Final Project Report

CAD 2018 Problem E

Group 20

蘇宸右物理四 B03202065 0937675298
陳映寰物理四 B03502067 0952729353
黃宣凱物理四 B03202007 0975603235

June 30, 2018

1 Problem description

Color-aware Routing for Double Patterning[1]. This problem is to find the optimal solution to pin connections in double patterning techniques. People have to find out the way to connect either two-pin wire or multi-pin wire without short/open violation and color conflict. The metals that the wires are set have their preferred direction, and the color distribution should be as uniform as possible. The input files consist of three smaller files: pin file, net file and block file. The net file denoted the combination of pins for each net, either for two pins or for more pins.

2 Previous Study

2.1 Introduction to two-pin connection

Hadlock's algorithm[2]
It uses a new method for cell labeling called detour numbers (instead of labeling wave-front in Lee's router), and it guarantees to find the shortest path between two pins.

1. d(P): the detour number of a path P connecting S and T is defined as the number of grid cells directed away from its target T.
2. MD(S,T): the Manhattan distance between S and T.
3. l(P): The path length of P is given by l(P) = MD (S,T) + 2d(P)

In conclusion, because MD(S,T) is fixed, cells with smaller detour numbers are expanded with high priority. We can minimize d(P) to find the shortest path. Time and space complexities is the same as Lee's (O(MN)), but it substantially reduces the number of searched cells.(See Figure 1)

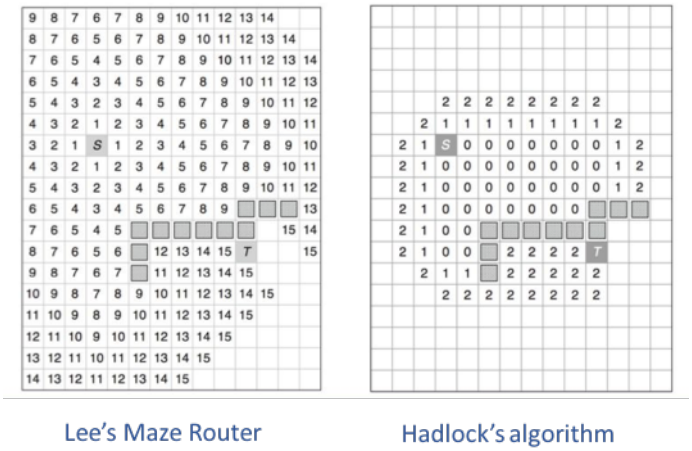


Figure 1: The wave-front comparison of Lee's Maze Router and Hadlock's algorithm.

A* algorithm

A* is a special algorithm with BFS as basic and using the heuristic function as the reference of the order of searching.

1. F(n) = G(n) + H(n)
2. G: actual distance from source to point n
3. H: heuristic estimation of the distance between n to the goal

Starting from the source, F-value of the neighborhood grids would be calculated, the grids with least F would be first visited. The similar process would continuously repeat till the goal is reached.

By clever set of H-function, it's possible to get a result with short runtime, short wire length, small number of vias, low detour ratio of critical nets at the same time (as long as one find a marvelous way to calculate H) .

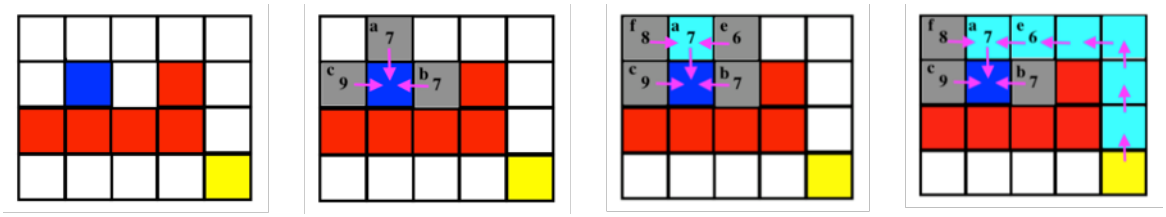


Figure 2: Steps of A* algorithm (from left to right)

2.2 Introduction to multi-pin connection

Minimum Spanning Tree method

Simply treat all the pins in a net as several pairs of pins, and connect each pair with two pin algorithm. The order of the connection is decided by MST, which uses the Manhattan distance as edge score and run any MST algorithms. Both Prim's or Kruskal's algorithms can be applied to compute MST with high efficiency.

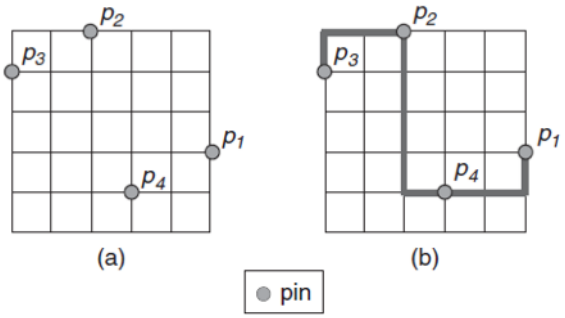


Figure 3: (a) is a set of pins, and (b) is the MST result of this set.[2]

This approach is fast, however, may not be the best as several pins may share the same position properties but we neglect it. For instance, the connections from P_4 to P_2 and P_3 can share

most of the path. Thus, the way of connecting them should not be independent but with some additional logic to improve the performance of multi-pins connection. [2]

Steiner method

Steiner tree (or minimum rectilinear Steiner tree, MRST) is an improved version of MST that minimizes the wire-length of multi-pin nets by connecting the pins through some extra points dubbed Steiner points. It can be referred to as employing wires with branches to some extent. To be specified, if P and S denotes the set of pins and Steiner points respectively. Then, MRST(P) = MST(PS).

M. Hanan has proved that there exist an MRST with all Steiner points chosen from the grid (named Hanan grid) constructed by drawing vertical and horizontal lines of the required pins [Hanan 1966] (Figure 4). In spite of the sharply reduction of searching space by Hanan's theorem, the MRST construction is still corroborated to be an NP-hard problem [Garey 1977]. Therefore, there have been many heuristics developed. Our heuristics ways to solve this problem will be discussed in the following parts. [2]

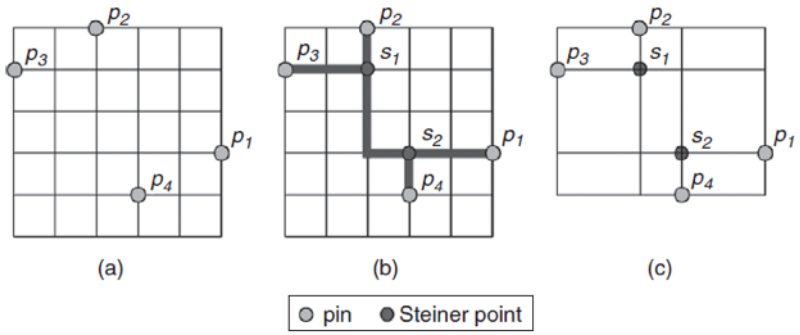


Figure 4: An example of Steiner tree.[2]

3 Our approach

We divided the problem into several steps, each part will be discussed separately in below.

3.1 Preprocessing

We first handle all the input files and separately store the information in three vectors. For pin files, which consist of position and layer number, we double the position because of the existence of ".5". In order to deal the problem with integer, this operation is required.

Next, we denote the block on our map. The map is made up of C++ std unordered_map, which provided constant time search and required less memory than three dimensional vector. In order to hash the structure (x,y,layer), we use a simple transformation $f(x, y, l) = x * xparameter + y * yparameter + layer$, which help us transform all the information about a point into a conflict-resisted integer, and the std hash is promising when handling simple integer.

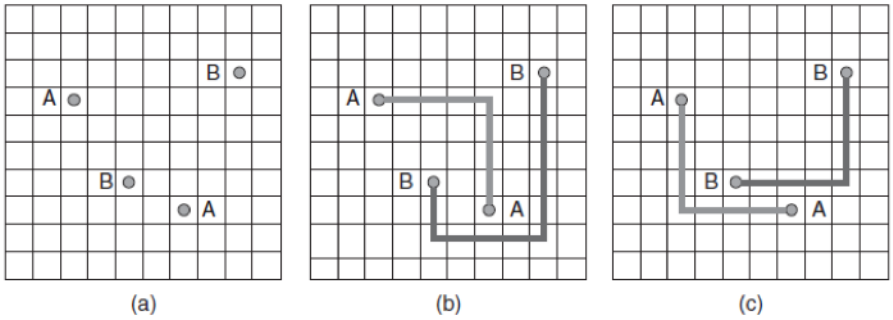


Figure 5: The influence of the order of the net, where case (c) is better than case (b). [2]

3.2 Net ordering

The nets have two properties: multi-pins or two-pins, critical or not critical. Thus, we reorder the nets by a heuristic score,

$$score = \frac{total\ manhattan\ distance}{2}\ if\ critical$$
$$= (total\ manhattan\ distance)\ if\ not\ critical$$

The lower the score is, the higher priority the net has. The definition of total Manhattan distance:

$$total\ MD = \frac{\sum_{i=0}^{n-1}\sum_{j=0}^{n} MD(p_{i-1}, p_i)}{n^2}$$

We can see that no matter whether the net is multi-pin, the net whose pins can span larger area always has a higher score, which is quite logical.

	No ordering	With order
Case_1	5185	5172.5
Case_2	10033	9322

As we can see, the performance of both cases get improvement. We take both total length and critical net into account, and finishing the ordering with quite good performance.

3.3 Two-pin connection

The algorithm applied in the two-pin connection is the A* search. The grid with the lowest heuristic score would be in the highest priority to be visited next by the use of priority queue. The heuristic function is the Manhattan distance of 2 point's (x, y) coordinates plus the change of layer multiplied by a tunable parameter α (= 1 in our recent work for now.):

$$H = MD(p_1, p_2) + \alpha\Delta L$$

From the table below we can see that in the frame of this problem, our heuristic can dramatically improve the performance of searching. As the layer itself defined the allowed direction, the parallel searching is simple, and didn't change when not using the heuristic. However, when

perpendicular case is taken into account, the heuristic function can give a brief direction to the terminal, making searching efficiently.

Distance	Perpendicular, with heuristic	Perpendicular, without heuristic	Parallel, with heuristic	Parallel, without heuristic
1	7	58	2	2
2	11	160	4	4
3	15	337	6	6
5	23	873	10	10
30	123	28786	60	60

The result of the two pin connection is in Figure 6. We can see as changing the direction increasing the heuristic score, our A* can route the net with minimum vias, which is the indicator of the number of changing direction.

Figure 6: The effect of heuristic.

Admissible and Consistent The target is to decrease searching time without losing too much performance, thus the heuristic function should be limited under some criteria. The optimal solution can be achieved if we assure admissible and consistent. However the proof of consistent is quite complicated, thus we secure the admissibility of our heuristic first.
Admissible, meaning that we never overestimates the actual cost. The most easy way to secure this property is to lower the original heuristic score, and 0 is a trivial solution as the A* become uniform cost search. We divide our heuristic score to half of origin, and the performance improve significantly. Still, we didn't mathematically prove that our heuristic is admissible, but the performance did say something.

3.4 Multi-pin connection

1. Choose Steiner points
   We came up with a new way to choose Steiner points: we get the average value of every pair of pins. For example, if there are two pins, pin1 (x_1, y_1) and pin2 (x_2, y_2), we will add a Steiner point at $(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2})$. Eventually, we will have the union set of Steiner points chosen from Hanan grid and our method.

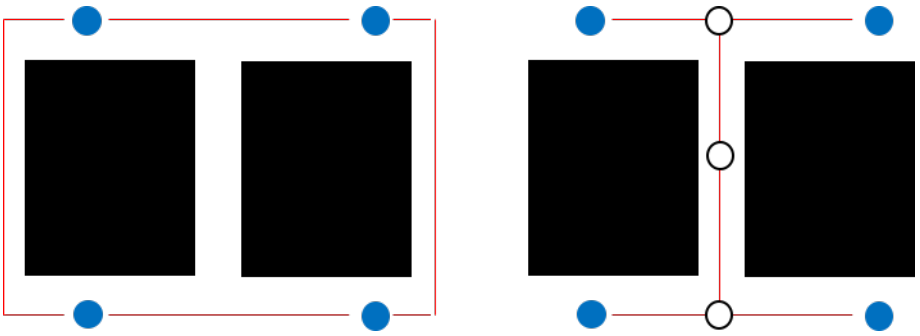


Figure 7: Blue dots are pins and white dots are Steiner points. The left picture: Using the method of Hanan gird to choose Steiner points. The right picture: Using average method to choose Steiner points. We can see that once the blocks exist, the Steiner points in the left picture may be useless.

2. Heuristic modification of Manhattan distance
   When we are calculating the edge weight between two pins on the graph, we usually use the Manhattan distance to represent the edge weight. However, there is a good chance that there are blocks between two pins, so the actual routing distance may be longer than the Manhattan distance. Thus, we came up with a new way to calculate edge weight based on the blocks between two pins. If the two pins have same x value or same y value, then the heuristic distance H is $H(x_1, x_2, y_1, y_2) = MD + 2 \times w$. If the two pins have different x and y values, then the heuristic distance H is

$$H(x_1, x_2, y_1, y_2) = \frac{1}{2}(H(x_1, x_1, y_1, y_2) + H(x_2, x_2, y_1, y_2) + H(x_1, x_2, y_1, y_1) + H(x_1, x_2, y_2, y_2))$$

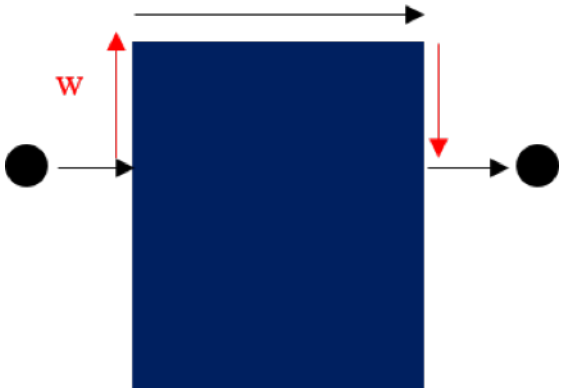


Figure 8: A schematic diagram of heuristic modification of Manhattan distance. The total length of black lines equals to Manhattan distance, and the total length of red lines is the heuristic parameters we added in the heuristic distance.

3. Steiner tree algorithm
   One of the earliest and most effective Steiner tree approximation is the Iterated 1-Steiner (I1S) approach of Kahng and Robins. The I1S heuristic is simple, easy to implement, and significantly outperforms previous approaches.

This algorithm starts with an MST path, then, iteratively select the Steiner point that lessen the wire-length most until it can't be further decreased. In the end, remove the degenerate Steiner points if necessary. See Figure 9. [4] [5]

Iterated 1-Steiner (I1S) Heuristic [36, 55, 57]

Input: set P of n points
Output: rectilinear Steiner tree spanning P

$S = \emptyset$
While Candidate_Set $= \{x \in H(P \cup S) | \Delta MST(P \cup S, \{x\}) > 0\} \neq \emptyset$ **Do**
  Find $x \in$ Candidate_Set which maximizes $\Delta MST(P \cup S, \{x\})$
  $S = S \cup \{x\}$
  Remove points in S which have degree ≤ 2 in $MST(P \cup S)$
Output MST$(P \cup S)$

Figure 9: The Iterated 1-Steiner (I1S) algorithm. [5]

Although a single 1-Steiner point may be found in $O(n^2)$ time, the required computational geometry techniques are complicated and not easy to implement. To address these issues, a batched variant of I1S was developed. [6][7] First we have to define a parameter called $\Delta MST(A, s)$. Given a point set A and a Steiner point s, we define the MST savings of s with respect to A as:

$$\Delta MST(A, s) = cost(MST(A)) - cost(MST(A \cup s))$$

Given a point set P and a set of Steiner points S, each round of B1S greedily adds into S a maximal set of independent 1-Steiner points. Termination occurs when a round fails to add any new Steiner points. See Figure10.

Batched 1-Steiner (B1S) Heuristic [55, 57]

Input: set P of n points
Output: rectilinear Steiner tree spanning P

While $T = \{x \in H(P) | \Delta MST(P, \{x\}) > 0\} \neq \emptyset$ **Do**
  $S = \emptyset$
  For $x \in \{T$ in order of non-increasing $\Delta MST\}$ **Do**
    If $\Delta MST(P \cup S, \{x\}) \geq \Delta MST(P, \{x\})$ **Then** $S = S \cup \{x\}$
  $P = P \cup S$
  Remove from P Steiner points with degree ≤ 2 in MST(P)
Output MST(P)

Figure 10: The Batched 1-Steiner (B1S) algorithm. [7]

Our method is slightly different from the Batched 1-Steiner (B1S) algorithm. We first sort the Steiner points set in order of non-decreasing $\Delta MST(A, s)$. And then we add all the Steiner points to terminal points and run MST. After that, we greedily remove the Steiner point, starting from the point with least $\Delta MST(A, s)$ value. We found out that the performance of our algorithm is better than I1S and B1S for test case 1.

3.5 Output transforming

We record all the point that a net may go through, thus to transform the record to the combination of pairs (start, end), we go through every point in the record vector and transformed them to data structure wire (start, end, layer, color). We also assign the color of net here with

a very naive idea: Even and Odd. That is, if the wire is in the even column, it will be painted color 1. On the contrary, if it is in the odd column, it will be painted color 2. We'll deal with the color balancing in the next part.

3.6 Color balancing

From the result from output transforming, We will modify the distribution of colors in each layer. We only change the color of safe wires, which means that they have no neighbors and can be assigned both colors.
First, we sort the wires in the same layer by their length. Second, we assign the color of wire in sorted order. The longer wire will receive higher painting priority. When assigning the color, we prefer colors that are currently less used.

Case 1: The length of color 1 / the length of color 2			
	Initial	Modify without sorting wires	Modify with sorting wires
Layer 1	1708/2014	1833/1889	1875/1847
Layer 2	1365/3546	2459/2452	2455/2456
Layer 3	475/898	599/774	683/690
Layer 4	52/287	177/162	168/171
Case 2: The length of color 1 / the length of color 2			
	Initial	Modify without sorting wires	Modify with sorting wires
Layer 1	3232/4306	3772/3766	3769/3769
Layer 2	2113/2070	2089/2094	2092/2091
Layer 3	2303/2317	2450/2170	2311/2309
Layer 4	1174/1129	1174/1129	1174/1129

Table 1: The wire length for each color from each layer. The length is two times larger than the real result.

It's obvious that after sorting the balancing is better. The wire length showed in the table is twice as long as the actual wire length, because we double the grid size during preprocessing.

4 Complexity Analysis

Again, we first discuss the complexity separately, and the final complexity will be concluded at the end of this part. Some notations are denoted as below:

$$D : the\ size\ of\ the\ map$$
$$P : the\ number\ of\ total\ pins$$
$$B : the\ number\ of\ the\ blocks$$
$$N : the\ number\ of\ the\ nets$$

4.1 Preprocessing

1. All the pins:
   $\Theta(P + N + B)$
2. Block recording:
   $O(D^2)$ as the maximum point we have to record is all the map.

Overall complexity: $O(P + N + D^2)$

4.2 Net ordering

1. Calculate the score for all the nets:
   $O(P^2)$
2. Sorted the nets:
   $\Theta(N \log N)$

Overall complexity: $O(N \log N + P)$

4.3 Two pin connection

1. Considering the worst case of BFS search, complexity is the same as maze router:
   $O(D^2)$

Overall complexity: $O(D^2)$

4.4 Multi pin connection

1. Initialization: $O(V)$, where $V = P + S$. P is the number of pins and at most all the pins are in a net and S is the number of Steiner points. $S = C_2^p = O(P^2)$.
   $O(P^2)$
2. Pop out a pin, we will push all the other pins in the queue which haven't been taken out. The complexity of extract min is
   $O(V^2 \log V)$
3. Then we will do the two-pin connection and the complexity is the complexity is
   $O(D^2)$
4. We examine the test case, there are at most 10 pins and at most 145 Steiner points. Thus, we have to do MST 145×2+1 times.
   $O(P^2(P^2D^2))$

Overall complexity: $O(P^2(P^2 \log P^2 + P^2D^2))$

4.5 Output transform

1. For each wire:
   $O(D^2)$

Overall complexity: $O(ND^2)$

4.6 Color balancing

1. Because we have to modify every wire, if the number of wires is PN multiplied by a constant, the complexity is
   $O(PN)$

2. The complexity of sorting the wires is
   $O(PN \log PN)$

Overall complexity: $O(PN \log PN + PN)$

4.7 Overall complexity

As the multi-pin handling being the dominant term, the complexity is:
Overall complexity: $O(P^2(P^2 \log P^2 + P^2 D^2))$

5 Difficulties and solution

5.1 Net ordering—The actual total Manhattan distance

The larger the area that spanned by the pins in a single net, the higher the score it should get. However, the combination of pairs (p_i, p_{i-1}) may not be the most appropriate arrangement. For the combination of {(1,2),(3,6),(3,2),(1,6)} should be able to be regarded as a tow-pin-similar case, however this order will increase the score. Many methods can be used to solve this problem, but considering the cost and the performance, we ignore this problem and still get a good result.

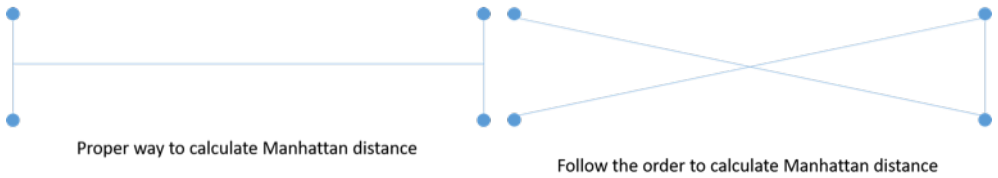


Figure 11: The difference of calculating the Manhattan distance of a net with original order of pins.

Solution
We use the pairs (p_i, p_j) instead, and sum over all $i\&j$. Thus we generalize the case and decrease the probability of encountering worst cases.

5.2 Two-pin connection—Over simplify the heuristic

The heuristic function can be as complicated as possible in order to optimize the performance of routing. However, to avoid over-fitting in this problem, we are still considering whether we should add more heuristic characteristics in our score function. The relationship between a point and the closest block, the weight of changing layers, and many other properties are still waiting for us to figure out.

Solution

After considering admissibility and consistency, we successfully tune our heuristic to not only being able to reduce searching time but also being able to approach optimal solution.

5.3 Multi-pin connection—High complexity

The complexity of multi-pin connection is extremely high. Although the running time of case 1 and case 2 is around 10∼20 seconds, there's a good chance that the running time of case 3 may exceed the limit (2 hours).

Solution

The Steiner tree can reduce the wire length while increasing running time dramatically, thus we set a stop point determined by the number of pins in a net. If the number is bigger than some constant, like 20, then we'll use simply MST to handle the case, and if the number is smaller than the constant, then Steiner is the choice. In this manner, we can take both running time and performance into account.

Another way to solve this problem is to keep track of the running time. If the running time exceeds 1 hour and 40 minutes, then we will stop to use MST, which is of much lower complexity, for multi-pin connection.

6 Result

6.1 Performance

For two given cases, the success rate is 100%.

	Total length	Critical Net Length	Vias	Color Usage Layer 1	Color Usage Layer 2	Color Usage Layer 3	Color Usage Layer 4	Running Time* (Sec)
Case 1	5172.5	1301.5	172	937.5 923.5	1227.5 1228	341.5 345	84 85.5	16.5
Case 2	9322	2515	292	1884.5 1884.5	1046 1045.5	1155.5 1154.5	587 564.5	12.5

*: Running time is estimate by time ./... ,which is a built-in function of linux bash. The number is the average of three times running.

The final score is combined by the following parts: Overall wire length (35%), detour ratio of critical nets (35%), number of vias (20%), mask usage balancing (10%).

6.2 Proof of correctness of out number

We made a program to calculate the wire length, color usage, and number of vias of our output answer for each case. The structure of code of our checker is briefly shown as below:

1. Total wire length and length of critical net : The input files of our code are the net file and our answer file. The total wire length is calculated by iteratively sum all segments (difference of 2 points) of each net in the answer file while the total critical net length can be sum independently with another variable since we have the information of critical net number from the net file.

2. Color usage : Assume each color**'**s usage is defined as its total wire length, we build a 4x2 vector to record 4 layers**'** usage of **their** 2 colors by iteratively sum the segments while reading input file.

3. Number of via : Simply speaking, our approach is like **drawing** the line at the same time checking vias. We construct two unordered map, check map and via map. The first one is a map of int to bool, for a single net, we let the points on each segment map to 1. In the mean time, we check whether these points**'** upper and lower layers points**'** values map to 1, if any one of they did, the via is calculated by the largest layer minus the lowest layer and recorded in the via map (int to int). We iteratively sum the vias during the process we built the via map (the value of some specified point in the via map may be update by lager value, we just sum the **extra** value to total vias at that situation) till the net is finished. Then we clear the 2 maps and repeat the steps , sum the via of other nets, finally get total number of via.

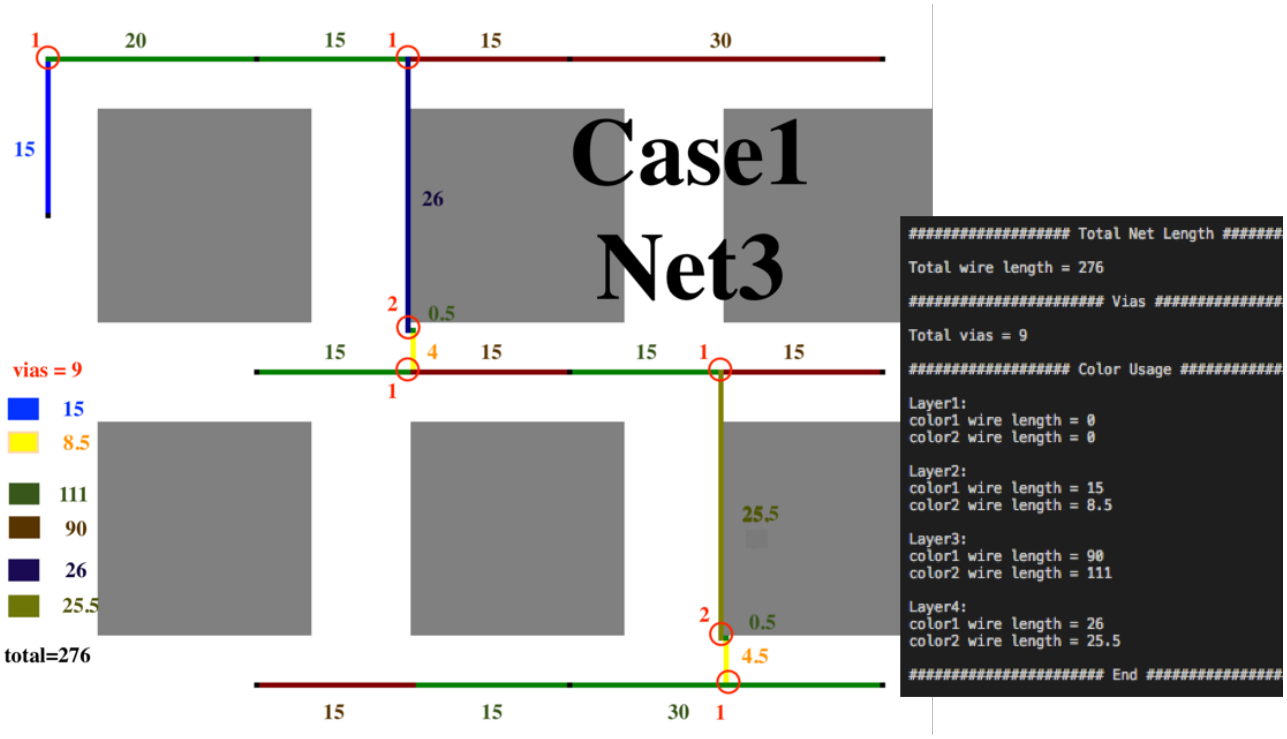


Figure 12: The Layout of Case1 Net3 and the checker output.

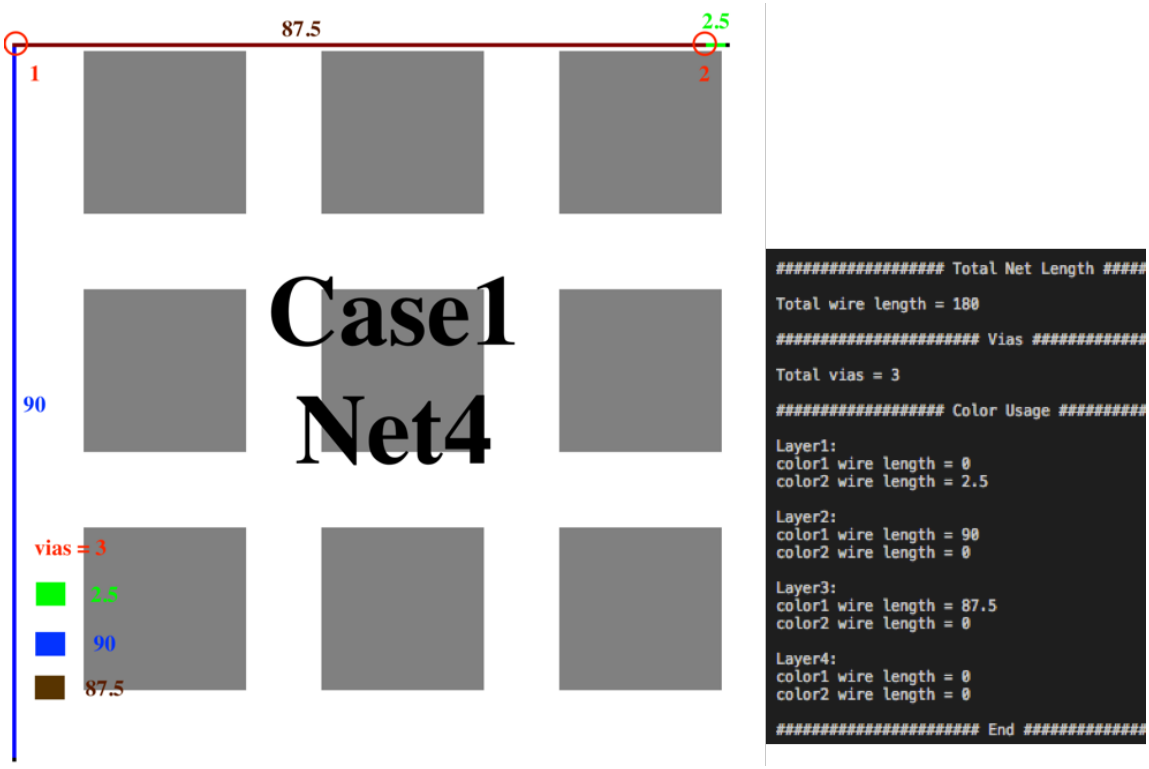
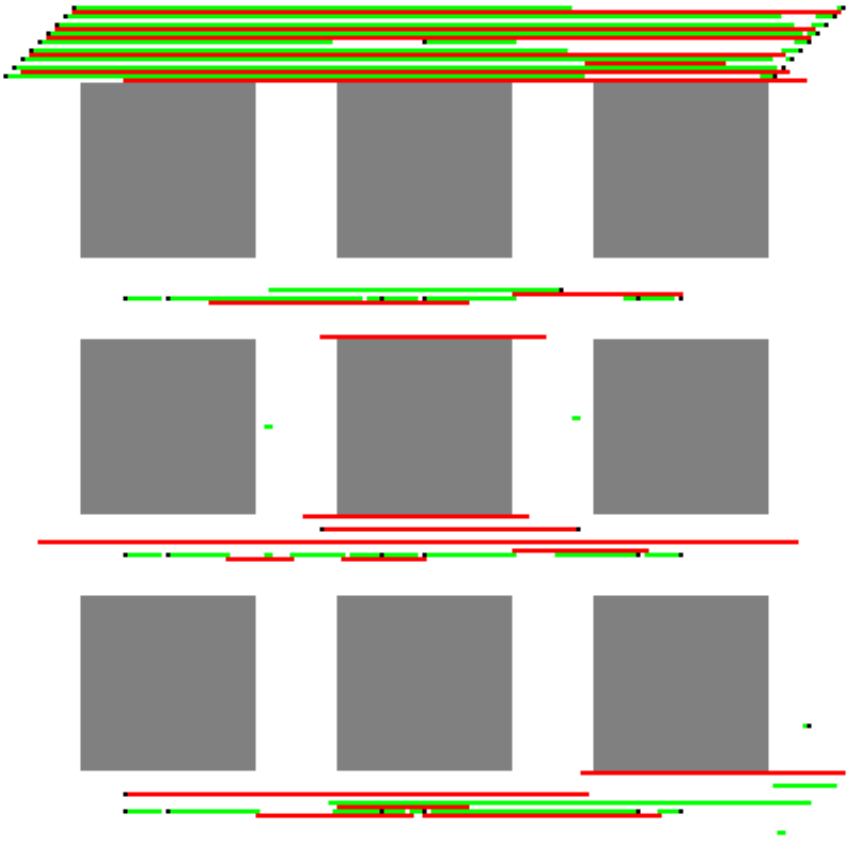


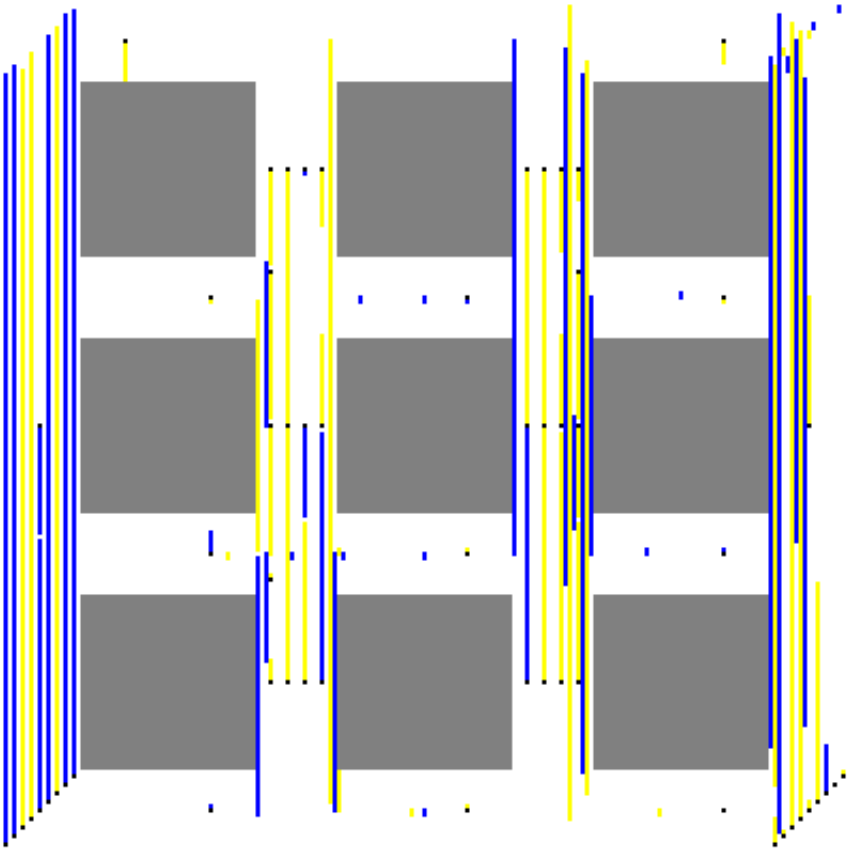
Figure 13: The Layout of Case1 Net4 and the checker output.

6.3 Visualize Result

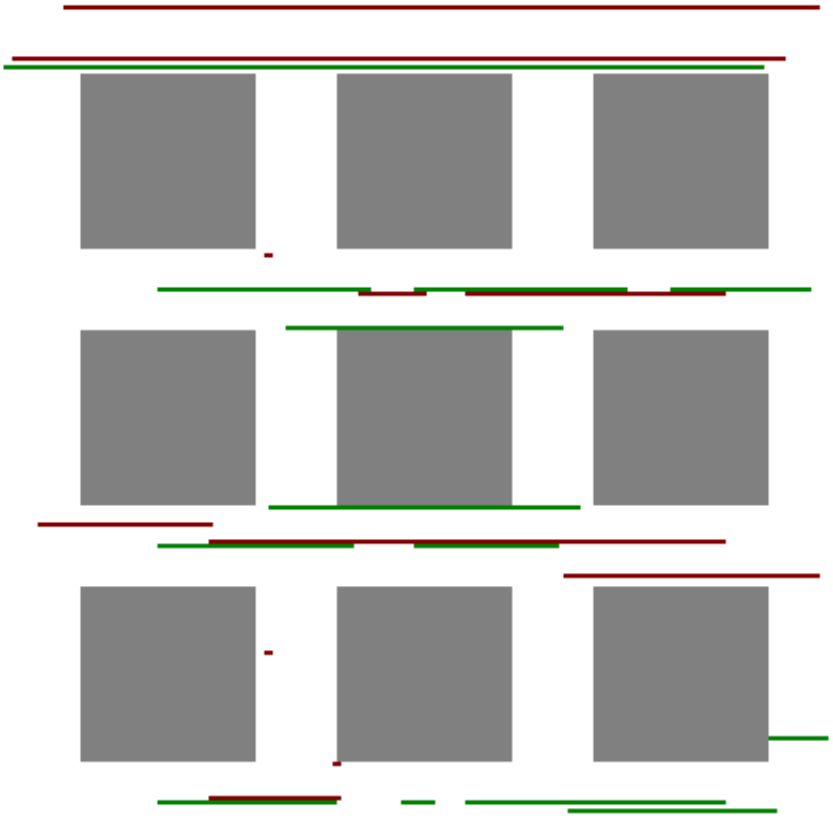
This part is implemented with modified TA's Code. We separate pins to the layer they belong. The result is at the end of this report.



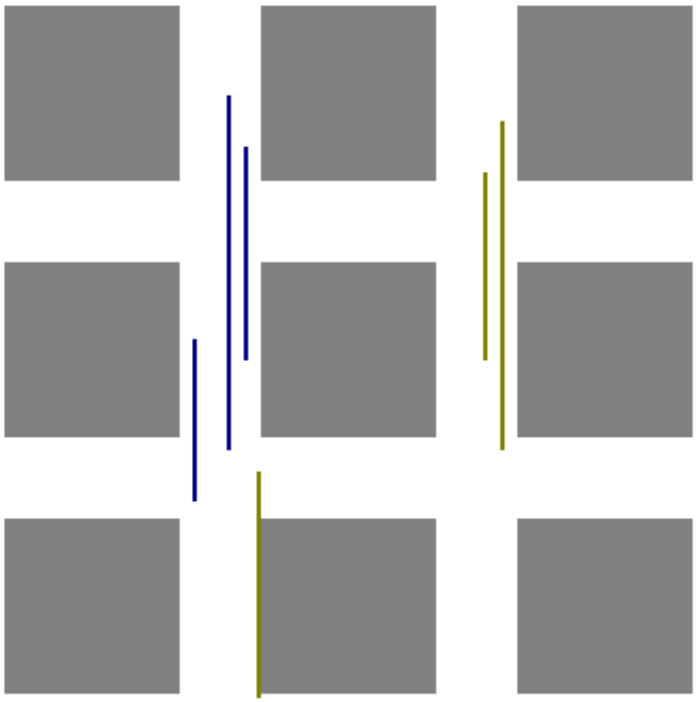
(a) Layer 1



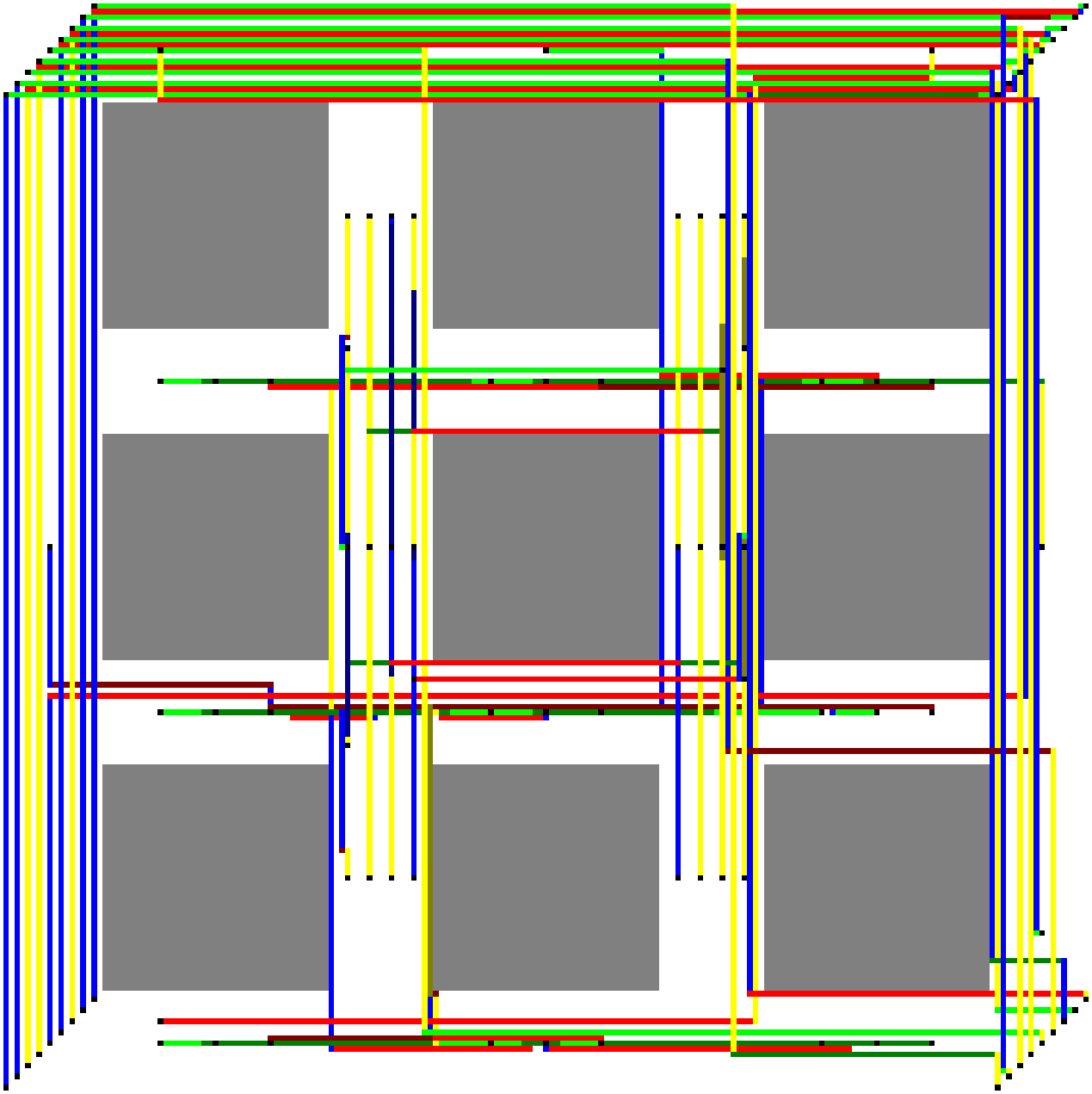
(b) Layer 2

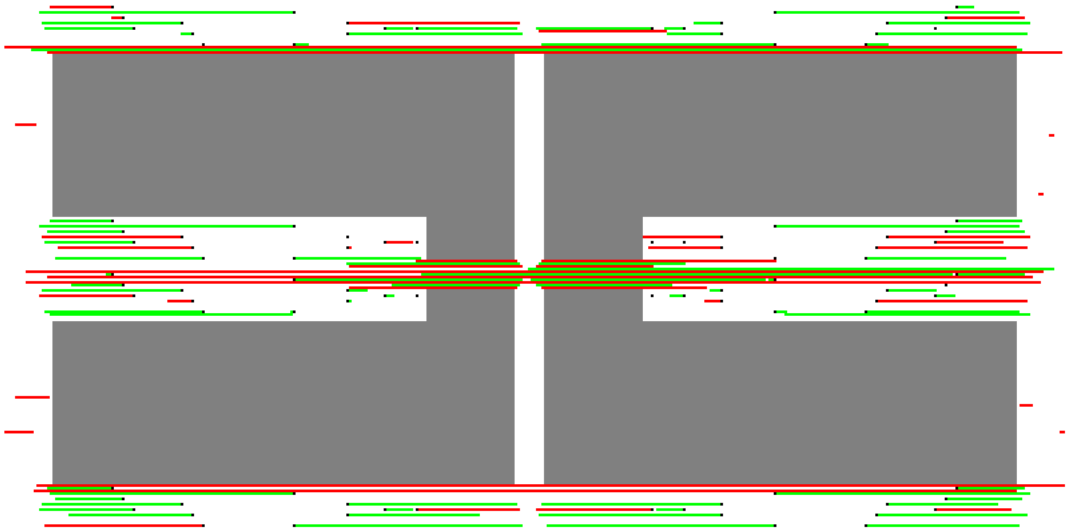


(c) Layer 3

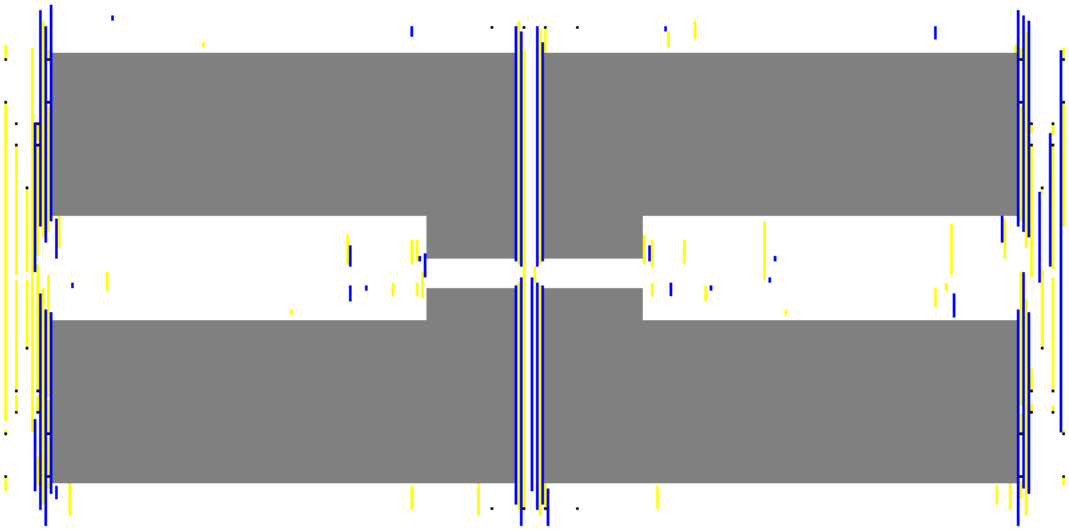


(d) Layer 4

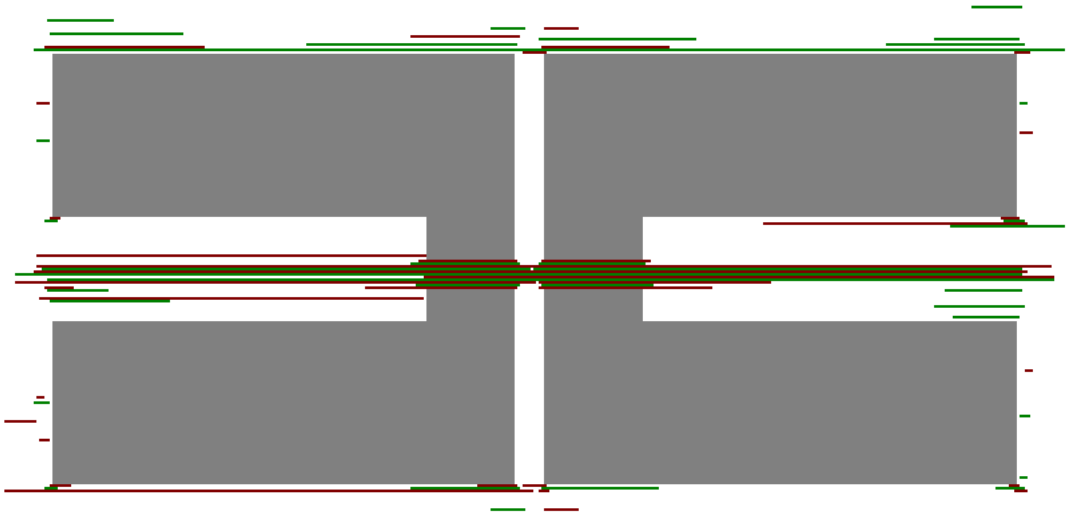




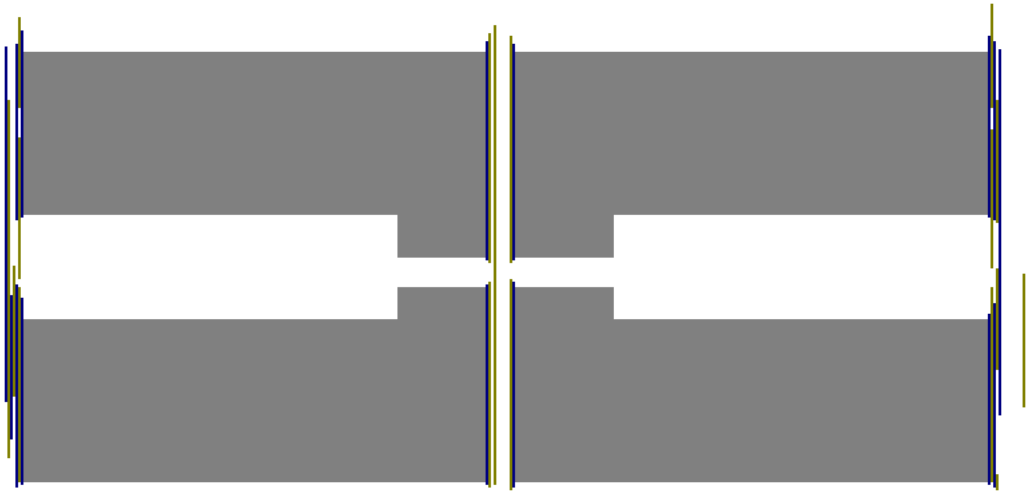
(a) Layer 1



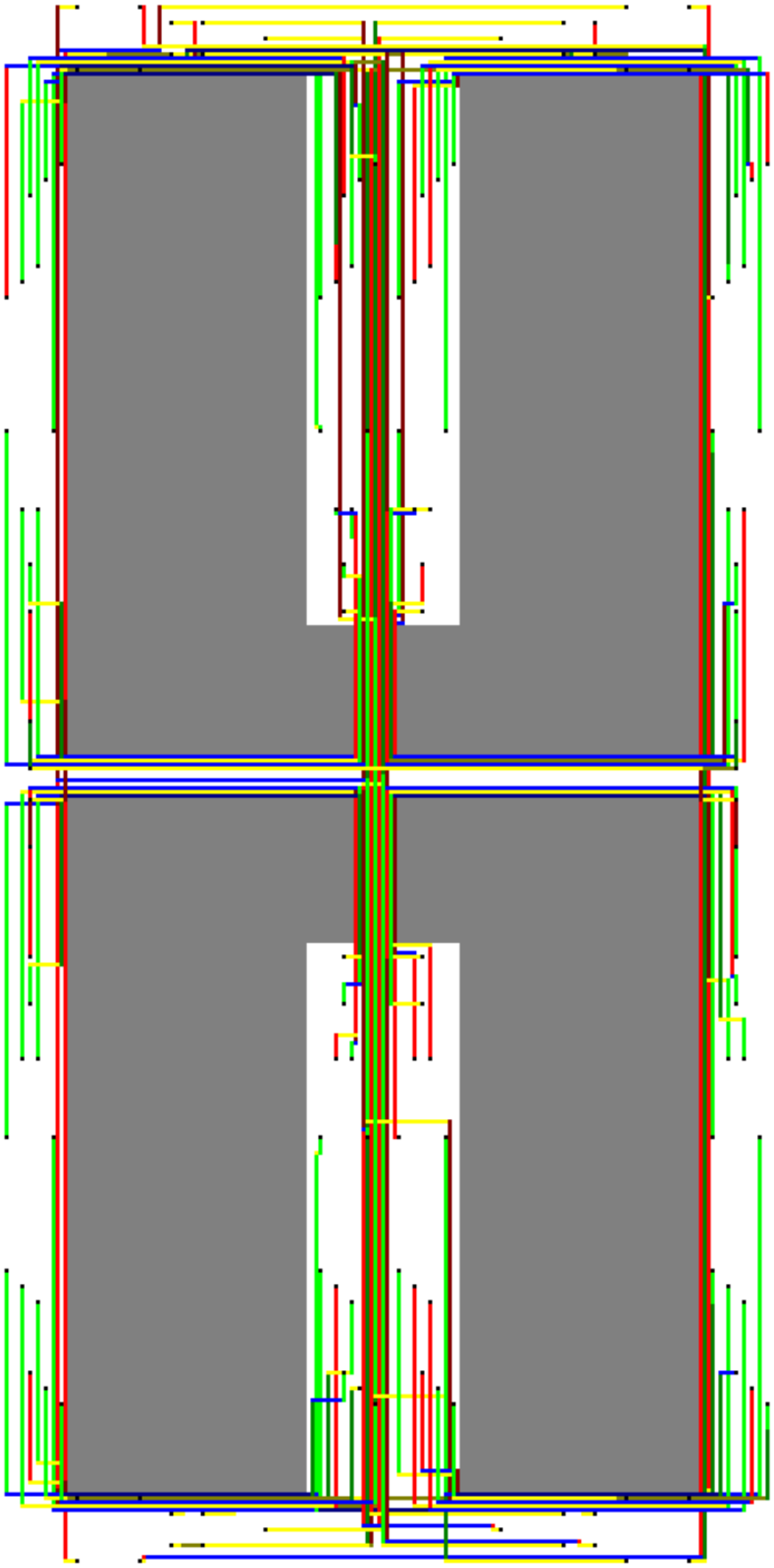
(b) Layer 2



(c) Layer 3



(d) Layer 4



7 Contribution

1. Main structure: **黃宣凱**
2. makefile Line 1∼21: **黃宣凱**
3. README Line 1∼21: **黃宣凱**
4. main.cpp Line 1∼103: **黃宣凱**
5. Routing.h Line 1∼51: **黃宣凱**
6. Routing.cpp:
   (a) Basic setting:
   Line 1∼66 class define: **黃宣凱**
   Line 68∼104 void visualize: **黃宣凱**
   (b) Preprocessing:
   Line 106∼126 void preprocessing: **黃宣凱**
   (c) Net ordering:
   Line 128∼132 struct cmpfororder: **陳映寰**
   Line 134∼208 int calH: **陳映寰**
   Line 210∼245 netordering: **黃宣凱** (**陳映寰**)
   (d) Two-pin connection
   Line 247∼249 unordered_map declaration: **蘇宸右**
   Line 251∼256 Heuristic of two pin: **蘇宸右**
   Line 258∼263 struct cmp2_p: **蘇宸右**
   Line 265∼283 Set_F: **蘇宸右** (**黃宣凱**)
   Line 285∼300 checkvalid: **蘇宸右**
   Line 302∼423 twopin: **蘇宸右** (**黃宣凱**)
   (e) Multi-pin connection
   Line 425∼430 struct cmp2: **陳映寰**
   Line 432∼523 MST: **陳映寰** (**黃宣凱**)
   Line 525∼527 bool compare: **陳映寰**
   Line 529∼544 multihandle0 (simply using MST): **陳映寰**
   Line 546∼646 multihandle3 (our Steiner algorithm): **陳映寰** (**黃宣凱**)
   (f) Output handling and map updating
   Line 648∼692 wiretran sform: **黃宣凱**
   Line 694∼699 markonmain: **黃宣凱**
   (g) Color balancing
   Line 701∼703 colorcompare: **陳映寰**
   Line 705∼808 color_balancing: **陳映寰**
   (h) Conducting
   Line 810∼846 routing: **黃宣凱**
7. Checker.cpp Line 1∼159: **蘇宸右**

References

[1] http://www.facweb.iitkgp.ernet.in/ isg/CAD/SLIDES/10-grid-routing.pdf

[2] Huang-Yu, Chen & Yao-Wen, Chang. (2009). Global and Detailed Routing. Electronic Design Automation. 687-749. 10.1016/B978-0-12-374364-0.50019-9.

[3] https://slideplayer.com/slide/9794852/

[4] Kahng, Andrew & Robins, G. (1992). A New Class of Iterative Steiner Tree Heuristics with Good Performance. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on. 11. 893 - 902. 10.1109/43.144853.

[5] A. B. Kahng and G. Robins. A new family of steiner tree heuristics with good performance: The iterated 1-steiner approach. In Proc. IEEE International Conf. Computer-Aided Design, pages 428–431, Santa Clara, CA, November 1990.

[6] A. B. Kahng and G. Robins. A new class of iterative steiner tree heuristics with good performance. IEEE Transactions Computer-Aided Design, 11(7):893–902, July 1992.

[7] A. B. Kahng and G. Robins. On Optimal Interconnections for VLSI. Kluwer Academic Publishers, Boston, MA, 1995.