CSE 537 Artificial Intelligence

Project 1 — The Searching Pac-Man

Group Members:

Weituo Hao (ID: ), Xiufeng Yang (ID: ),

Hongjin Zhu (ID: 109748818), Ruonan Hao (ID: 108719989)

Date:

September 25, 2014

Problem Description

Our project is a simple player game Pac-Man. The goal of Pac-Man is to find path through its maze world, both to reach a particular location and and to collect food efficiently. We have built several search algorithms including depth-first search, breath-first search, uniform-cost search and A\* search, and applied them to different Pac-Man scenarios.

Finding a Fixed Food Dot Using Search Algorithms

Question 1 — Depth-first search

We implemented a graph search version of DFS, which searches the deepest node first in the tree and returns a list of actions to reach the goal. The stack data structure was used in the algorithm. Test results show that it is able to find solutions in various scenarios in a short time, but tends to explore more nodes than Pac-Man actually on its way to the goal.

Test Results

1. Tiny Maze

Path found with total cost of 10 in 0.0 seconds

Search nodes expanded: 16

1. Medium Maze

Path found with total cost of 130 in 0.0 seconds

Search nodes expanded: 147

1. Big Maze

Path found with total cost of 210 in 0.0 seconds

Search nodes expanded: 391

Question 2 — Breath-first search

We implemented a graph search version of BFS, which searches the shallowest nodes first in the tree. The queue data structure was used in the algorithm. Test results show that BFS is able to find an optimal solution to reach the goal and has relatively lower costs compared to DFS. But the drawback is that when the maze’s size is big, BFS tends to explore more nodes.

Test Results

1. Medium Maze

Path found with total cost of 68 in 0.0 seconds

Search nodes expanded: 269

1. Big Maze

Path found with total cost of 210 in 0.1 seconds

Search nodes expanded: 620

Varying the Cost Function

Question 3 — Uniform-cost search

We implemented a graph search version of UCS, which searches the node of the least cost first in the tree. The priority queue data structure was used in the algorithm. Test results show that UCS is also able to find optimal solutions and has similar cost and explored nodes with BFS. Further, when we use the StayEastSearchAgent and the StayWestSearchAgent, we get very low and very high path costs respectively, due to their exponential cost functions.

Test Results

1. Medium Maze

Path found with total cost of 68 in 0.0 seconds

Search nodes expanded: 269

1. Medium Dotted Maze

Path found with total cost of 1 in 0.0 seconds

Search nodes expanded: 187

1. Medium Scary Maze

Path found with total cost of 68719479864 in 0.0 seconds

Search nodes expanded: 109

1. Open Maze

Path found with total cost of 54 in 0.1 seconds

Search nodes expanded: 683

A\* search

Question 4 — A\* search

We implemented a graph search version of A\* search, which searches the node that has the lowest combined cost and heuristic first. The priority queue data structure was used in the algorithm. We tested the A\* implementation on the original problem of finding a path through a maze to a fixed position using the Manhattan distance heuristic. Test results show that A\* can also find an optimal way to the approach the goal but finds solutions slightly faster than UCS. In the open maze scenario, A\* is much more faster than UCS to find the optimal solution with 212 and 683 search nodes expanded respectively.

Test Results

1. Big Maze

Path found with total cost of 210 in 0.1 seconds

Search nodes expanded: 539

1. Open Maze

Path found with total cost of 54 in 0.0 seconds

Search nodes expanded: 212

Finding All the Corners

Question 5 — Corners Problem

We implemented the CornersProblem that can find the shortest path through the maze that touches all four corners. A tuple “corners” was used to detect whether all the corners have been reached, which is initialized to four ones. Whenever all the four corners are updated to zero and the current position is one of the corners, we know it is the goal state. Test results show that breath-first search expands 1989 nodes on medium corners, which is similar to Berkeley’s result.

Test Results

1. Tiny Corners

Path found with total cost of 28 in 0.0 seconds

Search nodes expanded: 270

1. Medium Corners

Path found with total cost of 106 in 0.3 seconds

Search nodes expanded: 1989

Question 6 — Heuristic for Corners Problem

We used the Manhattan distance as a heuristic for the CornersProblem in cornersHeuristic. This function always returns a number that is a lower bound on the shortest path from the state to the goal so that it is admissible (Noted in the textbook).

Test Results

1. Medium Corners

Path found with total cost of 106 in 0.1 seconds

Search nodes expanded: 1155

Eating All the Dots

Question 7 — Food Heuristic

We implemented the heuristic function using the Minimum Spanning Tree (MST) for the FoodSearchProblem. To generate the MST, we first store the Manhattan distance of any two nodes including all the food positions as well as the start position in a table. These values can be treated as the weight of edges afterwards. Then we used the Prim’s algorithm for the tree generation. The process is described as follows.

Step-1 Initialize the a tree with a single vertex, chosen arbitrarily from the graph. (In our case, we just need to initialize the tree with the start position.)

Step-2 Grow the tree by one edge: of the edges that connect the tree to vertices not yet in the tree, find the minimum-weight edge, and transfer it to the tree.

Step-3 Repeat Step-2 (until all food positions are covered in the tree).

When the above steps are finished, the total weight of the generated MST is the return value of heuristic. We tested the AStarFoodSearchAgent on the tricky search board. Results show that it can find the optimal solution in 7.7 seconds, exploring 7198 nodes.

Test Results

1. Test Search

Path found with total cost of 7 in 0.0 seconds

Search nodes expanded: 13

1. Tricky Search

Path found with total cost of 60 in 7.7 seconds

Search nodes expanded: 7198

Admissible vs. Consistency

MST is a spanning tree with weight less than or equal to the weight of every other spanning tree[1]. In our case, it is the lowest-weight subgraph connecting all food positions and the start position together such that can never overestimates the cost to the goal. Also, the Manhattan distance has been used as the weight of the edges and the MST property guarantees that we always add the edge with the lowest cost first. Therefore, the values of H(n) for each node along the path to the goal are non-decreasing[2]. In conclusion, our MST heuristic is both admissible and consistent. Evidence can also be found in the paper by Michael Held and Richard M. Karp[3] and the textbook.

References

[1] <http://en.wikipedia.org/wiki/Minimum_spanning_tree>

[2] http://en.wikipedia.org/wiki/Heuristic\_function

[3] M. Held and R. M. Karp, The traveling salesman problem and minimum spanning trees, Operations Research, 18:1138-1162, 1970.