Constraint Programming

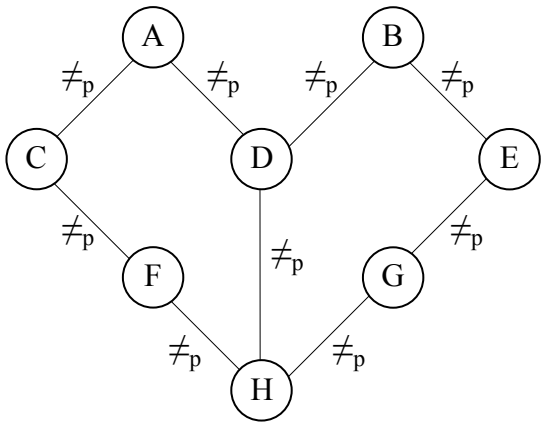
2015/2016 – Mini-Test #1

Friday, 13 November, 14:00 h in Room 115-II

Duration: 1.5 h (open book)

1. Finite domain Constraints - Propagation (7 pts)

Consider the constraint network on the right, where nodes represent variables, all with domain 1..10, and arcs (labelled $\neq_p$) represent constraints on the 2 variables attached so that they have different parity (if one variable is even the other should be odd).



a) **(2 pt)** Show that the problem is unsatisfiable (i.e. it has no solutions).

b) **(1 pt)** Show that imposing node- and arc-consistency does not detect such unsatisfiability.

c) **(2 pt)** Discuss the effect of imposing path-consistency on the network, both in terms of the values that are pruned from the variables domains, and on the binary constraints induced.

d) **(2 pt)** Assume now that your solver only maintains arc-consistency. What would be a good heuristic to select the variables to label. How many labelled variables would have to be backtracked in the search for solutions. Justify your answers.

Proposed Solution

a) The problem is unsatisfiable as shown with the following argument. A is either even or odd.

If A is even then on one hand D is odd, and on the other hand it is C odd, F even and H odd. But then, it cannot be the case that D and H are both odd and have a different parity, so A must not be even.

But, similarly, if A is odd, then on one hand D is even, and on the other hand it is C even, F odd and H even. But then, it cannot be the case that D and H are both even and have a different parity, so A must not be odd either.

Hence, A cannot take any value, neither even nor odd, and the problem is unsatisfiable.

b) Since there are no unary constraints, maintaining node-consistency does no achieve any pruning. Regarding arc-consistency, all variables have both even and odd values in their domain, so if they are constrained to have different parities each value of one variable has support on (several) values of the other variable. Hence no pruning is achieved if arc-consistency is maintained.

c) If path-consistency is maintained, a number of equal and different parity constraints are imposed on the problem variables. For example, C and D should have equal parity (i.e. C $=_p$ D), and C and H as well C $=_p$ F. But then another constraint D $=_p$ F should be imposed by path consistency. Hence no composed label {D-di and F-fi} is acceptable since di and fi must have simultaneaously the same parity (as imposed by path consistency) and different parity (as imposed by the initial constraint). So, path consistency detects unsatisfiability in this case.

d) If arc-consisency is maintained a good heuristic would be to start labelling by one of the variables D or H. Once this variable is labelled, the constraint graph becomes a tree and srac-consistency is "equivalent" to satisfiability. In this case, it is sufficient to label only one variable (D or H) that the propagation achieved by maintaining arc-consistency would be sufficient to detect unsatisfiability.

2. Modelling with Finite Domain Constraints (8 pts)

You are asked to solve the following problem: Find two sequences of 5 two-digit numbers such that

i. The 10 digits of the 5 two-digit numbers are all different;
ii. The difference between consecutive numbers in each of the sequences is the same; and
iii. No number in a sequence appears in the other.

For example, the sequences `S1 = <09,27,45,63,81>` and `S2 = <18,36,54,72,90>` are a solution to the problem as in both sequences the difference between consecutive numbers is `18`, in none of them a digit appears twice, and no number in a sequence appears in the other.

a) **(4 pt)** Specify a model for this problem in Comet, namely declaring the decision variables you use in the model (with their domains), as well as the constraints that should be posted to impose the restrictions of the problem. Make sure that your model does not yield repeated solutions, i.e. prevent the finding of the two solutions

`S1 = <09,27,45,63,81>` and `S2 = <18,36,54,72,90>`, as well as
`S1 = <18,36,54,72,90>` and `S2 = <09,27,45,63,81>`

which are in fact the same, i.e. they simply swap the values of S1 and S2.

b) **(1 pt)** Adapt your specification to accommodate a further constraint that the differences between consecutive numbers in the sequences are different in both sequences. In particular, the sequences above would no longer be a solution of the problem since the difference of `18` is the same in both sequences. In contrast, sequences `S1` above and `S3 = <50,61,72,83,94>` are a solution of this new problem (but `S2` and `S3` are not a solution since the two-digit number `72` appears in both sequences.

c) **(2 pt)** Assume you are asked to generalise the above problem for sequences longer than 5 numbers, i.e. to solve the problem: Find two sequences of **n** two-digit numbers such that

i. In the **2*n** digits of the **n** two-digit numbers, each digit does not appear more than **k** times;
ii. The difference between consecutive numbers in each of the sequences is the same; and
iii. No number in a sequence appears in the other.

Specify the new generalised problem stressing the changes with respect to the initial model.

Note: The initial problem is just a special case of the new one with **n = 5** and **k = 1**.

Proposed Solution

a) The problem is modelled with two arrays of variables one for as shown with the following argument. A is either even or odd.

```
Solver<CP> cp();
    var<CP>{int} x1[1..5](cp,0..99); // the first sequence of numbers
    var<CP>{int} d1[1..10](cp,0..9); // the digits in the first sequence
    var<CP>{int} g1(cp,0..99);       // the gap in the first sequence
    var<CP>{int} x2[1..5](cp,0..99); // the second sequence of numbers
    var<CP>{int} d2[1..10](cp,0..9); // the digits in the second sequence
    var<CP>{int} g2(cp,0..99);       // the gap in the second sequence
```

Given these variables the constraints are the following

```
    solve<cp> {
// the numbers are encoded by the digits
    forall(i in 1..5) {
        cp.post(x1[i] = 10* d1[2*i] + d1[2*i-1]);
        cp.post(x2[i] = 10* d2[2*i] + d2[2*i-1]);
// the numbers are all different
        forall(i in 1..5, j in 1..5)
            cp.post(x1[i] !=  x2[j]));
// the digits of the numbers in a sequence are all different
        cp.post(alldifferent(d1));
        cp.post(alldifferent(d2));
// numbers in the sequences are equidistant
        forall(i in 1..4) {
            cp.post(x1[i+1] == x1[i] + g1);
            cp.post(x2[i+1] == x2[i] + g2);
        }
// symetry breaking
                cp.post(x1[1] < x2[1]);

     }
```

b) Since the differences between consecutive numbers are explicit, it is sufficient to add constraint

```
            cp.post(g1 != g2);
```

c) To solve this generalization we change the ranges of the arrays from 1..5 to 1..n and from 1..10 to 1..2*n, as shown below (changes in blue). We also add two arrays c1 and c2 that count the occurrences of the digits in each of the arrays d1 and d2, which are limited to k

```
        var<CP>{int} x1[1..n](cp,0..99);   // the first sequence of numbers
        var<CP>{int} d1[1..2*n](cp,0..9);  // the digits in the first sequence
        var<CP>{int} g1(cp,0..99);         // the gap in the first sequence
        var<CP>{int} x2[1..n](cp,0..99);   // the second sequence of numbers
        var<CP>{int} d2[1.. 2*n](cp,0..9);// the digits in the second sequence
        var<CP>{int} g2(cp,0..99);         // the gap in the second sequence
    // number of occurrences of each of the  digits in the first sequence
        var<CP>{int} c1[0..9](cp,0..k);
        var<CP>{int} c2[0..9](cp,0..k);
```

In the new model the constraints that digits should be all different, i.e.

```
    // the digits of the numbers in a sequence are all different
        cp.post(alldifferent(d1));
        cp.post(alldifferent(d2));
```

is replaced by one that takes the counting into account, as follows:

```
// the digits of the numbers in a sequence do not appear more than k times
        cp.post(cardinality(c1,x1));
        cp.post(cardinality(c2,x2));
```

The other constraints remain the same, changing the bounds (from 5 to n, and from 10 to 2*n as done before).

3. Global Constraints (5 pts)

Consider a global constraint `sort([X1,X2,…,Xn], [Y1,Y2,…,Yn], [I1,I2,…,In])` that maps two arrays `X` and `Y` of decision variables (where `X` and `Y` can be regarded as representing the start time and duration of the tasks) into an array `I` of indices that constrains not only the order of the variables but also the non-overlapping of the corresponding tasks, i.e.

$$X[I[1]] + Y[I[1]] \leq X[I[2]] \ , \ \ldots \ , \ X[I[n-1]] + Y[I[n-1]] \leq X[I[n]]$$

a) If you find it useful, use this global constraint to implement a Comet function, named `schd`, that, given a set of `n` tasks with starting times `S = [S1,S2,…,Sn]` and durations `D = [D1,D2,…,Dn]`, contrains the variables so that the corresponding tasks not only do not overlap, but also that each task starts at least with a gap of `g` time units after the end of the previous task. For example, if the gap is `g = 2`, and there is one task with duration of `10` time units starting at time unit `23` then the next task could only start in time unit `35` (`=23+10+2`) or later.

b) Assume that the implementation of the constraint in your solver guarantees generalised arc-consistency (or domain consistency), even when the variables `X` and `Y` are subject to inequality constraints between themselves. Show what pruning would be achieved in the program in the case of `3` tasks, each with duration `12` (`D = {12,12,12}`), and starting times in the range `1..30` (i.e. declared in a solver `cp()` as `var<CP>int S[1..3](cp,1..25))` if the intended gap is `g = 2`.

Proposed Solution

a) The function below simply sorts the tasks and impose the extra gap required:

```
function void schd(Solver<CP> cp(), var<CP>{int} [] S, var<CP>{int} [] D, int g){
   Solver<CP> cp()= S[S.getLow()].getSolver();
   int n = S.getSize();
   cp.post(S,D,I);
   forall(k in 1..n-1)
     cp.post(S[I[k+1]] >= S[I[k]]+ D[I[k]] + g;
}
```

b) All 3 tasks have similar duration of `10` time units. So the assuming the first task starts as early as possible (in time unit `1`), then the second can start no earlier than at time unit `15 (1+12+2)` and the third in time unit `29 (15+12+2)`.

On the other hand, if the last task starts as late as possible (in time unit 30) than the second task can start as late as `16` (since `16+12+2=30`) and the first task as late as time unit `2` (since `2+12+2=16`). Hence the starting times of the 3 tasks can range, respectively in the intervals `1..2`, `15..16` and `29..30`.

But since the tasks have all the same duration, each of the tasks would have their initial domain narrowed from `1..30` to the union of intervals `1..2 ∪ 15..16 ∪ 29..30`.