CS170 Project 1

Brandon Paulsen

October 31st, 2022

UCR CS170 Fall 2022

Professor Eamonn Keough

Contents

List of Figures

1 Introduction

In this project, we are asked to write a program that uses a general search algorithm (A*) to solve 8 puzzles. An 8 puzzle is a puzzle that has 8 tiles arranged in a 3x3 grid, with one space. The goal of the puzzle is to arrange the tiles such that the first row is [1,2,3], the second row is [4,5,6], and the last row is [7,8,empty] by moving tiles around. Tiles next to the empty tile can swap places with the empty tile, or we can understand this as the empty tile swapping with tiles next to it.

A* is a search agorithm that uses both the depth of a state and a heuristic function to gague which unexplored states in the frontier should be explored first. It does this by assigning each state in the frontier a priority that is equal to the sum of the depth and heuristic, and as long as the heuristic is admissible (that is, it never overestimates the number of moves from the state to the goal state), A* will be optimal. In this project, we are tasked with implementing 3 heuristic functions: Uniform Cost, Misplaced Tiles, and Manhattan Distance.

As has been discussed in class multiple times, the branching factor and diameter of a search problem are very important. In fact, the branching factor and the problem diameter of a search problem define the magnitude of the search space, so it might be good to investigate them a little bit. Based on the loose idea that the branching factor is the average of the number of valid moves of a state, we can approximate the branching factor as $4 \cdot \frac{1}{9} + 3 \cdot \frac{4}{9} + 2 \cdot \frac{4}{9} = \frac{24}{9} \approx 2.6$, because there are 4 corner spaces with 2 moves, 4 edge spaces with 3 moves, and 1 center space with 4 moves. This seems reasonable, but if we further consider that we cannot go back to any states previously visited, and at least one move for each state goes back to a previous state, we can reduce the approximate branching factor to $3 \cdot \frac{1}{9} + 2 \cdot \frac{4}{9} + 1 \cdot \frac{4}{9} = \frac{15}{9} \approx 1.7$. This is a very small branching factor, which is great news for us. We can further extend this approximation of the branching factor to any n puzzle as $\frac{4}{n^2} + \frac{8 \cdot (n-2)}{n^2} + \frac{3 \cdot (n^2 - 4 \cdot (n-2) - 4)}{n^2} = 3 - \frac{4}{n}$.

2 Heuristics

Heuristics are a the heart of the A* algorithm, without them, A* would be no better than breadth first search. As such, it is important to discuss the heuristics used in this project both in terms of their implementation and their impact on space and time complexity.

2.1 Uniform Cost

Uniform Cost Heuristic is the simplest of the 3 heuristics. It simply evaluates to 0 for all states. This means that in effect, A* with Uniform Cost Heuristic devolves to breadth first search and searches level-by-level. This clearly implies that the algorithm will be complete and optimal, though it will be very slow and memory inefficient as well.

2.2 Misplaced Tile

Misplaced Tile Heuristic, though slightly more complex than Uniform Cost Heuristic (which is trivial), is not overly complex. Evaluating the Misplaced Tile Heuristic of a state amounts to counting up the number of tiles in a given state that are not in the same position as they are in the goal state. Though it is much better than Uniform Cost Heuristic, it is clearly not the best heuristic (either in terms of time or memory), as children of a state (those states that are reachable from that state in a move) often are assigned the same priority, and therefore the search algorithm has a hard time distinguishing which is better and should be explored first.

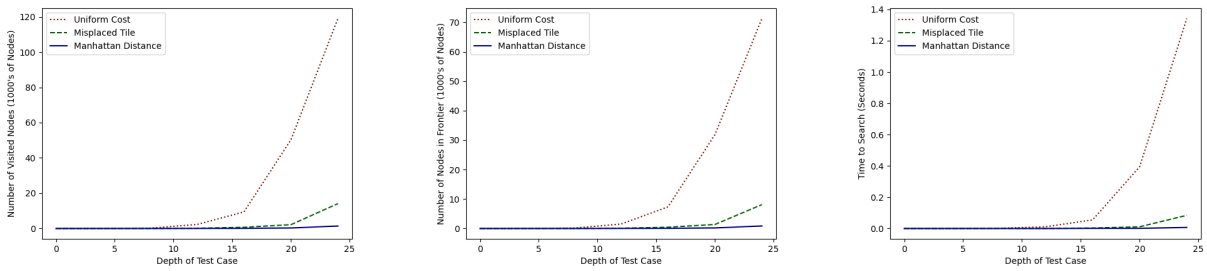
2.3 Manhattan Distance

Though Manhattan Distance Heuristic is the most complex of the 3, it offers some potential benefits that warrant its use. In all cases, Manhattan Distance Heuristic dominates Misplaced Tile Heuristic - that is to

say, its value is greater than Misplaced Tile Hueristic without overestimating the number of moves left and therefore destroying the optimality of the A* algorithm. The Manhattan Distance Heuristic is calculated by totaling the differences in x and y positions of tiles between a given state and the goal state, or in math terms, $\sum_{\text{tiles} \neq \text{empty}} \Delta x + \Delta y$. Though Manhattan Distance Heuristic is admissible, it is clearly not perfect, and there is still room for improvement.

3 Test Cases

Several cases were given in the project assignment. They range from trivial (depth 0) to very complex (depth 24) and make for good tests of algorithms. For a significant portion of time, my implementation of the general search algorithm when used with Manhattan Distance Heuristic failed to find the most optimal path (the path it found was at depth 26). This was very helpful to me in that it indicated an error in my code. The number of vistied nodes, frontier nodes, and searh time trends are plotted below for each heuristic:



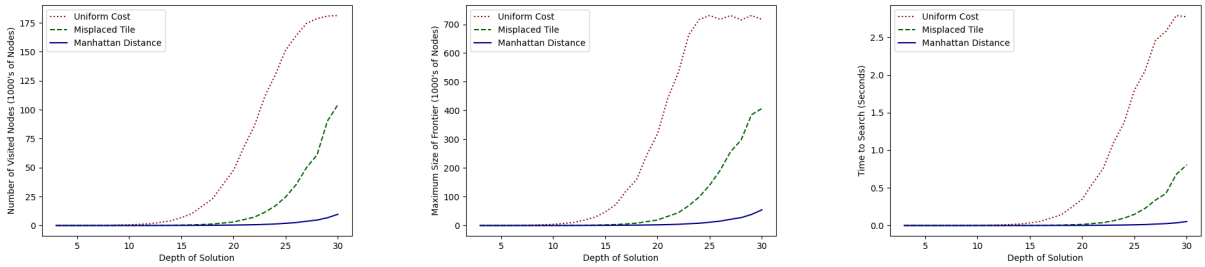
(a) Visited Nodes vs Depth (b) Maximum Frontier Size vs Depth (c) Search Time vs Depth

Figure 1: Test Cases Trend Comparisons

As we can see quite clearly in the plots in Figure 1, Uniform Cost performs significantly worse than either of the other heuristics over the test cases in terms of the visited nodes, maximum frontier size, and search time. Misplaced Tile performs slightly better, though still worse than Manhattan Distance. It is important to note that though these plots seem to indicate that Manhattan Distance is linear in the number of visited nodes, maximum frontier size, and search time, it is very much still exponential (as seen in Figures 2a and 2b), though it grows much much slower than uniform cost.

4 Monte Carlo Simulation

I ran a monte carlo simulation on 10% of the number of solvable states of the 8 puzzle (so, $\frac{9!}{20} = 18,144$ simulations) on each of the heuristics in order to see the trends of these heuristics beyond the test cases. It is probably overkill to run such a simulation, but I just had to satisfy my curiousity. All results of the monte carlo simulation have been plotted using python and included below. Below are plotted trend comparisons for the number of visited ndoes, the number of nodes in the frontier, and the search time vs depth for each heuristic:



(a) Visited Nodes vs Depth (b) Maximum Frontier Size vs Depth (c) Search Time vs Depth

Figure 2: Monte Carlo Simulation Trend Comparisons

In the Monte Carlo simulation, we observe many of the same trends as in the test cases. Uniform Cost search is still much more inefficient in all regards, while Manhattan Distance is stil the fastest. Now, however, in Figures 2a and 2b, all heuristics are clearly nonlinear, whereas in the test cases, it appeared as though they were.

5 Conclusion

In conclusion, through this project, we have seen that each heuristic is optimal and complete, though they vary in their runtimes and memory complexity. Each heuristic is exponential with respect to its time and space complexity, though the Manhattan Distance is clearly much faster and uses much less memory on average than the other two (as can be seen in Figure **??** and Figure **??**, respectively).

Though the code is capable of solving 15 puzzles currently (the object based C++ version), it takes somewhere on the order of 5-10 minutes, so I would want to modify it to implement island based search. I would expect this to significantly speed up the search (though it would no longer be optimal), possibly to something on the order of 30-50 seconds.

6 Code

6.1 C++

My object based C++ code (slower but more memory efficient) can be found at `https://github.com/Poly1581/SlidingPuzzlesV2`. My pointer based C++ code (slightly faster but much uglier and more memory inefficient) can be found at `https://github.com/Poly1581/SlidingPuzzles`

6.2 Python

My python code can be found at `https://github.com/Poly1581/PYSlidingPuzzles`

7 References

In the process of streamlining the randomization function in my object oriented code, I referenced the wikipedia page for the 15 puzzle to get a better understanding of the parity argument that allows for discrimination between solvable and unsolvable states. The page can be found here: `https://en.wikipedia.org/wiki/15_puzzle`.

8 Example Trace of a Simple 8 Puzzle (Manhattan Distance)

What size puzzle would you like to solve?
3 What heuristic would you like to use?
1: Uniform Cost
2: Misplaced Tile
3: Manhattan Distance
3 Would you like to use your own matrix or a random one?
1: User matrix
2: Random matrix
1 Enter tile at row 0, and col 0
1 Enter tile at row 0, and col 1
2 Enter tile at row 0, and col 2
3 Enter tile at row 1, and col 0
5 Enter tile at row 1, and col 1
0 Enter tile at row 1, and col 2
6 Enter tile at row 2, and col 0
4 Enter tile at row 2, and col 1
7 Enter tile at row 2, and col 2
8 Best state to expand with depth (g(n)) 0 and heuristic (h(n)) 0
1|2|3
5|0|6
4|7|8

Best state to expand with depth (g(n)) 1 and heuristic (h(n)) 3
1|2|3
0|5|6
4|7|8

Best state to expand with depth (g(n)) 2 and heuristic (h(n)) 2
1|2|3
4|5|6
0|7|8

Best state to expand with depth (g(n)) 3 and heuristic (h(n)) 1
1|2|3
4|5|6
7|0|8

Best state to expand with depth (g(n)) 4 and heuristic (h(n)) 0
1|2|3
4|5|6
7|8|0

Solution found
Path
1|2|3
5|0|6
4|7|8

1|2|3
0|5|6
4|7|8

1|2|3
4|5|6
0|7|8

1|2|3
4|5|6
7|0|8

1|2|3
4|5|6
7|8|0

Depth of solution 4
Number of visited nodes 5
Max frontier size 5
Search took 0 milliseconds

9 Example Trace of a Difficult 8 Puzzle (Manhattan Distance)

What size puzzle would you like to solve?
3 What heuristic would you like to use?
1: Uniform Cost
2: Misplaced Tile
3: Manhattan Distance
3 Would you like to use your own matrix or a random one?
1: User matrix
2: Random matrix
1 Enter tile at row 0, and col 0
1 Enter tile at row 0, and col 1
6 Enter tile at row 0, and col 2
7 Enter tile at row 1, and col 0
5 Enter tile at row 1, and col 1
0 Enter tile at row 1, and col 2
3 Enter tile at row 2, and col 0
4 Enter tile at row 2, and col 1
8 Enter tile at row 2, and col 2
2 Best state to expand with depth (g(n)) 0 and heuristic (h(n)) 0
1|6|7
5|0|3
4|8|2

Best state to expand with depth (g(n)) 1 and heuristic (h(n)) 11
1|0|7
5|6|3
4|8|2

Best state to expand with depth (g(n)) 1 and heuristic (h(n)) 11
1|6|7
0|5|3
4|8|2

Best state to expand with depth (g(n)) 2 and heuristic (h(n)) 10
1|7|0
5|6|3
4|8|2

Best state to expand with depth (g(n)) 2 and heuristic (h(n)) 10
1|6|7
4|5|3
0|8|2

Best state to expand with depth (g(n)) 3 and heuristic (h(n)) 9
1|7|3
5|6|0
4|8|2

Best state to expand with depth (g(n)) 4 and heuristic (h(n)) 8
1|7|3
5|6|2
4|8|0

Best state to expand with depth (g(n)) 4 and heuristic (h(n)) 8
1|7|3
5|0|6
4|8|2

Best state to expand with depth (g(n)) 5 and heuristic (h(n)) 7
1|0|3
5|7|6
4|8|2

Best state to expand with depth (g(n)) 5 and heuristic (h(n)) 7
1|7|3
0|5|6
4|8|2

Best state to expand with depth (g(n)) 6 and heuristic (h(n)) 6
1|7|3
4|5|6
0|8|2

Best state to expand with depth (g(n)) 1 and heuristic (h(n)) 13
1|6|7
5|3|0
4|8|2

Best state to expand with depth (g(n)) 6 and heuristic (h(n)) 8
0|1|3
5|7|6
4|8|2

Best state to expand with depth (g(n)) 2 and heuristic (h(n)) 12
0|1|7
5|6|3
4|8|2

Best state to expand with depth (g(n)) 7 and heuristic (h(n)) 7
1|7|3
4|5|6
8|0|2

Best state to expand with depth (g(n)) 2 and heuristic (h(n)) 12
1|6|7
5|3|2
4|8|0

Best state to expand with depth (g(n)) 5 and heuristic (h(n)) 9
1|7|3
5|6|2
4|0|8

Best state to expand with depth (g(n)) 8 and heuristic (h(n)) 6
1|7|3
4|5|6
8|2|0

Best state to expand with depth (g(n)) 3 and heuristic (h(n)) 11
1|6|7
4|5|3
8|0|2

Best state to expand with depth (g(n)) 2 and heuristic (h(n)) 12
0|6|7
1|5|3
4|8|2

Best state to expand with depth (g(n)) 4 and heuristic (h(n)) 10
1|6|7
4|5|3
8|2|0

Best state to expand with depth (g(n)) 2 and heuristic (h(n)) 12
1|6|0
5|3|7
4|8|2

Best state to expand with depth (g(n)) 6 and heuristic (h(n)) 8
0|7|3
1|5|6
4|8|2

Best state to expand with depth (g(n)) 3 and heuristic (h(n)) 11
1|0|6
5|3|7
4|8|2

Best state to expand with depth (g(n)) 7 and heuristic (h(n)) 7
7|0|3
1|5|6
4|8|2

Best state to expand with depth (g(n)) 4 and heuristic (h(n)) 10
1|3|6
5|0|7
4|8|2

Best state to expand with depth (g(n)) 1 and heuristic (h(n)) 13
1|6|7
5|8|3
4|0|2

Best state to expand with depth (g(n)) 6 and heuristic (h(n)) 8
1|3|0
5|7|6
4|8|2

Best state to expand with depth (g(n)) 5 and heuristic (h(n)) 9
1|7|3
5|8|6
4|0|2

Best state to expand with depth (g(n)) 5 and heuristic (h(n)) 9
1|3|6
5|7|0
4|8|2

Best state to expand with depth (g(n)) 5 and heuristic (h(n)) 9
1|3|6
0|5|7
4|8|2

Best state to expand with depth (g(n)) 6 and heuristic (h(n)) 8
1|7|3
5|8|6
4|2|0

Best state to expand with depth (g(n)) 6 and heuristic (h(n)) 8
1|3|6
5|7|2
4|8|0

Best state to expand with depth (g(n)) 6 and heuristic (h(n)) 8
1|3|6
4|5|7
0|8|2

Best state to expand with depth (g(n)) 2 and heuristic (h(n)) 12
1|6|7
5|8|3
4|2|0

Best state to expand with depth (g(n)) 7 and heuristic (h(n)) 9
5|1|3
0|7|6
4|8|2

Best state to expand with depth (g(n)) 4 and heuristic (h(n)) 12
1|6|7
4|0|3
8|5|2

Best state to expand with depth (g(n)) 4 and heuristic (h(n)) 12
0|1|6
5|3|7
4|8|2

Best state to expand with depth (g(n)) 7 and heuristic (h(n)) 9
1|3|6
4|5|7
8|0|2

Best state to expand with depth (g(n)) 3 and heuristic (h(n)) 13
1|6|7
5|8|0
4|2|3

Best state to expand with depth (g(n)) 8 and heuristic (h(n)) 8
5|1|3
7|0|6
4|8|2

Best state to expand with depth (g(n)) 3 and heuristic (h(n)) 13
1|6|7
5|3|2
4|0|8

Best state to expand with depth (g(n)) 8 and heuristic (h(n)) 8
1|3|6
4|5|7
8|2|0

Best state to expand with depth (g(n)) 8 and heuristic (h(n)) 8
1|7|3
4|0|6
8|5|2

Best state to expand with depth (g(n)) 6 and heuristic (h(n)) 10
1|7|3
5|0|2
4|6|8

Best state to expand with depth (g(n)) 9 and heuristic (h(n)) 7
1|3|6
4|5|0
8|2|7

Best state to expand with depth (g(n)) 9 and heuristic (h(n)) 7
1|0|3
4|7|6
8|5|2

Best state to expand with depth (g(n)) 4 and heuristic (h(n)) 12
1|6|0
5|8|7
4|2|3

Best state to expand with depth (g(n)) 3 and heuristic (h(n)) 13
5|1|7
0|6|3
4|8|2

Best state to expand with depth (g(n)) 7 and heuristic (h(n)) 9
1|7|3
0|5|2
4|6|8

Best state to expand with depth (g(n)) 5 and heuristic (h(n)) 11
1|0|6
5|8|7
4|2|3

Best state to expand with depth (g(n)) 6 and heuristic (h(n)) 10
1|7|3
5|8|6
0|4|2

Best state to expand with depth (g(n)) 6 and heuristic (h(n)) 10
1|7|3
5|6|2
0|4|8

Best state to expand with depth (g(n)) 7 and heuristic (h(n)) 9
1|3|6
5|7|2
4|0|8

Best state to expand with depth (g(n)) 5 and heuristic (h(n)) 11
1|0|7
4|6|3
8|5|2

Best state to expand with depth (g(n)) 8 and heuristic (h(n)) 8
1|7|3
4|5|2
0|6|8

Best state to expand with depth (g(n)) 8 and heuristic (h(n)) 8
5|1|3
4|7|6
0|8|2

Best state to expand with depth (g(n)) 3 and heuristic (h(n)) 13
6|0|7
1|5|3
4|8|2

Best state to expand with depth (g(n)) 6 and heuristic (h(n)) 10
0|3|6
1|5|7
4|8|2

Best state to expand with depth (g(n)) 9 and heuristic (h(n)) 7
1|7|3
4|5|0
8|2|6

Best state to expand with depth (g(n)) 6 and heuristic (h(n)) 10
1|7|0
4|6|3
8|5|2

Best state to expand with depth (g(n)) 4 and heuristic (h(n)) 12
6|7|0
1|5|3
4|8|2

Best state to expand with depth (g(n)) 2 and heuristic (h(n)) 14
1|6|7
5|8|3
0|4|2

Best state to expand with depth (g(n)) 7 and heuristic (h(n)) 9
1|7|3
4|6|0
8|5|2

Best state to expand with depth (g(n)) 5 and heuristic (h(n)) 11
6|7|3
1|5|0
4|8|2

Best state to expand with depth (g(n)) 7 and heuristic (h(n)) 9
1|3|6
5|7|0
4|8|2

Best state to expand with depth (g(n)) 8 and heuristic (h(n)) 8
1|7|3
4|6|2
8|5|0

Best state to expand with depth (g(n)) 6 and heuristic (h(n)) 10
6|7|3
1|5|2
4|8|0

Best state to expand with depth (g(n)) 7 and heuristic (h(n)) 9
1|7|3
5|8|0
4|2|6

Best state to expand with depth (g(n)) 8 and heuristic (h(n)) 8
1|3|6
5|0|2
4|7|8

Best state to expand with depth (g(n)) 5 and heuristic (h(n)) 11
1|6|7
4|5|0
8|2|3

Best state to expand with depth (g(n)) 4 and heuristic (h(n)) 12
5|1|7
4|6|3
0|8|2

Best state to expand with depth (g(n)) 9 and heuristic (h(n)) 7
1|3|6
5|2|0
4|7|8

Best state to expand with depth (g(n)) 9 and heuristic (h(n)) 7
1|3|6
0|5|2
4|7|8

Best state to expand with depth (g(n)) 6 and heuristic (h(n)) 10
1|6|0
4|5|7
8|2|3

Best state to expand with depth (g(n)) 10 and heuristic (h(n)) 6
1|3|6
4|5|2
0|7|8

Best state to expand with depth (g(n)) 7 and heuristic (h(n)) 9
1|0|6
4|5|7
8|2|3

Best state to expand with depth (g(n)) 11 and heuristic (h(n)) 5
1|3|6
4|5|2
7|0|8

Best state to expand with depth (g(n)) 10 and heuristic (h(n)) 6
1|3|0
5|2|6
4|7|8

Best state to expand with depth (g(n)) 12 and heuristic (h(n)) 4
1|3|6
4|5|2
7|8|0

Best state to expand with depth (g(n)) 11 and heuristic (h(n)) 5
1|0|3
5|2|6
4|7|8

Best state to expand with depth (g(n)) 8 and heuristic (h(n)) 8
7|5|3
1|0|6
4|8|2

Best state to expand with depth (g(n)) 12 and heuristic (h(n)) 4
1|2|3
5|0|6
4|7|8

Best state to expand with depth (g(n)) 5 and heuristic (h(n)) 11
1|3|6
5|8|7
4|0|2

Best state to expand with depth (g(n)) 13 and heuristic (h(n)) 3
1|2|3
0|5|6
4|7|8

Best state to expand with depth (g(n)) 6 and heuristic (h(n)) 10
1|3|6
5|8|7
4|2|0

Best state to expand with depth (g(n)) 14 and heuristic (h(n)) 2
1|2|3
4|5|6
0|7|8

Best state to expand with depth (g(n)) 7 and heuristic (h(n)) 9
1|3|6
5|8|0
4|2|7

Best state to expand with depth (g(n)) 15 and heuristic (h(n)) 1
1|2|3
4|5|6
7|0|8

Best state to expand with depth (g(n)) 8 and heuristic (h(n)) 8
1|3|0
5|8|6
4|2|7

Best state to expand with depth (g(n)) 16 and heuristic (h(n)) 0
1|2|3
4|5|6
7|8|0

Solution found
Path
1|6|7
5|0|3
4|8|2

1|6|7
5|3|0
4|8|2

1|6|0
5|3|7
4|8|2

1|0|6
5|3|7
4|8|2

1|3|6
5|0|7
4|8|2

1|3|6
5|7|0
4|8|2

1|3|6
5|7|2
4|8|0

1|3|6
5|7|2
4|0|8

1|3|6
5|0|2
4|7|8

1|3|6
5|2|0
4|7|8

1|3|0
5|2|6
4|7|8

1|0|3
5|2|6
4|7|8

1|2|3
5|0|6
4|7|8

1|2|3
0|5|6
4|7|8

1|2|3
4|5|6
0|7|8

1|2|3
4|5|6
7|0|8

1|2|3
4|5|6
7|8|0

Depth of solution 16
Number of visited nodes 90
Max frontier size 59
Search took 5 milliseconds