Naomi Igbinovia
Joshua Phillips
CSCI 4350
2 October 2023

Informed Search and the A* Search Algorithm

Introduction

The eight-puzzle problem, based around the n-puzzle traditionally used for modeling algorithms with heuristics, involves a 3x3 board with tiles numbered one through eight and a blank tile. The goal of the puzzle is to rearrange the board's given tiles set by a series of random moves, to the board's base state (also known as the goal state). This can be achieved by sliding around the board's blank tile. Our board's goal state was the following from left to right in three rows: 0 (the blank tile), 1, and 2 on the first row, 3, 4, and 5 on the second row, and 6, 7, and 8 on the third row. The blank tile can be moved up, down, left, and right.

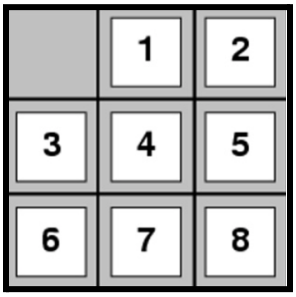

Figure 1. an 8-puzzle problem goal state.

Code Development

The solving of this algorithm can be broken into two parts: creating the puzzle and solving the puzzle. In creating the puzzle, the random-board.py program was developed. This program takes in inputs of a seed number and several random moves to create a randomized configuration from the base board state and said base board state is printed to the user. In solving the puzzle, the astar.py program was developed. This program takes in the randomized board configuration read in from left to right in a single line and an integer value corresponding to a matching heuristic option to solve the given board configuration by employing A* search. A* search is an informed search algorithm that uses path cost information and heuristics to find its solution.

The astar.py program utilizes two classes prior to the main function, the Node class and the Explore class. The Node class represents a given node in the search tree. Each instance or node of this class contains a two-dimensional list that represents the puzzle's current state called 'nodes', a dictionary that maps out each tile value to its corresponding position in the puzzle called 'node_positions', the path cost to the given node from the starting node represented by 'g', and a reference to the parent node in the search tree called 'parent'. The Explore class solves the board configuration using the A* search algorithm and contains different heuristic functions that can be used.

Here is how the A* search works: first, the algorithm is implemented with the use of priority queue called 'open list' to explore the nodes. Then, the nodes are expanded based on their estimated cost [f = g + h] where the path cost is from the starting node and h is the heuristic value. Lastly, the algorithm continues to expand nodes until all nodes are explored or

the goal state is reached. When the search has been completed and a solution has been found, the total number of visited nodes, the maximum number of nodes stored in memory, the optimal solution's depth, and the approximate effective branch factor are printed to the user, along with the optimal path.

Heuristics Implemented

The program has four different heuristic functions that can be chosen by the user. No Heuristic [h(n) = 0] assigns zero cost to all of its states and reduces A* search down to uniform cost search. Misplaced Tiles Heuristic [h(n) = 1] counts the number of tiles displaced from their correct positions compared to the goal state. Manhattan Distance Heuristic [h(n) = 2] calculates the sum of horizontal and vertical distances of each tile form its goal position. Manhattan Distance + Linear Conflicts Heuristic [h(n) = 2] was the choice of a novel heuristic, and considers the sum of horizontal and vertical distances of each tile form its goal position in pair with when two tiles in the same row or column need to cross each other's path to reach their goal locations.

Experimentation

To test the performance of the A* search algorithm and the different heuristics available, 100 random boards were generated using the random-board.py. program. Each board was then solved using all four algorithms; the printed statistics were stored in a table to later calculate the minimum, median, mean, maximum, and standard deviation value for each statistic. The final calculations are displayed below.

Total Number of Nodes Visited

Heuristic	Minimum	Maximum	Median	Mean	Standard Deviation
h(n) = 0	11	158747	7689	21297.11	32065.56
h(n) = 1	3	36770	433.5	433.5	5385.31
h(n) = 2	3	3050	83	267.19	501.24
h(n) = 3	3	4667	175.5	523.65	885.71

Maximum Nodes Stored In Memory

Heuristic	Minimum	Maximum	Median	Mean	Standard Deviation
h(n) = 0	32	426492	21255	57467.71	86270.19
h(n) = 1	12	99112	1192	5872.48	14532.75
h(n) = 2	12	8140	226.5	718.47	1334.75

h(n) = 3	12	12506	475	1413.69	2378.96

Depth

Heuristic	Minimum	Maximum	Median	Mean	Standard Deviation
h(n) = 0	3	26	15.5	14.97	4.91
h(n) = 1	3	26	15.5	14.97	4.91
h(n) = 2	3	26	15.5	14.97	4.91
h(n) = 3	3	28	16	15.39	5.27

Approximate Effective Branch Factor

Heuristic	Minimum	Maximum	Median	Mean	Standard Deviation
h(n) = 0	1.6464	3.41995	1.90078	1.97308	0.26308
h(n) = 1	1.54797	2.28943	1.58689	1.61689	0.11373
h(n) = 2	1.29574	2.28943	1.44326	1.47735	0.14930
h(n) = 3	1.28427	2.28943	1.48795	1.52234	0.16267

Figure 2. a table of all the statistics experimented on all four heuristics with 100 test cases.

Analysis & Performance

After the experiment had been completed, it was found that the Manhattan distance heuristic [h(n) = 2] worked the best when it came to solution depth and efficiency. Even though the Manhattan Distance + Linear Conflicts heuristic was more expensive, deeper solutions were found using it than just the Manhattan Distance heuristic by itself. Lastly, the No heuristic [h(n) = 0] was the least successful of them all, as it produced the most nodes but gave back the shallowest solutions.

Limitations

Even though astar.py and random-board.py were successful in solving the 8-puzzle problem, one of the challenges that was made apparent was memory usage. Storing all the states consumed a lot of memory and took longer to compute in terms of harder problems.

Conclusion

The experiments conducted show how effective the A* search algorithm is in solving the 8-puzzle problem, as well as the importance in how the heuristics can affect the algorithm's performance. The differing heuristic functions can create a search process that gets to its solution the quickest. But it is important to choose the right heuristic based on the problem's traits and the computer's ability.

Sources

"A* Search Algorithm." GeeksforGeeks, GeeksforGeeks, 7 Mar. 2024, www.geeksforgeeks.org/a-search-algorithm/.

"Manhattan Distance + Linear Conflicts Scoring Function for Sliding Tile Puzzle Solver." Code Review Stack Exchange, 1 July 1962, codereview.stackexchange.com/questions/144462/manhattan-distance-linear-conflicts-scoring-function-for-sliding-tile-puzzle-s.

Roy, Baijayanta. "A-Star (A*) Search Algorithm." Medium, Towards Data Science, 4 Feb. 2023, towardsdatascience.com/a-star-a-search-algorithm-eb495fb156bb#:~:text=A%2Dstar%20(also%20referred%20to,s%20to%20find%20the%20solution.

Sonawane, Ajinkya. "Solving 8-Puzzle Using A* Algorithm." Medium, Good Audience, 24 June 2020, blog.goodaudience.com/solving-8-puzzle-using-a-algorithm-7b509c331288.