Group 27 Report

July 14, 2020

1 Part 1A:

The program deals with implementing A* graph search with consistent heuristics, where a consistent heuristic h(n) ≤c(n,a,n') + h(n'). We know this program will be trying to find the most optimal cost to get from the start node to the goal node. We calculate the heuristic values by Manhattan distance. As a result, the heuristic value for cost-to-go, g(n), added to the heuristic value for cost-to-go, h(n), will have a smaller sum, f(n), for the state in the east at E3 rather than the state in the north at D2. The A* algorithm will prefer moving to the smaller value of f(n), from the open list of states that was derived from expanding E2 which is the start state. Thus, the agent moves to E3 since f(E3)= 1 + 2 = 3 which is less than f(D2) = 1 + 4 = 5 for state D2. And does this regardless of initially knowing which states are blocked.

2 Part 1B:

In this program, the agent will either be able to reach the target state or not reach the target state from the starting state. The reasoning is the there may or may not be enough blocked states/cells that prevent the agent to circumnavigate to the target state. As a result, the agent/program will realize that it is indeed able reach the target or to discover that it is impossible to reach the target in finite time. For instance, in the gridworld there will be n total cells where x number of cells are blocked. Likewise, to find the number of unblocked cells we take the total number of cells and subtract from the number of blocked cells since cells can only be either blocked or unblocked. So, n - x = (number of unblocked cells, y). When we run the A* algorithm through either our Repeated Forward or Repeated Backward A* algorithms, we see that two outcomes result from it; the agent will reach or not reach the target cell. In the worst case the number of cells the agent will need to come across would at most be the total number of unblocked cells in some edge case. Thus, the number of movements, m, made by the agent would be m ≤y. The openList can only contain unblocked cells since they are the only type of cell to be traversed on by the agent since we checked to prevent putting children cells that happened to be blocked. Every iteration of checking the openList until all

unblocked cells in the gridworld are checked would be the worst case to find the target cell.This means that every iteration, i, that checks from the openList that the agent moves towards will be at most y as well, so i $\leq$y. Hence, the number of movements caused by each iteration of checking against the openList will be (i * m) and since every iteration of checking the openList would indicate exactly one movement of the agent and would be at most the number of unblocked cells squared. Thus, the number of moves made by the agent until it reaches the target cell or realizes that it is impossible is bounded by the number of unblocked cells squared, so (i * m) $\leq y^2$.

3 Part2:

In our implementation of Repeated A*, we chose to break ties based on cardinal directions. In the event the multiple neighbors of a node had the same f-value, then we would expand the neighbor with the lowest g-value. In the event that two or more neighbors had the same f-value and g-value, then it chooses which to expand next based on cardinal direction. It would prioritize the North neighbor first, then the East, then the South, and then the West.

4 Part 3:

Our implementation of Repeated Forward A* and Repeated Backward A* were very similar in how they checked neighbors on cell expansion. They both followed the same order, where first the North child was checked, then the East child, then the South child, and the West child is checked last. As a result, mazes where the target cell was Northeast relative to the start cell had better run times than mazes where the target cell was Southwest relative to the start cell. Due to the random generation of the 50 mazes, the relative positions of the start and end could be anywhere. Therefore, we observed that the average runtime of Repeated Forward A* was identical to the average run time of Repeated Backward A*.

5 Part 4:

The project argues that Manhattan distances are consistent, or where h(n) $\leq$ h(n') + (n,a,n') and h(x_{Goal}) = 0. This consistent heuristic is defined as the estimated distance from the node n to the goal must be less than or equal to the estimation of its successor n' plus the action cost of the transition from n to successor n'. In addition, the heuristic value of the goal must also equal 0, h(x_{Goal}) = 0. This illustrates the triangle equality. We need to inductively prove that the Manhattan distance is responsible for a consistent heuristic. To do this, we need to prove that n is the goal as the base case. Thus, we need the calculation for the manhattan distance of both and h(s) and h(s_{Goal}). This would be, h(s) = $|x_s - x_{sGoal}| + |y_s - y_{sGoal}|$, where x and y are the indices of the rows and columns respectively. Likewise, h(s_{Goal}) = $|x_{sGoal} - x_{sGoal}| + |y_{sGoal} - y_{sGoal}| = 0$. For n to be the goal, their heuristic value needs to be equivalent.

In this case, 0. So here, h(s) = h(s_{Goal}) = 0. Now if we are to prove if s ≠s_{Goal} we can show that since the grid world is maneuverable by cardinal directions . To do this we see h(s) + h(s') = ($|x_s - x_{sGoal}| + |y_s - y_{sGoal}|$) - ($|x_{s'} - x_{sGoal}| + |y_{s'} - y_{sGoal}|$). We know that s and s' cells are neighbors, so we can extrapolate that Xs = Xs' or Ys = Ys'. If Xs = Xs', then $|x_s - x_{sGoal}| and |x_{s'} - x_{sGoal}$ are equal and will be able to be canceled out in the equation for h(s) + h(s'). Thus, $|Ys - Ys'| = 1$. This means that the action cost is one when we calculate for the heuristic for the successor of n, which would be n', and the successor for n', which would be n".As a result, h(s) + h(s') = $|y_s - y_{sGoal}| - |y_{s'} - y_{sGoal}|$ where if we have every successor in line till we find the target cell, we see that the f-value will result increase by one, and shows that it is monotone, even though the heuristic value of a cell is decreasing as it gets closer to the target cell. Thus, we see that h(n) ≤h(n') + (n,a,n') and h(x_{Goal}) = 0. And this tells us that Manhattan distance is a consistent heuristic.

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.

The h-values in Adaptive A* are not Manhattan distance and are newly calculated h-values are updated from the node that has been expanded from the previous A* search. So the new Adaptive A* heuristic is $h_{new}(s)$ = g(s_{Goal}) - g(s). There are going to be three cases where the new h-value can be calculated, when both s and s' have been expanded in a previous A* search, when s had been expanded but s' has not, or when neither s nor s' has been expanded. In the first case where both s and s' have been expanded in a previous A* search, we see that $h_{new}(s)$ = g(s_{Goal}) - g(s) and $h_{new}(s')$ = g(s_{Goal}) - g(s'). In addition, we want to prove that our conclusion $h_{new}(s)$ ≤$h_{new}(s')$ + c(s,a,s'). To do this we know g(s_{Goal}) - g(s) ≤g(s_{Goal}) - g(s') + c(s,a,s'), which would be equivalent to g(s') ≤g(s) + c(s,a,s'). This statement is true because g(s') is less than or equal to g(s). So when s' is being expanded with s being the parent g(s') = g(s) + c(s,a,s'). But, if s' were to be expanded with n being the parent, then s cannot be the parent of s'. This will result in having g(s') ≤ g(s) + c(s,a,s'). This means that g(s') can be at most what is on the right side of that equation. As a result, we see that the h-values $h_{new}(s)$ are not only admissible, but also consistent. Thus, proving that Adaptive A* leaves the initially consistent h-values consistent even if action costs can increases.

6 Part 5:

Cheese for this part, please.

7 Part 6:

Suggest additional ways to reduce the memory consumption further:

There are a number of ways that we could have better optimized our code. For one, we stored a cell's coordinates as 2 separate attributes, rowNum and colNum. We could have simply used a tuple to store the coordinates as a single variable. Additionally, there were many times where we used variables to store values that could have been stored as static values instead. For example, the totalRows and totalCols variables.

Calculate the total amount of memory needed to operate on a grid of size 1001 × 1001:

Each cell in our implementation has a total of seven attributes, six integers, and a pointer. Therefore, each cell uses 6*4+8 = 32 Bytes. As a result, a grid of size 1001 × 1001 would use 1001*1001*32 = 3,206,432 Bytes = 32.064032 MegaBytes (MB).

Calculate the largest grid world that they can operate on within a memory limit of 4 MegaBytes (MB):

Each megabyte is equal to 1024 Bytes, so the total amount of memory the 4 MegaBytes can store is 4096 Bytes. Therefore, with that memory constraint we can create a total of 4096/32 = 128 cells. Does, the largest grid world that can be created is an 11 x 11 grid.