Eight Puzzle Project Report

Dylan Chung

CS 4200.01 Spring 2020

Dominick Atanasio

February 23, 2020

Program Description:

The program solves an 8 puzzle with specifically the goal state <0-8>, using A*Graph and A* TreeSearch implementations. The user may evaluate the puzzle using Hamming Distance or Manhattan Distance as a heuristic. The user may either generate a random solvable puzzle or enter one of their own. The program makes sure that the puzzle inputted, whether by user input or randomness, is a solvable puzzle by calculating the number of inversions and checking that it is not odd. It also returns the intermediate steps used to achieve the solutio, the cost to reach there, the depth, and the time it took, if a solution is found.

My Approach:

My program contains 3 classes; (Driver, Node, and Puzzle). In short, a Puzzle object is created to go into a Node, and the Nodes which hold these Puzzle Objects are linked together in a graph/tree like structure by setting their child/parent nodes to the proper references. In Driver, which contains the UI and the main function, we run the entire program by creating a root Node object, and performing the desired algorithm with the desired values on that Node, which in turn generates childrens according to the specified search algorithm. The search algorithm returns the final goal node, and with that we iterate up the Node's parents to achieve the list of Nodes used to get to that point from beginning to end.

To go in depth, the Puzzle class holds the fields int [] puzzle, int h1, h2, totalCost, and direction. The puzzle is simply an array that holds the array presentation of the puzzle. The "h1" and "h2" values refer to the Hamming and Manhattan distance respectively. Direction is a

numerical representation of which direction the current puzzle configuration moved from in order to achieve the current puzzle. This direction is determined by getting the current blank space index of the puzzle and seeing which indexes it can move up, down, left, or right into without going out of bounds or moving in a way it should not.

Node holds the Puzzle Objects as well as has fields to tie them together with a child and parent node, if applicable. This allows for traversal of the Nodes/Puzzles and also for creating and linking nodes together. The search algorithm were implemented as the pseudocode was given, and only very slight changes were made. Data structures used included a PriorityQueue for the frontier. Hashset for the exploredSet. Integer array to hold the puzzle array. Nodes to hold the Puzzle objects.

Data:

	Search Cost (Graph Search)				
d	**A*** (using h1**)**	**A*** (using h2)	avg runtime using h1 (nanoseconds)	avg runtime using h2 (nanoseconds)	# runs
2	6	6	2172012000	362500	10
4	10	10	2107625300	92000	10
6	15	13	3220184100	79800	10
8	68	54	2020199600	110900	10
10	86	55	1397759900	823400	10
12	211	85	1794947300	904700	10
14	742	304	1545684400	1030600	10
16	1582	353	1298345300	564900	10
18	3907	1158	1844456300	1083800	10
20	14648	1484	1943203500	1545400	10

	Search Cost (Tree Search)				
d	**A*** (using h1**)**	**A*** (using h2)	avg runtime using h1 (nanoseconds)	avg runtime using h2 (nanoseconds)	# runs
2	7	7	1672272300	384600	10
4	12	12	2464054800	380500	10
6	46	21	1871997200	397600	10
8	59	29	1791186000	470700	10
10	144	60	1045999700	570200	10
12	333	205	2301859400	860200	10
14	1139	447	2932273400	793100	10
16	3320	1297	2040482300	2226700	10
18	5495	2200	1588790000	5103700	10
20	16332	2350	1586016300	5277900	10

Analysis:

From the data here, it is apparent that the A* Graph Search with Manhattan distance as the heuristic is a lot faster. It generates significantly less nodes as shown by the search cost. In some cases H2 will return a search cost that's even a tenth of it's H2 search cost for bigger depths. The further and further the depth goes, the more we can see how much better Manhattan is as a heuristic. As expected, since H2 generates less Nodes or in other words has a lower search cost, it also has an equally shorter average run time. It seems I did not run enough test-cases to

generate an accurate comparison between tree and graph, but in general it seems graph was the cheaper search.