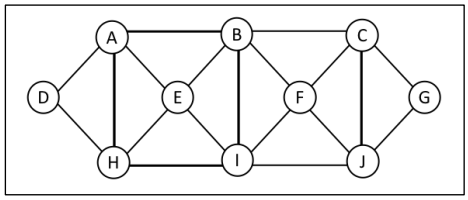
Constraint Programming

2018/2019– Mini-Test #1
Friday, 26 October, 18:00 h, Room 204-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 {**1,2,3**}. Arcs represent constraints of difference (i.e. X ≠ Y).

a) **(2 pt)** Is the problem satisfiable? If so, how many different solutions exist? Justify your answer.

b) **(1 pt)** What pruning is achieved initially, if node-consistency is maintained? And arc-consistency?

c) **(2 pt)** Are there any implicit binary constraint in the network? What type of consistency would be needed to infer such constraints? Justify your answer.

d) **(2 pt)** Since any solution to the problem has a symmetric solution that is a permutation of the values of variables A, D and H, could you impose symmetry breaking constraints to avoid obtaining symmetric solutions? With these constraints would there be any ordering of the labelling of variables that would be backtrack free? What type of consistency would there be needed? Justify your answer.

Proposed Solution

a) The problem is satisfiable and there are 6 possible solutions. One possible solution is

A = 1, D = 2, H = 3, and then E = F = G = A, I = C = A and B = J = H.

The other solutions are obtained by permutation of the values of A, D and H that must all be different.

b) Node-consistency does not prune any domain, since there are no unary constraints. Arc-consistency does not achieve any pruning either, since with 2 or more values in the domain of 2 different variables, every value in one of the variables has at least one support in the other variable.

c) There are many implicit binary constraint in the network For example, any consistent labelling of variables, A, D and H (making them all different), would impose constraint E = A. Similarly, several other implicit equality constraints could be inferred , e.g

- D = E = F = G
- A = I = C
- H = B = J

as well as many other difference constraints, such as D ≠ B, D ≠ I, etc…

Given the way they are inferred, these constraints would be explicit after imposing strong 4-consistency (i.e. any consistent label of 3 variables, e.g A, D and H could only be extended to a 4th variable, E, if E = A.

d) To avoid symmetric solution obtained from a permutation of variables A, D and H, it is enough to impose ann ordering on the variables, e.g. A > D > H, and obtain the symmetric solutions by the permutation of the solutions so obtained.

e) But now, simple bounds consistency (weaker than arc-consistency) would impose the constraint with no backtracking, since it would impose A = 3, D = 2 and H = 1. From these, node-consistency is sufficient to obtain fixed values for the other variables, i.e. E = 1 (since E ≠ A = 3 and E ≠ H = 1, so E = 2. Subsequently, node-consistency would impose then B = 3, I = 2, F = 1, C = 2, J = 3 and G = 1.

2. Modelling with Finite Domain Constraints (8 pts)

The Secret Code

Damien's grandfather left him a safe with an opening secret code of 4 digits and a card with the following codes written in it.

3702 4329 4961 5781 5781 5905 7685 8314 8764

The card also mentioned that each of these 9 codes had at least one correct digit (in the correct place), i.e. the same digit in the same place of the secret code. Can you find the code?

a) **(2 pt)** Specify a model for this problem in Comet. First, declare the data structures being used, including the decision variables you adopt as well as their domains. Assume a generalization of the problem, where the secret code has n digits (n=4 in this case) and there are m different codes (m=8 in this case), and these are provided in a m*n matrix (an 8*4 matrix, here) of digits.

b) **(4 pt)** To complete the models, what constraints would you consider in your model? Would you use any specific heuristics to label the variables? Justify your answer.

c) **(2 pt)** Assuming that there are more than one secret codes that satisfy the constraints, how would you change your model to obtain the largest such code (assuming the codes are read as integer numbers, i.e. code 7685 > 7641).

Proposed Solution:

a) We should organise the solution as an array of n digits, and hence use the following data structures. In the model below, we show matrix "codes" corresponding to this instance but this matrix should be replaced for other values of m and n (and the codes).

```
import cotfd;
     int m = 8;
     int n = 4;
     int codes[1..m, 1..n] = [
          [3,7,0,2],
          [4,3,2,9],
          [4,9,6,1],
          [5,7,8,1],
          [5,9,0,5],
          [7,6,8,5],
          [8,3,1,4],
          [8,7,6,4]
     ];
Solver<CP> cp();
     var<CP>{int} secret[1..n](cp,0..9);
```

b) The constraints should impose that at least one digit of each code matches the corresponding digit of the secret code. This can be imposed by a disjunctive constraint, where every digit of the secret code must be either the first, the second, …, or the n^{th} digit of each of the codes.

On the other hand, each digit of the secret code must be the same as, at least, the corresponding digit of one of the codes. Again this can be imposed by a disjunctive constraint imposing that each digit of the secret code is the same of the corresponding digit of the first, second, …, or the m^{th} code.

The disjunctive constraints can be modelled by reifying them into 0/1 values, and imposing that their sum is greater or equal to 1 (i.e. at least one of the constraints is satisfied). Hence the constraints of the model can be specified as follows

```
solve<cp> {
     forall(j in 1..4)
          cp.post(sum(i in 1..8) (secret[j] == codes[i,j]) >= 1);
     forall(i in 1..8)
          cp.post(sum(j in 1..4) (secret[j] == codes[i,j]) >= 1);
}
```

As to the heuristic there is no obvious best heuristics so we cn rely in the "general purpose" first fail heuristic and label the variable accordingly, as

```
using
     labelFF(secret);
```

c) The model before simply returns the first solution that is found. If one requires the greates of all possible solutions thand the solutions must be mapped into an integer decision variable that should be maximised. This can be done by first declaring this variable with the appropriate domain $0..10^n$

```
Solver<CP> cp();
     ...
     var<CP>{int} secret_value(cp,0..10^n);
```

and impose the channeling constraint between this variable and the secret code variables, for example as shown below

```
     cp.post(secret_value == sum(i in 1..n) (10^(n-i)*secret[i]));
```

Then the solver should be specified for find the maximum of this secret value, suject to all the constraints, as in

```
maximize<cp> secret_value {
     cp.post(secret_value == sum(i in 1..n) (10^(n-i)*secret[i]));

     forall(j in 1..4)
          cp.post(sum(i in 1..8) (secret[j] == codes[i,j]) >= 1);
     forall(i in 1..8)
          cp.post(sum(j in 1..4) (secret[j] == codes[i,j]) >= 1);
}
```

Again, there is no obvious heuristic to explore, so we could keep the first fail heuristic. However, being interested in the largest solution, we should label the values by decreasing order. There is no specified function for this, but we may program it from scratch as in

```
while !bound(secret) {
     selectMin (i in 1..n: secret[i].getSize() > 1) secret[i].getSize()
          tryall(v in secret[i].getDomain()) by (-v)
}
```

assuming that the methods x.getSize() and x.getDomain() return the size of the domain of decision variable x, and the set of values in the domain of such variable.

3. Global Constraints (5 pts)

Consider a "global" constraint that enforces two arrays (assumed to be indexed by the same range, thus having the same number of elements) to have no repeated values, and sharing exactly `k` values in their elements, not necessarily in the same positions. For example, the arrays `x = [2,5,7,9]` and `y = [9,5,7,2]` satisfy this constraint for `k = 2`, since values `2` and `7` are assigned to variables of both arrays and no other elements of `x` (resp. `y`) appear in `y` (resp. `x`). In contrast, `x` and `z = [3,7,1,7]` do not satisfy the constraint, not only because there is only one shared value (`7`) but also because it is repeated in `z`.

a) **(2pt)** Implement in Comet this constraint by means of a function declared as

```
function void share(var<CP>{int} k, var<CP>{int} [] x, var<CP>{int} []y){…}
```

b) **(2 pt)** If your implementation of this global constraint maintained domain consistency, what values would be pruned from the domain of the variables `x` and `y` if their domains are

```
                    x[1] in 1..3        y[1] in 1..8
                    x[2] in 1..3        y[2] in 2..4
      k in 0..3     x[3] in 1..3        y[3] in 2..4
                    x[4] in 2..6        y[4] in 2..4
                    x[5] in 3..6        y[5] in 3..7
```

c) **(1 pt)** In the same conditions of the previous item, assume that constraint `y[1] < 5` is posted. Would there be any further pruning of the domains?

Proposed Solution

a) After enforcing the arrays to be all different, counting the number of shared values can be obtained by counting the times an element of `x` has the same value of an element of `y`. Since each `x[i]` can be equal to at most one `y[j]`, this counting can be done by simply summing the times an element of array `x` is equal to an element of array `y`, as in

```
cp.post( k == sum(i in Rng, j in Rng) x[i] = y[j]);
```

The full code is shown below:

```
function void share(var<CP>{int} k, var<CP>{int} [] x, var<CP>{int} []y){
    Solver<CP> cp = k.getSolver();
    cp.post(alldifferent(x));
    cp.post(alldifferent(y));
    range Rng = x.getRange();
    cp.post( k == sum(i in Rng, j in Rng) x[i] = y[j]);
}
```

b) Because of the "pigeon-hole principle" imposed by the alldifferent constraints, values `1` and `3` must be taken by variables `x[1]`, `x[2]` and `x[3]`, thus pruning the domain of `x[4]` and `x[5]` to `{4,5,6}`. Similarly, values `2` to `4` are taken by variables `y[2]`, `y[3]` and `y[4]`, leaving variables `y[1]` and `y[5]` with domains `{1,5,6,7,8}` and `{5,6,7}`, respectively. Hence, values `2` and `3` are common to `2` variables of arrays `x` and `y`, and the domain of `x` is pruned to `2..3`.

Since, on one hand, one more value (`5`, `6` or `7`) may be taken in common by variables of arrays `x` and `y`, and on the other hand no other value needs to be in common to arrays `x` and `y` (as shown by the possible assignment `x[4]=5, x[6]=5, y[1]=7` and `y[5]=8`), both the remaining values `{2,3}` of the domain of `k` are possible in a solution.

c) If constraint `y[1] < 5` is posted, than `y[1]` is fixed to `1`, the same value of one of variables `x[1]`, `x[2]` and `x[3]`; hence the number of shared increases and is fixed to `3`, its upper limit. Since no more values can be shared between arrays `x` and `y`, then `y[5]` should be different from `x[4]` and `x[5]` and these different from any values in `y`.

Since `4` is taken by one of `y[2]`, `y[3]` or `y[4]`, then the domains of `x[4]` and `x[5]` become `{5,6}`, leaving no more options to `y[5]` than being fixed to `7`.