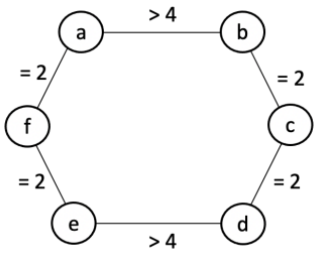
Constraint Programming

2020/2021– Mini-Test #1
Tuesday, 3 November, 9:00 h, Room 204-II
Duration: 1.5 h (open book)

1. Finite domain Constraints - Propagation (6 pts)



Consider the constraint network on the right, where nodes represent variables, all with domain {**1,2,3,4,5,6}**. Arcs labelled "**>p**" and "**=q**" constrain the absolute difference between the connected variables to satisfy that condition (i.e. **| a - b | > 4** and **| a - f | = 2**).

a) **(1 pt)** Is the problem satisfiable? If so, how many different solutions exist? Justify your answer.

b) **(2 pt)** What pruning is achieved initially, if node-consistency is maintained? And bounds-consistency? And arc-consistency?

c) **(2 pt)** Are there any implicit binary constraint in the network? Would path consistency infer such constraints? Justify your answer.

d) **(1 pt)** Notice that the problem exhibits some symmetry, namely horizontal and vertical reflections? Given the analysis you did in the previous items could you add additional symmetry breaking constraints. Justify.

Proposed Solution

a) The problem is satisfiable, and has two solutions:
- The **>4** constraints prune the domains of variables **a, b**, **d** and **e** to the set **{1,6}**, and **a ≠b**;
- The **=2** constrain variables **a** and **e** to have the same parity (in fact a difference of 0 or 4).
- Hence, **a** must be equal to **e** and, likely, **b** must be equal to **d**.
- Hence the solutions are
  - S1: **a = e = 1, f = 3, b = d = 6, c = 4**; and
  - S2: **a = e = 6, f = 4, b = d = 1, c = 3**.

b) Node-consistency does not prune any domain, since there are no unary constraints.

Bounds consistency does not prune any of the domains since the bounds of the domain of all variables have support on the other connected variables. For example, **a = 1** has support on **b = 6** and **c = 3**; **a = 6** has support on **b = 1** and **c = 4**. Hence the domain of **a** is not pruned. And the same applies to variables **b**, **d** and **e**. Regarding variable **c**, its upper bound **6** has support on **b = 4** and **d = 4**, and its lower bound **1** has support in **b = 3** and **d = 3**, and the same applies to **f**.

Arc-consistency removes values **2,3,4** and **5** from the domain of **a**, as they have no support on **b**, and the same applies to **b**. Hence variables **a, b, d** and **e** have their domain reduced to **{1,6}**. And these pruned domains imply, in turn, that variables **c** and f have their domains pruned to **{3,4}**.

c) Path consistency imposes that variables **a** and **e** have a difference of either 0 or 4. With their domains pruned to **{1,6}**, this means that arc-consistency would infer that **a = e** (and **b = d**).

d) Given the vertical symmetry of the problem, a constraint **a ≥ b** should eliminate solution **S1**, that can be inferred from **S2** changing a ↔ b, d ↔ e, and c ↔ f.

2. Global Constraints (5 pts)

Consider a "global" constraint, `distribute`, that enforces the elements of an array so as the number of times the values appear in a solution are all different, except if they do not appear at all. For example, given an array of 4 elements with domains {0,1,2}, the assignment [2,2,2,2] is a possible solution (2 appears 4 times and both 0 and 1 appear 0 times), as is [1,1,1,2] (1 appears 3 times, 2 appears once, and 0 does not appear), but not [0,2,2,1] since values 0 and 1 appear the same number of times (once) in that assignment.

a) **(3pt)** Implement in Choco this constraint in a function **distribute** with signature

```
function void distribute(Model md, IntVar [] A, int n)
```

where the domain of all variables in array A can only take values in the range 0..n-1.

Suggestion: Use the predefined constraints

- `count(int v, IntVar [] A , IntVar c) : it` constrains the number of variables in `A` that have value `v` to be exactly `c`; and
- `allDifferentExcept0(A): it` constrains the variables in `A` to have different values, except value 0 that may appear more than once.

b) **(2 pt)** If your implementation of this global constraint maintained domain consistency, what values would be pruned from the domain of the array A, with the initial domains shown.

```
A[0] in {2, 4}
A[1] in {0, 1}
A[2] in {1, 4}
A[3] in {0, 3, 5}
A[4] in {0, 3, 5}
A[5] in {2, 3}
```

Proposed Solution

a) Since we know that each of the values **0..n-1** may appear in a solution at most **n** times, the we only have to count the number of occurrences of each of these values in an IntVar array **C**, and guarantee that these counts are not repeated, except if they are zero.

```
function void distribute(Model md, IntVar [] A, int n){
    IntVar [] C = intVarArray("C", n, 0, n)
    for (int v = 0; v < n; v++){
        md.count(v, A, C[v]);
    }
    md.allDifferentExcept0(C).post();
}
```

b) Because there are **6** values in the domain, a possible assignment has all **6** variables with the same value. But there is no value appearing in all **6** variables. Neither is any value appearing in **5** or **4** variables. Hence the values that appear in a solution must be such that one value appears **3** times, another value appears **2** times and a third value appears once. The other values do not appear in the solution.

For the domains given, only 0 and 3 appear 3 times in the domain of all variables.

If A[1] = A[3] = A[4] = 0 then the only values that appear twice in the domains of the other variables are either

a. A[0] = A[2] = 4 leaving A[5] to be 2 or 3; or
b. A[0] = A[5] = 2 leaving A[2] to be 1 or 4; or

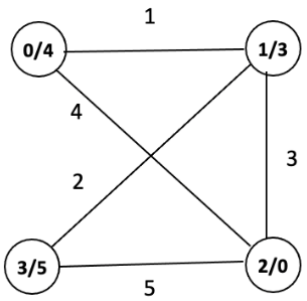
If A[3] = A[4] = A[5] = 3 then the only values that appear twice in the domains of the other variables are either

a. A[0] = A[2] = 4 leaving A[1] to be 0 or 1; or
b. A[1] = A[2] = 1 leaving A[2] to be 1 or 4; or

In all these cases, value 5 does not appear in any solution and may be pruned from the domains of variables A[3] and A[4].

3. Modelling with Finite Domain Constraints (9 pts)

Graceful Graphs (Prob. 53 of CSPLIB)



As shown in the figure, a labelling **f** of the nodes of a graph with **q** edges is graceful if:

- **f** assigns each node a unique label from 0, 1, … , q;
- each arc (**x,y)** is labelled **with |f(x)−f(y)|**; and
- the arc labels are **all different**.

The graph shown represents a graceful labelling, where each node is denoted with a pair a/b where a in the *order* of the node and b its **label** (for example the arc connecting **node 3** with **label 5** and **node 1** with **label 3** has **label 2 = 5 – 3**).

Assume that you are given the number of nodes, **n,** and a graph specified by an **m x 2** matrix where each row represents an arc, and the two columns represent the order of the nodes connected by the arc. For example, the graph above is represented by matrix **G** and the number of nodes **n**:

```
int [][] G = {{0,1},{0,2},{1,2},{1,3},{2,3}};
int n = 4;
```

a) **(2 pt)** Specify a model for this problem in Choco. More precisely, declare the decision variables you chose as well as their domains, together with the model and solver you propose.

b) **(4 pt)** (SAT) What constraints would you consider to modelling the satisfaction problem, i.e. to find a graceful labelling of the graph?

c) **(3 pt)** (OPT) Assume that a rank is defined for a label of the graph as the sum of the product of the order of the nodes by their label. For example, the label shown has rank

```
R = 0*4 + 1*3 + 2*0 + 3*5 = 18
```

Adapt your model to find the minimal graceful label of the graph.

Proposed Solution:

a) We consider an array N of decision variables for each of the nodes, and another array A for the arcs. The domains of the variables denoting the labels of the nodes and arcs are respectively **0..na** and **1..na**.
The choco code may thus be

```
int [][] G = {...};
int n = ...;
int na = G.length;
Model md = new Model();
Solver sv = md.getSolver();
IntVar [] N = md.intVarArray("N", n,  0, na);
IntVar [] A = md.intVarArray("A", na, 1, na)
```

b) There are two types of constraints. The labels of the arcs must all be different,

```
md.allDifferent(A);
```

and the label of each arc should be the absolute difference between the labels of the connected nodes. Form the above specification of the graph, these constraints can be modelled as:

```
for (int a = 0; a < na; a++){
    //A[a] = abs( N[G[a][0]] - N[G[a][1]]);
    A[a].eq( (N[G[a][0]].sub(N[G[a][1]])).abs() ).post();
```

```
}
```

c) To obtain the minimal graceful label we must define the objective function as a variable, and then set the search to meet this objective.
Variable **rank**, corresponding to the objective function, may be specified as the sum of the products of the orders of the nodes by their labels, as in

```
IntVar [] rs = md.intVarArray("rs", n,  0, n*na); // n*na is upper bound
IntVar rank  = md.intVar("rank", 0, n*n*na);    // n*n*na is upper bound
for (int i = 0; i < n; i++)
     rs[i].eq(N[i].mul(i)).post();
md.sum(rs, "=", rank).post();
```

The minimization objective is set by invoking the model method

```
md.setObjective(Model.MINIMIZE, rank);
```