Assignment 1 : Fast Trajectory Replanning
Eric He, Bandy Wang
Due: October 14, 11:55pm

Part 1a

Let the agent start at cell E2 as shown in figure 8. In the first iteration of A*. After running ComputePath() for the first time, the OPEN list contains the following:

Cell	h , g value	f value
E3	2,1	5
D2	4,1	5
E1	4,1	5

Since E3 has the lowest f value, it gets popped out and we expand to E3. At this point, the algothrium is not aware of any blocked cells between the agent and the goal. So E3's neighbors E4 and D3, will be treated as open cells and placed into the OPEN list. After ComputePath() is completed, the computed best path will be

$$E2(A) \rightarrow E3 \rightarrow E4 \rightarrow E5(T)$$

This explains why A* expands east rather than north.

Part 1b

Since A* has a CLOSED list, the algorithim will never expand to a cell that it had already expanded before. Assuming that the gridworld is finite and the target is reachable, the target is always reached because worst case scenario all of the cells will be traversed once and the target is once of them.

If the target is unreachable, the agent will explore all cells that it has access to. The agent wll realize that the are no more cells to be explored once the OPEN list is empty, and know that the target is unreachable if the target is never reached at that point.

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, let n represent the number of unblocked cells in a given gridworld. In a single iteration of A*, the number of moves m made by ComputePath() is at most n (an example of m = n is if the gridworld is a 1 x m with no blocked cells). Thus:$m \leqslant n$.

Once a best path is computed, the worst case is that the block will move always move only one step on the path, then realize that there is a blocked cell and must compute a path using a neighbor cell that it has not gone to before. There is a case where the agent only moves one step down the path in each iterations, and has gone down all possible unblocked cells before reaching the target. Letting a be the number of A* iterations, we can state that $a <= n$.

Taking both inequalities, we can combine to get $am \leqslant n^2$. am represents the total moves that the agent makes in all iterations of A* (hense, Repeated Forward A*), and it is bounded above by the number of unblocked cells squared and thus completing the proof.

Part 2 The Effects of Ties [15 points]: 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

We gathered our observations by including a counter for expanded cells for each A star algorithm. Whenever a cell is added to the closed list, we increment the counter by running each A star algorithm on 50 different mazes but with constant start and goal cells we can analyze the speed of each algorithm in terms of expansions.

We observed the following run times for each tie breaker.

large G	Small G
2202	188825

We see that tie breakers in favor of large g values are significantly faster than the other. This is within our expectations because tie breakers in favor or larger g values means that A star search is choosing to expand cells that are further away from the current position of our robot. We remind ourselves that the calculation of f values is g value + h value. G values represent the distance the current cell is from our robot. H values are the manhattan values, our heuristic for estimating the distance of the current cell to the goal cell. This means there is an inverse relation between g values and h values. Given many cells in the open list with the same f values, we wish to choose cells with high g values because that implicitly tells us the cell has a low h value. Thus, by breaking ties in favor of high g values, A star search effectively chooses cells that are estimated to be closer to the goal cell.

Part 3

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.

Given the same start cell, goal cell and 50 different 101x101 mazes we observe the following averages for expanded cells:

Forward A Star average expanded cells: 1334
Backward A Star average expanded cells: 3972

Between Forward A star and Backwards A star, Forward A Star expands significantly less cells than Forward A Star. These results are within expectations since Backwards A Star does not make use of where the robot currently is. Whenever Backwards A Star computes

a path, the path is computed starting from the goal position for every iteration of compute path. Because A star search is a greedy algorithm, it first puts cells near the goal state into the open list with f values calculated. Now let's assume A star search is midway in computing a path and encounters a blocked cell that it remembers. The f value of nearby valid cells of the currently expanded cell will increase by 1 due to the discovery of the block and then A star search will choose the next lowest f value in the open list. However the next lowest f values belongs to the cells near the beginning of the compute path which is the goal position. This means the search is forced to calculate a path virtually from the beginning again and thus backwards A star explores more cells than needed.

A Star Version	Maze 1	Maze 2	Maze 3	Maze 4	Maze 5
Forward A*	728	1277	1047	1018	1171
Backwards A*	4432	5962	4395	8103	3220

Part 4

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.

Assume that Manhattan distaces are not consistent in gridworlds for the sake of contridiction. It is also implied that the cost to perfrom any action in this world is uniformed. In this gridworld, there exists a cell c such that

$$h(c) > c(succ(c, a)) + h(succ(c, a))$$

where $c(succ(c, a))$ is the action cost to go from cell c to its successor using action a.

By moving $h(m)$ to the left side, we have the following:

$$h(c) - h(succ(c, a)) > c(succ(c, a))$$

The left side of the inequality represents the Manhatten distance between c and $succ(c, a)$. The inequallity implies that the Manhatten distance betwewen c and $succ(c, a)$ is greater then the cost to get from c and $succ(c, a)$. However, this is not possible in best case sceniro, both the cost value and Manhatten distance are both equal. The only way such inequality exist if if the agent can travel diagonally, which is not possible in this gridworld. Thus, a contridiction as occured and it is proven that the Mahattan distance is consistant.

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

Assume that the h-values are initally connsistent such that for all cells c in the gridworld, it is true that $h(c) \leqslant c(c, a) + h(succ(s, a))$. Let $c'(c, a)$ denote the increase in action cost such that $c(c, a) \leqslant c'(c, a)$. Thus, it follows that

$$h(c) \leqslant c(c, a) + h(succ(s, a)) \leqslant c'(c, a) + h(succ(s, a))$$

$$h(c) \leqslant c'(c, a) + h(succ(s, a))$$

Thus the h-values are still consistent even after the action costs increases.

Part 5

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.

Given the same start cell, goal cell and 50 different 101x101 mazes we observe the following averages for expanded cells:

Forward A Star average expanded cells: 1334
Adaptive A Star average expanded cells: 1310

Here we also observe the expanded cells for 5 mazes:

A Star Version	Maze 1	Maze 2	Maze 3	Maze 4	Maze 5
Forward A*	728	1277	1047	1018	1171
Adaptive A*	728	1247	1041	1013	1172

We observe that adaptive A star is only slightly faster than forward A star. Based on the 5 mazes listed, we can see that adaptive A star usually either computes the same amount of expanded cells or only slightly less. These results are within expectations since adaptive A star's heuristic makes use of the previous iteration A star search history which means adaptive A star should be faster. Adaptive A star's new heuristic takes advantage of the previous A star search iteration by taking a cell's old g value and subtracting it from the goal state's old g value. By using this new heuristic adaptive A star is implicitly remembering where blocked cells are because it correctly assumes that the previous iteration of A star search must have calculated the shortest path available with known blocked cells taken into account. Not only is this heuristic more accurate, it is also never smaller than the old heuristic, manhattan values, because taking into account where blocked cells are will only increase f values. Thus, adaptive A star effectively makes use of information from the last A star search to guide its decision in choosing which cell to expand next.

Part 6

In our implementation of the 101 by 101 gridworlds, we have a 2d integer array(denoted as array a) that represents the world, with elements as 1 as blocked cells and 0 as open cells. We also have a seperate integer 2d array (denoted as array b) to store the blcoked cells that the agent has knowledge of. To store the g and f values, we have a seperate 2d array (denoted as array c).

One way we can reduce the memory consumption is to use boolean 2d arrays instead of an integers to represent the first two 2d arrays.

Calculate the amount of memory that they need to operate on gridworlds of size 1001*1001

Using this new way, we can calulate the memory usage for a 1001 * 1001 with the folowing:
Array a: 1001*1001*1 = 1,002,001 bytes
Array b: 1001*1001*1 = 1,002,001 bytes
Array c: 1001*1001*8 = 8,016,008 bytes
Total = 10,020,010 bytes = 10.02001 Mb

Calculate the largest gridworld that they can operate on within a memory limit of 4 MBytes

Under the new memory implementation, each cell would need 10 bytes. Thus, using only 4Mb, the number of cells that the gridworld can hold would be

$$\frac{4 * 1024}{10} = 409.6$$

This would be roughly equal to a 20*20 gridworld.