40205163 Coursework Report

# Calum Hamilton

## Types Of Searches:

### Iterative Deepening Search:

It is a search technique that limits a search to a certain depth, to try and efficiently find the results in an ever-expanding tree. It combines benefits from both Depth-First and Breadth-First, by following a Depth-First search pattern while using much less memory on each iteration than Breadth-First Searching. It tries to find a search at the shallowest goal node, while using as little memory as possible and can be a very strong choice in the case of a search tree with a huge branching factor.

The iterative deepening search method would take individual nodes from the map, and search through them one by one. It would go to a certain depth in the tree and attempt to find the goal node. If it couldn’t find the goal node, it would then analyse from the nodes it had reached, which would be the most likely node to lead to the goal node and continue.

### Pros:

* Uses very little memory
* ‘Best of both worlds’ of depth-first and breadth-first searching
* Finds the result at the shallowest node on the search tree

### Cons:

* Struggles to find solutions further down the tree
* Can take a lot of time if the branching factor is high, further down the tree
* Needs to limit search to only a few levels to retain efficiency

### Bidirectional Search

It is a search technique that has 2 different searches at once. One search starts from the start node and one search starts from the end node. The searches then stop when they meet each other in the middle. Each tree will check to see if the new node belongs to the other search before proceeding, meaning if the end node finds a frontier node on the start tree, they can link the trees up through those nodes. It can be a very efficient way to search, especially if the search is not immediately apparent to the program, or high up the search tree.

The bidirectional search method would start at the beginning node and end node of the tree. They would then continue to try each possible route, in a breadth-first search style, until the two searches collided. They would then connect the start node to the end node through the frontier node that joined the two searches.

### Pros:

* Very time efficient compared to a lot of other algorithms
* Fairly space efficient
* Can be useful joining the trees together, through non-obvious nodes

### Cons:

* Can be difficult to program
* In the cave system, because some caves are one way, the end and start nodes could go different ways, or the end node could go through a one way system to meet a frontier, and then the start node can’t join the end

### Dijkstra’s Algorithm

It is a search technique that is good for finding the shortest distance between two points, by finding the shortest paths between nodes in a graph. The most common variant of the algorithm selects a single node as a “source” node and finds the shortest paths from the source to all other nodes in the tree, producing a shortest-path tree. It is designed to find the shortest path between a single node and all other nodes in the tree.

In the case of the cave diagram, it would stop searching when it has reached the goal cave. It is an ideal algorithm for the cave system, as it is specifically designed to find the shortest distance between two points.

### Pros:

* Efficient for the task at hand
* Good at finding the shortest distance in a real-life situation
* Can change algorithm to cut out nodes that are further away than the goal node

### Cons:

* Can take a lot of space
* Can take a lot of processing power
* Searches every possible node until it reaches the goal node

### For this project I will be using a mix of Dijkstra’s Algorithm and Iterative Deepening Search. I believe Dijkstra’s Algorithm is slower than Iterative Deepening Search, while IDS isn’t as easy to use in a ‘real life’ situation, like the cave mapping. I will try and use the best of both techniques.