Brandon Palomino
CS 420

8-Puzzle Project Analysis

Approach

The purpose of this project is to understand, apply, and evaluate the A* search algorithm implemented with Graph-Search and two specific types of heuristics in order to solve the 8-puzzle problem. The two heuristics used are denoted by h1 = number of misplaced tiles and h2 = the sum of the distances of the tiles from their goal positions (manhattan distance).

To begin my approach I define the structure of each state of the 8-puzzle that will be manipulated by the program. For simplicity I used a string of numbers with 0 indicating the blank tile to represent each state i.e. “012345678”. The main object being handled is a specific node I designed holding this string state, along with the evaluation cost f(n), goal cost g(n), and parent node. These nodes are implemented in a way where they can easily be managed by the heap to determine the node and state with highest priority. Leading the second part which is that I used a heap in order to handle prioritizing my lowest f(n) nodes. Third is to implement both heuristic functions noted earlier by handling index positions relative to 0 position in the string and summing tiles for manhattan distance. After all this has been structured out, the A* search can quite easily be applied to the 8-puzzle problem.
I implemented my version using two lists to represent the frontier and explored sets, a heap, and helper methods to check for the goal state and get a list of successors for a given state. When the goal is found the “current” node is returned which holds the path and costs.

Comparison of the Two Approaches

When testing and using the A* search algorithm using both heuristics, considerable difference can be observed. With h1 (misplaced tiles) we can see that as the solution depth increases so does the search cost which is the number of nodes, and the average time it takes to solve a specific puzzle of such depth. Which is normal but when compared directly to h2 (manhattan distance), one can see that h1 is several times more costly than h2 in all areas. When the solution depth hits the average case of 20 steps h1’s search cost is already larger than h2’s by thousands.

Other Analysis

In addition the comparison of the two heuristics by search cost and depth there was the comparison of average run time for puzzle solutions. I found this quite interesting in my data as it can be seen that for depths 2,4, and 6 A* search running h2 had a slower avg time than A* with h1. It could be due to the number of cases I used. I found that while the depth is smaller the search cost stays relatively equal and the run time perform very differently from when the depth increases to 20 along with the number of nodes being generated. Also I would point out in my analysis that although my data points seemed to be a smaller result and scale, they follow the same trend and nature of using the two heuristics side by side.

Results Comparison Table

	Search Cost		Average Run Time (ms)		
d	A*(h1)	A*(h2)	A*(h1)	A*(h2)	Number of Cases
2	5	5	0.171494483948	0.252509117126	10
4	10	10	0.355100631714	0.505924224854	10
6	18	17	0.744032859802	0.916838645935	10
8	34	26	1.62507692973	1.37402216593	15
10	70	42	3.36833000183	2.23083496094	20
12	152	74	8.14371109009	4.35514450073	25
14	361	195	20.00041008	11.9668324788	30
16	829	350	48.4106336321	21.4455263955	35
18	2219	888	133.3583951	56.5150678158	40
20	4872	1450	298.468556404	93.7810087204	50