Michael Trinh

CS 4200

Professor Atanasio

# Project 1: 8-Puzzle

# Introduction

In Project 1, I have to use the A* algorithm to solve an 8-puzzle. The goal state I used for the project was the blank piece on the top left corner and the rest in ascending order (0,1,2,3,4,5,6,7,8). Unfortunately I ran out of time programming for this project and got stuck at a bug I cannot swiftly fix (Index out of bounds exception). Almost all of the code should be correctly done. However, from the work I've coded, I let the user choose whether to choose to generate random puzzles or input their own puzzle.

The A* algorithm would take each algorithm in the list of puzzles. In the algorithm, there is a frontier for the node, a state frontier and a estimated cost frontier to be used to check whether a child node's state is in the frontier and whether that child's estimated cost is less than the cost of a current node in the frontier. The initial state from the input puzzle is put into a node with a path cost of 0 and put into the frontier. This starts the loop where the node is removed from the frontier and checked whether it is the goal state or not. If not, the node will expand its branch into n children where each children is a swap from that blank space with an adjacent block in the puzzle, if possible.

This adds a depth to that current node to be used to calculate the search cost at that depth. Search cost for that depth will replace the correct cost in the list of search costs to get the most updated search cost value. This loop will continue until the frontier is empty. Heuristic 1 is calculated by comparing the goal state and the state and incrementing the cost whenever a tile in the node's state is different from the corresponding tile in the goal state. Heuristic 2 is calculated by the absolute value of subtracting the index of where the tile in the goal state by that tile in the node's state.

# Analysis

Because I didn't get to complete the program in time, I cannot show a table of my findings showing the average search costs of each heuristic per depth from depth = 2 to depth = 24. However, from how each heuristic works, heuristic 2 will always have average search costs that is less than or equal to the average search costs of heuristic 1. And as depth increases, average search costs for heuristic 2 will increase at a slower rate than heuristic 1. The reason for this is that the number of misplaced tiles in heuristic 1 progresses differently than in heuristic 2.

In heuristic 1, if none of the actions in from the node will make a tile in the correct spot, the heuristic cost will be the same for each child. This will possible create unnecessary expansions, increasing the search cost, unless the order of the removed nodes from the frontier was optimal. In heuristic 2, the manhattan distance in a child order the frontier in a more optimal way than heuristic 1. This is because the manhattan distance guarantees that progress in the puzzle will be done since the tiles that are changed in a way that estimate cost is decreased if the changed tiles are closer to the goal state and increased if the changed tiles are farther from the goal state.

# Findings/Conclusion

Overall, heuristic 2 yields a much lower average search cost rate than heuristic 1. This is due to how each heuristic works. Because heuristic 2 yields a lower average cost than heuristic 1, solving an 8-puzzle under heuristic 2 is faster than heuristic 1 since expanding more nodes means that more nodes are tested in the frontier. Although I couldn't test how long each heuristic will take in the program, I estimate that the time to solve each puzzle under heuristic 2 will take a

fraction of the time done under heuristic 1. But the times would not be as accurate as it would be if tested under a correct version of my program due to the code in my program not being optimized in terms of memory.