Chris Saba

CS4200 Artificial Intelligence

7/19/20

Project 1

8 Puzzle solver using A\* with Graph Search, using 2 Heuristic functions.

An 8 puzzle is a game that can be played using a number scale from 0-8, numbers adjacent to 0 can be swapped with it in a 3x3 matrix, but can be on an endless scale. Not every configuration is solvable, the program will calculate if the puzzle is solvable, then use A\* to solve it.

Here is a table in accordance with depth of 100 different puzzles with different depths.

--------------------------Averages over 100 iterations-------------------------------------

d | Total Cases | Search Cost H1 | Total Time H1 | Search Cost H2 | Total Time H2

----------------------------------------------------------------------------------

13 | 1 | 251 | 1 ms | 64 | 0 ms

----------------------------------------------------------------------------------

14 | 2 | 491 | 1 ms | 132 | 1 ms

----------------------------------------------------------------------------------

16 | 4 | 920 | 4 ms | 141 | 0 ms

----------------------------------------------------------------------------------

17 | 2 | 1688 | 7 ms | 392 | 1 ms

----------------------------------------------------------------------------------

18 | 4 | 2304 | 9 ms | 230 | 1 ms

----------------------------------------------------------------------------------

19 | 8 | 3441 | 15 ms | 430 | 2 ms

----------------------------------------------------------------------------------

20 | 6 | 5712 | 26 ms | 720 | 3 ms

----------------------------------------------------------------------------------

21 | 6 | 8105 | 36 ms | 761 | 3 ms

----------------------------------------------------------------------------------

22 | 15 | 12907 | 63 ms | 1138 | 5 ms

----------------------------------------------------------------------------------

23 | 10 | 21809 | 104 ms | 1410 | 6 ms

----------------------------------------------------------------------------------

24 | 15 | 28220 | 138 ms | 2013 | 9 ms

----------------------------------------------------------------------------------

25 | 11 | 50050 | 255 ms | 3558 | 16 ms

----------------------------------------------------------------------------------

26 | 10 | 61207 | 323 ms | 3544 | 17 ms

----------------------------------------------------------------------------------

27 | 3 | 94561 | 515 ms | 5987 | 29 ms

----------------------------------------------------------------------------------

28 | 2 | 105241 | 574 ms | 5539 | 27 ms

----------------------------------------------------------------------------------

30 | 1 | 170434 | 1000 ms | 11534 | 56 ms

Looking at the table, we see that Heuristic 1 (the number of misplaced tiles) is much slower than Heuristic 2 (the sum of the distances of the tiles from their goal positions.) AKA Manhattan distance. The difference between the heuristics may be subtle, but there is a big difference between knowing that a tile is misplaced, and knowing how far a tile is from its goal position. For almost-worst case scenario, depth 30, we see that the search cost for H1 is at 170,434. This is close to the actual number of reachable states, being 181,440. Much lower is the second heuristic, at 11,534. An entire second calculates H1 while only 56 ms is H2. It seems as though H1 is moving at random, trying to get new states and see which tiles are in the right position. H2 seems to be a smarter approach, as it takes into account all tiles and how far their step cost is from the goal position, slowly moving them in the right direction. H2 is proven to be much more effective, and there are possibly even better heuristics that can solve it much faster.