Brandon Choi

Professor Tang

CS420

23 October 2016

Project 1 Report

**APPROACH**

The A\* algorithm, particularly its implementation in solving the 8-puzzle problem, needs substantial setup. In order to ensure that implementing the A\* algorithm would go smoothly, I created two accompanying classes: Board.java and Node.java. These were essentially made to logically organize the code and encapsulate the proper parts of the 8-puzzle problem. The Board object was responsible for all functionality relating to the creation and modification of a single puzzle. It handles display, generation of possible move sets, and varying functions relating to the relative state of the puzzle. I decided to have the puzzle board represented as a one-dimensional array, rather than a two-dimensional array because I wanted to test myself and also see the limitations of implementing the board this way. In any case, the results are completely identical, and one could definitely make a case for using either structure. The Node object plays a larger role in the functionality of the entire program because each of the Node objects represents a part of each puzzle’s solution tree. It is also a key player in determining the path from the starting node to the goal node because each Node object holds references to its parent. Finally, the main class, puzzle8Solver, handles the implementation of the A\* which becomes slightly easier to manage when accompanied with the Board and Node classes.

**COMPARISON OF HEURISTICS**

**h1 = # of misplaced Tiles**

**h2 = Manhattan Distance**

|  |  |  |  |
| --- | --- | --- | --- |
| **Depth** | **Search Cost**  **A\*(h1)** | **Search Cost**  **A\*(h2)** | **# of Cases**  **Tested** |
| **2** | **4** | **4** | **200** |
| **4** | **12** | **12** | **200** |
| **6** | **22** | **17** | **200** |
| **8** | **41** | **26** | **200** |
| **10** | **96** | **41** | **200** |
| **12** | **250** | **71** | **200** |
| **14** | **563** | **113** | **200** |
| **16** | **1282** | **215** | **200** |
| **18** | **3089** | **354** | **200** |
| **20** | **7210** | **681** | **200** |
| **22** | **18117** | **1234** | **200** |
| **24** | **39115** | **1603** | **200** |

As shown from the chart, there is quite a large gap in performance between heuristic functions. This is important to note because heuristic functions play a large part in the efficiency of a particular search algorithm. Since A\* depends largely on its heuristic function for results, it is important to design a heuristic function that will optimize the search.

**OTHER ANALYSIS AND FINDINGS**

Although I was unable to gather enough data for the average time for differing solution depths, I noticed in my testing that the solutions using the better heuristic would come out much quicker. This is only noticeable when having large solution depths, but if one times the function, then it becomes clear that using Manhattan Distance over misplaced tiles makes the A\* algorithm much quicker.