Jacob Anderson

Dr. Phillips

OLA 1 Report

The 8-puzzle problem is a puzzle invented by Noyes Chapman in the 1870's. It is played on a 3x3 grid with 8 square tiles labeled 1-8. The goal of solving the puzzle is to rearrange the tiles so they are in order. The rules of the game are that you can move the tiles up, down, right, and left. You are not allowed to move the tiles diagonally. An example of the puzzle can be seen below in figure 1.

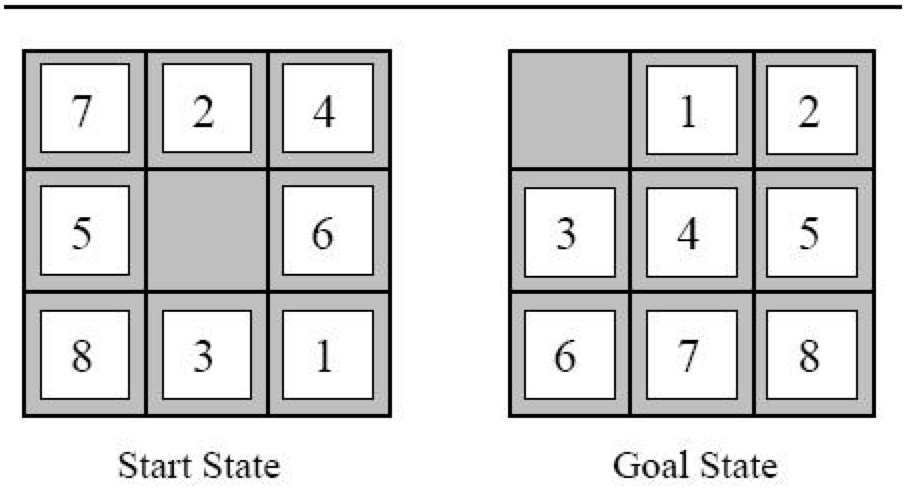

Figure 1 Example of a 8 puzzle problem

For this open lab we were assigned to create a python program which uses random actions to generate a random start state for the 8-puzzle problem called Random_board.py. After this we were to develop a program called A-star.py that takes in the board generated from random_board.py and solves it to the goal state. Once it is solved to the goal state the program outputs the optimal path to get to the goal state, the total number of nodes visited, the total number of nodes stored in memory, the depth of the optimal solution, and the approximate branching factor.

The code developed for this project takes in 3 arguments at the command line for the program to know what it must do. Random_board.py takes in a number to generate the random seed to help generate the board and another number to know how many random moves to make to make the different board. A-star.py takes in a number 0-3 to know which heuristic to use to help solve the puzzle. A heuristic is defined as "A rule of thumb, simplification, or educated guess that reduces or limits the search for the solutions in the domains that are difficult and poorly understood."

The a-star.py program uses the A-star search algorithm to find the shortest path to take to achieve the goal state. At each state the algorithm picks the node/state according to a value-f. The algorithm picks the node with the lowest f-value and then expands that node. The F-value is found by calculating F = g + h. Where g is equal to the movement cost to move from the staring point to a given square on the grid following the path to get there, and where h is equal to the heuristic. In a-star.py there are 4 heuristics used.

- h = 0
- h = the number of misplaced tiles in the start state
- h = the sum of Manhattan distances of all tiles from the goal state
- h = the number of tiles in the right spot in the start state.

Once these heuristics were calculated by using function definitions that I wrote, the f-value is then calculated and assigned to one of the children nodes of the starting state of the puzzle. From here the algorithm selects the child node with the smallest f-value and expands that node. From then on, this process is repeated until the goal state is achieved.

	h (0) average / median	h (1) average / median	h (2) average / median	h (3) average / median
V- total number of nodes visited	883.8 / 43	2,945.94 / 5	2,260.27 / 5	2,253.52 / 375
N- The max number of nodes visited	2,477.06 / 119	8,287.43 / 13	6,441.64 / 13	6,328.93 / 1,042
d – The depth of the optimal solution	4.84 / 5	4.84 / 5	4.84 / 5	4.84 / 5
b – approximate branching factor	2.698524 / 2.771171	1.873095 / 1.817121	1.84688 / 1.8171271	4.102984 / 4.012984

Figure 2 Average and median of heuristic data gathered from a-star.py

On the table in figure 2 you can see the average and the median of how each heuristic preformed while traversing through the tree. According to the table h (0) had the lowest average of total number of nodes visited between h (1), h(2), and h(3). This would make sense because the A-star algorithm pops off the node with the smallest f-value. However, if you look at the median of h(0), h(1), h(2), you will see that both h(1) and h(2) are significantly smaller than h(0). This is because with a-star the time complexity becomes exponential with increase in path length. So, for the most part h (1) and h (2) visited less nodes than h (0), until the algorithm ran in to more complex paths then the value of g rose exponentially making h (1) and h (2) increase in time complexity.

Throughout the project I tried to implement node and state to be the same thing, but it eventually clicked to me that it would be easier to think of them as two separate entities. I would append the states in to their own open list at the same time that I would append the nodes to the open list and I would use the I would use the id value on the nodes class to pull the associated state with the node.

Works Cited

Aiai.ed.ac.uk. (2019). *The 8-Puzzle*. [online] Available at: http://www.aiai.ed.ac.uk/~gwickler/eightpuzzle-uninf.html [Accessed 27 Sep. 2019].

Informed Search PowerPoint