University of British Columbia, CPSC 322, Winter 2009 Assignment on Constraint Satisfaction Problems

Giuseppe Carenini, David Poole and the CPSC322 Team
(Kevin Leyton-Brown. Mark Crowley, Erik P. Zawadzki, David R.M. Thompson, Alan Mackworth)

Total Points 115 + (10 bonus points)

Read the assignment carefully. If anything is unclear/ambiguous, please post your questions on the WebCT discussion board for this assignment. We'll be happy to help you as much as possible.

1. **[20 points] Grid Coloring Problem**

   You are given a 2x3 grid (two rows 0,1 and three columns 0,1,2). You need to paint each cell so that each cell is painted differently from the cells it shares an edge with. You can use only three colors: Blue Red and Green. Also you are told that the middle-bottom cell should be Green and the middle-top one should be Blue.

   (a) **[5 points]**

   - Navigate to the webpage http://www.aispace.org/, click on 'Main Tools' then click on ' CSP Consistency Based CSP Solver' applet.
   - If you are not familiar with the applet read the tutorials.
   - Run the CSP applet.

   Build the constraint network for the given problem. Remember, you need to specify variables, domains, and constraints.

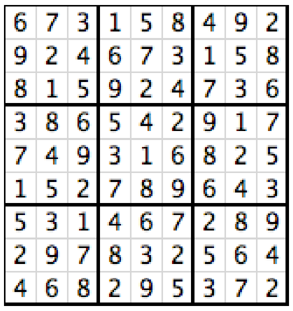
   (b) **[10 points]** Run the arc consistency (AC) algorithm on the network (using fine step). What was the first inconsistent arc found by AC? how was it made consistent? What was the second one? How was it made consistent? Can AC solve this problem? If this is the case provide the solution, otherwise verify (with pencil and paper) whether a solution can be found by applying DFS-with-pruning to the output of AC. You can use the simplified notation for states (see lecture 12), to show your work.

   (c) **[5 points]** Assume you are given the additional constraint that the cells in the top-right and left-bottom corners must be painted in the same color. Would AC find a solution to this more constrained problem? Why?

   The code part of the following questions can be done alone or working with one other person. We **strongly recommend working in partners, as pair programming can be a good learning experience and can expedite your project development. Both students have to submit the code by handin. Each student should answer the written parts individually**

2. **[35 points] Sudoku**

   Sudoku is a game played on a square grid of 9 large blocks, each containing its own 3x3 grid of cells. Here is an example:



6	7	3	1	5	8	4	9	2
9	2	4	6	7	3	1	5	8
8	1	5	9	2	4	7	3	6
3	8	6	5	4	2	9	1	7
7	4	9	3	1	6	8	2	5
1	5	2	7	8	9	6	4	3
5	3	1	4	6	7	2	8	9
2	9	7	8	3	2	5	6	4
4	6	8	2	9	5	3	7	2

The rules are simple:

- each cell contains a number from 1-9
- the number in a given cell (i, j) cannot match the number in any other cell in row i or column j
- each block can only contain one entry of each number from 1-9

Some cells are initially filled in and the goal is to find the unique assignment of numbers to the remaining cells that satisfies the above rules.

This game can be seen a straightforward system of constraints.

(a) [**7 points**] Design the CSP you will use to solve the Sudoku problem. Describe the variables, their domains and the constraints needed. We encourage you to use only binary constraints, it will make the implementation easier. You do not need to list all of them for this sub-question! Just specify all the constraints involving one variable, let's say the one in the center of the board.

(b) [**28 points**] Implement the Arc Consistency algorithm with Domain Splitting in Java and apply it to solve a game of Sudoku as a constraint satisfaction problem. We have provided a few utility classes and a class to be implemented as described below.

Download the file `sudoku.zip` (from WebCT Vista), it contains the following files:

- **SudokuUtil.java**: This is a class that handles reading and writing the Sudoku boards to text files for you. The board is represented as a two dimensional array of integers.
- **SudokuTester.java**: This is a test script that will be used to test your code. You may want to modify it for testing/debugging of your implementation, but make sure that the code you submit works correctly with an unmodified copy of SudokuTester.
- **SudokuSolver.java**: This is the class where you put your implementation. Please, make sure your code is well commented and easy to read. Points will be deducted if this is not the case.
- ***.sud** and ***Solution.sud**: several sample Sudoku boards with solutions that will be used to test your algorithm. They are described in detail in SudokuTester.java.

For this question, you can assume that the given Sudoku board is valid, that is, it has exactly one solution. When testing, keep in mind that some Sudoku boards are not solvable with arc consistency alone.

A straightforward implementation is about 200 lines long. If you need many more lines of code, maybe you are not on the right path.

SudokuTester.java includes time estimates for how long it should take to solve the given sample Sudoku boards. If your implementation needs significantly more time to solve these instances it is probably not correct.

To be submitted (by handin): **a single file SudokuSolver.java**. If you decide to use additional classes, implement them as inner classes inside the SudokuSolver class.

(c) Bonus [**5 points**] Modify SudokuSolver.java so that it throws an exception when given an invalid Sudoku board. An invalid Sudoku board has no solutions or has more than one solution. Note that such modified implementation may take significantly longer to solve valid boards.

To be submitted (by handin): **a single file RobustSudokuSolver.java**. If you decide to use additional classes, implement them as inner classes inside the Robust-SudokuSolver class.

3. **[25 points] Allocating Developments Problem**

CSP techniques are useful in solving complex configuration and allocation problems. You are given the task of allocating four developments in a new site in Whistler. You have to place a housing complex, a big hotel, a recreational area and a garbage dump. The area for development can be represented as 3x3 grid (three rows 0,1,2 and three columns 0,1,2) and you need to place each development in one cell of the grid. Unfortunately there are some practical constraints on the problem that you need to take into account. In the following, A is close to B if A is in a cell that shares and edge with B.

- There is a cemetery in cell 0,0.
- There is a lake in cell 1,2.
- The housing complex and the big hotel should not be close to the cemetery.
- The recreational area should be close to the lake.
- The housing complex and the big hotel should be close to the recreational area.
- The housing complex and the big hotel should not be close to the garbage dump.

(a) [**15 points**] Represent this problem as a CSP. Be as precise as you can in specifying the constraints. Also do not forget some basic constraints that are inherent in allocating objects in space but are not listed above.

(b) Apply the Arc Consistency algorithm with Domain Splitting you have developed for the previous question to solve this CSP.

To be submitted (by handin): Your representation of the given problem as a CSP and the solution(s) generated by the Arc Consistency algorithm with Domain Splitting.

4. **[35 points] Stochastic Local Search (SLS) for Scheduling**

Consider the following problem of scheduling university exams, in which:

There are a fixed number of dates (five) and timeslots per day (four) when exams can be run. There are a limited number of rooms that the exams can run in (though this is variable). An exam slot specifies a date, time and room. No two exams can be run in the same exam slot. There are a variety of students taking different combinations of courses. No student can be in two rooms at once.

This can be represented as a CSP as follows:

CSP feature	Representation
Variables	There is one variable per exam.
Domains	The domain of each variable is the set of exam slots.
Constraints	There is a constraint between every pair of variables, that no two take the same value. There are also a number of constraints between pairs of variables that the two exams can't have simultaneous exam slots if any student has to attend both.

You will be provided with code that can generate instances of this scheduling problem. Your task will be to:

(a) **[20 points]** Implement two SLS algorithms for solving instances of this scheduling problem. You can choose from algorithms covered in class. The scoring function that should be used by your algorithms is provided (see below).

(b) **[15 points]** Using a variety of test cases (you should generate as many as you need), plot the performance of these two algorithms (experimenting with different values for at least one tunable parameter that each algorithm has). Which algorithm and parameter values do you think are the best? why? (Remember that SLS algorithms are not guaranteed to terminate, neither to find the optimal solution)

Download the file `scheduler.zip` (from WebCT Vista), it contains the following files:

- **SchedulingProblem.java**: this class stores a scheduling problem.
- **Student.java, Room.java, Course.java**: components of the scheduling problem.
- **Generator.java**: generates a random instance of the scheduling problem.
- **ScheduleChoice.java**: This class stores the scheduling information for one exam (i.e., an exam slot). A possible world of a particular instance of the scheduling problem is an array of these with a ScheduleChoice for every exam.
- **Evaluator.java**: A scoring function that scores a schedule (i.e., a possible world of a particular instance of the scheduling problem). You should use it to guide your local search algorithms. We will use it to asses the quality of your solutions. It penalizes a lot schedules that schedule two exams at the same room at the same time and applies a smaller penalty for each student that is scheduled to be in two exams at the same time.
- **Driver.java**: A test driver that we will use to grade your code. You may want to modify it for testing/debugging of your implementation, but make sure that the code you submit works correctly with an unmodified copy of Driver.java.
- **Scheduler.java**: scheduler interface that your algorithms have to implement.
- **Scheduler1.java,Scheduler2.java**: stubs for your code. Please, make sure your code is well commented and easy to read. Points will be deducted if this is not the case.

Note that the only files you should modify are Scheduler1.java and Scheduler2.java. Adjust the number of iterations for both algorithms so they terminate after 15-30s.

To be submitted (by handin): **two files Scheduler1.java, Scheduler2.java**. If you decide to use additional classes, implement them as inner classes inside the Scheduler1 and Scheduler2 classes.

5. Bonus [**5 points**] Come up with a better scoring function and implement it in class MyEvaluator. Explain the new scoring function and describe how it changed the performance of your search algorithms.

   To be submitted (by handin): **a single file MyEvaluator.java**. If you decide to use additional classes, implement them as inner classes inside the MyEvaluator class.