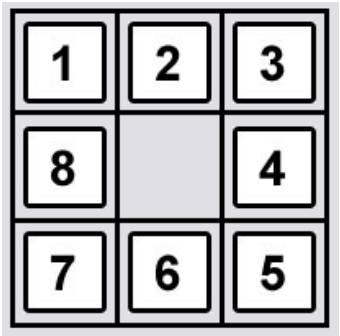

8-Puzzle Solver

Harrison Williams (hwill006)

CS205 - Introduction to Artificial Intelligence
University of California, Riverside

May 8th, 2025

Introduction

The first project for Introduction to AI taught by Dr. Eamonn Keogh is developing a program to solve the 8-puzzle problem using three different search algorithms; Uniform Cost Search, A* search using the Misplaced Tile heuristic, and A* search using the Manhattan Distance heuristic. The 8-puzzle created in 1874 by Noyes Palmer Chapman is a 3x3 square with 9 slots, where there are 8 pieces numbered from one to eight and one empty space.[1] The starting state of the puzzle is the 8 numbered pieces scrambled in random order and the goal is to place the 8 pieces in numerical order going from left to right and top to bottom. This project's goal is to solve this puzzle efficiently with the three algorithms mentioned above. The implementation follows an object-oriented approach written in Python. This report will explore the design of the program and the three implemented algorithms followed by a performance comparison of the three algorithms using a few test cases from trivial to difficult.

Overview of Algorithms and Heuristics

Uniform Cost Search (UCS)

This algorithm is considered an uniformed or blind search which means the search algorithm searches its problem space without any additional information.[2] This algorithm follows a breadth-first search approach which is considered to be a complete and optimal algorithm, but its additional benefit is that it can also handle various edge weights.[3] It selects the cheapest node to expand to keep the overall cost (denoted as $g(n)$) to the solution depth as low as possible.[2] However, in terms of performance, this algorithm has an exponential time and space complexity of $O(b^d)$ which is quite expensive.[4] For this project, each operator's cost to move a piece is 1, so this means that the depth at which the goal state is met is the total cost of the algorithm.

[1]UC Berkeley GamesCrafters Research Group. https://gamescrafters.berkeley.edu/site-legacy-archive-sp20/games.php?puzzle=8puzzle

[2]Kendall, G. 5AIAI: Blind Searches: Blind Search Methods.https://www.cs.ucdavis.edu/~vemuri/classes/ecs170/blindsearches_files/blind_searches.htm

[3]CS440 Lectures. (n.d.). https://courses.grainger.illinois.edu/cs440/fa2021/lectures/search4.html

[4]Keogh, E. Blind Search_part2. https://www.dropbox.com/scl/fo/lucvvfzc0fi7zf3tdlvpx/AJFYrHAXs3fO1nj_OeFxwvc?dl=0&e=18&preview=2__Blind+Search_part2.pptx&rlkey=cpr5hsj0grbd3pueqm05iao21

A-Star Search

The A* search algorithm is an informed search algorithm, also called heuristic search as it uses a heuristic function (denoted as $h(n)$) to estimate the cost remaining to the goal state from the current state.[5] It will estimate this cost by the following function; $f(n) = g(n) + h(n)$.[5] This algorithm combines the best features from the complete and optimal Uniform Cost Search algorithm and the fast and greedy Hill Climbing algorithm.[6] A* is the fastest algorithm dependent on the quality of the heuristic and polynomial space complexity of $O(b \cdot d)$.[6] For this project, the $h(n)$ value is estimated using the Misplaced tile or Manhattan distance heuristics.

Misplaced Tile Heuristic

This heuristic is combined with A* search where it simply counts the number of misplaced pieces in the puzzle or pieces not in their goal state positions as an estimate to the goal state.[7] This $h(n)$ value is added to the $g(n)$ value which is essentially the cost spent expanding previous nodes to the current node. Both of these values are summed to build the f(n) value which is used to select the child node from the frontier to expand.

Manhattan Distance Heuristic

This heuristic is considered the "standard" for 2-dimensional grids where the operators move along the horizontal and vertical x-y axes.[8] The following equation is used to determine the distance of two points on a plane: $|x1 - y1| + |x2 - y2|$.[9] The idea is to apply this distance equation to n-number of points and take the sum of those distances to estimate how close or far the current state's point on the path is to the goal state. In terms of this project, the Manhattan distance equation is to each tile and sum those values to build the heuristic $h(n) = \sum(|x1 - y1| + |x2 - y2|)$ value for each state.

---

[5]The A* Algorithm: A complete guide. Datacamp. https://www.datacamp.com/tutorial/a-star-algorithm

[6]Keogh, E. Heuristic Search. https://www.dropbox.com/scl/fo/lucvvfzc0fi7zf3tdlvpx/AJFYrHAXs3fO1nj_OeFxwvc?dl=0&e=18&preview=3__Heuristic+Search.pptx&rlkey=cpr5hsj0grbd3pueqm05iao21

[7]Tuccar, M. Analyzing the A* search heuristics with the solutions of the 8-Puzzle. https://www.cs.uml.edu/ecg/uploads/AIfall12/tuccar_project.pdf

[8]Heuristics. (2025). https://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html

[9]Manhattan distance. https://xlinux.nist.gov/dads/HTML/manhattanDistance.html

Design

To complete the 8-puzzle project, the design of the program is split into four files; main.py, search.py, node.py, and tree.py. The file containing main() serves as an user-interface for the users to interact with the 8-puzzle. It will prompt the user to choose the default "hard-coded" puzzle or enter a custom puzzle, followed by their choice of algorithm to execute. Based on the user's algorithm choice, it calls the generic search algorithm in search.py passing in the boolean flag parameters on the user's choice of puzzle. The search.py file is responsible for creating the root node of the puzzle's starting state, creating the frontier, and maintaining the frontier with its child nodes via a heap priority queue. The other half of the program contains the Node and Tree classes. The Node class serves as an efficient approach for the creation of node objects representing various states of the puzzle and its child states to expand after applying the possible operators. Node objects track its own state in relation to the goal state, depth, moves, child nodes and most importantly the heuristic for the search algorithms. Also, the logic for the heuristic functions are implemented in this file as part of the class functions along with the node expansion routine. Once arriving at the goal state, the tree object tracking the expanded nodes is able to quickly trace the path from the goal state back to the initial state. This approach is ideal for outputting clean traces through the solution path.

Performance

Test Cases

All puzzle test cases are randomized and built with an online tool.[10]

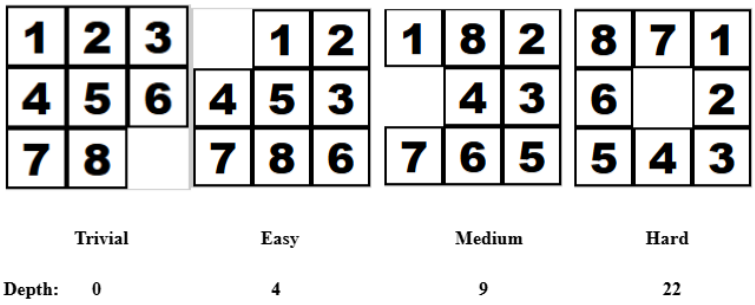

Figure 1: Test cases used for the 8-puzzle solver and their solution depths

[10]Gurkaynak, D. 8-Puzzle Solver. `https://deniz.co/8-puzzle-solver/`

Results

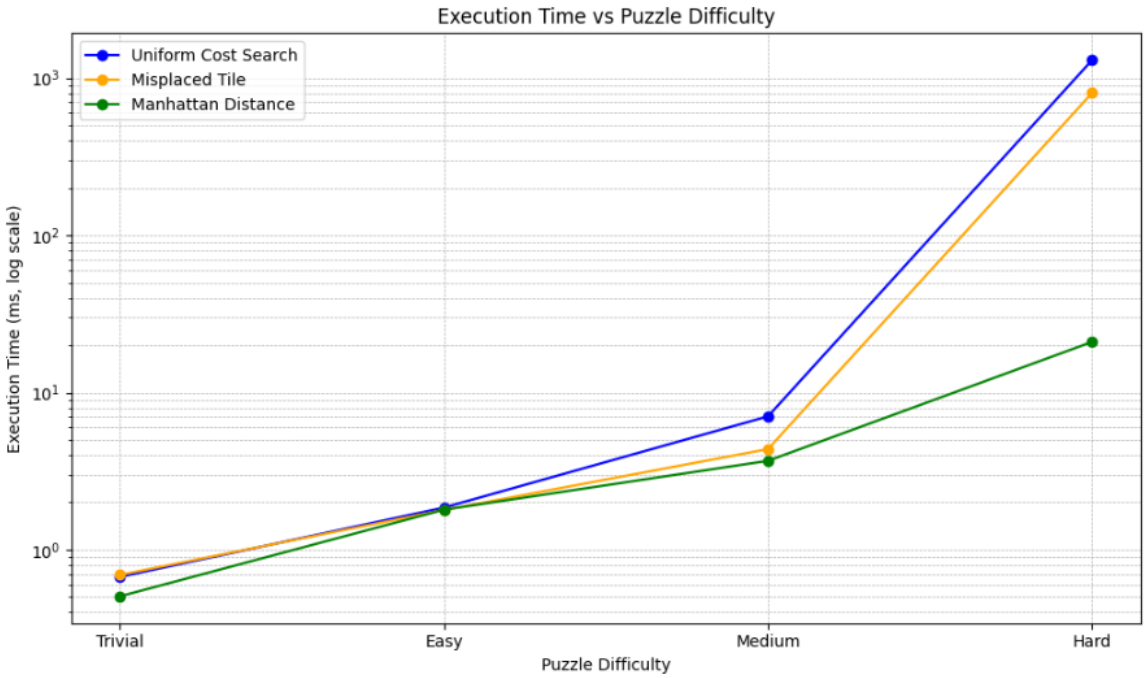

Figure 2: Time measured by puzzle hardness using three algorithms

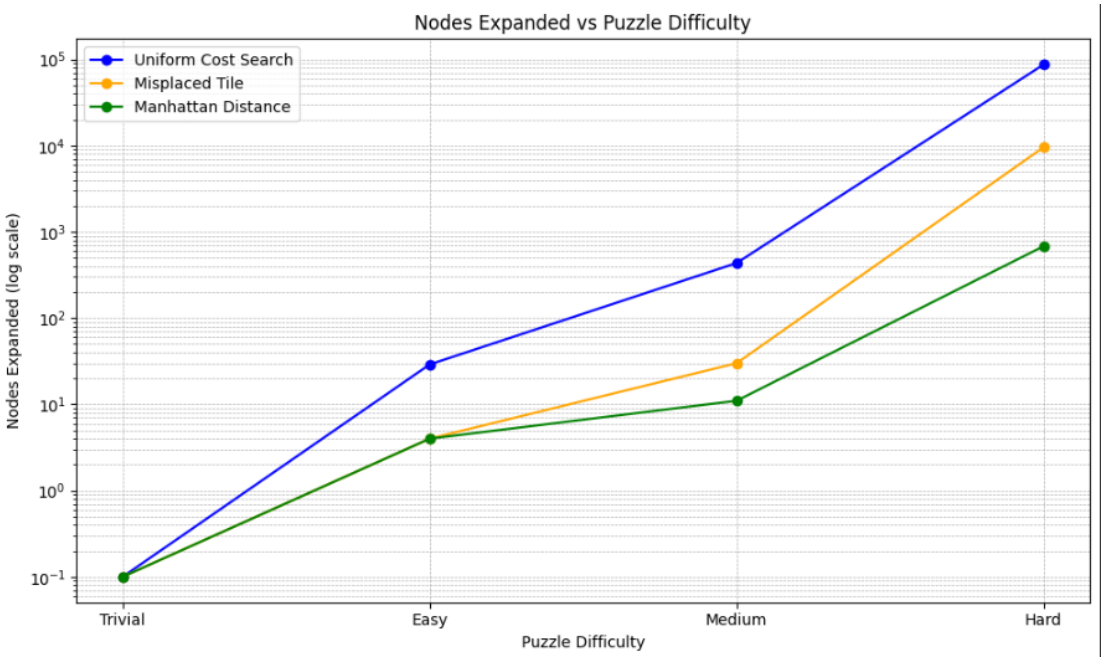

Figure 3: Nodes expanded by puzzle hardness using three algorithms

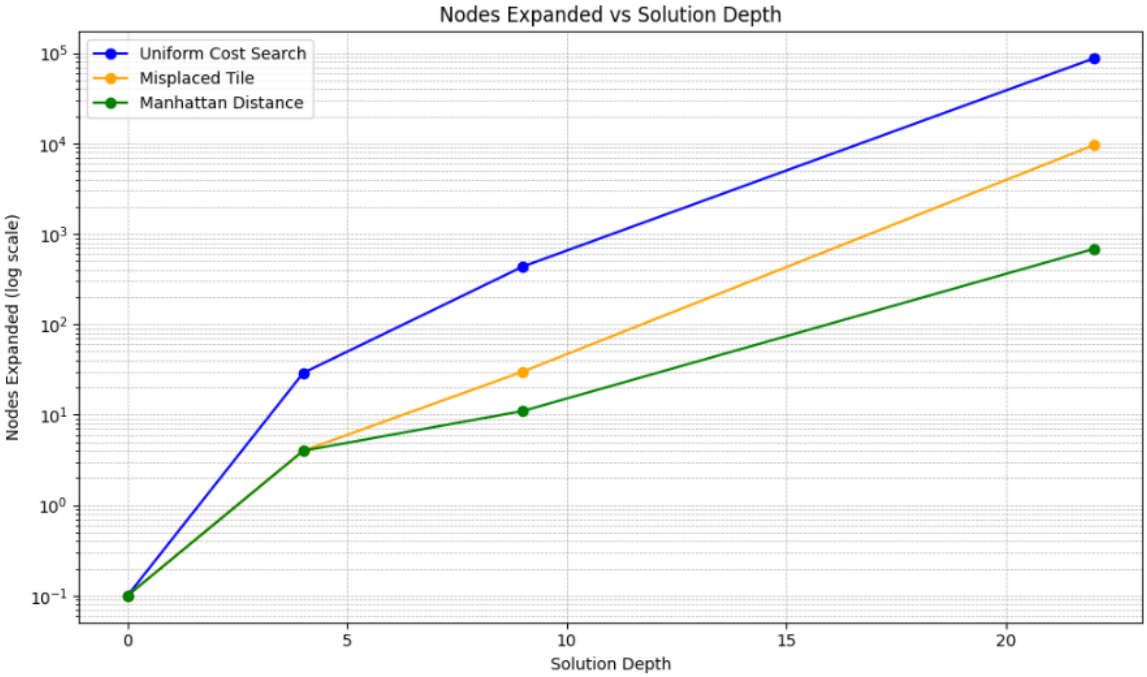

Figure 4: Nodes expanded by three algorithms and their solution depths.

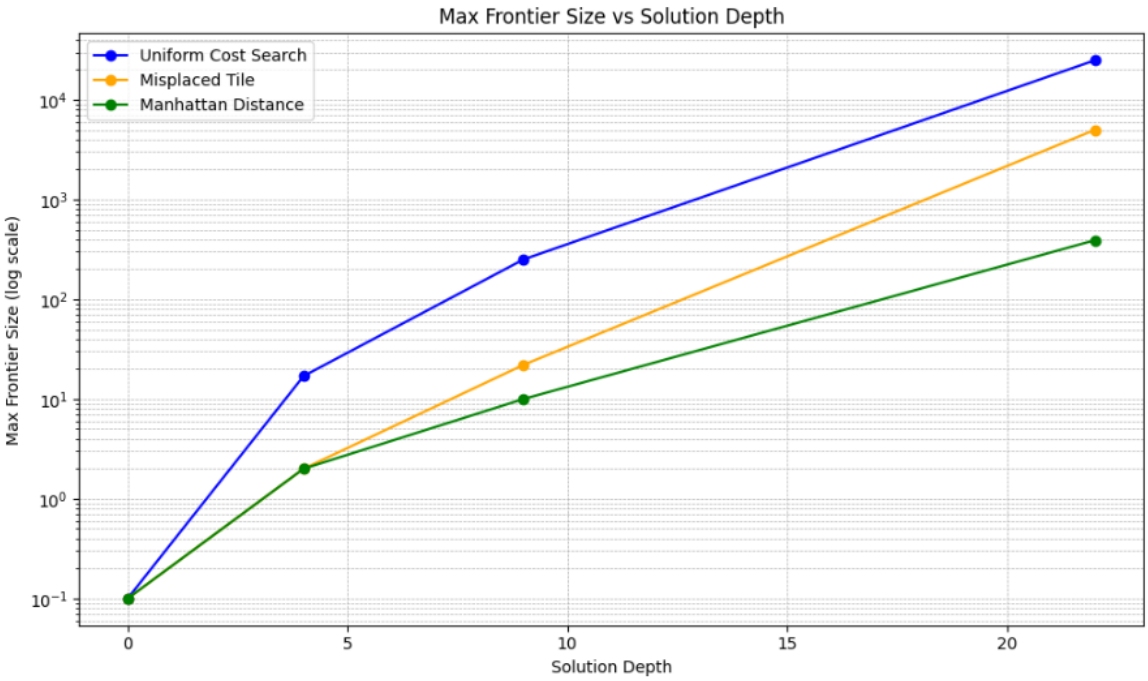

Figure 5: Max frontier size by three algorithms and their solution depths

Nodes Expanded via Algorithms

	Uniform Cost Search	Misplaced Tile	Manhattan Distance
Trivial	0	0	0
Easy	29	4	4
Medium	435	30	11
Hard	87129	9577	682

Max Frontier Sizes via Algorithms

	Uniform Cost Search	Misplaced Tile	Manhattan Distance
Trivial	0	0	0
Easy	17	2	2
Medium	251	22	10
Hard	24982	5014	392

Runtimes via Algorithms

	Uniform Cost Search	Misplaced Tile	Manhattan Distance
Trivial	0.000672102 s	0.000692129 s	0.000505686 s
Easy	0.001853704 s	0.001790285 s	0.001798391 s
Medium	0.007052422 s	0.004368305 s	0.003679037 s
Hard	1.299380541 s	0.803491592 s	0.020983696 s

Solution Depths via Algorithms

	Uniform Cost Search	Misplaced Tile	Manhattan Distance
Trivial	0	0	0
Easy	4	4	4
Medium	9	9	9
Hard	22	22	22

Figure 6: Tabular Values of Nodes Expanded, Max Frontier Size, Runtimes, Solution Depth

Conclusion

After interpreting the results from running the 3 algorithms on 4 different puzzles of varying difficulty, it is clear that A* search is far superior than Uniform Cost Search in terms of time and space complexity. Also within A*, its evident that using the Manhattan Distance heuristic results in better quality results especially for nodes expanded as well as Max frontier sizes. However, the common trend is that the harder the puzzle, meaning the deeper the solution depth, the more time and memory it takes to reach the goal state.

Example Trace

The following trace is performed on the medium puzzle with a solution depth of 9.

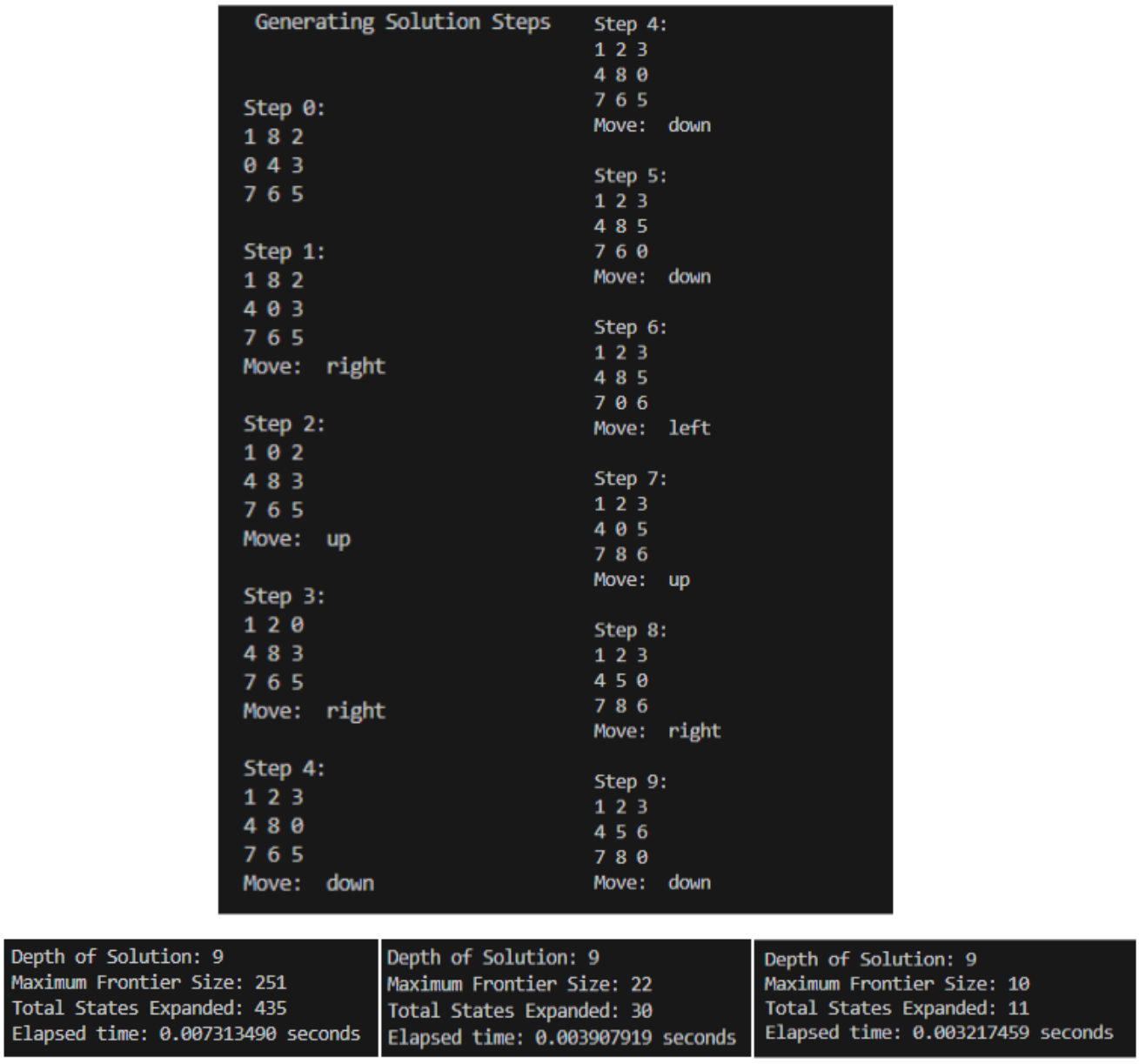

Figure 7: Trace of Solution for Medium Puzzle difficulty with 3 algorithms applied