Take Home Midterm – Winter 2010

Prof. Kerstin Voigt

The total number of available points is 75. The midterm is due on Thursday, February 25, beginning of lecture.

Please select and solve **3 out 4** of the following exam problems. Each problem is worth 25 points. Each involves programming in Lisp.

The problem options are:

1. **Random restart in hillclimbing:** Add random restart as a measure to escape from local maxima (or minima) that are possible with the basic hillclimbing implementation in file `hillclimb.lisp`.
2. **Multiple solutions with graph-search:** Enhance the graph-search implementation in `gs.lisp` so that function `itgs` produces not just one, but up to 5 alternative solutions.
3. **Generate the longest solution path with graph-search:** Modify the graph-search implementation in `gs.lisp` so that solution returned is the one along the **longest** (cycle-free) path.
4. **Successor function for the "water-jug" problem:** On page 137 of Nilsson, you find a description of the "water-jug" problem. Implement a **successor function** that will produce all possible next states given a water-jug state.

Details on Option 1. Obtain a copy of file `hillclimb.lisp` from ∼`voigt/cs512`. For testing purposes, also obtain a copy of `puzz8.lisp`. Enhance the hillclimber with the ability to escape from a local maximum/minimum in the following manner: when a state is reach that is not a solution and has no successor better than itself, randomly generate a new state and restart hillclimbing. Set a limit to how many restarts you allow (say, 25). When the maximal number of restarts is not sufficient to solve the problem, let the hillclimber fail (with informative message).

Demonstrate the operation of your hillclimber with problems in the domain of the "8-puzzle". Remember that our current basic hillclimbing implementation does not perform well with `puzz8.lisp`. Let's see whether random restarting can help. (There is not much use working with `twocolor.cpp`, since hillclimbing was very successful in this particular domain.

Details on Option 2.: Work with files `gs.lisp` and `puzz8.lisp` from your instructors cs512 directory. Focus on function `itgs`. Consider adding another parameter that indicates the number of solutions (here: same solution, but along different paths), that are to be produced.

```
(defun itgs (state nosolns &optional ...)
  ...
)
```

Typically, when the first solutions is found, list `OPEN` is not empty. It contains other nodes which may give rise to alternative paths to reach the goal state. After outputting the first (or, current) solution, one can continue to graph-search with the current list `OPEN` and produce up to `nosolns` many alternatives. In cases where `nosolns` is instantiated with a number that is larger then number of alternatives, `OPEN` will eventually be empty, and in this case, graph-search will terminate as usual.

Details on Option 3.: Work with files `gs.lisp` and `simplegraph.lisp` from your instructor's cs512 directory. We know that running graph-search as breadth-first search will produce the solution along the shortest path. It is <u>not</u> true in general, that depth-first search will produce the path along the longest path. For this, more work is needed ... Enhance function `itgs` so that it will generate all possible solutions paths (i.e., run until `OPEN` is empty), and keep track of the longest solution path encountered. Let `itgs` return with this longest path.

You may assume that you are working with a domain that is finite. Thus, there will be a finite number of possible paths to a solution. The domain in `simplegraph.lisp` has this property.

Details on Option 4.: The water-jug problem involves two jugs, one that holds 3 units (jug3), and one that holds 4 units (jug4). The following actions are possible: (1) fill jug3 to capacity, (2) fill jug4 to capacity, (3) dump all of jug3, (4) dump all of jug4, (5) pour as much as possible from jug3 into jug4, with possible remainder in jug3, and (6) pour as much as possible from jug4 to jug3, with possible remainderin jug4.

The typical water-jug problem is stated as: jug3 and jug4 are empty; what sequence of actions will result in jug4 containing 2 units (jug3 may contain any amount)?

In this option, you are not required to solve the water-jug problem. You are only asked to implement in Lisp the successor function that could be used if we wanted to let graph-search solve the problem. The water-jug successor function should produce all possible next states, given incoming water-jug state. For example, if `(2 3)` were the state where jug3 contained 2 units, and jug4 contained 3 units, a call to the successor function should produce the following list of next states:

```
(succ-fct '(2 3))  --> ((0 3) (2 0) (3 3) (2 4) (1 4) (3 2))
```

Work out some other examples of input/output behavior in order to make sure you are aware of all the possibilities.

Submitting your Midterm: For each of your three options, submit the following by the duedate:

1. Hardcopy of the .lisp source code.
2. Hardcopy of a typescript that demonstrate successful loading of your program, and 3 representative test runs. Edit out excessive text that may show intermediate program activity, but under no circumstances edit your "solution" ...
3. Softcopy of your .lisp source code, by email to `kvoigt@csusb.edu` with subject line "CS512 MIDERM".