Eight Puzzle Search Algorithms: A Comparison

Khushi Chaudhari
862335154, kchau047@ucr.edu
CS170: Introduction to Artificial Intelligence
Dr. Eamonn Keogh
February 8, 2025

Resources

- The Blind Search and Heuristic Search lecture slides and notes annotated from lecture
- Python 3 Documentation: https://docs.python.org/3/
- GeekForGeeks.com: This was used for syntax and functions for specific packages in python. The exact links are included in the code as comments.
- Definitions: https://www.almabetter.com/bytes/tutorials/artificial-intelligence/8-puzzle-problem-in-ai
- Any code snippets taken from other sources are linked directly in the code comments
- Help from TAs and Dr. Keogh

Github URL

https://github.com/khushi-04/Eight-Puzzle-Search

Outline

Introduction

The Eight-Puzzle is a game where you have a set of tiles (in this case, a 3x3 grid) numbered from 1-8 with a blank tile. The goal of this game is to move the blank tile (up, down, left, right) until the tiles are arranged in order, as seen in *Figure 1*. This puzzle also has many versions to it, such as the 15-puzzle (4x4 board).

For this project, I implement three search algorithms to solve the eight puzzle: Uniform Cost Search, A* with Misplaced Tile Heuristic, and A* with Manhattan Distance Heuristic. Each of these search algorithms have similar implementations, other than the heuristic used. According to AlmaBetter, the search space for our problem is “all possible states that the puzzle can transition through, from the initial state to the goal state” [1].

In this report, I will first discuss each of these search algorithms. Then, I will go into details about my findings regarding the time and space complexity of each algorithm, explaining which algorithm performs better as the puzzle increases in complexity/depth. For the implementation, I chose to use Python 3 and a min heap queue as the primary data structure for the search algorithm. It also uses a hash function to store repeated states. I included my github link above, which contains all of the implementation and sources used in the code.

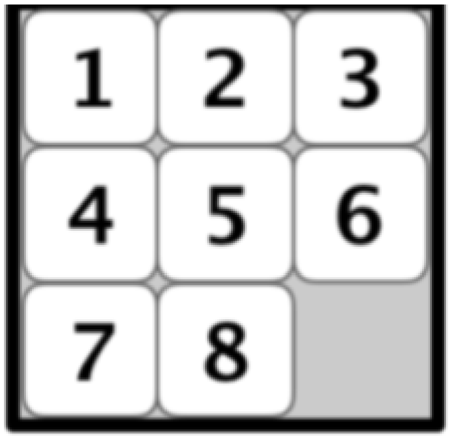


Figure 1: A solved eight puzzle

Description of Algorithms

Uniform Cost Search

Uniform cost search is the first algorithm that I implement in this project. It is a general greedy search algorithm that chooses its path based on the lowest cost. For the 8 puzzle, we assume that each movement of the tile has the same cost to move, or as Dr. Keogh explained, each tile movement requires the same “finger effort” to move. So, essentially, uniform cost search becomes a breadth-first search algorithm. To further elaborate, the uniform cost search will expand the cheapest puzzle state, where the cost is defined as f(n) = g(n) + h(n). g(n) holds the level of the node and h(n) is always 0.

A* with Misplaced Tile Heuristic

The second algorithm I implement is the A* search with the misplaced tile heuristic. The misplaced tile heuristic is where we calculate the number of tiles that are not in the goal spots for each puzzle state. For example, looking at *Figure 2*, we can see that the h(n) would be 4 because 4, 5, 7, and 8 are not in their goal states. The algorithm then adds the level of the state to this heuristic to decide which the cheapest state is to expand. Let’s assume that *Figure 2* was the result of moving one tile in the original problem. So, the assigned cost of this puzzle would be 5. The

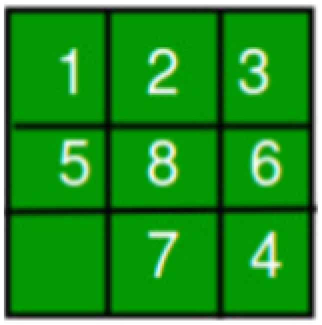


Figure 2: Unsolved Eight Puzzle

algorithm calculates each new state's heuristic and chooses to expand the state that has the lowest cost.

A* with Manhattan Distance Heuristic

The last algorithm I implement is very similar to the Misplaced Tile Heuristic, the Manhattan Distance Heuristic. These two are similar because their implementation and logic is nearly the same, except for the calculation of the h(n) itself. For Manhattan Distance, we calculate how far each misplaced tile is from its goal location using the distance formula in *Figure 3*. All other aspects of the algorithm are the same as the previous algorithm. Looking at *Figure 2*, the h(n) based on the distance formula would be 6. Similar to the Misplaced Tile Heuristic, the algorithm calculates each new state's heuristic and chooses to expand the state with the lowest cost.

$$|x_i - x_G| + |y_i - y_G|$$

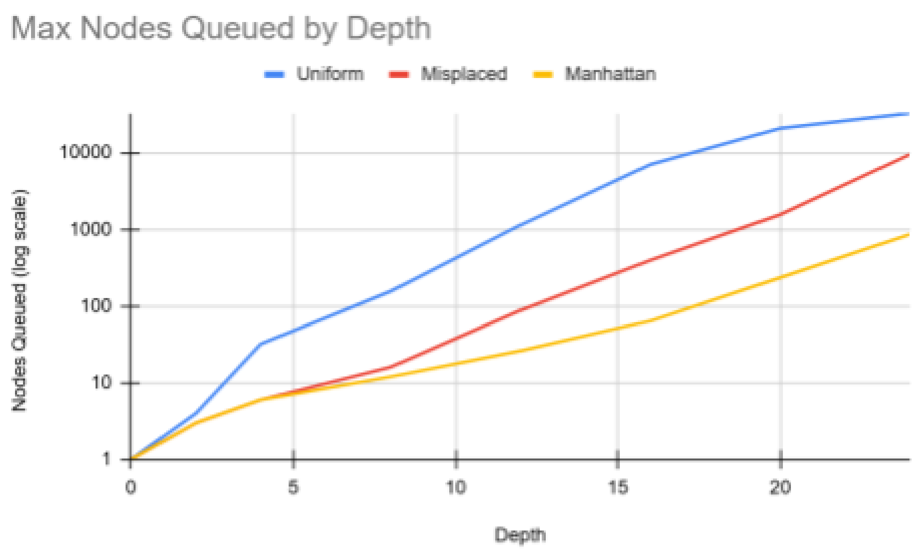
Figure 3: Manhattan Distance Formula

Comparison of Algorithms (Time and Space Complexity)

Using the given test puzzles in the project description, I tested each algorithm from depth 0 to depth 24. There was a negligible difference in the lower depths, but as the depth increased, the Uniform Search algorithm became exponentially worse in terms of time and space complexity.

Space Complexity

To measure the space complexity of my algorithms, I tracked the max nodes queued. Essentially, this represents the maximum number of nodes that were queued at any time while the search algorithm ran in my minimum heap queue. Dr. Keogh's slides on Heuristic Search explain that the average space complexity for Manhattan is better than Misplaced Tile, which is better than Uniform Cost Search. Based on *Figure 4,* the max queue length was very similar for all three algorithms at lower depths. But, using a logarithmic scale, it is very easy to see how the Uniform Cost Search algorithm has exponentially more nodes at its maximum with higher depths, starting around depth 5. It requires exponentially more memory because it expands each node in the level and stores it before expanding its children, similar to breadth first search. Also, based on the same figure, the Misplaced Tile Heuristic requires exponentially more memory with higher depths compared to the Manhattan Distance Heuristic starting around depth 10.



Time Complexity

To compare the time complexity of all three algorithms, I measured the numbers of nodes expanded and the time taken for the algorithm to run. The number of nodes expanded refers to all the nodes that the algorithm chose to expand based on selecting the smallest heuristic, as explained for each individual algorithm. The time taken refers to how long, in seconds, the algorithm ran from the start of the search to when it found the goal state.

Dr. Keogh's slides explain that Manhattan Distance and Misplaced Tile Heuristics (using A*) are much faster than the Uniform Cost Search in terms of time complexity. He also mentions that Manhattan Distance Heuristic with A* is better than Misplaced Tile Heuristic with A* because it is able to estimate the true distance closer than Misplaced Tile without overestimating. Looking at *Figure 5,* we can see that even at depths as low as 4, the Uniform Cost Search expands far more nodes compared to the other two algorithms. This is likely because the search space becomes exponentially bigger and the Uniform Cost Search always expands every unexplored node like a breadth first search, compared to the other two searches. Also, the Misplaced Tile Heuristic expands more nodes for depths more than 8 compared to the Manhattan Distance Heuristic, but less than the Uniform Cost Search.

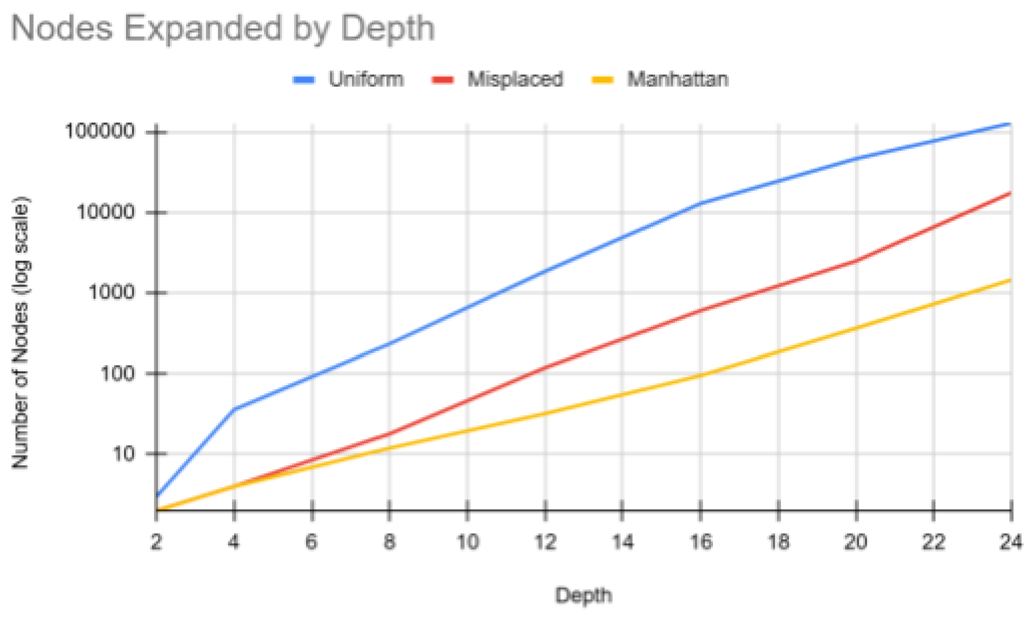


Figure 5: This graph shows the total number of nodes expanded for each depth by Uniform Cost Search, Misplaced Tile Heuristic, and Manhattan Distance Heuristic.

Figure 6 gives another visual representation of how the algorithms compare in terms of time complexity. Since the Uniform Cost Search has to expand every unexplored puzzle state, it requires far more time to find a solution, especially at higher depths. Similar to the number of nodes expanded, the Uniform Cost Search takes exponentially more time at depths higher than 4 compared to the other two algorithms. Misplaced Tile Heuristic also takes exponentially more time at depths higher than 8 compared to Misplaced Tile Heuristic. This does follow the observations from the previous graph and shows that Manhattan Heuristic is faster than Misplaced Heuristic, which is faster than Uniform Search.

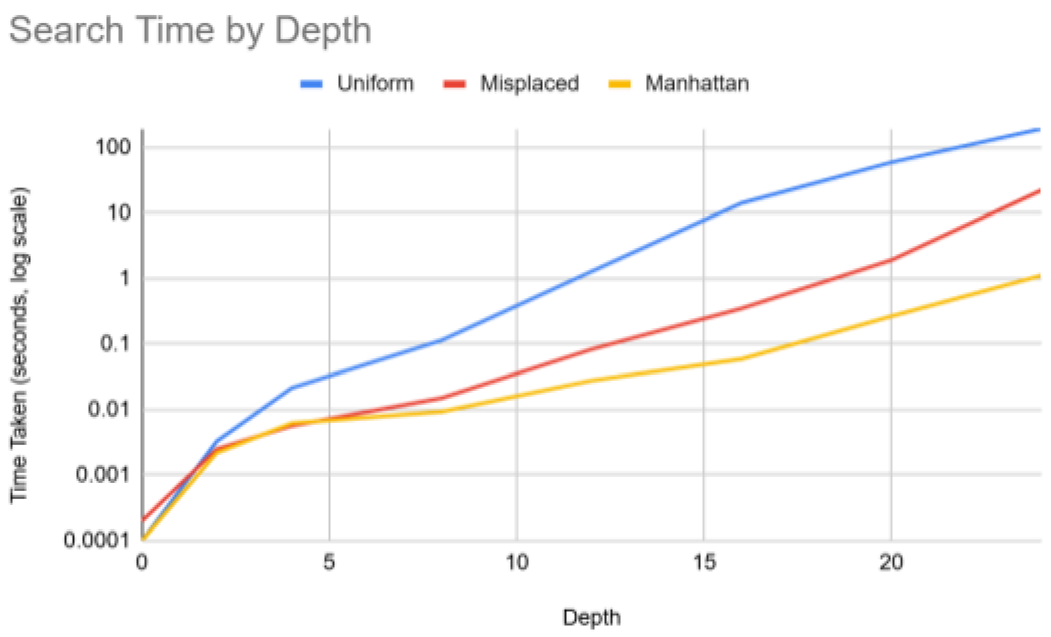


Figure 6: This graph shows time taken to solve an 8-puzzle for each depth by Uniform Cost Heuristic, Misplaced Tile Heuristic, and Manhattan Distance Heuristic.

Conclusion

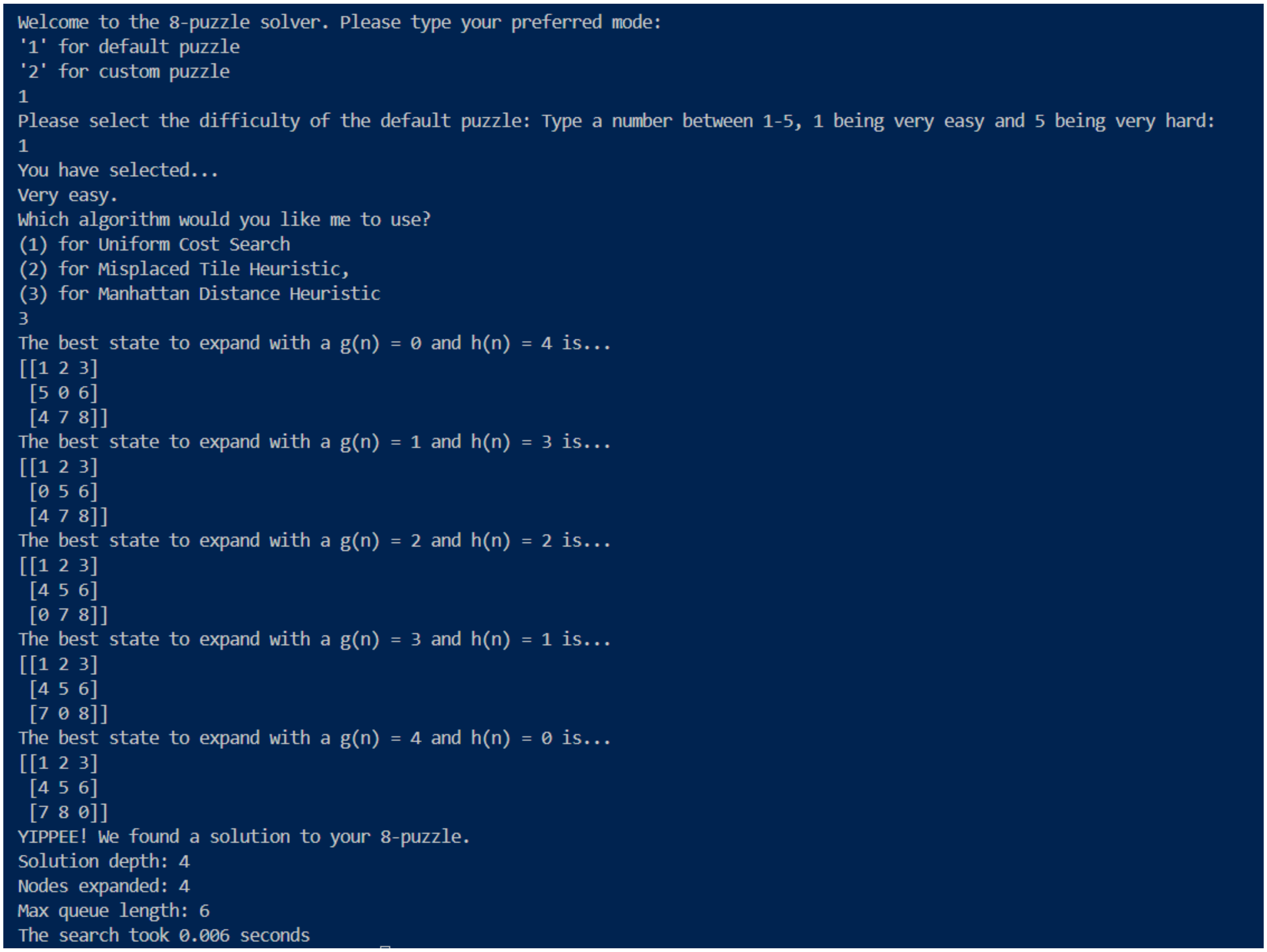
After looking at all the graphs and analyzing the differences and similarities between the three algorithms, we can conclude that:

All three algorithms perform very similarly at lower depths, especially at depth 4 and below, but deviate for both time and space complexity at higher depths. Uniform Cost Search, also effectively Breadth First Search in our case, performs the worst in terms of time and space complexity. It becomes exponentially slower and requires an exponentially larger amount of memory as the depth increases compared to the other two algorithms, so it is the least efficient search algorithm.

Overall, the A* algorithm with heuristics is much more efficient than the Uniform Cost Search because it uses heuristics to have a closer approximation of the solution, choosing puzzle states that are closer to the solution. The Manhattan Distance Heuristic is more efficient than the Misplaced Tile Heuristic because it considers how far each misplaced tile is from its actual location. It is able to estimate the distance from the solution more accurately compared to the Misplaced Tile Heuristic.

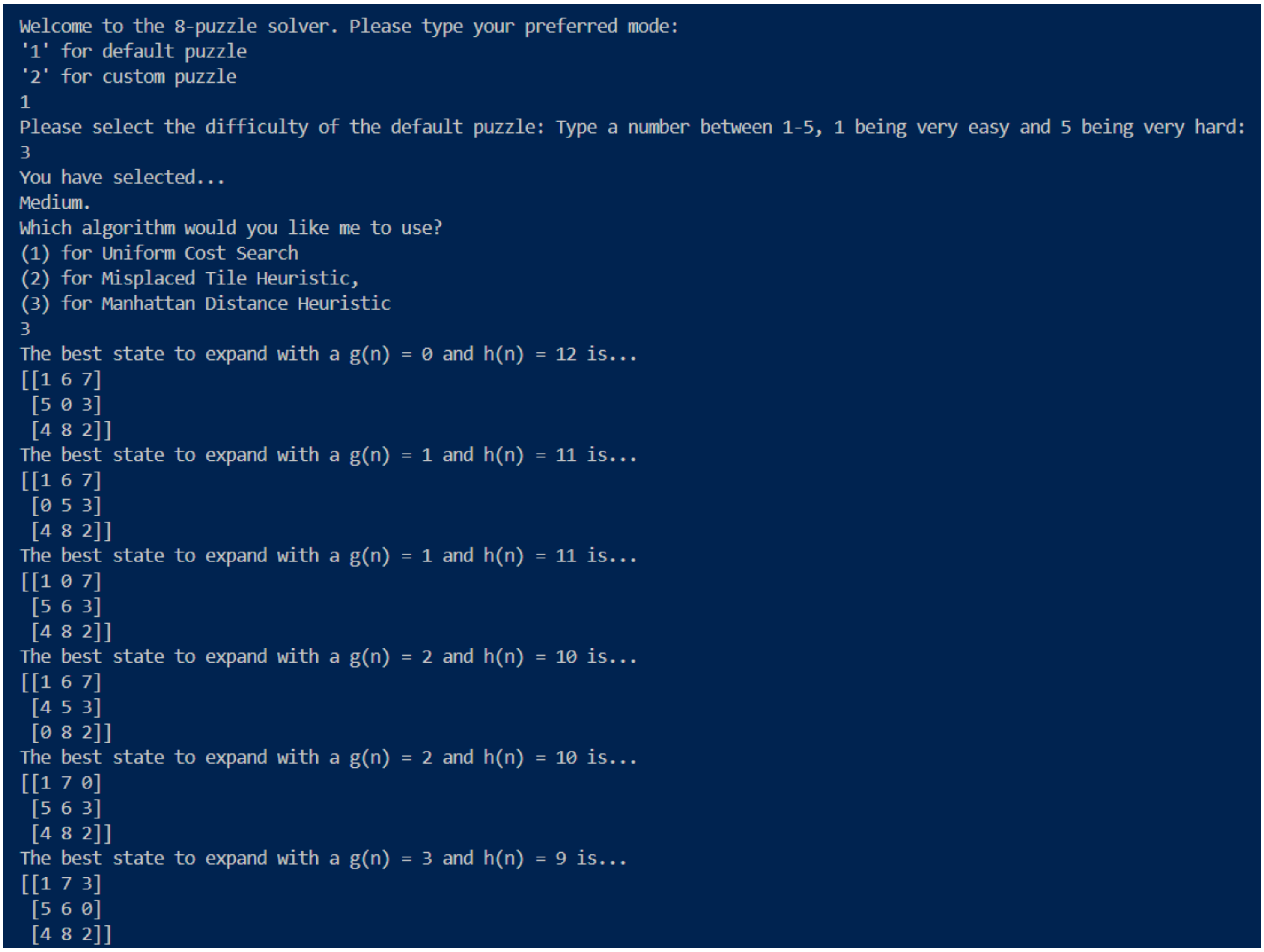
In conclusion, the A* with Manhattan Distance Heuristic is more efficient than the A* with Misplaced Tile Heuristic and Uniform Cost Search, as seen in the comparisons using different puzzle depths in my graphs.

Trace Back for Depth 4 Eight Puzzle:



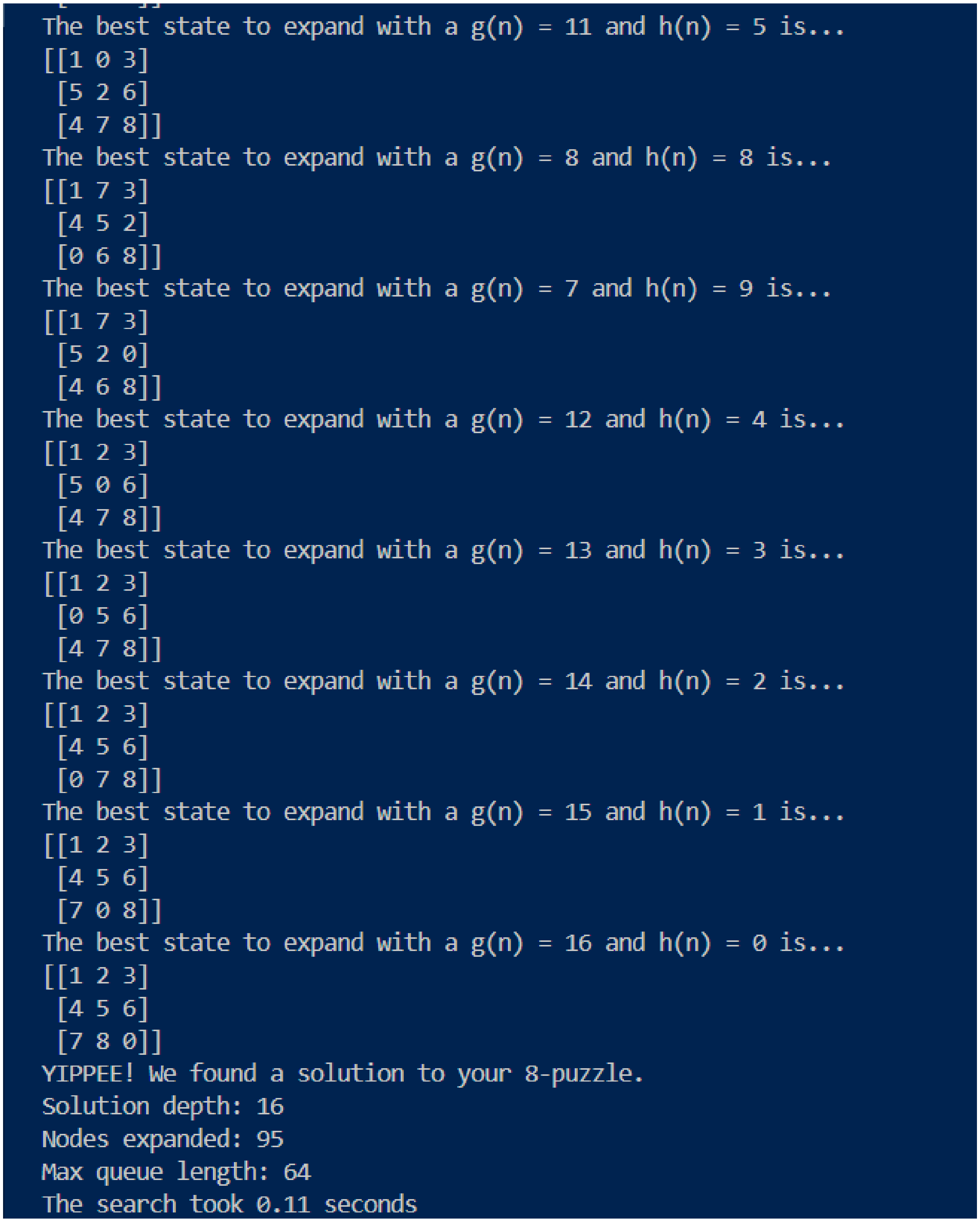
```
Welcome to the 8-puzzle solver. Please type your preferred mode:
'1' for default puzzle
'2' for custom puzzle
1
Please select the difficulty of the default puzzle: Type a number between 1-5, 1 being very easy and 5 being very hard:
1
You have selected...
Very easy.
Which algorithm would you like me to use?
(1) for Uniform Cost Search
(2) for Misplaced Tile Heuristic,
(3) for Manhattan Distance Heuristic
3
The best state to expand with a g(n) = 0 and h(n) = 4 is...
[[1 2 3]
 [5 0 6]
 [4 7 8]]
The best state to expand with a g(n) = 1 and h(n) = 3 is...
[[1 2 3]
 [0 5 6]
 [4 7 8]]
The best state to expand with a g(n) = 2 and h(n) = 2 is...
[[1 2 3]
 [4 5 6]
 [0 7 8]]
The best state to expand with a g(n) = 3 and h(n) = 1 is...
[[1 2 3]
 [4 5 6]
 [7 0 8]]
The best state to expand with a g(n) = 4 and h(n) = 0 is...
[[1 2 3]
 [4 5 6]
 [7 8 0]]
YIPPEE! We found a solution to your 8-puzzle.
Solution depth: 4
Nodes expanded: 4
Max queue length: 6
The search took 0.006 seconds
```

Traceback for a Depth 16 Eight Puzzle:



```
Welcome to the 8-puzzle solver. Please type your preferred mode:
'1' for default puzzle
'2' for custom puzzle
1
Please select the difficulty of the default puzzle: Type a number between 1-5, 1 being very easy and 5 being very hard:
3
You have selected...
Medium.
Which algorithm would you like me to use?
(1) for Uniform Cost Search
(2) for Misplaced Tile Heuristic,
(3) for Manhattan Distance Heuristic
3
The best state to expand with a g(n) = 0 and h(n) = 12 is...
[[1 6 7]
 [5 0 3]
 [4 8 2]]
The best state to expand with a g(n) = 1 and h(n) = 11 is...
[[1 6 7]
 [0 5 3]
 [4 8 2]]
The best state to expand with a g(n) = 1 and h(n) = 11 is...
[[1 0 7]
 [5 6 3]
 [4 8 2]]
The best state to expand with a g(n) = 2 and h(n) = 10 is...
[[1 6 7]
 [4 5 3]
 [0 8 2]]
The best state to expand with a g(n) = 2 and h(n) = 10 is...
[[1 7 0]
 [5 6 3]
 [4 8 2]]
The best state to expand with a g(n) = 3 and h(n) = 9 is...
[[1 7 3]
 [5 6 0]
 [4 8 2]]
```

Skipping the in between steps to save space…



```
The best state to expand with a g(n) = 11 and h(n) = 5 is...
[[1 0 3]
 [5 2 6]
 [4 7 8]]
The best state to expand with a g(n) = 8 and h(n) = 8 is...
[[1 7 3]
 [4 5 2]
 [0 6 8]]
The best state to expand with a g(n) = 7 and h(n) = 9 is...
[[1 7 3]
 [5 2 0]
 [4 6 8]]
The best state to expand with a g(n) = 12 and h(n) = 4 is...
[[1 2 3]
 [5 0 6]
 [4 7 8]]
The best state to expand with a g(n) = 13 and h(n) = 3 is...
[[1 2 3]
 [0 5 6]
 [4 7 8]]
The best state to expand with a g(n) = 14 and h(n) = 2 is...
[[1 2 3]
 [4 5 6]
 [0 7 8]]
The best state to expand with a g(n) = 15 and h(n) = 1 is...
[[1 2 3]
 [4 5 6]
 [7 0 8]]
The best state to expand with a g(n) = 16 and h(n) = 0 is...
[[1 2 3]
 [4 5 6]
 [7 8 0]]
YIPPEE! We found a solution to your 8-puzzle.
Solution depth: 16
Nodes expanded: 95
Max queue length: 64
The search took 0.11 seconds
```