Routing With Cell Movement

曾煒哲, 洪世昌, 柯昱豪
NTUEE

Abstract: In this report,we introduce a placement & routing-separated method to accomplish the CAD contest problem b (http://iccad-contest.org/2020/Problem_B/ProblemB_0224.pdf)

Index term – Force-diredcted method, semi-perimeter method, LT algorithm, shield algorithm

Introduction

Splitting the physical implementation problem to 2 sub-problems (placement and Routing) is a common approach to solve the modern complex physical implementation problem. Inside these 2 major steps, the common approach is to further divide the placement/routing problem into more sub-problems. The purpose of this divide-and-conquer approach is to make sure this complex problem would be manageable and be converged in reasonable runtime.

Placement

Algorithm we applied

- Force-directed method

靜力平衡法，想法類似將 cell 與他的每一個 fanout 接上彈簧，找到靜力平衡處，因為每一個 fanout 對其重要性相同，所以我們對其 fanout 的 X, Y 座標求平均值即為選點。

- Semi-perimeter method

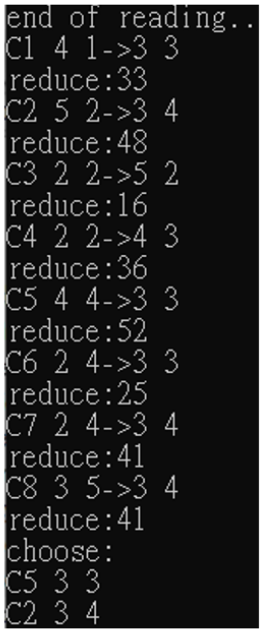
半周長估計法，用一個長方形將所有的 pin 框起來，此正方形的半周長便是估計的線長。所以我們找到 cell 的 fanout 的 X, Y 座標的最大最小值，X 最大值與 X 最小值的差加上 Y 最大值與 Y 最小值即為我們的估計線長。

Procedures

對每一個 cellInst 進行以下步驟：
Step 1:對其 fanout 做 Force-directed method，找出若單移動該 cell 的最佳選點。
Step 2:做 Semi-perimeter method，估計出移動該 cell 至選點的線長。接著將估計的線長與原線長比較，計算出減少量。
最後，對每個 cell 的減少量進行排序後，在 MaxCellMove 的限制下移動 cell 到最佳選點。
以 case1 (Fig. 1. (a)) 來做說明，圖中的"C1 4 1->3 3"，代表 C1 原本座標在(4, 1)，進行 force-directed method 後的選點是(3, 3)，而他下面的"reduce:33"則是進行 semi-perimeter method 後估計移動 C1 會減少的線長。最後，根據 reduce 的大小來排序，又因為 MaxCellMove 是 2，所以選擇了 C5 跟 C2 來移動。

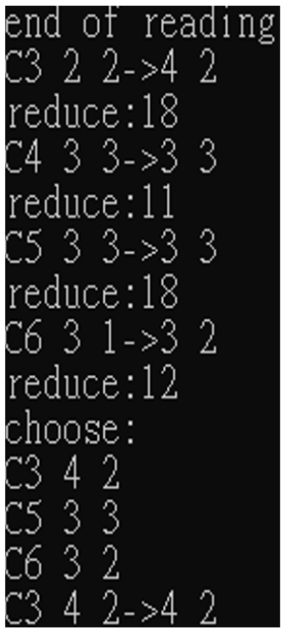
Problem & solve

其實上述步驟還有一些瑕疵之處，所以我們又進行了些演算法的優化，如下述：
●問題 1：
做 Force-directed method 時只考慮移動該 cell，但此時移動了多個，每個 cell 的 fanout 可能已經不在先前的位置了，所以



```
end of reading..
C1 4 1->3 3
reduce:33
C2 5 2->3 4
reduce:48
C3 2 2->5 2
reduce:16
C4 2 2->4 3
reduce:36
C5 4 4->3 3
reduce:52
C6 2 4->3 3
reduce:25
C7 2 4->3 4
reduce:41
C8 3 5->3 4
reduce:41
choose:
C5 3 3
C2 3 4
```

(a)



```
end of reading
C3 2 2->4 2
reduce:18
C4 3 3->3 3
reduce:11
C5 3 3->3 3
reduce:18
C6 3 1->3 2
reduce:12
choose:
C3 4 2
C5 3 3
C6 3 2
C3 4 2->4 2
```

(b)

Fig. 1. (a) testing output of placement of case1. (b) testing output of placement of case2.

已不是最佳選點。
※解決方法：
對已經移動過的 cells 再進行數次上述的動作，因為每重複一次，選點與原本的點位置會越差越小，以此來逼近最佳選點。

●問題 2：
若進行完 force-directed method 後的選點，該選點的 avalibility 並不夠放下該 cell。
※解決方法：
對該選點的左右前後進行嘗試，嘗試的順序為 availability 越大，越先做嘗試。若可以成功放入，則把該點更新為最佳選點；若全都不行，則退回原位置或上一次 recursive 所得到的選點。

●討論：
Semi-perimeter method 只在 2D 上進行估計，並沒有考慮到垂直的 route，這樣計算出來的 reduce 不會有偏差嗎？

因為 reduce 只是我們選擇哪一個 cell 該移動的依據，所以只

需要能夠比較就好，而我們認為每一個 cell 在垂直方向的線長並不會差異太大，故維持原估計法。

Pseudo code

```
Algorithm  placement(cells)
for i (0  to  # of cells)
if (Cell is moveable)
        Force-directed method:
        best position = the average of row and col of the fanout of cell
        if (avalibility of best position < 0)
                try the Ggrids near by the best position.
                The try order is ordered by the avalibility of each Ggrids.
        Semi-perimeter method:
find the maximum row, col and minimum row, col of the fanout of cell
    estimation = (max row - min row) + (max col - min col)
        reduction = (original fanout length) – estimation
order = sort by reduction of cells
move the first # of MaxCellMove by the order
reset some variables and routes of moved cells
placement(moved cells)   // recursive
```

Psuedo code: Algorithm of placement

Routing

前置作業:

I. 建立 pin list

首先我們先將一個 net 的 pin(s)存成一個 list, 每個 pin 的 next_pin 都是還沒被加進 list 的所有 pin 中離自己最近者, (這裡距離使用 Manhattan distance), 而待會在一個 net 的繞線過程中我們會從 input 給我們的 pin[0]開始一個個地將 pin list 中的 pin 連接起來, 如此一來可以保證每次接線的距離不會過長。

II. Decide base routingLayer_H and base routingLayer_V

有了所有的 pin 我們也可以決定'H'向的 routing layer 以及'V'向的 routing layer, 決定的方法就是從 min_routing_layer 到最高的 layer 中, 找一個 layer 使所有 pins 到該 layer 的距離和最小, 我們取此 layer 與緊鄰其上的 layer, 得到兩個相鄰而方向不同的 layer. 在之後的操作中若沒有遇到 ggrid 爆掉的狀況, 這個 net 'H'向與'V'向的 route 都會放在這兩個 layer 上。

Routing phase1

I. Plot by routing_2D, check by routing_3D

先不考慮跨 layer 的接線, 以俯視的角度觀察這個 net 的所有 pin, 我們將 3D 簡化為 2D routing problem:

我們的想法是, 如果先一次把 net 中的所有 route 繞好再重新調整去避開 full 的 ggrids, 會很不好操作; 如果能在繞每一條線的過程中都做完避障, 後面就不用大費周章的重新繞線. 因此, 繞一個 net 的過程中, 對於已經連接好的 pin(s)與 route(s), 若我們要再加一個未連結的 pin 進去, 則可以考慮兩種繞線的方法: T 形繞線與 L 形繞線。

II. T 形繞線

如 Fig.2 , 對於一條已經連好的 Route r1, Pin to_be_added 可以連一條 Route r2 到其上, 使得兩條 route 形成一個 T 字形的架構。

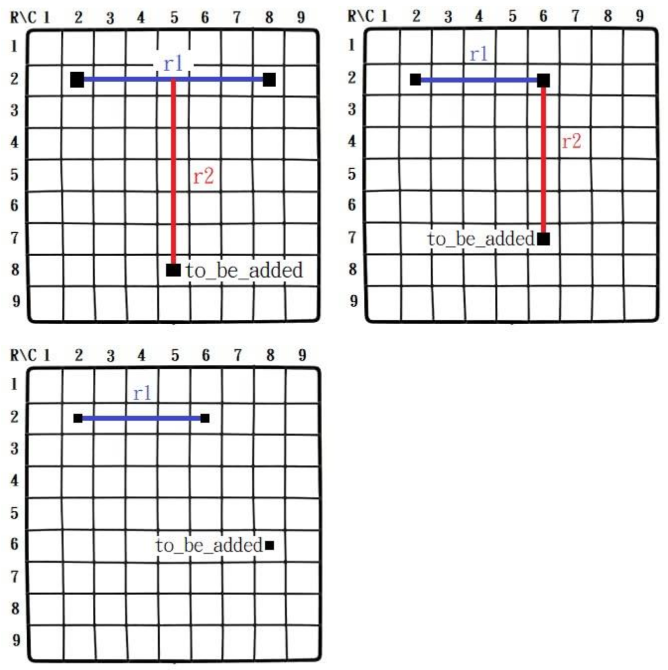
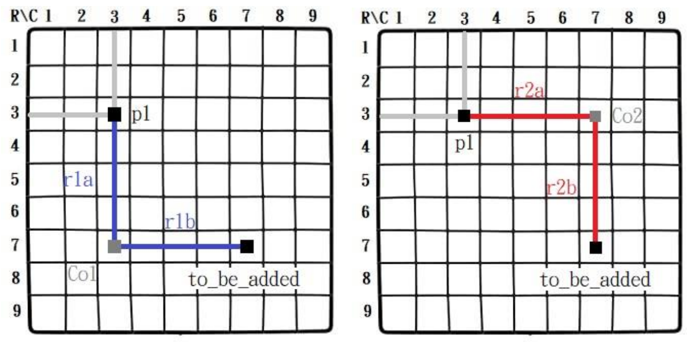


Fig.2 . (左上)T 形繞線 succeed (右上)T 形繞線 succeed
(左下)T 形繞線 fail

III. L 形繞線

如 Fig.3，對於一個已經連好的 Pin p1, Pin to_be_added 可以連兩種 L 形架構 L1(r1a, r1b) and L2(r2a, r2b)到 p1. 在未考慮 ggrid overdemanded 的狀況下, L 形繞線不可能 fail。



(a) (b)

Fig.3 (a) L 形繞線: L1 (b)L 形繞線: L2

接著我們會從以上方法所得到的 route(s)中, 尋找最短的繞法, 比較步驟如下:

Step1.

For all connected routes in the net, find r_exist s.t. r_added got from T_routing(r_exist, p_added) reaches minimum, and record such T_routing instance. Skip this step if all T_routing fail.

Step2.
Of all the connected pins in the net (except the ones contained in the r_exist recorded in Step1), choose the pin (including the existing corners) that is nearest to p_added. Create L1(r1a, r1b), L2(r2a, r2b), and record them. (The so-called "nearest" refers to having minimum Manhattan distance.)
Step3.
Decide the priority of T and L. Compare the length of r_added recorded in step1(if T_routing succeeds) with the length of L1 or L2 recorded in step2. The routing with smaller length should be considered with higher priority.
Step4.
Decide the priority of L1 and L2. As indicated in figure. the one with its corner nearer to p_added's next_pin should be considered with higher priority.
Step5.
According to the priority order decided in step3 and step4, execute routing_3D to check if ggrid occupied by that routing instance is overdemanded. If yes, move on to the next priority routing.
details in routing_3D:
For all the routes in the input 2D routing instance, we place the horizontal ones onto routingLayer_H decided previously, the vertical ones, routingLayer_V. Then we connect horizontal routes with vertical routes by creating cross-layer routes among the two routing layers. Pins that are meant to be connected via this routing instance should be linked as well, so we create more cross-layer routes to accomplish the task.

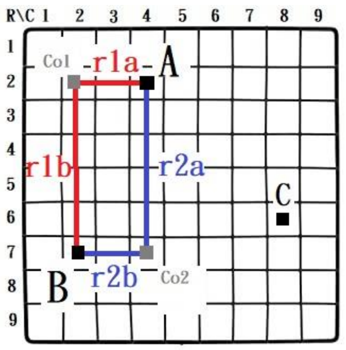


Fig.4

Fig.4 中的 A 點為 p_exist, B 點為 p_added, C 點為 p_added 的 next_pin, 考慮 Co1, Co2 與 C 的距離, MHdist(Co1, C) = 10 > MHdist(Co2, C) = 5, 則取 r2a, r2b append 進 2D routes 中。

If the three routings all fail, skip this p_added, and move on to the next_pin of p_added. Hope that in the next iteration, this won't be the case.

After executing the above procedure to the last pin in the pin list, if there are still unconnected pins left, do more iterations to these unconnected pins until no pins can be further connected through the iteration. If there are still unconnected pins left, we should move on to Routing phase2.

Routing phase2

Routing phase1 會失敗通常是因為 routingLayer_H 或 routingLayer_V 已經有很多 full 的 ggrids(可能是先前繞的 net 有不少集中在這兩個 layer 上, 或是這兩個 layer 有密集的 blockage……), 因此在 Routing phase2 我們會將 routingLayer_H 放到 routingLayer_V 之下, 或將 routingLayer_V 放到 routingLayer_H 之下, 直到 routingLayer_H 或 routingLayer_V 觸底(也就是 min_routing_layer 的限制), 再回到最開始的 routingLayer_H, V, 將 routingLayer_H 提到 routingLayer_V 上, 或將 routingLayer_V 提到 routingLayer_H 上, 直到 routingLayer_H 或 routingLayer_V 觸頂(即最高的 layer), 每升或降一次 layer 就再對 unconnected pins 做 Routing phase1 的 iteration, 理論上只要每個合法的 layer 都考慮過就能解決 routing phase1 的缺失。
然而, 要是所有相鄰的 H-V layer pairs 都考慮過後, 還是有留下 unconnected pins, 我們又該如何處理呢?這時候就必須進入 Routing phase3, 用比較 general 的方法來處理避障的問題。

Routing phase3

首先, 先介紹一套新的演算法, 我們稱之為 shield algorithm:
對於一條被 full ggrid(s)阻斷的 route, 我們先收集其經過的所有 full ggrid(s), 再去 recursive search 與這些 ggrid 相鄰的其他 full ggrid(s)(所謂的相鄰可參考 figure. 除了中心外的 26 個 grid 都與中心 grid 相鄰), 最終找到阻斷 route 的 full ggrid 群, 接著我們可以建立這個 full ggrid 群的 bounding box。

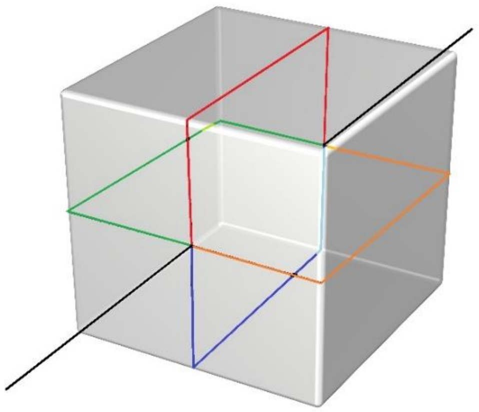


Fig.5

如 Fig.5 所示, 有了 bounding box 我們就可以讓原本 route 的起點與終點分別對 bounding box 做投影, 再用圖中的四種連線法繞在 bounding box 上, 取總長度最短者為我們避障的 routing。
若起點或終點被包在 bounding box 裡面, 我們可以對 bounding box 的六面牆分別作投影, 若沒碰到 full ggrid(S)就可以用上述的方法取避障 routing。
若 bounding box 在 corner, 我們一樣能用投影的想法取得避障 routing。
, 而且還不用比較四種線路的長度, 因為在數學上如圖中的繞線是最短的可能。

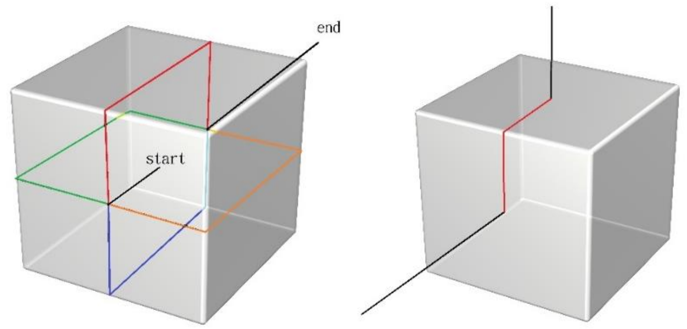


Fig. 6

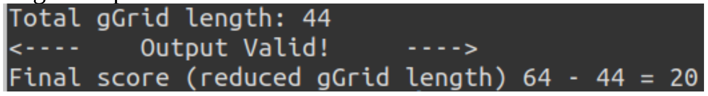
接著, 附上整個 routing 在 phase1,2 的 pseudo code:
In Appendix

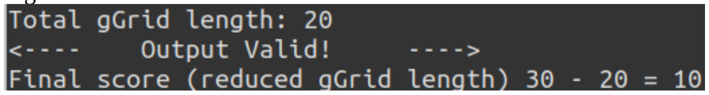
多個 net 做 routing 的順序
Net 的 pin 數越少就越先做 routing, 若兩個 net 的 pin 數一樣多, 則 pins 的 bounding box 越小, 該 net 就越先做 routing
會採用這樣的順序, 我們主要的考量是: iteration 的演算法對繞線自由度越高的 net 越有利, 如果繞線自由度太低就很難在 iteration 後避開障礙物, 因此應該比較早做 routing; 而繞線自由度高的 net 比較能藉由 iteration 進行避障, 因此繞線的優先序可以不用太前面.

Prospect

在 force-directed method 方面，並沒有考慮到擁擠度，所以可能 input file 一大，cell 便擠到一塊，若 Ggrids 的 availability 不夠，之後的繞線反而會更麻煩，甚至會增加線長。所以未來希望能對擁擠度方面適度修改演算法，甚至可以稍微先進行一點 routing，確認選點的可行度。
在 semi-perimeter metod 方面，他並不是一個精確的估計法，不僅如前述的討論所述的只考慮到 2D，如果每個 cell 的 fanout 都很大量，每個 cell 得到估計值都會很接近，這時我們可能就無法精確知道該移動哪些 cell。希望之後能對此演算法做更多優化。
最後，因為在移動 cell 的過程中，很多資料需要移動、確認或更改，而且我們又進行了 recursive，所以希望能再對資料結構更有研究，藉此提升速度及記憶體用量。

Results:
We realize our ideas with c# ,and compiled by g++(version)
For the cases gien by the CAD contests ,we can successfully run through two of those(case1& 2), and we are also able to run the evaluator(Also given by CAD contest) on our environment,
and thus getting a result with output length shorter then the original input.


```
Total gGrid length: 44
<----     Output Valid!     ---->
Final score (reduced gGrid length) 64 - 44 = 20
```
Fig.7 case1's result


```
Total gGrid length: 20
<----     Output Valid!     ---->
Final score (reduced gGrid length) 30 - 20 = 10
```
Fig.8 case2's result

CONCLUSION

Although we didn't apply it in our code, we also found out that gradient searching method is a more powerful case in finding the new CellInsts' place. we also learnt another method for approaching faster re-routing which is a placement for small amount of cells once a cycle and reroute those.Nevertheless, this method may have a saller probability to have a overflow ggrid when applying the algorithm we used in placement. Therefore,we chose a separate method whem relizing the thoughts.

ACKNOWLEDGMENT

The authors would like to thank Prof. J.H Jiang for consulting about the algorithm's we used in this study.

REFERENCE

[1] Ripple: An Effective Routability-Driven Placer by Iterative Cell Movement Xu He, Tao Huang, Linfu Xiao, Haitong Tian, Guxin Cui and Evangeline F.Y.Young Department of Computer Science and Engineering
[2] NCTU-GR 2.0: Multithreaded Collision-Aware Global Routing with Bounded-Length Maze Routing Wen-Hao Liu, Wei-Chun Kao, Yih-Lang Li, and Kai-Yuan Chao
[3] FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design Chris Chu and Yiu-Chung Wong

AUTHORS

First Author – W.T.Tsen, ntuee, b07901165@ntu.edu.tw
Second Author – S.C.Hung, ntuee, b07901151@ntu.edu.tw
Third Author – Y.H.Ko, ntuee, b07901150@ntu.edu.tw

JOB DIVISION

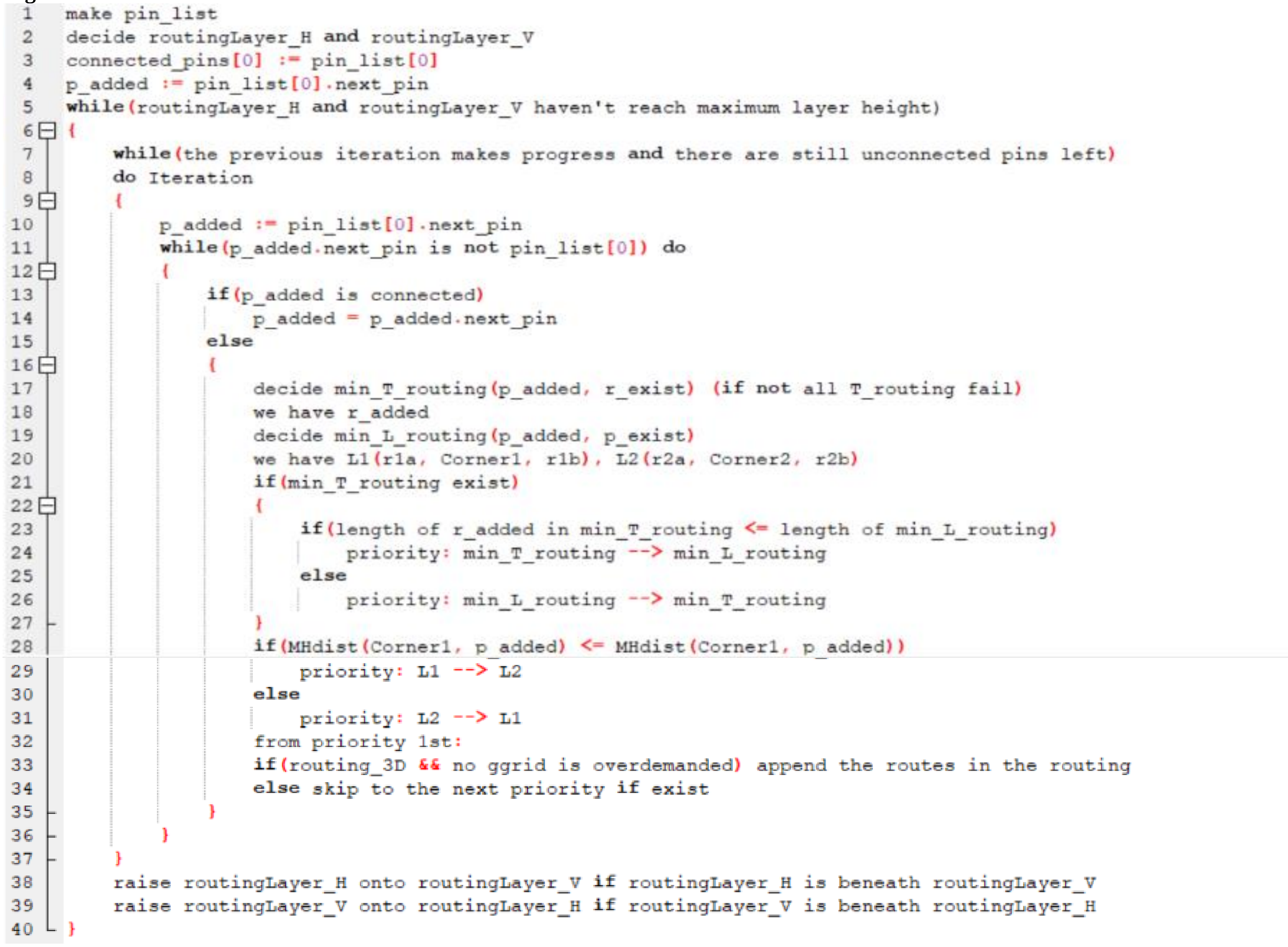
曾煒哲: routing’s code and report and presentation, data structure.

洪世昌: readfile, makefile, data structure, result’s report and presentation.

柯昱豪: placement’s code and report and presentation.

APPENDIX

Algorithm LT-ROUTING
Input: the listed pins of a net
Output: a collection of routes linking all the pins of the net
begin



```
make pin_list
decide routingLayer_H and routingLayer_V
connected_pins[0] := pin_list[0]
p_added := pin_list[0].next_pin
while(routingLayer_H and routingLayer_V haven't reach maximum layer height)
{
    while(the previous iteration makes progress and there are still unconnected pins left)
    do Iteration
    {
        p_added := pin_list[0].next_pin
        while(p_added.next_pin is not pin_list[0]) do
        {
            if(p_added is connected)
                p_added = p_added.next_pin
            else
            {
                decide min_T_routing(p_added, r_exist) (if not all T_routing fail)
                we have r_added
                decide min_L_routing(p_added, p_exist)
                we have L1(r1a, Corner1, r1b), L2(r2a, Corner2, r2b)
                if(min_T_routing exist)
                {
                    if(length of r_added in min_T_routing <= length of min_L_routing)
                        priority: min_T_routing --> min_L_routing
                    else
                        priority: min_L_routing --> min_T_routing
                }
                if(MHdist(Corner1, p_added) <= MHdist(Corner1, p_added))
                    priority: L1 --> L2
                else
                    priority: L2 --> L1
                from priority 1st:
                if(routing_3D && no ggrid is overdemanded) append the routes in the routing
                else skip to the next priority if exist
            }
        }
    }
    raise routingLayer_H onto routingLayer_V if routingLayer_H is beneath routingLayer_V
    raise routingLayer_V onto routingLayer_H if routingLayer_V is beneath routingLayer_H
}
```

end