COMP2208 - Search Methods Coursework

Daniel Best (Student ID: 29777127)

November 25, 2019

1 Approach

My approach to this assignment was to first create support classes that would handle any details that didn't directly relate to the search algorithm itself. These are as follows:

1.1 Node

This class defines both a singular node in the tree structure, as well as the entire tree structure itself by means of its **parent** and **children** variables. It contains a single **Grid** object, the **value** variable. It also implements the Comparable interface, where it compares an optional **estimatedCost** variable - a feature that was specifically added for the A* Search algorithm.

1.2 Grid

A class that handles the state by means of manipulating a **char[][]** multidimensional array. This class stores the actual state of the problem, storing the location of the **agent** and all of the non-white space blocks; it also allows for that state to be manipulated in a multitude of ways:

- Generates the start and solution state.
- Moves the agent in the grid, thereby changing the location of both the agent and the block it moves to.
- Calculates the **Manhattan distance** between the Grid and another Grid object passed to it, which is used as the heuristic for A* Search.

1.3 Search

An abstract class that defines a common start and solution state for each of the individual search methods, and provides a common **expandNode()** method, which generates the children of a given node.

Using this **Search** class, I was able to easily implement a class for each of the four search algorithms:

1.4 Breadth First Search (BFS)

Uses a **Queue** to store expanded nodes, meaning nodes are checked in the order they are expanded.

1.5 Depth First Search (DFS)

Uses a **Stack** to store expanded nodes, meaning the last node to expanded is checked next.

1.6 Iterative Deepening Search (IDS)

Uses **Depth Limited Search (DLS)**, a modified version of **DFS** that does not expand nodes at a given depth, which then iteratively increases this limit.

1.7 A* Heuristic Search

Makes use of an evaluation function to determine which node to pick next, which is the **depth** of the node plus the **Manhattan distance (heuristic)** to the solution.