Cesar Pedroza
Professor Tang
CS 420
3 February 2017

8-puzzle Project Report

My approach to this project was to create byte arrays that held puzzles. Made a Node class that held all of the information about the board states. The puzzleResult class creates an object that stores the final information about the solution that was found like the amount of nodes we expanded and the time started so we can calculate the runtime. The manhattanLookUp array is to easily determine how far a position is to being in the correct one.

The comparison between the Manhattan heuristic and the Misplaced Tiles heuristic is that Manhattan is way more efficient for 8 puzzle. The reason for this is because the misplaced heuristic only cares about if the tile position is wrong. If it is wrong it will change it even if hurts the path to the solution because it is only accounting for it being misplaced. The Manhattan heuristic takes into consideration how much the position of the tile is far from the goal position of the tile. Unlike misplaced which only cares about if its “correct position” and “incorrect position”, Manhattan takes more into account like how much it would cost to go through this node and if I take this node, how much does it take me closer to the goal.

I initially thought that Manhattan heuristic would lead to better results because on paper it does look smarter. I did find that it was a lot more efficient than misplaced tiles in both run time and the number of node it created (search cost). I was confused how I was suppose to implement the chart after I had wrote the createRandomPuzzles() function because I just made a function that solved a board and outputted the depth of the path solution, each of the search costs for the heuristics, and the run times of each heuristic. I noticed that the depth of both solutions for the same puzzles were the same. Not sure if there was an error on my part about that. I had written the program to output runtime in milliseconds but then I noticed that for Manhattan, the run time would be so fast that it would come out to be 0 for some of the solutions. I thought this would be a problem with the average runtime but a 0.68 millisecond that was left out wouldn't affect the average that much anyway.

Below is a table I made after running the program thought 200 different puzzles at a specific depth starting from 2 all the way up to 20. I had used the long text file from your project page.

d	A*(h1) SC	A*(h2) SC	A*(h1) RunTime (ms)	A*(h1) RunTime (ms)
2	5.04	5.04	0.05	0.185
4	8.78	8.57	0.06	0.095
6	15.625	13.29	0.075	0.045
8	30.065	19.995	0.07	0.015
10	68.205	33.015	0.125	0.035
12	158.015	56.93	0.21	0.055
14	374.805	107.345	0.93	0.23
16	899.53	201.595	2.065	0.155
18	2287.545	376.425	12.53	0.555
20	5598.81	666.505	51.375	0.945