Ming Tai Ha (133004530) [mingtha]

Timothy Yong (143009812) [ty114]

**Part 1) Understanding the Methods**

1. In this homework, we are assuming that the cost to move from one node to an adjacent node in any cardinal direction is uniform. Thus, given the starting state n\_s, the estimated cost of the path from n\_s to the goal state n\_g via a successor n is **f(n) = g(n) + h(n) ⇔ f(n) = 1 + h(n)**. In Figure 8 of the HW1 description, the agent moves east since f**(n\_east) = 1 + h(n\_east) = 1 + 2 = 3**, whereas **f(n\_north) = 1 + h(n\_north) = 1 + 4 = 5**. So, the agent moved east since f(n\_east) < f(n\_north)
   1. Consider a finite gridworld. Then there are a finite number of states which an agent can traverse to. When the agent searches for an optimal path, the agent keeps track of possible successors in the open set and visited successors in the closed set. Since the gridworld is finite, the time to discover the optimal other is a function of all the nodes is a function of all the nodes which the agent visits, which is finite. So, if there is a path to the goal state, the agent will require finite time to discover the path. If, however, there is no path to the goal state (i.e. blocked), then the agent will explore all possible nodes to which it has access to (i.e. unblocked) before discovering that there is no path to the goal state. Since there is a finite number of nodes and the A\* algorithm keep track of which nodes have been visited using the closed set, there can be no cycle. Thus, the agent will discover whether it is possible or impossible to reach a goal in finite time.
   2. When an agent is traversing the nodes to which it has access, the agent is creating a minimum spanning tree, where the nodes of the minimum spanning tree indicate the current position and the edges indicate adjacency. Since the agent is given a starting state and has to discover the path to the goal, the minimum spanning tree is constructed in a way similar to Prim’s Algorithm. Thus, the upper bound to discover that there is no path to the goal is O(n^2), where the n is the number of unblocked states. If there is no path between the start and goal states, then the agent must access all of the unblocked states before discovering that there is no path; this is equivalent to constructing a minimum spanning tree using Prim’s Algorithm, and thus is bounded by O(n^2). If there is, however, a path from the start and goal states, the agent will still have to access a subset of the unblocked states. The worst case is when the last unblocked path the agent accesses is the goal state. Thus the time required for the agent to find an optimal path is at worst O(n^2) as well.

**Part 2) The Effects of the Ties**

The effect of having a minimum g-value tiebreaker vs a maximum g-value tiebreaker is akin to bfs vs dfs. In the case that a minimum g-value tiebreaker is used, the heuristic function weight h(s) is set to 0, therefore causing the tie-breaker to grab the closest node since it is the node with the minimal g-value, which makes the search to behave like bfs. In the case that a maximum g-value tiebreaker, h(s) is still 0, but due to the maximum g-value being selected, this time the tiebreaker grabs the farthest node, which makes the search behave like dfs.

**Part 3) Forward vs. Backward**

The traversal of the frontier tends to behave like dfs in the forward case, however after repeated cases, the forward A\* algorithm tends to jump around quite a bit. In the backward case, this goal does not move, and therefore the path is more statically defined that in the forward case.

**Part 4) Heuristics in the Adaptive A\***

1. Given a map M with no collisions, subgoal states a and b and with motion vectors in the cardinal directions, the exact path cost f(s) for any node s on path P is the manhattan distance: f(s) = gd(s, [start]) + gd(s, [goal]). For any P that is on map M now with collisions, if there does exist a subpath from a to b such that f(s) < gd(s, a) + gd(s, b), then the cardinal direction motion constraint would be violated in that subgraph, and therefore the manhattan distance is the least possible distance between two nodes a and b through contradiction. Thus, the manhattan distance is consistent.
2. The heuristic function gd is already consistent; by subtracting something from gd, the cost proves to remain consistent, since (gd - j) < gd (where j is a positive function).
3. Suppose that n\* is a neighbor of the goal state. Then the goal state is a successor n’ or n\* generated by some action A. Thus, the cost of reaching the goal is equivalent to the cost of performing one action. Thus, the heuristic gives the exact cost of reaching the goal and does not overestimate and is admissible. Assume that a consistent heuristic is admissible up to k steps away from the goal state. Suppose now that we are at some state n, which is k+1 steps away from the goal state. Then the estimated cost of reaching the goal state is h(n). By consistency, we know that the estimated cost of reaching the goal from n is no greater than the step cost of getting to a successor n’ by action A plus the estimated cost of reaching the goal state from n’. Since we know that a successor n’ of n is one step closer to the goal state than n is to the goal state, n’ is k steps away from the goal state. Thus, the heuristic does not overestimate the cost of getting to the goal state from the successor state n’. Since the cost of reaching n’ from n by action A is known exactly, the heuristic cannot overestimate the cost of reaching n’ with action A. Thus the heuristic does not overestimate the cost of reaching the goal state from state n. So a consistent heuristic is also admissible.

**Part 5) Heuristics in the Adaptive A\***

The adaptive A\* algorithm tends to explore less nodes that deviate from the given “path”, since the cost is decreased amongst that path. Thus, the adaptive A\* algorithm actually equals or outperforms repeated A\* at the cost of storing extra information.

**Part 6) Memory Issues**

The optimizations made are as follows:

1. Tree Pointer is Child-to-Parent pointers, which are 2 bits to indicate cardinal direction. This is so that we can construct the path from the goal back to the start by tracing through the parents. This grows as O(n^2) because each node needs to store its pointer, and the agent may explore the entire maze
2. The Maze 1 bit per wall. This grows as O(n^2) since we can construct a map with where every bit but the starting state is a wall.
3. The Closed and Open indicators require 1 bit per node, and the number of nodes grow to O(n^2)
4. The Heap Position stored on the heap is 4 Bytes. This scales O(n^2) as we can add all the nodes onto the heap in an empty map.
5. The Heap Cost is 4 Bytes. Here, we are assuming that h(s) is calculated on the fly every time a comparison is required. We still need to store g(s), as the cost to get to the state s is 1 g(s\_parent).

In total, we require 69 bits per node

1. For a gridworld of size 1001 x 1001 there are 1,002,001 nodes. So, we need 69,138,069 bits, or about 8.642 MB.
2. Given a 4MB memory, or 33,554,432 bits (4MB \* 2^20 B/MB \* 8 bits/B), we perform A\* over a gridworld of size 697 x 697.