CS440 - hw1

Thomas Burns, Jesse Lin

October 14, 2019

Part 0

All code files for this assignment are included in the submission file named: hw1_code_submission.zip.

Part 1

(a) Explain why the first move of the agent for the example search problem is to the east rather than the north given that the agent does not know initially which cells are blocked.

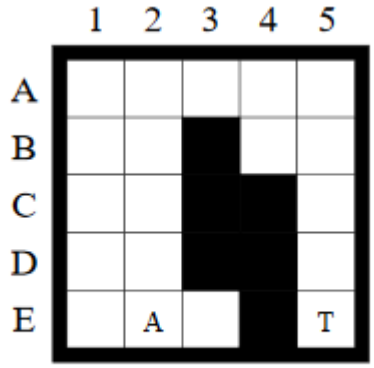

Figure 8: Second Example Search Problem

(b) Give a convincing argument that the agent in finite gridworlds indeed either reaches the target or discovers that this is impossible and is bounded from above by the number of unblocked cells squared.

(a) In the agent's current position, there are 3 directions which the agent can go: west, north, and east.

h(n) = Manhattan distance to T
g(n) = cost so far to reach T
f(n) = h(n) + g(n)

West: h(n) = 4, g(n) = 1, f(n) = 5
North: h(n) = 4, g(n) = 1, f(n) = 5
East: h(n) = 2, g(n) = 1, f(n) = 3

Because the agent is not aware of the obstacles, it will go East and only then will it realize that this cell does not lead to a successful path and will thus expand on the other nodes in the queue.

(b) The A* algorithm, in worst case scenario, has the same run time complexity as breadth first search (BFS). Since the big O for BFS is better than O(

$$n^2$$

), at most the algorithm will

Part 2

Our forward A* always found an optimal path more efficiently when tie breaking with a bias on higher g values. The runtimes of repeated forward A* algorithm with higher g value tie breaking averaged to about 5 seconds when a path existed from the agent to the goal. This runtime escalated to about 60 seconds when no goal path was possible. For forward A* where tie breaking was done with bias to lower g values, the runtime of the algorithm algorithm was anywhere from 400 to 800 seconds. This runtime was harder to average because an accurate runtime average requires multiple runs and the algorithm took such a long time to find an optimal path.

The reason for this observation is that when tie breaking with a bias on lower g values, the algorithm is essentially performing a BFS (breadth first search) on the board. It will expand to all nearby cells before expanding outward toward where the potential goal is on the board. When tie breaking with a bias on higher g values, the algorithm behaved similarly to a DFS (depth first search). The agent would expand outward in an attempt to find an optimal path before expanding outward in terms of breadth. This resulted in a much more accurate path finding algorithm for our agent on a 101x101 grid, as can be seen from the runtimes we recorded.

Part 3

Our implementation of forward A* always ran faster than backward A*. The runtime of forward A* averaged out to about 5 seconds when a path from the starting point of the agent to the goal existed. When the path was completely blocked the runtime sometimes escalated to about 60 seconds. For backward A* the runtime averaged to about 300 seconds.

This can be explained by the fact that for backward A* it is moving backward from the goal to the agent. The surrounding area of the target is not known

to the algorithm. This means that multiple cells will have similar f values and each of them might be expanded on while finding the optimal path. For forward A* the agent knows the surrounding area and is able to more optimally choose a path based on the seen obstacles and the f values it calculates for each coordinate on the board. Less paths need to be explored for the algorithm to find the optimal path for the agent.

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.
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.
To prove that the Manhattan distance heuristic is consistent, we will first allow movement in 8 directions. Then the competing heuristic is the Chebyshev heuristic. The Manhattan distance heuristic is defined as:

$$h_{manhattan}(n) = |x_1 - x_2| + |y_1 - y_2|$$

Chebyshev distance is defined as:

$$h_{chebyshev}(n) = min(|x_1-x_2|, |y_1-y_2|)+(|x_1-x_2|+|y_1-y_2|)-2*(min(|x_1-x_2|, |y_1-y_2|))$$

For Chebyshev heuristic worst case it matches Manhattan heuristic, and in best case it will reduce the heuristic function value significantly. However, if we constrain the movement to the four cardinal directions, the Manhattan heuristic is easier to compute than the Chebyshev heuristic. Thus this shows how the Manhattan distance is only consistent if the agent is allowed to move in four directions, as it cannot compete when multiple directions are added.
Furthermore, Adaptive A* also leaves initially consistent h-values consistent, even as action costs increase. Adaptive A* works by having an already run Repeated Forward A* search on the maze and having the final cost of its search, defined as gd(n) = cost_of_last_search, the value of which is constant. We then run Adaptive A* with the renewed heuristic function of:

$$h_{new}(n) = gd(n) - g(n)$$

This heuristic is consistent and admissible because in worst case scenario, it matches cost of the previous search (which is consistent).
We can show this using the following inequalities:

$$f(s) \geq g(s) + h(s)(g(s) = costatstates, h(s) = Manhattandistancetotarget$$

$$f(s) \leq gd(s)$$
$$g(s) + h(s) \leq f(s) \leq gd(s)$$
$$g(s) + h(s) \leq gd(s)$$
$$h(s) \leq gd(s) - g(s)$$
$$h(s) \leq h_{new}(s)$$

Thus the heuristic cost is consistent as it remains less than or equal to the previous heruistic cost. Adaptive A* guarantees that it will at worst match the performance of Repeated Forward A* even if action costs increase.

Part 5

Our implementation of Forward A* always ran faster than Adaptive A*. The runtime of forward A* averaged out to about 5 seconds when a path from the starting point of the agent to the goal existed. For adaptive A*, runtime was similar to backward A* with a bias on higher g values, running between 60 and 400 seconds. Our adaptive A* algorithm ran similarly to forward A*, but upon each new path calculation, the g values of all values on the path were recalculated and updated on the board. This caused later path decisions to be made on those previous calculations. This caused more ties amongst f values and more paths to be explored compared to the more straightforward forward A* with no g value updates. The observation of our runtime of our adaptive A* algorithm reflects this conclusion.

Part 6

Our implementation of the A* algorithms was completed in Python, but if we wanted to be more memory efficient we could have done this project in a more memory efficient language like C (if implemented correctly). In C, a struct could be used which contains a Coordinate, parent pointer, neighbor pointer, g value, f value. This struct would add up to about 24 bytes because of the way C does memory cutoffs in byte blocks of 8 or 16. From testing of our implementation of forward A* we know that about 4% of coordinates are used in path traversal. That means we could have about 300,000 coordinates. A 1001x1001 grid would cost about 3.5 MB, and memory allocations and deallocations could be made appropriately.