Pacman Search: Report

Sujit Mandava- 112001043

The project was aimed at designing an intelligent agent to traverse and find efficient solutions to the given goal in the Pacman environment. The project can be divided into two parts: implementing **generic search algorithms (DFS, BFS, UCS, A*)** and formulating **admissible and consistent heuristics** for different search problems.

Search Algorithms:

In each of the search algorithms, the search state to be explored was taken from a frontier list subjected to the goal test. It is then added to the list of explored nodes. The successor states of the explored state were then added to the frontier list. This process was repeated until a goal state is found or all states were explored.
All of the algorithms share the same basic structure. However, each of them differs in the selection of the search state and the addition of its successors to the frontier list.

1 Depth First Search

(DFS) explores the successors of the most recently explored search state. It follows the **LIFO (Last In First Out)** or **FILO (First In Last Out)** principle while selecting search states from the frontier list. Thus, a **stack** data structure is used to implement the frontier list.
DFS does not provide an optimal solution as it may fully explore multiple undesirable paths before it returns the actual solution. This increases the cost of the search multifold.

Maze	Nodes Explored	Cost
tinyMaze	15	10
mediumMaze	146	130
bigMaze	390	210

2 Breadth-First Search

(BFS) explores all the nodes at a particular depth before it proceeds to the next level. It follows the **FIFO (First In First Out)** principle while selecting search states from the frontier list. Thus, a **queue** data structure is used to implement the frontier list.
BFS provides the solution which takes the least number of actions to reach the goal state. However, this solution need not be optimal as different actions can have varying costs.

Maze	Nodes Explored	Cost
tinyMaze	15	8
mediumMaze	269	68
bigMaze	620	210

3 Uniform Cost Search

(UCS) explores the nodes with lesser path costs first. This is done by using a **priority queue** to store the frontier list. A priority queue sorts the incoming elements based on an additional parameter called the priority associated with it. Here, this parameter is given by the **path cost** associated with the node.

BFS and UCS behave similarly if the cost associated with each action is the same. However, UCS provides the solution with the least path cost when each action has a varying cost.

Maze	Nodes Explored	Cost
mediumMaze	269	68
mediumDottedMaze	186	1
mediumScaryMaze	108	68719479864

4 A* Search

A* search is considered one of the best best-first search algorithms. It works and is implemented identical to UCS, but with a substantial change to the priority parameter. A function that describes the degree of closeness of a particular search state to the goal state, called a **heuristic**, is introduced.

If ***g(n)*** is the path cost associated to reach the node and ***h(n)*** is the estimated cost the shortest path to reach the goal from the node, then the priority parameter for A* is given by ***f(n) = g(n) + h(n)***. UCS, however, uses ***f(n) = g(n)*** only.

Note that the heuristic function is considered admissible only if for all states, ***h(n) <= h*(n)***, where *h*(n)* is the actual cost of the shortest path to the goal from the node. Provided that the given heuristic is both admissible and consistent, A* search provides the most optimal solution.

Maze	Nodes Explored	Cost
bigMaze	549	210

Comparision of various searches on openMaze:

Search	Cost	Nodes Explored
DFS	298	576
BFS	54	682
UCS	54	682
A*	54	682

Heuristic Design:

5,6 Finding all corners/Corners Heuristic

The search state of the problem is defined as a tuple of the **current position of the agent** and a **list of corners visited** so far. A goal state is defined as any configuration of the above tuple where every corner has been visited at least once by the agent.

The heuristic for this problem is defined as the **minimum Manhattan distance** (or) the **shortest Manhattan path** to be covered by the agent to visit all the unvisited corners. This heuristic is admissible as the Manhattan distance between two points is the shortest distance between two coordinates in a grid, provided the agent can travel only in the directions up, down, left, and right or any movement analogous to so. The agent must travel a minimum of the computed distance for any configuration of walls and corners, and thus the heuristic is both admissible and consistent.

The strength of A* search compared to other search algorithms is evident when the number of expanded nodes and the cost of the search are compared with one another. The number of search states explored by A* search is significantly lower than that of BFS.

Maze	Search	Cost	Nodes Explored
mediumCorners	BFS	106	2448
mediumCorners	A*	106	**901**
bigCorners	BFS	162	9904
bigCorners	A*	162	**1705**

7 Eating all the dots

The heuristic of the problem is defined as the following:

The sum of the distances between the current position of the agent and the closest food dot and the largest distance between the former food dot and any other food dot. Here, distance is defined as the maze distance (or) actual path distance taken by the agent.

The chosen heuristic is both admissible and consistent. This is because the solution can be considered an extended version of the above-mentioned path. In the case there are no food particles, the state is a goal state. In case there is only one food particle, it returns the shortest distance to the food dot. In the case of two or more, every other food dot can be considered a detour from the mentioned path and thus the agent must travel the minimum distance mentioned above.

Search Problem	**Nodes Explored**	**Cost**
testSearch	7	7
trickySearch(Null Heuristic)	16688	60
trickySearch(Defined Heuristic)	**1818**	**60**

We notice that the number of nodes explored by the agent decreases significantly as the heuristic used is improved, making the search very efficient. We can thus conclude that A* Search is the most efficient search algorithm out of those implemented so far.