Fast Trajectory Replanning

Assignment 1

Robert Kulesa, Michael Li, Daniel Liu

CS440 Fall 2021
Professor Boularias
Rutgers University - New Brunswick
October 15, 2021

Part 0 - Setup your environments

Our gridworlds are initialized using DFS from a random start node. A random, unmarked neighbor node is selected and initialized as unblocked with 70% probability (in which case DFS is called on this neighbor), or an obstacle with 30% probability. DFS is then called on any remaining uninitialized nodes.

Part 1 - Understanding the methods

a) The agent moves east, because the unblocked, unvisited neighbor with the lowest cost $f(x) = g(x) + h(x)$ is the eastern neighbor. Using manhattan distance as $h(x)$, the eastern neighbor has $f(x) = 1+2 = 3$, whereas the northern neighbor has $f(x) = 1 + 4 = 5$. Therefore, the eastern neighbor is selected, and the agent explores the eastern cell.

b) The agent must be able to reach the target cell, or realize it cannot reach it, in finite time in a finite gridworld. A* finishes when the open list is empty, in which case there is no valid path, or when the agent reaches the goal state. The open list starts out with the initial state, and all potential neighboring states are expanded to compute a potential path. Thus, in the worst case, all nodes with a valid path from the initial state will be expanded in finite time. Therefore, the agent must be able to reach the target cell, or discover it cannot reach it, in finite time in a finite gridworld. When ComputePath returns a path for the agent to follow, the agent follows it until it reaches a blocked square, in which case it calls ComputePath again (unless it cannot reach the target). After one ComputePath execution, the agent can move a maximum of n nodes, where n is the number of unblocked cells in the world. ComputePath can be called once for each unblocked cell, so thus, the number of moves the agent makes in the gridworld is upper bounded by $O(n^2)$

Part 2 - The Effects of Ties

For this experiment, we ran forward repeated A* on 50, 101×101 gridworlds generated as specified in Part 0. When breaking ties in favor of smaller g values, the experiment finished in a total of 16.014958 s, at an average of 32.029916 ms per gridworld. When breaking ties in favor of larger g values, the experiment finished in a total of 5.808179 s, at an average of 11.616358

ms per gridworld.

This is because by favoring larger g values, the agent favors exploring states that are further from the start state first, so the agent is more likely to reach the target state in less time.

Part 3 - Forward vs. Backward

This experiment was conducted on the same 50, 101 × 101 gridworlds as specified in Part 2. When using repeated forward A*, the experiment finished in a total of 5.808179 s, at an average of 11.616358 ms per gridworld. When using repeated backward A*, the experiment finished in a total of 22.95945 s, at an average of 45.9189 ms per gridworld.

Repeated forward A* runs much faster than repeated backward A*. This is because repeated backward A* must expand more states closer to the starting state of the agent before the agent is able to traverse far, which ends up wasting time.

Part 4 - Heuristics in the Adaptive A*

Part 5 - Repeated Forward A* vs. Adaptive A*

This experiment was conducted on the same 50, 101 × 101 gridworlds as specified in Part 2 and 3. When using repeated A*, the experiment finished in a total of 5.808179 s, at an average of 11.616358 ms per gridworld. When using adaptive A*, the experiment finished in a total of 5.627615 s, at an average of 11.25523 ms per gridworld.

Adaptive A* improves upon Repeated Forward A* by increasing the heuristic value of nodes expanded by ComputePath(), which refines future searches by lowering the number of states in $OPEN$ which will satisfy

$$g(s_{goal}) > min_{s' \in OPEN} \left(g(s') + h(s') \right)$$

Since fewer nodes are expanded by Adaptive A* than in Repeated Forward A*, Adaptive A* takes less time to find the path, if one exists.

Part 6 - Statistical Significance