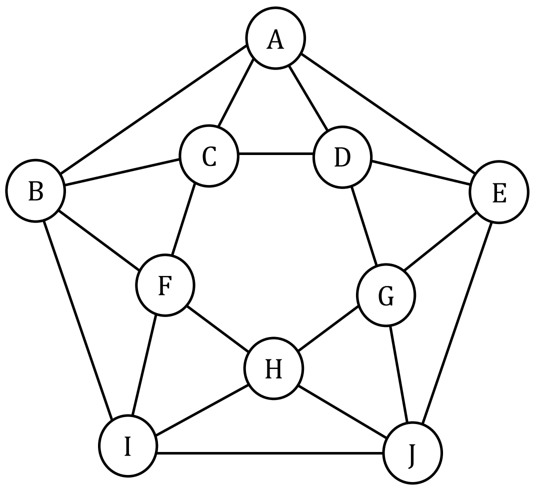
Constraint Programming

2017/2018– Mini-Test #1
Thursday, 2 November, 11:00 h in Room 114-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 different constraints (≠).

a) **(2 pt)** Is the problem satisfiable? Justify your answer.

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

c) **(2 pt)** What pruning is achieved initially, if path-consistency is maintained?

d) **(2 pt)** Assume now a problem with the same structure (variables and constraints) but where the variables and the constraints are arbitrary. Show that the problem is satisfiable if it is path consistent after labelling variables A and C. Justify your answer.

Proposed Solution

a) Given the domains of the variables, there are some implicit constraints that can be made explicit, namely:

A = G (A ≠ D, A ≠ E, D ≠ E and since A ≠ D, A ≠ E it must be A = G)

For similar reasoning, G = I and I = C, hence A = C, which contradicts constraint A ≠ C.

b) The initial network (with all variables with domain {1,2,3} is already node-consistent, as there are unary constraints. So maintaining node-consistent would not achieve any pruning.

Similarly, the network is already arc-consistent, since in difference constraints between variables with 2 or more values in their domain, any value of a variable has at least ione support on the other variable.

c) Path-consistency would not achieve any pruning either. Any label on two adjacent variables can be extended to a third, since there are 3 values in the domain of the variables.

d) If A and C are labelled, the remaining variables form a network of width 2, e.g. if the nodes are ordered as D, E, G, J, H, I, F, B, since any node is connected at most to 2 nodes of less order (for example, node H is connected to node G and J. Hence, maintaining path consistency, i.e. strong 3 consistency guarantees the satisfiability of the constraint network (any assignment of values to variables G and J can be extended to variable H).

2. Modelling with Finite Domain Constraints (8 pts)

Ian lives with his wife Lou, their two sons, Joe and Ken, and their mothers (Ian's mother, Mia, and Lou's mother's, Sue). He bought a set of ten birthday candles, each with a distinct digit. Then he noticed the coincidence that he could use each candle exactly once to celebrate the 6 birthdays of the family in this year. What are the ages of the six members of the family (after their birthday), given that a) no one had children before 19 nor after 32, b) the age difference between Ian and his wife Lou is no more than 2 years, and c) the age difference between the brothers is only one year.

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.

b) **(2 pt)** For the model you adopted, are there symmetric solutions, i.e. solutions that can be obtained from others by a simple mapping? If so, can you add extra constraints that prevent searching for these symmetric solutions.

c) **(2 pt)** Adapt the above problem, for another family with the same structure (for simplicity we keep the same names) for any age difference (≥ 1) between the brothers and where the 10 candles can be used exactly twice over two consecutive years.

Proposed Solution:

a) We may use six decision variables to encode the age of the six members of the family this year, namely

```
Solver<CP> cp();
     var<CP>{int} ian(cp, 1..100);
     var<CP>{int} lou(cp, 1..100);
     var<CP>{int} joe(cp, 1..100);
     var<CP>{int} ken(cp, 1..100);
     var<CP>{int} mia(cp, 1..100);
     var<CP>{int} sue(cp, 1..100);
```

Moreover, we must encode the digits of the ages of the members of the family, and map them into the previous variables. Since there must be 10 digits (each candle is used once) it is easy to see that the ages of the children must be encoded in single digits, whereas the other members of the family are encoded with 2 digits. We can thus adopt an array with 10 elements to represent the candles (i.e. the digits of the ages).

```
     var<CP>{int} c[0..9] (cp, 0..9);
```

We must impose that the candles are all different, since each candle is used only once.

```
solve<cp> {
     cp.post(alldifferent(c));
```

Of course, the ages must be mapped into the ages of the members of the family, which can be done with the following bridging constraints.

```
     cp.post(ian == 10 * c[0] + c[1]);
     cp.post(lou == 10 * c[2] + c[3]);
     cp.post(mia == 10 * c[4] + c[5]);
     cp.post(sue == 10 * c[6] + c[7]);
     cp.post(joe == c[8]);
     cp.post(ken == c[9]);
```

We can now impose the other constraints, to constrain the age when the adults had children

```
     cp.post(ian >= joe + 19); cp.post(ian <= joe + 32);
     cp.post(lou >= joe + 19); cp.post(lou <= joe + 32);
     cp.post(mia >= ian + 19); cp.post(mia <= ian + 32);
```

as well as the difference of age between the parents and the children

```
     cp.post(ian >= lou - 2); cp.post(ian <= lou + 2);
     cp.post(joe == ken +1);
}
```

b) The constraints above do not impose whether Ian is older than his wife Lou, or the other way around. By symmetry, for each solution where Ian is older than Lou, there is a corresponding solution where Lou is older than Ian. Hence, a symmetry breaking constrain to avoid this duplication is simply

```
            cp.post(ian > lou);
```

In the previous item, it was assumed that Joe is older than Ken, nd this already eliminated another potential symmetry (between the ages of the brothers).

c) We may adapt the model above by considering now a matrix of ages with 2 rows, with the corresponding bridging constraints

```
Solver<CP> cp();
    var<CP>{int} ian(cp, 10..100);
    var<CP>{int} lou(cp, 10..100);
    var<CP>{int} joe(cp, 1..9);
    var<CP>{int} ken(cp, 1..9);
    var<CP>{int} mia(cp, 10..100);
    var<CP>{int} sue(cp, 10..100);
    var<CP>{int} ages[1..2, 0..9](cp, 0..9);
solve<cp> {
    cp.post(ian == 10 * c[1,0] + c[1,1]);
    cp.post(lou == 10 * c[1,2] + c[1,3]);
    cp.post(mia == 10 * c[1,4] + c[1,5]);
    cp.post(sue == 10 * c[1,6] + c[1,7]);
    cp.post(joe == c[1,8]);
    cp.post(ken == c[1,9]);
```

We must impose that the ages in the second row are one more than the ages of the first row, since the years are consecutive

```
    cp.post(10 * c[1,0] + c[1,1] + 1 == cp.post(10 * c[2,0] + c[2,1]);
    cp.post(10 * c[1,2] + c[1,3] + 1 == cp.post(10 * c[2,2] + c[2,3]);
    cp.post(10 * c[1,4] + c[1,5] + 1 == cp.post(10 * c[2,4] + c[2,5]);
    cp.post(10 * c[1,6] + c[1,7] + 1 == cp.post(10 * c[2,6] + c[2,7]);
    cp.post(c[1,8] + 1 == cp.post(c[2,8]);
    cp.post(c[1,9] + 1 == cp.post(ages[2,9]);
```

and relax the age difference between the brothers to

```
    cp.post(joe > ken);
```

Finally we must impose that each of the digits occurs exactly twice in matrix ages, which can be imposed by a cardinality constraint after "flattening the matrix into an array

```
    int exact[i in 0..9] = 2;
    cp.post(cardinality(exact, all(i in 1..2,j in 0..9) c[i,j], exact));
}
```

3. Global Constraints (5 pts)

Consider a function that constrains an array of `n` decision variables, `a`, that take values in the range `1..m`, such that the maximum difference between the number of occurrences in a solution of any values in this range is less or equal to `k`. For example, array `a = [2,2,1,1,2,2]` does not satisfy the constraint when `m = 3` and `k = 2` since the difference between the number of occurrences of the values 2 (that occurs 4 times) and 3 (0 times) is **4**, hence greater than `k = 2`. However, it satisfies the constraint for `m = 2` and `k = 2`.

a) **(2 pts)** Implement in Comet this "global constraint" with the signature below

```
function void balanced(var<CP>{int} [] a, int m, int k) {…}
```

b) **(2 pts)** If your implementation of this global constraint would maintain domain consistency, i.e. it pruned any values from the domain of the variables that do not occur in any solution, what would be such pruning for `n = 8`, `m = 5`, `k = 2` and constraint variables with domains

```
a[1] in 1..4;     a[2] in 2..3;     a[3] in 2..3;     a[4] in 1..4;
a[5] in 1..4;     a[6] in 2..3;     a[7] in 2..3;     a[8] in 1..4;
```

c) **(1 pts)** Assume that the constraint also imposes the array a to be monotonically increasing (i.e. `forall(i in 1..n-1) a[i] <= a[i+1]`). Again, assuming that the implementation of the constraint guarantees domain consistency, would there be any further pruning of the domains?

Proposed Solution

a) The function below includes a counting scheme for all values in the array, and restricts the counting between any two values to be less or equal to the value k

```
function void balance(var<CP>{int} [] a, int m, int k){
     Solver<CP> cp = a[a.getLow()].getSolver();
     int n = a.getSize();
     var<CP>{int} c[1..m](cp,0..n);
     forall(v in 1..m)
          cp.post(c[v] == sum(i in 1..n) (a[i] == v));
     forall(i in 1..m, j in 1..m : j > i)
          cp.post(abs(c[i]-c[j]) <= k);
}
```

b) Since `m = 5`, one is interested in counting the occurrences of values from 1 to 5 as values of variables `a[i]`. But since 5 does not appear in the domain of any variable, the number of occurrences of any other value must be at most 2.

Since there are 8 variables, then each of the values 1, 2, 3 and 4 should appear exactly twice in a solution. But variables **a[2], a[3], a[6]** and **a[7]** only have values 2 and 3 in their domain, so two of them must take value 2 and the others value 3.

Regardless of which variables take which values 2 and 3, from the the other 4 variables (**a[1]**, **a[4]**, **a[5]** and **a[8]**), two must take value 1 and the other two must take value 4.

Therefore, the domain of variables **a[1]**, **a[4]**, **a[5]** and **a[8]** should be pruned to **{1,4}.**

c) Indeed, the problem becomes impossible!

Because `a[3]` is greater than 1, variables **a[4]**, **a[5]** and **a[8]** should be fixed to 4. But then, the value 4 would appear 3 times in the solution, which contradicts the upper limit of 2 on the counting, discussed in the previous item.