1 Introduction

In our previous lab, we implemented uninformed search algorithms. Uninformed search algorithms such as DFS, BFS and UCS have their own disadvantages. They may not always find the optimal solution. As Pac-Man's maze-like environment consist of many obstacles and thus quite complex to navigate, informed search algorithms are better option to handle such complexity.
The objective of this lab was to implement Informed Searched algorithms that can be applied to Pacman problem to help him reach a particular location and to collect food efficiently as well.

2 Problem Analysis

In the lab, we were assigned to do five tasks.

2.1

Our first task required us to implement A* graph search . A* takes a heuristic function as an argument. A heuristic is an estimate of the cost from a particular state to the goal state. Using heuristic A* gives the optimal path with the least cost will for a problem.

2.2

This task required us to design a problem and a heuristic for itnas well.The problem named CornersProblem had four corners, each with a dot. Our task was to find the shortest path that touches all four corners.

2.3

Our third task was to implement a consistent heuristic for the CornersProblem.

2.4

Our fourth task was to be able to eat all the Pacman food in as few steps as possible .

2.5

Our fifth task was to implement a greeedy algorithm that always eats the closest dot .

3 Solution

3.1

A* search takes a heuristic function as a parameter. Heuristic function is an important part of A* algorithm. It gives an estimation of the cost of reaching the goal state from current state.

In A* algorithm we first initialize a fringe with the start state along with its heuristic cost and actual cost to reach the state. Here the fringe is an priority queue whose priority is the total cost of a given node or state. As a priority queue maintains its elements in a way that allows for quick retrieval of the element with the lowest cost, it allows A* to expand nodes in

the correct order from lowest to highest cost.

While fringe is not empty we pop the node with the lowest total cost from the fringe. If the node is the goal state, we return the path to it. Other than these two cases, we expand the node and add its successors to the fringe with updated costs. Lastly, we return failure if no goal state is found. This is how a general A* algorithm works.

In the Pac-man scenario, this algorithm takes the configuration of pac-man's environment which includes Pac-Man, ghosts, food pellets and other relevant game dynamics. Here the goal is to eat all the food pellets while avoiding any interaction with the ghosts. The possible moves Pac-Man can make are moving up, down, left and right. The cost are estimated in terms of distance traveled or the time taken. Heuristic function provides an estimated cost. Using this estimated value and actual cost of the path, A* gives pac-man the optimal path to accomplish its goal.

3.2

This task was implemented by our instructor. To formulate this new problem that finds the shortest path through the maze that touches all four corners, we needed to define the states- start state, goal states and the successor function as well.

To store the walls, pacman's starting position and corners necessary code were already provided. For this problem as the pac-man needs to touch all the corner, we need to keep track of which corner is already visited and which not. We had a tuple named corners. We initialized a dictionary named visitedCorner, where the keys were the corners of the maze. The values for all the corners were initially set to False to indicate that Pac-Man has not visited any corner yet.

To obtain the start state a method named getStartState needed to be implemented. This method constructs a tuple startNode consisting of Pac-Man's starting position and the visitedCorner dictionary, which tracks whether Pac-Man has visited each corner or not. Finally, it returns this tuple as the start state.

isGoalState method tell us if the pac-man has reached the goal state or not. It takes two parameters- the problem itself and a state. In this problem a state represents the position of the pac-man and the information of the corners whether they are visited or not. After retrieving the corners' dictionary we traverse through the dictionary and check whether the corners are visited or not. If there is a single corner which has not been visited yet, whose value is yet False, it means there are still corners for the pac-man to visit and the pacman has not reached the goal state. So, the method returns false, If all the values of all the corners in the dictionary is True, the method return true as the pac-man has visited all the corners and thus reached goal state.

To get the successor given a state we needed to implement getSuccessors method which takes same parameters as the isGoalState. The possible actions for pac-man in this problem is to move north, south, east and west. For each action, we calculate the next position for pac-man based on pac-man's current position which is retrieved from the given state and checks if it hits a wall. If the next position is not a wall, we update the visited corners information accordingly. Ater that we construct successor states by combining the next position and the updated visited corners dictionary, and appends them to the list of successors. The method increments the count of expanded nodes and returns the list of successor states.

3.3

To construct a non-trivial, consistent heuristic for the problem, I chose Manhattan Distance between the position and corners.

At first I identified the unvisited corners and then calculated the Manhattan distance from Pac-Man's position to each unvisited corner. The heuristic value is the maximum distance among all unvisited corners. If all corners are visited, the method returns 0.

3.4

We need to define a path that collects all of the food in the Pacman world. For this, lab solutions only depend on the placement of walls, regular food and Pacman's position.

The implemented method estimates the minimum number of moves Pac-Man needs to make to reach the nearest food pellet. At first, it Extracts Pac-Man's current position and the list of food positions.If there are no food pellets left, it means the Pac-Man has already consumed all food. So the function returns 0.If the list is not empty, the method the iterates through each food position and for each food position, it calculates the maze distance from Pac-Man's position to that food position using the mazeDistance function which is already defined. This function returns the length of the shortest path between the two points in the maze using breadth-first search. After calculating all distances, foodHeuristic method finds the maximum distance among them and returns that.

3.5

To find a reasonably good path quickly, I chose to perform BFS.

This method uses AnyFoodSearchProblem method to initialize a problem instance for finding paths to any food pellet. Then it utilizes Breadth-First Search to efficiently find the shortest path to the closest food pellet.

4 Findings and Insights

4.1

To complete this task, I used set to keep track of visited nodes and to ensure that nodes are not expanded more than once. This prevents revisiting already explored states and thus optimizing the search process.

Sets provide constant-time complexity O(1). In contrast, lists require linear time complexity O(n).So it's efficient to use set to check if a node has been visited.

4.2

The problem's representation of states, goal and successor generation, provides clarity and direction for Pac-Man's path finding task.

Using a dictionary to keep track of visited corners increases efficiency in managing the exploration process. This approach allows for quick checks to determine if a corner has already been visited, optimizing the search process and preventing redundant exploration.

4.3

The implemented heuristic is admissible because it always underestimates the true cost to reach the goal. This is ensured by returning the maximum distance to any unvisited corner, which guarantees that the Pac-Man will need to travel at least that maximum distance to reach a goal state.

The heuristic is consistent because the Manhattan distance ensures that the heuristic value increases as Pac-Man moves away from unvisited corners, maintaining consistency in the search process.

4.4

The implemented method iterated over the remaining food dots, ensuring that no information about potential paths is overlooked.

By computing the maximum distance to any remaining food dot, the heuristic provide consistent estimate as reaching the farthest food dot would require traversing the most costly path.

In this task, I used Maze Distance which takes BFS into account for implementation. Thus the method expands 4137 nodes. Earlier I used simple Manhattan distance. Using Manhattan distance, it would expand 9551 nodes.

As Maze distance computed with BFS takes into account obstacles, walls, or any other barriers present in the environment, it allows to accurately measure the actual distance along traversable paths.Thus it provides better performance and more optimal solutions.

4.5

Breadth-First Search (BFS) is considered suboptimal or greedy because it prioritizes exploring nodes based on their depth level rather than their actual cost.

BFS guarantees finding the shortest path from the start node to any reachable node, but it does not necessarily find the globally optimal path. It may return a solution that is longer than the shortest possible path, especially if there are different costs associated with different actions.

BFS explores all nodes at a given depth level before moving to the next depth level. This greedy strategy ensures that the shortest path is found within a given depth level, but it may overlook shorter paths that exist at deeper levels of the search tree.

So the pac-man may pursue a path that seems locally closest or optimal but actually there are better paths to follow.

5 Challenges

Although Task 2 was already implemented, it took me some time to grasp the whole content. After the lab hours, I revised it a few times on my own for better understanding.

For Task3, I could not complete it in lab time. I was making a mistake in constructing the corners list. However, I solved it later by the help of internet and other resources and completed it.

In Task4 also, I could not get full marks during lab hours as I was using Manhattan distance. I searched about this problem over the internet and realized that Maze distance was the optimal way to calculate heuristic, So I mend it later also.