Abhinav Allam (aalla009) 862200322
Jacqueline Gardea (jgard046) 862305608
Majd Kawak (mkawa025) 862273310
Rovin Soriano (rsori013) 862149089

Project 1 Report

Introduction

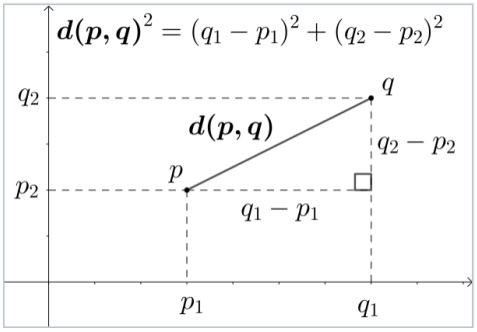
This program's primary goal is to solve Eight Piece Puzzles, we also made it to solve bigger puzzles(>3X3). Our algorithms consists of Uniform Cost search algorithm to search for a solution, as well as 2 different variations of A* algorithm with Misplaced Tiles Heuristic and Euclidean Heuristic. Users can input their own puzzle if they decide to, programs also have a default puzzle embedded in. Upon finding a goal state, the program will output the solution path from root to solution node.

Design

Uniform Cost Search (Done by Jacqueline Gardea):
Uniform cost search expands the nodes with the cheapest cost. When the cost is one for g(n) it is basically a breadth first search. The algorithm implemented uses a graph search and keps the previous states stored so it doesn't visit them again. The code to find previous states was optimized by using a row major hash function to store each state depending on where the blank (0) spot was located. A priority queue was used to expand the cheapest state first based on only the current g(n).

A* with Misplaced Tile Heuristic (Done by Abhinav Allam, Rovin Soriano):
When A* algorithm with Misplaced tile is chosen, it calculates f(n) based on g(n) + h(n), g(n) being the depth and h(n) being the current distance between the current node and the goal node by counting the number of misplaced tiles in the current puzzle. We will keep expanding the lowest f(n) till we reach our goal. For this one in order to minimize the cost search we constructed a tree data structure for tree search. This made the process more efficient because it avoids revisiting previously explored nodes since we added a function that checks if it's visited or not . We also used a priority queue which allows us to quickly access and expand the node with the lowest f(n) value.

A* with the Euclidean Distance Heuristic (Done by Majd Kawak):
When A* algorithm with Euclidean is chosen, it calculates f(n) based on h(n) which is the distance from "blank" tile to all misplaced tiles using:



Which minimizes the cost of search drastically compared to other search techniques, followed by utilizing a priority queue to decide which node to explore next based on the value of f(n). This was the best optimization method we can ever use. This algorithm discovers the most affordable path to a goal state. This algorithm uses Tree structure to store and search nodes, where we kept record of all individual nodes in the form of a linked list parent⇔child. We also implemented a vector data structure “Unordered Map” to store nodes as strings for fast lookups when trying to search a graph for a specific node(if it exists).

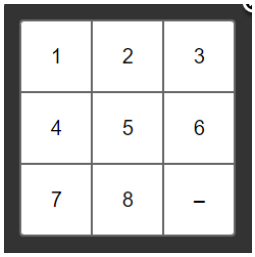
Challenges

Uniform Cost Search:
The priority queue for the function required some looking up to make sure that each node in the queue was sorted based on g(n). It required overloading the “>” operator to both make sure it was a minimum priority queue and was sorting based on the correct values.

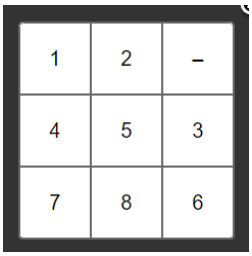
A* with Misplaced Tile Heuristic
We encountered some difficulties dealing with pointers not properly accounting for children of each node. That was remedied by updating the node structure to store children. An issue regarding the logic was fixed when we realized we failed to properly check for previous nodes which resulted in above normal depth. Additionally, changing the program to not only work with 3x3 matrices of puzzles was expanded to include any size matrices, and the code was altered to work as such which was fairly simple.

A* with the Euclidean Distance Heuristic:
We encountered some difficulties dealing with Node pointers and how to keep track of what nodes to explore and what nodes not to. We also ran into memory issues when we tried to save Nodes in a vector to check if a node exists or not, and found a way to convert a state to a string of sequential numbers that represent a visited state.

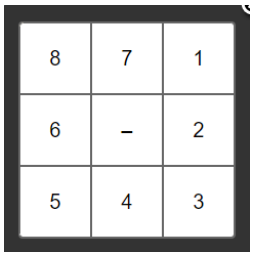
Test Cases



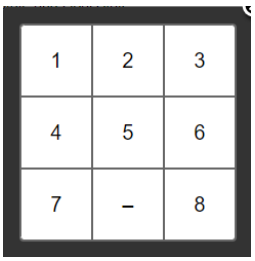
Trivial



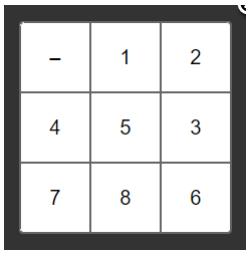
Easy



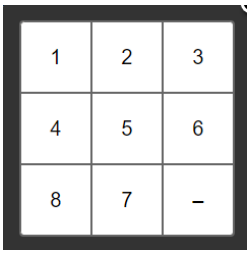
oh Boy!



Very Easy



doable

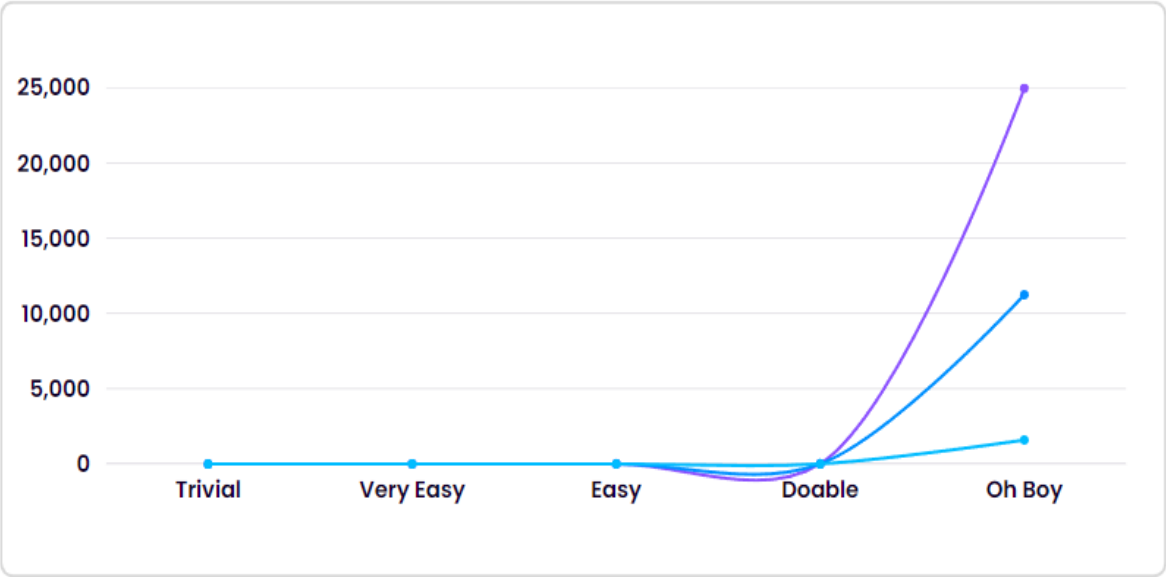


IMPOSSIBLE

Maximum Queue Size

	Uniform Cost Search	Misplaced Tile Heuristic	Euclidean Distance Heuristic
Trivial (0)	1	1	1
Very Easy(1)	3	4	4
Easy (2)	8	5	5
Doable(3)	18	8	8
Oh Boy (4)	25037	11257	1596
Impossible(5)	*impossible(818440)*	*impossible(181440)*	*impossible(181440)*

Graph

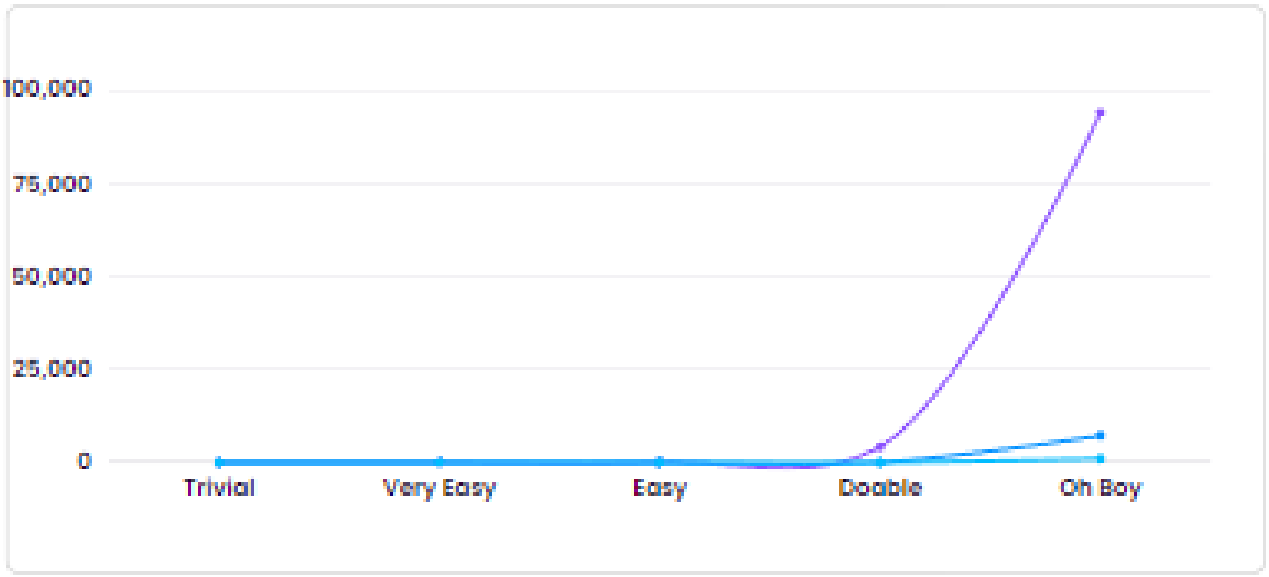


Number of Nodes Expanded

	Uniform Cost Search	Misplaced Tile Heuristic	Euclidean Distance Heuristic
Trivial (0)	0	0	0
Very Easy(1)	1	1	1
Easy (2)	6	2	2
Doable(3)	16	4	4
Oh Boy (4)	104645	7150	1022
Impossible(5)	*impossible(181440)*	*impossible(181440)*	*impossible(181440)*

Graph

- Uniform Cost Search
- Misplaced Tile Heuristic
- Euclidean Distance Heuristic



Findings

Upon examining the above graphs, where we denoted all cases max queues and nodes explored, it becomes clear that Uniform Cost Search's performance deteriorates drastically when our puzzles get more difficult. Although, it's worth noting that all 3 algorithms have similar performance when the puzzle's difficulty level is low.

An area we observed dramatic change is memory when using Uniform Cost Search, it demanded high memory and computing power to store and sort nodes. On the other hand, A* with both Misplaced Tiles and Euclidean Distance outperformed Uniform Cost Search in each and every way, this indicates their superiority against Uniform Cost Search.

What gave A* With both Misplaced Tiles and Euclidean Distance the upper hand was their way to to identify optimal paths by considering both h(n) and g(n). These two algorithms possess an edge, as they take into account the cost of transitioning from the current node to the goal state (h(n)), as well as the cumulative sum from the starting node to the present node (g(n)).

In conclusion, A* in general combined with any heuristic is one of the most optimal algorithms in solving complex puzzles. While Uniform Cost Search has other benefits in other areas of search.

```
A* with the Euclidean distance heuristic.
Goal Found
-------------------------------
State:
1 2 3
4 5 6
7 8 0
Heuristic values:
f = 4, g = 4, h = 0, children count = 0
-------------------------------
Depth 0
State:
0 1 2
4 5 3
7 8 6
Heuristic values:
f = 0, g = 0, h = 0, children count = 2

Depth 1
State:
1 0 2
4 5 3
7 8 6
Heuristic values:
f = 4, g = 1, h = 3, children count = 2

Depth 2
State:
1 2 0
4 5 3
7 8 6
Heuristic values:
f = 4, g = 2, h = 2, children count = 1

Depth 3
State:
1 2 3
4 5 0
7 8 6
Heuristic values:
f = 4, g = 3, h = 1, children count = 2

Depth 4
State:
1 2 3
4 5 6
7 8 0
Heuristic values:
f = 4, g = 4, h = 0, children count = 0
```