The Eight Puzzle

Jasmine Kim

February 16, 2018

1 Introduction

This project was created for Dr. Eamonn Keogh's CS 205 Artificial Intelligence at the University of California, Riverside. For this project, I wrote a program that solves the 8-puzzle using:

- Uniform Cost Search
- A* with the Misplaced Tile heuristic
- A* with the Manhattan Distance heuristic

All three searching algorithms follow the general search algorithm. There is a simple text line interface that allows the user to use a default puzzle or input a puzzle of his or her own choosing. It also allows the user to select the searching algorithm. I chose to use Python3 as my language of choice to implement this project. In addition, this code can be easily changed to support puzzles of other sizes.

2 Algorithms

In this section, I will go further into explaining the differences of the three search algorithms. All three of the following algorithms use the A* algorithm, which sums the cost to get to the node, g(n), and the estimated distance, h(n), in order to decide what the expand next.

It's also worth noting, I kept track of repeated states and did not expand nodes that were seen previously.

2.1 Uniform Cost Search

Uniform cost search (UFS) is a form of blind uninformed search. It is uninformed because it assumes h(n), the estimated distance, is 0. Therefore, this algorithm enqueues nodes based only on g(n), the cost to get to a node. In this case, because each move is the same cost, g(n) at each node will be equal to the depth of the tree.

As for the implementation, uniform cost search is the same as breadth first search in this problem. This is because the cost to get to each node in the tree is the same. UFS will enqueue nodes into a FIFO queue and pops the first item to test against the goal puzzle.

2.2 A* with the Misplaced Tile Heuristic

A* with the misplaced tile heuristic is a form of informed search. It utilizes a misplaced tile heuristic which calculates h(n) as the number of tiles that do not match the goal state. Table 1 shows an example. The circled values are the misplaced tiles and therefore h(n) is 2. Note, you do not count the blank space as a misplaced tile.

As mentioned previously in UFS, because each move is the same cost, g(n) at each node will be equal to the depth of the tree. Once h(n) and g(n) is calculated, it will be placed into a queue and popped in the order of priority, where the smallest f(n) gets the highest priority. For the implementation, I utilized a heap queue for this.

Table 1: Calculating h(n) = 2 for the misplaced tile heuristic

Puzzle			Goal		
1	2	3	1	2	3
4	0	⑤	4	5	6
7	8	⑥	7	8	0

2.3 A* with the Manhattan Distance Heuristic

A* with the Manhattan Distance heuristic is another form of informed search. It calculates h(n) as the amount of moves required for the puzzle to get to the goal state. Table 2 shows an example. The circled values are misplaced tiles and by looking at the goal state, here are the following moves:

- 8 needs to move down 1
- 6 needs to move up 1 and right 1
- 5 needs to move up 1 and left 1

This sums up to h(n) = 5.

Table 2: Calculating h(n) for the Manhattan Distance heuristic

Puzzle			Goal		
1	2	3	1	2	3
4	⑧	0	4	5	6
7	⑥	⑤	7	8	0

As mentioned in the two previous search trees, g(n) at each node is equal to the depth of the tree. And when h(n) and g(n) is calculated, it will placed into a queue and serviced in the order of priority.

3 Results

3.1 Testing Puzzles

The goal state of the puzzle is when the eight puzzle is solved, as shown in Table 3. In order to compare the time and space complexity of the algorithms, I used five puzzles varying in difficulty from trivial to painful. Trivial is when the initial state of the puzzle is the goal state. Painful is when the depth of the solution is very deep. Below in Table 4, I have shown the different puzzles with varying levels of difficulty that was used during the analysis of the project.

Table 3: The goal state of the eight puzzle

Goal		
1	2	3
4	5	6
7	8	0

The difficulty of the solutions depend highly on the depth of the goal solutions, as shown in Table 5.

3.2 Analysis

Two metrics, time and space complexity, were used in order to compare the algorithms against the test puzzles of different difficulties. I also measured the raw time to run the algorithms.

Table 4: Different difficulties of puzzles used to measure time and space complexity

Trivial			Easy			Medium			Hard			Painful		
1	2	3	1	2	3	5	1	2	4	3	6	8	7	6
4	5	6	4	0	5	6	3	0	8	7	1	5	4	3
7	8	0	7	8	6	4	7	8	0	5	2	0	2	1

Table 5: Depth of goal solutions for puzzle difficulty

Depth of Solutions	
Trivial	0
Easy	2
Medium	11
Hard	18
Painful	23

3.2.1 Time Complexity

In order to measure time complexity, I counted the number of nodes expanded for each puzzle and algorithm combination. Again, mentioned previously, I kept track of repeated states and did not expand notes that were seen previously. This must be taken into consideration when looking at the results.

As you can see in Table 6 and in Figure 1, from the easy puzzles, both misplaced tile and Manhattan Distance heuristics expand significantly less nodes than uniform cost search. And as the depth of the solution gets deeper, it is around medium difficulty when the misplaced tile heuristic requires expanding more nodes than the Manhattan Distance heuristic. However, as shown in Figure 1, as the solution gets much more difficult, such as painful which has a solution depth of 23, the heuristics head towards re-converging.

Table 6: Time Complexity - Number of nodes expanded for puzzle difficulty

Number of Nodes Expanded			
	UCS	Misplaced	Manhattan
Trivial	1	1	1
Easy	18	7	7
Medium	1800	60	27
Hard	42114	2089	295
Painful	261967	5980	1587

3.2.2 Space Complexity

In order to measure space complexity, I counted the maximum size of the queue. As you can see in Table 7 and Figure 2, all three search algorithms perform similarly until the medium puzzle difficulty. Around the medium puzzle difficulty, the Manhattan Distance heuristic requires significantly less space than the other algorithms. Just as with time complexity, as shown in Figure 2, as the solution gets much more difficult the heuristics head towards re-converging.

3.2.3 Time Measured

I also measured the time lapsed in running the three search algorithms as shown in Table 8 and Figure 3. From the user's point of view, the difference in uniform cost search versus the misplaced tile and Manhattan Distance became very clear around the medium difficulty. As for the misplaced tile and Manhattan Distance heuristics, it was around the hard difficulty when there was a visible difference between the two algorithms.

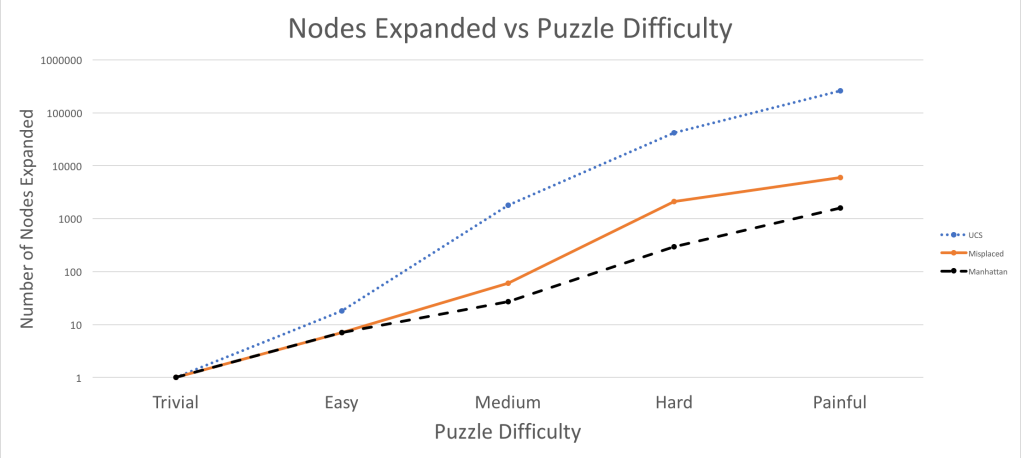

Figure 1: Time complexity - Number of nodes expanded per puzzle difficulty

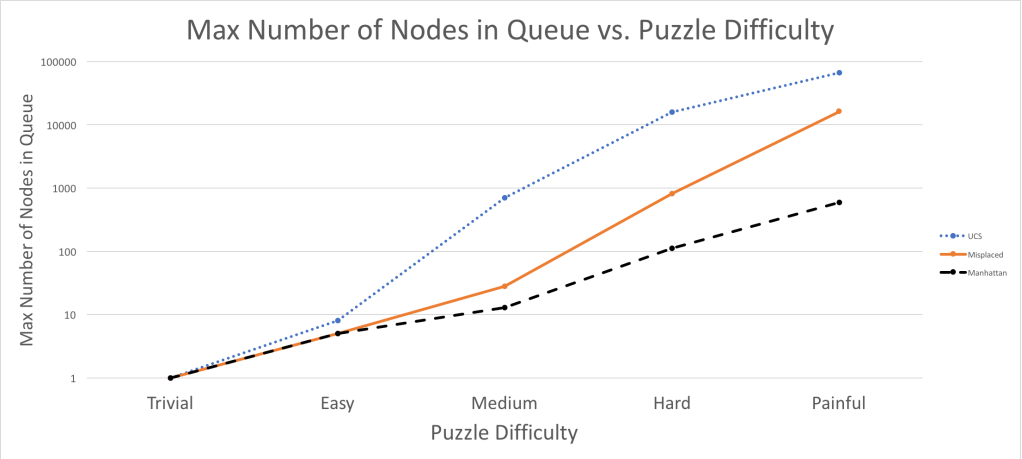

Figure 2: Space complexity - Maximum number of nodes on the queue per puzzle difficulty

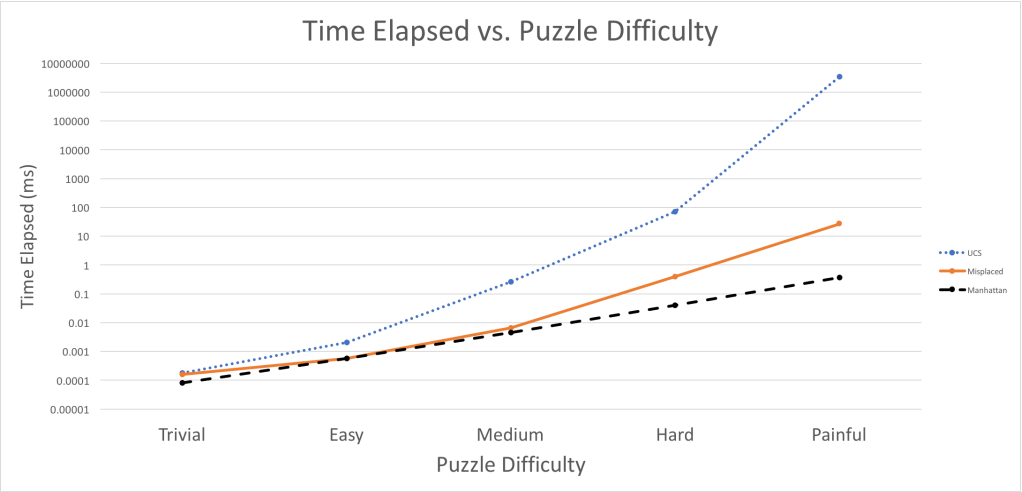

Figure 3: Time measurement per puzzle difficulty

Table 7: Space Complexity - Maximum number of nodes in the queue for puzzle difficulty

Maximum Number of Nodes in Queue			
	UCS	Misplaced	Manhattan
Trivial	1	1	1
Easy	8	5	5
Medium	704	28	13
Hard	15704	811	112
Painful	66349	16193	588

Table 8: Actual Time Lapsed

Time measured (milliseconds)			
	UCS	Misplaced	Manhattan
Trivial	.18	.16	.08
Easy	2.02	.57	.57
Medium	260	6.68	4.46
Hard	71268.00	400.19	39.96
Painful	3438915	26884	361

4 Conclusion

It's clear that in both time and space complexity, the search algorithms perform in the following order from best to worst:

1. Manhattan Distance Heuristic
2. Misplaced Tile Heuristic
3. Uniform Cost Search

As discovered empirically, for middle to hard puzzle difficulty, the memory cost and time cost for the uniform cost search is the cost of the Manhattan Distance heuristic squared. However, as the puzzle becomes much more difficult, all three algorithms begin to converge in both time and memory complexity.