Massachusetts Institute of Technology

6.034 Artificial Intelligence

Assignment # **6** Due: **20/Oct/98** 6.034 Item # **10**

PLEASE WRITE YOUR TA'S NAME ON YOUR PAPER.

Goals

This assignment is intended to familiarize you with incremental constraint satisfaction methods. These methods are quite different from the backtracking schemes discussed in Problem Set 5.

Code

The files mentioned below can be found in `/mit/6.034/ps6`. You will need the same files you used in PS 5 as well as an additional file `min-conflicts.scm`. They are all loaded by loading `load.scm`.

Background

The file `min-conflicts.scm` implements a (sort of) hill-climbing (without backup) strategy for CSP problems. This code starts with a complete assignment to the variables, e.g. a random one, and then repairs it (that is, tries to reduce conflicts). During each cycle, it selects a variable involved in a conflict and assigns it the value from its domain that minimizes the total conflicts (ties are broken randomly).

Problem 1

Run the code a few times on a very simple example, e.g. coloring the `*trivial-map*` in `examples.scm` using four colors.

```
(set! *verbose* #t)
(test-map-4-coloring
  (lambda ()
    ;; the 100 indicates the max number of cycles of improvement
    (min-conflict-hill-climb 100 (random-assignment)))
  *trivial-map*)
```

Pick a run that converged quickly and explain in a few words each of the steps that the algorithm took.

Problem 2

Now, let's run the code 20 times on a simple line-labeling problem, the `*cube-diagram*` you looked at in PS 5.

```
(begin
  (set! *verbose* #f)
  (do ((i 0 (+ i 1)))
      ((= i 20))
    (test-line-labeling
     (lambda ()
       ;; Note we have allowed 500 cycles.
       (min-conflict-hill-climb 500 (random-assignment)))
     *cube-diagram*))
  )
```

You should have seen that this often fails to find a consistent answer. Note, that this is a very simple problem for any of the backtracking ethods. What do you think might be happening when it fails? Hint: If you re-run `(test-line-labeling ...)` using `*verbose* = #t`, this might help you see what's happening. Warning: This generates a lot of output.

Problem 3

The code we have provided you supports a mechanism for attempting to avoid the failures you observed in the last problem. The method is controlled by `*use-conflict-weight*`. When this variable if `#t`, the code keeps track of situations when it is "stuck", that is, when the number of conflicts does not decrease in response to a change in a variable. Then, it increments a "weight" associated with the violated arcs for the changed variable. Initially all weights are equal to one. When `*use-conflict-weight*` is on, the weights of some arcs will rise if they are repeatedly involved in failed attempts at resolving conflicts.

If you do `(set! *use-conflict-weight* #t)` before running the `*cube-diagram*` test we ran above, then you'll see that it is more likely to find an answer. Explain briefly why this might be the case.

Problem 4

For the examples below, we ran `min-conflicts-hill-climb` twenty times for a maximum of 750 cycles each. We had `*use-conflict-weight* = #t`. We report the average number of arc tests for the succesful runs and the number of successful runs out of twenty tries (`x/20`). Remember that the initial assignment is random, so one expects substantial variation in the performance from run to run.

Problem	`backtrack-fc-dynamic`	`min-conflicts-hill-climb`
cube line-labeling	136	409 (17/20)
tower line-labeling	370	7,125 (20/20)
4 color map-coloring	1,385	4,128 (20/20)
reversed course-schedule	127,143	340,011 (16/20)
10 queens	2,346	16,098 (20/20)
20 queens	35,025	46,903 (20/20)
50 queens	1,081,158	471,512 (20/20)

As you can see, the performance of the modified `min-conflicts-hill-climb` strategy is respectable (except for the occasional failures). On pretty much every problem, except the line-labeling problems, hill climbing is much better than straight backtracking (and even than backtracking with forward checking) – see data in PS 5. Even compared to `backtrack-fc-dynamic`, which was the best method from PS 5, hill climbing does reasonably well (and will do better as the problems get bigger).

Part A

There are two surprises here: (1) how well `min-conflict-hill-climb` works on relatively hard problems and (2) how poorly it works on some very simple problems. Speculate on what might make some problems easy or hard for the hill-climbing strategy.

Part B

Which of the following two methods works better at reducing the failure rate we noted in the data above,e.g. for labeling `*cube-diagram*`:

1. Increase the number of cycles in the call to `min-conflict-hill-climb`.
2. Call `min-conflict-hill-climb` multiple times (with different random initial assignments).

Can you explain why this may be so?

Part C

This is optional. The performance of the hill-climbing procedure depends drastically on the initial assignments. In all the examples above, we used a random assignment. For any problem, one can generally construct a simple method of doing the assignments that comes much closer to being a correct assignment (in terms of number of violated constraints) than would a random one. Try to construct such a procedure for one of the problems given and document the improved performance.