EE215A 2025 Project: Router with Dijsktra & A*

Shijie Meng
2022231101

Enci Tang
2024231046

Yining Jiang
2024134014

***Abstract*—Routing is a critical stage in the physical design of Very Large Scale Integration (VLSI) circuits, particularly for Application-Specific Integrated Circuits (ASICs), as it dictates circuit functionality, timing closure, and overall cost. Modern routing methodologies grapple with challenges such as complex routing spaces, multi-layer interconnects, and the pursuit of high-quality paths. This report presents a maze router that implements two selectable pathfinding algorithms: Dijkstra's and A*. Both algorithms adopt a layer-preferred routing strategy, assigning a single wiring direction per layer. For the A* algorithm, a Manhattan distance-based heuristic function is specifically designed to improve path quality and accelerate the search process. Experimental results demonstrate that the proposed router successfully completes routing for all benchmark circuits. Notably, our A* strategy implemented without preferred directions achieved the minimum routing cost and shortest runtime compared to the other strategies evaluated in this work.**

I. Introduction

In the domain of Very Large Scale Integration (VLSI) design, the efficiency and reliability of integrated circuits (ICs) are profoundly influenced by the employed routing strategies. Routing, the process of establishing electrical connections between components on a chip, is a pivotal step that dictates the overall performance and manufacturability of electronic devices. As IC complexity escalates, sophisticated routing algorithms become increasingly indispensable for effectively managing intricate design requirements.

A prominent approach in VLSI routing is the maze routing algorithm, renowned for its capability to identify optimal paths within grid-based layouts. This methodology is particularly advantageous for the dense and intricate environments typical of VLSI designs. Maze routing algorithms systematically explore potential pathways on a grid to determine the most efficient route between two points, considering factors such as path length, signal integrity, and manufacturing constraints. Effective routing is paramount to ensuring the functionality and performance of VLSI circuits. It directly impacts signal delay, power consumption, and the overall chip area. By optimizing the routing process, designers can achieve superior performance metrics and enhance circuit reliability.

Modern routers primarily face three key challenges. First, the **Complexity of the Routing Space**: Routing typically occurs on grids characterized by varying cell costs (non-unit costs) and obstacles (blocked cells), demanding that routers identify cost-effective paths while navigating around these blockages. Second, **Multi-Layer Routing and Vias**: Modern ICs utilize multiple layers for interconnection. Routers must manage both intra-layer connections and inter-layer transitions using vias, each incurring a 'via penalty'. Third, **Path Quality and Constraints**: Merely establishing connectivity is insufficient; the quality of the routed path is critical. A 'bend penalty' is introduced to discourage superfluous turns, which can adversely affect signal integrity and manufacturing yield. Furthermore, routers must optimize for the 'total path cost'.

To address these challenges, this work presents a maze router incorporating selectable Dijkstra's and A* pathfinding algorithms. The primary contributions of this research are as follows:

- The developed router offers the flexibility to employ either Dijkstra's or A* algorithm, both capable of identifying optimal routing solutions within complex routing terrains.
- A via penalty is integrated into the routing cost function to manage multi-layer routing requirements. Additionally, a layer-preferred routing strategy, assigning a single dominant direction per layer, is implemented to minimize intra-layer interference.
- A bend penalty is incorporated to minimize the number of turns, thereby discouraging unnecessary detours and enhancing overall path quality.

The remainder of this report is organized as follows. Section II elaborates on the fundamental Dijkstra's and A* algorithms utilized in the router, along with their characteristics. Section III details our algorithmic implementation. Section IV presents and analyzes the routing results. Finally, Section V concludes the report.

II. Algorithms Description

In this section, we first briefly introduce the Dijkstra algorithm. Second, we provide an overview of the A* algorithm. Third, we compare the key characteristics of both methods.

A. Dijkstra's Algorithm for Routing

Dijkstra's algorithm is a foundational algorithm in graph theory, primarily employed to solve the single-source shortest path problem for a weighted directed graph where edge weights are non-negative. In the context of VLSI routing, particularly maze routing, it systematically explores possible paths from a source node (pin) to a target node (pin) on a grid-based layout.

The operational steps of Dijkstra's algorithm, as adapted for routing, are as follows:

1) **Initialization:** The distance (cost) from the source node to itself is set to zero, while the distance to all other nodes is initialized to infinity. All nodes are initially marked as unvisited.

2) **Node Selection:** From the set of unvisited nodes, the node with the currently smallest known distance from the source is selected. A priority queue is often employed to efficiently manage this selection process.
3) **Distance Update:** For each neighbor of the selected node, the algorithm calculates the distance from the source to this neighbor via the selected node. If this newly calculated path is shorter than the previously known distance to the neighbor, the neighbor's distance is updated.
4) **Path Blocking:** Once a net is routed (i.e., the path for the selected node to its target is determined and finalized), the grid cells constituting this path are typically marked as obstacles or blocked to prevent subsequent nets from using the same resources, ensuring exclusivity.
5) **Iteration:** These steps are repeated, marking the selected node as visited, until the target node is reached or all reachable nodes have been processed.

Dijkstra's algorithm guarantees finding the absolute shortest path in terms of accumulated cost (e.g., path length, number of vias) from the source to all other nodes in a graph with non-negative edge weights. However, its uninformed search strategy, exploring equally in all directions, can lead to longer computation times for large search spaces compared to heuristic-guided algorithms.

B. A Algorithm for Optimized Routing*

The A* (A-star) algorithm is a heuristic search algorithm renowned for its efficiency in pathfinding problems, including VLSI routing. It enhances Dijkstra's algorithm by incorporating a heuristic function to guide the search process more intelligently towards the target node. The core idea is to prioritize nodes that are not only close to the start node (actual cost) but also estimated to be close to the goal node (heuristic cost).

The A* algorithm evaluates nodes based on the function $f(n) = g(n) + h(n)$, where:

- $g(n)$ is the actual accumulated cost from the start node to node n.
- $h(n)$ is the heuristic estimate of the cost from node n to the target node.

The general steps are:

1) **Initialization:** The start node is added to an "open set" (nodes to be evaluated), and its $g(n)$ value is set to 0.
2) **Node Selection:** The node in the open set with the lowest $f(n)$ value is selected as the current node.
3) **Goal Check:** If the current node is the target node, the path is found, and the algorithm terminates.
4) **Neighbor Expansion:** Otherwise, for each neighbor of the current node:
   - Calculate the tentative $g(n)$ value for the neighbor (cost from start to current + cost from current to neighbor).
   - If the neighbor is not in the open set, add it, calculate its $h(n)$, and set its $f(n)$.
   - If the neighbor is already in the open set and the new path to it (new $g(n)$) is shorter, update its $g(n)$ and $f(n)$ values.
5) **Iteration:** The current node is moved from the open set to a "closed set" (evaluated nodes), and the process repeats from step 2 until the target is found or the open set is empty.

The efficiency of A* heavily depends on the quality of the heuristic function $h(n)$. For A* to guarantee the shortest path, $h(n)$ must be *admissible*, meaning it never overestimates the actual cost to reach the target. The project report details a specific heuristic for VLSI routing:

$$h(x, y, l) = \text{Manhattan}(A, T) + \text{ViaPenalty}(\text{if } l \neq T_l). \quad (1)$$

This heuristic guidance generally allows A* to find paths more quickly than Dijkstra by exploring fewer nodes.

*C. Comparative Analysis of Dijkstra and A**

Both Dijkstra's and A* algorithms are fundamental to solving pathfinding problems in VLSI routing, but they exhibit different performance characteristics regarding speed, cost, and behavior under varying conditions, such as the presence or absence of preferred wiring directions. Table I provides a qualitative comparison.

TABLE I: Comparative Features of Dijkstra's and A* Algorithms

Feature	**Dijkstra's Algorithm**	**A* Algorithm**
Search Type	Uninformed	Informed
Basis	$g(n)$ (Actual cost)	$f(n)=g(n)+h(n)$ (Actual + Est.)
Guidance	No specific direction	Goal-oriented
Efficiency	Lower (more nodes explored)	Higher (fewer nodes if $h(n)$ good)
Optimality	Guaranteed (non-negative edges)	Guaranteed (if $h(n)$ admissible)
Primary Use	Single-source, all-nodes shortest	Specific start-to-goal shortest path

III. Router Implementation

In this section, we present the implementation details of our router. First, we provide an overview of the router architecture. Next, we describe the routing algorithms employed. Finally, we discuss our code implementation, including several optimization techniques.

A. Overview

Fig. 1 illustrates the overall architecture of our router. The router reads the input test files and routing strategy parameters, such as whether to use preferred-direction routing and the choice between Dijkstra's or A* algorithms. After routing is performed, the results are saved for further analysis.

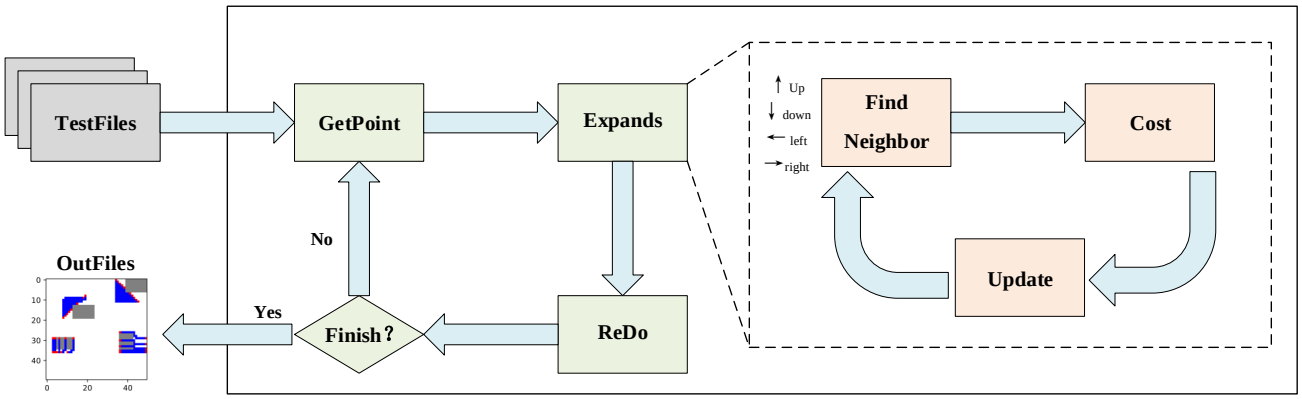

Fig. 1: The overview of our router.

B. Routing Algorithm Process Flow

Algorithm 1 presents our optimized circuit routing approach. The routing process operates through the following key steps:

1) Initialization Phase (Lines 1-4): The algorithm first extracts the start pin S and end pin E from the specified net index i (line 1). To ensure routability, these pins are temporarily unblocked in the grid (line 2), allowing connections even when pins are located in seemingly congested areas. A priority queue $open$ is initialized with the start pin, using either a heuristic distance estimate (for A* mode) or zero (for Dijkstra mode) as the initial cost function value (line 3). The algorithm also initializes a closed set to track visited nodes and a predecessor map for path reconstruction (line 4).

2) Search Phase (Lines 5-20): The main search loop processes nodes from the priority queue until either finding the target or exhausting all possibilities within the step limit. For each iteration:

- The node with minimum cost function value f is extracted from the queue (line 6).
- If the node has been previously visited or the step counter exceeds the limit, it is skipped (lines 7-9).
- The current node is marked as visited in the closed set (line 10).
- If the current node equals the target, the search terminates successfully (lines 11-13).
- Otherwise, the algorithm expands to neighboring nodes (lines 14-19):
  - Layer transition is explored by switching to the other layer at the same (x, y) coordinates, applying a via penalty (line 15).
  - Horizontal movements (left and right) are always explored, with bend penalties applied when direction changes occur (line 16).
  - If the strategy parameter $redo$ is set to 1, vertical movements (up and down) are also explored with appropriate penalties (lines 17-19).

3) Path Reconstruction Phase (Lines 21-26): If the search terminates without reaching the target, an empty path with zero cost is returned (lines 21-23). Otherwise, the algorithm constructs the complete path by backtracking through the predecessor map (line 24), restores the original grid values (line 25), and returns the path with its total accumulated cost (line 26). The algorithm adaptively handles circuit routing constraints through two key parameters: a global ASTAR flag that switches between Dijkstra and A* search strategies, and the $redo$ parameter that controls whether routing is restricted to preferred directions or allowed in all directions. This flexibility enables optimization for different design requirements and grid topologies while maintaining high completion rates.

Algorithm 1 Optimized Circuit Routing Algorithm

Require: NetGrid G, Net Index i, Strategy $redo$;
Ensure: Path P, Cost $cost$;
1: Extract start pin S and end pin E from net i;
2: Temporarily unblock S and E in grid;
3: Initialize priority queue $open$ with S, $f(S) = ASTAR?h(S, E) : 0$;
4: Initialize $closed$ map and $pred$ tracking map;
5: **while** $open$ not empty AND $steps < maxSTEPS$ **do**
6: $curr(g, l, x, y, f) \leftarrow open.pop()$; {Extract min f}
7: **if** $curr$ in $closed$ OR $steps$++ exceeds limit **then**
8: **continue**;
9: **end if**
10: $closed[x][y][l-1] \leftarrow true$;
11: **if** $curr = E$ **then**
12: **break**;
13: **end if**
14: {Expand neighbors with appropriate cost calculations}
15: Expand via layer: $(3-l, x, y)$ with via penalty;
16: Expand horizontal: $(l, x, y-1)$, $(l, x, y+1)$ with bend penalty if needed;
17: **if** $redo = 1$ **then**
18: Expand vertical: $(l, x-1, y)$, $(l, x+1, y)$ with bend penalty if needed;
19: **end if**
20: **end while**
21: **if** $curr \neq E$ **then**
22: **return** $\{\emptyset, 0\}$; {No path found}
23: **end if**
24: Construct path P by backtracking through $pred$;
25: Restore original grid values;
26: **return** $\{P, g$ of $curr\}$;

TABLE II: Performance Comparison of Routing Algorithms

Test File	#Nets	A* + Pref.		Dijkstra + Pref.		A*		Dijkstra	
		Cost	Time(ms)	Cost	Time(ms)	Cost	Time(ms)	Cost	Time(ms)
bench1	20 / 20	**372**	58.32	557	76.44	**372**	**30.97**	557	46.02
bench2	20 / 20	2,790	142.17	3,090	149.12	**2,490**	**106.83**	3,090	136.01
bench3	16 / 16	738	64.35	808	71.38	**523**	**43.08**	633	57.60
bench4	15 / 15	1,732	60.06	1,778	47.78	**1,688**	113.47	1,778	**43.76**
bench5	128 / 128	**9,022**	6,029.34	11,268	9,568.61	**9,022**	**3,033.17**	11,268	6,492.27
fract2	125 / 125	**8,841**	4,348.90	11,201	8,259.35	**8,841**	**2,298.24**	11,201	6,284.29
industry1	1,000 / 1000	376,577	878,165.00	469,669	2,854,980.00	**376,477**	**518,833.00**	469,569	2,372,030.00
primary1	830 / 830	96,121	180,010.00	116,011	307,568.00	**95,941**	**97,490.80**	115,831	220,956.00

C. Optimization Techniques

To improve both the success rate and efficiency of routing, as well as to prevent excessive path lengths, we employ three key optimization strategies:

- **Rerouting Strategy:** If routing fails under the preferred-direction strategy, we automatically switch to a non-preferred-direction scheme to enhance the success rate.
- **Priority Queue Optimization:** We utilize a priority queue to ensure that paths with the lowest local cost are explored first. This is implemented using C++'s built-in priority queue.
- **Step Limit Constraint:** A maximum step limit is imposed to avoid excessively long paths and infinite loops during the routing process.

1) Subsubsection Heading Here: Subsubsection text here.

IV. Experiment results and Analysis

A. Experimental Setup

*1) **Platform**:* Our router is implemented in C++ and evaluated on a machine running Ubuntu 20.04, equipped with an Intel® Core™ i9-10920X CPU @ 3.50 GHz and 16 GB of RAM.

*2) **Benchmarks**:* All benchmarks provided in the course are utilized to comprehensively evaluate the performance and effectiveness of our router.

*3) **Evaluation Metrics**:* We employ two metrics to evaluate router performance:

1) **Total Routing Cost:** The sum of costs after routing each benchmark. A lower cost indicates higher routing quality.
2) **Total Routing Time:** The overall runtime required to complete routing on each benchmark. Shorter time reflects higher routing efficiency.

B. Results Comparison

The comparative results of our routing strategies are presented in Table II, where the bolded values indicate the minimum cost and time achieved for each benchmark. As shown in the table, all our routing strategies successfully completed routing for all benchmarks. Notably, the A* algorithm without preferred direction achieved both the lowest routing cost and the shortest runtime across all benchmarks.

C. Visualization and Analysis

The routing visualizations for each strategy and benchmark are shown in Fig. 2-9. It should be noted that for several single-layer benchmarks (Fig. 2 and Fig. 3), the second-layer images appear blank. We analyze the results from Table II in conjunction with the visualizations as follows:

1) **Dijkstra vs. A*:** For single-layer circuits (e.g., Fig. 2), the Dijkstra algorithm produces routing paths with a large number of turns, while the A* algorithm minimizes turns. Although Dijkstra achieves shorter total wire length, its cost increases due to excessive bend penalties. In two-layer circuits (e.g., Fig. 6), Dijkstra tends to generate unnecessary vias and longer detours, leading to higher overall cost. This is primarily because Dijkstra does not sufficiently account for bend and via penalties during search, unlike A*. Moreover, the lack of a clear search direction results in longer routing time compared to A*.
2) **Preferred vs. Non-Preferred Direction:** The preferred direction and non-preferred direction strategies yield similar routing costs. However, routing time for the preferred direction strategy is significantly higher. Visualization shows that the preferred direction strategy does not consistently produce unidirectional routes within the same layer (e.g., Fig. 8), resembling the results of the non-preferred direction strategy. Analysis reveals that the preferred direction strategy fails some times, requiring fallback to the non-preferred direction, which leads to similar costs but increased overall routing time.

V. Conclusion

In this report, we presented the design and implementation of a flexible maze router for VLSI physical design, capable of operating with either Dijkstra's or the A* search algorithm. By incorporating cost functions that penalize vias and bends, our router was designed to address the challenges of multi-layer routing and path quality.

The experimental results successfully validated the effectiveness of our approach, as the router achieved 100% routing completion on all benchmark circuits. Future work could focus on extending the router's capabilities to address more advanced effects, such as signal crosstalk and timing-driven routing.

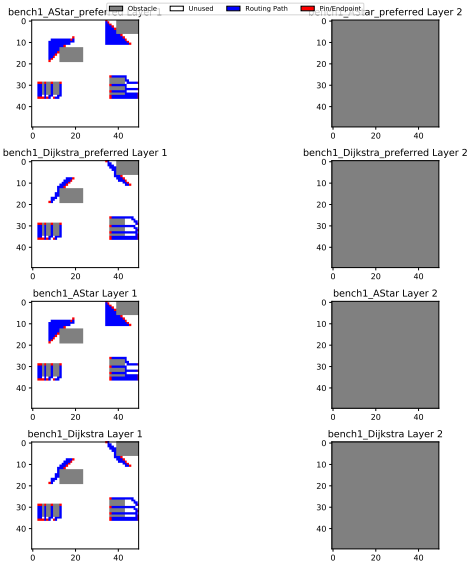

Fig. 2: Routing visualizations of bench1.

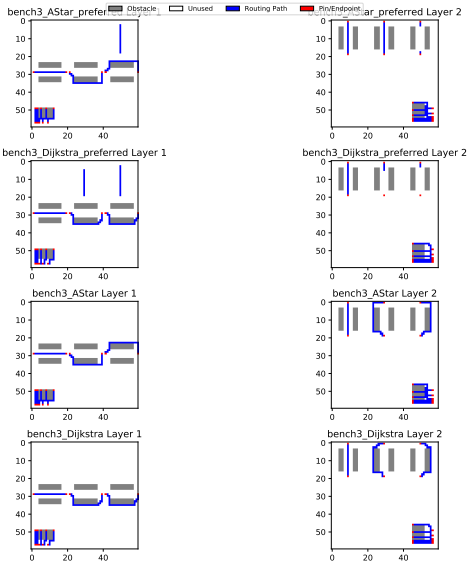

Fig. 4: Routing visualizations of bench3.

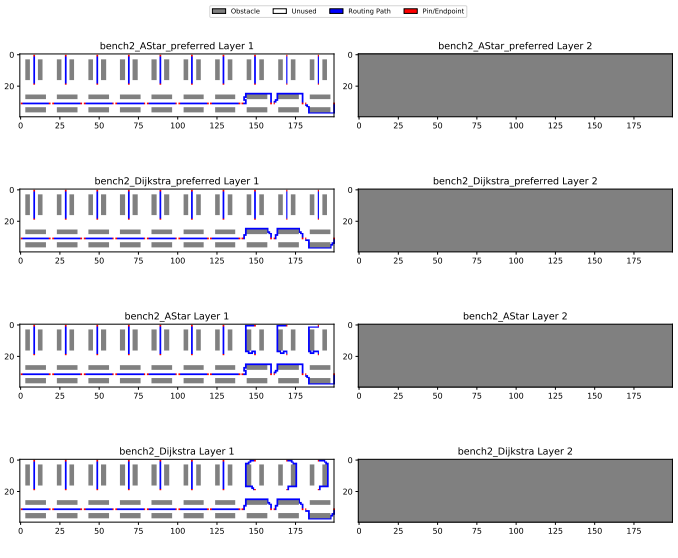

Fig. 3: Routing visualizations of bench2.

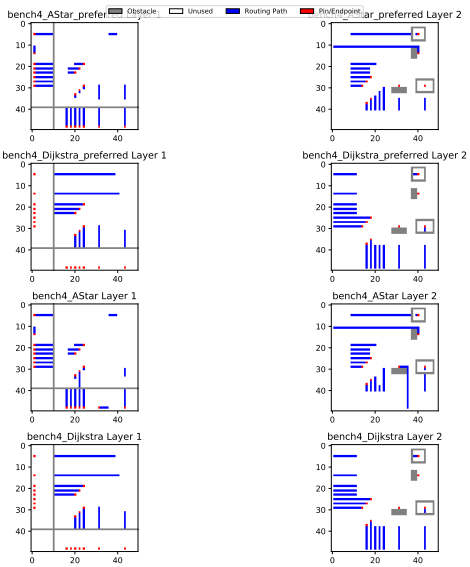

Fig. 5: Routing visualizations of bench4.

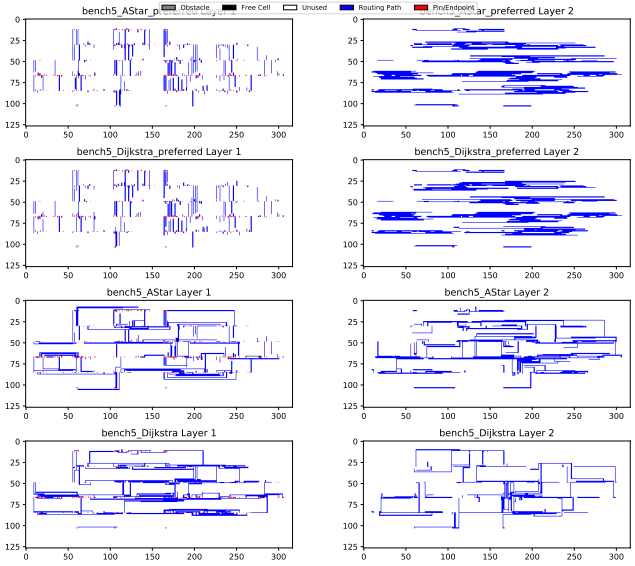

Fig. 6: Routing visualizations of bench5.

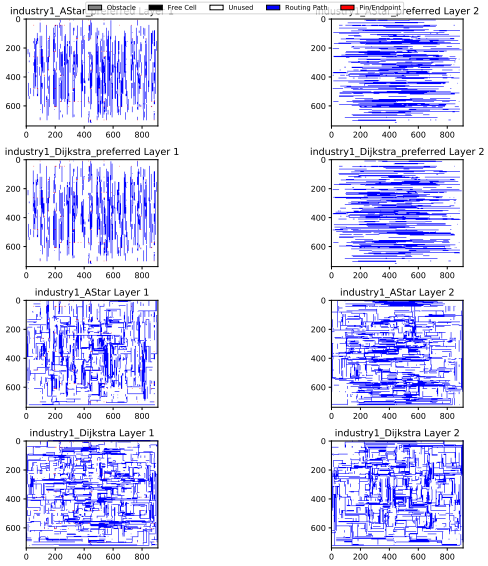

Fig. 8: Routing visualizations of industry1.

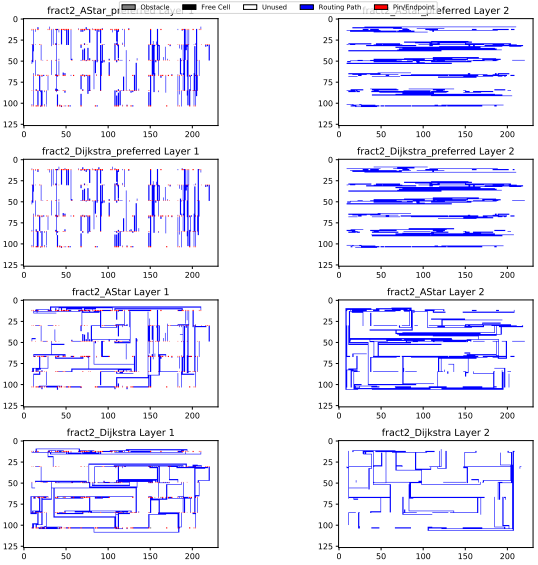

Fig. 7: Routing visualizations of fract2.

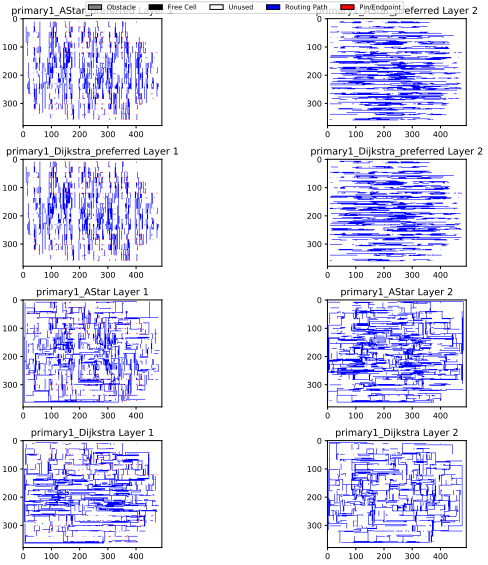

Fig. 9: Routing visualizations of primary1.