Introduction to Artificial Intelligence CS440: HW1 Analysis

Angeline Jacob and Stephen Thomas

02/27/19

In this assignment we analyze and compare 3 fundamental search algorithms, Forward A*, Backward A* and Adaptive A*

Forward A* vs. Backward A* vs Adaptive A*				
	Forward A* (tie breaking with smaller g values)	**Forward A* (tie breaking with larger g values)**	**Backward A***	**Adaptive A***
Average Number of Expanded Cells	3798.14 cells	2927 cells	4121.72 cells	3161.18 cells
Average Run Time	0.10s	0.14s	0.18s	0.08s

1. **Understanding the Methods**

   (a) The first step of the agent in the search problem in Figure 8, is to the East. This is because, A* first plans a path from the starting state to the goal state without considering blocked cells. Thus, when considering the neighbors from the start state, the algorithm will chose the cell with the smallest f value, in this case it is the cell to the East. The heuristic for the cell to the east is lower than the other cells, (since its closer to the goal state and has a smaller Manhattan Distance) and will therefore have a smaller f value.

   (b) Since our agent is in a finite grid world, in the worst case, it will have to visit all the unblocked cells at least once. Once all the unblocked nodes have been visited, either the goal has been visited or the target is in the open list or the target is blocked off and is not in the open list. Once the list is empty we know that there is no solution to the maze. The reason the upper bound is the number of unblocked cells squared is because, each of the unblocked cells will go in the list, and then (worst case) we iterate through the list and expand each of the nodes, this number would equal the number of unblocked cells squared.

2. **The Effect of Ties: Smaller G Values vs. Larger G. Values**

   We ran repeated forward A* in 50 grid-worlds twice, once favoring smaller g values and one favoring larger g values in the instance of a tie. As shown above, we observed tie breaking with smaller g values to be faster than tie breaking with larger g values. Additionally, the number of expanded cells are higher with smaller g value tie breaking compared to the number of expanded cells with larger g value tie breaking. These observations hold true considering, choosing a cell with a smaller g value allows A* to explore the state space more and potentially expand more cells while choosing a cell with a larger g value will cause A* to expand less cells with a higher potential time complexity as it visits nodes that have been previously visited. Favoring cells with larger g values will help the optimality of A* as it expands fewer cells but risks repeating paths that do not lead to the goal state.

3. **Forward Vs. Backward A***

   When comparing Forward and Backward A*, breaking ties at higher g values, we observe that one algorithm does not dominate the other indefinitely. However, if the goal state is densely surrounded by blocked cells, Backward A* will more easily navigate out of it since its goal state is outside the area and will not repeat A* as many times as adamantly as Forward A* as it nears its goal. The same holds true in reverse. Forward A* will have a easier time navigating out of a densely blocked off area near the start state vs the goal state. Hence, depending on the nature of the grid either Forward or Backward A* will dominate.

4. **Heuristics in the Adaptive A***

   The Manhattan distance is the sum of horizontal and vertical moves between two points. Thus, that no matter the combination of horizontal and vertical steps, the Manhattan distance will always be the same. Essentially, having the Manhattan distance as your heuristic will never overestimate the cost of getting to the goal node. This of course only works for grid worlds where you're moving in the 4 main compass directions only. If you were allowed to move diagonally, then the heuristic would overestimate the distance from start to finish, thus making it inconsistent.

   The new h value is consistent even if the action cost increases. The triangle inequality must hold true for a heuristic to be consistent. So **$h(n) \leq h(n')$ + $c(n,a,n')$**. Where h(n) is the heuristic for the current node, h(n') is the heuristic for the next node, and c(n,a,n') is the cost from node n to n'. Since this inequality holds, it stands true that h(n) is consistent. In order for H_{new} to be consistent, it must also satisfy this inequality. $\mathbf{h_{new}(n) = h_{new}(n') + c(n,a,n')}$. From here we can substitute for h_{new}. $\mathbf{h_{new}(n) = g(goal) - g(n)}$ and $\mathbf{h_{new}(n') = g(goal) - g(n')}$. From this we get the following:

$\mathbf{g(goal) - g(s) \leq g(goal) - g(n') + c(n,a,n')}$
$\mathbf{- g(s) \leq - g(n') + c(n,a,n')}$
$\mathbf{- g(s) \leq - g(n') + c(n,a,n')}$
$\mathbf{g(s) \geq g(n') - c(n,a,n')}$

From this it is clear that $h_(new)$ is consistent.

If there is an action cost increase from c to c'. Then the inequality changes as such: $\mathbf{h_{new}(n) \leq h_{new}(n') + c(n,a,n') \leq h_{new}(n') + c'(n,a,n')}$

Here we can see that the inequality is still maintained. Therefore h_{new} is consistent even if the action cost increases.

5. **Heuristics in the Adaptive A***

   When comparing Forward A* to Adaptive A*, breaking ties at higher g values, we observe that, for the majority of cases, Adaptive A* runs significantly faster, expanding fewer total nodes. This observation holds true considering Adaptive A* updates heuristic values for previously visited nodes that are potential paths to the solution with a good evaluation function. Updating the heuristic for these nodes will prioritize potential solutions and prevent A* from expanding nodes it does not find viable. This will reduce the total number of nodes expanded and in most cases reduce the time the algorithm requires to find the solution

6. **Memory Issues**

   Additional ways to reduce the memory consumption for this implementation include removing references to each child node from each cell. A more efficient way is to determine the neighbors of each cell as A* goes from cell to cell rather than saving the neighbors for all children, many of which will never be used. By removing references to the children you only use 72 MB in a 1001 x 1001 grid world.

   72 bits * 1001 * 1001 = 72,144,072 bits.

   The largest grid world you can have with a 4 MB limit is 235 x 235. 4,000,000 bits/ 72 bits = 55,555 nodes. Square root of 55,555 is 235 x 235 grid.