CS 440 : Intro to Artificial Intelligence

Fast Trajectory Replanning

Ritwika Das - RUID: 214003221

September 30, 2024

Abstract

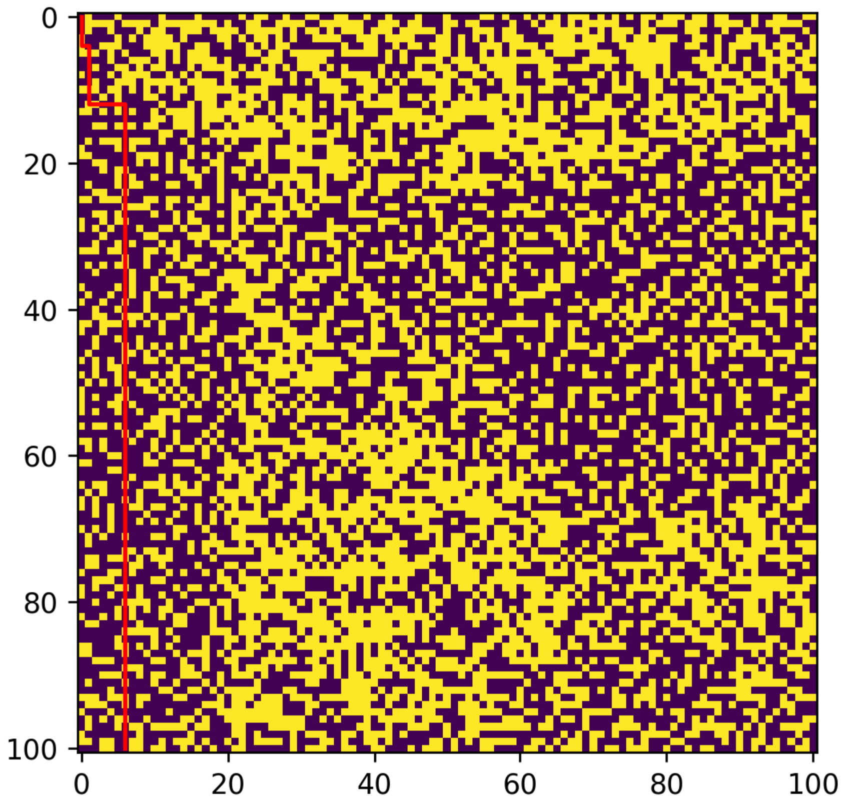
For this project we created a program that will the best way possible for a robot to reach it's goal from a random starting point. We used Python to write the code for this project. To run this program you'll have to create the mazes using the ... (Add More stuff here)

Part 0

To create a 101x101 maze, we begin by generating a NumPy array filled with zeros. Next, we employ a random depth-first search algorithm to create paths from neighboring points. We start at a random coordinate and maintain a visited set to track processed and completed coordinates until every cell in the maze is visited.

Within a loop, we randomly select a coordinate from the neighbors of the most recently processed cell and generate a uniform random number between 0 and 1. If this number is less than 0.3, we block the cell; if it's 0.3 or greater, the cell remains unblocked. We then add the visited coordinate to a stack.

If we reach a dead-end (when there are no unvisited neighbors), we backtrack by popping from the stack until we find a cell with unvisited neighbors, at which point we select a new random coordinate from those neighbors. If the stack becomes empty at any point, we restart the algorithm from a new random point. An example of a generated maze is shown in the accompanying image.



Part 1

(a) The objective of this program is to determine the shortest path for an agent to reach the destination at Cell E5. In this scenario, the agent starts at Cell E3. From this position, the agent has four possible movement options. However, moving South is not feasible since Cell E3 is at the bottom edge of the grid. Additionally, the agent cannot move West or North due to blockages in those directions. Consequently, the only viable option for the agent is to move East, which it proceeds to do.

(b) There are several ways to demonstrate this concept. The most straightforward method is to note that the number of grids is finite, ensuring that the agent will eventually reach the goal. In the worst-case scenario, the agent may need to traverse the entire map. Additionally, the A* algorithm employs a closed list, which prevents the agent from expanding cells that have already been processed. This effectively addresses the issue of infinite loops. To handle situations where the goal is blocked, the program will cease execution once all cells have been expanded. Therefore, if the goal becomes inaccessible due to obstructions, the program will simply stop running.

Part 2

After implementing both versions of Repeated Forward A*, we observed that the algorithm with the smaller g-values performed worse compared to the one with larger g-values, both in terms of the number of nodes expanded and the overall runtime.

This behavior occurs because the algorithm with smaller g-values tends to expand many more cells near the starting point at the beginning of the search. As a result, the runtime for the algorithm with larger g-values is significantly faster than that of the one with smaller g-values.

In fact, the algorithm with smaller g-values was found to be approximately 15 times slower than its counterpart with larger g-values.

Part 3

After comparing the two algorithms, we found that their runtime and the number of nodes expanded were fairly similar overall. However, despite both algorithms covering the same distance, we observed that Repeated Backward A* outperformed Repeated Forward A* in terms of speed and efficiency. Specifically, Repeated Backward A* had a faster runtime and expanded approximately 10 percent fewer nodes compared to Repeated Forward A*.

Part 4

(a) Manhattan distance refers to the distance between two points on a grid, calculated strictly based on horizontal and vertical movements. In a grid world, the agent can only move in four directions: up, down, left, and right. Therefore, Manhattan distance represents the fastest possible path for the agent to reach its goal.

Since the agent cannot move diagonally, we can confidently assert that the shortest path will always be found using this metric, making Manhattan distance a consistent heuristic. However, if diagonal movement were allowed, the heuristic would likely overestimate the actual distance, as it would not account for the possibility of shorter, diagonal paths.

(b) A heuristic $h(n)$ is considered consistent (or monotonic) if, for every node n and every successor n' generated by an action a, the estimated cost of reaching the goal from n does not exceed the step cost of moving to n' plus the estimated cost of reaching the goal from n'. In mathematical terms, this can be expressed as:

$$h(n) \leq c(n, a, n') + h(n')$$

Now, let's introduce a scenario where the cost of an action increases. Let c represent the cost before the increase and c' the cost after the increase. We want to demonstrate that the heuristic remains consistent after the cost increase.

Using the identity above, we can write:

$$h_{new}(n) = h_{new}(n') + c(n, a, n')$$

Since $c(a, a, n')$ is the cost before the increase, we can also relate it to the new cost c' after the increase:

$$h_{new}(n) \leq h_{new}(n') + C(n, a, n') \leq c'(n, a, n')$$

This shows that the new heuristic $h_{new}(n)$ does not exceed the increased action cost $c'(n, a, n')$. Hence, we can conclude that the heuristic remains consistent even after the action cost increase.

Part 5

After implementing both algorithms, we found that their run times were fairly similar. However, Adaptive A* outperformed Repeated Forward A* by (INSERT NUMBER). The number of nodes expanded was also nearly the same for both algorithms, but Adaptive A* was approximately 4 percent more efficient than Repeated Forward A*. Based on these findings, we can conclude that Adaptive A* is a superior choice compared to Repeated Forward A*.

References

[1] Binary Heaps, https://runestone.academy/runestone/books/published/pythonds/Trees/BinaryHeapImplementation.html

[2] Basic A* Information, https://www.geeksforgeeks.org/a-search-algorithm/

[3] Real-Time Adaptive A*, http://idm-lab.org/bib/abstracts/papers/aamas08b.pdf