CS440: Assignment 1 Write-Up

# Ramaseshan Parthasarathy, Vamshikrishnan Balakrishnan, Benjamin Yan

**Part 1:**

a) In figure 8, the agent would move east instead of north, because the cell east of the agent is closer to the goal than the cell north of the agent. If we are computing h(x) using Manhattan distances, the h(x) value for the cell east of the agent is 3. While the h(x) for the cell north of the agent is 4. If we are computing h(x) using straight-line distances the h(x) value for the cell east of the agent is 3. And the h(x) value of the cell north of the agent is sqrt(17), which is approximately 4.123. The g(x) values for the cell to the east of the agent and the cell to the north of the agent are 1 and 1 respectively. Therefore, if we are to compute h(x) using Manhattan distances the f(x) value of the cell to the east of the agent is 4 and the f(x) value of the cell to the north of the agent is 5. And if we are to compute h(x) using straight-line distances, the f(x) value of the cell to the east of the agent is 5, and the f(x) value of the cell to the north of the agent is 5.123. So no matter what you are using to compute h(x), the cell to the east of the agent is always preferable to the cell north of the agent, since the f(x) value of the cell east of the agent is always less than the f(x) value to the north of the agent.

b) If we visualize A\* search as a tree, and let all of the nodes of the tree be the cells of the environment, we would have a tree with a finite number of nodes. Cells are represented as nodes and their connections to their neighbors are represented by edges. The algorithm will be traversing the tree based on the smallest f(x) value among the nodes in the open list. In the worst case, we would have to traverse the whole tree before reaching the goal node. Since there are a finite number of nodes, that means that there are a finite number of blocked nodes. Therefore, in a finite amount of time, the algorithm should be able to determine if the search is possible or not, given that it has exhausted all possible paths. Given an n x n gridworld, a move consists of moving from one unblocked cell to another unblocked cell. Let m be defined as the number of unblocked cells in the gridworld, where 2 <= m <= n. The maximum number of times the agent can visit a particular unblocked cell is 4. The agent must initially reach an unblocked cell from either the north, south, east, or west, given that the cell does not lie on an edge of the grid. Regardless of what direction the agent reaches the unblocked cell, there are now 3 possible directions for the agent to travel. Assume the worst case in which the immediate neighbors of the agent are unblocked but each path results in a dead end, which causes the agent to backtrack. There will be a total of 3 backtracks plus 1 initial visit, which results in 4 total visits to an unblocked cell. If we upperbound the number of maximum visits to an unblocked cell to 4, then we have a maximum number of 4\*m possible moves. Even if the agent were to make the maximum possible number of moves by backtracking to every cell it has visited 3 times, m2 will be an upperbound to the number of moves an agent can make for m >= 6. m must be >= 6 for backtracking in three directions to take place, 2 cells for agent and target, and 4 for the neighbors of the agent. For 2 <= m <= 5, we must consider circumstances where the agent has less than 4 neighbors. For the case where m = 2, if the target is a neighbor of the agent, then it will take one move to reach the target. Otherwise, it will take 0 moves to discover that the task is impossible. Therefore, 4 serves as a viable upperbound in this situation. For the case where m = 3, the agent can reach the target in one move if the target is a neighbor of the agent or in two moves if the agent moves to an unblocked cell and then to the target. In the case where the task is impossible, the agent will have moved a maximum of once. Therefore, 9 serves as a viable upperbound in this situation. For the case where m = 4, the agent can reach the target in either 1, 2 or 3 moves. If the target is a neighbor of the agent, then it will take the agent 1 move to reach the target. If the agent moves to an unblocked cell and then to the target, the agent moves twice. If the agent moves to an unblocked cell and then to another unblocked cell and then to the target, the agent will have moved thrice. It takes the agent a maximum of two moves to discover that the task is impossible. Therefore, 16 serves as a viable upperbound in this situation. For the case where m = 5, the agent can reach the target in either 1,2,3, or 4 moves. If the target is a neighbor of the agent, then it will take the agent 1 move to reach the target. If the agent moves to an unblocked cell and then to the target, the agent moves twice. If the agent moves to an unblocked cell and then to another unblocked cell and then to the target, the agent will have moved thrice. If the agent moves to an unblocked cell then to another unblocked cell then to another unblocked cell and then to the target, the agent will have moved four times. It takes the agent a maximum of three moves to discover that the task is impossible. Therefore, 25 serves as a viable upperbound in this situation. In all cases 2 <= m <= n, m2 serves as a viable upperbound.

**Part 2:**

Breaking ties in favor of smaller g values:

Trial 1: 1007907728 nanoseconds

Trial 2: 1455872540 nanoseconds

Trial 3: 1439560585 nanoseconds

Trial 4: 956870541 nanoseconds

Trial 5: 1064401799 nanoseconds

Trial 6: 8854319000 nanoseconds

Trial 7: 1233440049 nanoseconds

Trial 8: 835073757 nanoseconds

Trial 9: 1432089754 nanoseconds

Trial 10: 1213176520 nanoseconds

On Average: 1949271227 nanoseconds

Breaking ties in favor of larger g values:

Trial 1: 775270947 nanoseconds

Trial 2: 1287598377 nanoseconds

Trial 3: 1058307391 nanoseconds

Trial 4: 1652876540 nanoseconds

Trial 5: 847576372 nanoseconds

Trial 6: 1004614719 nanoseconds

Trial 7: 1068484403 nanoseconds

Trial 8: 870354068 nanoseconds

Trial 9: 906583240 nanoseconds

Trial 10: 1219061826 nanoseconds

On Average: 1069072788 nanoseconds

Breaking ties in favor of larger g values is on average roughly two times faster than breaking ties in favor of smaller g values.

**Part 3:**

Repeated A\* Forward:

Trial 1: 873079900 nanoseconds

Trial 2: 1034859163 nanoseconds

Trial 3: 1122366675 nanoseconds

Trial 4: 781828665 nanoseconds

Trial 5: 978880223 nanoseconds

Trial 6: 975576218 nanoseconds

Trial 7: 977225415 nanoseconds

Trial 8: 2466504808 nanoseconds

Trial 9: 750320731 nanoseconds

Trial 10: 1207171370 nanoseconds

On Average: 1028680987 nanoseconds

Repeated A\* Backward:

Trial 1: 1331522895 nanoseconds

Trial 2: 1160061981 nanoseconds

Trial 3: 1053726236 nanoseconds

Trial 4: 922918181 nanoseconds

Trial 5: 1042151044 nanoseconds

Trial 6: 925549205 nanoseconds

Trial 7: 1108425774 nanoseconds

Trial 8: 1455896576 nanoseconds

Trial 9: 1184367723 nanoseconds

Trial 10: 1190291375 nanoseconds

On Average: 1137491099 nanoseconds

Repeated A\* Forward is on average a little faster than Repeated A\* Forward. The rates are pretty much the same because the two algorithms are identical except for the fact that the starting position of the agent is swapped with the starting position of the target. Given any grid, going from agent to target is going to take the same time as going from target to agent. Therefore the running times of Repeated A\* Forward are going to be the same as Repeated A\* Backward if they are working on the same grid. If they are working on different grids, then their running times will be close to one another.