CSCI 5511 ARTIFICIAL INTELLIGENCE 1

Prof. Nikolaos Papanikolopoulos

PROGRAMMING ASSIGNMENT

8-PUZZLE PROBLEM SOLVER

Manish Kumar Keshri

ID: 5417554

keshr005@umn.edu

Due Date: 12/07/2017

Problem Statement:

There exists a puzzle state which comprises of 8 numbers and an empty slot E randomly arranged. The goal state is to reach the puzzle state that will have the numbers arranged in order with the 0 coming after all other numbers have been arranged. For example, consider the initial puzzle state as (E 1 3 4 2 5 7 8 6), then the final or the goal state will be (1 2 3 4 5 6 7 8 E). We should be using A* algorithm to solve the 8-puzzle problem with the heuristic being the total number of misplaced tiles in the given puzzle state. The empty slot can move only once to generate the new puzzle state.

Problem Description

A* implementation with evaluation function given by,

f(n)= g(n)+h(n)

where,

g(node) = 0.1 + g(parent node)

h(node) = number of misplaced tiles in the given node

g(intial_node)=0

1. The input condition is taken as the first node.
2. Check for the feasibility of the puzzle by evaluating the initial state.
   Feasibility Check:
   1. Find the number of inversions in the given state (inversion is where the value of an array at i position is greater than value at j position and i<j and values at i and j positions are not empty)
   2. If odd number of inversions, it is not solvable
3. If solvable, push the first node in a queue.
4. A recursive function to pop out a node from the queue and check if it is the goal node or not. If it is not the goal node, we mark it as visited. If it is the goal node, program is terminated. A function is called to print the current node.
5. Move the empty spot in the puzzle in all possible direction and generate next nodes and push them in a queue.
6. Calculate heuristic and path cost so far for each node.
7. Sort the queue so that whenever a new node is popped out of the queue, it has smallest f value among all nodes in the queue. Quick sort is implemented so that the sorting is faster.

Program Functions:

solution: Function to initialise the first node and call functions for solvability check and solving the problem.

is_solvable: checks for the feasibility of the puzzle

find0: finds the location of empty spot in the given puzzle state, helps in feasibility check

inversion_count: finds the number of inversions in the given puzzle state, helps in feasibility check

sort_states: quick sort implementation to sort the states based on their f-value

partition: partition the array, helps in quick sort
puzzle: recursive function to solve the problem and generate next nodes
display: displays the contents of array of given node
move: checks if the generated node is visited or not, and if not visited push it into the queue
isvisited: to check if the given node is visited or not
calculate_h: to calculate the heuristic value for the given node

Program Variables:
all_state: to store all unvisited generated nodes
visited_states: to store the nodes that have been visited
initial_state: an array to store the initial condition
inv_count: to store number of inversion pairs

Coding Guidelines
1) Definitions of each variable
2) Definitions of each function
3) Comments describing each function and variable
6) Proper indentation for readability.

Files Included
1) 8puzzle.lisp : Main source code file
2) 8puzzle_ReadMe.txt : Includes the guidelines to run the source code
3) 8puzzle _Output.txt : Sample output for '#(E 1 3 4 2 5 7 8 6)

Deficiencies
1. Though quick sort implementation of sorting the queue is O(nlogn) in time complexity where n is the number of nodes, time can be significant large for larger size of puzzle because the nodes generated at each step will be huge in number.