California State Polytechnic University, Pomona

Solving 8-Puzzle with A* Using Two Heuristics

Project 1

Joshua Gibbs
CS 420 - Artificial Intelligence
Dominick Atanasio
11 February 2018

Approach

The project contains three class files, EightPuzzle, Astar, and Node. The Node class simply holds the state for that node, the parent node if there is one, the number of steps it took to get there, and the total cost given by f(n) = g(n) + h(n). Node also contains functions for generating new nodes with states after the four “up”, “down”, “left”, and “right” actions. The Astar class works by receiving an initial node, adding it to the frontier, and beginning the search. Search grabs the lowest cost node from the frontier, removes it from the frontier, and adds it to an “explored” hash set. Search then explores the node by performing the four actions on it, and adding them to the frontier if they are valid actions. Afterwards, we traverse up the tree through parent nodes to get the solution. Nodes are stored as a unique key in a frontier and explored hash table to optimize lookup times.

H1 vs. H2

The first heuristic we use is simply the number of misplaced tiles. The second heuristic calculates the sum of distances of the tiles from their goal position. Puzzles with complexities 1-23 performed on average better using heuristic one by creating fewer nodes and running for a shorter amount of time. However, beginning at complexity level 24, heuristic two outperformed heuristic one in nodes generated and time run. We can see this by analyzing the averages for each complexity in **Table A**, which was created by generating 500,000 random nodes.

Table A

d	Nodes Generated		Run Time		Number of Cases	
	A*(h1)	A*(h2)	A*(h1)	A*(h2)	A*(h1)	A*(h2)
0	1	1	0ms	0ms	6	6
1	3	3	0ms	0ms	8	8
2	4	5	0ms	0ms	8	8
3	7	9	0ms	0ms	23	23
4	8	14	0ms	0ms	44	44
5	11	23	0ms	0ms	53	50
6	16	33	0ms	0ms	103	101
7	22	54	0ms	0ms	171	159
8	31	73	0ms	0ms	297	293
9	48	120	0ms	0ms	446	439
10	67	181	0ms	0ms	793	752
11	105	289	0ms	0ms	1081	978
12	156	399	0ms	0ms	2035	1836
13	242	641	0ms	0ms	2679	2392
14	363	910	0ms	0ms	5234	4632
15	590	1399	0ms	0ms	6962	5925
16	833	1957	0ms	0ms	12398	10406
17	1429	2922	0ms	1ms	15407	12636
18	1956	4068	0ms	2ms	26197	21113
19	3249	5910	1ms	3ms	29959	23750
20	4905	7967	2ms	4ms	46874	37384
21	7195	11291	3ms	6ms	46754	37581
22	11853	14548	6ms	8ms	66071	53934
23	16903	19986	9ms	12ms	55246	47827
24	25826	25203	15ms	16ms	66112	61898
25	35436	33442	22ms	22ms	43239	46725
26	51892	40930	37ms	28ms	40357	50864
27	63401	53628	46ms	40ms	17532	30668
28	82123	64021	64ms	49ms	11100	27506
29	107136	80680	93ms	65ms	2190	10856
30	121803	93571	111ms	78ms	616	6898
31	145071	112572	138ms	102ms	5	1687
32		126385		121ms	0	561
33		144981		144ms	0	51
34		153962		153ms	0	9

Findings

Analyzing the data above, we notice that when solving a complex puzzle, it is better to use a more complex heuristic. However, easy puzzles are only only slowed down by complex heuristics.