AI Lab 2

Karl-Johan Djervbrant

December 6, 2019

Task 1 Path Planning

1.1 a) Unified Search Algorithms

In this task we were supposed to implement three different types of path finding search algorithms.

1. Random Agent
2. Breadth First Agent
3. Depth First Agent

The random agent works by simply adding a random weight to the node between 1 and 10, then sort the list of all nodes by the weight and remove the first node in the list.
Breadth First and Depth first are very similar in how they are implemented, it's simply a difference in how one removes a node from the list. In Depth first, the last element in the list is removed first whereas in Breadth first you remove the first element.

1.1.1 Performance

On a set of 100 runs on a 100 by 100 map, this is the mean for each algorithm.

Mean of 100 runs	Breadth First Search	Depth First Search	Random Search
nodes	4725.61	5857.09	5445.28
pathLength	144.84	4413.16	292.60

Out of the three Unified Search Algorithms we can se that out of 100 runs, the Breadth First Search algorithm performs best, though none of the algorithms are particularly good when it come to the amount of expanded nodes.

1.2 a) Informed Search Algorithms

Greedy Search and A* were also implemented, with two different algorithm to calculate the weight of each node. Manhattan distance where each node weight W_n is calculated by subtracting current node position (X_n, Y_n) from the goal (X_g, Y_g).

$$D_n = |X_g - X_n| + |Y_g - Y_n| \tag{1}$$

Euclidean distance where the distance to the goal is a scalar.

$$D_n = \sqrt{(X_g - X_n)^2 + (Y_g - Y_n)^2} \tag{2}$$

In cases where you only can travel in a grid pattern (just as in this case), Manhattan distance should perform best. Since in this case you cant travel diagonal which is a pre-condition to get best performance out of the Euclidean distance formula. Both lists of nodes in Greedy and A* were sorted one the key `node.cost`. The difference between the Greedy and A* algorithm is that A* also increase it's weight in each node by it's search depth.

1.3 b) USA vs ISA on map with obstacle

Mean of 100 runs	BFS	A* Manhattan	A* Euclidean	Greedy Manh.	Greedy Eucl.
nodes	6864.73	3905.48	4693.62		
pathLength	318.80	322.76	318.80		

Here we can see som interesting statistics, the A* Euclidean does find the shortest path just as the BFS algorithm, though it expands more nodes. The A*

1.4 b) A* with heuristic

Mean of 100 runs	A* Custom Manhattan	A* Custom Euclidean
nodes	2591.90	3096.58
pathLength	326.72	302.78

This custom algorithm has a heuristic where it increases the weight of a node if it's close to the left side of the obstacle, this decreases the amount of expanded nodes, but it doesn't always find the most optimal path.

1.5 What I learned in Task 1

I learned how different search algorithms works and how much performance can be gained by choosing the correct algorithm.

Task 2 Poker Bidding

In this task we had to implement three different kinds of search algorithms to beat an opponent player in a game of poker. The implemented algorithms were a Random-, Breadth First- and a Greedy Search algorithm. The key used when sorting the list of states for the Greedy Search algorithm were the `nn_current_hand` which is a measure of how many hands the dealer have handed out in this state. I use this key since we sought to find the state were the agent win more than 100 coins from the opponent in the least amount of hands.

2.1 The implemented agents

2.1.1 Performance

Mean of 100 Games	Random Search	Breadth First Search	Greedy Search
stack	306.50	400.30	402.10
nNodes	9102.80	135871.28	4209.28
depth	13.24	10.36	10.64
nHands	7.22	2.30	2.10
opponentStack	293.50	199.70	197.90

After playing 100 games of poker we can see that the Greedy Search algorithm performs the best out of the three algorithms, with fewest number of expanded nodes. It also wins over the opponent for most of the times with only two hands. The search depth on the other hand is on average a bit larger compared to the BFS algorithm. Random search performs the worst out of all three, not only does it expand many nodes, it also doesn't win very much.

2.2 What I learned in Task 2

In this task I increased my understanding of reading and understanding code someone else had written, and also the importance of sorting your data on the correct key to increase the performance of a search algorithm.