Artificial Intelligence for Robots - Homework 3

Santosh Thoduka and Muneeb Shahid

Due Date: October 29, 2013

1 Question 1

When is IDS better than BFS and DFS?

IDS is better when dealing with a dense graph, or a graph with high depth.

- BFS: IDS beats BFS in terms of space complexity. IDS is more likely to use less memory, this is especially true when dealing with more depth.
- DFS: IDS is better than DFS in terms of time complexity. Particularly if the solution lies right next to the start, and DFS starts in the opposite direction. IDS is much more likely to return a solution quicker.

Test Case: Binary Tree
Total No of nodes: 8000
Starting Position: 1
Goal Position: 15
Results:

- IDS Nodes expanded: 26 Elements in memory: 4
- BFS Nodes expanded: 15 Elements in memory: 15
- DFS Nodes expanded: 7169 Elements in memory: 12

2 Question 2

Define the state space and heuristics to use A*

The state space for this problem are the stations and the connections between them. Each station can be considered as a node in a graph, and the connections between them the edges.

For the purpose of finding the shortest path between two given nodes at least two heuristics can be considered to estimate the cost of reaching the end node from any given node. For example:

1. Euclidean distance (L_2 norm) between current node and end node
   This heuristic is admissible because it is less than or equal to the actual cost of travelling from a node to the end node. If the current node is directly connected to the end node, the actual cost will be exactly the euclidean distance. If it's not directly connected, the cost will be the sum of euclidean distances to each node in the path (i.e more than the direct distance from current node to end node)

2. Rectilinear distance (L_1 norm) between current node and end node This heuristic is not admissible since in some cases it overestimates the cost of moving from current node to end node. The sum of x and y displacement is more than the direct distance to a neighbouring node. If the current and end node are neighbours the rectilinear distance would be higher than the euclidean distance.

The Euclidean distance heuristic was chosen for this implementation.

Also consider the case when the starting point is not a Train station on the map, but a given point in cartesian coordinates.

The case of starting from a "non-station" point can be solved by making the current position a node and adding connections to all stations. However, the weights associated with the connections will be higher than the connections between stations. For example, by assuming the train travels at an average speed of 50km/hr and an average human walks at 5 km/hr, the weight assigned to the new connections will be 10 times higher than the station-station connections. The A^* algorithm will find a path that minimizes the walking distance while also considering the connections between the train stations. Ideally it should direct the user to the closest station and continue the journey by train.

Finally, consider the case that after planning a path, a delay can occur at random in any path segment. How would you change the heuristic to consider such random events? (For example, considering that each path segment can have a success probability).

There must be prior knowledge about the probability of delays on a given connection for it to have an effect on the heuristic function. If such knowledge exists, then the edges (connections) can be weighted (as done in the case of connections by walking) to have higher costs as the probablility of a delay increases. If there is no prior knowledge, all connections have to be treated the same and the heuristic should not be affected by the possibility of a delay.