EE215A Project: Router with Dijsktra & A*

Yifei Chen
2023233122

Yuxin Zhou
2023131094

***Abstract*—This project report delves into the comparative performance analysis of the A* and Dijkstra algorithms in the context of VLSI routing. Through a series of tests on various benchmarks, this study examines the speed, cost, and accuracy of both algorithms under conditions with and without preferred wiring directions. The A* algorithm demonstrates faster computation times in most cases, especially with a preferred direction, whereas Dijkstra's algorithm is more economical in terms of routing cost. Both algorithms achieve high accuracy, with A* slightly outperforming in some scenarios without a preferred direction. The results provide insights into the practical application of these algorithms in VLSI routing.**

Index Terms*—Router, Dijkstra, A

I. Introduction

In the realm of Very Large Scale Integration (VLSI) design, the efficiency and reliability of integrated circuits are highly influenced by the routing strategies employed. Routing, the process of establishing electrical connections between various components on a chip, is a critical step that determines the overall performance and manufacturability of electronic devices. As integrated circuits grow in complexity, sophisticated routing algorithms become increasingly essential to manage intricate design requirements effectively.

A key approach to VLSI routing is the maze routing algorithm, renowned for its ability to find optimal paths within grid-based layouts. This method is particularly advantageous for the dense and complex environments typical of VLSI designs. The maze routing algorithm explores all potential paths in a grid to find the most efficient route between two points, considering factors such as path length, signal integrity, and manufacturing constraints.

Effective routing is crucial for ensuring the functionality and performance of VLSI circuits. It impacts signal delay, power consumption, and the overall area of the chip. Through the optimization of routing processes, designers can achieve superior performance metrics and more reliable circuits.

This report will then contain the following sections. Section II will present the basic Dijkstra algorithm for implementing routers. Section III will present the a* algorithm, optimized on the original router. Section IV will present the results of the performance of the optimized router. Section V will provide a summary.

II. Algorithm Description

Dijkstra's algorithm is a classic algorithm used to solve the single-source shortest path problem in a weighted directed graph. It finds the shortest paths from a source node to all other nodes by iteratively expanding the node with the shortest distance.

The algorithm steps are as follows:

- Initialization: Set the distance from the source node to all other nodes to infinity, and the distance from the source node to itself to 0.
- Select the node with the shortest distance: Choose the node with the shortest distance from the set of unprocessed nodes. During this process, priority queue is employed to guarantee speed.
- Update distances: For each neighboring node of the selected node, if the distance to reach the neighbor through the selected node is shorter than the previously known distance, update the distance.
- Block the path: One the selected node has been processed, i.e., routed, the path it went through should be block so that no other nodes can cross the path.
- Repeat the above steps until all nodes have been processed.

Dijkstra's algorithm guarantees to find the shortest paths from a single source in a directed graph, but it encounters problems when negative weight edges are present.

III. A* Optimization

A* algorithm is a heuristic search algorithm commonly used to solve pathfinding problems in graphs. Similar to Dijkstra's algorithm, but A* introduces a heuristic function to estimate the distance from the current node to the goal node. This heuristic function helps the algorithm find the optimal path more quickly.

The algorithm steps are as follows:

- Initialization: Add the start node to the open set and set its cost from the start to 0.
- Choose the node with the minimum cost from the open set as the current node.
- If the current node is the goal node, the path is found, and the algorithm terminates.
- Otherwise, for each neighbor of the current node:
- Compute the actual cost from the start node to the neighbor.
    1) If the neighbor is not in the open set, add it and update its actual cost from the start node.
    2) If the neighbor is already in the open set and the newly computed actual cost is smaller than the previous cost, update the cost.
- Repeat the above steps until the goal node is found or the open set is empty.

A* algorithm guides the search direction by using a heuristic function when selecting the next expanded node, making it generally more efficient than Dijkstra's algorithm. However, A* requires selecting an appropriate heuristic function and does not guarantee the shortest path; it only guarantees a relatively optimal path.

A* Heuristic Function and Proof:

$\forall$ points A(x, y, l) in the grid, we need to compute the heuristic function which estimate the distance between this point and the end point $T(T_x, T_y, T_l)$, the heuristic function is defined as the following:

$$h(x,y,l) = Manhattan(A,T) + Via\ Penalty(\text{if } l \neq T_l) + Bend\ Penalty(\text{if } x \neq T_x \text{ or } y \neq T_y)$$

Consistency: the real cost for the path from (x, y, l) to end point $T(T_x, T_y, T_l)$ should be:

$$GridSum(x,y,l) + k \cdot Via\ Penalty + j \cdot Bend\ Penalty \geq h(x,y,l)$$

where, k, j ≥ 0

Admissible: for any points $A(x_a, y_a, l_a)$ and $B(x_b, y_b, l_b)$:

$$\begin{aligned} h(A) - h(B) &\leq Manhattan(A,T) - Manhattan(B,T) \\ &\quad + Via\ Penalty + Bend\ Penalty \\ &\leq Manhattan(A,B) + Via\ Penalty \\ &\quad + Bend\ Penalty \\ &\leq RealCost(A,B) \end{aligned}$$

As the heuristic function defines, via penalty and bend penalty only occurs at most once in the function, so if A,B are not in the same layer or do not have the same x or y, it will have extra penalty which have been included once. So our heuristic function is admissible.

IV. Results

We test the A* Algo and Dijkstra algorithms for time, cost, and accuracy by preferred direction and no preferred direction, respectively. The testbench are bench1, bench2, bench3, bench4, bench5, fract2, primary1 and industry1. In this paper, the results have been organized into the following TABLE I, TABLE II and TABLE III.

A. Time Analysis

For TABLE I, The A* algorithm is generally faster than the Dijkstra algorithm, particularly when handling large datasets. In scenarios with a preferred direction, the A* algorithm shows significantly faster computation times in most test benches, such as bench1, bench2, and bench4, compared to the Dijkstra algorithm. Similarly, in scenarios without a preferred direction, the A* algorithm maintains its speed advantage in most test benches, like bench1, bench2, and fract2. However, there are exceptions; for instance, in bench3, the A* algorithm is slower than the Dijkstra algorithm without a preferred direction. This indicates that while the A* algorithm usually provides higher time efficiency, there are specific cases where the Dijkstra algorithm might be more advantageous.

TABLE I
Testbench Time Results

	Preferred Direction		No Preferred Direction	
TestBench	A* Algo	Dijkstra	A* Algo	Dijkstra
bench1	34.3974 ms	41.6058 ms	19.0679 ms	38.2015 ms
bench2	84.9535 ms	112.117 ms	64.3081 ms	112.567 ms
bench3	82.7566 ms	88.538 ms	99.4953 ms	54.7018 ms
bench4	70.6776 ms	95.2052 ms	38.7188 ms	63.5986 ms
bench5	2778.33 ms	7064.81 ms	2906.51 ms	8639.31 ms
fract2	2511.28 ms	6145.23 ms	2738.32 ms	7301.02 ms
primary1	67944.5 ms	163504 ms	126485 ms	335450 ms
industy1	400951 ms	1.48e6 ms	688334 ms	1.7008e6 ms

B. Cost Analysis

For TABLE II, the Dijkstra algorithm generally performs better, especially without a preferred direction. With a preferred direction, the costs of the A* algorithm and the Dijkstra algorithm are very similar, being almost identical in some benchmarks (e.g., bench3, bench4). In some benchmarks, the A* algorithm even shows slightly lower costs than the Dijkstra algorithm (e.g., bench1, primary1). However, without a preferred direction, the Dijkstra algorithm demonstrates lower costs in most benchmarks, such as bench2, bench3, and fract2. The A* algorithm exhibits higher costs in some benchmarks, like bench5 and primary1. This highlights the Dijkstra algorithm's advantage in cost control.

TABLE II
Testbench Cost Results

	Preferred Direction		No Preferred Direction	
TestBench	A* Algo	Dijkstra	A* Algo	Dijkstra
bench1	352	372	352	372
bench2	2640	2493	2170	1822
bench3	2655	2655	1355	473
bench4	1807	1807	1793	1793
bench5	12132	12066	13626	11906
fract2	11890	11888	12829	10040
primary1	119583	119771	157788	131805
industy1	405951	406131	529458	574864

C. Accuracy Analysis

For TABLE III, both algorithms perform identically with a preferred direction, achieving 100% accuracy across all test benches. However, without a preferred direction, the A* algorithm shows slightly higher accuracy in some benchmarks, such as bench5, fract2, primary1, and industy1. Overall, the A* algorithm exhibits higher accuracy in certain specific scenarios without a preferred direction, while both algorithms are equally accurate with a preferred direction. This suggests that the A* algorithm can provide higher solution accuracy in some particular cases.

D. Testbench Analysis

The testbench are bench1, bench2, bench3, bench4, bench5, fract2, primary1 and industry1. We will next display the results visually, as shown in Fig 1 - Fig 8.

From Fig 1 - Fig 4, both the A* algorithm and the Dijkstra algorithm exhibit relatively uniform data point distributions

TABLE III
TESTBENCH ACCURACY RESULTS

	Preferred Direction		No Preferred Direction	
TestBench	A* Algo	Dijkstra	A* Algo	Dijkstra
bench1	20/20	20/20	20/20	20/20
bench2	20/20	20/20	20/20	20/20
bench3	16/16	16/16	16/16	16/16
bench4	15/15	15/15	15/15	15/15
bench5	128/128	128/128	128/128	116/128
fract2	125/125	125/125	124/125	111/125
primary1	830/830	830/830	797/830	687/830
industy1	1000/1000	1000/1000	993/1000	950/1000

across different layers, indicating a degree of stability and consistency, whether there is a preferred direction or not. With a preferred direction, the performance of both algorithms is similar, with data points distributed within the mid-range of the graph (0 to 40), and consistent results across both layers, indicating stable performance. Without a preferred direction, both algorithms also show roughly the same performance, although there might be more fluctuations, indicating differences in adaptability. Overall, the graphs suggest that the performance differences between the two algorithms under different conditions are minimal, though there may be more variability without a preferred direction.

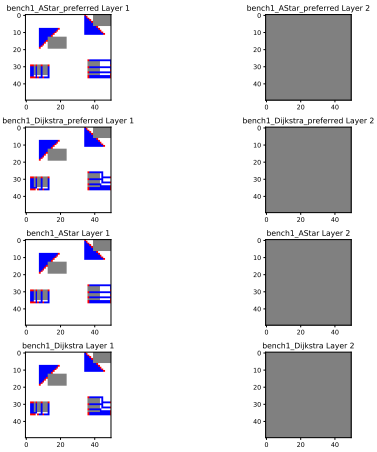

Fig. 1. Bench1 result

From Fig 5 - Fig 8, we can notice that:

The A* algorithm is an efficient pathfinding algorithm that uses a heuristic function to estimate the cost from the current node to the target node, thereby quickly finding a solution. In layout routing, the A* algorithm can optimize the path according to given preferences (for example, vertical wiring on the first layer, horizontal wiring on the second layer), which may give it an advantage over the Dijkstra algorithm when following specific wiring directions.

Dijkstra's algorithm is a deterministic shortest path algorithm that does not rely on heuristic information, thus always ensuring the shortest path is found. In layout routing, Dijkstra's algorithm may be more stable and reliable when there is no clear preference or when the problem size is large. However,

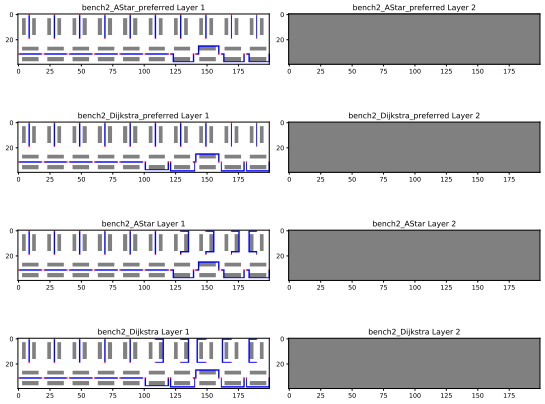

Fig. 2. Bench2 result

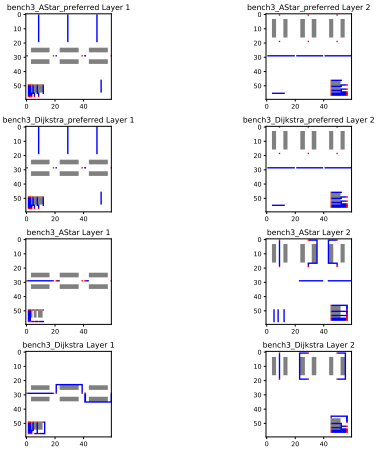

Fig. 3. Bench3 result

it may not be as fast as the A* algorithm, especially in cases where there is a clear direction preference.

In the case of specific preferred wiring direction, the heuristic nature of the A* algorithm may make it more efficient in the search process. If the heuristic function can well reflect this preference, the A* algorithm may find a better wiring path more quickly, thus surpassing Dijkstra's algorithm in performance.

V. CONCULSION

This project has provided an in-depth analysis of two prominent routing algorithms, A* and Dijkstra, within the context of Very Large Scale Integration (VLSI) design. The comparative study, conducted across multiple test benches, has shed light on the strengths and weaknesses of each algorithm when applied to routing tasks with and without preferred directions.

The A* algorithm generally offers faster processing speeds, especially when there is a preferred direction. However, in certain cases, such as the bench3 test bench, its speed advantage is not evident.Dijkstra's algorithm performs better

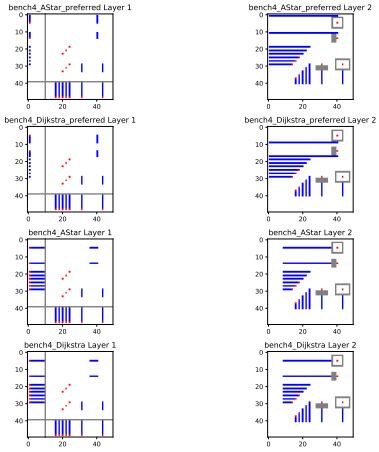

Fig. 4. Bench4 result

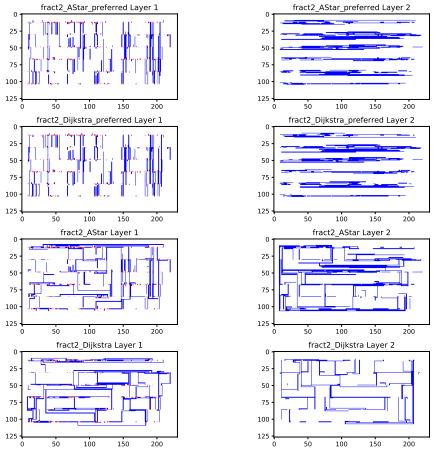

Fig. 6. Fract2 result

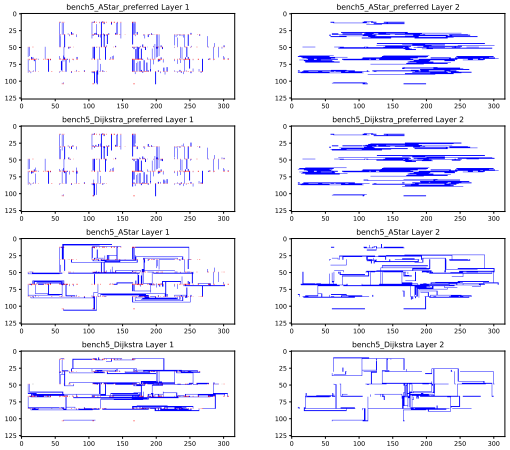

Fig. 5. Bench5 result

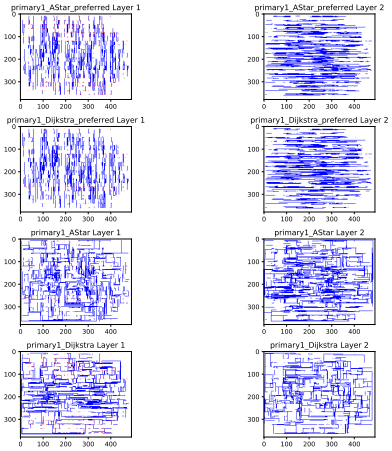

Fig. 7. Primary1 result

in terms of cost control, particularly in scenarios without a preferred direction. It demonstrates lower costs across most test benches.In terms of accuracy, both algorithms achieve a 100% accuracy rate when there is a preferred direction. Without a preferred direction, the A* algorithm performs slightly better in some specific cases.

The visual representation of the test benches further illustrated the performance patterns of both algorithms. With a preferred direction, both A* and Dijkstra algorithms showed stable and uniform data point distributions, indicative of their robustness. However, without a preferred direction, the variability in performance was more pronounced, suggesting a need for adaptive strategies to handle less structured routing problems.

In conclusion, the choice between A* and Dijkstra algorithms for routing in VLSI design depends on the specific requirements of the task at hand. If speed and accuracy are paramount and the routing environment has clear preferences, the A* algorithm is likely to be more effective. Conversely, if cost minimization is the primary goal, especially in less structured environments, Dijkstra's algorithm may be more suitable. Future work could focus on further optimizing the A* algorithm's heuristic functions and exploring hybrid approaches that combine the strengths of both algorithms to tackle an even broader range of routing challenges in VLSI design.

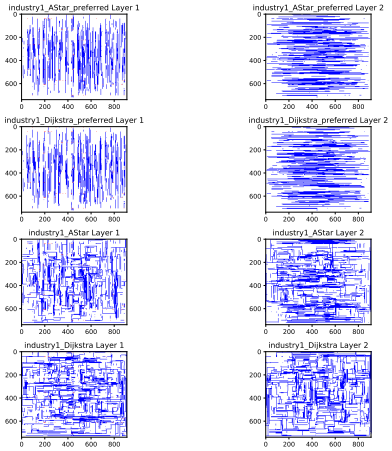

Fig. 8. Industry1 result