Eda Gür

0050488

Q1.

With bigMaze, dfs expanded 390 nodes and had a cost of 210 and bfs expanded 620 nodes and had a cost of 210.

With openMaze, dfs expanded 576 nodes and had a cost of 298 and bfs expanded 682 nodes and had a cost of 54.

As we can see, dfs expands smaller number of nodes, but bfs has less cost. This is because bfs uses a queue(FIFO) and dfs uses a stack. So, I would use dfs when I care more about memory space (because it expands less nodes) and bfs when I care more about optimality. (because BFS is optimal, DFS is not.)

Q2.

For the command:

python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar (or ucs),heuristic=manhattanHeuristic

UCS has 620 expanded nodes and A* had 549 expanded nodes. They both have a cost of 210 as they are both optimal.

I would use UCS when I care about other things than just cos tor number of expanded nodes. For example, if one area is ghost-heavy, because I can adjust my cost function and avoid that area. I would use A* search when I care more about finding the solution quickly, because it expands less nodes since it has a lower bound supplied by a heuristic function and ucs expands in every direction and thus expands more nodes.

Q3.

I have a short answer: I only keep the current position and the corners explored in the state, because covering the corners is our main goal, not our food. So we get the *utility* from visiting the corners. I can determine a goal state by checking if the number of visited corners are 4 and if I am at a position and it is one of the corners, I can add it to the explored corners. So, current position and corners explored is enough information on the state.

Q4.

In a position, I find the lower-bound costs for visiting all the corners by Manhattan distance: We have 4 unvisited positions at first named A, B, C, D. I find the lower bounds on all of them. Then, I find the lower bound for A-B, A-C, A-D, B-A, B-C, … I find the lower bound costs for all the possible combinations and add them to a list. Then I find the minimum of the list. This heuristic is called for every visited node. It’s admissible because I find the lowest cost of the corner combinations assuming we

don’t face any walls. If we face walls, the actual cost will be greater than or equal to this lower bound. So lower-bound <= actual cost. It is consistent, because, in a possible combination A-B-C-D, going from corner A to D has cost (going from A to C) + (going from C to D). This is the definition of admissibility: The cost of going from current node to successor node plus the estimated cost of going from current node to goal should be less than or equal to the estimated cost of reaching the goal from current node.

Q5.

I got partial credit from this part, because I couldn’t implement my full idea. It is smaller to corners heuristic. I start with a node, find the cost of going to all the food, and draw all possible food combination routes. I find the costs for all of these and add them to a list and find the minimum cost one. This heuristic is called for every visited node. This is admissible, because it finds the lowest cost combination assuming Manhattan distance. Manhattan Distance <= actual distance in Pacman. So lower bound <= actual cost. This is consistent, because the estimated cost of going from current_food to goal is less than or equal to going to the food before the goal and estimated cost continuing to the goal. Est(current, goal)<= Est(next, goal) + Cost(current, next)

Q6.

As stated in the homework webpage, finding an optimal solution can be hard. For a solution cost as near to the real cost as possible, we should find our heuristic bounds as good as possible. Instead, we can go for a suboptimal greedy search. It can result in a suboptimal solution which expands less number of nodes. The cost is more, but expanded nodes are less. This is why we should go with inconsistency if we want less nodes and admissibility if we want optimality.