CS 440: Introduction to Artificial Intelligence

Assignment 1: Fast Trajectory Replanning

Anay Kothana (215008100)   Joon Song (210007540)

October 2, 2024

Introduction

The goal of this assignment is to implement and compare various A* algorithms for pathfinding in gridworld environments. The task involves generating gridworlds, planning paths for agents using different variants of the A* algorithm, and analyzing their performance. The primary objective is to understand how different A* algorithms behave, how tie-breaking strategies affect performance, and how adaptive heuristics can improve efficiency over time.

Part 0 - Setup of Gridworlds

Gridworlds are generated using Depth-First Search (DFS) and are stored as 101×101 grids. Cells are randomly blocked with a 30% probability after the DFS exploration to create obstacles for the agent. The gridworlds are saved as `.npy` files for later use. A total of 50 gridworlds are generated, each representing a different environment in which the agent must plan a path from a random start to a random goal.

An example of one of the gridworlds can be visualized below:

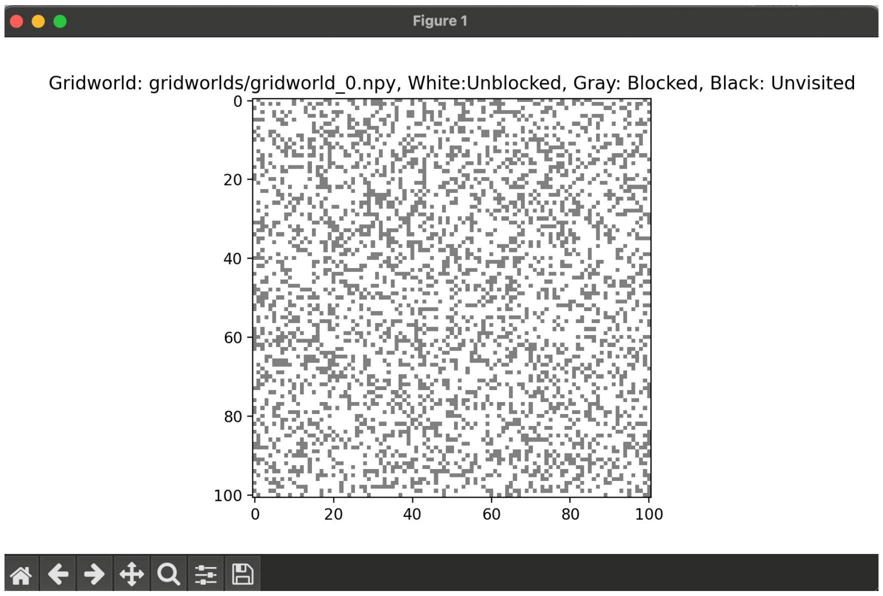
Figure 1
Gridworld: gridworlds/gridworld_0.npy, White:Unblocked, Gray: Blocked, Black: Unvisited
0
20
40
60
80
100

Figure 1: Visualization of a generated gridworld.

Part 1 - Understanding the Methods

Agent's First Move

The agent's first move is to the east (right) in the example due to the Manhattan distance heuristic used in A*. Manhattan distance is calculated by

$$|\text{current\_x} - \text{goal\_x}| + |\text{current\_y} - \text{goal\_y}|\,.$$

Looking at the figure, moving east would be favorable since the heuristic of that cell would be lower than moving in any other direction (since its closer to the goal).

Proof of Finite Time

We can prove that the agent will either reach the target or discover that no valid path exists in finite time. The number of moves is bounded by the number of unblocked cells, n, squared. Since the agent only explores unblocked cells and blocks discovered during exploration, it cannot move infinitely within a finite grid. As shown in the A* algorithm implementation, the open list grows as new nodes are added and shrinks as nodes are processed, guaranteeing that the search will terminate once all paths are explored.

Part 2 - The Effects of Ties

Tie-Breaking Strategies

In the Repeated Forward A* implementation, two tie-breaking strategies are tested: favoring higher g-values and favoring lower g-values. The tie-breaking strategy determines how the agent prioritizes nodes with the same f-value. - **Higher g-values**: The agent prefers paths farther from the start. - **Lower g-values**: The agent prefers shorter, cheaper paths to the current node.

Comparison

We ran data using each algorithm on all 50 gridworlds and these were the results:

Algorithm	**avg Runtime (s)**	**Expanded Nodes**
Higher g-values	3.07	88,588
Lower g-values	1.12	88,588
Repeated Forward A*	0.021	1,982

Table 1: Comparison of tie-breaking strategies in Repeated Forward A*.

Both tie-breaking strategies (Higher and Lower g-values) expand the same number of nodes, but favoring lower g-values results in significantly faster runtime. Repeated Forward A* outperforms both in terms of runtime and expanded nodes.

Part 3 - Forward vs. Backward A* Comparison

Explanation

As we were running the visualization for Repeated forward A* and Repeated Backwards A*, whenever the start and goal locations were close to each other, we observed that the runtime and expanded cells were similar. But, when the start and goal were not close to each other, Backwards Repeated A* tended to go all over the place.Repeated Backward A* is often slower than Repeated Forward A* because planning from the goal to the start gives less useful heuristic information, especially if the start is in a complex or obstacle-filled area. As a result, the algorithm explores more nodes and has to re-plan more often when it encounters new obstacles. On the other hand, Repeated Forward A* benefits from knowing about nearby obstacles right from the start, making the path-finding process quicker and more efficient. For comparison, pictures are added below when Backwards Repeated A* and Forwards Repeated A* are run.

Comparison

The data for avg runtime and expanded nodes shows that Repeated Forward A* performs significantly better in terms of runtime and expanded nodes compared to Repeated Backward A*.

Algorithm	**avg Runtime (s)**	**Expanded Nodes**
Repeated Forward A*	0.021	1,982
Repeated Backward A*	0.625	36,648

Table 2: Comparison of Forward A* vs. Backward A*.

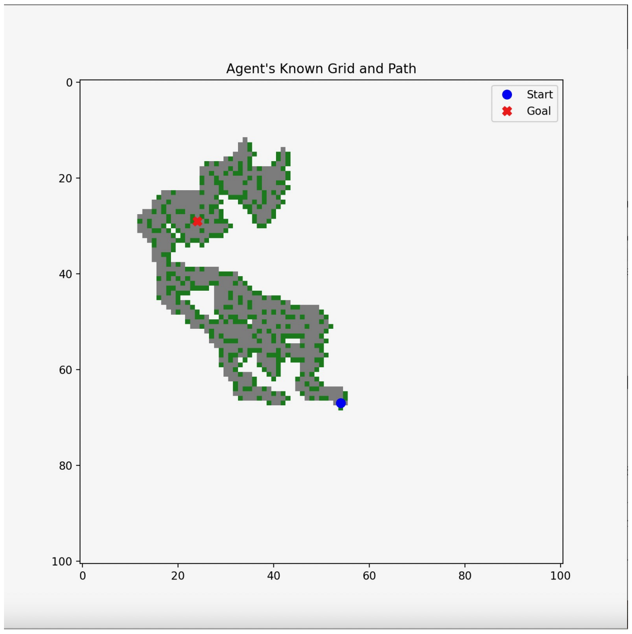

Figure 2: Backwards Repeated A*

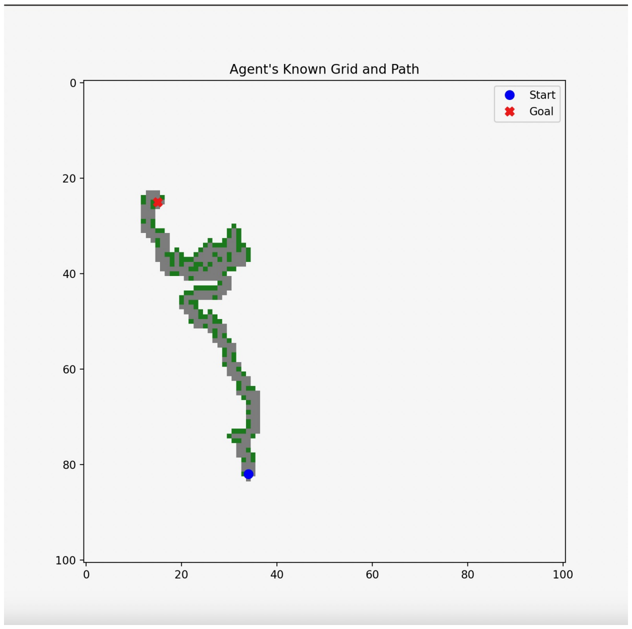

Figure 3: Forwards Repeated A*

Part 4 - Heuristics in Adaptive A*

Proof of Consistent Heuristic

Manhattan distances are consistent in gridworlds where the agent can only move in four directions. Since the agent moves without diagonal steps, only the changes in the x and y coordinates matter, aligning with the Manhattan distance metric. The cost of reaching any node is also consistent with the triangle inequality, meaning the direct path to the goal is never more expensive than detouring through another node.

Consistency in Adaptive A*

Adaptive A* ensures that initially consistent h-values remain consistent even if action costs increase by updating the heuristic values after each search. These updates reflect the actual path costs from the nodes to the goal, ensuring that the h-values continue to satisfy the consistency condition. As a result, the heuristic remains both admissible and consistent throughout the process, even when the environment changes.

Part 5 - Adaptive A* vs. Repeated Forward A* Comparison

Explanation

Adaptive A* improves upon Repeated Forward A* by updating its heuristic values after each search, allowing the agent to reuse information from previous searches to reduce the number of expanded nodes in future searches. This makes Adaptive A* more efficient in dynamic environments where the agent must frequently re-plan paths.

Comparison

The data shows that Forward A* was faster and expanded less nodes over the 50 gridworlds.

Algorithm	**avg Runtime (s)**	**Expanded Nodes**
Repeated Forward A*	0.021	1,981
Adaptive A*	0.535	21,100

Table 3: Comparison of Adaptive A* vs. Repeated Forward A*.

Adaptive A* took significantly longer and expanded more nodes in this case. From the comparisons, it is evident that Repeated Forward A* consistently performs the best in terms of runtime and expanded nodes across all gridworlds. While we were running the visualization of Adaptive A* and Repeated Forward A*, Repeated Forward A* only explored in a thin path to reach the goal while Adaptive A*'s path is much thicker. This could be the case because in Adaptive A*, it runs the search and then it updates the heuristic values accordingly to be the most efficient. That doesn't always mean it will result in the best outcome though; since we used Manhattan distances for the heuristic values, the heuristic value itself is pretty accurate compared to the actual distance. The return cost for running Adaptive A* doesn't make up for the initial searches it had to make to get the good heuristic value.

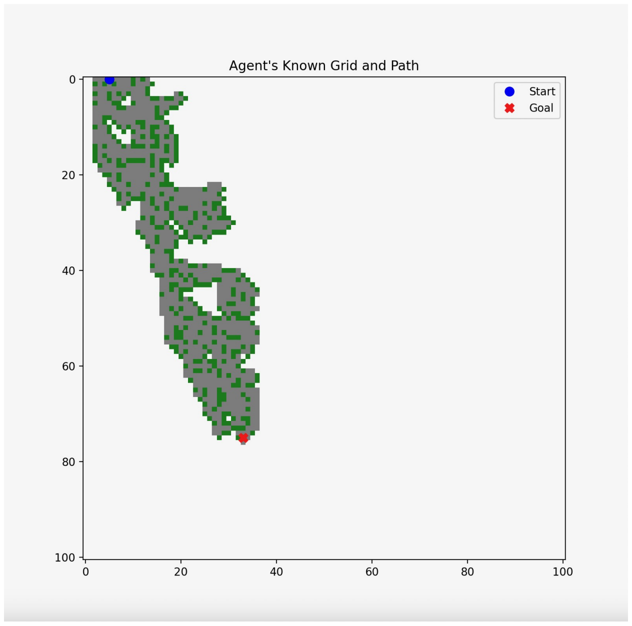

Figure 5: Example of Adaptive A* being run

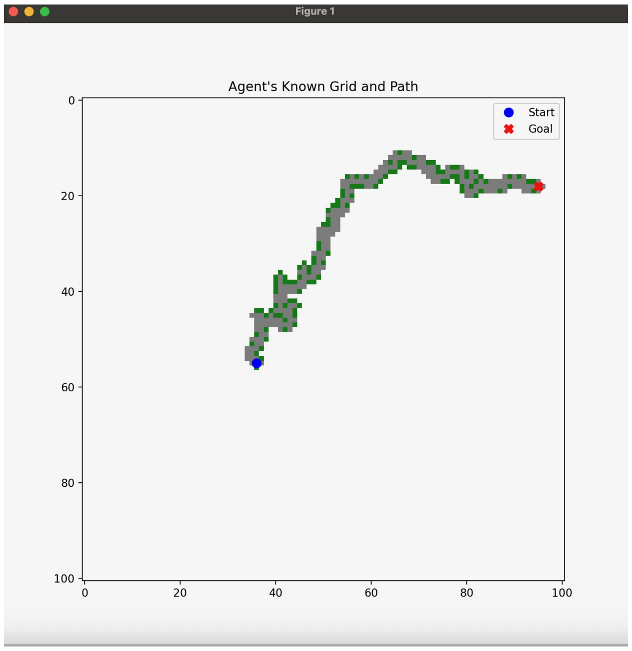

Figure 4: Example of Repeated Forward A* being run