Maze Router Project: Technical Overview & Analysis

...

Technical Background

What is Maze Routing?

- Maze routing is a fundamental problem in VLSI (Very Large Scale Integration) physical design.
- The goal: Connect multiple pins (terminals) on a grid, avoiding obstacles and minimizing cost (e.g., wire length, vias, bends).

Key Concepts

- Grid-based representation: The chip or board is modeled as a 2D or 3D grid.
- Layers: Multiple metal layers (e.g., M1, M2) allow wires to cross without shorting.
- Obstacles: Represent existing wires, blocks, or forbidden regions.
- Pins: Start and end points for each net (connection).
- Costs: Penalties for vias (layer changes), bends (wrong direction), and wire length.

Design and Implementation

High-Level Architecture

- Input Parsing: Reads grid size, penalties, obstacles, and nets from a file.
- Grid Model: Represents the routing area, obstacles, and pins.
- Routing Algorithm: Uses a variant of the Lee algorithm (Dijkstra/A* style) for pathfinding.
- Output Generation: Writes routed paths to a file and generates visualizations.

Main Components

1. `MazeRouterInput` (parser.py)

- Parses the input file.
- Stores grid dimensions, penalties, obstacles, and nets.

2. `Point` (router.py)

- Represents a location in the grid (layer, x, y).
- Supports hashing and comparison for efficient use in sets and priority queues.

3. `Grid` (router.py)

- Manages the grid, obstacles, and validity checks.

4. `PathFinder` (router.py)

- Implements the Lee maze routing algorithm.
- Considers via and wrong direction penalties.
- Avoids obstacles and other nets' pins.
- Orchestrates the routing of all nets.
- Allows user to override penalties via command-line arguments.

Main Components

5. `MazeRouter` (router.py)

- Orchestrates the routing of all nets.
- Allows user to override penalties via command-line arguments.

6. `main.py`

- Handles command-line interface.
- Runs the routing process and outputs results.

Find_path Implementation

The find_path method in the PathFinder class implements a cost-aware shortest path search using a priority queue (min-heap) to efficiently explore the grid. Starting from the source point, it repeatedly selects the lowest-cost node to expand, tracking the cumulative cost and parent for each visited node. For each neighbor, it checks if the move is valid (not blocked by obstacles or other nets' pins) and calculates the move cost, which includes penalties for vias and wrong-direction moves. If a neighbor can be reached with a lower cost than previously recorded, it updates the cost and parent, and adds the neighbor to the heap. When the destination is reached, it reconstructs the path by backtracking through the parent pointers. If no path is found, it returns None. This approach ensures the algorithm finds the minimum-cost path while respecting all routing constraints.

Design Choices

Class-based design: Improves modularity, readability, and maintainability.

- Penalty customization: Users can set via and wrong direction penalties via CLI.
- Layer preference: M1 prefers horizontal, M2 prefers vertical routing.
- Visualization: Generates images for routed nets

Example

Example Input (input.txt)

text

Apply to parser.py

```
8,8,1,1

net1 (1, 0, 0) (2, 7, 7)

OBS (1, 4, 4)

net2 (2, 0, 7) (1, 7, 0)

...
```

Example

Example Command

bash

Apply to parser.py

Run

```
python3 main.py input.txt output.txt --via-penalty 5 --wrong-direction-penalty
3
```

Example Output (output.txt)

text

Apply to parser.py

```
net1 (1, 0, 0) (1, 1, 0) ... (2, 7, 7)
```

Example

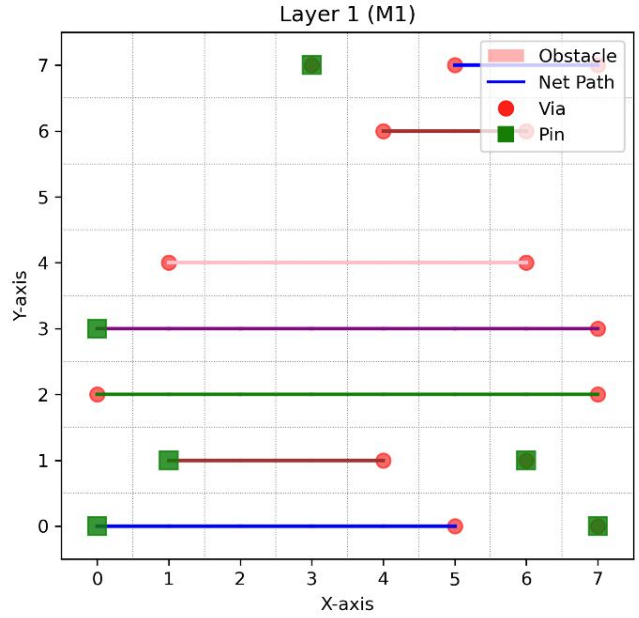
Example Output (output.txt)

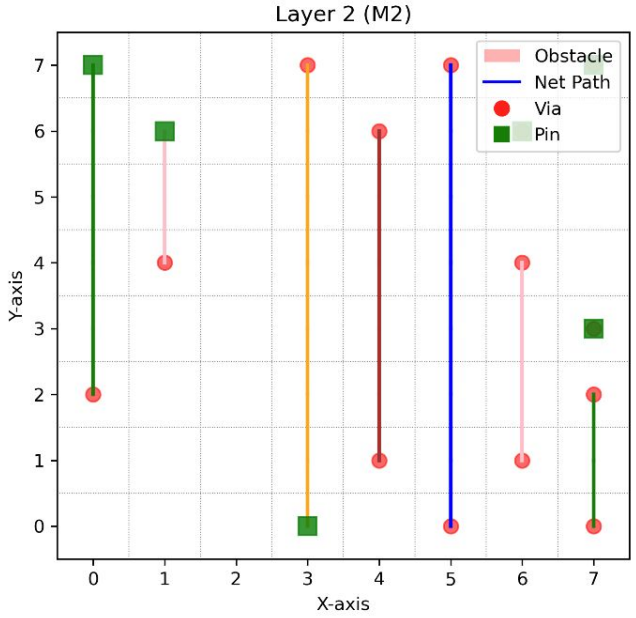
text

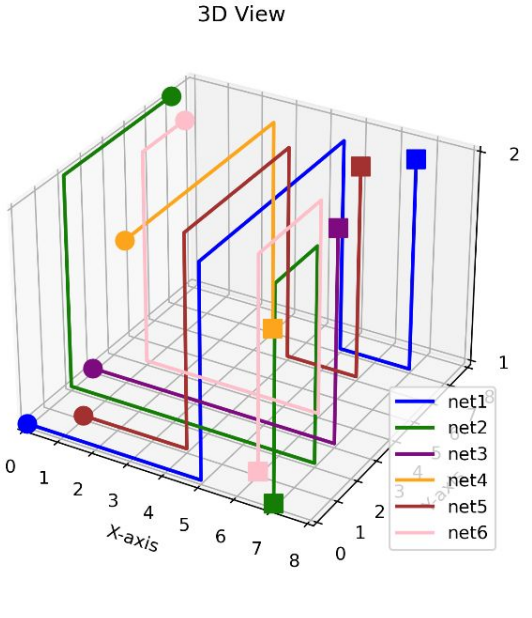
Apply to parser.py

```
net1 (1, 0, 0) (1, 1, 0) ... (2, 7, 7)

net2 (2, 0, 7) ... (1, 7, 0)

...
```

Visualization







Problems & Limitations

Algorithmic Limitations

- Scalability: The Lee algorithm is exhaustive and can be slow for large grids or many nets.
- No Net Ordering Optimization: Nets are routed in input order; no prioritization or rip-up-and-reroute.
- No Congestion Handling: If a net cannot be routed due to blockage, it is skipped.
- No Multi-terminal Net Optimization: Pins are connected sequentially, not as a Steiner tree.

Conclusions

- Demonstrates grid-based routing and OOP design
- Extensible for new features/algorithms
- Usable via CLI and visualizations
- Future work: scalability, congestion, advanced features