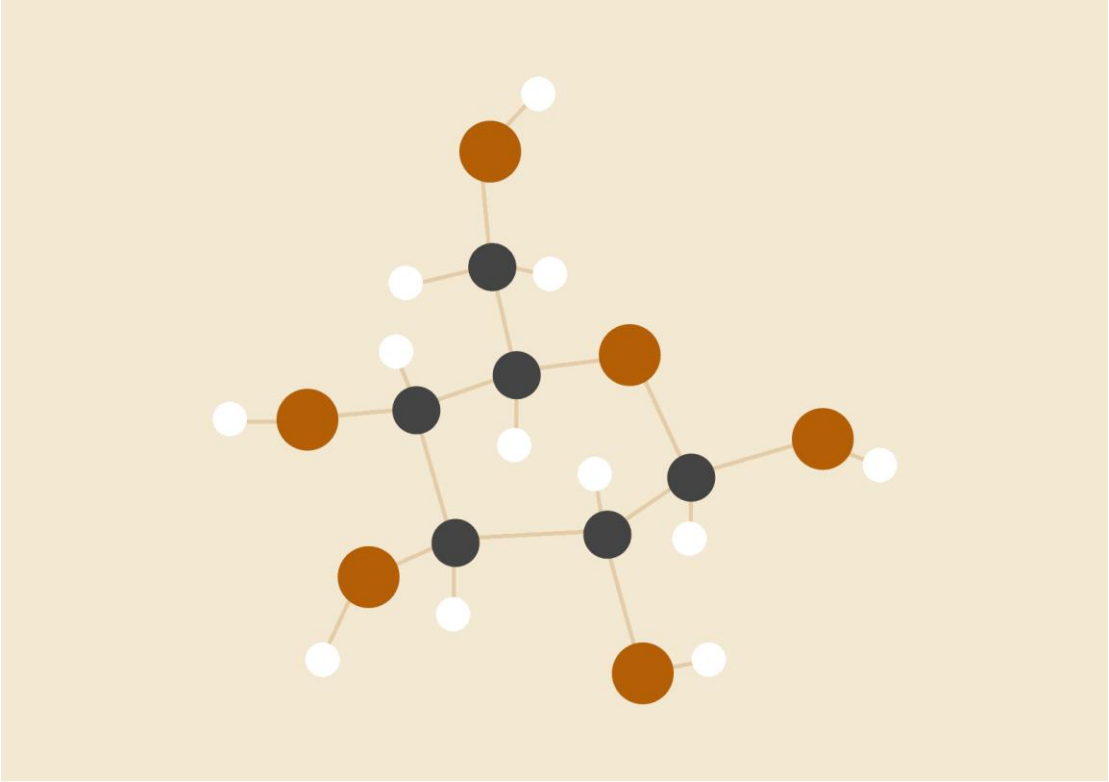
2018 ICCAD-E
Color-aware Routing for Double Patterning



Group 6

機械四 劉光瑋 B03502134 0917041164

機械四 尤聖嘉 B03502161 0912235042

機械四 許晉豪 B03502167 0988729669

目錄

1. 題目簡介

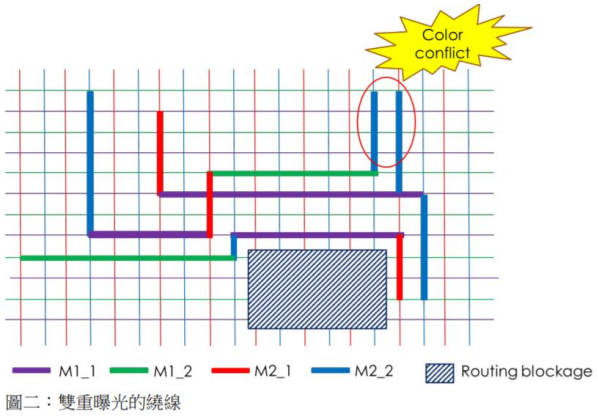
Routing problem is a critical issue in circuit design. It is a practical problem in the real world .In this project, we need to route wires to connect the pins. Given the locations and layers of pins, locations of blockages, and netlist, we need to finish the routing that satisfies all requirements by using horizontal and vertical wires in different layers and color them appropriately.

Requirements

- All pins in a single net need to be connected (No open pins)
- No wire across or contact any blockage
- No color conflict (No short violation)
- Only horizontal and vertical line

Color conflict:

If the same color of wires are only separated by 0.5 grid.



圖二：雙重曝光的繞線

Fig.1. Color conflict

Layer – Four metals:

Metal 1 and 3 are horizontal wires. On the other hand, Metal 2 and 4 are vertical wires

Layer	Layer No.	Pitch	Direction
Metal 1	1	0.5	Horizontal
Metal 2	2	0.5	Vertical
Metal 3	3	0.5	Horizontal
Metal 4	4	0.5	Vertical

Each metal has two colors

2. 前人解法

2.1 Net Connecting

2.1.1 two-pins connecting

Breadth First Search

利用類似水波擴散的方式，從 source vertex 向四面八方探索，直到找到 target vertex 再循著 predecessor 的紀錄回溯找到路徑。

Pseudo Code :

```
for each vertex u ∈ GV-{s}
    u.color = WHITE
    u.d = ∞
    u.pi = NIL
s.color = GRAY
s.d = 0
s.pi = NIL
Q = ∅
ENQUEUE(Q,s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.pi = u
            ENQUEUE(Q,v)
        u.color = BLACK
```

Fig.2. Pseudo Code of BFS

Hadlock's Algorithm

Hadlock's Algorithm 為 BFS 的一種改良。雖然複雜度一樣是 O(V+E)，但極有可能提早結束。不同於 BFS 將所有探索過的 vertex 存進 queue 中以 first in first out 的方式取出，Hadlock's Algorithm 將所有的 vertex 新增一個 detour 的屬性，其意義為偏離 target 的格數。因此在 Hadlock's Alg 中，會將所有探勘過的 vertex 依據 detour 放進 min heap 中，之後便可以 extract min 的方式取出離 target 較近的點。應用在 Routing 時，edge weight 為兩點之間的 Manhattan Distance。

Pseudo Code:

```
for each vertex u ∈ GV-{s}
    u.color = WHITE
    u.d = ∞
    u.pi = NIL
    u.detour = ∞
s.color = GRAY
s.d = 0
s.pi = NIL
s.detour = 0
Q = ∅
INSERT(Q,s)
while Q ≠ ∅
    u = EXTRACT_MIN(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.pi = u
            if v deviate from the target
                v.detour = v.pi.detour + 1
            else
                v.detour = v.pi.detour
            INSERT(Q,v)
        u.color = BLACK
```

Fig.3. Pseudo code of Hadlock's Algorithm

2.1.2 Multi-pins Connecting

Multi-sources Search

先選定一個 vertex 為 source vertex, 連至另一個 vertex 後，回溯路徑，並將路徑上的每個 vertex 設為新的 source，在用每個 source 去搜尋其他的 pin。

Minimum Spanning Tree by Prim's Algorithm

Mininum Spanning Tree (MST) 為眾多將所有 vertex 連接之 Spanning Tree 中，總路徑最短的。

Prim's Algorithm 是利用 Greedy Algorithm 的理念，先選定一 source 後，取出當下可觸及到的 vertex 中, edge weight 最小的連過去。接著再取出此二 vertex 能夠能夠觸及到的 vertex 中 edge weight 最小的連過去。重複此動作直到所有的 vertices 連成一 spanning tree。

Pseudo Code :

```
MST_PRIM(G,w,r)
for each u ∈ G.V-{r}
    u.key = ∞
    u.pi = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT_MIN(Q)
    for each v ∈ Q and w(u,v) < v.key
        v.pi = u
        v.key = w(u,v)
```

Fig.4. Pseudo code of Prim's Algorithm

2.2 Coloring

參考 Welsh-Powell Algorithm 的染色方法。屬於 Greedy Algorithm，每一個點都先嘗試塗第一種顏色，若牴觸了已塗色的點，就換下一種顏色，直到顏色不牴觸為止。

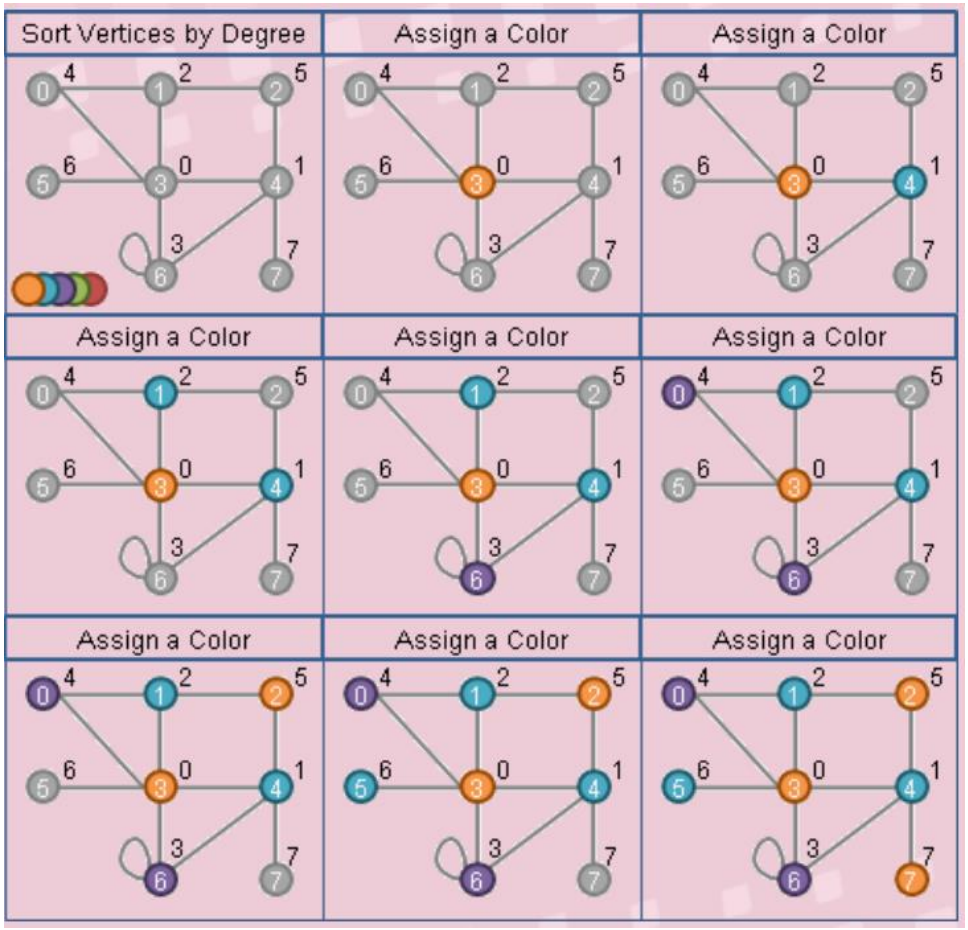


Fig.5. Welsh-Powell Algorithm

每個點都著色的時間複雜度等同一次 Graph Traversal 的時間，如果圖的資料結構為 adjacency matrix 就是 $O(V^2)$ ，如果圖的資料結構為 adjacency lists 就是 O(V+E)。

2.3 想法及總結

存在太多 Graph 相關的 Algorithm 可應用在 Routing，因此前述演算法只取我們有考慮過，或者決定應用的方法。

在 two-pin connecting 中，BFS 是最簡單的地毯式搜索方法，但是因為毫無方向的全範圍搜索，因此效率較差。 我們最後採用 Hadlock's Algorithm 來連接兩兩 pin。

至於如何將一條 net 中的 pin 連接起來，我們上述提到兩種方法。Multi-Source 的方法因為有多少 pin 就要同時進行多少次 Hadlock's Algorithm，因此時間複雜度會非常可觀。基於上述原因我們採用 Prim's Algorithm 來建立一條 net 的 MST。除了此二種方法，其實還有尋找 Steiner points 的方法，但較為複雜，且不好 implement，因此還未考慮使用。再未來若有時間進行改善可能會考慮簡化 Steiner Tree Algorithm。

3. 本組解法

3.1 地圖初始化 (Map Initialization)

在程式的一開始，我們會先根據所給的 pin 範圍創造出差不多兩倍大的地圖。接著將 pin 以及 blockage 的座標點先塗上不同的顏色，以便分辨以及之後接線時的函數辨別。

本次題目還有兩樣特性促使我們必須做更多的處理：1. 地圖是以 0.5 為單位，並非整數點。2. 從 FAQ 知道 pin 有可能出現在負數座標點。根據以上兩點特性，我們會先尋找有無出現在負數座標點的 pin，並將整體平移至第一象限。接著再將整個地圖放大兩倍，消滅非整數座標點，方便之後的函數運作。

這些調整接會在最後 output 前轉換回來，故 output 出來的答案會是正確的。

3.2 類別 (Class)

以下將簡單介紹我們所使用的 class。

- **Class node**

  屬性包括：x,y 座標、層數(Layer)、detour、via(紀錄跳層次數)、pred(紀錄 predecessor)、MSTPred(紀錄 MST 中的 predecessor)、color、order(紀錄探索次序)。

  其中顏色包刮 0~8，其意義如下，

  0 : 為探索　1 : 黑線　2 : 藍線　3 : pin　4 : blockage

  5 : pin 四周阻隔　6 : MST 中記錄是否建立過關係之顏色

  7 : 線路終端四周阻隔

  其中 5,7 兩色是為了避免之後上色容易產生衝突而設立，若無阻隔，兩條線的終端很有可能只相隔 0.5，這在之後的上色容易產生許多矛盾。

- **Class line**

  屬性包括：node1(紀錄線的起點)、node2(紀錄線的終點)、lineColor(紀錄線色)、layer(紀錄線所在層數)

3.3 Two-pin Connecting (Hadlock's Algorithm)

1. 設一個 vector<node> Tree 記錄探索過的所有 node，方便做完一次 Hadlock 可以還原地圖，並記錄 node 之間 predecessor 的關係。
2. 設一個 priority_queue <node, vector<node>, struct cmpHL> pQ 做為 minheap，將探索過的 node 放入，並依照 detour、via、order 的順序比較，將最小的 node 放到 top，做為下一次探索的點。
3. 在一條 net 中會先把 node 塗成藍色變成藍線(color: 2)，假使之後 pin 在連

的過程遇到藍線，就可以提前結束。例如 net 中的 pin1 執行 Hadlock 連接到 pin2 後，此時整條線會是藍色的，而 pin3 接著執行 Hadlock 要連接至 pin2 的過程中，若中途遇到藍色的線便提前結束此次 Hadlock，因為代表已成功接上 net 線路。

4. 檢查要連的兩個 pin 周圍是否有相鄰的 pin，假設是 1、3 層在水平方向有相鄰的 pin 就不要將水平方向的 blockage 5 變成 color: 0，避免發生 color patterning 無解的情形，2、4 層也一樣，假設在垂直方向有相鄰的 pin 就不要將垂直方向的 blockage(color: 5)變成 color: 0。假設都沒有相鄰，就把周圍的 blockage(color: 5)都變成 color: 0。
5. 從 priority queue extract min，並從這個 node 開始探索，假設是在 1、3 層，就依序探索右左下上四個 node，假設是在 2、4 層，就依序探索前後下上四個 node。先檢查是否已經連到終點的 node，再檢查是否連到藍線，也就是同一個 net 的線。假設都沒有，且此點未探索過，依序將 x、y、layer、color 、pred、detour、via、order 等屬性設定完畢。其中 color 先設為 1，detour 則是和 predecessor 比較，假設距離終點的某個方向距離變遠，detour 就等於 predecessor 的 detour 加一，沒變就相等，via 則是看有沒有跟 predecessor 同層，沒有就加一。
6. 結束探索有兩種情形，Priority queue 空掉，也就是堵住，或是找到藍線或終點。結束後會先將 Tree 中所有的 node 顏色都變回零。
7. 假設沒找到可行的路線，回傳一個空的 vector<line> ans。找到的話從 Tree 的最後一個點依循著 pred 找回起點，將路線上每個點上色，MST 為真的線上藍色(color:2)，不然就上黑色(color:1)，並同時記錄轉彎點，最後儲存到 vector<line> ans 中回傳。

3.4 Net Connecting (Prim's Algorithm)

1. 將 Net 中的 pins 皆塗成綠色，並先取出其中一 pin 當作 source。
2. 初始化除了 source 以外的所有 pin，將 predecessor 皆設成 source 後以與其 predecessor 的 Manhattan Distance 為 edge weight 放入 min heap 中。
3. Extract min，判斷取出的 node 是否為綠色，是的話代表此點還未被取出過。接著將除了 source 以及此點以外的所有 pins node 複製，並將 predecessor 皆設為此點後，全部 insert 進入 min heap。如此一來同個 pin 點可能會重複出現在 heap 中，但 predecessor 不同。何者會被取出取決於與 predecessor 的 Manhattan Distance。
4. 重複執行上述動作直到 Net 中所有 pin 皆與彼此建立好關係。

3.5 Coloring

因為我們染色要直到全部線接完到才處理，且每層的線又和彼此不相關。所以會先進行分類。將線依照層儲存在矩陣裡。

classifyline(vector<line> &outline)，會把讀進的 vector<line>分類、儲存進 routing class 的 public member:*m[4]裡。

蒐集完全部的線以後，才會呼叫 ColorPat()這個 function，開始染色。染色方法為 Greedy Algorithm ，如果沒有 color conflict 的話，則優先選擇染比較少使用的顏色(color balancing)

而 checkline(int layer,int posi)設為 private，是因為這只有在 ColorPat()這個 function 裡會需要，故無需外露。這個 function 主要是協助 ColorPat 判斷是否會產生 color conflict。

3.6 Optimization

1. Net Ordering

Net Ordering 是本組設定的其中一項經驗法則，我們會先做佔總矩形面積比較小的 net。總矩形面積的定義方式為，能夠包住 net 中所有 pin 的最小矩形。如下圖，三個 pin 的 net 之矩形面積為紅色框所包住的面積。這個優化的想法是，如果先做矩形面積很大的 net，許多原本短短幾條線便可連接的小 net 很可能被迫需要繞遠路。而先接小面積 net 的化，因為佔面積小，比較不會對其他 net 造成太大的影響。

經過測試後，我們推論在較密集的測資中，此方法會有比較顯卓的效果。在官方的 case1 測資中，可能因為 net 較少，因此加了 net ordering 後總線長幾乎沒改變。而在 case2 測資中，總線長減少了三百多。

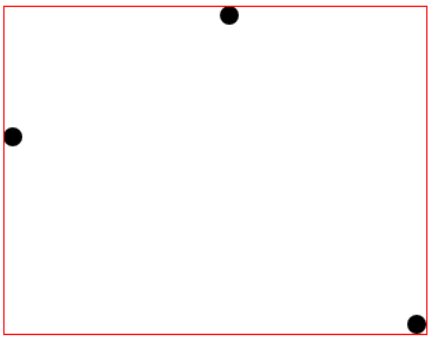


Fig.6. Net Area

2. Early Termination

在同一條 net 中，因為 pin 很可能不只 2 個的原因，通常會需要進行許多次 Hadlock's Algorithm。在有提前終止這個條件下，當一條 net 中的 hadlock 接線途中撞到同一條 net 的其他線，會直接連上並提前終止。這個設定能夠減少許多不必要的時間與線長，若是沒有這個設定，線路便會沿著撞到的線一路繞直到接到預設連回的 pin。

能夠使用這個設定是因為我們的接線一律是從孤立、還未被加進 MST 的 pin 連回已經被 MST 拓展到的 pin，因此只要撞到 net 中的線便可直接連上去，並不一定要連到 pin。

4. 複雜度分析

Map 總 node 數 : S
Pins 數量 : P
單層 Map 寬 : W
Nets 數量 : N

1. Initializing the map

- pins 上色 : $O(P)$
- blocks 上色 : $O(S)$

2. Net Connecting

Minimum Spanning Tree by Prim's Algorithm

3. Initialization : $O(S)$
4. Extract Min : $O(P^2 \log P)$
5. Hadlock's Algorithm of Every Edge : $O(P) \times O(S \log S) = O(PS \log S)$

3. Coloring

$O(W + L)$

4. 總時間複雜度

$O(NP^2 \log P + NPS \log S)$

5. 遭遇困難

5.1 Multi-Pin Connecting

目前的方法是使用 MST 來連接一條 Net 中的每個 pin，但 MST 所產生出來的接法不一定能讓我們滿意。很多時候從兩個 pin 連接路徑中段的某個點接出去連接另一 pin 會短許多，但 MST 無法做到這件事情。MST 產生出來的接線會比最好的接法長許多，我們擔心這可能會在未來導致堵線的問題。因此在未來工作中，我們會研究如何找出簡化版的 Steiner Points 來讓我們的接線更為精簡。

5.2 Coloring

染色問題可能會無解，需要從排線開始來避開無解情況。所以在讀檔和在做 Hadlock 時，有進行額外的擋住、隔開的動作。

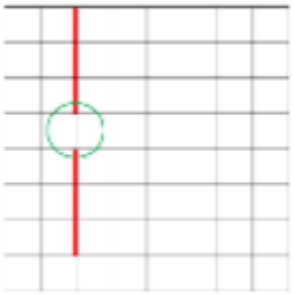


Fig.7. line end

為了避免(如上圖)的狀況發生，所以我們會在每接完一條新的線後(如下圖)，在接點以及轉角做封鎖。

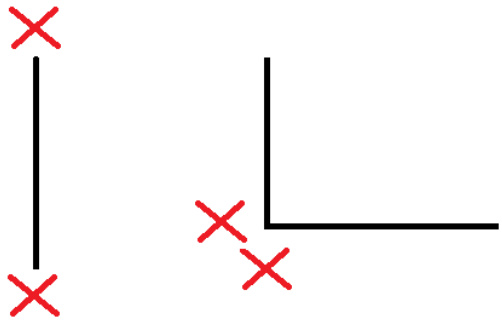


Fig.8. Blocking of line end

如此一來就可避免掉無解的狀況了。

6. 結果與探討

	Total length	Critical Net length	Vias	Color Usage Layer 1	Color Usage Layer 2	Color Usage Layer 3	Color Usage Layer 4
Case 1	5306	1434	160	694 653.5	1382.5 1314	513 556	81 112
Case 2	9361	2527	351	1402.5 1400.5	945 918	1507.5 1789.5	693.5 704.5

目前我們 Color Balancing 的演算法效果沒有到很理想，因此我們打算在比賽前，利用暑假的時間將原本使用的方法從 Greedy Algorithm 改成 Dynamic Programming。首先，先將地圖的線都分成獨立的 set 和其的 enemy set。如果有其他的 set 會和本身的 set 產生 color conflict 的話，就將其加到 enemy set 裡。反之要把 enemy set 的 enemy set 加到本身的 set 裡。做好初始化後，假設有 n 組成雙成對的 friend set 和 enemy set，所以總共會有 2 的 n 次種組合可以上色，接著使用 DP 找到最佳解。目前想法是先 sort 所有 friend set 和 enemy set，後續還在研究中。

除了 Color Balancing 的問題，我們在接線的部分還會再做更進一步的加強。目前有寫了一個版本的簡易 Steiner point 優化法。方法中用 net 中的 pin 找出其 hanan grid 來當作潛在的 steiner points。Hanan grid 為以每個 pin 為中心畫水平及垂直線，相交出來的許多點 (候選點)。我們將每個候選點算出其 MST Cost，與原本的 Cost 相減得到 Cost 差，將每個候選點依照 Cost 差由大排到小，一一加入 MST 中。在每個點加入前會確定 Cost 不會是增加的。

但寫好後線長卻沒有改善，認為應是有 bug 還未被發現，但因許多個人原因，大家並沒有時間在 Demo 前將這個方法完成，因此也會留待七月完成。

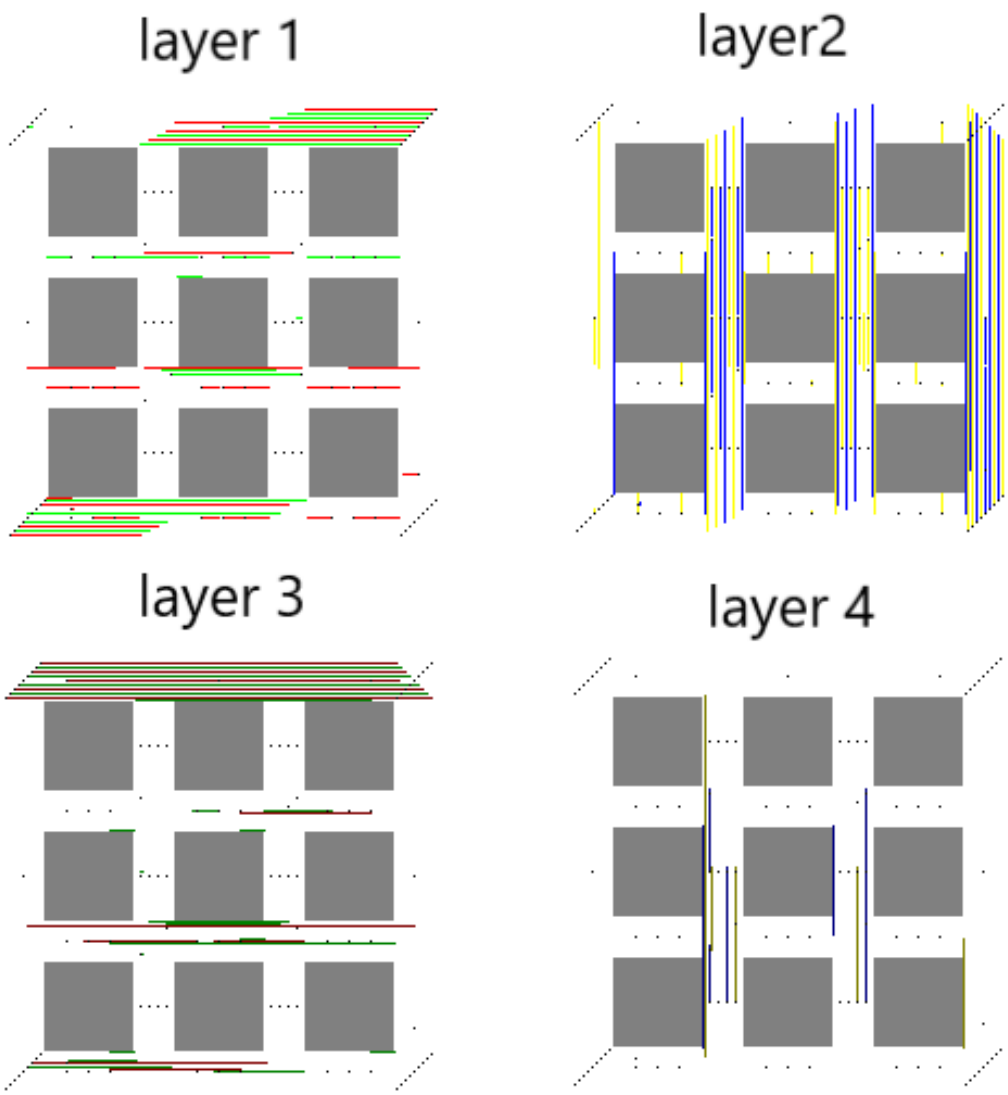


Fig.9. Case 1 layout

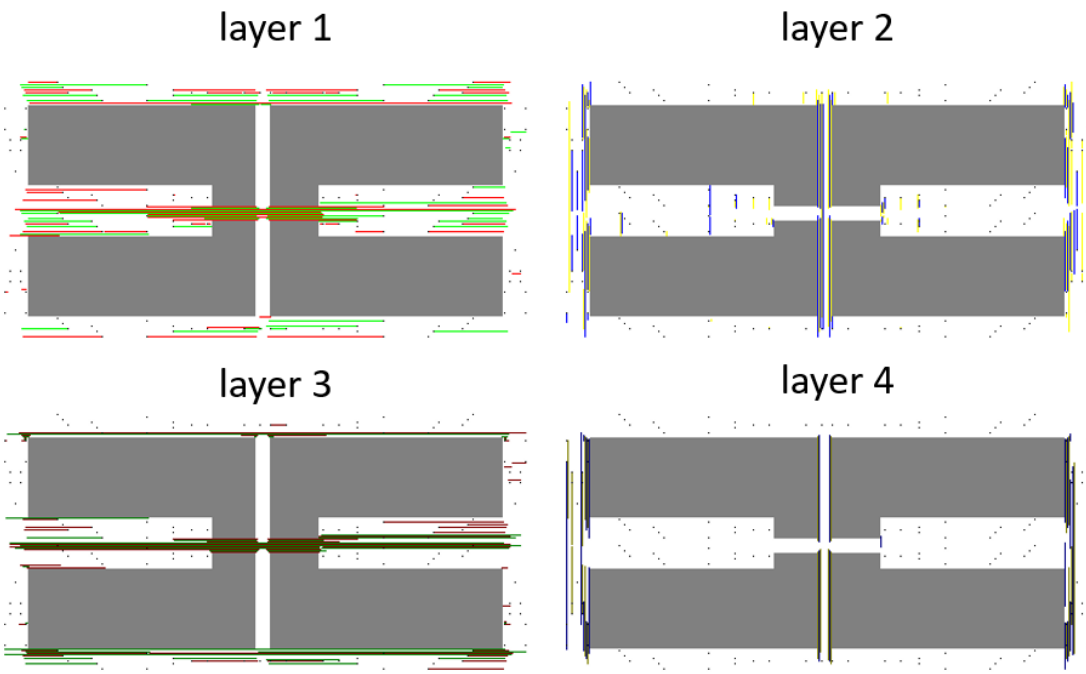


Fig.10. Case 2 layout

7. 參考資料

1.http://users.eecs.northwestern.edu/~haizhou/357/lec6.pdf
2.http://www.csie.ntnu.edu.tw/~u91029/Coloring.html

8. 具體貢獻

Main.cpp： 許晉豪(主), 劉光瑋
標頭檔架構建立：劉光瑋、尤聖嘉、許晉豪
Vector<line> Hadlock(node n1, node n2, bool MST)： 尤聖嘉
Vector<line> MST(vector<vector<int>> netCoor)： 劉光瑋

Void classifyline(vector<line> &outline) : 許晉豪
Void ColorPat() : 許晉豪
Int Mdis(node a, node b) : 劉光瑋
Void prt_Map(int map_layer, int origin_x, int origin_y) : 尤聖嘉
Void createblock(vector<node> b) : 許晉豪
Int checkline(int layer, int posi) : 許晉豪
Hadlock, MST 中所需 heap 資料結構 : 劉光瑋、尤聖嘉
Connecting Checker 劉光瑋
Color Checker 許晉豪

具體程式碼行數貢獻(不含標頭檔)：
劉光瑋：320 行
尤聖嘉：318 行
許晉豪：445 行