Fast Trajectory Replanning Using Forward and Backward Repeating A* and Adaptave A*

Part 0 - Setup your Environments [10 points]:

For the visualization of the grid world we referenced a github page that used the graphics.py library that is found in the book "Python Programming: An Introduction to Computer Science" (Franklin, Beedle & Associates). The maze uses a 2D array of state objects and generates the blocked/unblocked states by iterating through the array. Each state had a 30% chance to be blocked. Our project does not store the gridworlds that it produces, so the attached 50 images of gridworlds are just examples. Our program generates 50 new worlds and tests the A* algorithms on each of the worlds each time it is run.

Part 1 - Understanding the methods [10 points]:

a) Given that the agent does not know that square E4 is blocked, its first move will be to square E3, at which point it will observe that E4 is blocked. The agents first move is to E3 because according to the heuristic, the Manhattan distance between the start node and finish node is 2 nodes east. The Manhattan distance is the idealized estimated path, so the agent follows the heuristic method until it hits a dead end, at which point it back tracks to the nearest parent with open links.

b) In the case where the end goal is blocked and cannot be reached, A* will continually expand nodes that it has visited. In a finite gridworld there is a finite amount of states that can be expanded therefore, eventually A* will run out of states that can be expanded. When A* has expanded all unblocked cells that it can reach and still has not found the goal then there is no solution. The number of expandable states cannot exceed the number of unblocked states. The agent is also guaranteed to find the goal if it is reachable because the goal will be an expandable state, which will also touch at least one expandable state and so on until the agent eventually finds the goal.

Part 2 - The Effects of Ties [15 points]

In our trials, we noticed that using the Forward Repeated A* with the smaller g values expanded a larger amount of nodes, on average, than the using the Forward Repeated A*

Fast Trajectory Replanning Using Forward and Backward Repeating A* and Adaptave A*

with the larger g values. In our trials, the larger g values search expanded 125 nodes less than the search conducted with the smaller g values. This may be because the distance from the starting node, plus the heuristic value must be less than or equal to the distance from the starting node of the goal node ($h(s) \leq g(s_{goal})+g(s)$). When the g-values, the distance from the starting node, are larger, there is less distance for the heuristic function to approximate, and the heuristic values become more accurate.

AVERAGE NODES EXPANDED ON SUCCESS USING FORWARD REPEATED A* AND LARGER G VALUES	877
AVERAGE NODES EXPANDED ON SUCCESS USING FORWARD REPEATED A* AND SMALLER G VALUES	1002

Part 3 - Forward vs. Backward [20 points]: Implement and compare Repeated Forward A* and Repeated Backward 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. Both versions of Repeated A* 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.

For our implementations of the Repeated Forward A* and Repeated Backward A*, we saw that the Repeated Forward A* expanded far fewer nodes, on average, than the Repeated Forward A*. According to our data, the Repeated Forward A* expanded 313 nodes fewer than the Repeated Backward A* search. This may be because the Repeated Forward A* search starts at the top left corner(0,0 position), and therefore, is cut off from being able to travel to the nodes West and North. Additionally, we noticed that Repeated Forward A*, and Repeated Backward A*, were more efficient in situations where a block is encountered towards the starting node, and goal node, respectively.

AVERAGE NODES EXPANDED ON SUCCESS USING FORWARD REPEATED A* AND LARGER G VALUES	877
AVERAGE NODES EXPANDED ON SUCCESS USING BACKWARD REPEATED A* AND LARGER G VALUES	1190

Fast Trajectory Replanning Using Forward and Backward Repeating A* and Adaptave A*

Part 4 - Heuristics in the Adaptive A* [20 points]: 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.

Manhattan distances are consistent in gridworlds in which the agent can move only in the four main compass directions. This is true because the Manhattan distances between points are equidistant. In the project, it states that "The Manhattan distance of a cell is the sum of the absolute difference of the x coordinates and the absolute difference of the y coordinates of the cell and the cell of the target." This means that all the step costs are equal and uniform. This means, that the heuristic modelling travelling from (0,0) to (2, 2) is equal to the heuristic modelling travelling from (3,3) to (5, 5).
To get to (2,2) from (0,0) the first successor state will be (1,0) with h = 3 and (0,0) will have an h of 4. Therefore if we follow the definition of consistency ($h(n) \leq c(N,P) + h(p)$) $\mathbf{4 \leq c(1) + 3}$. The cost to move the agent to the next state is 1 and the heuristic of the successor state will always be one less than the heuristic of the current state.

Furthermore, when the h-values are replaced during the Adaptive A* search by the $h_{new}(s)$ these new h-values are still consistent as they still satisfy the above inequality(def. of consistency) with any action cost greater than or equal to 1.
$\mathbf{4 \leq myC + 3}$ where myC >c(1).

Furthermore, it is argued that "The h-values $h_{new}(s)$ ... are not only admissible but also consistent." Prove that Adaptive A* leaves initially consistent h-values consistent even if action costs can increase.

Part 5 - Heuristics in the Adaptive A* [15 points]: 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 implementation of the Repeated Forward A* and Adaptive A*, we saw that the Repeated Forward A* search could be improved to expand 6 fewer nodes, on average. With each cycle, the heuristic is updated, such that the h(s) value is less than or equal to the most recently calculated distance to goal minus the distance from starting node($h(s) \leq g(s_{goal})-g(s)$). This makes the already most efficient Repeated Forward A* slightly faster.

Fast Trajectory Replanning Using Forward and Backward Repeating A* and Adaptave A*

AVERAGE NODES EXPANDED ON SUCCESS USING FORWARD REPEATED A* AND LARGER G VALUES	877
AVERAGE NODES EXPANDED ON SUCCESS USING ADAPTIVE A* AND LARGER G VALUES	871

Part 6 - Memory Issues [10 points]: You performed all experiments in gridworlds of size 101 × 101 but some real-time computer games use maps whose number of cells is up to two orders of magnitude larger than that. It is then especially important to limit the amount of information that is stored per cell. For example, the tree-pointers can be implemented with only two bits per cell. Suggest additional ways to reduce the memory consumption of your implementations further. Then, calculate the amount of memory that they need to operate on gridworlds of size 1001 × 1001 and the largest gridworld that they can operate on within a memory limit of 4 MBytes.

> Each state needs to store the f, g, h, s, and p values. F, g, s, and h can be stored as ints and p can be stored as a boolean value where North, South, East, or West can be marked as true if that is the direction of the parent. 4 bytes for an int = 16 bytes for f, g, h, and s + 2 bits for p. Total of 130 bits. For a gridworld of 1001x1001 there will be 1,002,001 nodes which will be 130,260,130 bits or 16.2 MB. A memory limit of 4MB would allow for 246,153 states which would be a grid world of 496x496 nodes (246,016 states).

Fast Trajectory Replanning Using Forward and Backward Repeating A* and Adaptave A*

Statistics:

NUMBER OF SUCCESSES 36
Success Table:

AVERAGE NODES EXPANDED ON SUCCESS USING FORWARD REPEATED A* AND LARGER G VALUES	877
AVERAGE NODES EXPANDED ON SUCCESS USING BACKWARD REPEATED A* AND LARGER G VALUES	1190
AVERAGE NODES EXPANDED ON SUCCESS USING FORWARD REPEATED A* AND SMALLER G VALUES 1	1002
AVERAGE NODES EXPANDED ON SUCCESS USING BACKWARD REPEATED A* AND SMALLER G VALUES	1285
AVERAGE NODES EXPANDED ON SUCCESS USING ADAPTIVE A* AND LARGER G VALUES	871

NUMBER OF FAILURES 14
Failures Table

AVERAGE NODES EXPANDED ON FAILURE USING FORWARD REPEATED A* AND LARGER G VALUES	6
AVERAGE NODES EXPANDED ON FAILURE USING FORWARD REPEATED A* AND SMALLER G VALUES	6
AVERAGE NODES EXPANDED ON FAILURE USING BACKWARD REPEATED A* AND LARGER G VALUES	6517
AVERAGE NODES EXPANDED ON FAILURE USING BACKWARD REPEATED A* AND SMALLER G VALUES	6517
AVERAGE NODES EXPANDED ON FAILURE USING ADAPTIVE A* AND LARGER G VALUES	5

Fast Trajectory Replanning Using Forward and Backward Repeating A* and Adaptave A*

Works Consulted:

1. https://github.com/mikepound/mazesolving/issues/13
2. https://www.cs.nmsu.edu/~wyeoh/docs/publications/aamas08-gaastar.pdf