COMP 3106 Assignment 1 Technical Document

Braeden Hall (101143403)

October 17, 2021

Question 1.

Data Structures:

- Graph: Stores information about the $m \times n$ grid received from the file. Including start state, goal state, number of rows and columns and of course the grid itself. The grid is implemented as a 2-dimensional array where each individual index is a Node type.
- Node: A node is created for each individual index in the Graph's grid. It stores its location in the grid (y^{th} row, x^{th} column) and its heuristic value. It also stores information about its parent node and its current path cost, these properties are updated as the A^* algorithm progresses.
- SquareType: This is an enumerated data type. It is meant to clearly define values for the different cell/square types in the grid, i.e. start state, goal state, wall, hazard etc.

My Implementation:

My implementation starts by reading the input csv file and creating a Graph with the data. For every square in the grid, the Graph will create a new Node of the specified type (hazard, goal state etc.) and store it in the grid in the correct index.

After the entire grid is constructed (and the goal state has been found), the Graph will loop through the grid and set the heuristic value for every regular Node to the Manhattan distance between that Node and the goal state. The exception to this is for walls, hazards and squares adjacent to hazards. The heuristic value for these nodes is set to "infinity" (`float("inf")` in the code) to ensure that they have the lowest possible priority. For this assignment, we are assuming that there is always a path from the start state to the goal state. For this reason there will always be at least one Node n on the frontier with higher priority ($f(n) < \infty$) than these "infinite" nodes. Therefore, we know that these nodes will never be explored and can never exist on the shortest path.

The A^* Algorithm:

I implemented the A^* algorithm pretty much exactly as seen in class. The main difference is that the frontier is not implemented as a priority queue. Instead, it is simply a list, and I sort it (least to greatest) by the priority function (path cost + heuristic value) each time through the loop before choosing the leaf. I understand this is not necessarily the most efficient way of implementing this, however, I could not find a standard implementation of a sorted set in Python where I could customize the comparison function.

Another design choice worth mentioning is how I implemented the getNeighboors function. This is a method on the Graph data structure that takes a Node and outputs a list of the nodes adjacent (left, right, up, down) to the given node and performs error checks. I also used this function when getting the nodes adjacent to hazards.

Finally, the optimal path is constructed after the while loop terminates by following the parent pointers from the final leaf (which must be equal to the goal state for the loop to terminate) all the way back to the start state.

Question 2.

The agent I have implemented is a model-based reflex agent. I believe this because the agent keeps track of the an internal state (nodes in grid it has already explored, the cost to get from the start state to any particular node, heuristic value of a particular node etc.) and makes decisions about the next node to explore based on this state. This is different from any other agent type because this agent only considers its model of the current environment state when making decisions.

Question 3.

I believe that the most informed heuristic to use in this environment is the Manhattan distance between a node and the goal state.

Consistency:

Manhattan distance is calculated as

$$h_M(node) = |node.x - goal.x| + |node.y - goal.y|$$

Let us first note that $h_M(goal) = |goal.x - goal.x| + |goal.y - goal.y| = 0$. Thus, we know that $h_M(goal) = 0$ which is needed for consistency.

Next let us consider an arbitrary node n and an arbitrary successor of n, n' on the shortest path to the goal state. We know that $cost(n, n')$ is some positive integer. Since we cannot move diagonally on the grid, and each "step" has a cost of 1 we know that $cost(n, n')$ is equal to the Manhattan distance between n and n'. As stated previously, the heuristic value $h_M(n)$ of a node is the Manhattan distance between the node and the goal state and we know that the node n' is closer to the goal state than the node n. So we have:

$$\begin{aligned} h_M(n) &= |n.x - goal.x| + |n.y - goal.y| \\ &= (|n.x - n'.x| + |n.y - n'.y|) + (|n'.x - goal.x| + |n'.y - goal.y|) \\ &= cost(n, n') + h_M(n') \\ &\leq cost(n, n') + h_M(n') \end{aligned}$$

Therefore, the heuristic is consistent.

Informedness:

Another heuristic that could have been used in this environment is the euclidean distance, otherwise known as the straight line distance between a node and the goal state. The equation for this is

$$h_E(n) = \sqrt{(n.x - goal.x)^2 + (n.y - goal.y)^2}$$

As it was stated in class (lecture 6 I believe), for all nodes n in a grid the Manhattan distance $h_M(n)$ is greater than or equal to the euclidean distance $h_E(n)$. Therefore, the Manhattan distance is a more informed heuristic.

Question 4.

Example:

O,O,O,O,O
O,O,O,O,O
O,O,S,O,O
O,O,O,O,O
O,O,O,O,G

A^* search will find the optimal path faster than Uniform Cost Search (UCS) on this grid because UCS does not take into account distance from the goal (the heuristic A^* uses) when choosing the next node to explore. UCS will explore all nodes adjacent to S before looking at nodes adjacent to those because nodes adjacent to S will have a lower path cost.

However, because A^* takes into account the heuristic, it will explore closer to the goal state first. It won't bother to explore the nodes in the opposite direction of the goal since their heuristic value will be so high that it will almost overwrite the small path cost. Therefore, it will find the optimal solution faster than UCS.

Question 5.

Example:

S,O,O,O,O
O,X,X,X,O
O,O,O,X,O
X,X,O,X,O
O,O,O,X,O
O,X,X,X,O
O,O,O,X,O
X,X,O,X,O
O,O,O,X,O
O,X,X,X,O
O,O,O,X,O
X,X,O,X,O
G,O,O,O,O

Greedy Heuristic Search (GHS) will not find the optimal solution on this grid because it will always explore the path down the grid since those nodes are closer to the goal state (as determined by the heuristic value). GHS will never explore the node horizontal of the start state because it is farther from the goal state, even though that path will end up having a lower cost in the end.

Question 6.

The strategy you should use to break ties is to choose the node that is closer to the goal state, i.e. with the lower heuristic value. This will lead you to getting closer to the goal state faster and it may lead to a shorter run time for the algorithm (assuming there are no hazards or walls between the goal state and the chosen node).

Question 7.

If the agent could make diagonal moves, then the best heuristic to use would be euclidean distance. This is because if you allow diagonal moves, the shortest distance between 2 points in the grid (that are not on the same row or column) will involve diagonal movements. Assuming that the cost of diagonal moves is not much greater than the cost of horizontal or vertical moves (I assume the cost would be approximately $\sqrt{2}$) then the previously mentioned heuristic (Manhattan distance) will be inconsistent.

The euclidean distance will obviously be consistent because the shortest distance between to points is always the straight line distance between them. This will always be less than or equal to the actual distance because it may be the case that it is not possible to travel directly towards the goal state, you can not travel form (0,0) to (1,2) in one step for example. This means you may have to travel in a horizontal, vertical or diagonal direction first, and this will lead to the path cost increasing to more than the straight line distance. Which means that $h_E(n) \leq cost(n, n') + h_E(n')$ for all nodes n and successors n'.