Assignment 1: Path Finding

Dan Blanchette

May 23, 2025

1 Introduction

In Artificial Intelligence (AI), a search problem refers to a framework for finding a solution among many possible options by systematically exploring them. It's foundational in fields like pathfinding, planning, game playing, and problem solving.

This type of approach consists of five elements:

1. Initial State: Where the agent(AI algorithm) starts.
2. Goal State: The objective for the algorithm is to complete, resulting in a successful outcome.
3. Actions: Possible operations/choices the agent can take from each state.
4. Transition Model: A description of a resulting state after an action is performed to move to a given state.
5. Path Cost: While not a part of the Breadth First Search approach or Depth First Search, Lowest Cost, Greedy Best First, and A* pathfinding use a scoring method. Scoring provides a more informed decision-making process at the Transition Model step by approximating distances to the goal and determining, based on a minimum or maximum metric, which node is best to move to next.

Subsequent sections will discuss the algorithms used and their performance.

2 Algorithm Analysis and Performance

Each analysis in this section will use the following legend to denote visited, unvisited, start, and end nodes. The symbology used to indicate the chosen path based on each algorithm is also shown.

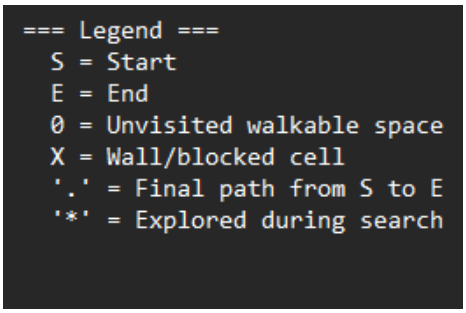


Figure 1: Legend

2.1 Breadth First Search

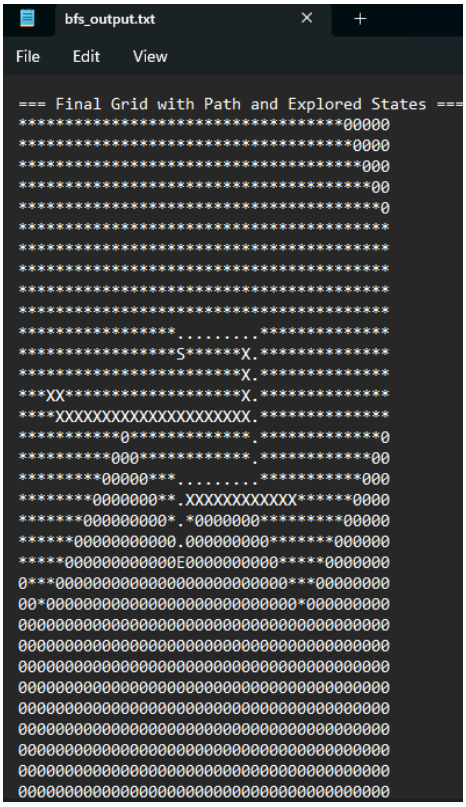


Figure 2: Breadth First Search Grid

Algorithm	Path Length	Cost	Closed List	Open List
BFS	29	n/a	733	39

Table 1: Breadth First Search Results

As observed, the breadth-first search(BFS) found a path that was 29 nodes (start point inclusive). This algorithm does not use scoring, as all nodes are treated as having a score of one, no matter the distance. The brute force algorithm is very apparent, as 733 nodes were "tried" before the solution was found, with 39 additional nodes to try on the open list. Breadth-first search by these accounts is not an optimal solution. It continually grows in all directions due to its horizontal sweeping method, which can be expensive both computationally and in memory resource management.

2.2 Lowest Cost (Dijkstra)

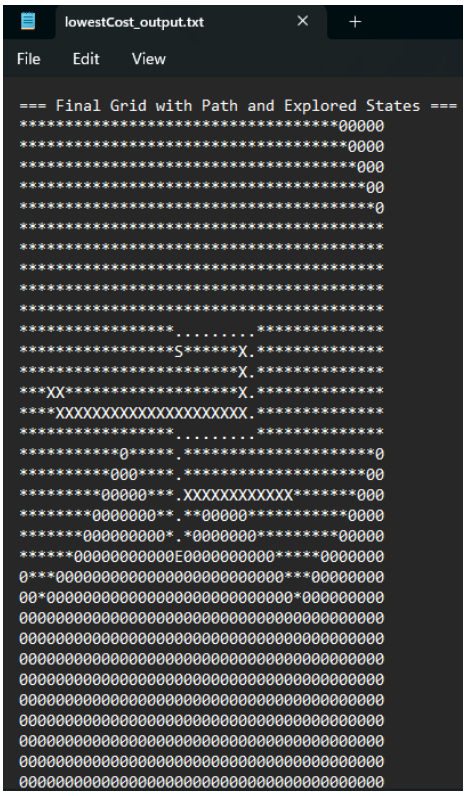


Figure 3: Lowest Cost Search Grid

Algorithm	Path Length	Cost	Closed List	Open List
Lowest Cost	29	28	755	41

Table 2: Lowest Cost Search Results

The lowest-cost algorithm did worse than BFS in terms of the number of explored nodes required to find the path. This outcome is because the scoring is weighted as 1,2,3, etc., based on the distance away from the goal. Although it did select the one-cost path more often, the cost was that this algorithm had to explore more possibilities to find other nodes with similar values.

2.3 Greedy: Best First

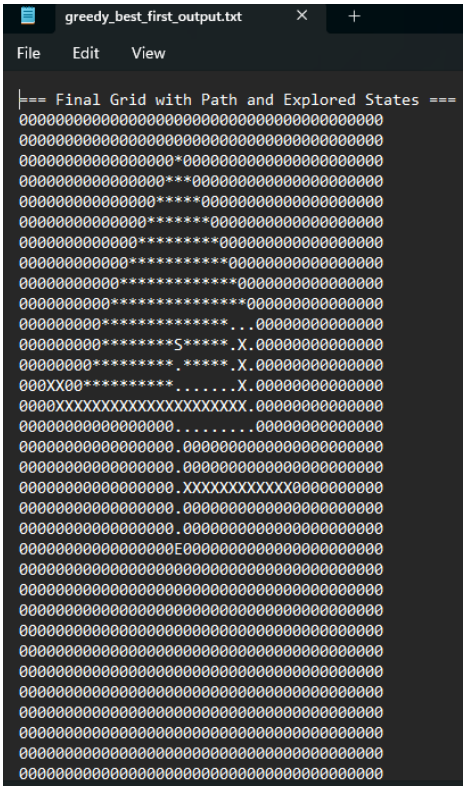


Figure 4: Greedy: Best First Grid

Algorithm	Path Length	Cost	Closed List	Open List
Greedy Best First	33	32	148	63

Table 3: Greedy: Best First Search Results

At each step, Greedy: Best First picks the tile (node) closest to the goal using Manhattan's (chosen for this implementation) distance, regardless of how far the agent has already walked. It does not consider walls or weights in the heuristic, just straight-line grid distance. If the heuristic keeps pointing that way, it may continue on expensive or long routes. That is observed in the output, where the path length is longer than the other algorithms thus far. The trade-off is that it had the fewest nodes to consider for its solution. However, the open list of nodes to be considered is still larger than all the algorithms that were tested.

2.4 A*: Manhattan Heuristic

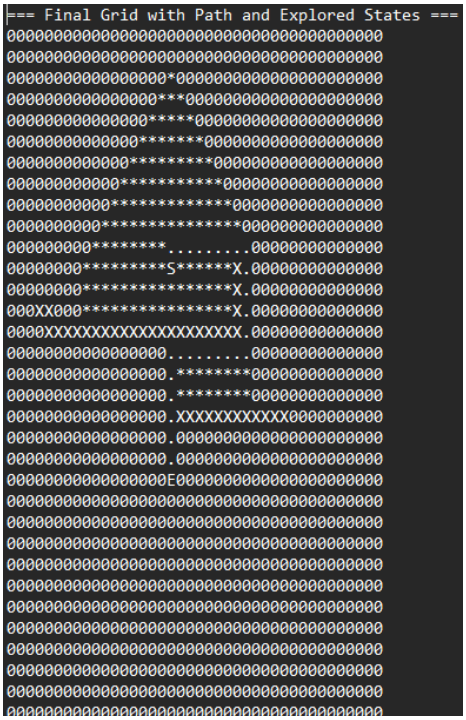


Figure 5: A*: Manhattan Heuristic Grid

Algorithm	Path Length	Cost	Closed List	Open List
A* (Manhattan)	29	28	164	37

Table 4: A* : Manhattan Heuristic Search Results

A* Manhattan heuristic is perfect for grid maps without diagonal movement. It's fast, accurate, and admissible — meaning it never overestimates the distance to the goal. A* with Manhattan is usually better than BFS (which ignores weights), Lowest Cost (which can ignore cost patterns while trying to get to the goal), and Greedy (which ignores path cost). The output did slightly worse by having more nodes visited on the closed list than Greedy: BF, but the visual data does not lie. It has a smaller footprint and fewer entries on the open list than the greedy approach.

2.5 A* : Euclidean Heuristic

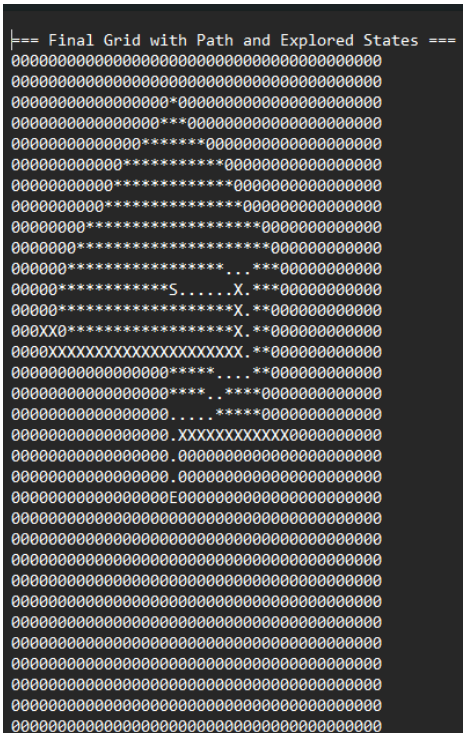


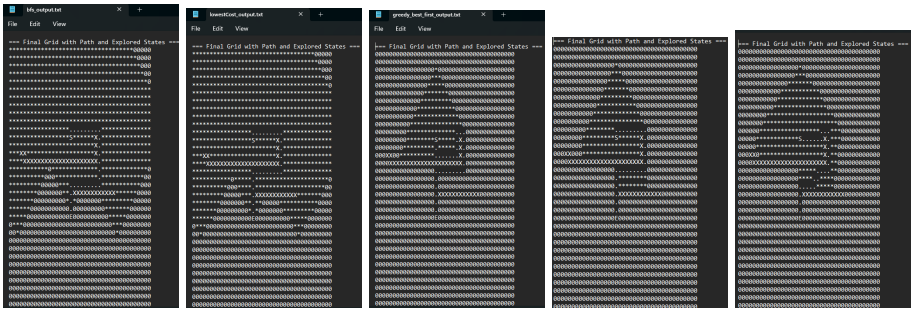
Figure 6: A* Euclidian Grid

Algorithm	Path Length	Cost	Closed List	Open List
A* (Euclidean)	29	28	217	43

Table 5: A* : Euclidean Heuristic Search Results

The A* Euclidean algorithm's cost agreed with most of the other methods used so far. While this algorithm was more efficient than BFS/Dijkstra, with only 217 explored nodes, it was less efficient than A* Manhattan's 164 nodes. The Euclidean heuristic was more computationally expensive per node. Euclidean uses square roots to accommodate diagonal movement, which is slower than simple additions using A* Manhattan. In certain circumstances, it may perform better, such as having a gap in the walls to find an optimal path, but, in this case, diagonality provided information that influenced the agent to try paths in the opposite direction and explore further out compared to A* Manhattan.

3 Algorithm Performance Summary and Comparison



(a) BFS (b) Lowest Cost (c) Greedy: BF (d) A* Man (e) A* Euclidean

Figure 7: Visualization Comparison

Algorithm	Path Length	Cost	Closed List	Open List
BFS	29	28	733	39
Lowest Cost	29	28	755	41
Greedy Best First	33	32	148	63
A* (Manhattan)	29	28	164	37
A* (Euclidean)	29	28	217	43

Table 6: Summary Search Results

BFS, Dijkstra, A* Manhattan, A* Euclidean all found a 29-step path with total cost 28 They are complete and optimal, assuming consistent heuristics and uniform costs. Greedy: BF visited the fewest nodes before finding the optimal solution on the closed list. However, it was the least efficient memory-wise due to the remaining nodes on its open list that still needed to be popped off.

Overall, A* with Manhattan heuristic is the best algorithm for this grid-based pathfinding map challenge. It delivers the optimal path with the least work, better than BFS, Lowest Cost, and Greedy in both quality and efficiency. For this case, it worked well due to the wall placement as an obstacle. Numerically, it managed its memory well with the fewest nodes in the queue to pop and as the second-best performer for nodes visited before finding a solution.

4 AI Tools: ChatGPT40 Utilization and Review

For this project, I had ChatGPT40 help me with a coding outline for the breadth-first search method. I'm currently specializing in industrial automation, SCADA(System Control Analysis and Data Acquisition), and PLCs, so I needed a refresher on my Python coding, including classes in Python. It also explained some key steps for the other algorithms when I was confused by the logic I found from different sources like GeeksforGeeks, StackOverflow, etc.

I learned that even with GTP40 at my disposal, prompting the LLM is a job within itself. Sometimes, it provides a method, then contradicts itself in an explanation. Code generation was about 50:50, and what the heck am I reading? Sometimes, GPT40 had a reference to the algorithm for the solution it was providing, and other times, it was clear that it was making things up. I bought the AI a Modern Approach book, and between the lectures, the pseudocode from the book examples, and some online references, as mentioned before, I was able to piece things together.

It definitely went back to that first lecture with the AI categories square that was shown. I can see where the rational "thinking" is not present in the LLM. However, as an assistant for setting up a foundation or explaining concepts in simpler terms, given an explanation from another source, GPT40 does a good job of distilling things.