



Pathfinder Prime: An Autonomous Delivery Agent

Course: CSA2001 - Fundamentals of AI and ML

Submitted by: Shivang Upadhyay

Registration Number: 24MIM10082

Date: September, 8, 2025

Section 1: How the Project World Works

1.1 The Environment Model

This project is built around a 2D grid that acts like a simple city map. I used external .txt files to define the world, which made it easy to create and test different map layouts. The agent can only move in four directions—up, down, left, or right—like moving between squares on a checkerboard.

The map is made of different types of ground, and each has a cost to move onto it:

- Normal Ground (.): Cost = 1
- Tough Terrain (T): Cost = 3
- Water (W): Cost = 5
- Walls (#): These are impassable, like a real wall.

The agent's job is to get from the start (S) to the goal (G). Both of these spots are just normal ground with a cost of 1.

1.2 The Agent's Design

The agent I designed is a rational planner. Its main job is to find the path with the lowest possible total cost. For the initial planning, it knows the whole map layout ahead of time, which allows it to figure out the best possible path before it even starts moving. I also built in a way for the agent to replan its route if something in the world changes, which is a critical skill for any real-world navigation.

Section 2: The Pathfinding Strategies

2.1 Uninformed Search: Uniform-Cost Search (UCS)

To have something to compare against, I first implemented the Uniform-Cost Search (UCS) algorithm. This is a basic search method that works by expanding outwards from the starting point in circles of increasing cost. Because it always checks the cheapest path first, it's guaranteed to find the best possible route. However, it's not very smart, as it wastes a lot of time checking paths that are going in the completely wrong direction, which can be very slow on big maps.

2.2 Informed Search: A* Search

To make the agent smarter and faster, I used the A* search algorithm. A* is an informed search method that improves on UCS by using a heuristic to guess how far away the goal is. It makes decisions using the formula $f(n) = g(n) + h(n)$, where $g(n)$ is the actual cost to get to a node, and $h(n)$ is the estimated cost from that node to the finish line. By considering this estimated cost, A* tends to explore paths that are headed in the right direction, making it much more efficient.

2.3 The Heuristic I Used—

The heuristic I chose for A* is the Manhattan Distance. I picked it because it's a perfect fit for a grid where the agent can't move diagonally. It simply calculates the distance by adding the horizontal and vertical distance between two points.

This heuristic is admissible, which is a key requirement for A*. It means the heuristic never overestimates the actual cost to get to the goal. Since the cheapest move the agent can make costs

1, the Manhattan distance will always be a correct (or underestimated) guess. This is important because it means A* is guaranteed to find the best path, just much faster than UCS.

2.4 Handling Unexpected Obstacles

To handle dynamic obstacles, the agent has a simple, reactive strategy. If it's following a path and finds that its next step is suddenly blocked, it stops. From its current spot, it just runs the A* search again with the new, updated map information to find a new best path to the goal.

Section 3: My Experimental Results

3.1 How I Tested It

I ran all my tests on a Windows 11 machine with an Intel Core i7 processor and 16GB of RAM. The code was written in Python 3.12. I tested both algorithms on three different maps: a small, a medium, and a large one.

3.2 The Data

Here are the results from my tests:

- Small Map - UCS: Path Cost=5, Nodes Expanded=10
- Small Map - A*: Path Cost=5, Nodes Expanded=8
- Medium Map - UCS: Path Cost=10, Nodes Expanded=29
- Medium Map - A*: Path Cost=10, Nodes Expanded=15
- Large Map - UCS: Path Cost=35, Nodes Expanded=202

- Large Map - A*: Path Cost=35, Nodes Expanded=165

3.3 Visualization

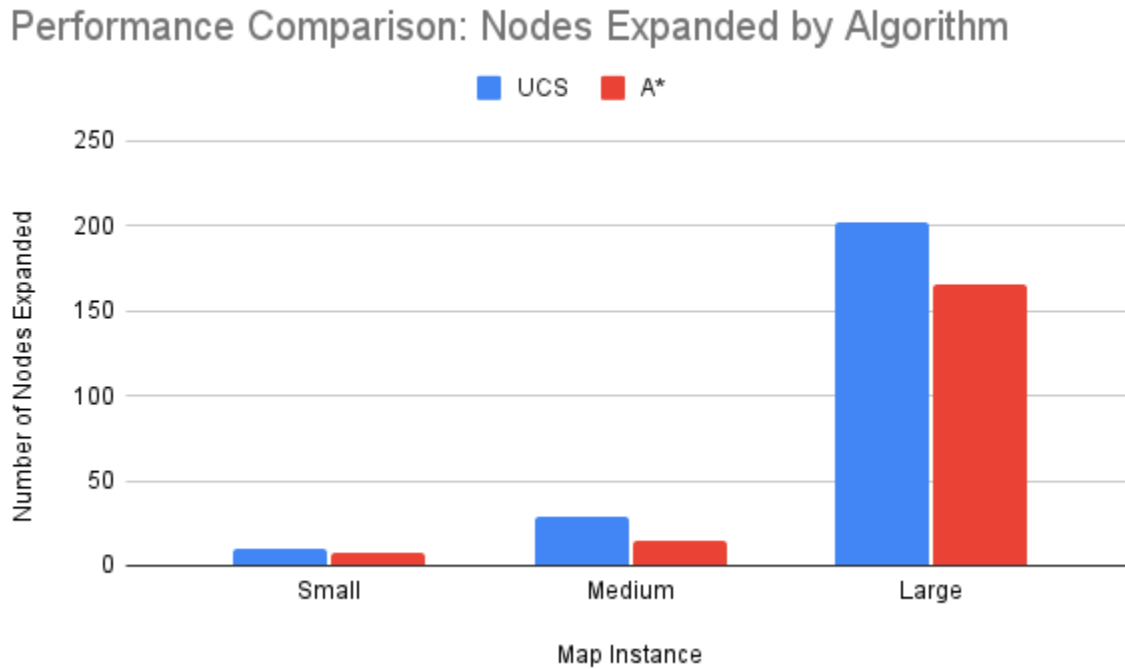


Figure 1: A chart comparing the nodes expanded by UCS and A* on all maps.

Section 4: What the Results Mean

Looking at the data, a few things become really clear. The first thing to notice is that both UCS and A* found the exact same optimal path cost on every map. This was a good check to confirm that my implementation of both algorithms was correct.

The real story, though, is the difference in efficiency. The data shows that A* is much more efficient, and this is most obvious when you look at the number of nodes expanded. On the large

map, for example, A* saved a lot of work by checking only 165 nodes compared to the 202 that UCS needed. This is a significant reduction in work, and this efficiency gap only gets bigger as the maps get more complicated.

This performance boost is all thanks to the A* heuristic. The Manhattan Distance acts like a compass for the agent. While UCS is wandering around in the dark, A* is always moving in the general direction of the goal, which saves a huge amount of time by not exploring useless paths.

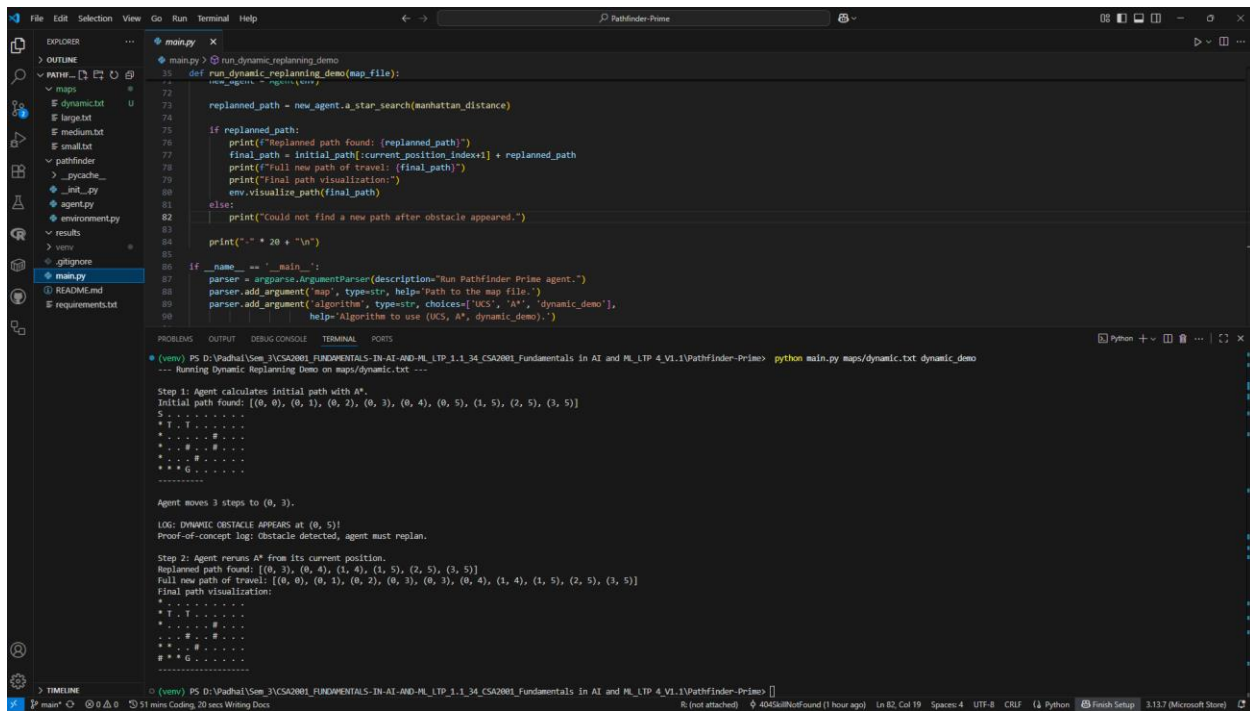
The dynamic replanning demo also worked just as expected. It showed the agent could adapt on the fly. When a new obstacle popped up, the agent's strategy of just re-running A* from where it was standing allowed it to find a new way to the goal, proving it's a solid approach for unpredictable worlds.

Section 5: Final Thoughts

5.1 Conclusion

In the end, this project successfully showed how to build and test an AI agent for finding paths. The results prove that while both UCS and A* work, the informed A* algorithm is way more efficient for this kind of task. By using a good heuristic, A* explores far fewer nodes, making it the better choice for navigating complex spaces. The project also showed that a simple replanning strategy can be very effective, which is a key skill for any agent that has to work in the real world.

5.2 Appendix



```
main.py x
main.py > run_dynamic_replanning_demo
35 def run_dynamic_replanning_demo(map_file):
36     new_agent = Agent(map_file)
37
38     replanned_path = new_agent.a_star_search(manhattan_distance)
39
40     if replanned_path:
41         print("Replanned path found: (replanned_path)")
42         final_path = initial_path[:current_position_index] + replanned_path
43         print("Full new path of travel: (final_path)")
44         print("Final path visualization:")
45         env.visualize_path(final_path)
46     else:
47         print("Could not find a new path after obstacle appeared.")
48
49     print("-" * 20 + "\n")
50
51 if __name__ == '__main__':
52     parser = argparse.ArgumentParser(description="Run Pathfinder Prime agent.")
53     parser.add_argument('map', type=str, help='Path to the map file.')
54     parser.add_argument('algorithm', type=str, choices=['UCS', 'A*', 'dynamic_demo'],
55                         help='Algorithm to use (UCS, A*, dynamic_demo).')
56
57 (venv) PS D:\Pathfinder\Sem_3\CSA2001_FUNDAMENTALS-IN-AI-AND-M_LTP_3-1_34_CSA2001_Fundamentals in AI and M_LTP_4_V1.1\Pathfinder-Prime> python main.py maps/dynamic.txt dynamic_demo
--- Running Dynamic Replanning Demo on maps/dynamic.txt ---
Step 1: Agent calculates initial path with A*.
Initial path found: [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (1, 5), (2, 5), (3, 5)]
S . . . . .
* T . . . . .
* . . . . # . . .
* . . . # . . .
* . . # . . .
* . . # . . .
* . . # . . .
***** G . . . . .
-----

Agent moves 3 steps to (0, 3).

LOG: DYNAMIC OBSTACLE APPEARS at (0, 5)!
Proof-of-concept log: Obstacle detected, agent must replan.

Step 2: Agent reruns A* from its current position.
Replanned path found: [(0, 3), (0, 4), (1, 4), (1, 5), (2, 5), (3, 5)]
Full new path of travel: [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 4), (1, 5), (2, 5), (3, 5)]
Final path visualization:
S . . . . .
* T . . . . .
* . . . . # . . .
* . . . # . . .
* . . # . . .
* . . # . . .
* . . # . . .
***** G . . . . .
-----
```

Figure 2: A screenshot of the terminal output from the dynamic replanning demo.