Group Members:
Katherine Sidebotham (198009560), Stephanie Lin (199005357),
Jasper Van Der Ree (193005325)

By submitting this file, I hereby certify that this paper represents my own work and I have read RU academic integrity policies.

Group Members:
Katherine Sidebotham (198009560), Stephanie Lin (199005357),
Jasper Van Der Ree (193005325)

0 Setup your Environments

1 Understanding the methods

(a) The first move of the agent is to the east rather than the north because the agent is performing the A* algorithm. Given that the agent has no prior knowledge of which cell is blocked or unblocked, the agent uses heuristic values to determine which direction to proceed in with the goal of minimizing the estimated total cost. In Figure 8, the estimated total cost to the east is less than the cost to the north, therefore the agent moves east.

(b) An agent in a finite gridworld with no blocked cells between it and the target is guaranteed to either reach the target or discover that it is impossible to reach the target in finite time. This is because a finite number of cells in the gridworld means there must be a finite number of possible paths from the agent to the target. We can prove this by thinking about the upper bound for finding the correct path to the target. The agent must explore all possible paths from its starting cell to the target, and each cell within the gridworld has at most four neighboring cells that are not blocked. That means that there are at most four possible moves from one cell to another. Using this logic, the maximum number of moves made by the agent is bounded by the number of unblocked cells multiplied by the maximum number of moves from each cell, both of which are finite. Moreover, the agent does not visit the same cell twice when moving along its path, so the number of possible paths taken by the agent is thus bounded by the number of arrangements of the unblocked cells along a single path. Mathematically, that can be written as n! where n = the number of unblocked cells in the gridworld, which is guaranteed to be finite as well.

2 The Effects of Ties

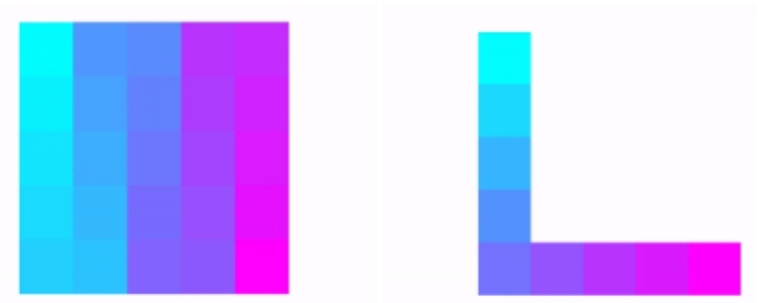
Let:

- Version 1: Repeated Forward A* is implemented with tie-breaking in favor of cells with smaller g-values.
- Version 2: Repeated Forward A* implemented with tie-breaking in favor of cells with larger g-values.

When both versions are implemented and run, we observe that version 1 expanded on fewer cells than version 2. This could be explained by the fact that in version 2, cells with larger g-values are favored, therefore cells that are further from the starting state are more likely to be explored versus the strategy used in version 1. As for the runtime between version 1

and version 2, we observe that version 2 has a longer runtime than version 1, the reasoning behind this is similar to the previous statement. Since the cells with larger g-values are favored, those that are further from the starting state will still be explored whereas in version 1, cells that are closer to the starting state are in favor of expansion, this would help reduce runtime in the program. In fact, when both programs run through a set of 50 grids, 101x101 in dimensions, Version 1 has an average runtime of 1.633282985687256 and standard deviation of 1.9642438398908 and Version 2 with an average runtime of 2.4150047206878664 and standard deviation of 1.8210313691088. Overall, it seems that the strategy used in version 1 is more optimal and efficient.

In Figure 9, when Version 1 of Repeated Forward A* is implemented, the search algorithm finds the most optimal path almost immediately, without having to expand through the rest of the maze (right image below). When Version 2 of Repeated Forward A* is implemented, the search algorithm goes through the entire maze and then returns the path (left image below). As we observe the performance differences, Version 1 is much more efficient in its search and cell expansion compared to Version 2 as it expands through fewer cells which reduces its runtime effectively.



3 Forward vs. Backward

When both Repeated Forward A* and Repeated Backward A* algorithms are implemented, we observe that Repeated Backward A* expanded on more cells than Repeated Forward A*, this is because forward search explores the search space with simpler space first rather than the cells that are near the goal state which tends to be more complex. The main difference observed between the two algorithms was that Repeated Forward A* searches from the start state to the goal state, whereas Repeated Backward A* searches from the goal state to the start state. Also, since the direction is reversed, as is the heuristic estimate – calc_h() function estimates the distances from the current state to the start state for Repeated Backward versus estimating the distance from the current state to the goal state which is done by Repeated Forward. As for runtime, Repeated Forward A* has a lower runtime than Repeated Backwards A* due to the fact that forward search explores the cells farther from

the goal which are less complex. When both programs run through a set of 50 grids, 101x101 in dimensions, Repeated Backwards A* has an average runtime of 1.5351191329956055 and standard deviation of 1.8060791411453 and Repeated Forwards A* with an average runtime of 1.4978095626831054 and standard deviation of 1.6660489961012. However, it is evident that the runtime varies based on the locations of the start and goal state, as well as the obstacles that are present in the gridworld and how they may impact possible paths.

4 Heuristics in the Adaptive A*

Let,

- Manhattan distance: "The distance between two points measured along axes at right angles."
- Grid-world: A 2-D Euclidean space consisting of a rectangular array of evenly distributed nodes.

As long as these definitions hold, it should be possible to calculate the distance between any two nodes as the amount of transitions between nodes it takes to move from the first to the second in Grid-world. The agent can only move in 4 cardinal directions, so it cannot move diagonally from one node to another and therefore may only move in such a way that would increase the distance from its starting node by 1 and decrease the distance to its destination node by 1. This change of 1/-1 in Manhattan distance cascades to all nodes in the grid-world, providing a consistent method of measuring and updating distance between the agent and every node it does not currently occupy.

Let,

- Admissibility: "An admissible heuristic is one that never overestimates the cost to reach the goal."
- Consistency: A heuristic h(n) is consistent if, for every node n and every successor n' of n generated by action a, the estimated cost of reaching the goal from n is no greater than the step cost of getting to n' plus the estimated cost of reaching the goal from n'. This can also be represented by the inequality h(n) $\leq$ (cost of action from state n to state n') + h(n')

Adaptive A* algorithm works by dynamically updating the heuristic for each state based on the actual costs recorded of reaching that state. As the search progresses, the algorithm uses the current estimated heuristic for the current state to decide the next step in the path. If the actual cost of the new state is higher than the estimated heuristic, the program will update the heuristic to reflect a more accurate value. This is done by the summation of the

path cost from the current state to the new state and the estimated cost of the estimated heuristic value. By using adaptive calculations, Adaptive A* ensures that the new heuristic is admissible and consistent regardless of increasing action costs between states.

5 Heuristics in the Adaptive A*

After running a set of 50 gridworlds with 101x101 dimensions to compare Adaptive A* and Repeated Forward A*, we observe that Adaptive A* algorithm performs with better run time measurements than Repeated Forward A*. Adaptive A* has a runtime of 0.16399239063262938 and standard deviation 0.10459342685379 of while Repeated Forward A* has a an average runtime of 1.4978095626831054 and standard deviation of 1.6660489961012. Considering the heuristic function of each program, it is expected that Adaptive A* search would be more efficient, as the algorithm adapts the heuristic calculation as the program runs while Repeated Forward A* uses a fixed heuristic calculation. The adaptive heuristic calculation increases accuracy and improves efficiency of the search thus allowing the program to explore the more promising paths of the maze.

6 Statistical Significance

To determine if the performance differences between two search algorithms is due to systematic difference or sampling noise, an experiment can be conducted by using statistical hypothesis t-testing.

Null hypothesis: There is no significant differences between the two algorithms in terms of runtime, therefore there is no systematic differences.
Alternative hypothesis: There is a significant difference between the two algorithms in terms of runtime, therefore there is systematic differences.

The t-test experiment would run both search algorithms through a set of randomly generated tests cases with different tie-breaking methods on each case. For example, the experiment would generate 50 random 101x101 dimension gridworlds for both algorithms to run through. After each iteration, the runtime of both algorithms are recorded. After all test cases are processed, the experimenters would compute the mean of the runtime for both algorithms and calculate the standard deviation, these values would then be used to find the t-statistic with the following formula:

$t = (\mu_1 - \mu_2)/(s * \sqrt{1/n_1 + 1/n_2})$

Where:

- μ_1 and μ_2 are the sample means of the two algorithms
- s is the standard deviation
- n_1 and n_2 are the sample size of the two algorithms

The t-statistic measures the difference of the sample data. If the t-statistic is large then the means of the two algorithms are different and unlikely to be due to chance. If the t-statistic is small then the difference between the means of both algorithms may not be statistically significant and is likely to be due to chance. After calculating the t-statistic, the experimenters could find the p-value, which is the probability value that indicates how likely the recorded data matches the null hypothesis. If the p-value is smaller than the significance value, typically set at 0.05 (5%), then the null hypothesis is rejected and the experiment concludes that the null hypothesis holds true. On the other hand, if the p-value is larger than the null hypothesis is accepted and the experiment concludes that the observed differences between the two algorithms are statistically significant.