Intro to AI Project 1

Tim Raymaker and Mike Rizzo

July 17, 2019

1 Part 1

1.1 a.

The agent will move toward the cell with the lowest f-value. The f-value is determined from the sum of the cost from the start, or g-value, and the heuristic value to the goal, or h-value. For the first move, each possible direction is one tile away from the start, meaning that the g-value for each move will be the same, and differences in the h-value will be the deciding factor on where to move. Since the agent does not know that cells D3 and E4 are blocked, it will move to the east, the direction that leads to the cell with the lowest h-value and therefore the lowest f-value.

1.2 b.

The agent will continue to act as long as the open list remains populated. The open list is created from cells adjacent to expanded cells. This means that any cells that is adjacent to a reachable cell will be expanded eventually, until the target is reached. Therefore, if an unblocked path from the start state to the end state exists, the agent will eventually find it, even if it has to expand every possible cell. The grid worlds are finite in size and so there is a finite number of cells that will be added to be expanded. Expansion can be done in a finite amount of time and traversal can be done in a finite amount of time. Considering both processes can be done in finite time and there are only finite cells to iterate over, in the scenario given a path can be found in finite time.

2 Part 2

Tiebreaking was important in the planning phase of the A* search. Low-g tiebreaking meant prioritizing tied neighbors that were closer to the start in the heap. High-g meant the opposite. From our data low-g was able to find very slightly shorter paths in slightly shorter times than high-g was. This seems counterintuitive, given that in the example problem a high-g search will drastically outperform low-g by cutting straight to the goal where low-g explores other nodes, but it is worth considering that high-g's worst case was often worse than low-g's. High-g searches favored diving toward the goal, which for an uncluttered map is optimal, but our map had enough obstacles that every search was significantly hampered. High-g, in its attempt to 'greedily' find a solution as fast as possible, sometimes ended up trapped in dead end areas with no way forward, being forced to backtrack significantly. Low-g, on the other hand, performed a slower but safer search. In this example low-g is more akin to BFS while high-g played a similar role to DFS. Of course, the average and best case scenarios of high-g are much better than those of low-g. With more maps and more data, perhaps a higher density of average cases would have shown high-g to perform better overall.
To see average runtimes as well as average solution length, see end of report.

3 Part 3

The Repeated Forward A* search and Repeated Backward A* search were both able to find possible solutions where solutions existed, but their exact paths differed in some cases. This is likely because that although they solved the same maze, each one had different information available to them as they could only see cells near them. As such, their differing starting locations led them down different paths. Neither method's path was consistently superior to that of the other, but the backwards paths tended to have worse runtimes, perhaps due to the fact that we only animated the projected paths of A* in the first quadrant. Although the forward searches would have longer projected paths, the amount of replans for the backward searches was likely higher in the first quadrant as the searches neared their goal and the amount of available moves became more limited.

4 Part 4

Manhattan distance can be described as the number of moves required to reach a goal given no obstacles. Therefore, it makes sense that said distance

will be consistent. If an h-value in adaptive A* becomes inconsistent, it is because something is in the way. In this case, the heuristic will be updated and will become consistent again.
Given a node s and neighbor m:

5 Part 5

Our adaptive A* method was able to find shorter paths than forward A* on average, but it also took a longer time. This is likely due to it attempting to avoid previous projected paths and move in new directions. Our A* recalculated the heuristic value of every tile that was expanded by the previous A* search upon replanning, increasing their cost by using g(sgoal)-g(s) and incentivising the algorithm to explore alternatives. By doing so it was sometimes able to find a better path at the cost of runtime. Ties were broken randomly and each search weighed in favor of larger g values.

6 Part 6

Our tree pointers pointed to a tuple coordinate, but changing them to point in a specific direction would decrease memory costs. Changing the method of visualizing the path of the agent would also likely yield a reduction in memory consumption. In order to illustrate the projected paths of the algorithm, we repeatedly filled in lines of boxes using pygame. Eliminating unecessary animation would streamline the program. The binary heap and use of tuples and Python dictionaries was fairly cost effective. Improving upon the A* algorithm outlined in the project description could also reduce runtime and memory usage. Plotting the entire path of the agent with each replan is inefficient when the agent is usually stopped after only a few moves. Planning a shorter path and replanning if necessary might be helpful.
(1,001 x 1,001) = 1002001 cells
At 2 bits per cell:
2 * 1002001 = 2,004,002 bits   250 kB

4MB = 32000000 bits
At 2 bits per cell:
16000000 cells possible
grid size = sqrt(16000000) = 4000 x 4000 grid

7 Data and Observations

Every search method was fairly similar in terms of the best path found, with low-g backward A* being an outlier with somewhat higher path length. In terms of search time, the forward A* search methods were the fastest, followed by adaptive A*, followed by the backward A* methods. It is unclear as to how much of this variance can be attributed to small sample size. Furthermore, each search method uses high or low g tiebreaking, but if g values are also the same, they determine the next tile to expand randomly from the pool of eligible tiles. This can lead to variance in the paths taken even by the same search on the same map. As such it is difficult to confidently make conclusions from the 50 maps we tested. That being said, the conclusions we made and the reasoning behind these are listed above.