Jason Farkas

AI: Assignment 1: 8-Puzzzle Project

**Intro**:

This puzzle is a frame that contains pieces that are each numbered randomly and can slide into a blank position such that eventually they can be placed in order. The problem can be categorized as a fully observable, deterministic, static, discrete problem. Even though in their categorization it appears to be on the easier end of the spectrum it is could be viewed as a difficult question for a computer to solve. This can be shown through the following calculations:

For a 3 by 3 board, there are 9! potential board states as determined by the permutations of the 9 distinct pieces. There are on average 2.6 potential moves calculated via (4\*1+3\*4+2\*4)/9. Assuming a solvable worst-case scenario, created by placing 4 tiles each 4 squares from their goal position and the other 4 tiles 2 squares away, creating a Manhattan distance of 24 between the start and goal state, takes at least 30 moves to solve as will be shown via the hardest starting state later. I made this estimate by maximizing the total distance between the positions in the starting state compared to the desired positions in the end state as measured by Manhattan distance. The other way to maximize the Manhattan distance between the start and goal state to 24 seems to create an unsolvable puzzle leaving the problem solved later to be the worst case scenario solvable in minimum 30 moves.

Since a breath first search approach will find the answer at bd, being the branching factor to the depth of the first solution, it will have search through all nodes on previous levels before arriving at the solution. This could be understood as the sum of i starting from 0 until he depth ( bi ). In this case it would sum to 1.7e12 nodes that must be inspected before arriving at a solution assuming the branching factor is 2.6 and the depth is 30 as explained. Presuming a computer could examine one move per millisecond the calculation would require around 55 years to compute as calculated by 1750000000000moves/ 1000(moves/milliseconds)/ 60(seconds/minute)/ 60(minutes/hour)/ 24(hours/day)/ 365.0(days/year).

Thus a brute force solution to the problem will take up so much time that the problem can be considered hard. This is where the realm of AI becomes useful. Using various AI algorithms is it possible to obtain the optimal solutions to this problem within a very short amount of time. The rest of this paper details the process of using 3 distinct AI algorithms, with variations in heuristics, to solve 4 different example cases of this problem.

**Methods**:

I implemented the A star, Depth first branch and bound, and Iteratively deepening a star algorithms in python using a Manhattan distance heuristic and also a additional run of a start with an out of place heuristic. My implementation uses 3 classes and one main program to run the simulation. I created one class to hold the information regarding a specific board state and another class that extends the first class to hold node related information such as a parent state and a time created. Additionally, I designed an AI class that takes in a starting state and end state and contains various functions that implement AI algorithms that return an end state that details the path from the start state till that end state.

To run the simulations on the assigned problems I instantiated 4 AI objects, passed them each the same start and end state, saved each generated solution state, printed out relevant data to three separate files, and repeated this process for each unique start state.

**Results**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Board# | Algorithm | Time of Algorithm | Expanded Nodes | Time till Optimal Solution | Steps to get there |
| 1 | a\_starwith a out of place heurisitic: | 0.00343 | 22 | - | ['up', 'right', 'up', 'left', 'down'] |
| 1 | a\_starwith a Manhattan distance heurisitic: | 0.002336 | 15 | - | ['up', 'right', 'up', 'left', 'down'] |
| 1 | Depth-first branch and boundwith a Manhattan distance heurisitic: | 0.005353 | 29 | 0.002346 | ['up', 'right', 'up', 'left', 'down'] |
| 1 | Iterative deepening a\*with a Manhattan distance heurisitic: | 0.004134 | 25 | - | ['up', 'right', 'up', 'left', 'down'] |
| 2 | a\_starwith a out of place heurisitic: | 0.017601 | 131 | - | ['up', 'right', 'right', 'down', 'left', 'left', 'up', 'right', 'down'] |
| 2 | a\_starwith a Manhattan distance heurisitic: | 0.010446 | 70 | - | ['up', 'right', 'right', 'down', 'left', 'left', 'up', 'right', 'down'] |
| 2 | Depth-first branch and boundwith a Manhattan distance heurisitic: | 0.066445 | 333 | 0.063124 | ['up', 'right', 'right', 'down', 'left', 'left', 'up', 'right', 'down'] |
| 2 | Iterative deepening a\*with a Manhattan distance heurisitic: | 0.007137 | 59 | - | ['up', 'right', 'right', 'down', 'left', 'left', 'up', 'right', 'down'] |
| 3 | a\_starwith a out of place heurisitic: | 0.096427 | 529 | - | ['right', 'up', 'left', 'up', 'right', 'right', 'down', 'left', 'left', 'up', 'right', 'down'] |
| 3 | a\_starwith a Manhattan distance heurisitic: | 0.023066 | 137 | - | ['right', 'up', 'left', 'up', 'right', 'right', 'down', 'left', 'left', 'up', 'right', 'down'] |
| 3 | Depth-first branch and boundwith a Manhattan distance heurisitic: | 0.049896 | 247 | 0.045045 | ['right', 'up', 'left', 'up', 'right', 'right', 'down', 'left', 'left', 'up', 'right', 'down'] |
| 3 | Iterative deepening a\*with a Manhattan distance heurisitic: | 0.048532 | 366 | - | ['right', 'up', 'left', 'up', 'right', 'right', 'down', 'left', 'left', 'up', 'right', 'down'] |
| 4 | a\_starwith a Manhattan distance heurisitic: | 2.671292 | 4743 | - | ['up', 'left', 'down', 'down', 'right', 'right', 'up', 'up', 'left', 'left', 'down', 'down', 'right', 'right', 'up', 'up', 'left', 'left', 'down', 'down', 'right', 'right', 'up', 'up', 'left', 'left', 'down', 'down', 'right', 'up'] |
| 4 | Depth-first branch and boundwith a Manhattan distance heurisitic: | 3.670632 | 15371 | 3.473856 | ['up', 'left', 'down', 'down', 'right', 'right', 'up', 'up', 'left', 'left', 'down', 'down', 'right', 'right', 'up', 'up', 'left', 'left', 'down', 'down', 'right', 'right', 'up', 'up', 'left', 'left', 'down', 'down', 'right', 'up'] |
| 4 | Iterative deepening a\*with a Manhattan distance heurisitic: | 3.599851 | 26024 | - | ['up', 'left', 'down', 'down', 'right', 'right', 'up', 'up', 'left', 'left', 'down', 'down', 'right', 'right', 'up', 'up', 'left', 'left', 'down', 'down', 'right', 'right', 'up', 'up', 'left', 'left', 'down', 'down', 'right', 'up'] |

Here are some charts to depict the results visually. I broke up the graphs into two sets one set for all starting boards and one for just the first three starting boards since the fourth case is very different. As we can see from the charts there seem to be 3 factors affecting the efficiency of our results: Algorithm, Heuristic, and Problem. We can clear see how different algorithms consistently expand more or less nodes or take more or less time relative to other algorithms. A star tends to be the fastest as well as expand the least number of nodes while branch and bound is the slowest but iterative deepening search expands the most nodes. We can also see that a star algorithm performs radically differently depending on the heuristic used. While a star with Manhattan distance performs the best a start with the out of place distance performs the worst out of everything and can’t even produce a result within 30 minutes for the last problem. Lastly, from the second set of charts we can most starkly see that changing the difficulty of the problem completely changes the performance of the algorithms.

Each algorithm has its advantages and disadvantages. A star is optimal when the goal is to expand as few nodes as possible and memory is not an issue. Iterative deepening a start is optimal when memory may pose a problem and expanding nodes is of no concern. Branch and bound is best when memory is a concern and so is the expansion of nodes, but time is not of the essence. Additionally, if you can use any good answer Branch and bound can be very useful as it provides a solution very quickly and it provides the optimal solution fairly fast although it takes time to confirm it. It appears that the harder the problem the faster Branch and bound will obtain the optimal solution in relationship to iterative a star so this is another factor to keep in mind when selecting the optimal algorithm.

**Discussion**:

Although the worst-case problem is hard for computers is tends to be the easiest one for humans. The first thing that stand out to a human, even one who is not attempting to solve the problem, is that rotating one board will result in the other board. This suggests a number of differences between a human and the AI algorithms. Firstly, humans do not automatically view each problem independent of other potential problems/questions while the AI algorithms are confined to a very specific problem and are looking for a particular solution. This difference enables humans to approach problem a in a different way but coming up with related problem b and then applying related problem b’s solution to problem a. Namely, in this case humans wonder how to transform the start board into the goal board without the constraints of the game and then apply the knowledge that rotation solves the first problem to figure out how to rotate the pieces within the constraints of the game. Secondly, human see the interrelationship between different states in multiple ways, as humans not only see the comparison between each individual value’s location between the boards but also the comparison between the relationships in the board. Since both boards contain a similar relationship between the different values in that starting from some position and going around the tile are sorted in both of the boards. The third and final point is that humans see patterns between different states and their transitions such that a human can figure out how to move a few pieces such that they are all moved over to the right without changing their relationship and then the human can reapply the logic to the rest of the board understanding that will lead to the correct result. The algorithms only look at a single transition between states and therefore they do not evaluate patterns of actions as if they were actions within themselves. Thus humans can evaluate patterns of actions to skip the evaluation of various levels of the search tree. Thus these AI algorithms don’t have the same type of creativity, pattern recognition, or abstraction abilities as humans.