**Brian Craft**

**CSC480 AI Homework 1**

**Youtube:**

<https://youtu.be/qsNAbmeU50E>

**Github:**

<https://github.com/priorfire4411/Craft-Repo/blob/master/artificial_intelligence/Assignment_1_8_Puzzle_Solver/puzzle_solver_module.py>

**Easy**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Search Algorithm** | **Length**  **(number of moves to solution)** | **Cost**  **(total path cost)** | **Time**  **(In seconds)** | **Space**  **(Longest Queue Length)** |
| **Breadth First** | 5 | 17 | .01 | 63 |
| **Depth First** | 2,319 | 10,399 | .28 | 5,248 |
| **Best First** | 5 | 17 | .001 | 15 |
| **Uniform Cost** | 5 | 17 | .001 | 25 |
| **Iterative Deepening** | No Convergence | No Convergence | No Convergence | No Convergence |
| **A Star Manhattan Distance** | 5 | 17 | .001 | 13 |
| **A Star Out of Position** | 5 | 17 | .001 | 14 |
| **A Star Euclidean Distance** | 5 | 17 | .002 | 13 |

**Medium**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Search Algorithm** | **Length**  **(number of moves to solution)** | **Cost**  **(total path cost)** | **Time**  **(In seconds)** | **Space**  **(Longest Queue Length)** |
| **Breadth First** | 9 | 31 | .02 | 379 |
| **Depth First** | No Convergence | No Convergence | No Convergence | No Convergence |
| **Best First** | 39 | 141 | 2.16 | 1,322 |
| **Uniform Cost** | 9 | 31 | .016 | 156 |
| **Iterative Deepening** | No Convergence | No Convergence | No Convergence | No Convergence |
| **A Star Manhattan Distance** | 9 | 31 | .01 | 51 |
| **A Star Out of Position** | 9 | 31 | .01 | 72 |
| **A Star Euclidean Distance** | 9 | 36 | .01 | 36 |

**Hard**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Search Algorithm** | **Length**  **(number of moves to solution)** | **Cost**  **(total path cost)** | **Time**  **(In seconds)** | **Space**  **(Longest Queue Length)** |
| **Breadth First** | No Convergence | No Convergence | No Convergence | No Convergence |
| **Depth First** | 15,296 | 68,936 | 23.96 | 34,653 |
| **Best First** | 42 | 204 | .04 | 234 |
| **Uniform Cost** | No Convergence | No Convergence | No Convergence | No Convergence |
| **Iterative Deepening** | No Convergence | No Convergence | No Convergence | No Convergence |
| **A Star Manhattan Distance** | No Convergence | No Convergence | No Convergence | No Convergence |
| **A Star Out of Position** | No Convergence | No Convergence | No Convergence | No Convergence |
| **A Star Euclidean Distance** | 38 | 170 | .936 | 973 |

**Analysis**

After running this analysis, I found out how frustrating the depth first and iterative deepening can be. In my code, since there is only 9 places the blank space could be, I hard coded the possible moves. Little did I know the order in which I added the moves would produce vastly different results for depth first and iterative deepening searches. For sanity check, I had to create a super easy state to make sure the searches were actually coded correctly. This led me to learn just how easy it is to get caught in a dead end, or rabbit hole so to speak, when searching the tree using these methods. To further learn this, I adjusted the order in which I put appended the moves to nodes (for instance right before left and vice versa), which led me to go from no convergence to quick convergence. I also saw how efficient the cost based searches can be. This was made abundantly clear when running the ‘hard’ configuration of the 8 puzzle. Specifically, the hard configuration didn’t solve for some of my searches, but when using the Euclidean distance as a heuristic cost, the convergence was quick and efficient in all regards.

Overall, this gave me a good idea of the tradeoffs of each search method. If I was to have a problem with a limited search space, I would likely implement a simple breadth first given computing power currently. However, in just about every other case, I would implement something with the cost factored in, such as a star or best first. These seemed to be the most consistent performing algorithms.

In conjunction, I found the efficiency properly coded data structures can employ. In the case of my code, I began with a node and tree class, and then added in a queue class, which made popping off the nodes far more efficient. As well, I was able to condense quite a bit of code so it was clear and concise. While this was likely not the main goal of the assignment, it was a well-received tangential lesson.