Elements of Artificial Intelligence – Assignment 1

Problem 2, 15 puzzle submitted by Vinay Vernekar (vrvernek@iu.edu)

State space: Is the possible configuration of the 15 puzzle tile. In the 15 puzzle we two distinct states i.e. solvable and non-solvable; if the initial puzzle is in solvable state, it will be solvable and if the initial board is in not solvable state it will remain non solvable.

A pair of elements (pi,pj) is called an inversion in a permutation p if i>j and pi<pj odd permutations of the puzzle are impossible to solve, all even permutations are solvable. So in our given model based on the initial board and given the swap property, our state space can be any state which can be reached by making a legal move. A legal move is one move where the numbered tile can move into the place of the blank tile with one move including the swap property (without diagonal movements). Given the problem we have 16 factorial states i.e. close to 2.09*10^13 states.

Start state: Is the input that the user gives to the program. In this case it is a 15 tile problem where the user will input a board configuration consisting of tiles numbered from 1 to 15 and one blank tile numbered 0 arranged in 4 by 4 grid. It can either be solvable or non-solvable.

Goal State: The goal is to arrange the tiles from a given starting arrangement by sliding them one at a time into the configuration where all the tiles are arranged in ascending order like (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,blank) on a 4 by 4 grid. There is only one goal state.

Successor function: The successor function is defined as, given each state and the action taken on that state, what will be the outcome. In the given problem, each legal move will either help us reach closer to the goal state or take us away from the goal state. The successor function will move each tile towards the blank tile. This move also includes the swap move i.e. when the empty tile is on the edge of the board, the tile on the opposite side of the board can be slid into the opening. The successor function will calculate the cost for each successor state and return it for further evaluation.

Cost: The cost in the given problem consists of the sum of cost required to make one legal move (1) so far till the given state and the number of moves required to reach goal state. In our case we have taken heuristic as the summation of the cost (1 move) required to move the tile in legal direction and after the move we calculate the cost or moves required to reach the goal state. A* search is used for this problem. So number of moves required to reach each state is maintained and incremented for every move of the given board.

<u>Heuristic function:</u> The Heuristic consists of the summation of the legal move so far plus the modified Manhattan distance. Each legal move is considered to add a cost of 1.

<u>Working of the program:</u>

The program consists of the following parts

a)is_solve function: This function takes the initial board and returns whether it is solvable or not.

b)is_goal function: Check whether the board passed to it is the final or goal state

c) Fringe: is a list consisting of tuples of cost, state and directions arranged in terms of costs

d)Visited: is a list consisting of states which were visited or evaluated earlier

e) Successor(movement): Takes the board and make the legal move and passes the information to the heuristic function which is the cost function

d)Heuristic function: The heuristic functions take the input from the successor function and then calculate the cost which includes two parts as in the A* does. The cost consists of the number of moves required to reach the current state and the cost required to get to the goal state from the present state.

The fringe is sorted as per the cost and the one with least cost is picked and passed on for further evaluations to check whether this state was evaluated earlier or not. Based on the evaluation this state is appended to the visited list. If it's being vested for the first time and then, it is passed on to the next stage where the successor function is called to make moves and then pass each successor state to the heuristic function. The heuristic function calculates the cost and returns the state and cost, which is appended to a list. This list is passed to the fringe and sorted as per the cost and the process repeats till the goal state is reached or till the fringe is empty.

e) Solver: Calls other functions and solves 15 puzzle

<u>Types of heuristics tried:</u>

I tried 4 different type of heuristics

a) Manhattan distance including the adjustment for swap movement.

b) Row and column conflict

c) Diagonal Shortcut

d) Linear conflict

d) Misplaced Tiles

Out of all these heuristics Manhattan distance including the adjustment for swap movement gave the optimal output in reasonable time.

How my heuristic is admissible: The heuristic function used by me is a combination of Manhattan distance with modification for swap movement. For any tile in a given row if its goal state positions is in the same row and if the Manhattan distance of the tile form its current position to the goal position is 3, then it means that it is the edge tile and the total move required to reach the goal position for that particular tile is 1. Hence the Manhattan distance is modified to become 1 instead of 3.

Similarly For any tile in a given column if its goal state positions is in the same column and if the Manhattan distance of the tile form its current position to the goal position is 3, then it means that it is the edge tile and the total move required to reach the goal position for that particular tile is 1. Hence the Manhattan distance is modified to become 1 instead of 3. Note: We already know that Manhattan distance is admissible. Hence the modified Manhattan is also admissible as it never over estimates the number of move required to reach the goal state.

For all other cases the Manhattan distance calculation remains the same i.e $d = \sum_{i=1}^{n} |x_i - y_i|$

Where n is the number of variables, and ***X****i* and ***Y****i* are the values of the *i*th variable, at points ***X*** and ***Y*** respectively.

Problems: The main problem was with creating new copies from the old state. Python makes references the old ones and does not create new copies. This caused a lot of problem.

Assumptions: I am assuming that the number of edges are 12. That means that any tile on the edge row or column is eligible for swap when it is opposite to the blank tile.

References

http://www.math.ubc.ca/~cass/courses/m308-02b/projects/grant/fifteen.html

https://www.cs.bham.ac.uk/~mdr/teaching/modules04/java2/TilesSolvability.html

http://mathworld.wolfram.com/15Puzzle.html

http://www.geeksforgeeks.org/check-instance-15-puzzle-solvable/