Team Members:
Zachary Hill SID: 862130859
Khuaja Shams SID: 862041797

# Project #1 Report

## Introduction

This project is designed to find the shortest solution to solving an 8 piece slide puzzle so all the numbers are in order counting from top-left to bottom-right by counting right. Finding the path requires an AI that uses one of many search algorithms including: Uniform Cost Search (UCS), and Misplaced Tile and Euclidean Distance heuristics applied to A* to calculate all possible moves proceeding a given puzzle. The overall code was implemented using a Graph Search algorithm, where we stored any visited nodes within an explored set. The other three algorithms each uniquely calculate the depth of the Tree and the cost of each transition between two nodes, i.e. the cost of each edge. By providing additional header files, such as heuristic and tile movement, we were able to implement the code more efficiently but no special data structures were used. The code language used was C++, to implement the six default puzzles plus one more and six randomly generated puzzles, and all files work as one to create a code that is fully functional in calculating a multitude of 8 piece puzzles.

## Outcomes of Different Algorithms

Uniform Cost Search is simply A* an algorithm where h(n) is hardcoded to 0, and only the cost of g(n) will be calculated by expanding the cheapest node from initial state to the frontier. In the case of this assignment, there are no weights to the expansions, and each expanded node will have a cost of 1.

The Misplaced Tile Heuristic is one of two heuristic search algorithms specified as an A* search. This heuristic looks for any tiles out of place when compared to the goal state. Not counting the blank/missing tile, g(n) is incremented by 1 for each of the tiles not in their current goal state positions. This assigns a number, where lower is better, to node expansion based on how many misplaced tiles there are after a tile has been shifted. When applied to the n-puzzle, queue will expand the node with the cheapest cost, rather than expanding each of the child nodes as Uniform Cost Search would.

The Euclidean Distance Heuristic is similar to the Misplaced Tile Heuristic such that it considers the cost of future expansions and looks at misplaced tiles, but has a different rationale to it. It looks at all the tiles out of place and determines the shortest number of moves to arrive at their goal positions, using the pythagorean theorem and the number of squares moved horizontally and vertically the heuristic determines the total distance used for the edge cost. The resulting g(n) is the sum of all the cost of all the diagonal distances. Again all the distance for all misplaced tiles are calculated except for the blank tile.

Graph/Table Results

Nodes Expanded

In Figure 1, we can see the difference between the number of expanded nodes is much too small when solving the easier puzzles. When we reach the Hard puzzle a gap is present with UCS overtaking both A* heuristic searches and Misplaced is a bit higher than Euclidean. The following puzzle shows the gap widening but after that all three algorithms close in on each other and contain equal expanded nodes in the Impossible puzzle, as expected since this puzzle should fail. In Figure 3, any puzzles that require more moves have an increased number of nodes for all three algorithms while the Euclidean distance is barely visible in puzzles that are simpler.

Maximum Queue Size

In Figure 2, the flow of each algorithm is similar to those in Figure 1 but the nodes used are different for certain puzzles. The total number of nodes in each queue is fairly close to the expanded nodes in Figure 1 but those for harder puzzles are significantly smaller. In Figure 4, similar concept in Figure 3 but again there are fewer nodes in the queue than those expanded.

Additional Created Puzzles

Here is a list of six puzzles we created ourselves and tested the node expansion and queue size for:

1) [ {7}, {8}, {2},
{1}, {3}, {0},
{4}, {6}, {5}]

2) [ {5}, {7}, {3},
{4}, {1}, {6},
{2}, {0}, {8}]

3) [ {1}, {5}, {2},
{0}, {4}, {3},
{7}, {8}, {6}]

4) [ {3}, {1}, {6},
{5}, {0}, {7},
{4}, {2}, {8}]

5) [ {7}, {1}, {2},
{8}, {4}, {3},
{6}, {5}, {0}]

6) [ {7}, {4}, {1},
{8}, {5}, {2},
{0}, {6}, {3}]

Main Issues

Upon pushing new nodes that were developed after shifting tile up, down, left,or right to the frontier, the error was displayed that was eventually fixed when the push and pop functions in the Tree class were switched. Another problem was a system crash when the total number of expanded nodes reached twenty-thousand nodes in one of the more difficult puzzles. This was partially solved when we fixed the statements for Misplaced Tile Search that was causing a bug. Unfortunately, the node expansion for the two most difficult default puzzles continuously increases without stopping at a specific number of expanded nodes and does not show whether the process has succeeded or failed. So we limited the node expansion for any and all graphs created by solving a puzzle, which did solve the issue and provided a solution even though the results were different than expected.

Conclusion

The lowest total cost to the goal state of an nxn puzzle can be performed by various algorithms but some are more efficient than others. Based on the data found in each table/ graph, it is clear that of the three algorithms A* Euclidean Distance Heuristic performs the best, followed by A* Misplaced Tile Heuristic, and Uniform Cost Search (UCS) as the least efficient. Either of the two heuristics used can greatly improve the efficiency of algorithms as it compares the initial state with the goal state as the process continues on thus creating fewer nodes to expand and a smaller queue size. Although both heuristics improved the run time and space cost of Uniform Cost Search, not all heuristics provide the same results as they process the tree/graph differently. Keep in mind the difficulty of each puzzle and the number of moves it takes to solve the puzzle is determined by the arrangement of each tile but some puzzles may not be solvable at all. The difference between the three graphs is so small when solving easy puzzles but widens when solving more difficult ones. To be precise, the Euclidean Distance is approximately 148.5% more efficient than Misplaced Tiles and 173% more efficient than UCS whereas Misplaced Tile is 116% more efficient than UCS.

## Default Puzzles

**Number of Nodes Expanded**

| | UCS | A* MisplacedTiles | A* Euclid Distance |
|---|---|---|---|
| Trivial (1) | 0 | 0 | 0 |
| Very Easy (2) | 12 | 4 | 4 |
| Easy (3) | 28 | 8 | 8 |
| Doable (4) | 124 | 32 | 28 |
| Hard (5) | 14568 | 3528 | 2328 |
| Very Hard (6) | 300000 | 225232 | 53628 |
| Impossible (7) | 300000 | 300000 | 300000 |

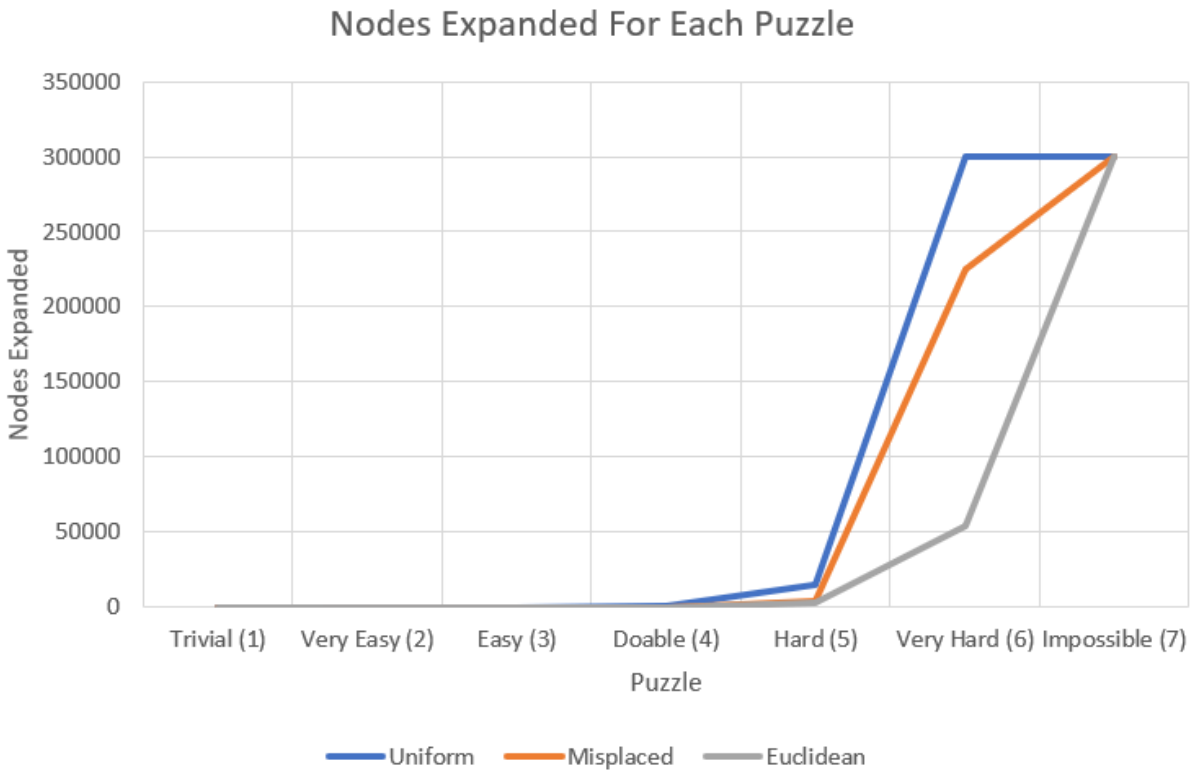


Figure 1. (Y-axis is “Total Number of Nodes Expanded”; X-axis is “Puzzle (Difficulty)”)

**Maximum Queue Size**

| | UCS | A* MisplacedTiles | A* Euclid Distance |
|---|---|---|---|
| Trivial (1) | 1 | 1 | 1 |
| Very Easy (2) | 7 | 3 | 3 |
| Easy (3) | 13 | 4 | 4 |
| Doable (4) | 27 | 11 | 11 |
| Hard (5) | 2185 | 535 | 393 |
| Very Hard (6) | 33540 | 27294 | 7670 |
| Impossible (7) | 33540 | 33540 | 33540 |

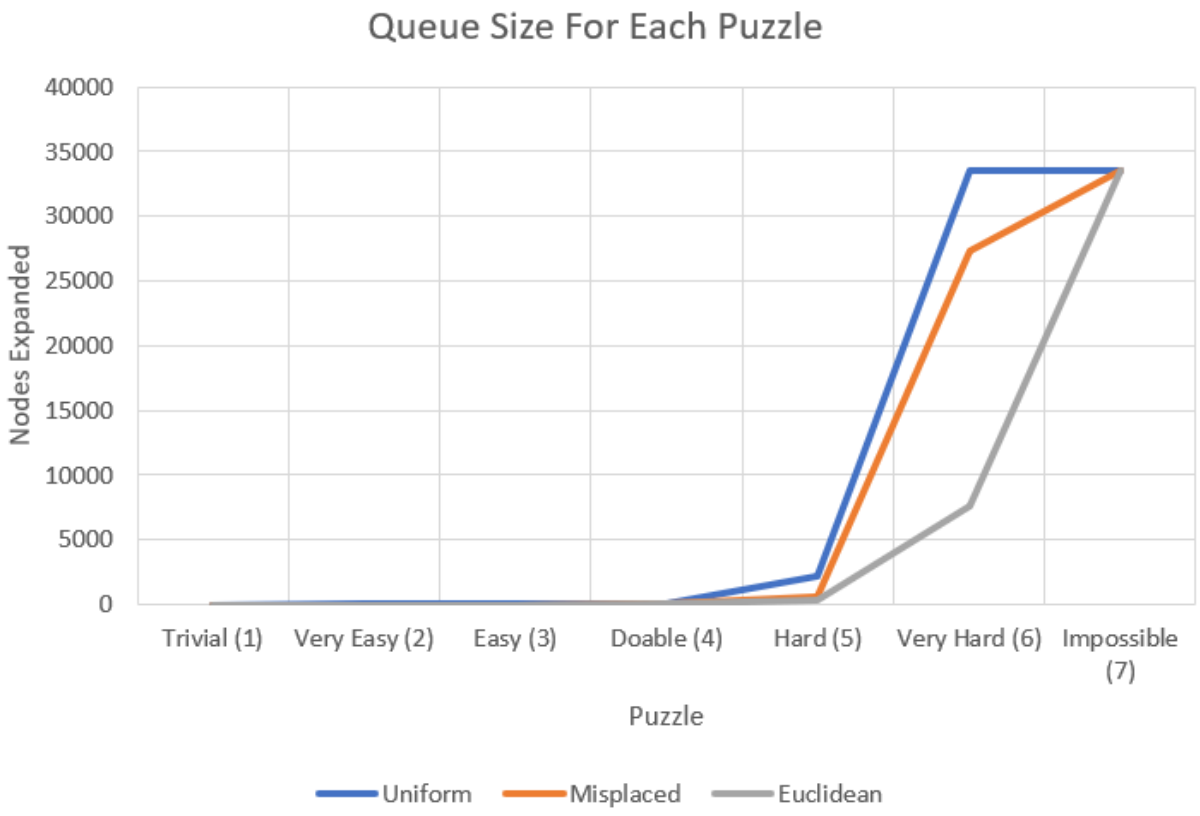


Figure 2. (Y-axis is “Maximum Queue Size”; X-axis is “Random Puzzles”)

## Created Puzzles

**Number of Nodes Expanded**

| | UCS | A* MisplacedTiles | A* Euclid Distance |
|---|---|---|---|
| Puzzle 1 | 111932 | 19716 | 9232 |
| Puzzle 2 | 121140 | 21740 | 11580 |
| Puzzle 3 | 316 | 84 | 84 |
| Puzzle 4 | 477692 | 143160 | 68240 |
| Puzzle 5 | 27036 | 4556 | 1904 |
| Puzzle 6 | 27760 | 4148 | 1800 |

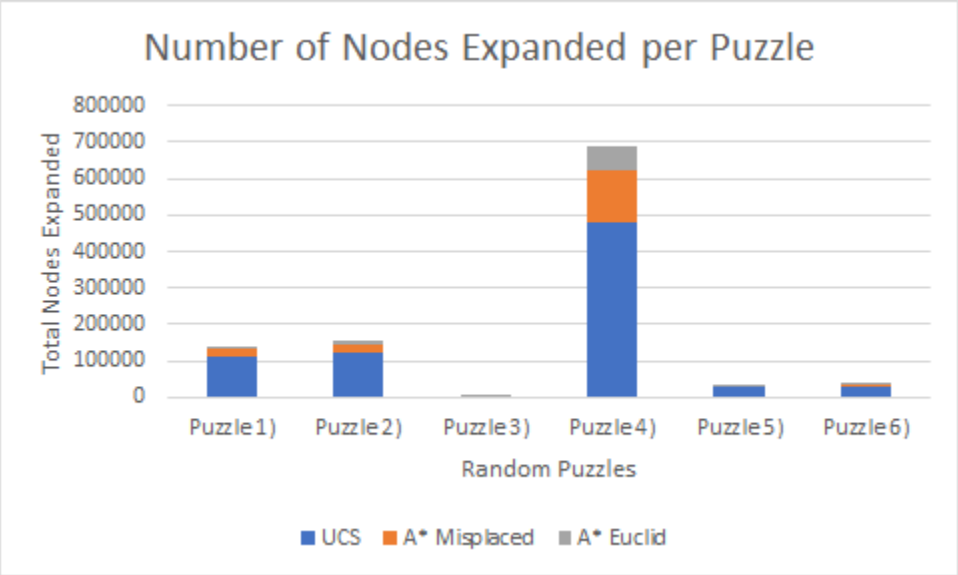


Figure 3.

**Maximum Queue Size**

| | UCS | A* MisplacedTiles | A* Euclid Distance |
|---|---|---|---|
| Puzzle 1 | 15080 | 3035 | 1386 |
| Puzzle 2 | 16090 | 3304 | 1756 |
| Puzzle 3 | 57 | 19 | 19 |
| Puzzle 4 | 33258 | 18294 | 9516 |
| Puzzle 5 | 4072 | 712 | 335 |
| Puzzle 6 | 4108 | 669 | 313 |

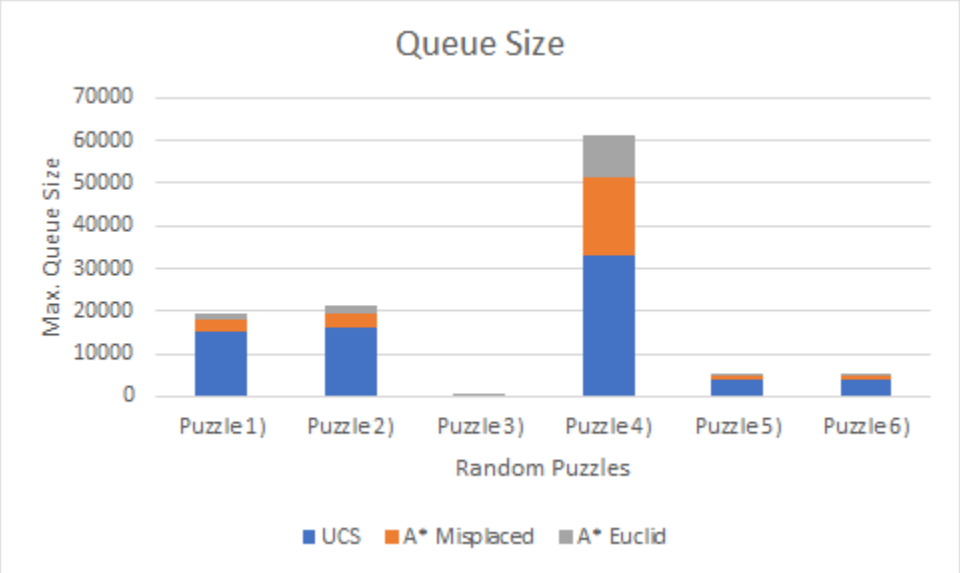


Figure 4.