Chip level global router

隋中彧
Department of Electrical Engineering
National Taiwan University
r12943117@ntu.edu.tw

孫定洋
Department of Electrical Engineering
National Taiwan University
r12943097@ntu.edu.tw

劉建亨
Department of Electrical Engineering
National Taiwan University
r12943094@ntu.edu.tw

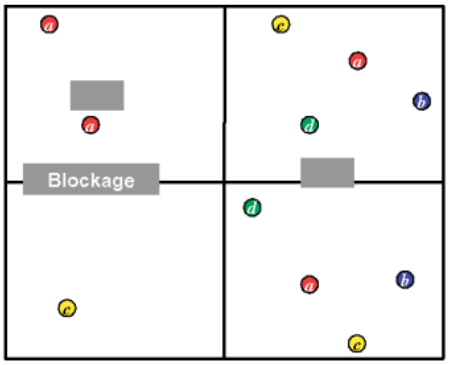
温千懿
Department of Electronics Engineering
National Taiwan University
r12921090@ntu.edu.tw

Abstract

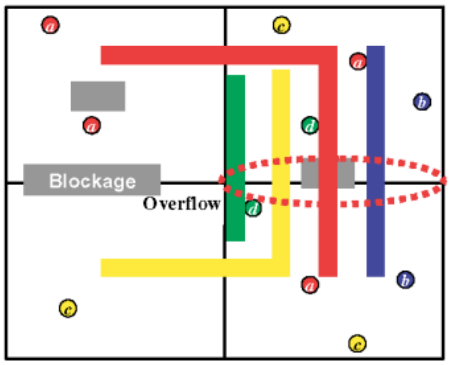
For modern large-scale circuit design, a chip may contain billions of transistors and millions of nets. To handle the high complexity, a routing algorithm often adopts the two-stage approach of global routing followed by detailed routing. In this final project, we implemented minimum spanning tree (MST) and modified Dijkstra algorithm on global routing. We also implemented hybrid-shape pattern and A*-search algorithm on detailed routing. When performing global routing at the chip level, four key cost factors must be considered: 1) channel overflow, 2) wirelength, 3) edge pin density, and 4) net turn cost - **Our primary objective function is to optimize the chip-level elements while considering *channel overflow* and *net turn cost,* ultimately generating the final routing layout.**

1 Introduction

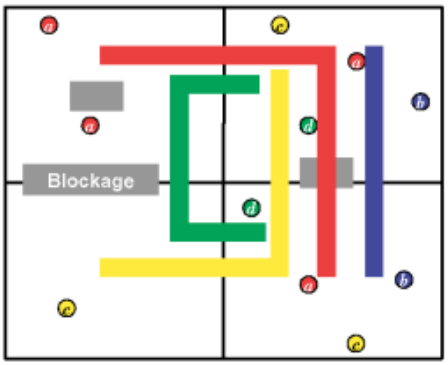
As manufacturing technology advances, modern circuits may contain billions of transistors, necessitating effective methods to reduce complexity. Today's physical design flow typically employs a hierarchical design style. After the planar layout is complete, the chip is divided into chip level and block level. Once the shape and position of the block are determined, the remaining area becomes the top channel for routing. Fig.1 Illustration of a modern routing system. The planning of the routing area for chip level is crucial because communication between blocks requires routing through the channel. To ensure the completion of the overall layout within a given timeframe, the shape of the channel must be adjusted to minimize detours and prevent congestion, while also minimizing the overall chip area by reducing the channel's size.



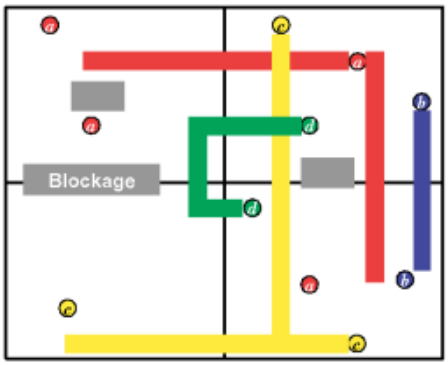
(a) After placement, the chip area is divided into global routing cells, with routing blockages in grey and four nets, *a, b, c*, and *d*.



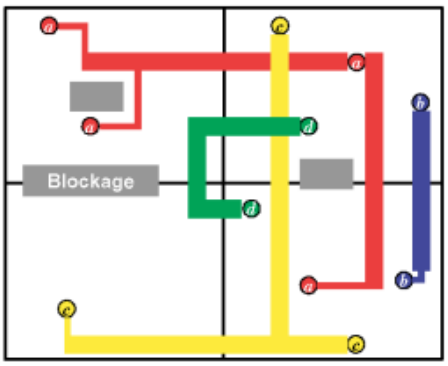
(b) A global routing solution where each net is routed in the shortest path, causing overßow (too many wires in a small region).



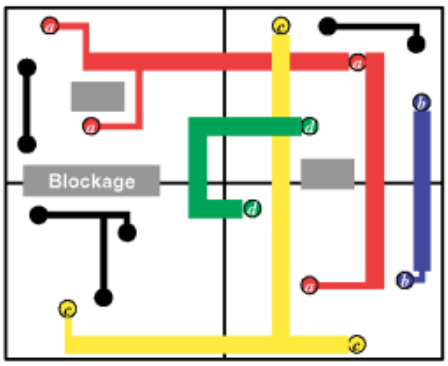
(c) The overflow is removed by altering the routing path of net *d* with more wirelength, which is the output from global routing.



(d) Track routing tunes the location of each segment in a way that the distance from the corresponding pins is minimized, avoiding routing blockages.



(e) Detailed routing connects a pin to the nearest wire for each net in a DRC clean manner.



(f) Detailed routing also completes local nets within each global routing grid.

Figure 1 : Illustration of a modern routing system which consists of three routing stages : global routing, track routing, and detailed routing

In this project, we will focus on global routing and detailed routing. Our chip level can be composed of the following five types of components:

1) **Channel**: The area of the chip remaining after removing blocks and tiles, which is the channel. This is illustrated in Figure 2 by the red dashed line.

2) **Feedthroughable block**: A block that allows nets to pass through. This is illustrated in Figure 2 by the deep blue dashed line.

3) **Non-feedthroughable block** : Except for hard macro feedthrough (HMFT), this block does not allow general nets to pass through. Its purpose is to further reduce the channel area and allow specific nets to pass through. This is illustrated in Figure 2 by the green dashed line.

4) **Region**: In addition to blocks, standard cells are scattered throughout the chip level and are allocated to different regions based on function. This is illustrated in Figure 2 by the yellow frame.

5) **Tile**: Composed of regions and blocks, allowing nets to pass through. The chip can be further divided into multiple tiles to reduce the execution time of the chip level, as shown in Figure 2 by the light red dashed line.

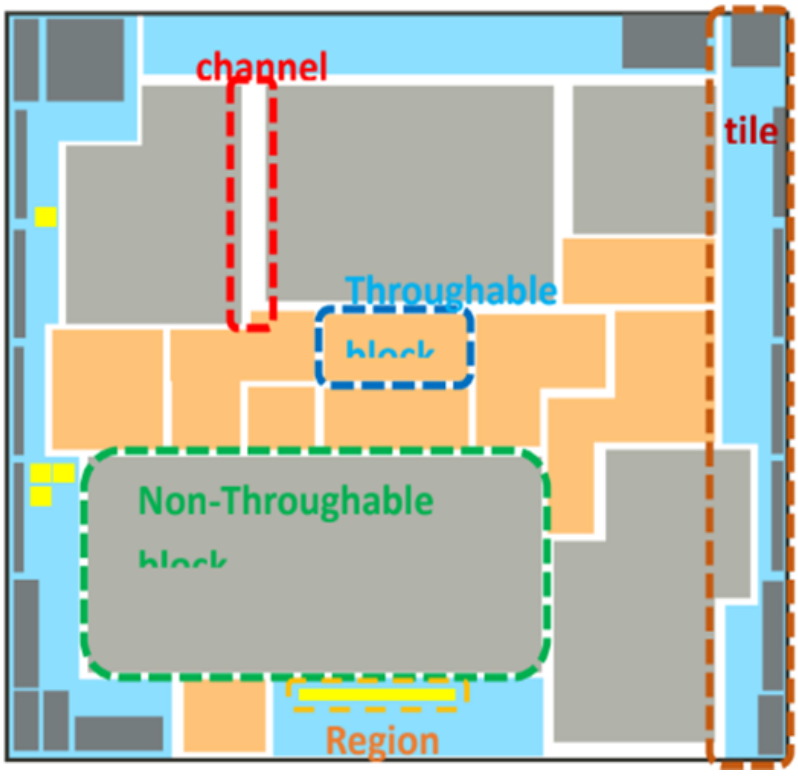


Figure 2

Designing a lightweight and fast channel global router that can optimize wire length is a crucial task in chip design, as it significantly impacts the overall performance of the chip. However, the differences between the routing areas at the chip level and within the blocks are significant, making chip-level routing more challenging compared to block-level routing.

Within the blocks, the routing area is typically intact. However, at the chip level, the routing can only be done in the remaining area after the blocks are placed, resulting in the routing area being composed of many irregularly shaped channel regions.

Since we will focus on chip-level routing in this project, we hope to achieve the following:

1) The first step is to convert the input file, which includes the status of physical layout, block placement, block shape used on the chip top, and net properties, into a grid-based graph for global routing.

2) The multilevel routing approach begins by applying the minimum spanning tree (MST) and modified Dijkstra algorithm to decompose each net into 2-pin connections. The order of each connectable point is then defined.

3) Hybrid-shape patterns and A* search are implemented on detail routing to calculate the best routing results.

4) The official evaluator formula is used to calculate the optimal routing layout for each case..

2 Related Work

Research in VLSI routing has garnered significant attention in the literature. Routing is typically a very complex combinatorial problem, which is challenging to solve. To make the routing problem manageable, a two-stage approach is commonly employed, involving global routing followed by detailed routing. After placement, we have a placed layout, as shown in Figure 3a, which includes the exact locations of blocks, pins of blocks at chip boundaries. Additionally, we are provided with a netlist that describes a list of connections by specifying which pins should be electrically connected to form a set of nets. Figure 3b illustrates the global-routing paths. The global-routing process begins by dividing the routing region into tiles and then generates a "loose" route for each connection by finding tile-to-tile paths to connect pins and/or pads. Figure 3c illustrates the result of detailed routing, which determines the exact route for each net by searching within the tile-to-tile path.

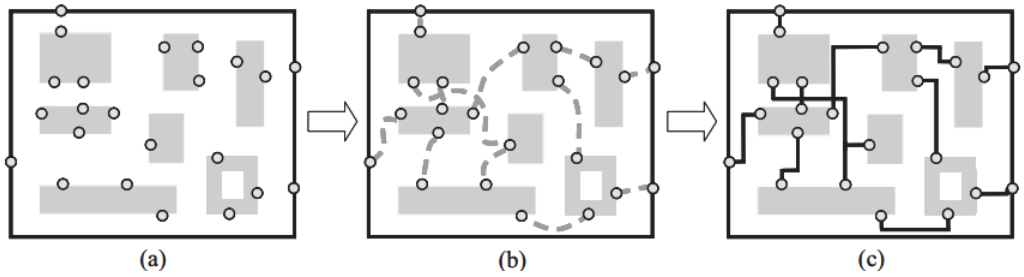


Figure 3 : (a) after placement (b) global-routing (c) detailed-routing

2.1 Routing Models

There are two kinds of detailed-routing models: (a) grid-based (b) gridless models. For grid-based routing, a routing grid is superimposed on the routing region, and then the detailed router finds routing paths in the grid, as shown in Figure 4. The space between adjacent grid lines is called **wire pitch**. Note that the router has to control the searching space such that the path in the horizontal/vertical layers can only run horizontally/vertically for the reserved layer model. In this way, the wires with the minimum width following the path in the grid would automatically satisfy the design rules. Therefore, we adopt a grid-based detailed routing model to deal our input information because it is much more efficient and easier for implementation.

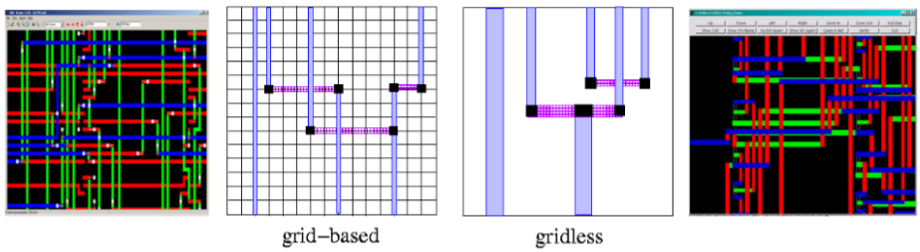


Figure 4 : Routing Models grid-based and gridless

2.2 Minimum Spanning Tree Algorithm

This is due to Kruskal and was first presented in [3] in 1956. Given a layout graph , the algorithm initializes |V| sets, each consisting of one vertex from the graph. At each iteration of the algorithm, it finds the minimum weight edge to connect two disjoint sets. It then takes the union of these sets and adds the edge to the collection of edges making up the minimum spanning tree. The pseudocode for the algorithm is given below.

Algorithm : Minimum Spanning Tree of global routing

V : all the TX(start point of routing), RX(end point of routing) of the net
E : V x V
w : simply use the Manhattan distance between the TX, RX coordinate of block given by problem to calculate edge weight

```
1: T← empty;
2: for each vertex v ∈ V do
3:         MAKE-SET(v);
4: end
5: sort edges in non-decreasing order by weight w;
6: for each edge ( u, v) ∈ E taken in non-decreasing order by weight w do
7:         S(u)← FIND-SET(u);
8:         S(v)← FIND-SET(v);
9:         if  S(u) ≠  S(v) then
10:             T ← T U {(u, v)};
11:             UNION(S(u), S(v));
12:       end
13: end
14: return T;
```

2.3 Modified Dijkstra Algorithm

Dijkstra's algorithm finds the shortest path from one vertex to all other vertices. It does so by repeatedly selecting the nearest unvisited vertex and calculating the distance to all the unvisited neighboring vertices. Our modified Dijkstra algorithm is shown in below and Figure 5.

1) Find the RX node (red point) first, do Dijkstra till a shortest path to some TX node (green node).

2) View the shortest path on (1) as a supernode, relax other nodes that connect to this supernode.

3) While relaxing, if we found a node that has already popped out of the heap but not in a path of TX, RX node. We still relax it and push it back to the heap if its key value decreases.

4) And from supernode, repeat previous steps till all RX is connected.

Notice that in our Algorithm :

1) Nodes include blocks, regions, and cut channels. Thus, all the grid available belongs to some node.

2) Edge means (u, v) exists if u is beside v on the grid map.

3) Edge weight means the function of the

a) wire density and capacity of u and v

b) distance between u and v

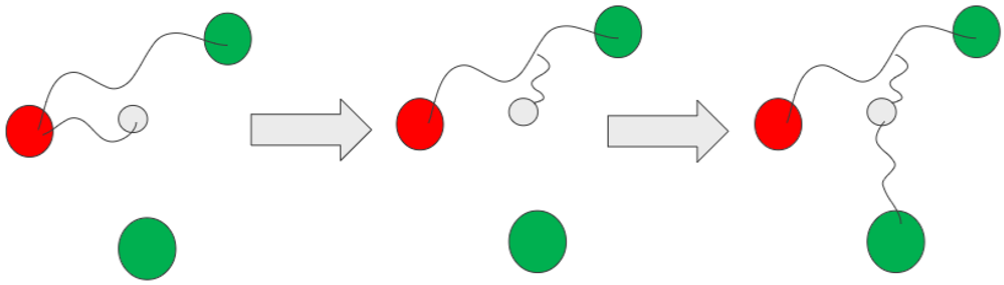


Figure 5 : Modified Dijkstra diagram

2.4 A*-search Algorithm

In [Hart 1968], a general graph search algorithm called A*-search was proposed, which uses the function f(x) = g(x) + h(x) to evaluate the cost of a path x, where g(x) is the cost from the source node to the current node of x, and h(x) is the estimated (or predicted) cost from the current node of x to the target node, the most commonly used distances are Chebyshev distance and Manhattan distance.(Figure 6) Every time the algorithm selects a node with the lowest path cost to propagate (i.e., the lower f(x)), the higher the priority for propagation. As a result, the A*-search is also called the best-first search, because at each decision making it first searches the routes that are most likely to lead toward the target.

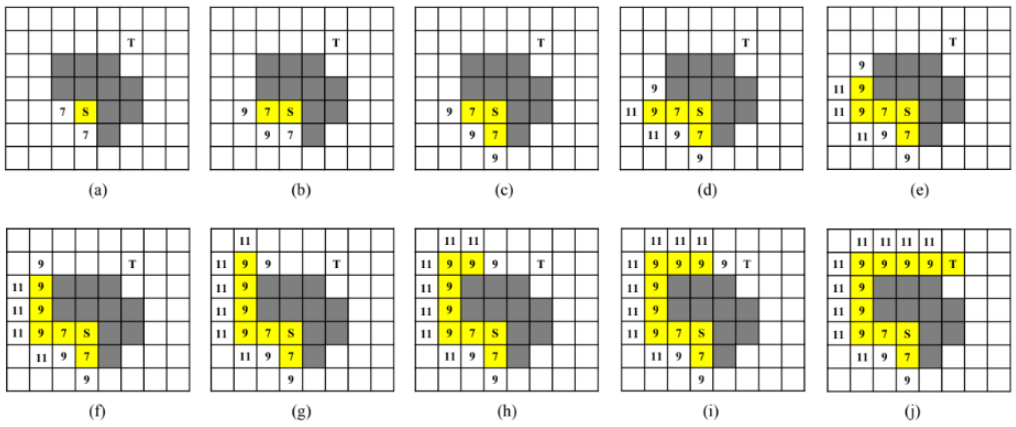
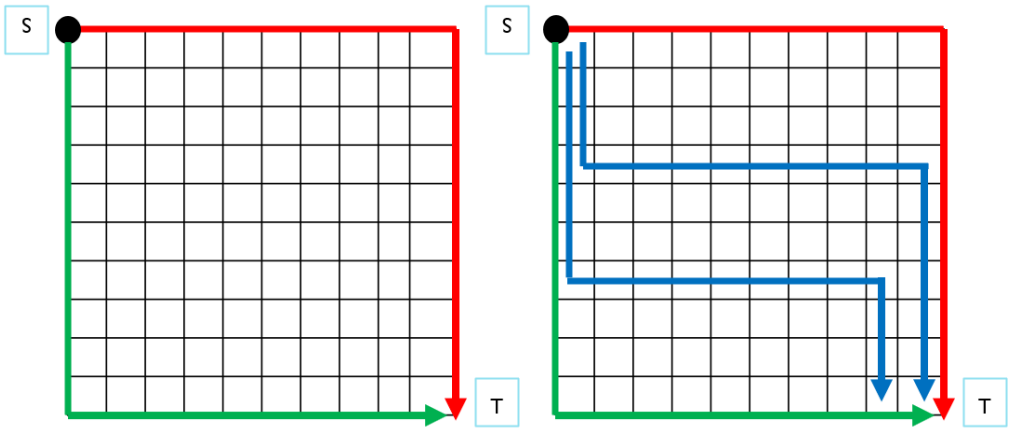


Figure 6 : A routing path from S to T using the A*-search routing

2.5 Hybrid-shape pattern : L-shape (1-bend) + Z-shape (2-bend) routing

The concept of L-shaped channels was first introduced in RRDO[5] to generate a feasible routing order for non-slicing-structure placement in building-block layout design. Our global router initially attempts L-shaped pattern routing because of its lower net turn cost. However, this approach can lead to the outer area being more crowded while the inner area remains idle, potentially causing an overflow problem, so we add the concept of Z-shaped pattern routing. (Figure 7) We combine L-shape and Z-shape pattern routing to form a hybrid-shape pattern routing algorithm, which enhances the two-stage global router framework with a better global routing solution and minimal runtime overhead. (Figure 8)



(a) (b)

Figure 7 : (a) The diagram only uses L-shape, the inner area remains idle (b) Combine L-shape and Z-shape pattern routing to form a hybrid-shape pattern that can make the chip more evenly distributed.

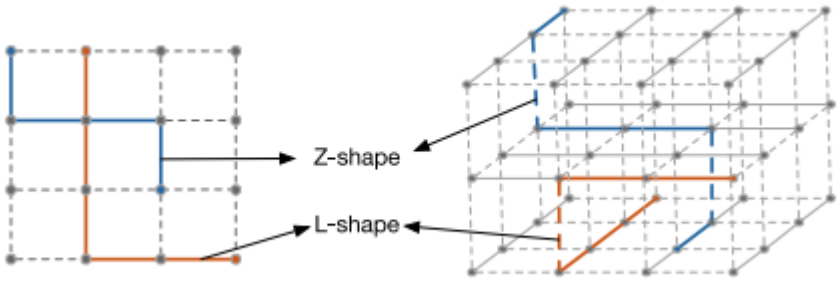


Figure 8 : 2-D/3-D pattern routing; the brown path represents one L-shape pattern routing solution, and the blue path is one of the candidate Z-shape pattern routing paths.

3 Proposed Method

Initially, we will implement global routing using the Minimum Spanning Tree and Dijkstra algorithms, followed by the A* search algorithm and hybrid-shape pattern for detailed routing. The results will be visualized using Python.

3.1 MST and modified Dijkstra on global routing

After processing the input file, we obtain the complete physical layout information for the circuit. For the netlist, the multilevel routing approach first applies the minimum spanning tree (MST) and modified Dijkstra algorithm to decompose each net into 2-pin connections.

3.2 Hybrid-shape pattern and A*-search on detail routing

In the implementation of detail routing, we have two methods: 1) use a hybrid-shape pattern (L-shape and Z-shape) for routing, 2) execute A* search. We will prioritize the first method because its edge turn cost is lower due to the fact that there are at most 2 turns per wirelength. However, this approach has the drawback of potentially causing overflow issues.

Therefore, we will first randomly select a point in the starting position area and begin executing the hybrid-shape pattern routing algorithm. If there are non-throughable areas in the middle, we will randomly select again and repeat the same steps. If both attempts are successful in obtaining a routing path, we will choose the path with the smaller cost and calculate the wirelength on this path.

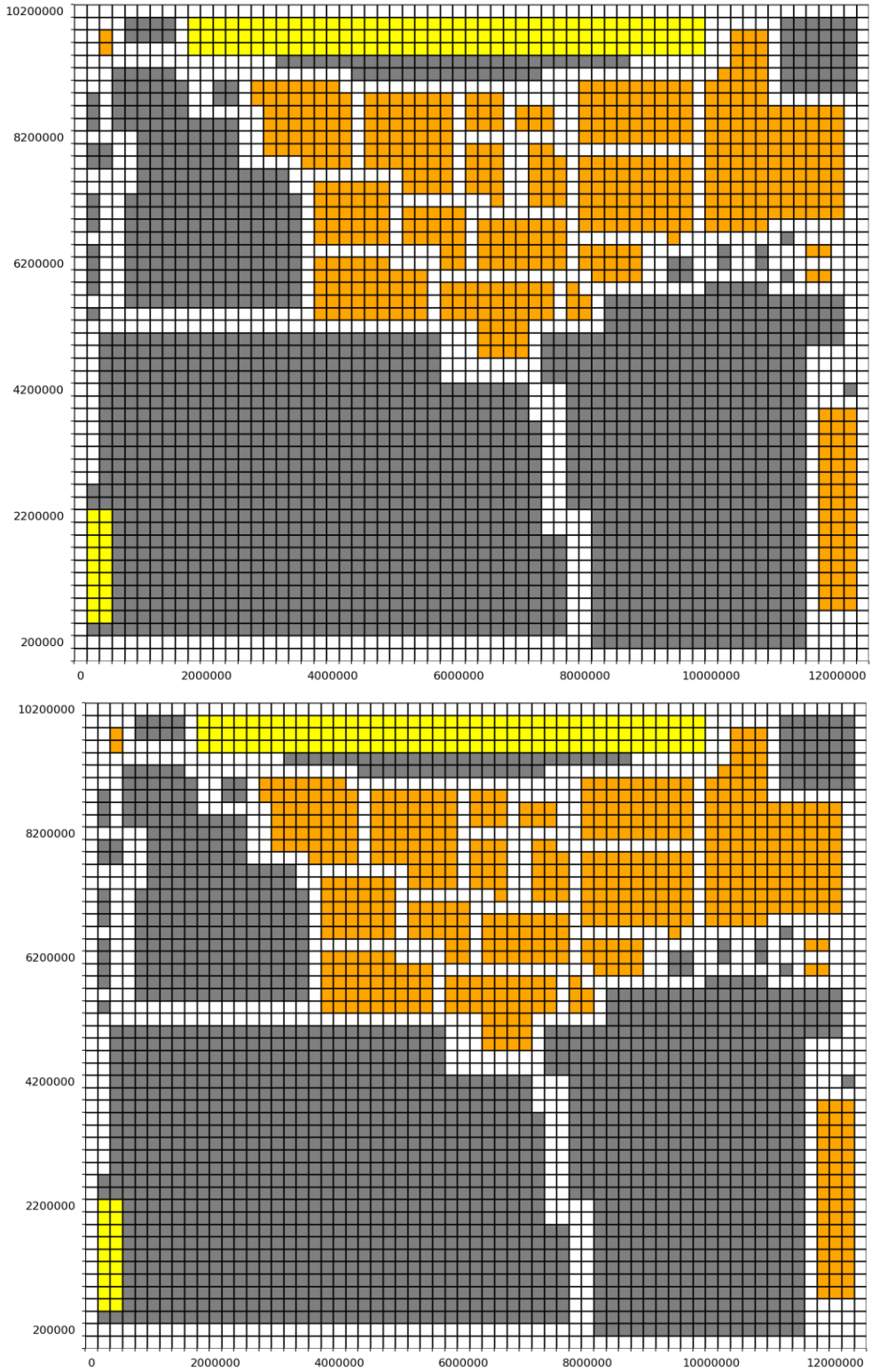
The second method is to execute A* search, which indicates that we encounter consecutive areas that cannot be traversed due to obstacles. Our method differs from the textbook in that we will add a new variable **w(x)** to the function, which calculates the number of wires that have passed through each grid.

As a result, the cost function becomes $f(x) = g(x) + h(x) + w(x)$, considering overflow and wirelength optimization at the same time.

4 Result

The results are in the following picture (use python tools to draw). We have encountered great difficulties on the issue of calculating optimal routing layout, and we still made a lot of improvements. The following data shows the result of case1(top) and case2(bottom). The yellow area is the region, the orange area represents the throughable block, the gray area that requires specific conditions to pass through.

The following is our original diagram of case1(top) and case2(bottom)



(A) The **initial result** (Figure 9) is calculated based on our test data (small number of data), which we have already shown in the final presentation. If density exceeds a certain value, it will turn red. Lines that are close together will overlap. The denser the line segments, the thicker they will be. We focus on the wirelength. Case 1 has a more crowded overall layout compared to Case 2

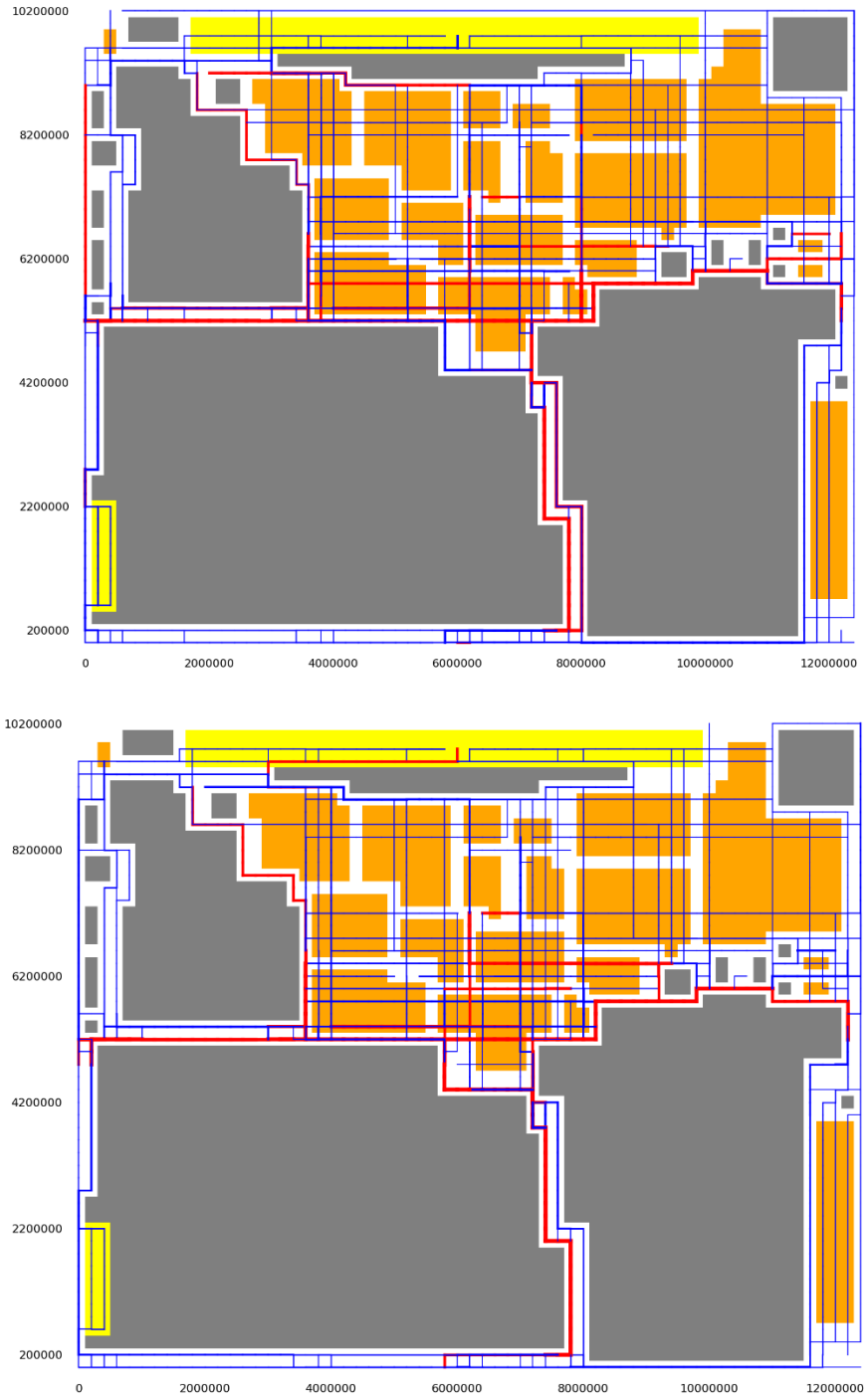


Figure 9 : case1(top) and case2(bottom)

(B) The **second result** (Figure 10) is calculated based on the complete contest case information. We focus on the wirelength and overflow, if an area exceeds 70 wirelengths, it will display red. Also, we add the grid graph on it, so that we can see the layout of the wires more clearly. We also corrected the issue where some wires would run into non-throughable blocks during the first result.

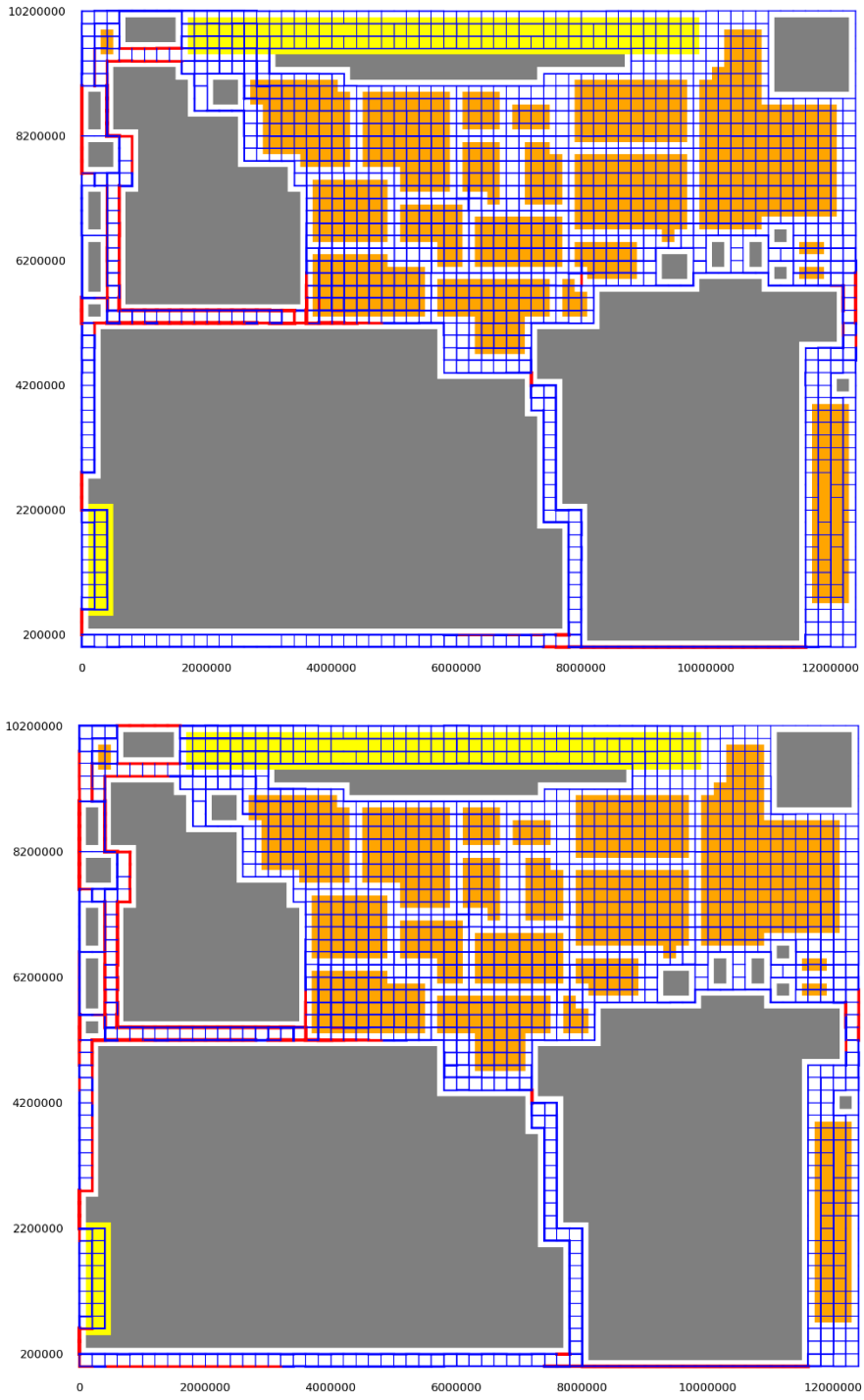


Figure 10 : case1(top) and case2(bottom)

(C) The **third result** (Figure 11) is calculated based on the complete contest case information. We add the consideration of net turn. The final improved version is clearly more balanced and uniform compared to the previous two. The red area representing congestion has shrunk.

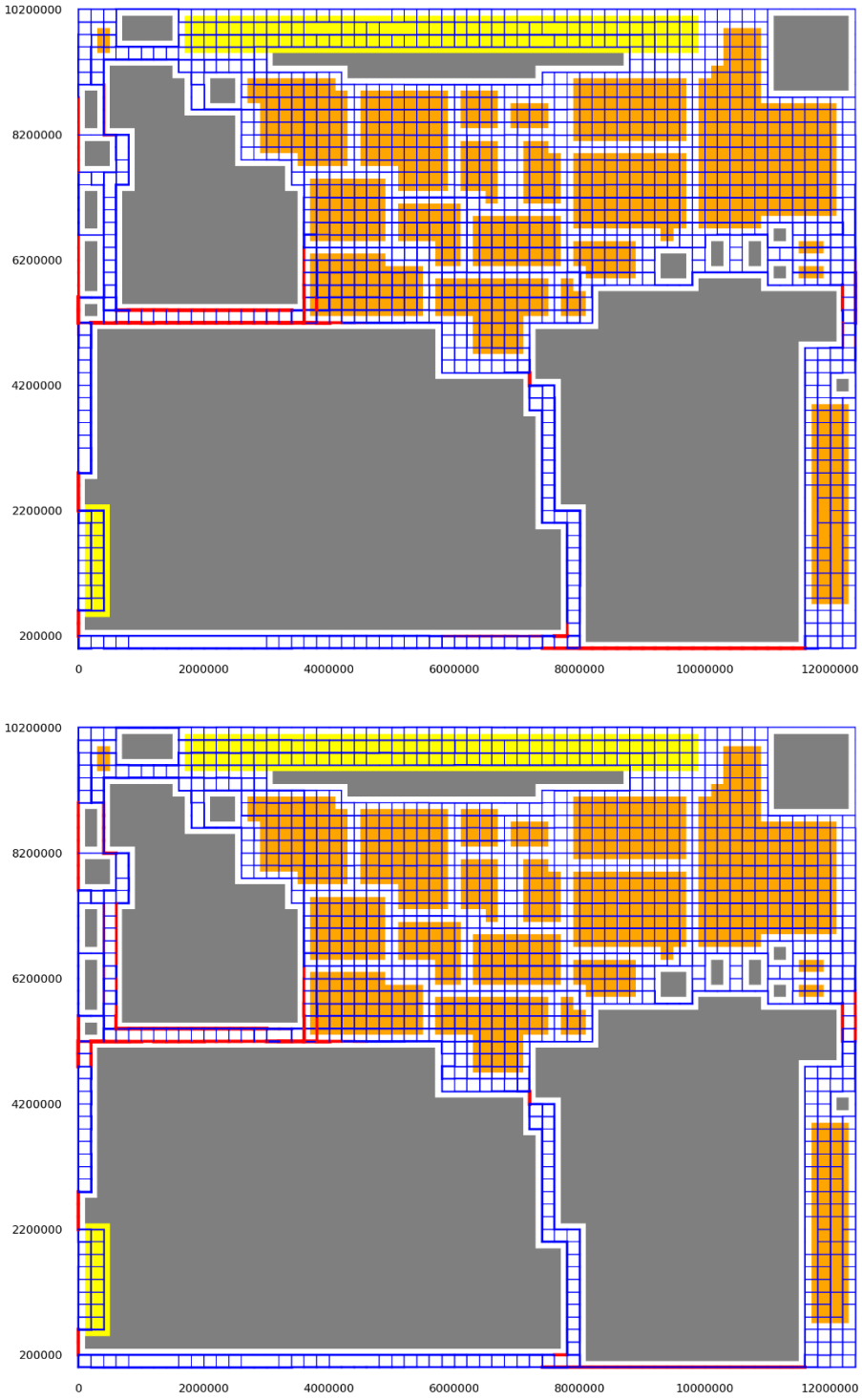


Figure 11 : case1(top) and case2(bottom)

5 Conclusion and Future work

	Data volume	Global routing	A*-search	L-shape	Z-shape	Cost
(A) initial result	test case (small number)	V	V	X	X	wirelength
(B) second result	total	V	V	V	X	wirelength + overflow
(C) third result	total	V	V	V	V	wire length + overflow + net turn

Figure 12 : Comparison table of algorithm implementation item for each case

		Overflow cost	Evaluator finish time
(A) initial result	Case 1	0.00125367	3.599
	Case 2	0.00123145	2.086
(B) second result	Case 1	0.000988011	4.185
	Case 2	0.000941805	3.489
(C) third result	Case 1	0.000912482	2.928
	Case 2	0.000836869	2.169

Figure 13 : The cost and finish time of every result and case

According to the tables in Figures 12 and 13, the third result shows that by adopting a more comprehensive Hybrid-shape pattern and considering more cost factors, the overflow cost for case 1 decreased by 7.6% compared to the second result, and the overflow cost for case 2 decreased by 11.5% compared to the second result. Additionally, the evaluation time for the evaluator was accelerated by 30% for case 1 and 37.8% for case 2.

In our implementation, due to some illogical aspects of the official test cases, we did not present the net turn cost data. We are also continuously thinking about how we would modify the algorithm if we encounter larger obstacles in real-world scenarios. In the future, we will strive to further enhance the practicality of this project by incorporating a clock routing algorithm. This addition will not only improve the overall performance of the design but also demonstrate the versatility and adaptability of our approach.

References

[1] 2024 積體電路電腦輔助設計軟體製作競賽

[2] Global and detailed routing

[3] Path finding solutions for grid-based graph

[4] J.B. Kruskal, On the shortest spanning subtree of a graph and the traveling salesman problem, Proceedings of the American Mathematical Society 7 (1956), 48-50.

[5] Global routing in VLSI: algorithms, theory, and computation by chris dickson, b.sc

[6] W.-M. Dai, T. Asano and E. S. Kuh, "Routing region definition and ordering scheme for building-block layout", IEEE Trans. Computer-Aided Design, vol. CAD-4, no. 3, pp. 189-197, July 1985.

[7] FastGR: Global Routing on CPU–GPU With Heterogeneous Task Graph Scheduler

[8] Manufacturability Aware Routing in Nanometer VLSI