Doruk ÖZER

70192

Q1

Main difference is, DFS algorithm is expanding the deepest node in the frontier ,while BFS algorithm expands all the nodes in the same level before the nodes which are at the next level. The difference of this expansion strategy is caused by the data structures which are used in those algorithms: BFS uses Queue data structure while DFS uses Stack data structure. BFS can be preferred if the goal of the problem is closer to the starting node and if the goal of the problem stands away from the starting node, it is said that DFS can be preferred. In terms of Pacman, if the food is located way away from the starting position, then DFS works faster. Since DFS expands the deepest node in the frontier if the goal of the problem is in the shallow levels and if there is a lot of deep levels and branching factor in deep levels then BFS is much better in terms of performance. On the other hand if the goal of the problem is dependent on the decision makings then there will be a lot of overload if we expand the nodes in the same levels so in that case DFS works overall faster.

Q2

We should start the comparison from the cost function that both algorithms evaluate in order to decide which nodes to expand. The UCS expands the least cost node with the the cost function that doesn't have the information about how far away the node to be expanded from the goal state of the problem, so the cost function of UCS the cost of moving to another node. However, A uses the information about the node's estimated cost to the goal state of the problem in it's cost function. In other words the cost function of A* has the information about have far the expended node is from the goal state but UCS doesn't have that information. The UCS can be used over A* algorithm in order to solve the problems that the cost of the node to the goal is only cumulative cost of the path. A* algorithm can be preferred if there is a* ***good heuristic function*** *that estimates the cost to get from the node to the goal state (the cost function of A*). A* is much more efficient in terms of number of expended nodes than UCS if there is a efficient* ***heuristic function*** *which is admissible and consistent.*

Q3

At first I, couldn't think that efficiently but I optimized it in my trial and error phase. I needed a state definition that differentiates the same state, after the agent visits one corner and before visiting the corner. What I mean is, think two cases where agent has to pass the location X before visiting leftmost top corner, and the case where the agent has to pass the location X after visiting leftmost top corner in order to go to rightmost top corner. Then the states when the agent reaches to X in both cases should be different then each other. I had already defined the isGoalState method before defining the state so I decided to use that Boolean tuple -which I hold in order to track the corners if they are visited or not-, in my state definition. Then all of the states for locations after visiting the corners and before visiting corners became unique and the state definition is worked well.

Q4

At first I decide to use the minimum Manhattan distance from the current state to corner among all of the 4 corners. That was also admissible and consistent. However, in terms of efficiency it failed. The reason it was inefficient was that using minimum distance between the state and among all corners is misleading as the by definition itself the heuristics is the function that evaluates the cost to get from the current or next state to the goal state. So the cost to the goal from the state was not the minimum distance from current state to corner among all corners it was the maximum distance from the current state to the corner among all corners. By that it never overestimates the true cost to the solution which is the definition of being admissible heuristic. Admissible heuristics cares not overestimating true cost of solution along the current path. So it should be capturing the cumulative cost of step-costs of state transitions. In terms of corner problem, since the true cost of the state visiting the all of the corners from current state is defined as maximum Manhattan distance corner among all of the corners, and moving current state to it's successors states can only improve costs by 1 which is step cost and cost of moving. So the Manhattan distance always captures cumulative cost to solution since true cost to the solution bounded between Manhattan distance and the number of moves required to go around the walls. The maximum operator and Manhattan distance estimates the true cost to the solution without overestimating it.It is consistent because the difference between next states' estimated cost of reaching goal from the that state and reaching goal from the current state is bounded by 1 which is step cost of moving from one state to another. In addition, cost of moving one block is captured, since I am using the Manhattan distance and since Pacman cant move diagonal. So the differences of cost of the current state and cost of the successors' to the goal can't be higher than 1.

Q5

I didn't have the intuition of finding the best heuristics for this problem but I had an idea. I decided to write a admissible and consistent heuristic for this part. I came up with counting the number of dots which are uneaten. By definition number of uneaten dots is the function that evaluates the cost to get from the state to the goal state. The heuristics doesn't capture the fact that the number of uneaten dots may stay the same when Pacman moves to a space where there is no food but it could be increasing it's probability to eat all of the food with that strategical move to the state where there is no food. It is consistent because the difference between next states' estimated cost of reaching goal from the that state and reaching goal from the current state is bounded by 1 which is step cost. The main reason of consistency is Pacman only can eat 1 dot or no dot which creates the step cost. So the differences of cost of the current state and cost of the successors' to the goal cant be higher than 1. It is admissible, because it doesn't overestimates the true cost to the path. The number of uneaten dots in current state captures the step cost of changing one state to another state and all of the step costs to the goal state. What I

mean is the step cost is eating a dot or not eating a dot which is 1. If we count number of dots in a state the cost of reaching the goal state is actually that number.

Q6

A consistent heuristic is required to be admissible. Consistency is more stricter than the requirement of being admissible. So in terms of practical difference of consistent and inadmissible heuristic I am not sure I understanded the question exactly but I think the question is about admissible heuristics and consistent heuristics. Admissible heuristics only cares not overestimating true cost of solution along the current path. So it should be capturing the cumulative cost of step-costs of state transitions. In addition, if the heuristic needs to be consistent then the difference between next states' estimated cost of reaching goal from the that state and reaching goal from the current state must be bounded by is step cost of transiting current state to it's successor state. So the consistent heuristics are much better in terms of number of nodes expended and path cost than just admissible heuristic.