Planning Search Project

In this project, you will be investigating some modifications of the the search algorithms discussed in Chapter 9 of the textbook. This assignment has 3 interrelated goals. You will be learning about some more sophisticated search algorithms by implementing them. In the process you will also be developing your intuition for the "size" of planning problems that can be approached using these techniques. Finally, by implementing your solutions in Prolog, you will continue to develop your skills for formalizing ideas in logic.

Starter Code

In the handouts/search_project folder you will find the following files:

`search_proj.pdf` This project overview.

`id_plan.pl` This file contains a slightly modified version of the general planner from the text. `id_plan/1` and `id_plan/2` are the top-level predicates for generating plans through iterative deepening search.

`ida_plan.pl` This file contains a modification of the planner to use a heuristic function to guide the search by pruning the search tree. The main top-level predicate is `ida_plan/3`.

`misscan.pl` Missionaries and cannibals as a general search problem.

`eightpuzzle.pl` The 8-puzzle as a general search problem.

`time.pl` Definition of some useful predicates for timing searches. The main workhorse here is `cpu_time(Goal, Time)` that can be used to get the running-time of a goal.

`sets.pl` This module defines a handy and efficient data structure for sets.

Tasks

1. Experiment with `id_plan` on the eight puzzle problem to get a sense for the size of problem that is solvable. Create a table that keeps track of the time to solve various problem instances on your system.

2. Add cycle-avoidance. Finish the implementation of the graph search planner `idg_plan/2` by implementing the `reachable/4` predicate. Use this new version of the planner to update your your table from part 1. Does this allow you to solve more complex puzzles?

3. Implement some search heuristics. Study the code for `ida_plan` to see how heuristic predicates are used.

a. Start by running the `ida_plan` predicate using the `h0` heuristic which just produces an (uninformed) iterative deepening search. Compare the results of this to the results in part 1.

b. Implement an `h1` predicate that computes the number of tiles (not including the blank space) that are out of place. Use this heuristic to do some timing tests to see whether it improves on previous results.

c. Implement an `h2` predicate that computes the total Manhattan distance over the misplaced tiles (see discussion in section 9.3.2). Use this heuristic to compare to previous results.

4. Add cycle-avoidance to the ida planner. Following the model of what you did for `idg_plan` write an `idag_plan` predicate and evaluate its performance using `h2`.

5. Write a short (1 or 2 paragraph) summary of your results. In this written report be sure to include the table of timings that you have run.