Artificial Intelligence: HW 1 (v1.1)
Uninformed and Informed Search Methods
Due Feb 5, 2020

Overview

The learning objectives for this assignment are:

- familarize yourself with the Python programming language
- implement admissible heuristics for search problems
- utilize counting and timing mechanisms to compare heuristics and search methods

Introduction

Searching for solutions to a problem is a core component of AI. Search methods create a *frontier*, which consists of a set of states that have not been expanded, but could be via some *action*. Once an action is performed on a state in the frontier, these new states are then added to the frontier and the original node is removed.

Varying strategies dictate which state is select from the frontier for expansion. In this assignment, you will review some uninformed strategies, where states are removed from frontier without consideration to their cost and/or proximity to the goal state (where feasible) and informed strategies, those that consider goal proximity and cost. These informed algorithms will be the primary focus of this assignment.

Problem and Node Classes

You are provided the *cs444_search_template.py* file. This file contains the *Problem* class, which is an abstract class that defines a problem to be solved. The *Node* class represents a node in a search tree. With respect to the search tree, the search algorithms we are working with are only concern with two items:

- the path from a node in the tree back to the root
- the frontier

The search method will be responsible for tracking nodes in the frontier. The *Node* class/object maintains a pointer to each node's parent (and thus, can create the path back to the start state). Thus, there is no direct **tree** object that is maintained (with a root and pointers to children, etc).

Augment the Eight-Piece Puzzle Problem

You have been provided a class, *EightPuzzle*, representing the eight piece puzzle problem we discussed in class. In addition to the design of variable which maintains the problem state, the following methods have been completed for you:

- __init__
- find_blank_square
- goal_test
- check_solvability

As discussed in class, it is possible to create a placement of the tiles such that some solution states may not be obtainable. The method *check_solvability* determines if the goal state can be reached from the state passed to it. If you are randomly generating initial configurations, you should run this method so that you can tell if you will ever find a solution.

This class has a few incomplete methods that you need to complete:

- **actions** This method returns a list of the actions (UP, DOWN, LEFT, RIGHT) that are possible from the state passed as an argument.
- **result** This takes a state, applies the provided action, and then returns a new state. As stated in the comments, your code can assume that the state is valid (and that it will not move you off the "edge" of the board), since the search method will call *action* to get a list of legal actions prior to calling *result*.
- **h** A heuristic method that, given a node (and using the state within the node), determines the number of misplaced tiles and uses that as the heuristic (which is returned by the method).
- **h2** A heuristic method that works like *h*, except that the heuristic in this case is the sum of the manhattan distances to place each tile into its goal position.

Performance Investigation

Once you have completed and tested this class, you will test the performance difference between A* search with the 2 heuristics and in some cases the iterative deepening search (IDS) method. Methods for each of these search algorithms are provided, with the heuristic method being the 2nd argument to the A* method. You should generate 10 different solvable starting configurations at random. For each of these starting configurations, tracking the wall clock time it takes to run the A* method with the *h* and *h2* heuristics (separately) . If the solution depth is less than 20 (you can check the solution returns after A* is called), attempt to call IDS. In my experience, when the solution is deeper than depth 20, IDS takes more than 30 minutes to run (and thus the reason to limit this to solutions with depth < 20).

In a report (PDF), show these 20+ executions (10 times for each A* method (20) and potentially 10 more for IDS). See the rubrics for a list of what needs to be submitted with this assignment.

Collaboration Policy and Package Usage

For this project, you are only allow to use the following python packages contained in the template file plus time, numpy and matplotlib if you choose. If you need more than these packages, please see me prior to using them in your assignment.

If you need more than these packages, please **request** them from me (as maybe I omitted a few here).

For this assignment, you must work **individually**. Helping colleagues to find a bug or to help them understand some concepts is OK, sharing of code or obtaining code that you did not write is prohibited. You should be able to explain all of your code to me, as failure to do so will be an indicating that you over-collaborated. I will be using plagiarism detection tools in this class.

If you have questions about this policy, please see me.

Deliverables

The following deliverables must be placed in a single zip file named PA1Canvas.zip and uploaded to canvas. This zip file must include the following:

1. Your python code in a file named *cs444_PA1.py*. Make sure that your name appears at the top of the file.
2. Sample output from your program (you can simply run your program and redirect the output to a file). Make sure this sample output show the start state to a problem, the move sequence of the solution, and the goal state. If you print more than one problem, please add some whitespace.
3. A PDF showing the results of the search techniques (timing of each method). These results should be shown in a 10 by 3 table, and if you wish, you can additionally add a plot.

Rubrics

The following is the breakdown how this assignment will be graded:

Item	Description	Points
Readability/Style	Program is commented, following naming style for either Python/Java. Points will be deducted for poor variable names or unreadable code	5
action states	are determined correctly	15
result method	result state is returned correctly	15
h heuristic	implemented correctly	20
h2 heuristic	implemented correctly	20
Eval searches	calling each search method (A* with h, A* with h2, and IDS as required) is performed correctly (10 different puzzles) with resulting solutions printed and moves shown in output, and the moves are correct	15
PDF report	shows a table comparing the performance of the methods	10