CS170
Serena Lew, 862068523
Karina Lee, 862104904

CS170 Project 1 Report

Introduction

In this project, we implemented three different search algorithms: Uniform Cost Search, A* Search with Misplaced Tile heuristic, and A* Search with Euclidean Distance heuristic. We chose to program in C++ and used GNU 7.6.1 to compile the code on Visual Studio Code. All major code is original. During the process, we referred to the following sources:

- Blind Search and Heuristic Search lecture slides
- Project documents with algorithm-specific details and example test cases
- GeeksforGeeks to efficiently implement a min priority queue: https://www.geeksforgeeks.org/implement-min-heap-using-stl/

Algorithm Comparison

In each of the following search algorithms, we used a priority queue for our frontier in order to expand the cheapest node, which has the lowest path cost. Each of the algorithms have different methods of calculating these costs.

Uniform Cost Search (UCS)
In UCS, the path cost is the depth, g(n). We can describe UCS as "A* search with h(n) hardcoded to equal zero," as it is stated in the project description document.

A* Search with Misplaced Tile heuristic
With the Misplaced Tile heuristic, we can add the cost of the depth, g(n), with the heuristic function, h(n), to get the total path cost. The heuristic function counts the number of tiles that are misplaced. This does not include the blank, which is 0 in our program. In the following example, h(n) = 5, because tiles 4, 1, 2, 5, and 8 are misplaced.

Initial state:
4 1 3
2 0 6
7 5 8

Goal state:
1 2 3
4 5 6
7 8 0

A* Search with Euclidean Distance heuristic

With the Misplaced Tile heuristic, we can add the cost of the depth, g(n), with the heuristic function, h(n) to get the total path cost. The heuristic function sums the direct distances of all the misplaced tiles to their goal states. The distances of tiles that are misplaced diagonally are calculated with the Pythagorean Theorem. The sum does not include the blank, which is 0 in our program. In the above example, h(n) using the Euclidean Distance heuristic is:

$$h(n) = 1 + 1 + \text{sqrt}(2) + 1 + 1 = 5.4142$$

Given test cases:

Trival

1 2 3
4 5 6
7 8 *

Easy

1 2 *
4 5 3
7 8 6

Oh Boy

8 7 1
6 * 2
5 4 3

Very Easy

1 2 3
4 5 6
7 * 8

doable

* 1 2
4 5 3
7 8 6

IMPOSSIBLE: The following puzzle is impossible to solve, if you *can* solve it, you have a bug in your code.

1 2 3
4 5 6
8 7 *

Number of nodes expanded:

	Uniform Cost Search	Misplaced Tile Heuristic	Euclidean Distance Heuristic
Trivial (0)	0	0	0
Very Easy (1)	2	1	1
Easy (2)	5	2	2
Doable(3)	15	4	4
Oh Boy (4)	-	6453	1021
Impossible (5)	-	-	-

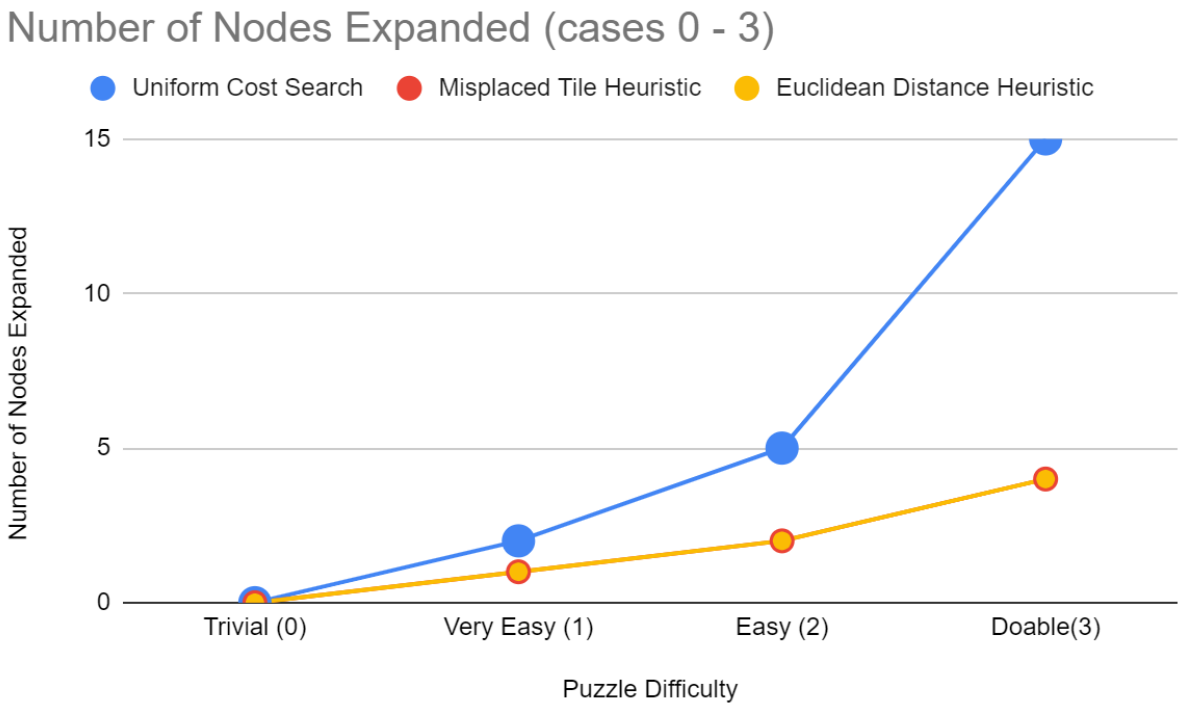

Figure 1: A comparison of the number of nodes expanded for each search algorithm for puzzles 0 to 3. We can see that Uniform Cost Search expands more nodes than both Misplaced Tile heuristic and Euclidean Distance heuristic starting from puzzle 1.

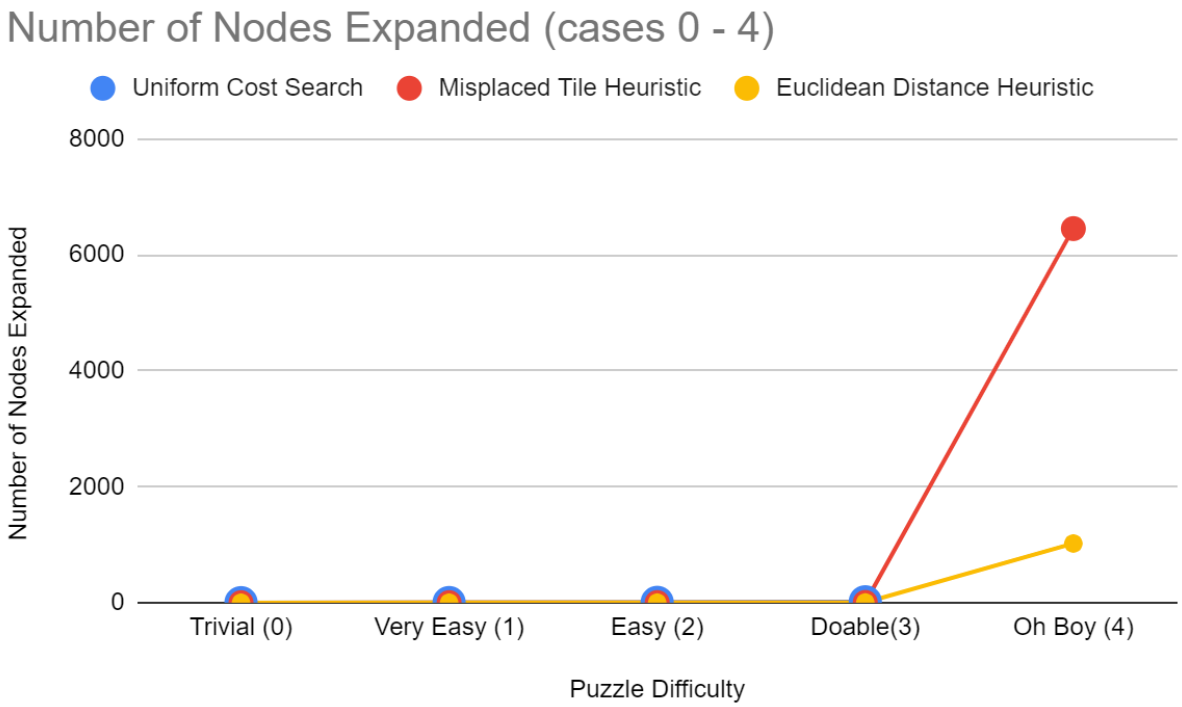

Figure 2: A comparison of the number of nodes expanded for each search algorithm for puzzles 0 to 4. The points on the graph for puzzles 0 to 3 are not distinguishable due to how small they are in comparison to the points for puzzle 4. We can see that at this point, the number of nodes expanded for the Misplaced Tile heuristic becomes much larger than the Euclidean Distance heuristic. Uniform Cost Search does not have a point for puzzle 4 since it takes an absurdly large amount of time to reach the goal state.

Max number of nodes in queue:

	Uniform Cost Search	Misplaced Tile Heuristic	Euclidean Distance Heuristic
Trivial (0)	0	0	0
Very Easy (1)	3	3	3
Easy (2)	6	3	3
Doable(3)	16	4	4
Oh Boy (4)	-	3711	572
Impossible (5)	-	-	-

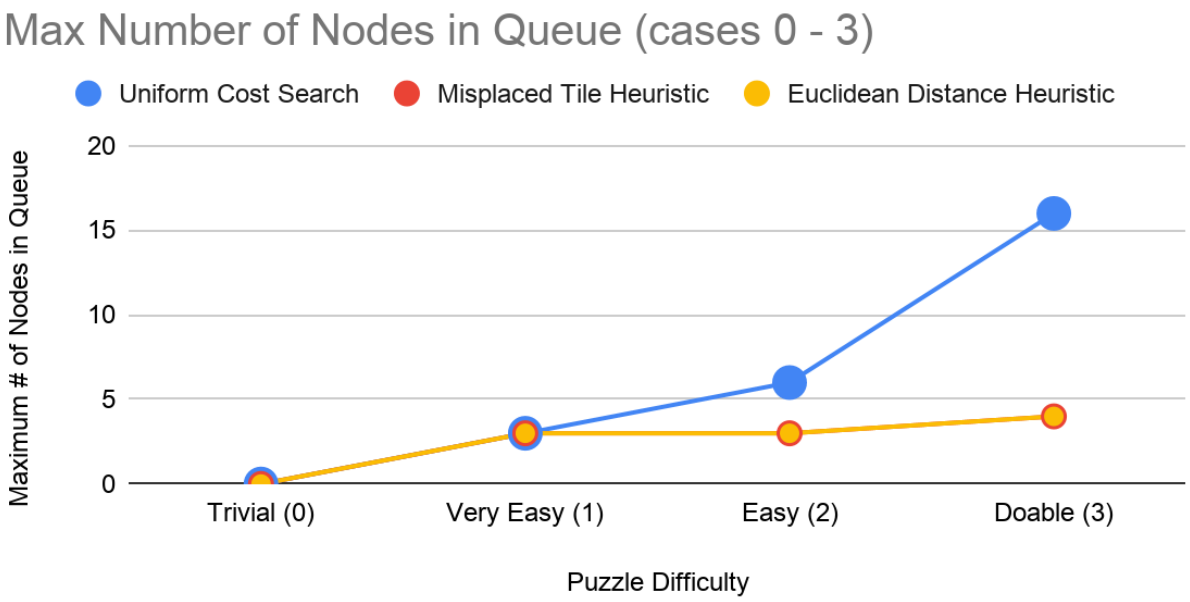

Figure 3: A comparison of the maximum number of nodes in the queue for each algorithm for puzzles 0 to 3. Note that Misplaced Tile Heuristic and Euclidean Distance Heuristic have the same values throughout the test cases while Uniform Cost Search begins to increase by puzzle 2. This means that there will be more nodes that Uniform Cost Search will explore than the latter two.

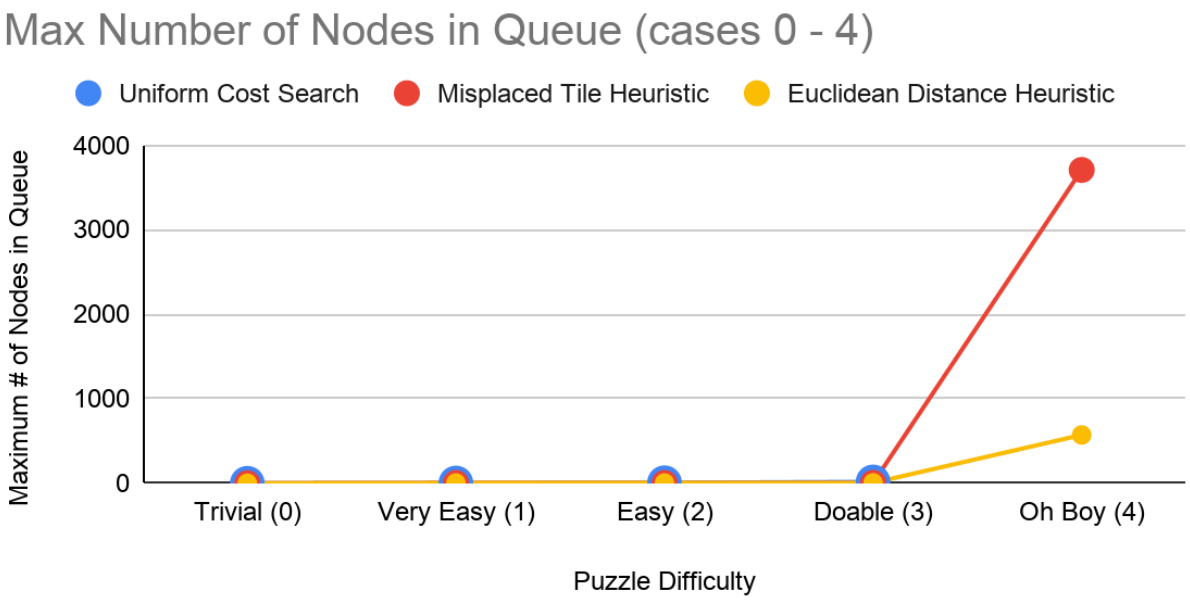

Figure 4: A comparison of the maximum number of nodes in the queue for each algorithm for puzzles 0 to 4. The points on the graph for puzzles 0 to 3 are not distinguishable due to how small they are in comparison to the points for puzzle 4. Once the algorithms are run on puzzle 4, we can clearly see that maximum queue size for Misplaced Tile Heuristic dramatically increases compared to Euclidean Distance Heuristic. Ultimately, Euclidean Distance Heuristic has smaller maximum queue sizes than Misplaced Tile Heuristic.

Note that it is not possible to solve the given impossible puzzle. The program expands an absurd amount of nodes and therefore takes way too much time and space to reach a solution. We were also unable to successfully get a result for running Uniform Cost Search on the 4th puzzle due to the same issue.

Results

Out of the three search algorithms that we have tested, we can clearly see that the Euclidean Distance Heuristic expands the least number of nodes and has the least maximum number of nodes in the queue. In other words, the Euclidean Distance Heuristic performs the best, with the Misplaced Tile Heuristic following somewhat closely. We can therefore conclude that Uniform Search Cost performs the worst out of the three.

We can notice that for simpler puzzles, the Misplaced Tile Heuristic and Euclidean Distance Heuristic have similar results. We can also note that the Uniform Cost Search algorithm has results that are slightly larger than the other two, but are still within a reasonable interval. However, as puzzle difficulty increases, each algorithm dramatically differs in output. For such difficult puzzles, we can see that heuristics are much more efficient than algorithms such as Uniform Cost Search.

Trace of Euclidean Distance A* (includes sequence of operators)

```
Welcome to Karina's [862104904] & Serena's [862068523] 8 puzzle solver!
Type 1 to use a default puzzle, or 2 to enter your own puzzle.
1
Type 1 for UCS, 2 for A* with Misplaced Tile heuristic, or 3 for A* with Euclidean distance heuristic: 3
Initial:
1 0 3
4 2 6
7 5 8
Expanding node...

The best node to expand with g = 1, h = 2 is...
1 2 3
4 0 6
7 5 8
Expanding node...

The best node to expand with g = 2, h = 1 is...
1 2 3
4 5 6
7 0 8
Expanding node...


Solution found! Depth: 3
Path to goal state:

g = 0, h = 3

1 0 3
4 2 6
7 5 8

______
g = 1, h = 2
down
1 2 3
4 0 6
7 5 8

______
g = 2, h = 1
down
1 2 3
4 5 6
7 0 8

______
g = 3, h = 0
right
1 2 3
4 5 6
7 8 0

______
To solve this problem the search algorithm expanded a total of 3 nodes.
The maximum number of nodes in the queue at any one time: 6.
```