CS440 - Assignment 1 Report

Akash Shah (199003227) and Zain Chalisa (200000191)

February 25th 2024

1 Understanding the methods

Explain in your report why the first move of the agent for the example search problem from Figure 8 is to the east rather than the north given that the agent does not know initially which cells are blocked.

- The agent's first move is east because on the planning grid, the A* algorithm assumes all the cells can be traversed through unless proved otherwise. Going east takes the agent along the shortest path under this assumption. If the agent went north, it would deviate from the assumed shortest path without any extra information, which goes against the algorithm.

This project argues that the agent is guaranteed to reach the target if it is not separated from it by blocked cells. Give a convincing argument that the agent in finite gridworlds indeed either reaches the target or discovers that this is impossible in finite time. 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.

Since the gridworld is finite, that means there is a finite amount of cells. We can denote the number of cells as N. We also have a finite number of states the agent can be in, as the agent can not enter an unblocked cell. Each move will bring the agent either closer to the target, keep the agent in the same position, or moves the agent to an another unblocked cell. Given that the agent can remember visited cells, each unblocked cell will be visited at most once when searching for the target. Hence, the agent will either find the target by moving towards it, or will exhaust all reachable unblocked cells without finding the target. Since the number of unblocked cells is finite, the process is guaranteed to terminate in finite time.

In the worst case scenario, the agent may have to backtrack multiple times from deadends. This could involve revisiting cells such that the total amount of moves is proportional to the square of unblocked cells. This is because for every cell visited, the agent might need to consider moving to every other unblocked cell to explore all possible paths.

2 The Effects of Ties

Repeated Forward A* needs to break ties to decide which cell to expand next if several cells have the same smallest f-value. It can either break ties in favor of cells with smaller g-values or in favor of cells with larger g-values. Implement and compare both versions of Repeated Forward A* with respect to their runtime or, equivalently, number of expanded cells. Explain your observations in detail, that is, explain what you observed and give a reason for the observation.

- In our code we implemented methods to compute tie breaking both larger g-values and smaller g-values. From our observations and running our code we found that larger g-value tie breaking is much more optimal than smaller g-value tie breaking. While running our 50 grids of size 101 by 101 on Board 8 of our results.txt file we found the number of expanded nodes for larger g-value tie breaking was 3969 while for smaller g-value tie breaking on the same grid the number of expanded nodes was 31569. Looking at this difference we can come to the conclusion that larger g-value is more optimal based on stats alone. However, while digging into the understanding of why this occurs we found that larger g-values favored expansions on the direct path, and only expanded nodes when there were obstacles in the way of the optimal path. On the other hand, using smaller g-values we find that it favors expanding nodes which are closer to the agents position. Looking at the purpose of our algorithm we're trying to find the most optimal of path from the initial point to the goal rather than exploring alternative routes which sums up why in the purpose of our algorithm larger g-value for tie breaking is much more optimal.

3 Forward vs. Backward

We implemented both Repeated Forward A* and Repeated Backward A*, both implementing tie-breaking by using larger g values. Repeated Backward A* is similar to Repeated Forward A*, except that during the planning stage, we perform the search for our optimal path from the target position to the initial position. When running our 50 101x101 grids, we saw that the biggest number of nodes Forwards A* expanded was 3969, while the biggest number of nodes Backwards A* expanded was 215739 (Board 8 in our results.txt file). Looking at this, we can most likely conclude that Forwards A* is more optimal than Backwards A* (with larger g tiebreaking). After doing some investigation, we realized this is because during Backwards A*, when an obstacle is encountered near the agent, the next shortest path may have a higher f value. This will, in turn, provoke the Backwards A* to explore all previously ignored states with a lower f value . It will keep going until a viable path or all states with that f value have been expanded. In scenarios where the majority of obstacles are located near the Agent's position, the Forward A* is more efficient, as it goes around obstacles in the beginning of the path and then proceeds directly to the goal, minimizing the number of expanded cells.

4 Heuristics in the Adaptive A*

4.1: The project argues that "the Manhattan distances are consistent in gridworlds in which the agent can move only in the four main compass directions." Prove that this is indeed the case.

- The agent in the grid worlds can move a total of 4 directions: North, South, East, and West. These directions are otherwise also known as the 4 cardinal directions. To prove that the Manhattan distance is consistent in gridworlds we first need to look at the formula. The Manhattan Distance formula is $ManhattanDistance = abs(x1 - x2) + abs(y1 - y2)$. Knowing that the agent can only move in 4 cardinal directions the agent on this grid would first need to move $abs(x1 - x2)$ horizontally, before it can move $abs(y1 - y2)$ vertically. This further supports the idea that the total number of steps the agent would need to take would be $abs(x1 - x2) + abs(y1 - y2)$. All in all, the Manhattan distance is consistent with how many steps the agent will take to reach the goal if no blockers were encountered.

4.2: Prove that Adaptive A* leaves initially consistent h-values consistent even if action costs can increase.

- While proving Adaptive A* we need to consider how the algorithm is calculating its new consistent heuristic values. The way in which Adaptive A* runs is by expanding all possible and optimal paths, and before checking if that path is possible on the execution grid world it will update the heuristic value by $h(n) = g(goal) - g(s)$. Using what we know now about Adaptive A* we can understand why the initial h-values are consistent even after the action costs increase. The h-values in adaptive A* are only updated when $h_new(s) <= h_new(s') + cost(s, a)$, this would result in the h-values only being the true cost since the triangle inequality is satisfied and the cost includes what it took to reach s in the first place. All in all, this allows us to come to the conclusion that the new h-value updated is consistent.

5 Heuristics in the Adaptive A*

Q: Implement and compare Repeated Forward A* and Adaptive A* with respect to their runtime. Explain your observations in detail,

that is, explain what you observed and give a reason for the observation. Both search algorithms should break ties among cells with the same f-value in favor of cells with larger g-values and remaining ties in an identical way, for example randomly

In our code we have implemented both Forward Astar* and Adaptive Astar* both using larger g-values for tie breaking. The way in which we implemented the these two algorithms consists of the following. We have two grids, one being the planning grid (initialized to be completely empty) and the other being the execution grid (initialized to contain all the blockers along with open spaces).

The way we move between the planning and execution grid:

1. We start on the planning grid. We find the most optimal path from the initial starting point to the goal.
2. We then take the found path and try and implement it on the execution grid, holding the path up until we find a blocker on the path. If there is no blocker found on the path we return the path as the most optimal, however, if we encounter a blocker we start the agent at the point on the path before we encounter the blocker.
3. We continue to move back and forth between the planning grid and the execution grid until an entire path is constructed from the initial point till the target.

The main difference between Adaptive Astar* and Forward Astar* is in Adaptive we update our heuristic values for every cell in the closed list before we go back to execution. The way in which we go about doing so is $h_new(n) = g(s_goal) - g(s)$. Updating the heuristic matrix on the fly in Adaptive Astar* helps benefit the possible over/underestimate of the heuristic in Forward Astar* because it give a much more accurate distance from the agent to the target compared to the already calculated heuristic in the matrix.

All in all, Adaptive Astar* and Repeated Forward Astar* are consistent between one another. When looking at Board 8's results in our results.txt file we find that Forward Astar* expanded 3969 nodes while on the same grid Adaptive Astar* expanded 3530 nodes. The main difference is found in the number of expanded nodes which the two algorithms use to find the target. Adaptive Astar* expands less nodes than Repeated Forward Astar*, however this did not seem to make a significant difference in their run times.

6 Statistical Significance

Q: Performance differences between two search algorithms can be systematic in nature or only due to sampling noise (= bias exhibited

by the selected test cases since the number of test cases is always limited). One can use statistical hypothesis tests to determine whether they are systematic in nature. Read up on statistical hypothesis tests (for example, in Cohen, Empirical Methods for Artificial Intelligence, MIT Press, 1995) and then describe for one of the experimental questions above exactly how a statistical hypothesis test could be performed. You do not need to implement anything for this question, you should only precisely describe how such a test could be performed.

We can use a paired t - test to see if the difference between 2 search methods. We first define a test statistic for the search methods to quantify it. For example, we can take the lengths of the paths generated by the 2 search methods. We run this test multiple times, generating different path lengths for the same size maze for the 2 search methods. We then take the mean difference between these path lengths and call it $\overline{d}$.

After that, we take the standard deviations of these differences and label it σ.

We can calculate the t value by setting $t = \overline{d}/(\sigma/\sqrt{n})$, where n is the number of observations you have.

You can calculate the p - value given n and t. If p is less than 0.05, it is not a statistically significant difference. If it is equal to or greater than 0.05, it is a statiscally significant difference.