DIGITAL DESIGN II

IC MAZE ROUTER

By: Amal Fouda, Nour Selim, Omar Anwar

Project Ideas

Goal: To develop a routing algorithm for integrated circuits (ICs) that efficiently connects nets while minimizing routing costs and avoiding obstacles

Implement A* algorithm to route, and rip and reroute to route nets that did not successfully find their path

Use Python to visualise the output

Why A* Algorithm?

Uses a best-first search strategy, which allows it to explore the most promising paths first

Employs a heuristic function (Manhattan) to estimate the cost from the start pin to the target pin

```python
def manhattan_distance(self, pos1, pos2):
    """Calculate Manhattan distance between two positions"""
    return abs(pos1[1] - pos2[1]) + abs(pos1[2] - pos2[2])
```

Easier to modify to accommodate for the different types of costs that are accompanied with IC routing

Grid.py

01 – Representing the grid

- Define a Grid class that represents a 2D grid for routing in integrated circuits (ICs). Initializes the grid dimensions (width and height) and creates two layers (M0 and M1) for routing.

02 – Obstacle management

- Add obstacles to the grid, which are represented as cells that cannot be traversed

03 – Move Validation

- Implement the is_valid_move method to check if a proposed move to a specific cell is valid

04 – Get Neighbours

- get_neighbors method identifies valid neighboring positions for a given cell, prioritizing preferred movement directions based on the layer
- Calculate movement costs, including penalties for direction changes (bends) and via movements between layers

Router.py

MazeRouter Class

- Initializes the router with an input file containing grid dimensions, obstacles, and net definitions
- Parses the input file to set up the grid and initialize the list of nets

Net Parsing

- parse_input method reads and interprets the input file, extracting grid parameters, obstacles, and net information
- Creates Net and Pin objects for each net, storing their respective pins for routing

Routing Algorithm

- route_all_nets methodattempts to route all nets using rip-up and reroute.
- Utilises the A* algorithm to find optimal paths for each net while considering obstacles and routing costs

Neighbour Retrieval

- Uses the get_neighbors method from the Grid class to find valid neighboring positions for routing, considering preferred movement directions and costs

Path Management

- Provides methods to manage the routing paths of nets, including clearing routes and marking paths on the grid
- Ensures that the routing process respects the constraints of the grid and the presence of obstacles

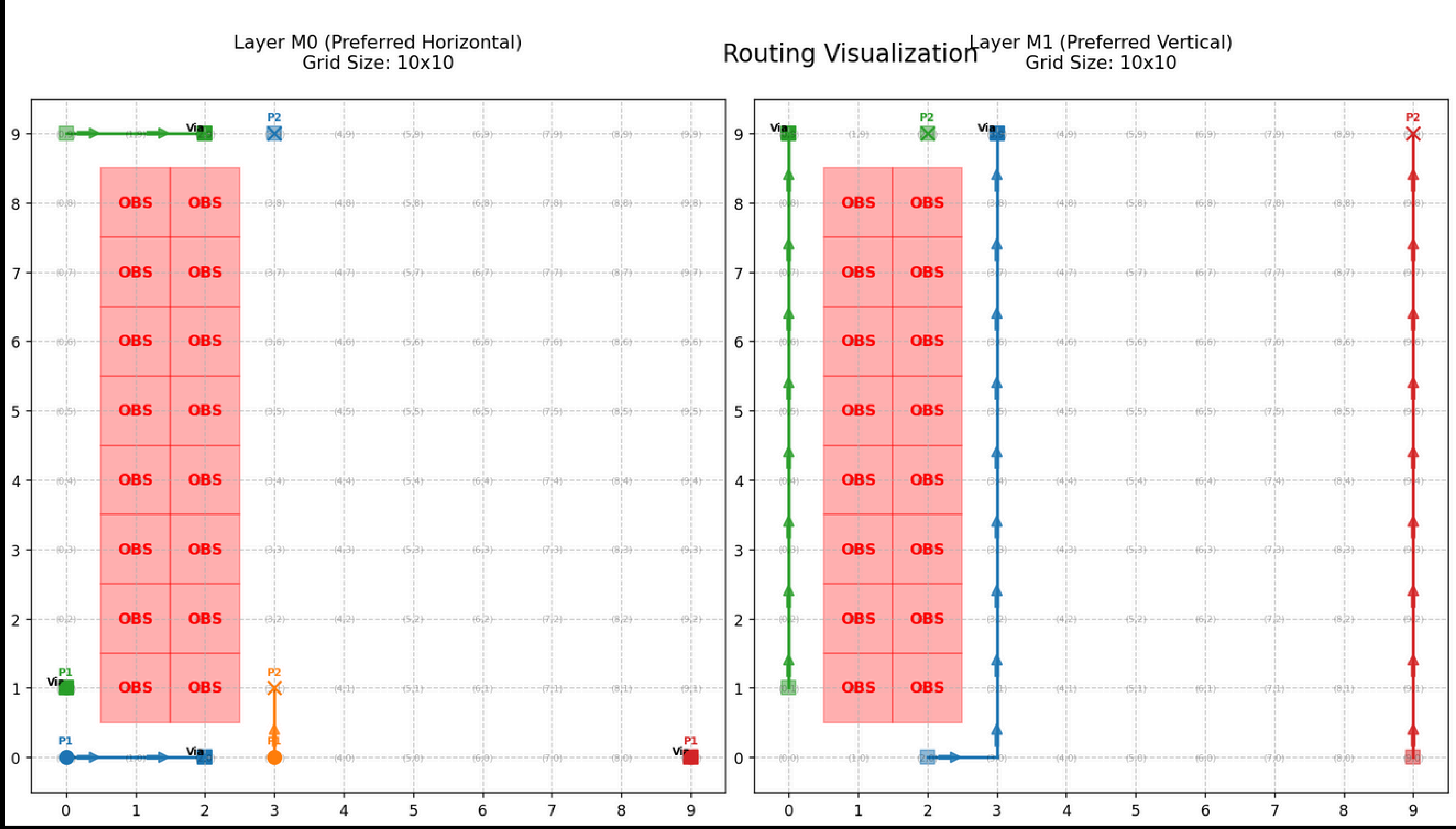
Rip and Reroute, Bounding Box

explained further in the next slide

Rip and Reroute Implementation:

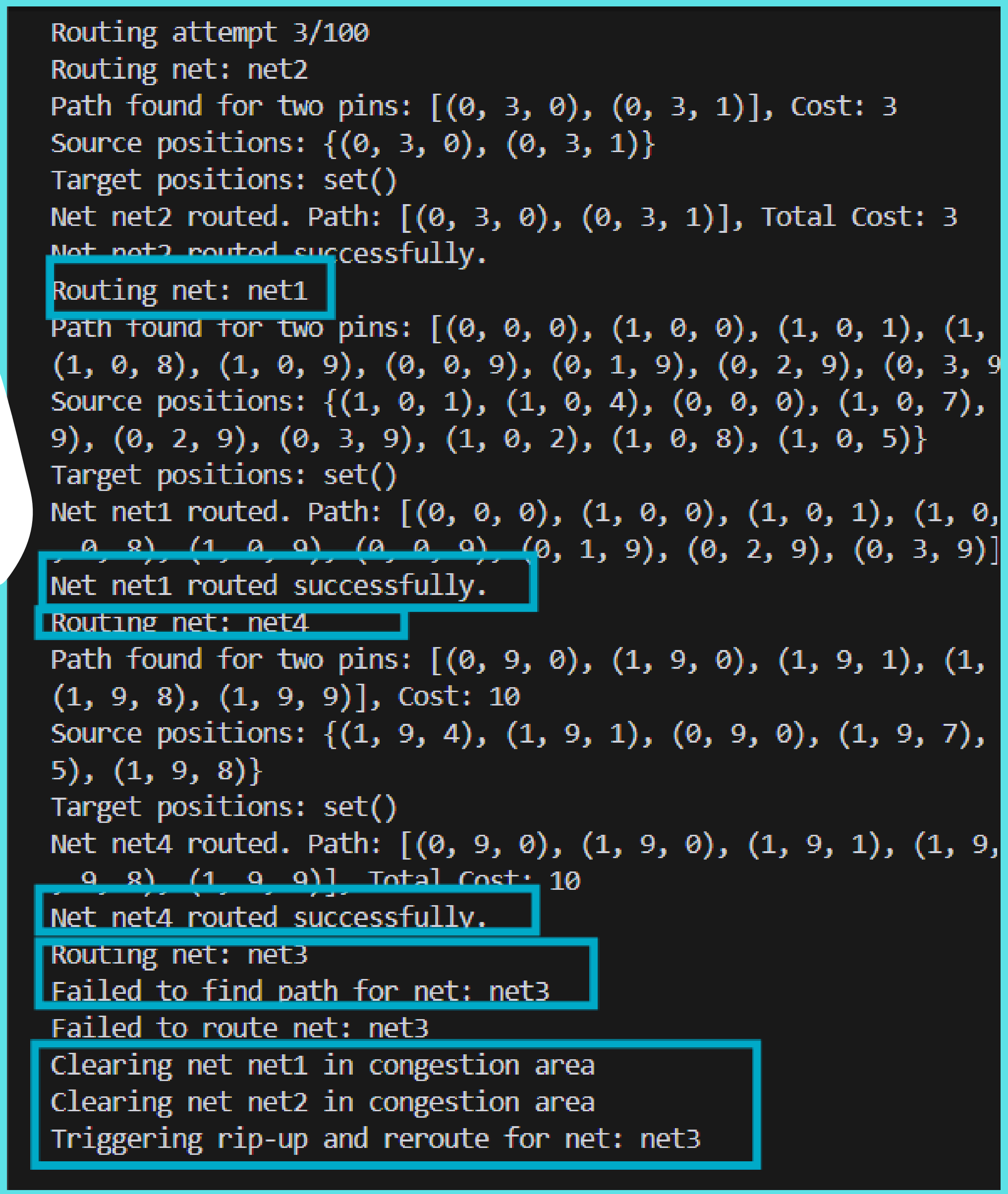
- The route_all_nets function attempts to route all nets for a specified number of attempts (100)
- For the **first** attempt, it clears any previous routes for each net and the grid
- For each net, the algorithm attempts to route it using the **route_net** function. If routing fails, it triggers the R&R process
- When a net fails to route, the bounding box around its pins is calculated to identify the area of congestion
- After clearing the affected nets, the algorithm can attempt to reroute the failed net in the next iteration.

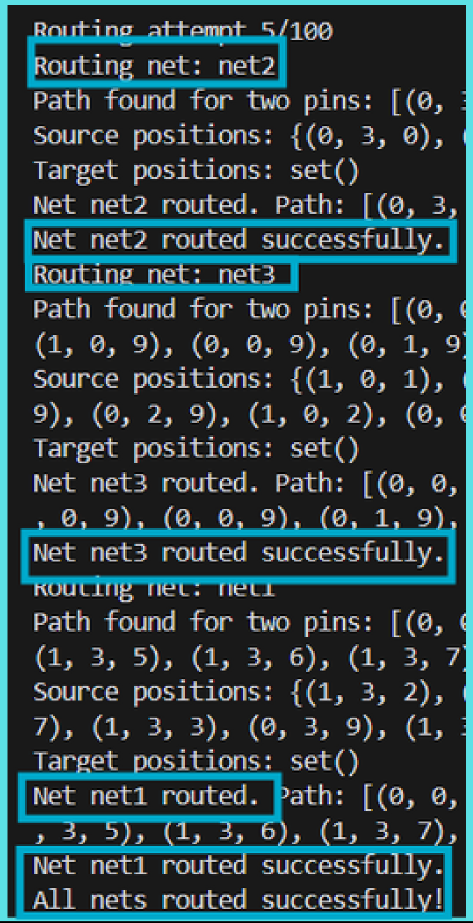
Test Case:



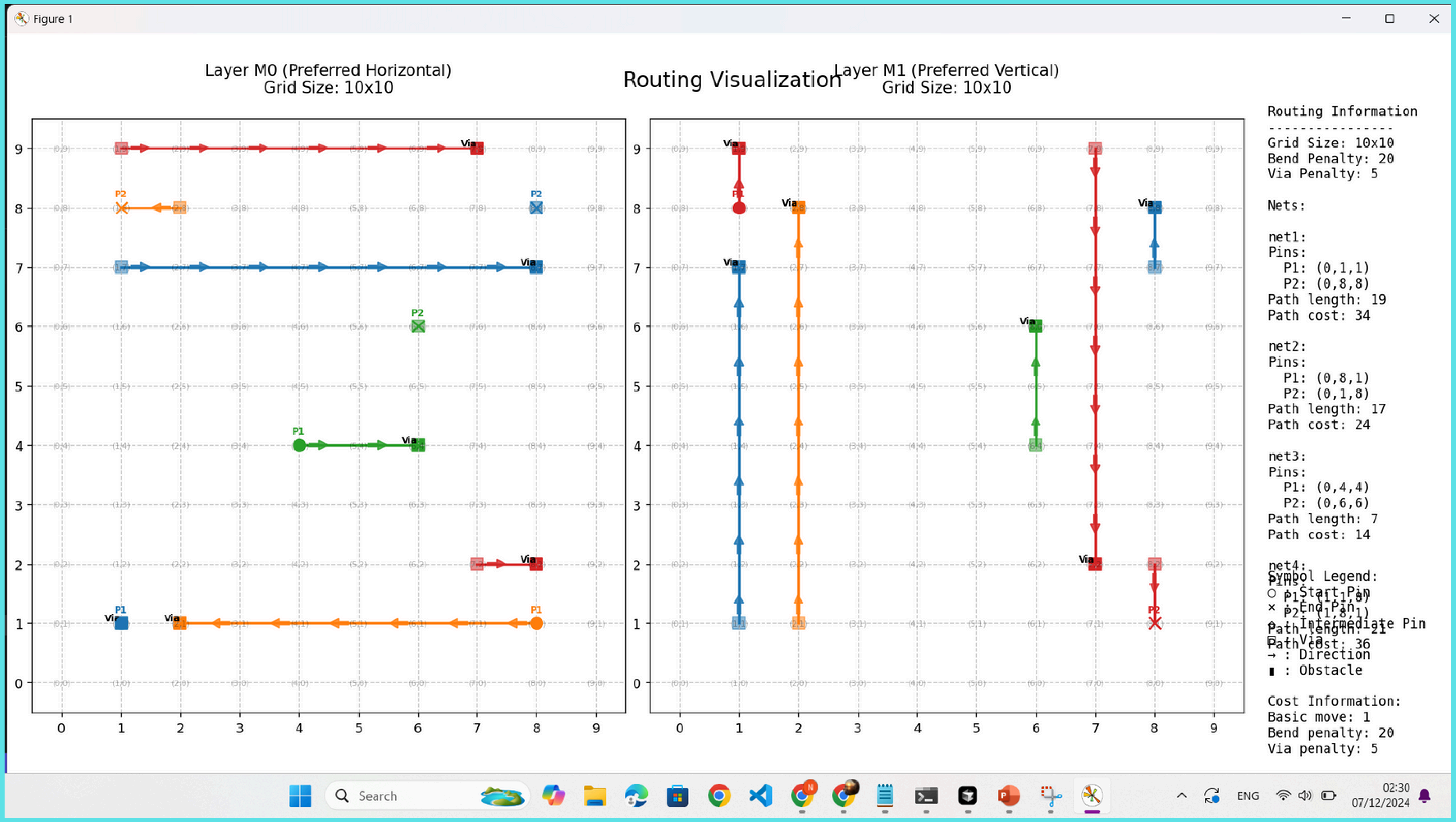
Rip and Reroute Output:

- Given that nets 1,2,3 are congested, and net 4 does not interfere with connecting net 1, 2, and 3, we expect the function to clear only the nets that cause congestion
- In the next attempt to reroute, we expect the router to not reroute net 4. Which is demonstrated correctly.





Test Case #1

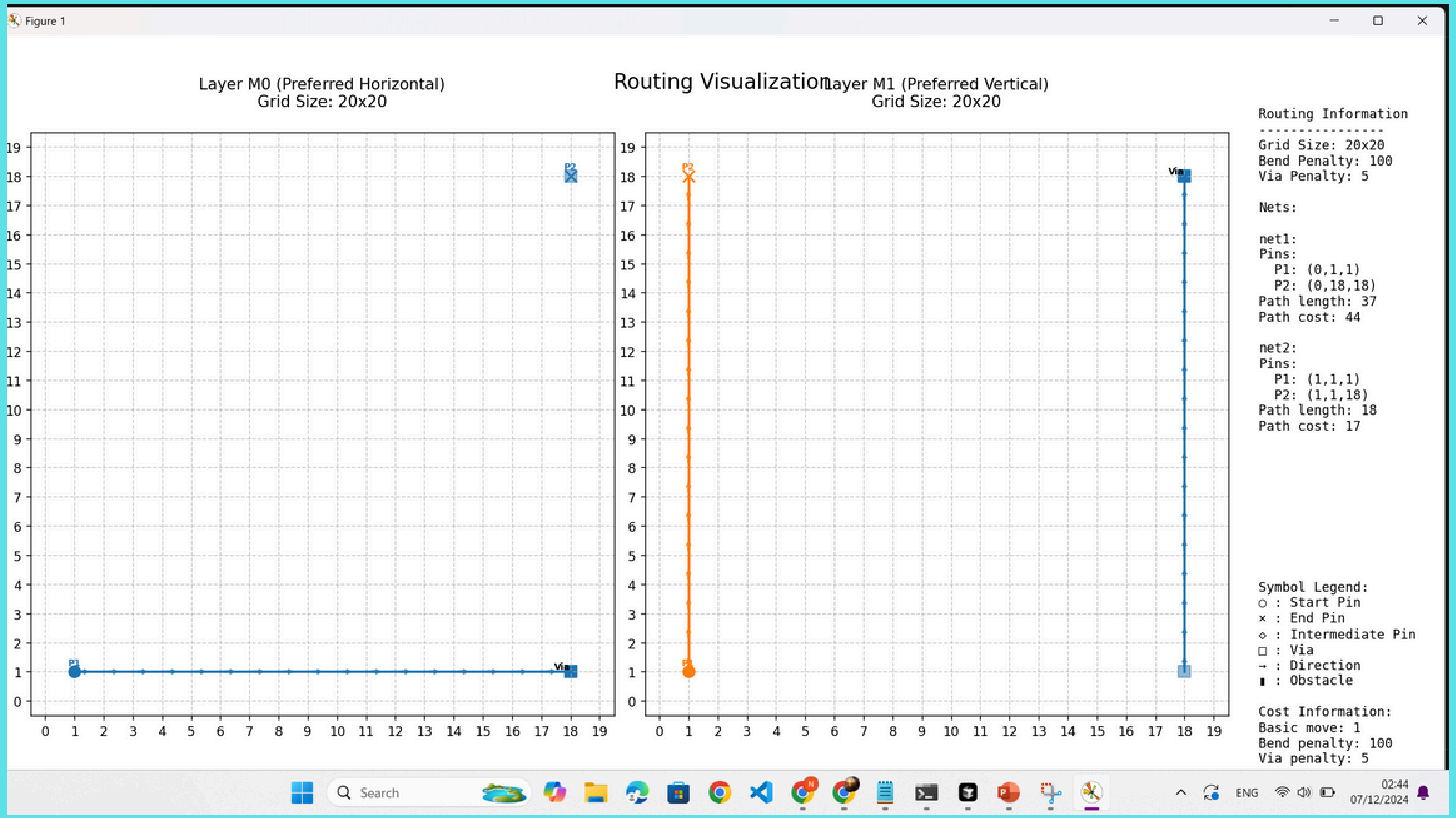


src > test_cases > case5_high_density.txt

```
10,10,20,5
net1 (0,1,1) (0,8,8)
net2 (0,8,1) (0,1,8)
net3 (0,4,4) (0,6,6)
net4 (1,1,8) (1,8,1)
```

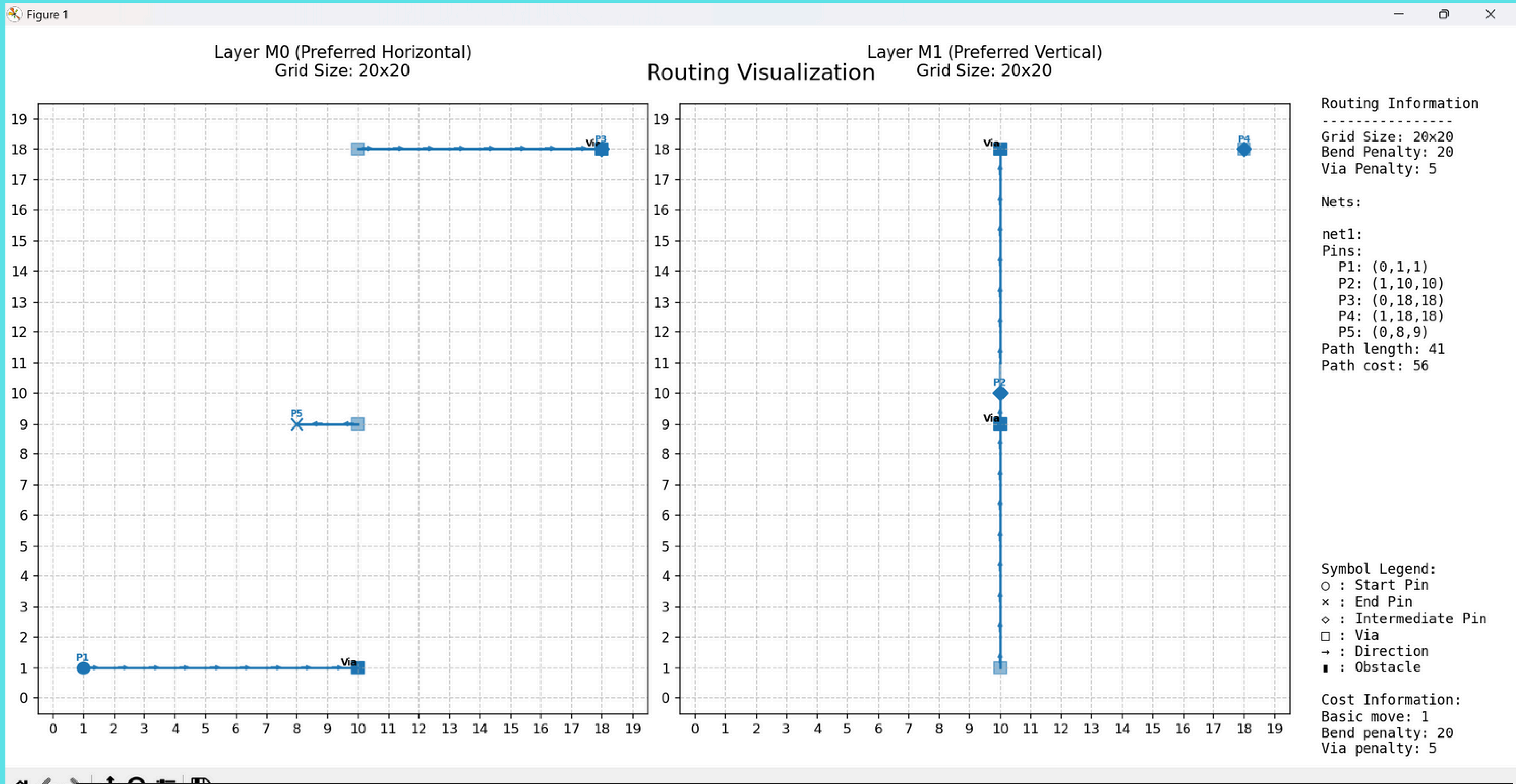
Test Case #2


```
est_cases > case4_layer_preference.txt
20,20,0,5
net1 (0,1,1) (0,18,18)
net2 (1,1,1) (1,1,18)
```




```
test_cases > case4_layer_preference.txt
20,20,100,5
net1 (0,1,1) (0,18,18)
net2 (1,1,1) (1,1,18)
```

Test Case #3



src > test_cases > case2_multipin.txt

```
20,20,20,5
net1 (0,1,1) (1,10,10) (0,18,18) (1,18,18) (0,8,9)
```

Thank You