Artificial Intelligence: Assignment 1

Project 1: Search

Omar Al Asaad 0075155

Report

Q1:

Path cost:

- **DFS:** Does not gurrantee the shortest path. It explores the as deep as it can before it backtracks. Solution can possibly have more edges in it which means it would have a higher cost.
- **BFS:** It always provides the shortest path as the first solution. It explores the nodes level by level.

Number of Expanded Nodes:

- **DFS:** If the solution is deep, it may explore fewer nodes. It might explore a lot of nodes before coming up with a solution.
- **BFS:** Expands nodes level by level which means it can possibly explore more nodes than DFS, especially if the solution is deep.

Preferences:

- **DFS Over BFS:**
  - When you have limited memory.
  - When any solution works and not searching for the shortest one
- **BFS Over DFS:**
  - When the goal is to find the shortest path.
  - When memory is not an issue

Q2:

Path cost:

- **UCS:** Focuses only on the costs spent up to this point. It does not use heuristic information but guarantees the lowest-cost path.
- ***A:** Takes into account both the actual cost and a heuristic estimation of the remaining cost. It uses additional heuristic guidance to guarantee the lowest-cost path.

Number of Expanded Nodes:

- **UCS:** Since it does not use heuristic trimming and only considers cumulative cost, it might expand to more nodes.
- ***A:** Able to expand fewer nodes by effectively weighing nodes according to cost and heuristic.

Preferences:

- **UCS Over A***
  - In situations where heuristic data is either unavailable or prohibited.
  - Appropriate for scenarios where path cost is the primary concern.
- **A * Over UCS**
  - When heuristic is available.
  - Good for balancing the path cost and heuristic.
  - Good in reducing the number of expanded nodes.

Q3:

I chose a tuple with Pacman's current position and a list showing whether or not each corner has been visited has been used as the state representation. Each entry in the list indicates whether the relevant corner has been touched (1 for visited, 0 for not visited)

The code uses the position of the current state and the action's associated direction vector to calculate the next position for each potential action (north, south, east, and west). Next, it sees if the place after that hits a wall.

The algorithm checks if the next position belongs to one of the four corners if it is not blocked by a wall. If true the state is updated to indicate the visit to that specific corner. In order to accomplish this, the food list is modified, with each entry identifying the status of a corner (0 for not visited, 1 for visited). The corner's status is set to 1 at its current location.

Q4:

The Manhattan distance between the current state's position and its nearest unexplored corner is the heuristic. It evaluates every unexplored corner, determines how far it is Manhattan from the present point and chooses the greatest distance between them. Because it never overestimates the actual cost to achieve the goal, this heuristic is acceptable. In particular, it ensures that Pacman must go at least that distance to attain a goal by returning the maximum distance to an unexplored region. As a result, the heuristic offers a lower bound for the cost of the shortest path.

Q5:

It is based on the maximum Manhattan distance between the current state's Pacman position and any remaining food pellets. The heuristic calculates the distance from Pacman to each remaining food and selects the maximum value among them. This heuristic is admissible because it never overestimates

the true cost to reach the goal. Each food pellet requires at least its Manhattan distance to be covered, and the heuristic considers the maximum distance, ensuring it provides a lower bound on the true cost.

Q6:

Path Cost:

- **Consistent Heuristic:** It is guaranteed that the solution will have the lowest path cost.
- **Inadmissible Heuristic:** May overestimate the actual cost, which could result in poor solutions.

Number of Expanded Nodes:

- **Consistent Heuristic:** Results in fewer expanded nodes. Elimanates paths that lead to suboptimal solutions which would reduce the exploration needed.
- **Inadmissible Heuristic:** May lead to more expanded nodes as suboptimal paths are explored.

Preferences:

- **Inadmissible Heuristic:** Preferred when computational resources are limited, and a reasonably good solution is acceptable.
- **Consistent Heuristic:** Preferred when the goal is to ensure optimality in the solution. Preferred when we need to find the most optimal solution.