CS440: Assignment 1 Write-Up

Ramaseshan Parthasarathy, Benjamin Yan, Vamshikrishnan Balakrishnan

Part 1

1. In figure 8, the agent would move east istead 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.

2. 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 connectios 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 meas 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 cosists of moving from one unblocked cell to another unblocked cell. Let m be defined as the number of unblocked cells in the gridworld, where $2 \leq m \leq n$. 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 directios 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 $m \geq 6$. $m \geq 6$ for backtracking in three directios to take place, 2 cells for agent and target, and 4 for the neighbors of the agent. For $2 \leq m \leq 5$, we must cosider 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 $2 \leq m \leq n$, m2 serves as a viable upperbound.

Part 2

Breaking Ties (smaller g-values)

Trial 1: 0.889908735 s
Trial 2: 1.626920612 s
Trial 3: 1.190200718 s
Trial 4: 0.662945985 s
Trial 5: 0.725735046 s
Trial 6: 0.858039396 s

Trial 7: 0.992606588 s
Trial 8: 0.724939105 s
Trial 9: 0.618976310 s
Trial 10: 0.965936916 s
Trial 11: 0.889908735 s
Trial 12: 0.673865247 s
Trial 13: 0.696835759 s
Trial 14: 0.775753944 s
Trial 15: 0.769164025 s
Trial 16: 0.677098849 s
Trial 17: 0.847573608 s
Trial 18: 0.710689580 s
Trial 19: 0.723463897 s
Trial 20: 0.628213449 s
Trial 21: 0.702496829 s
Trial 22: 0.759636526 s
Trial 23: 0.737197647 s
Trial 24: 0.616687792 s
Trial 25: 0.812586206 s
Trial 26: 0.688629415 s
Trial 27: 0.725737689 s
Trial 28: 1.419379492 s
Trial 29: 0.729100802 s
Trial 30: 0.638289951 s
Trial 31: 1.344752891 s
Trial 32: 0.672838227 s
Trial 33: 0.854792955 s
Trial 34: 0.663107589 s
Trial 35: 0.702843826 s
Trial 36: 0.686972213 s
Trial 37: 0.716895499 s
Trial 38: 0.621801371 s
Trial 39: 0.629531207 s
Trial 40: 1.921337437 s
Trial 41: 0.889908735 s
Trial 42: 1.626920612 s
Trial 43: 0.569676704 s
Trial 44: 0.720555769 s
Trial 45: 0.703591814 s
Trial 46: 0.693649731 s
Trial 47: 0.706856378 s
Trial 48: 0.689164070 s
Trial 49: 0.688656224 s
Trial 50: 1.182961359 s
On Average: 0.835306669 s

Breaking Ties (larger g-values)

Trial 1: 1.007051701 s
Trial 2: 1.399900848 s
Trial 3: 0.858473991 s
Trial 4: 0.664442715 s
Trial 5: 0.732672718 s
Trial 6: 0.844294318 s
Trial 7: 0.930471119 s
Trial 8: 0.605686972 s
Trial 9: 0.637752654 s
Trial 10: 1.043698194 s
Trial 11: 0.642095589 s
Trial 12: 0.688105710 s
Trial 13: 0.651825472 s
Trial 14: 0.729097027 s
Trial 15: 0.693863065 s
Trial 16: 0.657707049 s
Trial 17: 0.797715560 s
Trial 18: 0.617799390 s
Trial 19: 0.730049664 s
Trial 20: 0.658330813 s
Trial 21: 0.661412251 s
Trial 22: 0.697436490 s
Trial 23: 0.746523895 s
Trial 24: 0.620743389 s
Trial 25: 0.731459173 s
Trial 26: 0.783385609 s
Trial 27: 0.782199628 s
Trial 28: 0.781373858 s
Trial 29: 0.710431692 s
Trial 30: 0.683001194 s
Trial 31: 1.314089923 s
Trial 32: 0.759621045 s
Trial 33: 0.968766885 s
Trial 34: 0.960389875 s
Trial 35: 0.890202492 s
Trial 36: 0.881763938 s
Trial 37: 0.792143599 s
Trial 38: 0.681745360 s
Trial 39: 0.614301103 s
Trial 40: 1.440904624 s
Trial 41: 0.629131349 s
Trial 42: 0.841453400 s
Trial 43: 0.609119939 s

Trial 44: 0.729143469 s
Trial 45: 0.761497244 s
Trial 46: 0.684847565 s
Trial 47: 0.775661059 s
Trial 48: 0.697727228 s
Trial 49: 0.750453003 s
Trial 50: 0.630426451 s
On Average: 0.784047826 s

Breaking ties in favor of larger g values is on average roughly 1.05 times faster than breaking ties in favor of smaller g values. The explanation of why can be seen by running the two versios of Repeated A* on the grid in Figure 9. For the algorithm that breaks ties with smaller g values, the expanded cells would be: (A,1),(B,1),(C,1),(D,1),(E,1),(E,2),(D,2),(D,3),(D,4),(E,4),(E,5). Whereas with the algorithm that breaks ties with larger g values, the expanded cells would be: (A,1),(B,1),(C,1),(D,1),(E,1),(E,2),(E,3),(E,4),(E,5). The algorithm that breaks ties with smaller g values has eleven expasios; whereas, with the algorithm that breaks ties with larger g values has only nine expasios. Therefore, even in the case where all cells are unblocked, the algorithm that breaks ties with larger g values has to do fewer expasios. Therefore, the algorithm that breaks ties with larger g values is more efficient than that which breaks ties with smaller g values.

Part 3

Repeated Forward A*

Trial 1: 1.007051701 s
Trial 2: 1.399900848 s
Trial 3: 0.858473991 s
Trial 4: 0.664442715 s
Trial 5: 0.732672718 s
Trial 6: 0.844294318 s
Trial 7: 0.930471119 s
Trial 8: 0.605686972 s
Trial 9: 0.637752654 s
Trial 10: 1.043698194 s
Trial 11: 0.642095589 s
Trial 12: 0.688105710 s
Trial 13: 0.651825472 s
Trial 14: 0.729097027 s
Trial 15: 0.693863065 s
Trial 16: 0.657707049 s
Trial 17: 0.797715560 s
Trial 18: 0.617799390 s
Trial 19: 0.730049664 s

Trial 20: 0.658330813 s
Trial 21: 0.661412251 s
Trial 22: 0.697436490 s
Trial 23: 0.746523895 s
Trial 24: 0.620743389 s
Trial 25: 0.731459173 s
Trial 26: 0.783385609 s
Trial 27: 0.782199628 s
Trial 28: 0.781373858 s
Trial 29: 0.710431692 s
Trial 30: 0.683001194 s
Trial 31: 1.314089923 s
Trial 32: 0.759621045 s
Trial 33: 0.968766885 s
Trial 34: 0.960389875 s
Trial 35: 0.890202492 s
Trial 36: 0.881763938 s
Trial 37: 0.792143599 s
Trial 38: 0.681745360 s
Trial 39: 0.614301103 s
Trial 40: 1.440904624 s
Trial 41: 0.629131349 s
Trial 42: 0.841453400 s
Trial 43: 0.609119939 s
Trial 44: 0.729143469 s
Trial 45: 0.761497244 s
Trial 46: 0.684847565 s
Trial 47: 0.775661059 s
Trial 48: 0.697727228 s
Trial 49: 0.750453003 s
Trial 50: 0.630426451 s
On Average: 0.784047826 s

Repeated Backward A*

Trial 1: 0.643475648 s
Trial 2: 0.689753851 s
Trial 3: 0.867946364 s
Trial 4: 0.656414968 s
Trial 5: 0.636347297 s
Trial 6: 2.898249120 s
Trial 7: 0.715138993 s
Trial 8: 0.720332241 s
Trial 9: 0.669246299 s
Trial 10: 0.692143561 s

Trial 11: 0.650467313 s
Trial 12: 0.688105710 s
Trial 13: 0.692437697 s
Trial 14: 0.669929343 s
Trial 15: 12.310755128 s
Trial 16: 0.762648489 s
Trial 17: 0.710814181 s
Trial 18: 0.629986192 s
Trial 19: 0.671771561 s
Trial 20: 0.773261155 s
Trial 21: 0.822598519 s
Trial 22: 31.118595520 s
Trial 23: 0.676540785 s
Trial 24: 0.645826843 s
Trial 25: 0.677060714 s
Trial 26: 0.975667931 s
Trial 27: 46.137521385 s
Trial 28: 0.720572760 s
Trial 29: 0.775922345 s
Trial 30: 0.724410493 s
Trial 31: 0.760344868 s
Trial 32: 0.792768495 s
Trial 33: 1.064408129 s
Trial 34: 0.921223785 s
Trial 35: 0.834791362 s
Trial 36: 0.877594689 s
Trial 37: 0.762949042 s
Trial 38: 0.868065680 s
Trial 39: 2.212197827 s
Trial 40: 1.820954532 s
Trial 41: 0.657396678 s
Trial 42: 1.455191530 s
Trial 43: 0.768458325 s
Trial 44: 0.723300405 s
Trial 45: 0.648847869 s
Trial 46: 0.684847565 s
Trial 47: 0.737289777 s
Trial 48: 0.731090654 s
Trial 49: 0.794012624 s
Trial 50: 0.725235507 s
On Average: 2.597298235 s

Repeated Forward A* is on average 3.31x faster than Repeated Backward A*. Repeated Backward A* expands more nodes than the other types of A* algorithms since there exists many onbstructions surrounding the agent but not so much near the goal. Further more, heuristics are not reused between searches

and hence creates additional overhead. This can be seen in the results of the maze runs shown above.

Part 4

A heuristic function is said to be cosistent if for any states n and n' and any action a: $h(n) \leq c(n, a, n') + h(n')$, where c(n, a, n') is the step cost for going from n to n' using action a.

Given any two states on the grid, one can compute the Manhattan distances between the two states. Suppose that moving from n to n' results in the agent moving farther away from the target. Then c(n, a, n') will be the Manhattan distance between n and n' and h(n') will be the Manhattan distance from n' to the goal. Notice that $h(n) \leq h(n')$. Also c(n, a, n') is always greater than 0. Therefore, that implies that $h(n) \leq c(n, a, n') + h(n')$. Now suppose that moving the agent from n to n' results in bringing the agent closer to the target. c(n, a, n') will once again denote the Manhattan distance between n and n'. In this case $h(n') \leq h(n)$. However, the amount by which h(n') is less than h(n) is exactly c(n, a, n'). So then $h(n) = c(n, a, n') + h(n')$, which meas that $h(n) \leq c(n, a, n') + h(n')$ holds. This can only be guaranteed if the agent can only move in the four main compass directios: north, south, east, west. If the agent were able to move diagonally then in some cases $h(n) \neq c(n, a, n') + h(n')$, since h(n) and h(n') will differ by more than c(n, a, n'), in which case $h(n) > c(n, a, n') + h(n')$.

After replacing h(s) with hnew(s), the h-value is still cosistent because h(s') will also be updated to $h < sub > new < /sub > (s')$ and two values will differ by a value that is exactly equal to the difference between h(s) and h(s'). And so therefore when checking to see if $h < sub > new < /sub > (n) \leq c(n, a, n') + h < sub > new < /sub > (n')$, both sides of the inequality have increased by a costant amount and so the inequality should still hold.

Part 5

Repeated Forward A*

Trial 1: 1.007051701 s
Trial 2: 1.399900848 s
Trial 3: 0.858473991 s
Trial 4: 0.664442715 s
Trial 5: 0.732672718 s
Trial 6: 0.844294318 s
Trial 7: 0.930471119 s
Trial 8: 0.605686972 s
Trial 9: 0.637752654 s
Trial 10: 1.043698194 s

Trial 11: 0.642095589 s
Trial 12: 0.688105710 s
Trial 13: 0.651825472 s
Trial 14: 0.729097027 s
Trial 15: 0.693863065 s
Trial 16: 0.657707049 s
Trial 17: 0.797715560 s
Trial 18: 0.617799390 s
Trial 19: 0.730049664 s
Trial 20: 0.658330813 s
Trial 21: 0.661412251 s
Trial 22: 0.697436490 s
Trial 23: 0.746523895 s
Trial 24: 0.620743389 s
Trial 25: 0.731459173 s
Trial 26: 0.783385609 s
Trial 27: 0.782199628 s
Trial 28: 0.781373858 s
Trial 29: 0.710431692 s
Trial 30: 0.683001194 s
Trial 31: 1.314089923 s
Trial 32: 0.759621045 s
Trial 33: 0.968766885 s
Trial 34: 0.960389875 s
Trial 35: 0.890202492 s
Trial 36: 0.881763938 s
Trial 37: 0.792143599 s
Trial 38: 0.681745360 s
Trial 39: 0.614301103 s
Trial 40: 1.440904624 s
Trial 41: 0.629131349 s
Trial 42: 0.841453400 s
Trial 43: 0.609119939 s
Trial 44: 0.729143469 s
Trial 45: 0.761497244 s
Trial 46: 0.684847565 s
Trial 47: 0.775661059 s
Trial 48: 0.697727228 s
Trial 49: 0.750453003 s
Trial 50: 0.630426451 s
On Average: 0.784047826 s

Adaptive A*

Trial 1: 0.934853323 s
Trial 2: 1.451329632 s
Trial 3: 0.820583748 s
Trial 4: 0.702931803 s
Trial 5: 0.731953049 s
Trial 6: 0.825563662 s
Trial 7: 1.022041286 s
Trial 8: 0.609560575 s
Trial 9: 0.662398492 s
Trial 10: 0.957801180 s
Trial 11: 0.723278883 s
Trial 12: 0.666526584 s
Trial 13: 0.629883867 s
Trial 14: 0.683563035 s
Trial 15: 0.801491369 s
Trial 16: 1.177493988 s
Trial 17: 0.627006701 s
Trial 18: 0.623328308 s
Trial 19: 0.760108879 s
Trial 20: 0.652471890 s
Trial 21: 0.679613160 s
Trial 22: 0.828151979 s
Trial 23: 0.681731768 s
Trial 24: 0.594161315 s
Trial 25: 0.796142180 s
Trial 26: 0.717997280 s
Trial 27: 0.816296316 s
Trial 28: 0.781373858 s
Trial 29: 0.679681125 s
Trial 30: 0.622504426 s
Trial 31: 1.680471402 s
Trial 32: 0.661188345 s
Trial 33: 0.689955480 s
Trial 34: 0.710709969 s
Trial 35: 0.660953867 s
Trial 36: 0.687389062 s
Trial 37: 0.703769654 s
Trial 38: 0.615277528 s
Trial 39: 0.809494951 s
Trial 40: 1.416373570 s
Trial 41: 0.631129507 s
Trial 42: 0.742145468 s
Trial 43: 0.673017578 s

Trial 44: 0.747750656 s
Trial 45: 0.693783018 s
Trial 46: 0.653617848 s
Trial 47: 0.702565549 s
Trial 48: 0.706572437 s
Trial 49: 0.827789124 s
Trial 50: 0.887554141 s
On Average: 0.783266656 s

Due to adjustments made to $h(s)$, Adaptive A* should, on occasion, take different paths as opposed to Repeated Forward A*. The path generated by Adaptive A* is longer, but should, in theory, expand fewer nodes in total as opposed to Repeated Forward A*. This means that Adaptive A* should run faster overall. Results show that Adaptive A* runs about 1.001x faster (~0.0008 s) on average, which meets most of our expectations, though is not as significant of a speed up as we thought, perhaps due to internal book-keeping and update operations.

Part 6

Memory can be saved through the data type which are needed. For example, using byte instead of int will save a few bytes per cell. To conserve memory through the implementation, the use of another data structure, such as an array list, instead of a min heap to store the open and the closed list. This is because an array list will only allocate memory as needed.

Grid worlds with the size of 1001x1001 will take up the space of 250500.25 bytes (or 2004002 bits). This is calculated by allocating 2 bits cell. The largest grid world that can operate within a limit of 4 MB is a 4000x4000 grid.