Project 1

Student: Jose Matute Garcia

Partner: Anthony Dang

Answers

Section 3.2

1. Depth First Search

    1. (1 pt) Is the exploration order what you would have expected? Does Pacman actually go to all the explored squares on his way to the goal?

    The exploration order is something a person familiar with DFS should expect. Pacman seems to be always taking the leftmost path without caring about the depth or cost of the nodes. Pacman does not go to all explored squares.

    2. (1 pt) Is this a least-cost solution? If not, what do you think depth-first search is doing wrong?

    This is not the least cost solution because this algorithm does not care about cost when looking for a path.

2. Breadth-First Search

    1. (1 pt) Does BFS find a least-cost solution? If so, explain why

    BFS did find the least cost solution since all the costs were constant in the mazes we ran BFS with.

3. Varying the Cost Function (Uniform Cost Search)

    1. (1 pt) Specify the data structure used from util.py for uniform cost search

    Uniform cost search does what BFS does, which is calculating the shortest path, but by using the weights of the edges, or in this case, the cost of each node. Since the cost of a node is a priority, with the priority being the lowest cost per decision, the most useful data structure in this situation in util.py was the Priority Queue.

4. A* search

    1. (2 pts) What happens on openMaze for the various search strategies? Describe your answer in terms of the solution you get for A* and uniform cost search.

    They both find the best path with a cost of 54, but A* expanded fewer search nodes, a total amount of 535. This result is more efficient than uniform cost search, which expanded 682 nodes.

5. Finding All the Corners

1. (2 pts) Describe in few words/ lines the state representation you chose or how you solved the problem of finding all corners.

We chose to represent the state as a tuple, with its first item being the position of Pacman and the second item being a list of tuples, with each tuple being the position of an unvisited corner. This way, it is very easy to check if a state is a goal state by just checking if the list of unvisited corners is empty. In addition, whenever getSuccessors() gets called, we can simply check if a successor is a corner, which then I could simply eliminate it from the unvisited corners tuple list.

6. Corners Problem: Heuristic

1. (1 pt) Describe the heuristic you used for the implementation

We use the manhattan distance function provided to us to calculate the heuristic cost between Pacman and all the unvisited corners. We stored these values in a list and returned the biggest heuristic value in that list.

7. Eating All Dots

1. (2 pts) Describe the heuristic you used for the FoodSearchProblem.

We used the maze distance between the position of Pacman and every dot in the maze by taking advantage of the provided mazeDistance() function. We stored all these distances in a list and returned the biggest heuristic value in that list.

8. Suboptimal Search

1. (1 pt) Explain why the ClosestDotSearchAgent won't always find the shortest possible path through the maze.

For example, you have one dot to the right and the rest are to the left. And also, the closest dot is the dot to the left. If so, then Pacman will keep going to the left, and then when it finishes eating all the dots to the left, it will need to go all the way back to the only dot to the right of the map.

Let the distance from the beginning state to the last dot to the left is X, and the distance from the beginning state to the dot to the right is Y. Assume that X is much larger than Y, then the total distance Pacman traveled is X + X + Y.

If Pacman goes eat the dot to the left first and then eats all the dots to the right, the total distance would be Y + Y + X.

Since in this example, X is much larger than Y, therefore going to the closest dot first like what ClosestDotSearchAgent does not always find the shortest possible path through the maze.

Section 4

1. What was the hardest part of the assignment for you?

The hardest part of this assignment for me was determining the heuristics for both the corner and food heuristic problems.

2. What was the easiest part of the assignment for you?

Coding the depth-first search and breadth-first search algorithms, since as CS students, we have coded these algorithms many times in our careers.

3. What problem(s) helped further your understanding of the course material?

The A*start problem, combined with the heuristic problems, made me understand very clearly how heuristics, especially admissible and consistent ones, can help a search problem be more efficient.

4. Did you feel any problems were tedious and not helpful to your understanding of

the material?

Even though some problems were tedious, I believe all of the problems were helpful in understanding the usefulness of search algorithms and the help great heuristics provide.

5. What other feedback do you have about this homework?

No more feedback. I believe this homework was formulated in a very organized way and helped students understand graph search algorithms and heuristic concepts.