CS 440: Intro to Artificial Intelligence

Antonio Mena (211006469), Aryan Jairath (207002700), and Naman Bajaj (200001697)

February 2024

Part 0

To create a maze with dimensions 101 x 101, we firstly predefined two variables for the number of rows and columns of the maze. We then utilized a DFS approach to actually fill up the maze randomly. In the case of our maze, 1 represents a path and 0 represents a wall. As such, we created two random coordinates and set those to be 1 for the maze since the destination and starting coordinates cannot be a wall. Until all cells are not visited, we continuously iterate through neighbors of a given coordinate that are not visited and set it a value of 1 or 0 based on the probabilities of 0.7 for it being a path and 0.3 for it being a wall. In any case where there is a dead end, a value is popped from the stack to backtrack since that means there are no unvisited neighbors. When there are no neighbors that are unvisited, we needed a fresh cell that had not yet been visited to continue to maze filling. We grab an unvisited cell randomly from a list of unvisited cells that are dynamically generated when we realize that we have no neighbors left to visit. This process is continuously repeated until each cell is visited. We utilized pyplot from Matplotlib to illustrate the maze itself along with the color green to indicate a path versus red to illustrate a wall. We also illustrate the actual shortest path, if there is one, from the start to the goal using a blue color. This process is repeated 50 times which allows us to create 50 distinct mazes.

Part 1

a. The first move of the agent for the example shown in the writeup from Figure 8 is to the east rather than the north because the agent is only aware of the four directions of North, South, East and West upon starting. These means that, in the agent's perspective, there are currently no obstacles at all so the closest path to the goal state is simply the Manhattan path from the start state to the goal state. The agent will then follow the path and try to reach the goal state and it will recognize the obstacles that exist both to its east and north. Once this occurs, another A* search will be performed again this time with the knowledge of the new maze with the two new obstacles that it found, and it will now calculate the shortest path while avoiding those newly acquired obstacles.

b. An agent in finite gridworlds either reaches the target or discovers that this is impossible in finite time simply because the gridworld itself is finite and the A* algorithm systematically explore the gridworld until all nodes are exhausted. Since we know the number of squares in the grid to be finite and that A* uses a closed list which means that nodes are only expanded once, either all nodes are expanded and the algorithm comes to an end or the target is found and a subsequent path is returned. In the case that the agent can reach the target, then the path will be found, as every move by the agent either brings it to a new state or visits a previously visited state, progressing towards the goal in either case. In the case that no path exists, the agent will explore every possible state without finding the target. Since the number of states is finite, the agent will discover that it can't reach the target in finite time. Either way, the agent will discover a path to the goal or discover no such path exists in finite time. To prove that the number of moves of the agent until it reaches the target or discovers that this is impossible is bounded from above by the number of unblocked cells squared, consider U to be the number of unblocked cells in the grid and M to be the maximum number of moves made by the agent. In the worst case, for an agent to deem a target unreachable or to successfully reach it involves visiting every single unblocked cell. If we are to consider an algorithm that does not minimize revisiting cells, then we know that there are is potential for nodes to be explored more than once when trying to determine the path. The number of moves that an agent can make in the worst case is bounded by U since there can be at most $U - 1$ neighbors. Since there are also U unblocked cells we know that we are bounded by $U * (U - 1)$ which can be simplified simply to a safe upper bound of U^2

Part 2

After implementing both versions of tie-breaking, we noted some interesting findings regarding the runtime and number of expanded nodes that resulted from selected either smaller or larger-values. When we utilized $c * f(s) - g(s)$ as the priority to select larger g-values in case of a tie, we noticed significantly less nodes expanded than no tie-breaking and an even more drastic difference than when we compared it to tie breaking with smaller g-values. In our testing, we noticed around 7.34 times the number of nodes expanded when tie-breaking with smaller g-values when comparing it to tie-breaking with larger g-values. A reason for this observation is that higher g-values indicate that the a longer path had to be taken to get to that given node which means that more of the maze has actually been traversed and explored. Selecting this value in the tiebreaker may reduce total expanded nodes since we are using the coordinate that has observed more in the maze and has deeper exploration overall in comparison to selecting a g-value that is smaller.

Part 3

After implementing both repeated forward A* and repeated backward A*, we were able to compare their respective number of expanded cells. Upon observation, we noted that repeated forward A* expanded significantly less cells when compared to repeated backward A*. On average, for every 1 cell expanded by forward A*, the backwards variant expands 8.661 cells which means that the efficiency is far less for repeated backwards A*. There may be a couple of reasons as to why our backwards A* is significantly more inefficient in terms of visited nodes than our forwards A*. Firstly, our usage of Manhattan distance for calculating our backwards heuristic may not be as optimized for backwards traversal as it is forwards, as our initial implementation revolved around forwards A* (since that was the first part we began work on). Additionally, our tie breaking logic may have the same issue. Overall, since most functions were designed for forwards, using backwards on those same functions causes unnecessary overhead and occasionally additional steps being made by the agent.

Part 4

a. Manhattan distances are consistent in gridworld where the agent can only move in the four main compass directions since we know that each move is of cost 1. From a given node to its successor + the heuristic of the successor node will be greater than the heuristic value of the actual node. Since the Manhattan distance represents the shortest path, we also know that there is not possible that the cost from node n to $n' + h(n')$ will be less than $h(n)$ in any case since $h(n)$ is minimal. the estimated cost to reach the goal from the current node is always less than or equal to the cost of taking one step to a successor node plus the estimated cost from that successor to the goal. In other words, we can also say that it is admissible since the heuristic represents the shortest path from point A to point B in this case, so it is impossible for any path to cost less than the cost from the Manhattan Distance.

b. Adaptive A* leaves initially consistent h-values consistent even if action costs can increase. Firstly, a heuristic is consistent if following inequality holds: $h(a) \leq c(a, z, b) + h(b)$, that is, the cost of going from node a to every successor node b + going from node b to the goal is greater than or equal the estimated cost of going directly to the goal from node a. In this expression, we see that $c(a, z, b)$ represents the path cost of going to node a to node b using action z. We want to show that if the action costs do increase, then this will still result in initially consistent h-values staying consistent. Call the increased action cost c_2 and the new heuristic h_2. Since action costs have increased, $c(a, z, b) \leq c_2(a, z, b)$. Since adaptive A* aims to improve the heuristic with every move, we can also assume $h(b) \leq h_2(b)$. We can then add these inequalities and find that $h_2(a) \leq h(a) \leq h_2(b) + c(a, z, b) \leq c_2(a, z, b) + h_2(b)$. We know this to be true since the actions costs are increasing, so the provided inequality holds. This then shows that the updated heuristic h_2 remains consistent even after the action costs increase.

Part 5

Upon implementation of Adaptive A* and Repeated Forward A*, we noticed a noticeable difference in performance with respect to expanded cells. We also ensured that both algorithms broke ties with larger g-values in case of identical f-values. In our testing, Adaptive A* expanded on average 5% less cells than Repeated Forward A*. This can be explained by the fact that Adaptive A* learns from its previous searches and updates the heuristic value based on the actual costs that are discovered in prior searches. After one search, the algorithm adjusts h-values for all expanded nodes which means that the heuristic is more informed as a whole. This means that the heuristic better reflects a more accurate, up to date, prediction as compared to the static heuristic of the non adaptive repeated forward A*.

Part 6

A statistical hypothesis test can be performed by first formulating a null hypothesis and an alternative hypothesis. As highlighted in Empirical Methods for Artificial Intelligence, evaluation begins with claims. As such, our claim is that Adaptive A* search is faster and expands less nodes than repeated forward A* search on average. In our experiment, the independent variable is the style of A* search we use, and the dependent variable is the number of expanded nodes or the runtime. In this case the null hypothesis is that there is no significant variation in the runtime between adaptive A* and repeated A*. The alternative hypothesis would be that there is a substantial difference in search time between adaptive A* and repeated A*. Then we would define the metrics to compare, for this case we can use the total nodes explored/expanded of the repeated A* search compared to the adaptive A* search. Next, we can set a significance level of 0.05. Then, for the test we can run the two searches on 10 different mazes and see the outcome after searching. Then, once we calculate the p-value after these tests we can observe the results and decide whether or not we can reject the null hypothesis.