Artificial Intelligence and Applications Coursework

Contents

1 Implement a heuristic search algorithm: A* and the 8-puzzle game.

1.1 How you would frame the 8-puzzle problem as a search problem?

Framing the 8-puzzle as a search problem is fairly simple. A graph can be constructed, where the nodes represent different states of the game and an arc between the nodes represents that one game state can be reached by making a legal move from the other. Each arc will have a weight of 1 so that when it is searched a minimal solution is found. This graph can then

be searched from the start state until the goal state is reached, to obtain a set of moves to solve the problem. Another way this could be done is by constructing a tree from the start state in much the same way as before, where each node is a game state and the children of each node is a game state that can be reached from the game state by making a legal move. The tree could terminated at a certain depth where generating more nodes would be computationally infeasible or where a solution would definitely have been found. A breadth first search could then be run on the tree to find the shortest path.

1.2 Solve the 8-puzzle problem using A*

1.2.1 Briefly outline the A* algorithm

The A* search algorithm is a heuristic graph search algorithm. This means that when it is deciding which node to travel to next, it not only considers the the total length of arcs travelled, but also a heuristic function. For example if A* is being used for calculating the shortest route between two cities, the heuristic function may be the straight line Euclidean distance between the current position and the goal city. In this case the considered value at a node would be the total distance to travel to that position plus the euclidean distance to the goal.

1.2.2 Describe two admissible heuristic functions for the 8-puzzle problem

One heuristic function that could be used is the Hamming Distance, i.e. the number of tiles that are mismatched from the goal state. This is admissible as the number of moves needed to reach the goal state is at least the total number of mismatched tiles. This means that it is optimistically estimating how many moves left to reach the goal state, this means that the A* will be optimal.

Another admissible heuristic function is the Manhattan distance. This is the sum for all tiles of the total number of horizontal and vertical moves that would be needed to get a tile from its current position to its position in the goal state. For example to get a tile from (2,3) to (1,1) the tile needs to move 1 to the left and 2 down, so the distance for this tile is 2+1=3 and the Manhattan distance for the game state would be the sum of these over all the tiles. This again is admissible as it underestimates the number of moves needed to reach the goal state. This heuristic should be better than the Hamming distance as it is less optimistic and should reflect how far away you are from the goal state.

1.2.3 Implement two versions of the A* algorithm

1.2.4 Briefly discuss and compare the results given by A*

As expected A* worked faster when the chosen heuristic function is Manhattan distance to when Hamming distance is used. This is because the Manhattan distance is a less optimistic heuristic than Hamming distance is. This is apparent by calculating the heuristics for the start state. The Manhattan distance of the start state is 18, whereas the Hamming distance is only 8. This is a drastic difference and affects the performance of A* greatly. By using the *timeit* library in python I ran the search on the start configuration ten times for each heuristic and calculated the average time to run. The result of this was that it took on average 3 seconds for the A* to reach the goal state using the Manhattan distance, while the average time to find the goal using Hamming distance was 444 seconds. To calculate this average I set the *timeit* function to run A* with each heuristic 10 times. Ideally I would have used a larger sample size however the run time of using the hamming distance made this infeasible. This shows how important it is to use a heuristic function that is representative of the work left to reach the goal, while still being admissible.

1.3 Bonus question

2 Unsupervised learning: k-means clustering

2.1 Describe k-means clustering and how it applies to the task of hand-written digit recognition.

K-means Clustering is an unsupervised learning method of finding patterns in data. Given k, the algorithm will sort your training data into k clusters

2.2 Using k-means

2.2.1 Analyse the data using k-means clustering

2.2.2 Present and discuss the results of the k-means clustering on the dataset

2.3 Limitations of k-means

2.4 Bonus question