A Maze Router for 2-Point Nets in ASICs

Zhaojun Ni
School of Information Science and Technology
ShanghaiTech University
Shanghai, China
nizhj2022@shanghaitech.edu.cn

Bin Ning
School of Information Science and Technology
ShanghaiTech University
Shanghai, China
ningbin2022@shanghaitech.edu.cn

***Abstract*—In this project, we build a real maze router to handle 2-point nets with two layers. We improve the DFS algorithm learned in the class, while significantly accelerating the algorithm's speed and successfully route all the nets in the provided benchmarks.**

***Index Terms*—ASIC, routing, DFS**

I. Introduction

In ASIC design, routing is a crucial step that involves connecting the various components and interconnecting wires on a chip. The goal of routing is to establish an efficient and reliable network of paths for signals to travel between different elements of the circuit. The routing problem in ASIC design refers to the challenge of determining the optimal paths for these interconnections while considering various design constraints.

Maze Routing is a representative method to address the routing problem in ASIC design. This method treats the chip as a maze and employs graph-based algorithms like Depth-First Search (DFS) [1] or Breadth-First Search (BFS) to find feasible paths from source to destination.

The project assignment has two parts. In phase I, a core router should be built first to handle 2-point nets, non-unit costs in the routing grid, bend penalties, and 2 separate routing layers. Each benchmark has two input files which describe the geometry of the routing grids (.grid file) and the netlist of wires to connect (.nl file) respectively. The routing results should be outputted in the order of net_id and formatted according to the specified format (.route file). In phase II, the benchmarks are more industrial and more complex. The two layers are suggested to have different preferred directions, which means restricting alternating layers to be mostly vertical and mostly horizontal. The goal of the project assignment is to route as many nets as possible while minimizing the overall cost.

Our improved DFS-based algorithm can enhance routing efficiency while ensuring the completion rate of the routing. The main contributions of this work are as follows.

- When calculating the cost between the source grid cell and the current grid cell in Phase II, we have introduced additional bending costs for the first layer in the horizontal direction and for the second layer in the vertical direction. This cost increases with the length of the routing bend, allowing the first layer to be routed predominantly in the vertical direction and the second layer to be routed predominantly in the horizontal direction.
- We have drawn inspiration from the A* algorithm and improved upon the DFS algorithm. In addition to calculating the cost from the source grid cell to the current grid cell (actual cost), we also estimate the cost from the current grid cell to the target grid cell (estimated cost). This can significantly improve the speed of path searching.
- To ensure both the completion rate and lower overall cost of the routing, we employ multiple iterations. In each iteration, we prioritize handling the routing failures from previous iterations and nets that are far apart in terms of source and target grid cells. Additionally, we utilize a gradually conservative cost evaluation which allows gradually increasing time waits during the iterations. This approach allows for a more accurate evaluation of the overall cost, which helps to find a more optimal routing solution.
- We have implemented visualization for the routing problem and routing results, as well as a reliability assessment for the output results.

II. Overall Structrue

The overall structure of this work is illustrated in Fig. 1. The netlist file and grid file are parsed separately by the file parsing module to obtain the points to be routed and the routing layers. The layer information includes the cost and restricted regions of each layer grid, as well as bend penalty and via penalty, which can be visualized. The nets to be routed are sorted in descending order based on the distance from the source to the target, and then sequentially inputted into the maze router.

The router extends from the source pin like a wave, and based on the layer information, as well as the previously mentioned Preferred Direction and Cost Estimation, it calculates the cost and selects the point with the minimum cost for expansion. During the expansion process, if the target point is reached, the path is backtracked. In this module, if a node cannot reach the target point or exceeds the specified time limit, the node is added to the badnets collection and will be routed in the next iteration.

Based on the iterative mechanism, the iteration process is carried out to select the best routing result and evaluate it. Finally, this result is converted into an output file and visualized.

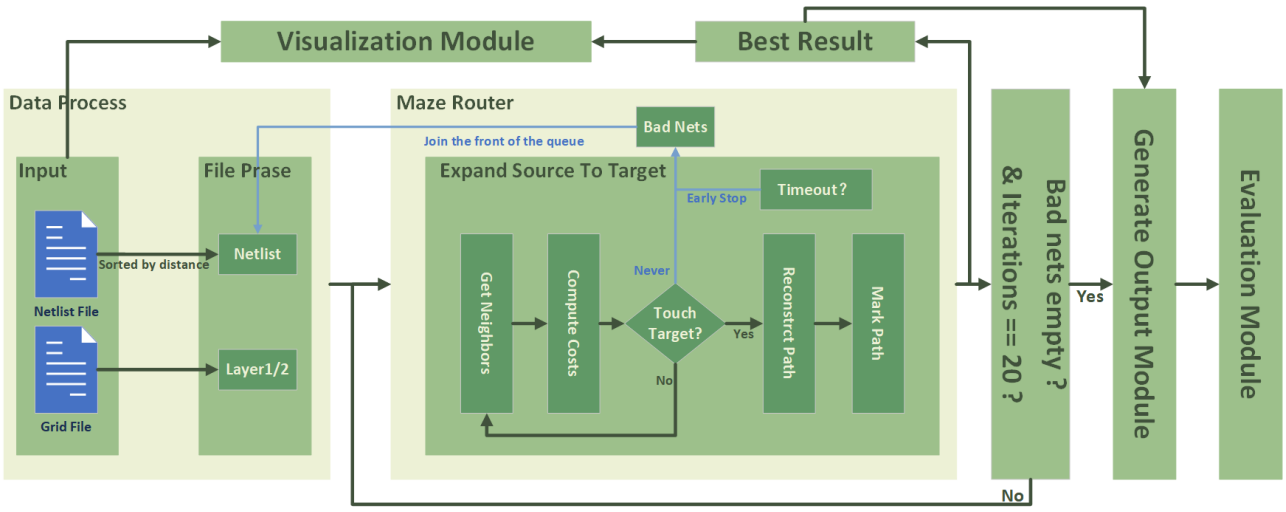


Fig. 1. The overall structrue of our router.

III. Proposed Methods

A. Preferred Direction

The way a real, industrial-strength router manages to route millions of nets on 8-12 layers of physical metal routing, is to impose preferred direction constraints on the layers. So, it is suggested to make Layer 1 mostly-vertical routing, and Layer 2 mostly horizontal. Although this approach may introduce a significant number of vias, it effectively utilizes the area of two layers during routing, reducing conflicts within the same layer and greatly improving the success rate of routing. Through experiments, we found that strictly one-way routing leads to a high penalty for via usage. Therefore, we opted for preferred one-way routing.

Our implementation is as follows. When expanding neighboring grid cells for the current grid cell, we still consider the 6 directions: forward, backward, left, right, up, and down. However, we do not expand into positions occupied by obstacles. Additionally, when calculating the actual cost, we first check if the current position has a via or a bend. Starting from bench5 and onwards, we introduced an increasingly severe bend penalty. We determine whether the path on Layer 1 has horizontal bends. If so, we measure the length of the bends and multiply it by the bend penalty to calculate the additional cost. Similarly, on Layer 2, we track the occurrence of vertical bends. This idea was derived from our experiments with strictly one-way routing. We found that if we only allowed neighboring grid cells to expand in 4 directions, we encountered situations where vias were placed adjacent to each other. This resulted in a cost of 40 (2 $\times$ via_penalty), while the cost for a single bend was only 10 (bend_penalty). However, we wanted to minimize the occurrence of bends during routing. Hence, we introduced the concept of an increasingly severe bend penalty (additional bend penalty), where the cost for progressing along a bend increases progressively.

By applying the increasingly severe bend penalty, we discourage excessive bending in the routing paths, effectively promoting more preferred-direction routes while still allowing for some flexibility when necessary.

B. Cost Estimation

In traditional DFS algorithm, we calculate the distance from the source cell to the current cell, which is known as the actual cost. When traversing the neighboring cells of the current cell in the traditional DFS algorithm, if a new neighbor cell has not been visited before or its actual cost is smaller than the cost stored in the original wavefront, the neighbor cell and its cost are updated in the wavefront. In the next iteration, the wavefront is first re-sorted in ascending order based on the cost, and then iterated through sequentially.

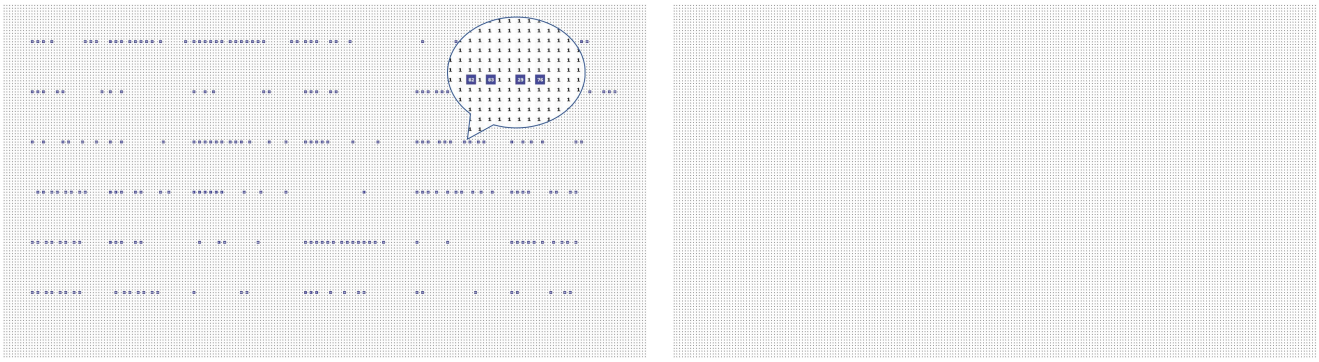
However, it is important to note that this method's efficiency can be low in cases where two cells have the same actual cost but different distances from the target cell, or when a cell has a very small actual cost but is far away from the target cell. To address this, we simultaneously consider the actual cost (g(n)) and the estimated cost (h(n)). The sum of the two is referred to as the overall cost, as shown in (1).

$$f(n) = g(n) + h(n) \tag{1}$$

The estimated cost consists of two components. The first component is the aforementioned increasingly severe bend penalty. The second component is the Manhattan distance between the current cell and the target cell, multiplied by the via penalty and the learning rate (eta). It is important to note that the second component is only an estimation. Thus, after the new f(n) calculation, the priority of cells in the wavefront may change, but the path connecting the source and target cells may not necessarily be optimal.

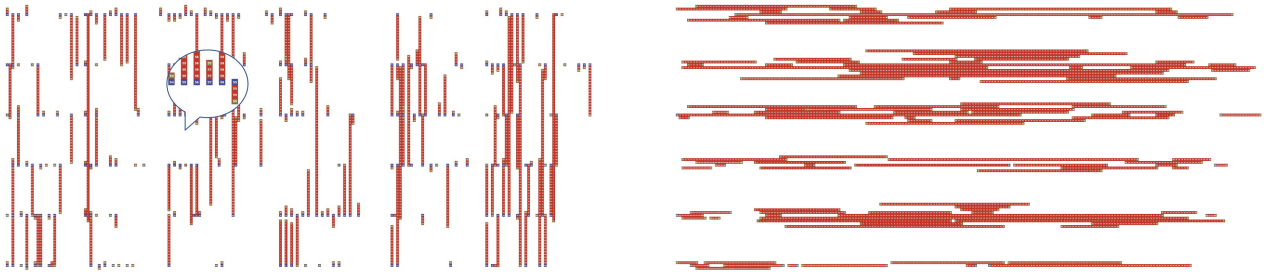
C. Iterative Mechanism

To address the problem raised at the end of the previous section and obtain a better global solution, we introduce an iterative mechanism. If we obtain successful routing results for all nets in the iteration, we will evaluate their total costs



(a) Layer 1 (b) Layer 2

Fig. 2. Visualization for the routing problem of fract2.



(a) Layer 1 (b) Layer 2

Fig. 3. Visualization for the routing result of fract2.

and save them. If there are subsequent iterations with better costs, we will save the best one.

a) Nets reordering: Before the initial iteration, we first sort the list of nets that require routing. We arrange the distances between the two points of each net in ascending order, so that the paths routed earlier have minimal impact on the subsequent routing. Additionally, we bring the furthest 10% of nets to the front of the list, thereby reducing the pressure of difficult routing problems. We then proceed with the first iteration, referred to as pre-routing. If there are any nets that have not been successfully routed during the first iteration, they will be stored in the bad net list. These nets will be prioritized by moving them to the front of the nets priority list, ensuring they receive priority routing in the subsequent iterations, which is called detailed-routing.

b) Learning Rate: As the iterations progress, we gradually decrease the second component of h(n) using a decaying learning rate. This ensures that the cost calculation becomes closer to the actual situation. For different benchmarks, we have set different initial learning rates and step sizes. We believe that more complex benchmarks should have higher initial learning rates and larger step sizes. The values of both parameters are linearly related to the number of nets that require routing. However, as the learning rate decreases, the routing process becomes more cautious and strict. It is possible that nets that were successfully routed previously may fail to be routed as the learning rate decreases. Therefore, we calculate the number of routing failures in each iteration. If the number of failures in the new iteration is less than or equal to the number of failures in the previous iteration, we continue to lower the learning rate. Otherwise, we keep the learning rate from the previous iteration.

c) Early Stopping: In order to ensure the efficiency of path searching, we introduced time management. When the path searching time of a certain net exceeds the set value, we consider this net as routing-hard and interrupt it in advance (early stopping), adding it to the bad net list mentioned above. It should be noted that as the learning rate decreases later on, the routing task itself tends to become slower. Therefore, as the iterations progress, we need to give them more time flexibility. In the experiment, we set this time threshold to 20 + 0.5$\times$iteration.

IV. Experimental Result

Our code is written in Python 3.10 and runs on an Intel(R) Core(TM) i5-11400 CPU with 24 GB RAM.

We have written a script for visualizing the routing problem, and an example visualized for fract2 is shown in Fig. 2.

The black numbers represent the cost of the grid cell at that location. The blue squares represent grid cells that serve as the source or target for the nets, and the white numbers on them indicate their net_id. In Fig. 2 (a), a partial zoom-in of the image has been done for better visibility of certain areas.

We have also written a script for visualizing the routing results, and an example visualized for fract2 is shown in Fig. 3. The blue grid cells have the same meaning as mentioned before, representing the source or target grid cells of the nets. The red grid cells and the white numbers above them represent the path taken by a specific net during routing. The green grid cells indicate locations where there are vias. In Fig. 3 (a), a partial zoom-in of the image has been done for better visibility of certain areas.

We further develop a script to evaluate our routing output results. This script assesses various aspects, such as whether all nets have been successfully routed, whether there are any path crossings, whether the routing paths are continuous, whether the starting and ending cells of the routing paths align with the input netlist, and overall cost calculation. An example evaluation result for fract2 is shown in Fig. 4.

```
>>>>>>>>>>Start evaluation: primary1 <<<<<<<<<<
All nets are routed
No overlapping points found
All paths are continuous
Start and end points match all Nets
Total costs: 111165.0
>>>>>>>>>>>>>>>Evaluation completed<<<<<<<<<<<<<<<
```

Fig. 4. Example evaluation result for fract2.

The performance of our algorithm on eight benchmarks are shown in Table 1.

TABLE I
THE PERFORMANCE OF OUR ALGORITHM ON EIGHT BENCHMARKS

Benchmark	Time(s)	Completion	Total Cost	Iteration
bench1	0.036	20/20	372	1
bench2	0.501	20/20	1760	1
bench3	0.983	16/16	469	1
bench4	0.734	15/15	1793	1
bench5	664.6	128/128	11458	18
fract2	1062	125/125	11050	18
primary1	554.4	830/830	111165	18
industry1	2155	1000/1000	430793	18

V. CONCLUSION

In this work, we propose an efficient router for 2-point maze routing. This router improves upon traditional maze DFS algorithms by incorporating attenuated cost estimation and iterative mechanisms, resulting in complete and low-cost routing solutions across eight benchmarks. The routing problem and its corresponding results are accompanied by visualization scripts to facilitate a visual inspection of the routing performance. The output router files also come with a evaluation script to ensure the integrity and reliability of the routing results.

REFERENCES

[1] Hao Geng. (2023). Lec11-Routing. Retrieved May 30, 2023, from ShanghaiTech University, School of Information Science and Technology. Web site: https://elearning.shanghaitech.edu.cn:8443/bbcswebdav/courses/EE215A-101848-4131-2022-20232/Lec11-Routing.pdf