Massachusetts Institute of Technology

6.038 Artificial Intelligence in Practice

Problem Set 3: Thursday, February 19, 1998
DUE: Tuesday, February 24, 1998

NB: For this assignment, we encourage (but do not require) you to work together in pairs and hand in a single solution per pair.

1 Background

This assignment is a direct sequel to the 6.034 problem set on constraint satisfaction problems (CSPs). In fact, the 6.034 problem set (PS 5) from the Fall is attached. However, in this assignment, you will focus on solving one example problem more completely.

2 Files

You will need to run Scheme to do this assignment. You will be using the Scheme code for that problem set (found in `/mit/6.034/ps5/` on Athena. If you are using Scheme-to-C (see below), use the (functionally identical) versions on `/mit/6.038/ps3/`.

If you have a PC (running Linux or Windows) you can install a version of Scheme that is available from `www-swiss.ai.mit.edu/scheme-home.html`. This is supposed to be the best version of MIT Scheme (aside from that on the 6.001 HP machines).

Otherwise, you can use the versions of MIT Scheme available on Athena (which you used in 6.034); the Sparc and SGI versions are known to be a bit flakey (and slow). Another choice, which I personally prefer, is to use Scheme-to-C. This doesn't have as many features as MIT Scheme and will require some initial getting used to, on the other hand it is small, fast, integrates relatively smoothly with C code and has a working compiler (actually just a front-end to the C compiler). It does not currently run on SGI's, but does run on Sparcs and (most) Linux machines. You have access to the interpreter (`sci-6.038`) and compiler (`scc`) when you do `add 6.038` and `add 6.034` (some of the binaries are links to 6.034). This version of the interpreter has some extra definitions pre-loaded to emulate some MIT Scheme functions used in the 6.034 code. To compile your own code, you should compile with the file `/mit/6.038/ps3/useful.sc` to get these extra definitions. For example, to produce a version of the interpreter that has ALL the CSP code compiled, you can do

```
add 6.034
add gcc
cd <mydir>
cp /mit/6.038/ps3/*.sc .
<<type the following on one line>>
scc -i -o ps3-sci useful.sc constraints.sc backtrack-dynamic.sc
          examples.sc min-conflicts.sc
```

In fact, you will find `ps3-sci` in the `bin` directories for (most) Linux machines and Sparcs. To include your additions, just add more file names to the command line.

By the way, Scheme-to-C prefers files with extension `sc` whereas MIT Scheme prefers `scm`. **If you use Scheme-to-C, DO NOT use the files in `/mit/6.034/ps5`, use the ones in** `/mit/6.038/ps3/`. We have a hardcopy handout with some documentation for Scheme-to-C, available from Jacqui in NE43-836. You can also find the Postscript files for these handouts on the 6.038 FTP site (in `Scheme-to-C-docs`).

3 Assignment

The basic objective in this assignment is to write a program that can construct a five year course schedule that satisfies all the MIT requirements plus the Course 6-3 MEng requirements. To save you some research work, the file `/mit/6.038/ps3/MEng` has some data that summarizes information on many potentially relevant courses, plus summarizing the MEng requirements as I understand them (this is not necessarily accurate or complete). I believe it is possible to satisfy the requirements listed there by choosing a schedule of courses from the given list. However, feel free to add your favorite courses if they are not listed. Note that we want to be able to tackle the problem of building a schedule for someone who has already taken some courses, e.g. for a sophomore; keep that in mind when choosing an implementation.

Note that this problem is different from the 6.034 assignment in that you have to pick the courses as well as schedule them.

Problem 1: Choose the variables, domains for the variables and constraints to represent this scheduling problem. Hand in an explanation of your choice as well as the code and data required to implement them. The file `examples.scm` has the implementation for the three examples used in 6.034. Some of the requirements may not be readily encodable in the constraints supported by the code; show how to handle these constraints in some other manner.

Problem 2: Pick an algorithm to solve the schheduling problems and implement it (try to base it mostly on existing code). Explain your choice and show a sample solution to the problem. Give some information on running time, number of tests, etc. If you try more than one approach, give the data on those. Hand in any code you wrote (or modified) for this; do not hand in code you did not touch.

Problem 3: Explain how you would go about representing and implementing preferences (as opposed to hard constraints) in this problem. For example, a preference for taking some course in the Spring rather than Fall or a preference for taking a particular course to satisfy a concentration requirement or a preference for not taking more than two Course 6 courses in one term. Keep in mind these preferences may not lead to a consistent schedule so they may have to be violated. You do not need to implement your approach but you may want to "hand-simulate" it to make sure it works. Indicate any potential problems with your scheme.

Problem 4: Could you have attacked this problem using a search algorithm such as A^*? Sketch how you might do this. Explain the advantages and disadvantages of that approach, when compared to the one based on CSPs.