School of Computer Science
University of St Andrews
2013-14
CS4402
Constraint Programming
Practical 2: Solving

This Practical comprises 50% of the practical component of CS4402. It is due on Tuesday 15th April 2014 at 23:59.

The deliverable of this practical is a report plus the source code for the constraint solver you will implement. Your report should document both your design and implementation decisions in detail. Similarly, it should justify your experimental design, describe it in sufficient detail that a competent reader could repeat your experiments, and contain a detailed analysis of your experimental results.

This practical will be marked following the standard mark descriptors given in the Student Handbook. Part 4 is an extension to the practical. It will be possible to obtain a mark of 17 without doing part 4 if parts 1-3 and the report are completed to a very high standard.

Part 1: A Forward Checking Constraint Solver

Implement, in the language of your choice, a Forward-Checking based constraint solver. For simplicity, your solver should employ **d-way branching**. In designing your solver, you will need to decide upon a suitable representation for variables, domains, and constraints. Table constraints are perfectly acceptable, but you may wish also to support some simple propagators for familiar binary relations like equality, disequality and inequality. Remember that Forward Checking is designed to work on **binary** constraints. You will also need to develop a mechanism for undoing pruning upon backtracking.

Since you are free to use a programming language of your choosing, your code must be very **well commented**. You should also include a README file with your submission with execution instructions.

Part 2: Heuristics

Extend your solver to support static variable orderings supplied as a list of variables in the order they should be assigned, as well as smallest-domain first ordering.

Part 3: Empirical Evaluation

Design and run a set of experiments to compare the merits of various static orderings (of your own choosing) with smallest-domain first. You will need to decide on a sensible basis for comparison, such as one or more of: time taken, nodes in the search tree, arc revisions.
You will also need to select a set of benchmark instances drawn from a problem class defined using binary constraints only. Suggested problems for your experiments include (but are not limited to):

- Graph colouring.
- Sudoku
- Langford's Problem (see http://www.csplib.org Problem 24)
- The (parameterised) crystal maze puzzle

Select **one** of the above (or another problem of your choosing) for your experiments.

In looking for other problem classes, remember that all-different constraints can be decomposed into binary disequalities. Furthermore optimisation problems can be transformed into decision problems, for example by replacing *minimising* x with an ordinary constraint $x < k$, for some k. This changes a problem asking for a solution with the smallest possible x to one that asks for a solution with an x smaller than k.

Part 4: Extensions

Part 3 asks you to experiment with a single problem class. Extend your empirical evaluation to include additional problem classes.

Extend your solver also to support **2-way** branching. Using the smallest-domain first heuristic, design and run a set of experiments to explore the relative performance of d-way and 2-way branching.