Assigned: Tuesday 13 September 2011

Due: Monday 19 September 2011, 11:59 p.m.

Objectives:

- Understand details of of informed search algorithm implementations
- Explore performance implications of uninformed versus informed search algorithms
- Learn how to develop admissible heuristics
- Reinforce principles of good science writing

Collaboration: This laboratory will be completed in pairs assigned by the instructor.

1 Introduction

In the last assignment, you implemented a variety of tree-search based uninformed search algorithms, measuring their efficiency and solution quality. In this assignment, we'll expand our horizons (nodes? frontiers?) and use information about the problem to implement A^*, an informed search routine.

2 Code and environment

The infrastructure code and problem for this assignment are the same as the last. In addition, an optional heuristic for the lights-out puzzle, `(lights-out-heuristic state)`, is provided with the other compiled files of the prior assignment. To use it, copy the `compiled` directory to wherever you are working and include the following command in your Scheme file

```
(require "lightsout-heuristic.scm")
```

Even though there is no file by this name, the Scheme environment will find the compiled version in the `compiled` subdirectory.

3 Lab assignment

Problem 1 - Implementing an A^* search algorithm [10 points]

Implement and document the A^* search routine `(a-star-search start-state problem heuristic)`. You should use the general tree search routine `search` from the previous assignment, which may be either your implementation or the compiled version provided.

Your solution may be based closely on prior specific search routine implementations, so long as they are properly attributed.

Problem 2 - Designing a heuristic [40 points]

Design the best heuristic for the lights out problem you can think of. This should be neither the trivial always-zero heuristic nor a perfect heuristic that simply solves the problem using full breadth-first search. You should strive to come up with an admissible heuristic, even if you do not end up implementing it. If you cannot implement your supposedly admissible heuristic, you should implement a simpler one that is not admissible but may outperform the other searches for efficiency (node expansions).

Part A

Write a clear, unambiguous description of your best heuristic in a mixure of high-level English and pseudo-code. Your description ought to allow a skilled programmer to create a working implementation.

Part B

Prove, or at least explain, why your heuristic is or is not admissible.

Part C

Implement the best heuristic you can. You should strive to implement your best heuristic from Part A, but if you do not complete it, a simpler, perhaps non-admissible heuristic can be given in its stead. Your write up should clearly indicate what you have done.

Problem 3 - Analysis [50 points]

In this problem, you will do some comparative analysis of your search routines.

Note that, while `random-eight-puzzle-state` and `random-lights-out-state` call `random` to produce their states, you can make your results repeatable by using the procedure `(random-seed` *`seed`*`)` to set the seed of the random number generator.

Part A

Generate a fairly easy eight-puzzle state and an easy lights-out state (or, even better, use the states from the previous assignment). Run all the search algorithms (breadth-first, depth-first, depth-limited, iterative-deepening, uniform-cost, and A^*) on both problems, creating two tables (one for each problem) listing the number of nodes expanded to find a solution, and the total number of actions in the solution. Be sure to specify the specifics of each problem or how they were generated.

Part B

Generate the hardest problems you feel like waiting for solutions to under most search algorithms. Run your search algorithms again, adding rows to the tables you created above. Be sure to specify the specifics of each problem or how they were generated.

Part C

Construct a table similar to the one in AIMA Figure 3.29, comparing IDS and A^* with progressively more difficult lights out problems (fix the size of the grid to one value). You should include at least six rows in your table.

You can calculate the effective branching factor `b` in Maple (interactive mathematical software available on the MathLAN via the shell command `maple`) with the command:

```
fsolve( (b^d-1)*b/(b-1)=N,b);
```

where `d` is your solution depth and `N` is the number of nodes expanded.

Part D

Using the data you have generated, draw some conclusions about the relative efficiency and effectiveness of the search algorithms on these problems.

- How does A^* compare to the other search algorithms?
- If you have multiple heuristics, how do they compare to one another? To the one provided? Why?

Note that a complete analysis will feature coherent paragraphs, a brief introduction stating the purpose and context, as well as your overall conclusions. It should be nicely formatted and feature a logical organization, complete sentences, and proper grammar, spelling, and punctuation. The audience is your peers in this class; they do not know *a priori* what problems you examined, nor what your results or conclusions may be.

What to turn in

Your submission should include the following

- Your completed implementation of `a-star-search` and a lights-out-heuristic in a `.scm` file
- A short driver program that demonstrates your heuristic applied to several meaningful states and your search algorithm and applied to a simple problem (i.e., one that does not take long to run).
- A single PDF containing (merged)
  - Your Scheme file(s)
  - A transcript of your driver program's output
  - Your analysis