Taylor Thurlow

23 September 2018

CS 4200 Fall 2018

Solving the 8-Puzzle with the A* Search Algorithm

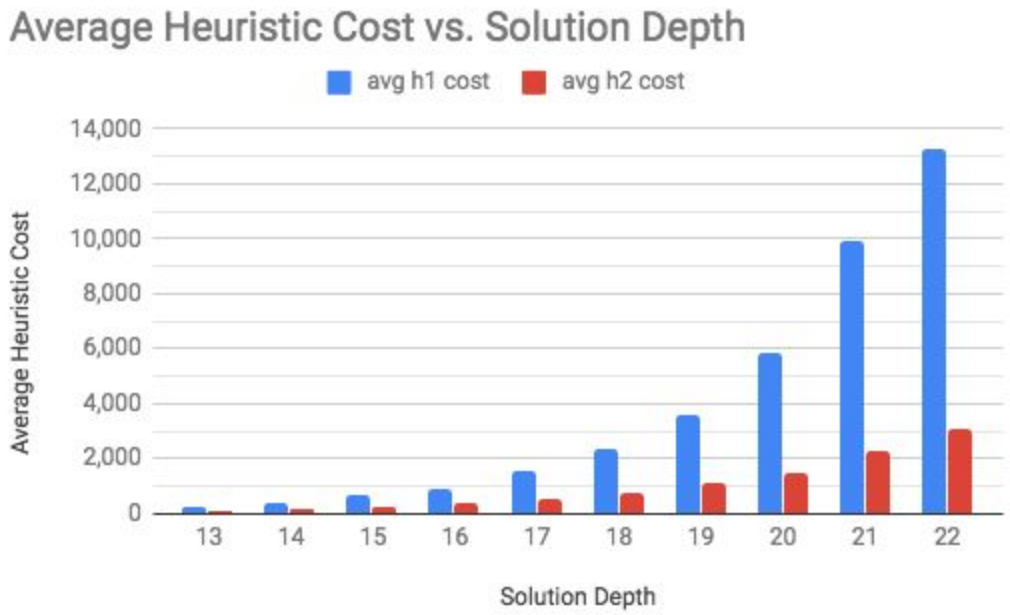
In order to represent the 8-Puzzle and A* programatically, I chose a simple object-oriented approach. Each puzzle state exists as its own object and every state except the puzzle's start state has a parent state. Each state stores the cost to reach that state, which is calculated by getting the depth of the parent state, adding 1, and adding the heuristic function. Once the goal state is found, finding the path from the start state to the goal state is as simple as following the parents starting from the goal state.

In order to test both heuristics, 2000 (not necessarily unique, but random) puzzles were generated. Randomly generating an 8-Puzzle results in an average solution depth of around 22, which is solvable in a reasonable amount of time. That said, solution depths of 24 or more are not uncommon, and take significantly more time to solve. Generating 2000 puzzles with no restrictions on solution depth is possible but the time required is prohibitive. In order to make things easier, I chose to cancel the calculation of the solution if the solution reaches depth 23 or more. If the solver reaches that point, the puzzle is thrown out and a new, random puzzle is generated. This continues until we have 2000 solved puzzles. Another problem with the high average

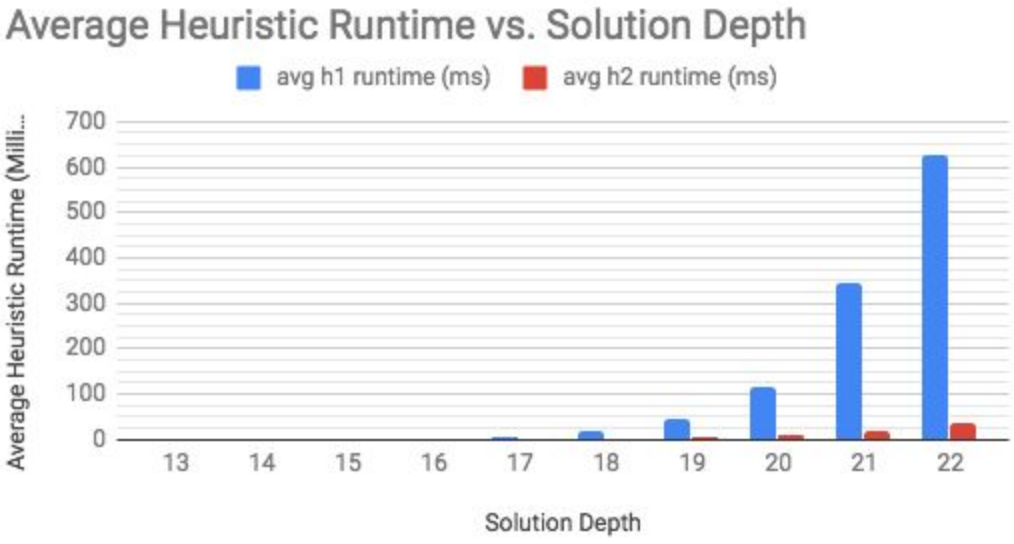
solution depth is the fact that it is rare to generate puzzles with solutions with depths lower than around 12 or 13. Of all 2000 solutions, only 30 of them were below depth 12. As such, it is important to remember that there is less meaning behind the data from these low-depth solutions. In some charts, I have chosen to eliminate the lower values entirely.

value	occurrences	avg h1 cost	avg h2 cost	avg h1 runtime (ms)	avg h2 runtime (ms)
22	458	13,247	3,087	625.637227	38.671398
21	373	9,896	2,262	345.030566	19.851946
20	310	5,813	1,503	117.194650	8.378062
19	284	3,579	1,095	43.345033	4.290332
18	175	2,323	762	17.581544	2.045512
17	134	1,528	533	6.964960	1.061252
16	90	911	363	2.461638	0.497938
15	58	650	215	1.226736	0.176683
14	46	378	163	0.434361	0.115421
13	21	243	126	0.195042	0.073553
12	14	162	91	0.100322	0.046242
11	9	106	51	0.050861	0.018222
10	7	75	49	0.030718	0.017030
9	5	38	34	0.013928	0.010655
8	3	29	27	0.008683	0.007259
7	4	24	20	0.007112	0.004932
6	2	12	19	0.003788	0.004632
3	1	9	10	0.002327	0.001895
2	1	5	5	0.001174	0.000706
0	1	1	1	0.000283	0.000133

Plotting the average heuristic cost versus the solution depth shows how quickly these puzzles increase in solution complexity. In the following chart, the y-axis represents the number of states generated in order to find the solution.



It is immediately clear that heuristic 2 (Manhattan distance) scales much better.



In general, these results show that it is worth spending the time finding a heuristic which is both optimistic in all cases, but as accurate to the true cost as possible. Looking at the average runtime graph, solutions at depth 22 took an average of about 39 milliseconds using Manhattan distance, whereas the Hamming distance took more than 8 times as long, generating more than 4 times the number of puzzle states. There is surely more room for optimization, but the Manhattan distance proves to be an excellent heuristic.