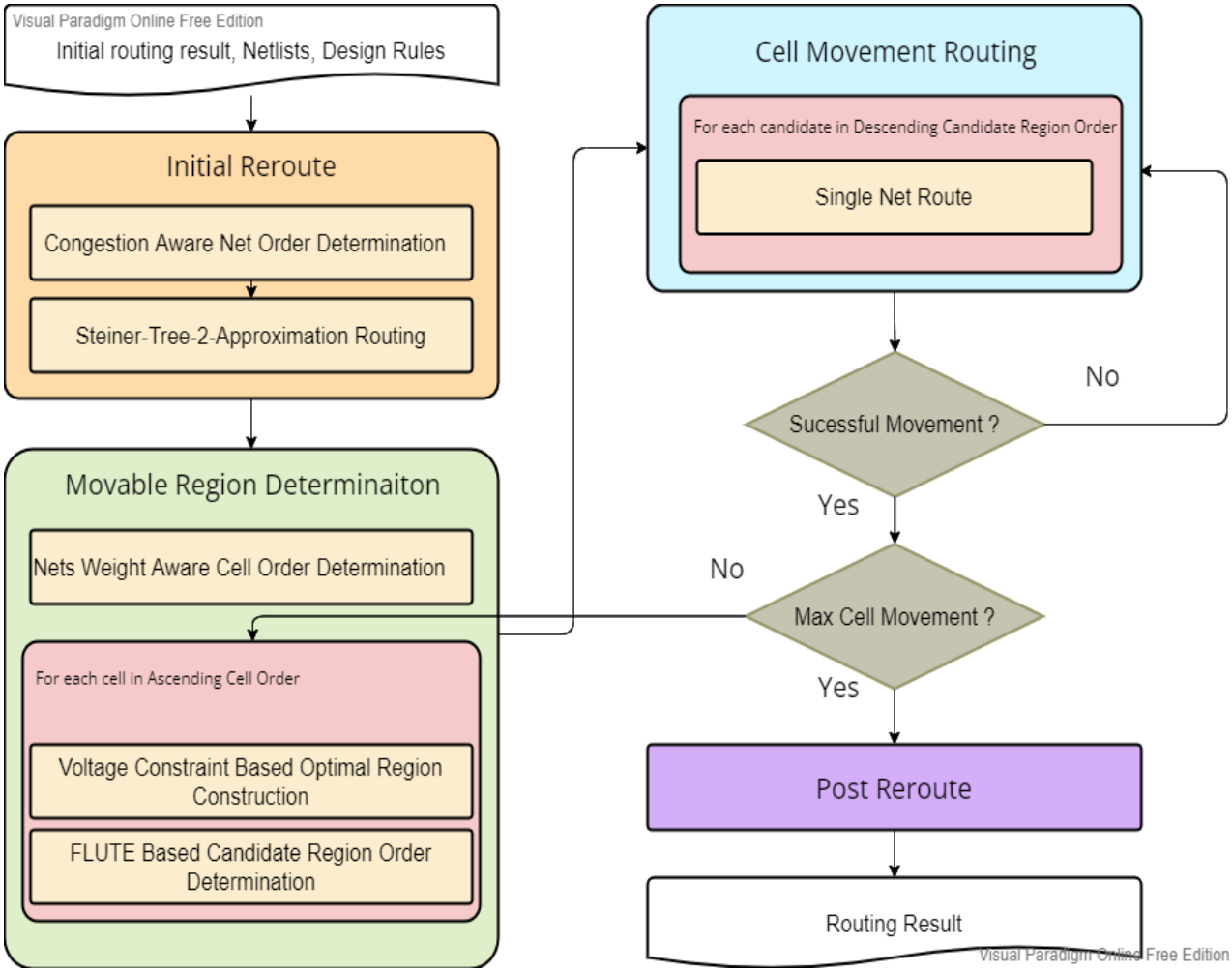
   另外，考慮到某一部份的 cell 是有 votage area 的限制的。對於這些 cell，我們在計算 optimal region 之後，會在 optimal region 中的 grid 再尋找那些符合 voltage area constraint 的 grid 來加到 CandidatePosition 中。

4. Cell Movement Routing
   接著就對目前的 movable cell 按照其 candidate_region_order 順序對 cell 進行 single net route。如果有任何一條連接其的 net，重繞後會有 overflow，則會將此 cell 移到原處，並進行下一個 cell 的 movement routing。，若無 overflow，則會去判斷是否已經達到 Max_Cell_Movement，是的話就結束 algorithm。

5. Post Reroute
   根據上述移動完(重新擺放後)的結果後，會再做最後一次的 reroute，並且跟 initial reroute 一樣是 Congestion Aware Routing，並使用 Steiner-Tree-2-Approximation Routing 得到最後的結果。

6. Output File
   將結果 output。

● Flow Chart



- # What tricks did you do to speed up your program or to enhance your solution quality?

在這次作業中，我們嘗試了以下幾種方法來改善最終的 performance。

1. FLUTE:
   利用 FLUTE 來模擬 cell 擺放後的結果來計算線長，並以此作為 CandidatePos 的排序依據。

2. Steiner tree cost:
   透過修改 rerouting area 中，各個連接不同 grid 的不同線段的 cost 來協助 reroute。我們將 Horizontal Layer 或 Vertical Layer 的 cost 設定為 LayerFactor(L)*2 + op_v + (cap_v1 + cap_v2) / 2，而 via 的 cost 則是設定成 LayerFactor(L)+LayerFactor(L+1)。

3. Voltage area:
   針對有 voltage area constraint 的 cell，我們會先找出它的 optimal region，接著再從 optimal region 中找出符合 voltage area constraint 的 grid 來作為此 cell 的 CandidatePos。

4. Virtual add and route:
   此方法是為了再計算 CandidatePos 的優先順序時，更精確的排序最佳的 cell 擺放位置，因此使用 virtual 的 add and route 就是將此 cell 擺放上去後進行 single net route 在計算 getRouteCost()，但最後會再將此 cell 還原回原本的狀態。

5. Multiple cell move:
   原本的 cells move 只會對每一個 cell 進行一次的重新擺放嘗試，因此我們希望可以重複進行 cell move 來改善 performance。但因為有 MaxMoveCnt 所以在某些 testcase 中並不能發揮作用。

6. Multiple reroute:
   透過重複進行 rerouteAll 來改善 performance。

7. Bounding_box_surrounded net ordering:
   此部分是用在 rerouteAll 的時候，需要對 net 進行 reroute 順序的排序，可以透過計算此 net 的 bounding box 中包圍了的 pin 個數來決定順序。

8. Net ordering weight consider first:
   此處是指在 rerouteAll 時對 net 進行 reroute 順序的排序，我們先考慮那些 net weight 較大的 net 來進行 reroute，來確保較重要的 net 可以有優先權，來降低 total cost。

9. Boundary enlarge:
   此部分是 reroute area 中的 k1 以及 k2（上述步驟 1 提到的 MinR - k1, MaxR + k1, MinC - k1, MaxC + k1, MinL - k2, MaxL + k2），這裡如果將 k1 和

k2 調大，則會大幅度的增加 performance，但是同時因為增加了可以 route 線的區域，所以也會需要更多的計算時間，最後，我們將 k1 設定為 6，k2 設定為 2。

10. Candidate region order:
    在排序 CandidatePos 時，除了考慮 FLUTE 給的線長之外，再乘以 pin 的數量，若是包含大量 pin 的 net 則會優先。

- # What have you learned from this project? What problem(s) have you encountered in this project?

  這次 project 中，我們學到了很多 routing 的演算法以及 placement 的一些技巧 (ex.optimal region)，也了解到 Preprocessing 和 Post processing 的重要性。並且也明白教授上課所提到的 Net order 也是對 routing 結果有相當大的影響，而我們這次也花很多心力在這上面。最後也明白到此作業的難度很高，要 concern 的東西很多，雖然結果不是特別理想，但也是受益良多。

  而面對的問題有很多，而最大的問題是在 Timing 上面，因為我們讓 routing 的 boundary 擴大，也讓時間大幅增加，所以我們在時間及 performance 上做了一些 trade off。那其他問題還有是，我們起初希望在選 candidate region 時，是根據題目給的 cost (routing cost * layer factor)，然而發現結果會很不好，是因為每個 cell 就會太傾向於 cost 最小的地方，導致 routing resource 不足，因此常常會 overflow。那我們最後是用 FLUTE*Net pins 的 cost 來決定，也就沒有考慮 layer factor。

Reference:

- 程式參考: https://github.com/jacky860226/ICCAD-2021-B -- [1]
- Starfish: An Efficient P&R Co-Optimization Engine with A*-based Partial Rerouting, Martin D.F. Wong
- NTHU Route2.0