Assignment 1. Search in Pacman
Project Report

Mehmet Enes Erciyes, 68906
Computer Engineering

I. Question 1: DFS v. BFS

A. What are some differences between DFS and BFS in terms of path cost and number of expanded nodes?

Algorithm/ Maze	Medium Maze	Big Maze
BFS	Path Cost: 68 Exp. Nodes: 269	Path Cost: 210 Exp. Nodes: 620
DFS	Path Cost: 130 Exp. Nodes: 146	Path Cost: 210 Exp. Nodes: 390

Depth First Search usually is more efficient expanding fewer nodes. However, this comes at the price of non-optimal solutions. Breadth First Search, on the other hand, always finds the optimal solution. However, it is more expensive.

B. When and why would you prefer BFS over DFS?

BFS essentially holds an advantage in efficiency. For applications that do not require the optimal solution and requires more memory, BFS can be preferred.

C. When and why would you prefer DFS over BFS?

Opposite to previous question, if an application requires the exact optimal solution found, DFS must be preferred.

II. Question 2: UCS v. A*

A. What are some differences between DFS and BFS in terms of path cost and number of expanded nodes?

Algorithm/ Maze	Medium Maze	Big Maze
UCS	Path Cost: 68 Exp. Nodes: 269	Path Cost: 210 Exp. Nodes: 620
A* (Manhattan distance heuristic)	Path Cost: 68 Exp. Nodes: 221	Path Cost: 210 Exp. Nodes: 549

UCS is a form of graph search that can be used with applications that has different costs for different actions. A* is an extension of UCS to include heuristics to make a more efficient search by using information on how to better achieve the goal. They both are complete algorithms, however, as seen in the table above, in the same maze, A* is more efficient than UCS.

B. When and why would you prefer UCS over A?*

A* requires an admissible and consistent heuristic function to be meaningful. However, for some problems, it is impossible or insignificant to find such a heuristic. In these situations, UCS is a better alternative.

C. When and why would you prefer DFS over BFS?

Opposite to previous question, if an application allows an admissible and consistent heuristic function to be found, it is almost always better to use A* over UCS.

III. Question 3: State In Finding All Corners

My choice in representing the state in finding all corners problem is a tuple as the following:

(positionTuple, ListOfUnvisitedNodes)

When getting the successors of a state, if the child state is one of the unvisited nodes, I removed it from the list of unvisited nodes. This approach worked because generic graph search algorithms tried to find the state with no nodes in unvisited list. While doing that, they expanded the nodes until they found all the corners.

At first, I tried to use visited corner count instead of a list. It was not effective, because at some point, all successors of a state become in the visited nodes list since there is no way to differentiate which corner was expanded by them. So, Pacman gets stuck at a state. The list approach eliminated this problem and allowed Pacman to find all corners.

IV. Question 4: Heuristic In Finding All Corners

The heuristic I last settled on is the maximum Manhattan distance of a state to an unvisited corner disregarding the walls.

```
position = state[0]
unvisitedCorners = state[1]
if len(unvisitedCorners):
    return max(map(lambda corner: util.manhattanDistance(corner,position), unvisitedCorners))
else:
    return 0
```

At first, I tried a heuristic that computes the Manhattan distance of a state to its closest corner using its distance to the center of the maze. This heuristic worked to a degree expanding 1682 nodes instead of about 2000.

However, this heuristic did not change after a corner is visited and still would give 0 when Pacman visited its first corner. Therefore, I thought of setting the heuristic as the Manhattan distance to the closest corner by getting the minimum of the distances to the unvisited states. This also worked to a degree, expanding about 1400 nodes.

Then, I realized that instead of getting the distance to closest corner, I should set the heuristic to be the distance to the farthest corner that is not yet visited. This way, A* star selected the node that is closest to the corner Pacman is closest to.

The reason this heuristic is admissible is because the maximum distance to farthest unvisited node is less than the cost of visiting all the nodes at every state. The reason that it is consistent is that for each adjacent node it is impossible for the difference between their Manhattan distance to farthest corner is more than 1. Therefore, for any A, B:

$$H(A) - H(B) <= \text{cost (A to B)}$$

V. Question 5: Heuristic For Eating All The Dots Problem

The heuristic I settled on was the maximum distance to a food. First, I tried to use the same logic in corners, using Manhattan distance. However, since there were many foods this time, it was not an effective solution expanding about 14000 nodes. Later, I said “Why not really compute the distance to the food using the BFS I already implemented?”. Then I wrote a custom search problem and solved it with BFS. Then at each step, I calculated the length of the path for Pacman to each food and got the maximum.

Getting the maximum of the length of paths to foods is obviously always lower or equal than the cost of collecting all the food. Therefore, it is admissible.

It is also consistent because the actual cost of moving between each adjacent node is 1 and the difference between the heuristics of two adjacent node cannot be larger than one.

VI. Question 6: Consistent v. Inadmissible Heuristic

If the heuristic function is bounded, then both heuristics would make a complete algorithm. The difference is in efficiency and optimality. The efficiency of an A* algorithm is based on two features: the number of expanded nodes and the time it takes to expand each node. An inadmissible heuristic may be faster to compute, therefore lowering the time on each node and making A* faster. Also, it may be closer to the actual cost (albeit it may be over it sometimes), thus result in fewer nodes expanded. So, if a usable consistent heuristic cannot be found, it may be okay to sacrifice optimality for a good enough solution using inadmissible heuristics. However, if we must find the optimal path, we should prefer a consistent heuristic.