Assignment 3

Anytime Search Algorithms

Important!

The work done on this assignment must be your own work. Think carefully about any problems you come across, and try to solve them yourself before you ask anyone else for help. Under no circumstances should you work together with another student on any code ***you or they*** write for the assignment. You are expected to modify the Prolog code that is provided.

However, it is allowable and even encouraged for you to discuss what the provided Prolog code does and how it does it. The class forum would be the most appropriate place for this discussion to take place so that all the class can participate and learn :^)

It is allowable and even encouraged to discuss, on the class forum, exactly what is being expected for this assignment.

Due: *11pm Thursday 4th June*

Worth: *10% of final grade*

Introduction

In class, we have seen that we can combine algorithms where each algorithm has strengths that complement the other's weaknesses to obtain an algorithm that combines those strengths. Specifically, we have seen how we can combine Depth-First search (which has a linear space complexity but is incomplete and non-optimal) and Breadth-First search (which is complete and optimal but has an exponential space-complexity) and a little "magic" (the iterative deepening component) to produce Iterative Deepening, which is complete, optimal, and has a linear space complexity.

In this assignment we are going to do something similar. Informed search algorithms usually search significantly smaller portions of a problem's search space than their corresponding uninformed search counterparts. The extent to which the search space can be reduced depends upon the quality of the information given to the algorithm. In this course, this information is in terms of estimates of how far a state is from its nearest goal state.

Even with "reasonable" heuristic evaluation functions, the search spaces explored by informed search algorithm can be immense. Sometimes, we are willing to accept a less than optimal solution if we can get it sooner. It turns out that we can, in effect, unify Greedy Best-First search with A* search by adding a weight to formula for ***f(n)***, specifically ***f(n) = w * g(n) + h(n)***. If ***w = 0*** then we have Greedy Best-First search, and

if ***w = 1*** then we have A* search. This weight will only vary between zero and one and its reciprocal directly determines the upper bound on how "bad" the solution can be. In other words, if ***w = .5*** then using A* with this f-formula will produce a solution that is no worse than twice the length of the optimal solution.

Unfortunately, there is no obvious relationship between the weight and how much faster the search will be. In general, there is a tendency for the search to go faster (because fewer nodes are expanded) as the weight decreases but there are no guarantees.

Goal

Why should decreasing the weight cause fewer nodes to be expanded? In particular, why does Greedy Best-First search work? For example, if the heuristic evaluation function simply produced values at random, we would assume that decreasing the weight of the ***g(n)*** would not make the search better. What is it about the heuristics' relationship to the search space that enables weighting to decrease the amount of search needed to find a solution? In particular, what is the relationship between the heuristic and the solutions they guide the search algorithm towards? So far, there have been some explanations conjectured, but, as far as I know, no one has yet come up with the full answer.

Unfortunately, it is unlikely that you will make much progress on answering these questions during this assignment. You would probably need a lot more time than you have available for this assignment. However, you can make a start. The first thing one needs to do is to build the software that can help you establish that the phenomenon exists. That is what you will do for this assignment.

Tasks

You will be given the Prolog code files for:

- [*aStar.pl*] A* search algorithm.
- [*heuristics.pl*] The out-of-place and the manhattan distance heuristics.
- [*eight.pl*] The 8-puzzle domain.
- [*problems.pl*] A database of 8-puzzle problems.

You will need to write code to:

1. [***weight** extension*] Extend the search algorithm to weight the ***g(n)*** factor in the ***f(n)*** formula (this includes adding a new input argument to the ***search*** predicate).
2. [*Inconsistency handling extension*] Extend the search algorithm to handle inconsistent heuristics (so that when it hits a state again, it will only retain information (e.g., the **g** value and **parent** state) for the shorter path to that state).
3. [*Extra control level extension*] Add an extra control level on top of the search algorithm that will automatically run the search algorithm for a range of weights, ranging from 0 to 1 in increments of .1.
4. [*Instrumentation extension*] Instrument the extended algorithm to capture and report the relevant data.

A copy of the desired output, for the sample problems, will be put up on the assignment's web page in a few days.

What you will be turning in

Marking Guide

There will be ten marks for this assignment. These will be divided up as follows:

1. 2 marks for correctly working weight extension.
2. 3 marks for correctly working inconsistency-handling extension.
3. 3 marks for correctly working extra control level extension.
4. 2 marks for correctly working instrumentation extension.

Submission

You will be submitting your code for your version of the search algorithm, which should be called **"***weightedAStar.pl***"**, via the ADB.