Assignment 1 – Part 2

Mertcan Aşgün 63948

1. DFS is not optimal, but BFS is optimal. Therefore, path cost of BFS is lesser or equal to path cost of DFS. In terms of number of expanded nodes, BFS works well when goal state is in shallower depths, whereas DFS works well when branching factor is minimum. In general, BFS expands more nodes than DFS before finding a solution. However, as an example, in “openMaze”, since branching factor is greater than normal pacman problems, DFS expands more nodes than BFS.

So, if we want an optimal path BFS is more appropriate. However, if the depth of the problem is too high, then BFS would take a lot of time. In such cases DFS might return a solution much faster, but it might not be optimal.

1. If we use nullHeuristic as our heuristic for A\*, then A\* is identical to UCS. So, when I mention A\*, I consider heuristic function as Manhattan Distance.

UCS is optimal, and if we pick a nice heuristic (Manhattan Distance is ok) then so is A\*. Therefore, their path costs are equal. For the number of expanded nodes, A\* performs better due to its heuristic function, which causes A\* to expand lower number of nodes.

Having a heuristic function, however, requires prior knowledge about the problem and which is not possible in every situation. Therefore, if we have any information about the problem, creating a heuristic function and using A\* is more useful. However, if we don’t have any information regarding the problem, then UCS is preferable.

1. I chose a state with 5 components. First one is the location of the pacman, whereas the remaining 4 components represent “flags”, indicates whether a corner is visited or not. Initially, all those 4 flags are initiated to 0. And the goal state checks whether all those flags became 1 or not. While getting the successor, required method checks that whether the corresponding successor is in a corner or not. If yes, it changes corresponding flag value to 1. Since, to be able to complete the problem we need to visit each corner, if each flag become 1, then we understand that each corner is visited.
2. Firstly, I calculated the Manhattan Distances between the pacman and each corner, and whenever a corner is visited, I set corresponding distance value to 0. My first idea was the sum them up. Despite, it found the optimal solution in quite few nodes, I suppose it was not admissible. Then I get the maximum of each distance. And it worked quite well. For its admissibility; for pacman to visit all corners, it must move at least by maximum of Manhattan Distances. So, it is admissible. For consistency; whenever pacman moves 1 unit, maximum of the Manhattan Distances could reduce by 1, at most. So, it is also consistent.
3. Realizing from previous question, that maximum of Manhattan Distances works quite well, I used it immediately. Even if it works nice, it was not able to get full points. Probably it was due to the location of the walls. Even one of the corners was close to pacman when calculated by Manhattan Distance, due to walls, it was very far in reality. So, instead of Manhattan Distance, I used mazeDistance method in searchAgents.py, which calculates real distance between objects. This time, I would be able to get full point. For admissibility and consistency, the argument is same as previous question.
4. Consistency enforces admissibility and consistent heuristics gives optimal paths to solutions. However, while doing that, they might expand many nodes, which slows them down. On the other hand, inadmissible heuristics overestimate the cost of reaching the goal. This might cause them to find a solution that is not optimal. But this process expands a smaller number of nodes and performs faster. Therefore, if we prefer optimality over time, consistent heuristics are more appropriate. On the other hand, if we need to find any solution fast, then inadmissible heuristics are preferable.