CSL 302 – Artificial Intelligence

Project 1

This project is due on February 10, 2014. Late submission is not allowed without the prior approval of the instructor. Delayed submission will face exponential decrease in the points awarded. You are expected to follow the honor code for the course while working on this project.

Submission Instructions:

Email one zip file containing the following to ckn@iitrpr.ac.in.

1. Neatly formatted PDF document with your answers for each of the questions in the homework. You can use latex/MS Word or any other software to create the PDF.
2. For homework/projects that contain problems that require programming, include a separate folder named as "code" containing the scripts for the homework along with the necessary data files. Name the scripts using the problem number. Include a README file explaining how to execute the scripts.
3. Name the ZIP file using the following convention rollnumber_p(number).zip.
4. Please **do not** submit a hardcopy of your solutions.

In this assignment you will be experimenting with the different AI search techniques that we discussed in the class on the 8-puzzle problem. I am providing you with C code (p1.c) that implements a simple breadth first search. The existing code does not require any input. It first generates a goal state and applies a number of moves to reach a start state. Search is then initiated with the start state. It also prints the solution path, along with the moves that need to be performed to reach the goal state. I would like you to extend it to do the following:

1. In the current version of breadth first search, the algorithm adds a child of a node that is its parent to the search queue. For example, let the blank tile of the parent node be at (2,2). We perform the operation 'move blank tile left', to obtain a child with the blank tile at (1,2). Now if we perform 'move blank tile right' on this node, we go back to the parent node. Include the intelligence in the code to avoid such conditions.
2. Add the function AppendSearchQueueElementToFront that adds children of a node to the front of the search queue. Use this function implement the depth first search (DFS) function. Note that in DFS, you are likely to run into infinite loops you will have to modify the search algorithm and put a depth cutoff.
3. Add an integer variable h_val to the structure Node. Use the function HeuristicMisplacedTiles to populate the values for the nodes that are being generated. These correspond to the h values discussed in the class. Implement a function InsertSearchQueueElementPriorityh that inserts into the search queue, children of a node according to the h value. You will now add the function GBEFS that performs greedy best first search.

4. We will now add to the code the functionality for doing the classic A* search. Add a new integer variable f_val to the structure Node. This variable will store values of $f = g + h$. Implement a function InsertSearchQueueElementPriorityf that inserts into the search queue children of a node according to the f value. Now go ahead and implement the function AStar that performs A* search.
5. Finally, implement iterative deepening A* search function IDAStar. Note that for IDA*, the cut off values are based on f values.
6. Let us now run some experiments to study these algorithms. First we need to have some metrics to evaluate their performances. So add the functionality to record
    a. Total number of nodes expanded
    b. Total number of nodes generated
    c. Maximum depth of the search tree explored by the algorithm
    d. Maximum amount of Memory consumed
    e. Computation time.

    You will have to display the values of these metrics at the end of each search run. Tabulate/plot these values for the different search algorithms. Document your observations.

7. Now let's experiment with a different heuristic for A*. Implement the heuristic that computes the sum of Manhattan distances - HeuristicSumManhattan. Use this heuristic in conjunction with A* search. What is the impact of this heuristic on A*?
8. Vary the value of the variable SOLDEPTH that determines the actual depth of the solution. How does this impact the performance of the different algorithms?
9. Implement a new evaluation function $f = w * g + (1 - w) * h;\ w \in [0,1]$. When $w = 0.5$, this corresponds to the normal evaluation function. Consider the effect of higher and lower values of w on the peformance of the A* algorithm. You can use just the Manhattan distance heuristic, with solution depth (SOLDEPTH) varying from 3 to 10. Tabulate/plot the performance values for varying values of w.

If you prefer to implement the program in C++ or Python instead of C you can port the code to one of these languages. However, the program needs to be able to be compiled and run on Linux machines. Because some of the searches require significant amounts of memory and/or computation time, you might need to impose a limit on the amount of storage and/or time that is spent before the algorithm gives up. If you do this, mention these limits in your report as well.