Constraint Programming

- An overview

  - Why Global Constraints
  - Examples: All-diferent, global cardinality, ...
  - Global constraints in SICStus / Comet
  - A catalogue of Global constraints

Why Global Constraints

- It is often important to define n-ary "global" constraints, for at least two reasons:
  a) to simplify the modelling of a problem (higher abstraction); and
  b) to exploit specialised algorithms that take into account the semantics of the constraint for efficient propagation (achieving generalised arc consistency?).

Example: all_different ([A_1, A_2, ..., A_n])

Constrain a set of n variables to be all different among themselves.

- The constraint definition based on binary difference constraints ($\neq$) does not pose much modeling problems. For example, it may be being defined recursively by binary diff constraints between any pair of variables A_i and A_j.
- However, constraint propagation based on binary constraints **alone** does not provide in general much propagation.
- As seen before, arc consistency is not any better than node consistency, and higher levels of consistency are in general too costly and do not take into account the **semantics** of the all_different constraint.

Why Global Constraints

Example:

X_1: 1,2,3	X_6: 1,2,3,4,5,6,7,8,9
X_2: 1,2,3,4,5,6	X_7: 1,2,3,4,5,6,7,8,9
X_3: 1,2,3,4,5,6,7,8,9	X_8: 1,2,3
X_4: 1,2,3,4,5,6	X_9: 1,2,3,4,5,6
X5: 1,2,3	

It is clear that constraint propagation based on maintenance of node-, arc- or even path-consistency would not eliminate any redundant label. Yet, it is very easy to infer such elimination with a global view of the constraint!

- Variables X1, X5 and X8 may only take values 1, 2 and 3. Since there are 3 values for 3 variables, these must be assigned these values which must then be removed from the domain of the other variables.
- Now, variables X2, X4 and X9 may only take values 4, 5 e 6, that must be removed from the other variables domains.

Global Constraints – All Different

- **Example: Sudoku**
- Cuts in green are all that are found by maïve all_diff.
- Global all_diff finds all values **with no** backtracking.
- The first cuts are illustrated in the figure
- (the indices show a possible order in which the cuts are made)

		8	4		6 4	3	5	
		6 12	37 7		1		8	
		3	9		8			6
2		1 12	258 6	9	347 5	7		
	9	4 2	258 6	6	347 5		1	
		5	258 6	1	347 5			3
6		7 1	1	378 9	2	4		
	4	2 11	6	378 9	9 3			
13 10	8	9	37 7	4 8	5	6	237 10	127 10

Why Global Constraints

- These prunings could be obtained, by maintaining (strong) 4-consistency.

- However, such maintenance is very expensive, computationally. For each combination of 4 variables, d^4 tuples whould be checked, with complexity $O(d^4)$.

- In fact, in some cases, n-strong consistency would be required, so its naïf maintenance would be exponential on the number of variables, exactly what one would like to avoid in search!

- However, taking the semantics of this constraint into account, an algorithm based on quite a different approach allows the prunings to be made at a much lesser cost, achieving generalised arc consistency.

Global Constraints – All Different

- Such algorithm (see [Regi94]*), is grounded on graph theory, and uses the notion of graph matching.
- To begin with, a **bipartite graph** is associated to an all_diff constraints. The nodes of the graphs are the variables and all the values in their domains, and the arcs associate each variable with the values in its domain.
- In polinomial time, it is possible to eliminate, from the graph, all arcs that do not correspond to possible assignments of the variables.

Key Ideas:

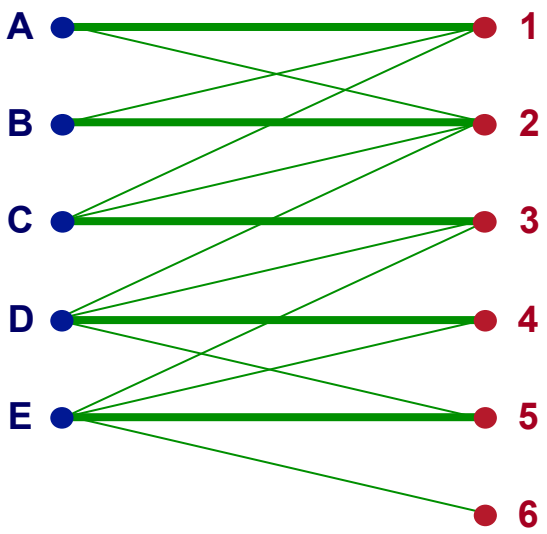
- For each variable-value pair, there is an arc in the bipartite graph.
- A matching, corresponds to a subset of arcs that link some variable nodes to value nodes, different variables being connected to different values.
- A **maximal matching** is a matching that includes all the variable nodes.
- For any solution of the all_diff constraint there is one and only one maximal matching.

* **J.-C. Régin, A Filtering Algorithm for Constraints of Difference in CSPs, Proceedings of AAAI-94, pp.362-367, 1994**

Global Constraints – All Different

Example: **A,B:: 1..2, C:: 1..3, D:: 2..5, E:: 3..6, all_diff([A,B,C,D,E]).**

- A maximal matching is found, connecting each of the variables nodes with one and only one value node; no values nodes are shared.



Maximal Matching

A = 1
B = 2
C = 3
D = 4
E = 5

Global Constraints – All Different

Example: **A,B:: 1..2, C:: 1..3, D:: 2..5, E:: 3..6, all_diff([A,B,C,D,E]).**

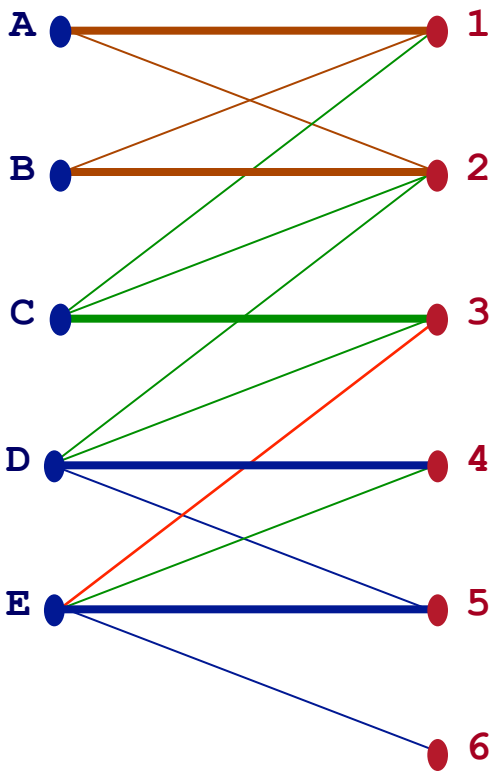
- The propagation (domain filtering) is done according to the following principles:

1. If an arc does not belong to any **maximal matching**, then it does not belong to any all_diff solution.
2. Once determined some maximal matching, it is possible to determine whether an arc belongs or not to any maximal matching.
3. This is because, given a maximal matching, an arc belongs to any maximal matching iff it belongs:
   a) To an **alternating cycle**; or
   b) To an **even alternating path**, starting at a free node.

Global Constraints – All Different

Example: For the maximal matching (MM) shown

- 6 is a free node;
- 6-E-5-D-4 is an even alternating path, alternating arcs from the MM (E-5, D-4) with arcs not in the MM (D-5, E-6);
- A-1-B-2-A is an alternating cycle;
- E-3 does not belong to any alternating cycle
- E-3 does not belong to any even alternating path starting in a free node (6)
- E-3 may be filtered out!



Global Constraints – All Different

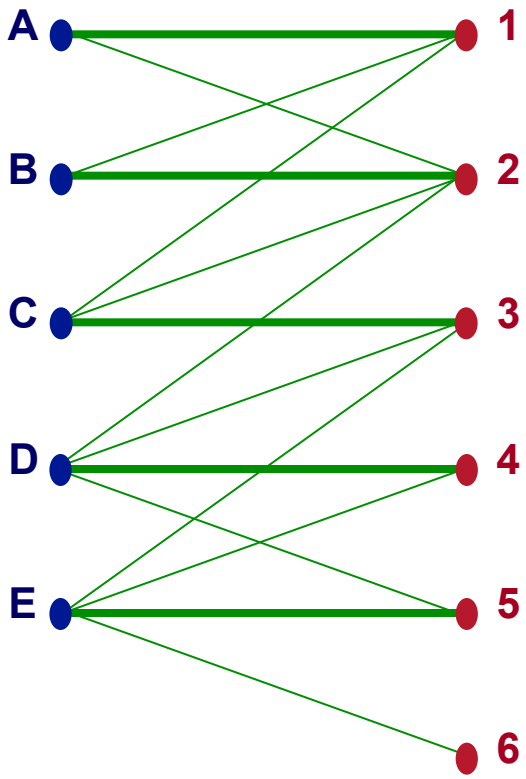
- **Compaction**

  - Before this analysis, the graph may be “compacted”, aggregating, into a single node, “equivalent nodes”, i.e. those belonging to alternating cycles.

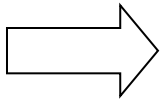
  - Intuitively, for any solution involving these variables and values, a different solution may be obtained by permutation of the corresponding assignments.

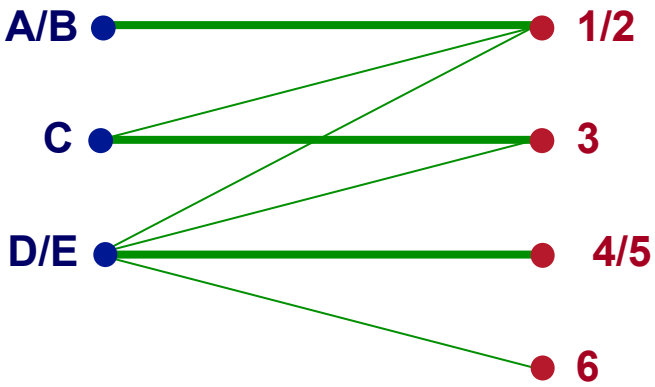
  - Hence, the filtering analysis may be made based on any of these solutions, hence the set of nodes can be grouped in a single one.

Global Constraints – All Different



- A-1-B-2-A is an **alternating cycle**;
- By permutation of variables A and B, the solution <A,B,C,D,E> = <1,2,3,4,5> becomes <A,B,C,D,E> = <2,1,3,4,5>
- Hence, nodes A e B, as well as nodes 1 and 2 may be grouped together (as may the nodes D/E and 4/5).

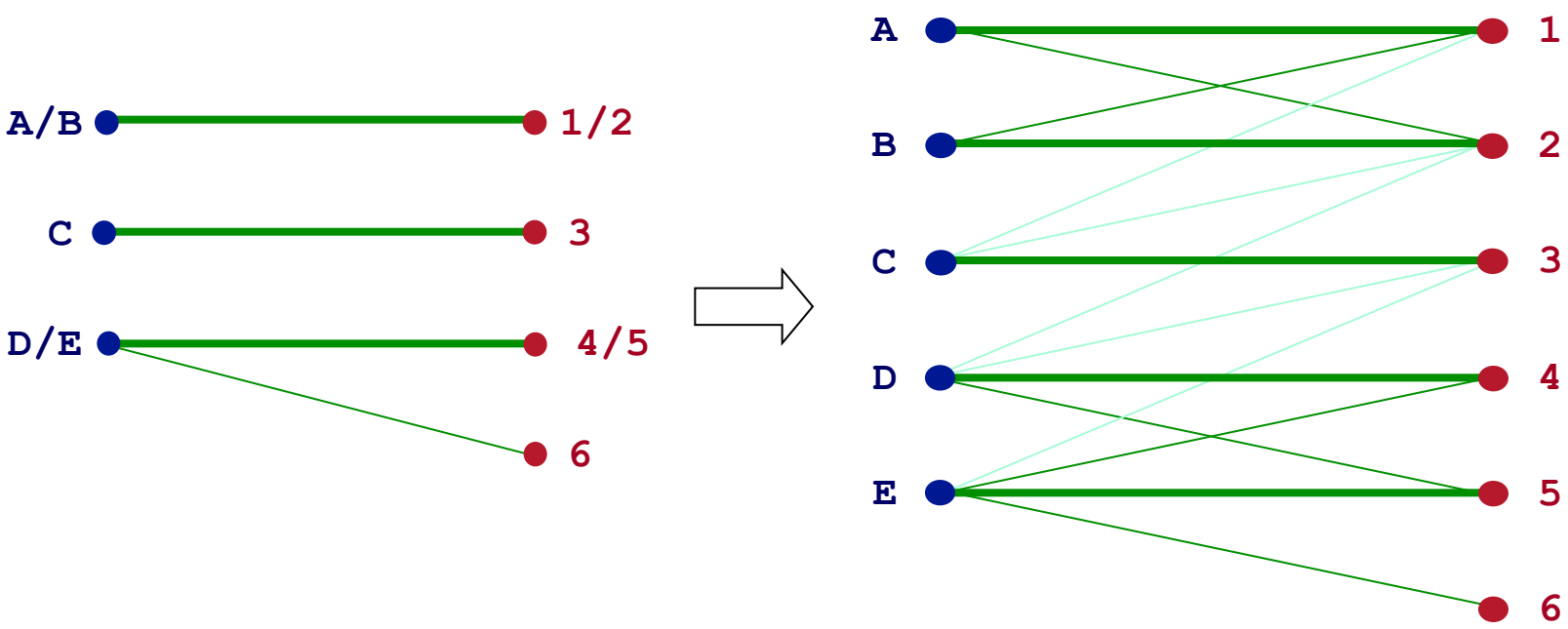




With these grouping the graph becomes much more compact

Global Constraints – All Different

By expanding back the simplified compact graph, one gets the pruned original graph



which immediately sets C=3 and, more generaly, filters the initial domains to

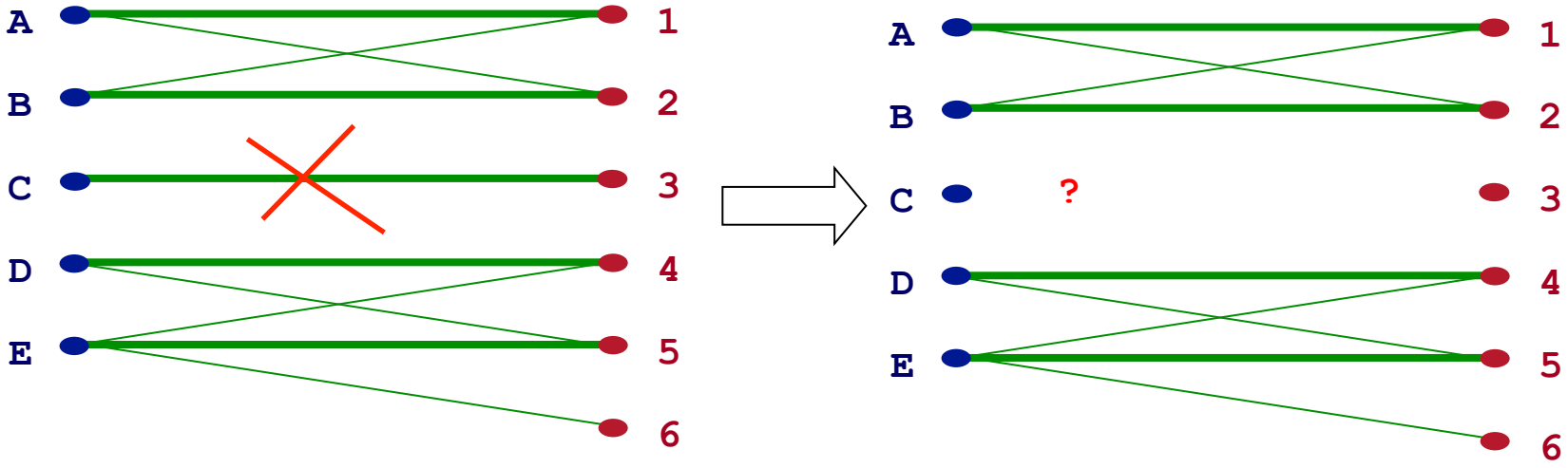
A,B :: 1..2, C:: 1,2,3, D:: 2,3,4,5, E:: 3,4,5,6

Global Constraints – All Different

Incremental Propagation:

- An important issue in CP is that propagation, can be done incrementally, upon elimination of values from variables.
- In this case, upon elimination of some labels (arcs in the bipartite graph), possibly due to other constraints, the all_different constraint propagates such prunings, incrementally. 3 situations are considered:

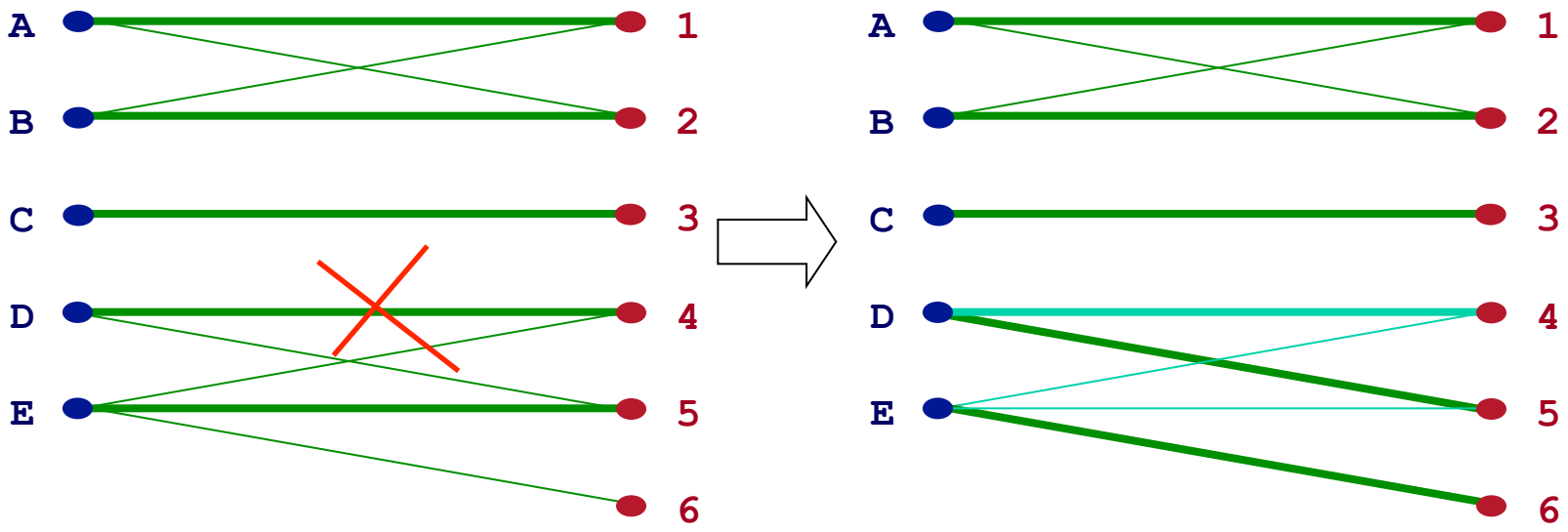
1. Elimination of a **vital arc** (the only arc connecting a variable node with a value node): The constraint **cannot be satisfied**.



Global Constraints – All Different

2. Elimination of a **non-vital arc which is a member** to the maximal matching

   - Determine a new maximal matching and restart from there.

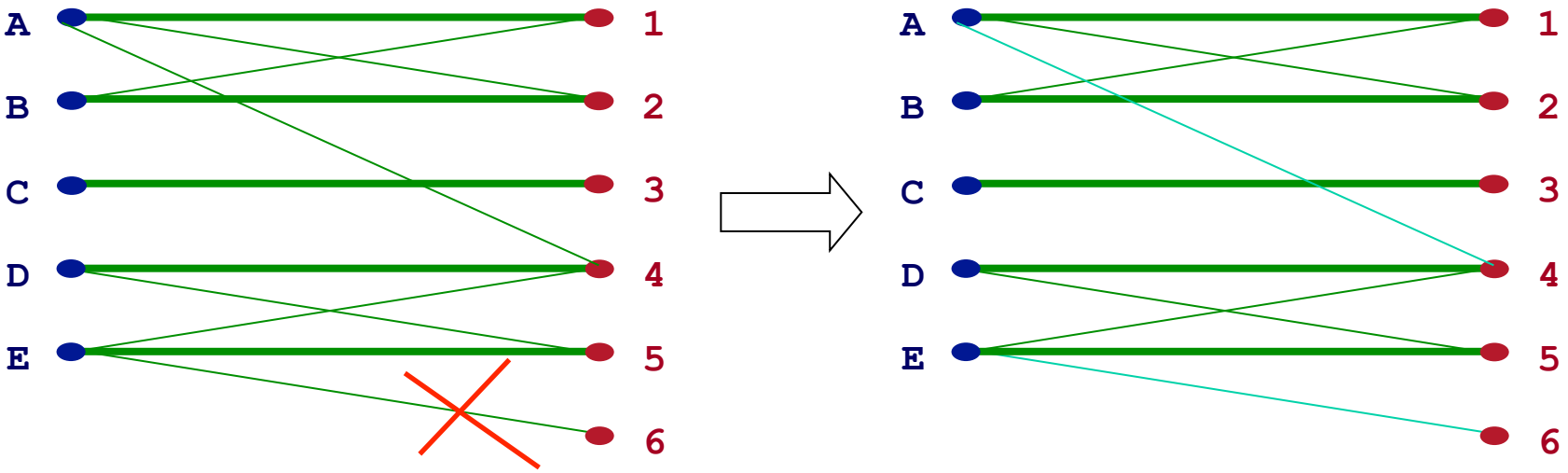


   - A new maximal matching includes arcs **D-5** and **E-6**. In this matching, arcs **E-4 and E-5** do not belong to even alternating paths or alternating cycles.

Global Constraints – All Different

3. Elimination of a **non-vital arc which is not a member** to the maximal matching

- Eliminate the arcs that belong nomore to an alternating cycle or path.



- Arc **A-4** belongs no more to the **even alternating path** started in node 6, and is eliminated.
- **D-5** also belongs to this path, but still belongs to an **alternating cycle**, and is kept!

Global Constraints – All Different

Time Complexity:

Assuming **n** variables, each of which with **d** values, and where **D** is the cardinality of the union of all domains,

1. It is possible to obtain a maximal matching with an algorithm of time complexity $\mathbf{O(dn^{3/2})}$.
2. Arcs that do not belong to any maximal matching may be removed with time complexity **O(dn+n+D)**.
3. Taking into account these results, we obtain complexity of $\mathbf{O(dn+n+D+dn^{3/2})}$. Since **D < dn**, the total time complexity of the algorithm is dominated by the last term, thus becoming

$$\mathbf{O(dn^{3/2})}.$$

which is much better than the poor result with a naïf analysis.

Global Constraints – All Different

History:

1. The **all_diff** constraint first appeared in the CHIP system (algorithm?).

2. The described implementation is incorporated into the ILOG system, and avalable as primitive **IlcAllDiff**.

3. This algorithm is also implemented in SICStus, through buit-in constraint **all_distinct/1**.

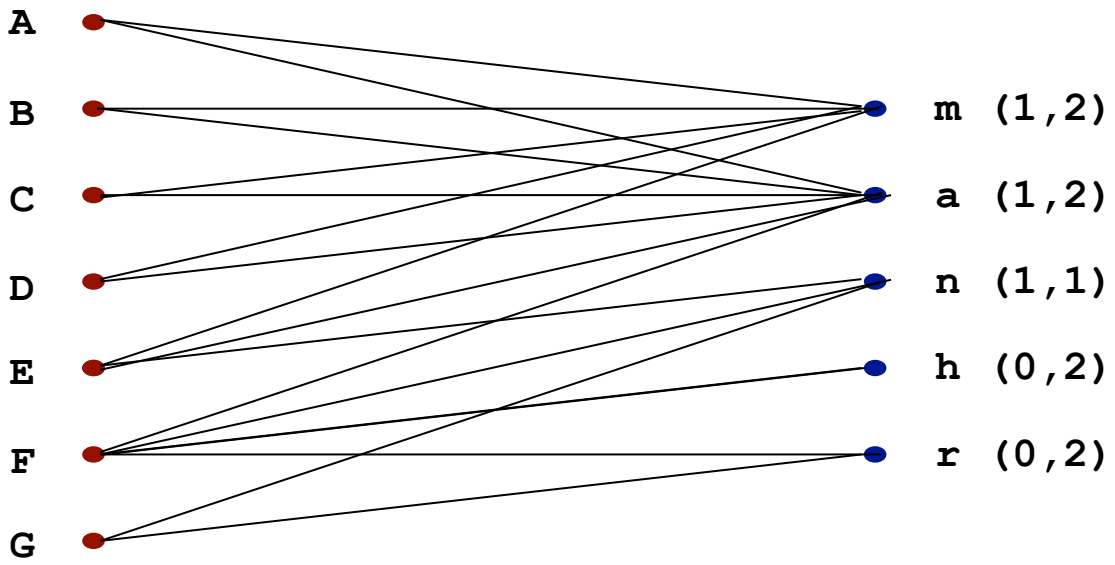
4. Other versions of the constraint, namely **all_different/2**, are also available, using a faster algorithm but with less pruning (only bounds consistency), where the 2nd argument controls the available pruning options (e.g. Bounds-consistency).

Global Constraints – Global Cardinality

- Many scheduling and timetabling problems, have quantitative requirements of type

In these N "slots" M must be of type T

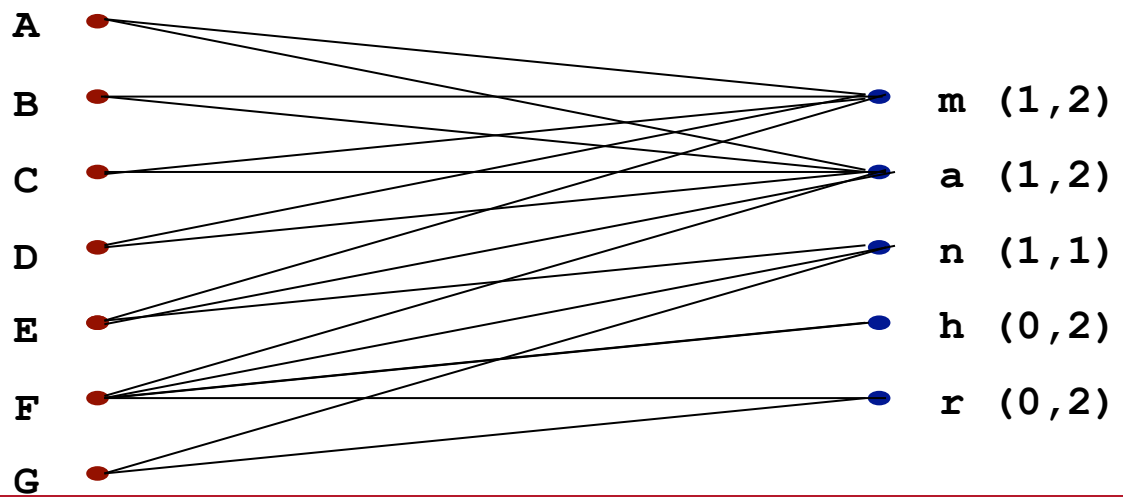
- For example, assume a team of 7 people (nurses) where one or two must be assigned the morning shift (m), one or two the afternoon shift (a), one the night shift (n), while the others may be on holliday (h) or stay in reserve (r).



Global Constraints – Global Cardinality

- To model these problems, let us consider a list L, whose variables L_i corresponding to the 7 people available, may take values in domain {m, a, n, h, r} (or {1, 2, 3, 4, 5} in languages like SICSTus that require domains to range over integers).

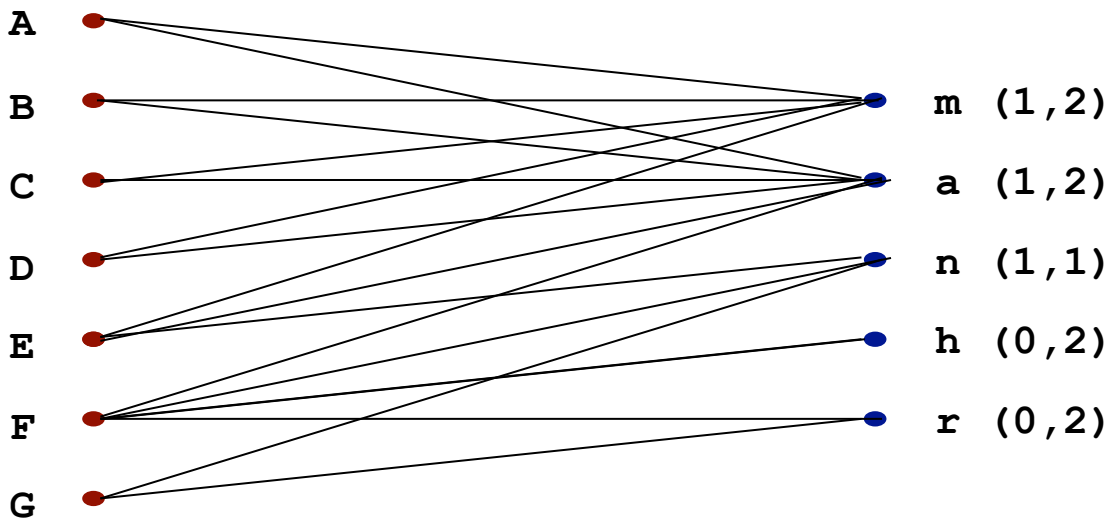
```
L = [A,B,C,D,E,F,G],
A,B,C,D in {m,a}, E in {m,a,n},
F in {a,n,h,r}, G in {n,r}
```



Global Constraints – Global Cardinality

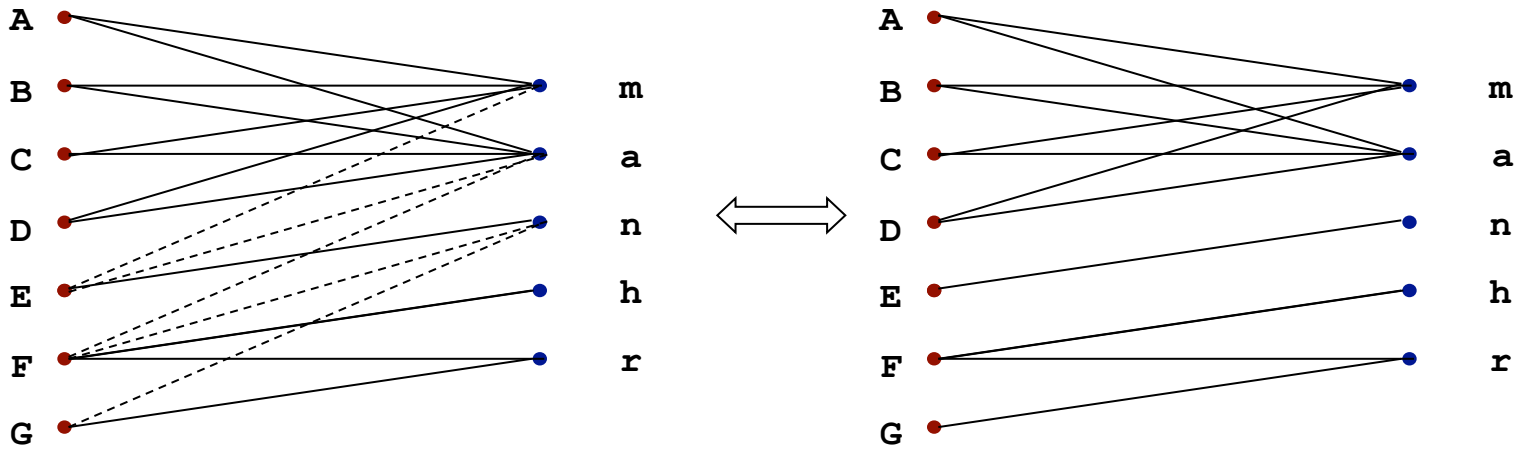
- A built-in constraint count/4 may be used to “count” elements in a list:

```
L = [A,B,C,D,E,F,G],
A,B,C,D in {m,a}, E in {m,a,n}, F in {a,n,h,r}, G in {n,r}
count(1,L,#>=,1), count(1,L,#=<,2)  % 1 or 2       m/1
count(2,L,#>=,1), count(2,L,#=<,2)  % 1 or 2       a/2
count(3,L,#=, 1) ,                  % 1 only        n/3
count(4,L,#>=,0), count(4,L,#=<,2)  % 0 to 2       h/4
count(5,L,#>=,0), count(5,L,#=<,2)  % 0 to 2       r/5
```



Global Constraints – Global Cardinality

- However, cardinality may be more efficiently propagated if considered **globally**.
- In fact, the separate, or local, handling of each of these constraints, does not detect all the pruning opportunities for the variables domains.
- **A, B, C** and **D** may only take values **m** and **a**. Since these may only be attributed to 4 people, no one else, namely **E** or **F**, may take these values m and a.
- Since **E** may now only take value **n**, that must be taken by a single person, no one else (e.g. **F** or **G**) may take value **n**.



Global Constraints – Global Cardinality

- This filtering, that could not be found in each constraint alone, can be obtained with an algorithm that uses analogy with results in maximum network flows [Regi96]*.
- A global cardinality constraint **gcc/4**,
  - constrains a list of **k** variables **$X = [X_1, ..., X_k]$** ,
  - taking values in the domain (with m values) **$V = [v_1, ..., v_m]$**,
  - such that each of the **v_i** values must be assigned to between **L_i** and **M_i** variables.
- Then, **m** constraints

  ...

  $count(v_i, X, \#>=, L_i)$, $count(v_i, X, \#=<, M_i)$

  ...

  may be replaced by a single global constraint

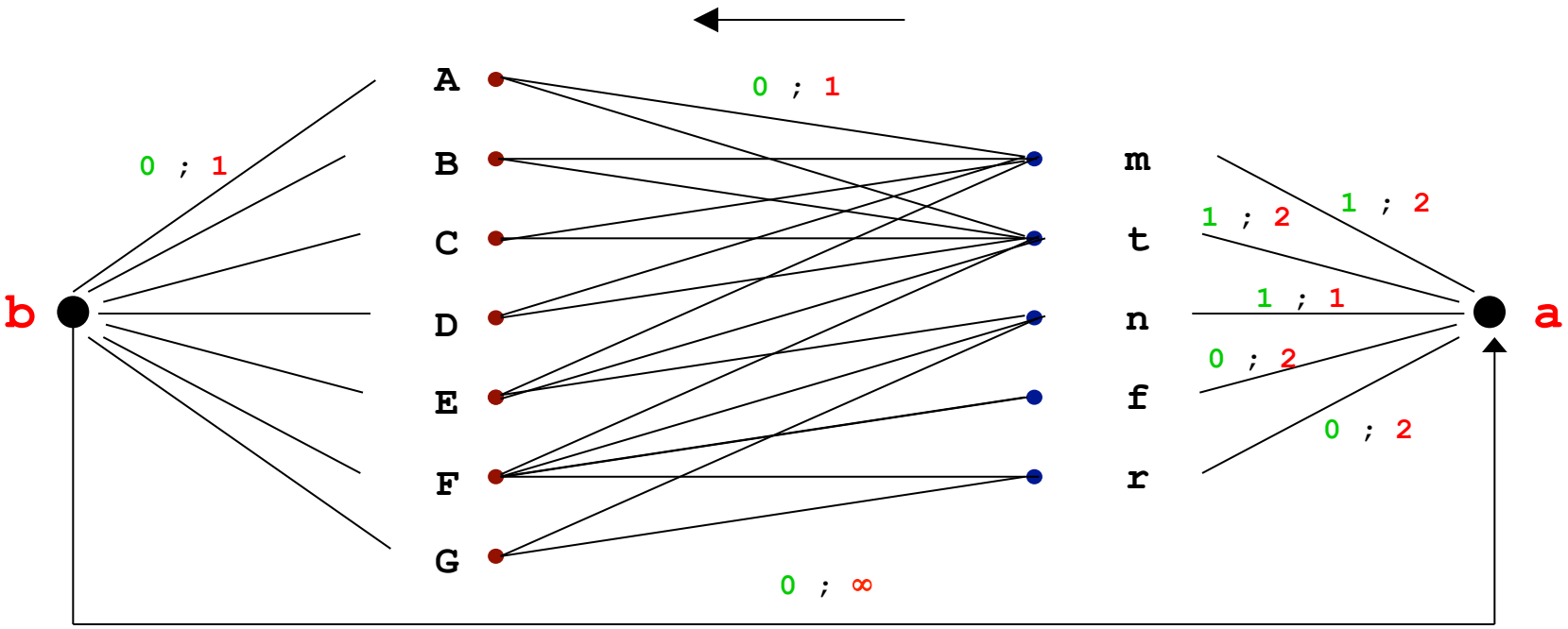
  $gcc([X_1, ..., X_k], [v_1, ..., v_m], [L_1, ..., L_m], [M_1, ..., M_m])$

* **J.-C. Régin, Generalized Arc-Consistency for Global Cardinality Constraint, Proceedings of AAAI-96, pp.209-215, 1996**

Global Constraints – Global Cardinality

- The constraint **gcc** is modelled based on a parallel with a directed graph (or network) with **maximum** and **minimum** capacities in the arcs and two additional nodes, a e b. For example:

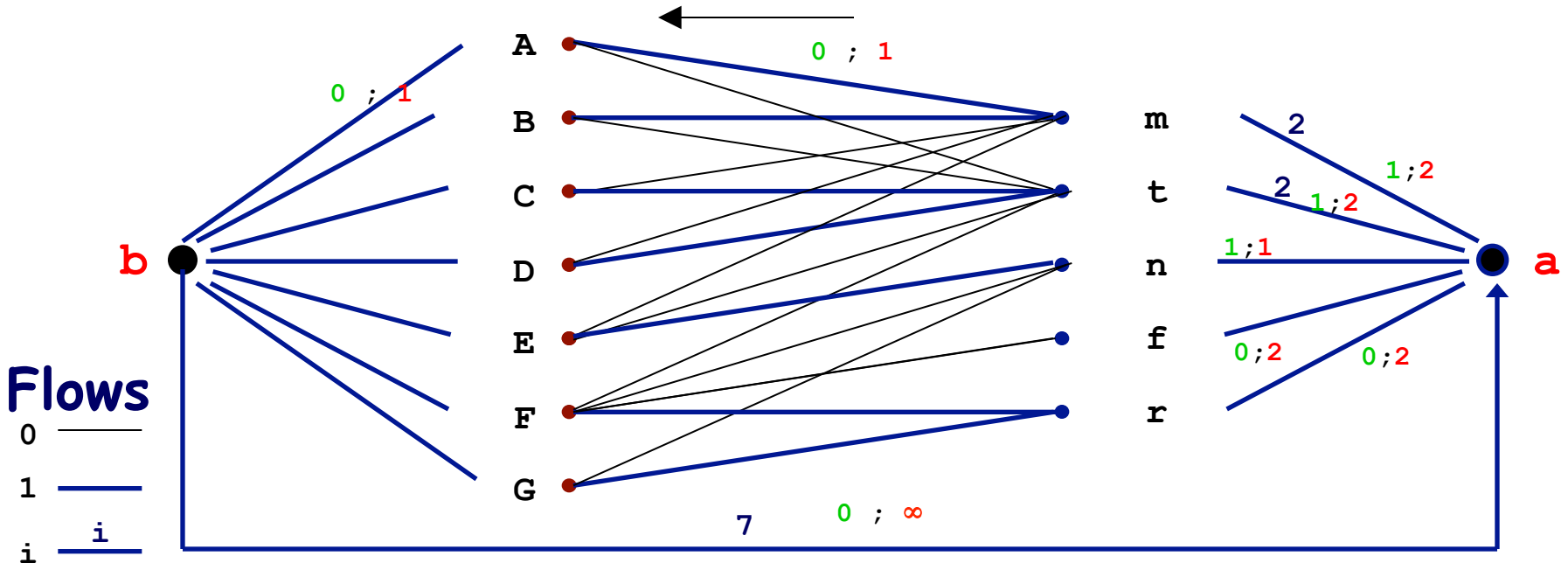
```
gcc([A,...,G],[m,t,n,f,r],[1,1,1,0,0],[2,2,1,2,2])
```



Global Constraints – Global Cardinality

- A solution for the gcc constraint, corresponds to a flow between the two added nodes, with a unitary flow in the arcs that link variables to value nodes. In these conditions it is valid the following

Theorem: A gcc constraint with k variables is satisfiable iff there is a maximal flow of value k, between nodes a and b, of the corresponding graph.



Global Constraints – Global Cardinality

Of course, being gcc a global constraint it is intended to

1. Obtain a maximum flow with value **k**, i.e. to show whether the problem is satisfiable.

2. Achieve generalised arc consistency, by eliminating the arcs between variables that are not used in any maximum flow solution, i.e. do not belong to any gcc solution.

3. When some arcs are pruned (by other constraints) redo 1 and 2 incrementally.

   In [Regi96] a solution is presented for these problems, together with a study on its polinomial complexity.

Global Constraints – Global Cardinality

1. **Obtain a maximal flow of value k**

- This optimisation problem may be efficiently solved by linear programming, that guarantees integer values in the solutions for the flows.

- However, to take into account the intended incrementality, the maximal flow may be obtained by using **increasing paths** in the **residual graph**, until no increase is possible.

- The residual graph of some flow **f** is again a directed graph, with the same nodes of the initial graph. Its arcs, with lower limit 0, have a residual capacity that accounts for the non used capacity in a flow **f**.

Global Constraints – Global Cardinality

Residual graph of some flow f

a. Given arc **(a,b)** with max capacity **c(a,b)** and flow **f(a,b)** such that **f(a,b) < c (a,b)** there is an arc **(a,b)** in the residual graph with residual capacity **cr(a,b) = c(a,b) - f(a,b).**

The fact that the arc directions are **the same** means that the flow may still increase in that direction by up to value **cr(a,b).**

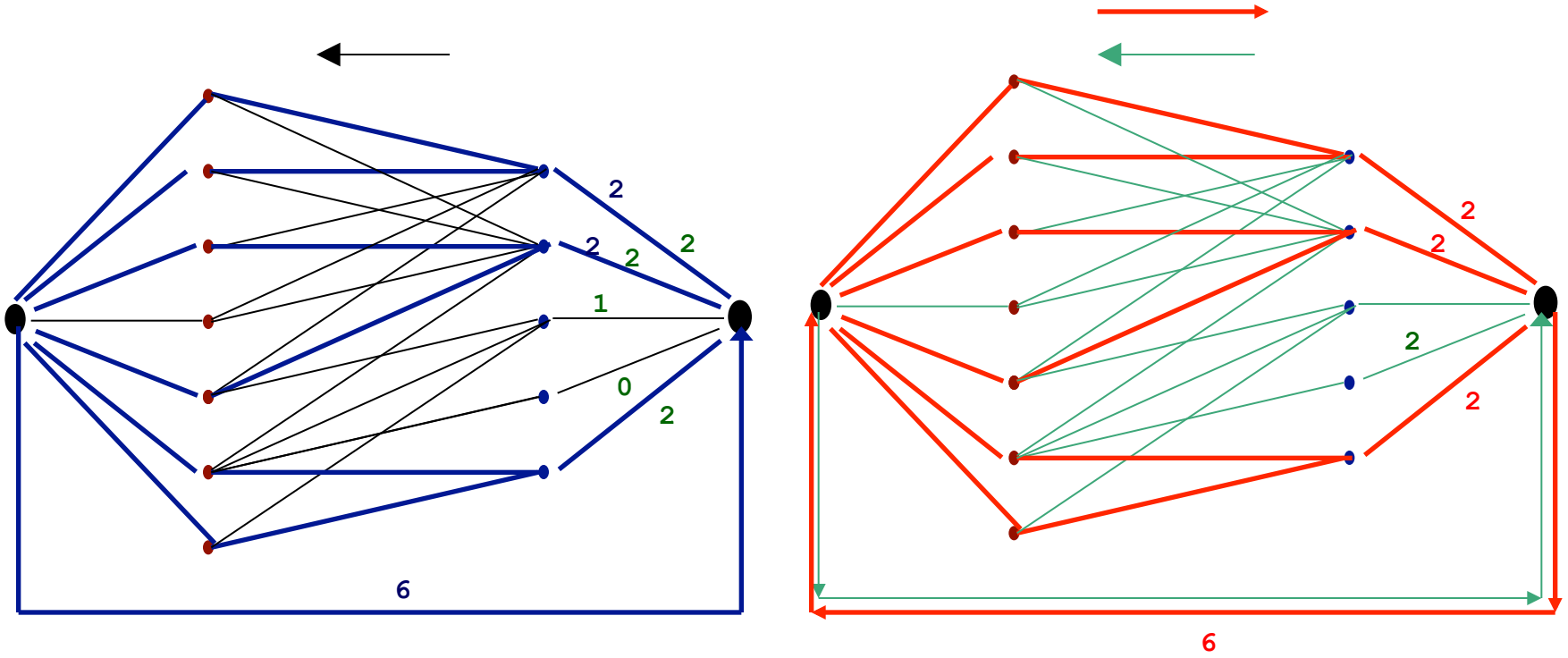
b. Given arc **(a,b)** with min capacity **l(a,b)** and flow **f(a,b)** such that **f(a,b) > l (a,b)** there is an arc **(b,a)** in the residual graph with residual capacity **cr(b,a) = f(a,b) - l(a,b).**

The fact that the arc directions are **opposed** means that the flow may decrease the initial flow by up to value **cr(b,a).**

Global Constraints – Global Cardinality

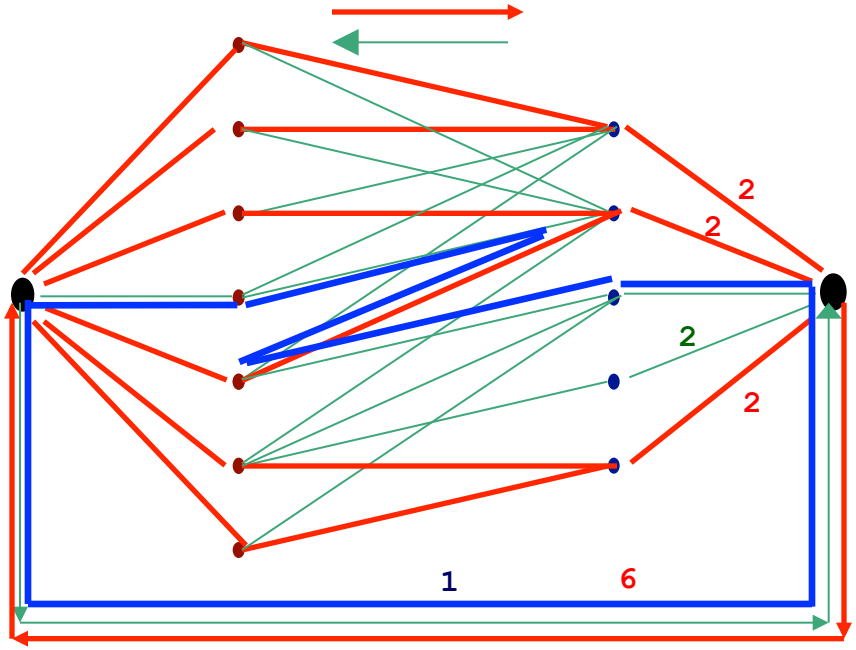
Example:

Given the following flow, with value 6 (lower than the maximal flow, which is of course 7) the following residual graph is obtained



Global Constraints – Global Cardinality

- If there exists an arc **(a,b)** (in the residual graph) whose flow is not the same as cr, there might be an overall increase in the flow between arcs a and b, if the arc belongs to an increasing path of the residual graph.
- In the example below, the path in **blue**, increases the flow in the original graph up to its maximum.



Global Constraints – Global Cardinality

- The incremental computation of a maximal flow is based on the following

Theorem: A flow f between two nodes is maximal iff there is no increasing path for f between the nodes.

Complexity

- The search for an increasing path may be obtained by a breadth-first search with complexity **$O(k+d+\delta)$,** where δ is the number of arcs between the nodes and their domains, with size **d**. As $\delta \approx kd$ this is the dominant term, and the complexity is **$O(\delta)$.**

- To obtain a maximum flow **k**, it is required to obtain **k** increasing paths, one for each of the **k** variables. The complexity of the method is thus **$O(k\delta)$.**

- As $\delta \leq kd$, the complexity to obtain a maximal flow k, starting from a null flow is thus **$O(k^2d)$.** It is of course less, if the starting flow is not null.

Global Constraints – Global Cardinality

- To eliminate arcs that correspond to assignments that do not belong to any possible solution, we use the following

Theorem:

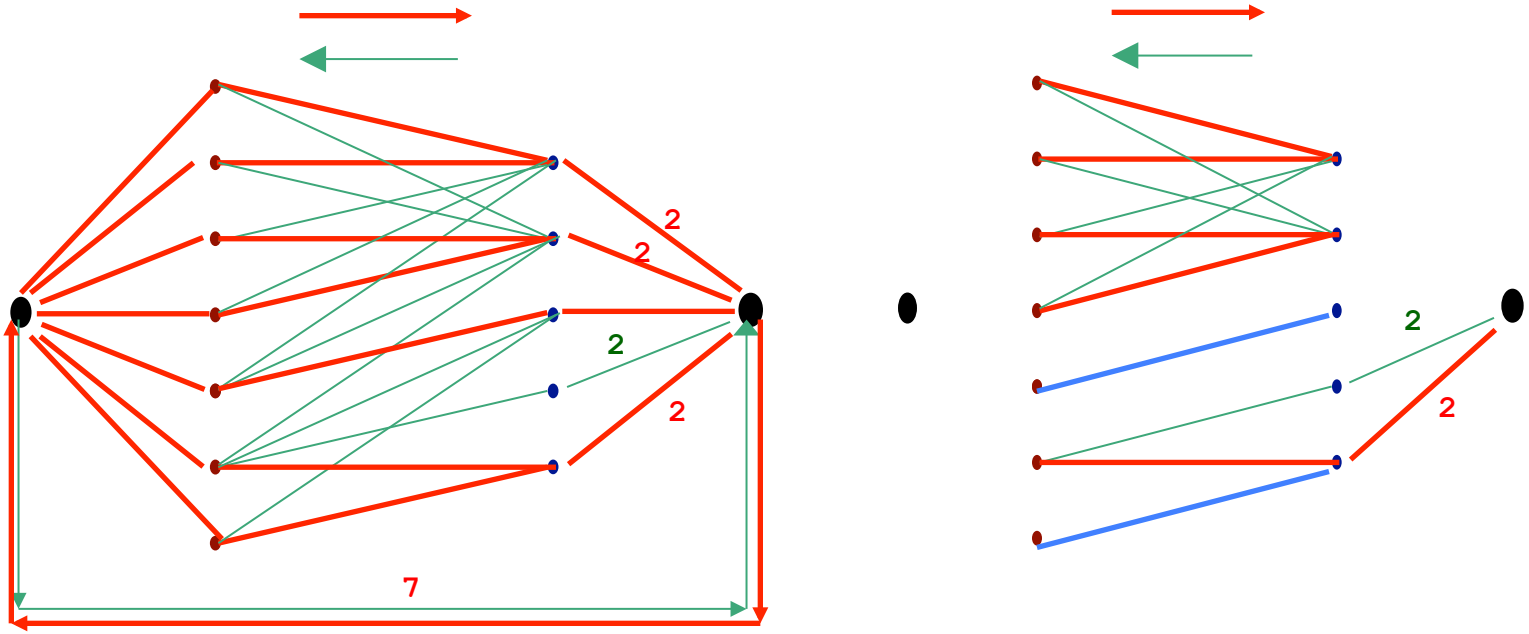
Let **f** be a maximum flow between nodes, **a** and **b**. For any other nodes **x** and **y**, flow **f(x,y)** is equal to any flow **f´(x,y)** induced between these nodes by another maximal flow **f', iff** the arcs **(x,y)** and **(y,x)** **are not included in a cycle (with more than 3 nodes*) that does not include nodes a and b**.

* these cycles correspond to a return through the same path

- Since the cycles considered do not include both a and b they will not change the (maximal) flow. Hence, if there are no cycles including nodes x and y, there are no increasing or decreasing paths through them that do not change the maximum flow. But then their flow will remain the same **for all maximal flows**.

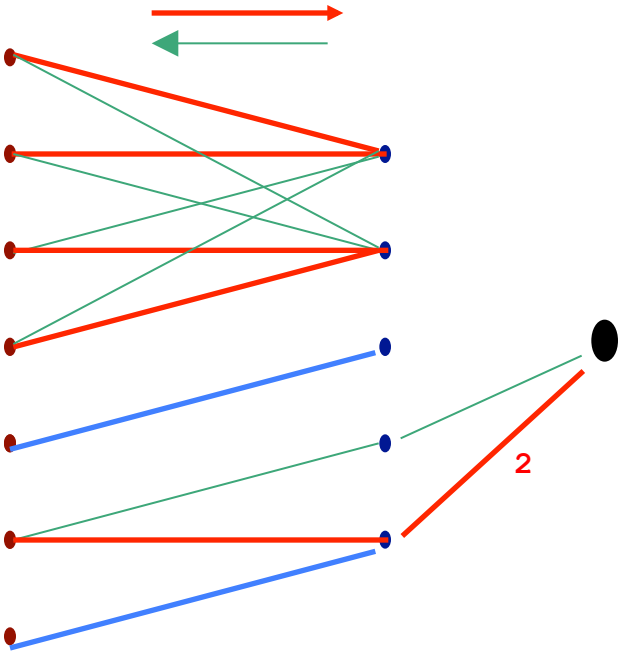
Global Constraints – Global Cardinality

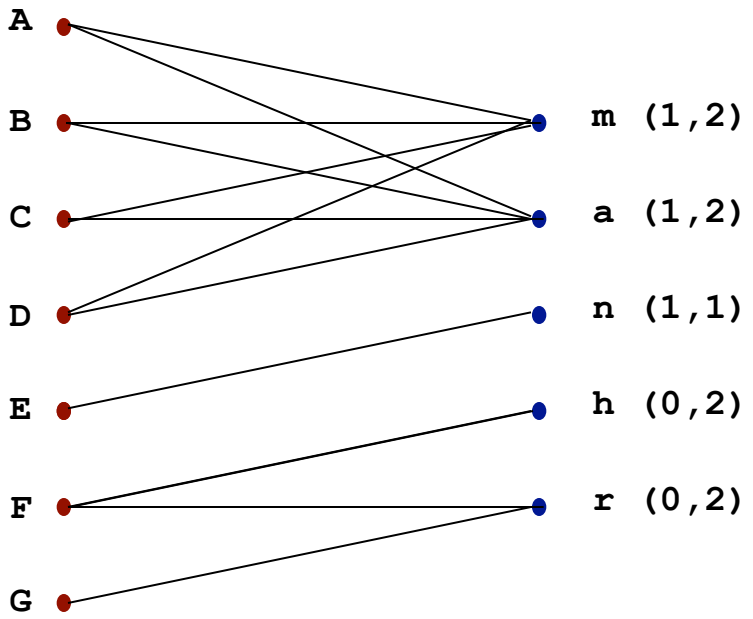
- Given the previous theorem, if no maximum flow passes in the arc between variable node **X** and value node **v**, then for no solution of the gcc constraint is **X = v**. This can be illustrated in the maximal graph shown below.
- In the residual graph, the only paths (with more than 3 nodes) that do not include nodes **a** and **b**, are shown at the right. In **blue**, are the arcs with non null flow in the initial situation. These are all the variable-value in a maximal flow, corresponding to possible solutions of constraint **gcc/4**.



Global Constraints – Global Cardinality

- We may compare the initial graph with that obtained after the elimination of the arcs.
- As expected, the latter fixes value **n** for variable **E**, and removes values **m**, **a** and **n** from variables **F** and **G**.





Global Constraints – Global Cardinality

Complexity

- Obtaining cycles with more than 3 nodes corresponds to obtain the subgraphs strongly connected, which can be done in **O(m+n)** for a graph with **m** nodes and **n** arcs. Here, **m = k+d+1** and **n ≤ kd+d**, hence a global complexity of

$$\mathbf{O(kd+2d+k+1) \approx O(kd)}$$

- When some constraint removes a value from a variable that was included in the current maximal flow, a new maximal flow may have to be recomputed, with complexity at most $\mathbf{O(k^2d)}$, so the complexity of using this incremental implementation of the **gcc**/4 constraint is

$$\mathbf{O(k^2d+kd) \approx O(k^2d).}$$

Global Constraints – Flow

- In CP2001, a global constraint was proposed to model and solve many network flow situations appearing in several problems, namely transport, communication or production chain applications.

- As the previous ones, the goal of this constraint is to be integrated with other constraints, as a part of a more general problem, but allowing the efficient filtering that would not be possible if the constraint were decomposed into simpler ones.

- Its use is described for problems of maximal flows, production planning and roastering

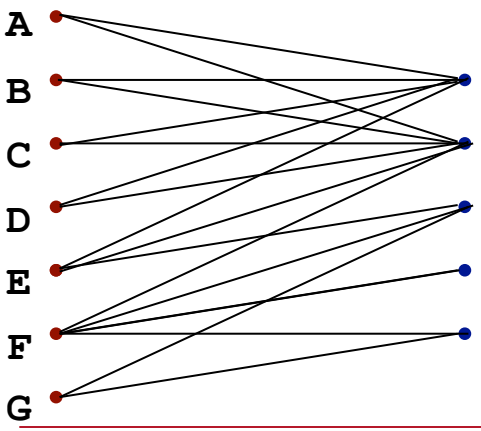
- See **A. Bockmayr, N. Pisaruk and A. Aggoun, Network Flow Problems in Constraint Programming, Proceedings of CP2001, LNCS 2239, Springer, 196-210.**

Global Constraints – Global Cardinality

- The COMET code for the illustration problem

```
range Rng = 1 .. 7;
range Dom = 1 .. 5;
set{int} d[Rng];
d[1] = {1,2};
d[2] = {1,2};
d[3] = {1,2};
d[4] = {1,2};
d[5] = {1,2,3};
d[6] = {1,2,3,4,5};
d[7] = {3,5};
```

```
Solver<CP> cp();
var<CP>{int} n[Rng](cp,Dom);
int lo[Dom]= [1,1,1,0,0];
int up[Dom]= [2,2,1,2,2];
solve<cp>{
  forall(i in Rng, j in Dom)
    if(! d[i].contains(j)) cp.post(n[i] != j);
  cp.post(cardinality(lo,n,up), onDomains);}
}
using{
  labelFF(n);
}
```




```
m (1,2)
a (1,2)
n (1,1)
h (0,2)
r (0,2)
```

Global Constraints – Circuit

- The previous global constraints may be regarded as imposing a certain "permutation" on the variables (e.g. when n all different variables take n values) .

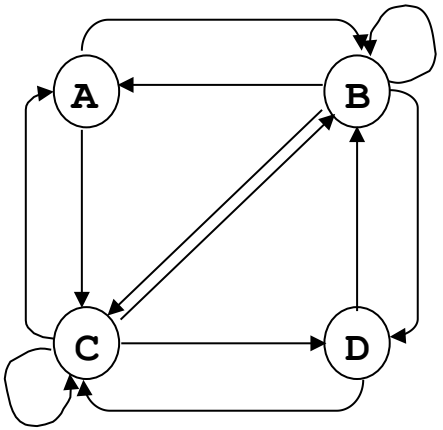
- In many problems, such permutation is not a sufficient constraint. It is necessary to impose a certain "ordering" of the variables.

- A typical situation occurs when there is a sequencing of tasks, with precedences between tasks, possibly with non-adjacency constraints between some of them.

- In these situations, in addition to the permutation of the variables, one must ensure that the ordering of the tasks makes a single **circuit**, i.e. there must be no sub-cycles.

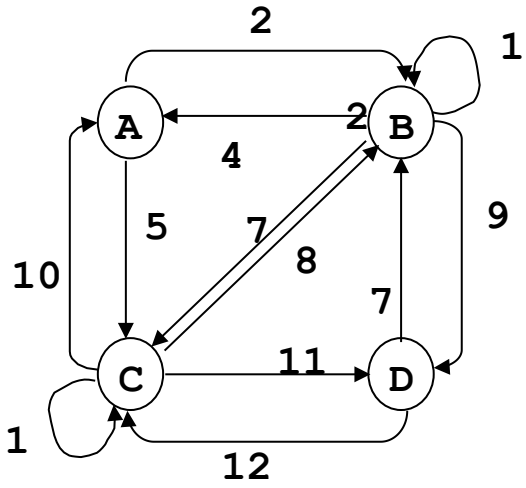
Global Constraints – Circuit

- These problems may be described by means of directed graphs, whose nodes represent tasks and the directed arcs represent precedences.



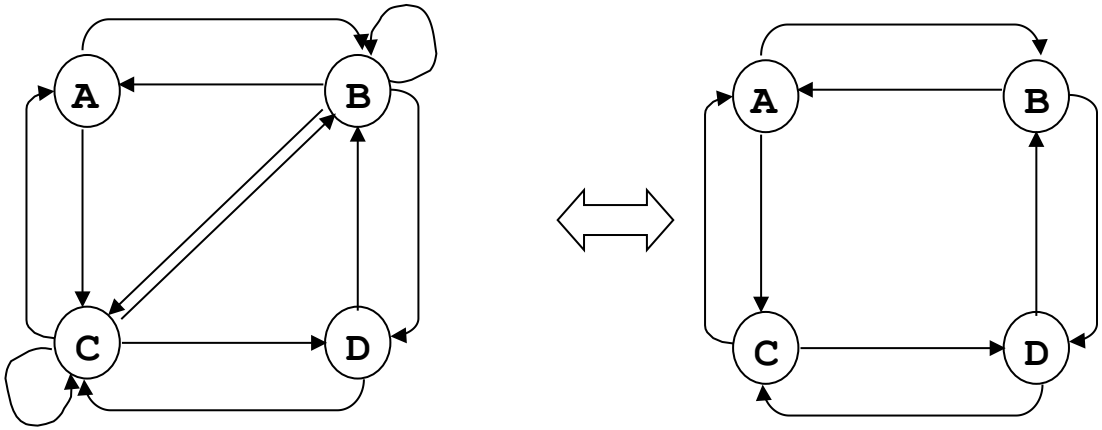
- The arcs may even be labelled by “features” of the precedences, namely transition times.

- This is a situation typical of several problems of the **travelling salesman** type.



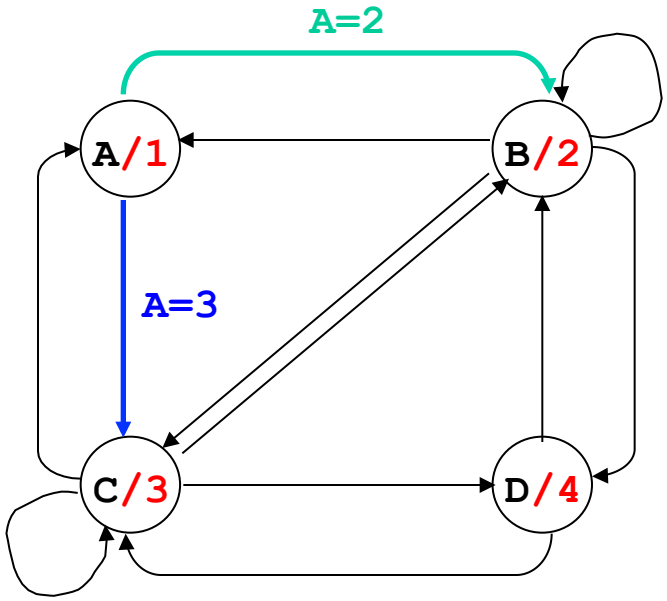
Global Constraints – Circuit

- **Filtering:** For these type of problems, the arcs that do not belong to any **hamiltonian circuit** should be eliminated.
- In the graph, it is easy to check that the only possible circuits are
- **A->B->D->C->A** and **A->C->D->B->A**.
- Certain arcs (e.g. **B->C, B->B**, ...), may not belong to any hamiltonian circuit and can be safely pruned.



Global Constraints - Circuit

- The pruning of the arcs that do not belong to any circuit is the goal of the global constraint circuit/1, available in several systems, namely SICStus and Comet.
- This constraint is applicable to a list / array of domain variables, where the domain of each corresponds to the arcs connecting that variable to other variables, denoted by the order in which they appear in the list.

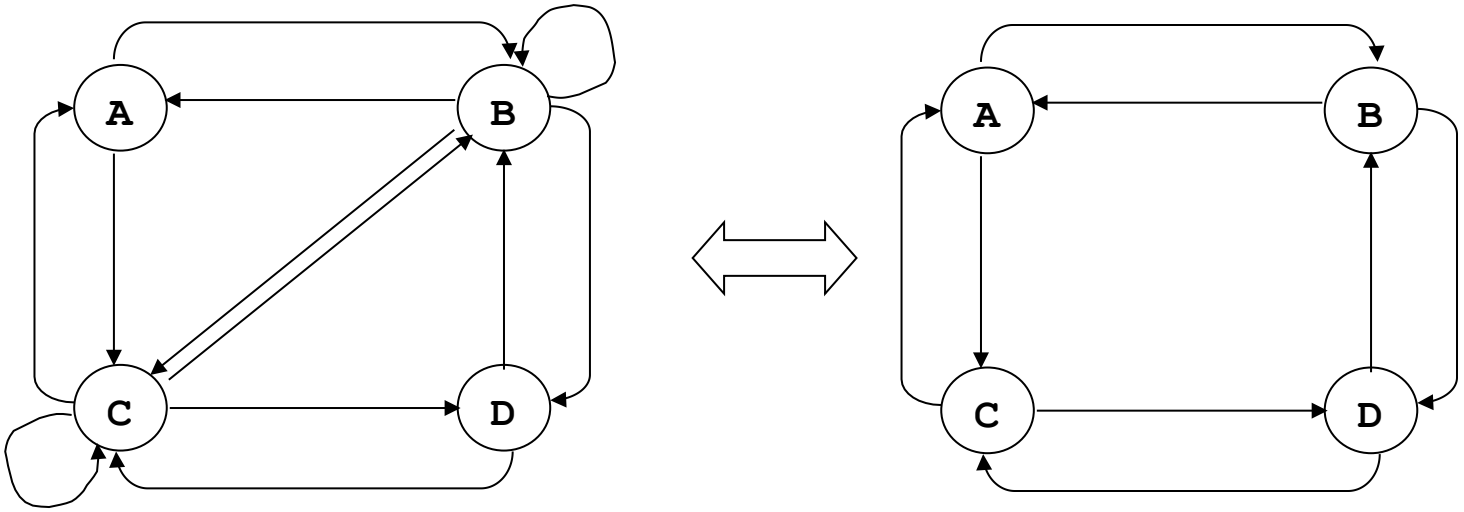



```
circuit([A,B,C,D])
```

Global Constraints – Circuit

- Global constraint **circuit/1**, incrementally achieves the pruning of the arcs not in any hamiltonian circuit. In SICStus

```
- A, D in 2..3, B, C in 1..4
- circuit([A,B,C,D]).
```



... the following prunning is achieved

```
A in 2..3, B in 1,2,3,4, C in 1,2,3,4, D in 2..3,
```

- since the possible solutions are

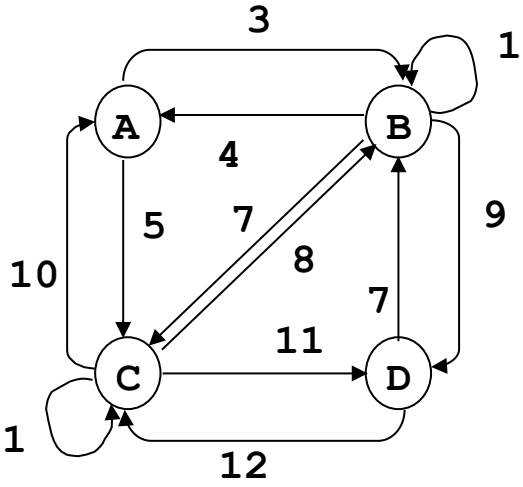
```
- [A,B,C,D] = [2,4,1,3] and [A,B,C,D] = [3,1,4,2]
```

Global Constraints – Element

- Often a variable not only has its values constrained by the values of other variables, but it is actually defined **conditionally** in function of these values.

- For example, the value X from the arc that leaves node A, depends on the arc chosen:

```
if A = 2 then X = 3,
if A = 3 then X = 5;
otherwise X = undefined
```



- The disjunction implicit in this definition raises, as well known, problems of efficiency to constraint propagation.
- The value of **X** may only be known upon labelling of variable **A**. Until then, a naïf handling of this type of conditoinal constraint would infer very little from it.
- However, if other problem constraints impose, for example, **X < 4**, an efficient handling of this constraint would impose not only **X = 3** but also **A = 2**.

Global Constraints – Element

- The efficient handling of this type of disjunctions is the goal of global constraint **element**, available in SICStus and CHIP.

$$\textbf{element(I, [V}_1\textbf{,V}_2\textbf{,...,V}_n\textbf{], X)}$$

- In this constraint, **X** is a variable with domain **1..n**, and both **V** and the $\mathbf{V_i s}$ are either finite domain constraints or constants. The semantics of the constraint can be expressed as the equivalence

$$\mathbf{X = V_i}$$

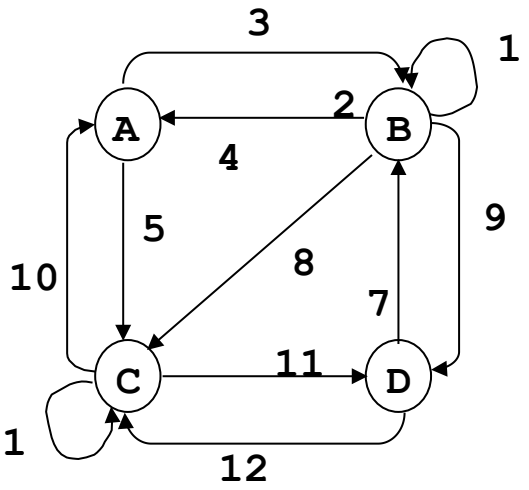
- From a propagation viewpoint, this constraint imposes arc consistency in **X** and bounds consistency in **V**. It is particularly optimised for when all $\mathbf{V_i s}$ are ground.
- In COMET, the syntax is closer to the definition above. Given an array **v** of integers, and domain variables **i** and **x**, we have

```
cp.post( v[i] == x );
```

Global Constraints – Circuit + Element

- Global constraints may be used together. In particular, constraints element and circuit may implement the travelling salesman:
- For some graph, determine an hamiltonian circuits whose **Cost** does not exceed a (given) **Max** .

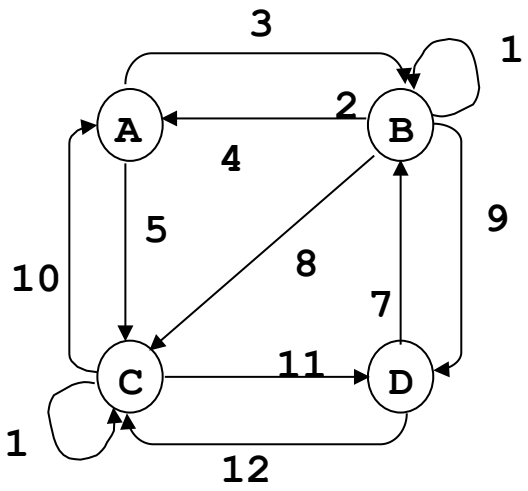
```
circ([A,B,C,D], Max, Cost):-
    A in 2..3, B in 1..4,
    C in {1}\/{3,4}, D in 2..3,
    circuit([A,B,C,D]),
    element(A,[ _, 3, 5, _],Ca),
    element(B,[ 2, 1, 8, 9],Cb),
    element(C,[10, _, 8,11],Cc),
    element(D,[ _, 7,12, _],Cd),
    Cost #= Ca + Cb + Cc + Cd,
    Cost #=< Max,
    labeling([],[A,B,C,D]).
```



Global Constraints – Circuit + Element

Example: The Comet code for a simple TSP (existence)

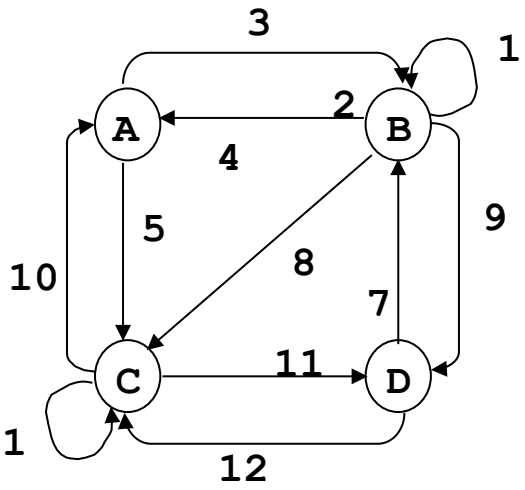
```
range Rng = 1 .. 4;

Solver<CP> cp();

var<CP>{int} g[Rng](cp,Rng);

solveall<cp>{
  cp.post(g[1]!= 1);
  cp.post(g[1]!= 4);
  cp.post(g[4]!= 1);
  cp.post(g[4]!= 4);
  cp.post(circuit(g));
}
using {
  labelFF(g);
}
```



Global Constraints – Circuit + Element

Example: The **Comet** code for a simple TSP (satisfaction)

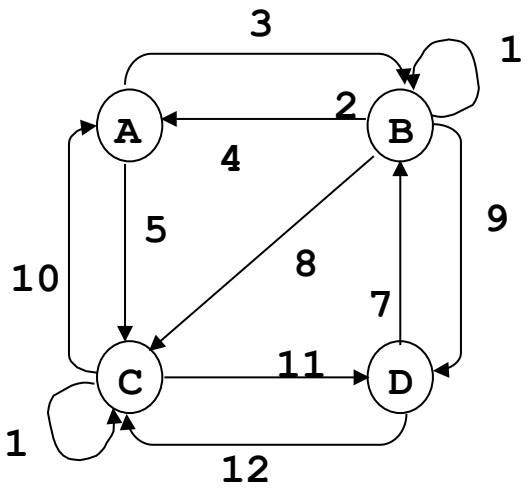
```
range Rng = 1 .. 4;
int x[Rng,Rng];
x[1,1] = 0; x[1,2] = 3; x[1,3] = 5; x[1,4] = 0;
x[2,1] = 4; x[2,2] = 1; x[2,3] = 7; x[2,4] = 9;
x[3,1] =10; x[3,2] = 8; x[3,3] = 1; x[3,4] =11;
x[4,1] = 0; x[4,2] = 7; x[4,3] =12; x[4,4] = 0;
Solver<CP> cp();
var<CP>{int} g[Rng](cp,Rng);
var<CP>{int} c[Rng](cp,0..20);
solveall<cp>{
  /* remove values from domains */
  cp.post(circuit(g));
  forall(i in 1..4)
     cp.post(x[i,g[i]] == c[i]);
  cp.post(sum(i in 1..4) c[i]) <= Max;}
using {
  labelFF(g);
}
```



Global Constraints – Circuit + Element

Example: The Comet code for a simple TSP (optimisation)

```
range Rng = 1 .. 4;
int x[Rng,Rng];
/* x[1,1] = 0; …  ; x[4,4] = 0; */
Solver<CP> cp();
var<CP>{int} g[Rng](cp,Rng);
var<CP>{int} c[Rng](cp,0..20);
minimize<cp>
   sum(i in 1..4) c[i]
subject to{
   /* remove values from domains */
   cp.post(circuit(g));
   forall(i in 1..4)
      cp.post(x[i,g[i]] == c[i]);
using {
  labelFF(g);
}
```



Global Constraints – Global Sequence

- In some applications it is not simply intended to impose cardinality constraints on a set of variables, i.e. that in this set of variables each value does not appear more than a certain number of times.

- It is additionally required that these variables are considered in certain **accepted sequences**, namely that guarantee that **each value does not appear more than a certain number of times in every sub-sequence of a given size**.

- Hence, the goal is to implement an efficient **global sequence constraint**, **gsc/5**

```
gsc(X, V, Seq, Lo, Up)
```

with the following semantics:

Given a sequence of variables **X**, in each subsequence of **Seq** variables, **Lo** and **Up** represent the minimum/maximum number of times they may take values from the list **V**.

Global Constraints – Global Sequence

Example (Car sequencing):

The goal is to manufacture in an assembly line 10 cars with different options (1 to 5) shown in the table. Given the assembly conditions of option i, for each sequence of n_i cars, only m_i cars can have that option installed as shown in the table below.

		cars									
option	**capacity**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	**1 / 2**	**X**						**X**		**X**	
2	**2 / 3**			**X**		**X**				**X**	
3	**1 / 3**	**X**						**X**			
4	**2 / 5**	**X**	**X**			**X**					
5	**1 / 5**			**X**							
	Configuration	**1**	**2**	**3**		**4**		**5**		**6**	

For example, in any sequence of 5 cars, only 2 may have option 4 instaled.

Global Constraints – Global Sequence

- Given the similarity of the constraints, it was proposed in [RePu97] an efficient implementation of the **global sequencing constraint**, **gsc/5**, based on the **global cardinality constraint**, gcc/4.

- A better algorithm was proposed in CP2006 for the implementation of the sequence constraint, with reference

  Willem-Jan van Hoeve, Gilles Pesant, Louis-Martin Rousseau, and Ashish Sabharwal, Revisiting the Sequence Constraint, Proceedings of CP'06, LNCS vol. 4204, Springer, 620-634, 2006

- The next slide presents a possible modeling of the problem in COMET .

		cars									
option	capacity	1	2	3	4	5	6	7	8	9	10
1	1 / 2	X						X		X	
2	2 / 3			X		X				X	
3	1 / 3	X						X			
4	2 / 5	X	X			X					
5	1 / 5			X							
	Configuration	1	2	3		4		5		6	

Global Constraints – Global Cardinality

- The COMET code for the illustration problem

		cars									
option	capacity	1	2	3	4	5	6	7	8	9	10
1	1 / 2	X						X		X	
2	2 / 3			X		X				X	
3	1 / 3	X						X			
4	2 / 5	X	X			X					
5	1 / 5			X							
	Configuration	1	2	3		4		5		6	

```
range Rng = 1 .. 10;
range Dom = 0 .. 5;
range Opt = 1 .. 5;
int dem[Dom]= [1,1,2,2,2,2];
int lo[Opt] = [1,2,1,2,1];
int up[Opt] = [2,3,3,5,5];
set{int} cfg[Opt] =[{0,4,5},{2,3,5},{0,4},{0,1,3},{2}];

Solver<CP> cp();
var<CP>{int} seq[Rng](cp,Dom);
solve<cp>{
  forall(o in Opt: o >= 1)
     cp.post(sequence(seq, dem, lo[o], up[o], cfg[o]));
}
using {
  labelFF(seq);
}
```

Cumulative Constraints

- In general, edge finding require more sophisticated techniques, namely in problems combining scheduling and resource management.
- In fact, if many units of a resource are available, then more than one of the tasks that use these resources may execute simultaneously. All that is needed is that the number of resources required at any given time does not exceed the existing resources.
- This is the semantics of the cumulative constraint, initially introduced in CHIP, and which had an enormous impact in the area of constraint programming.
- Let **S** be the set of starting times of **n** tasks S_i, **T** be the set of their durations T_i and **R** the set of the number of resources of a given type required by the tasks, R_i. Denoting by

  $a = \min_i(S_i)$; $b = \max_i(S_i+T_i)$;

  $R_{i,k} = R_i$ if $T_i =< t_k =< T_i+D_i$ or 0 otherwise.

  then

  $$\text{cumulative}(S,T,R,L) \Leftrightarrow \forall_{k\in[a,b]} \sum_i R_{i,k} \leq L$$

Cumulative Constraints

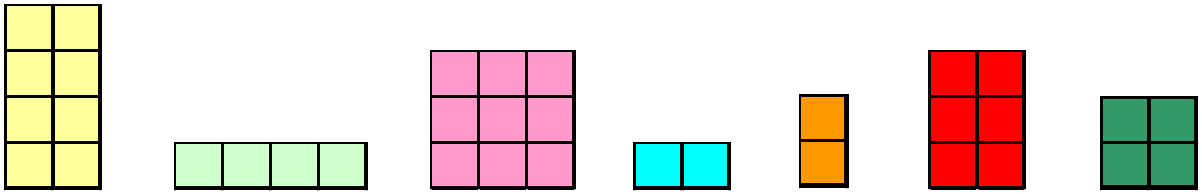
Example:

Take 7 tasks (A a G) with the duration and resource consumption (e.g. number of workers needed to carry them out) specified in the following lists

T = [2 , 4 , 3 , 2 , 1 , 2 , 2] ; R = [4 , 1 , 3 , 1 , 2 , 3 , 2]

Find whether the tasks may all be finished in a given due time Tmax, assuming there are Rmax resources (e.g. Workers) available at all times.

Graphically, the tasks can be viewed as



Many instances of the problem may be modelled by a simple constraint

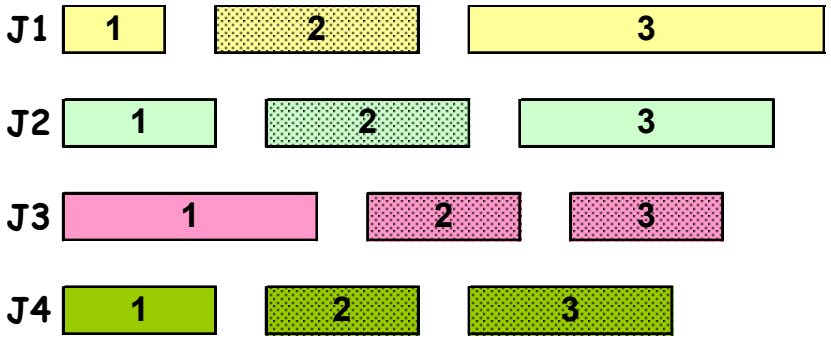
cumulative(S,T,R,L)

and some additional constraints regarding the domains of the starting time variables,**S**.

Cumulative Constraints: Job Shop

- The job shop problem consists of executing the different tasks of several jobs without exceeding the available resources.
- Within each job, there are several tasks, each with a duration. Within each job, the tasks have to be performed in sequence, possibly respecting mandatory delays between the end of a task and the start of the following task.
- Tasks of different jobs are independent, except for the sharing of common resources (e.g. machines). Each task must be executed in one machine of a certain type. The number of machines of each type is limited.
- A simple instance of the problem (with 2 machines) is given in the table below (with the corresponding graphic representation).

		Tasks Y		
	Z, D	1	2	3
Jobs X	1	1 , 2	2 , 4	1 , 7
	2	1 , 3	2 , 4	1 , 5
	3	1 , 5	2 , 3	2 , 3
	4	1 , 3	2 , 3	2 , 4



Cumulative Constraints: Job Shop

- This instance was proposed in the book Industrial Scheduling [MuTh63]. For 20 years no solution was found that optimised the “makespan”, i.e. the fastest termination of all tasks.
- Around 1980, the best solution was 935 (time units). In 1985, the optimum was lower bounded to 930. In 1987 the problem was solved with a highly specialised algorithm, that found a solution with makespan 930.
- With the cumulative/4 constraint, in the early 1990’s, the problem was solved in 1506 seconds (in a SUN/SPARC workstation).

		Tasks Y									
	Z, D	1	2	3	4	5	6	7	8	9	a
Jobs X	1	1, 29	2, 78	3, 9	4, 36	5, 49	6, 11	7, 62	8, 56	9, 44	a, 21
	2	1, 43	3, 90	5, 75	a, 11	4, 69	2, 28	7, 46	6, 46	8, 72	9, 30
	3	2, 91	1, 85	4, 39	3, 74	9, 90	6, 10	8, 12	7, 89	a, 45	5, 33
	4	2, 81	3, 95	1, 71	5, 99	7, 9	9, 52	8, 85	4, 98	a, 22	6, 43
	5	3, 14	1, 6	2, 22	6, 61	4, 26	5, 69	9, 21	8, 49	a, 72	7, 53
	6	3, 84	2, 2	6, 52	4, 95	9, 48	a, 72	1, 47	7, 65	5, 6	8, 25
	7	2, 46	1, 37	4, 61	3, 13	7, 32	6, 21	a, 32	9, 89	8, 30	5, 55
	8	3, 31	1, 86	2, 46	6, 74	5, 32	7, 88	9, 19	a, 48	8, 36	4, 79
	9	1, 76	2, 69	4, 76	6, 51	3, 85	a, 11	7, 40	8, 89	5, 26	9, 74
	a	2, 85	1, 13	3, 61	7, 7	9, 64	a, 76	6, 47	4, 52	5, 90	8, 45

Placement Problems

- Several applications of great (economic) importance require the satisfaction of placement constraints, i.e. the determination of where to place a number of components in a given space, without overlaps.
- The non overlapping of the rectangles defined by their X_i and Y_i origins and their widths W_i (X-sizes) and heights H_i (Y sizes) is guaranteed, as long as one of the constraints below is satisfied (for rectangles 1 and 2)

$X_1+W_1 =< X_2$	Rectangle 1 is	left of	Rectangle 2
$X_2+W_2 =< X_1$	Rectangle 1 is	rigth of	Rectangle 2
$Y_1+H_1 =< Y_2$	Rectangle 1 is	below	Rectangle 2
$Y_2+H_2 =< Y_1$	Rectangle 1 is	above	Rectangle 2

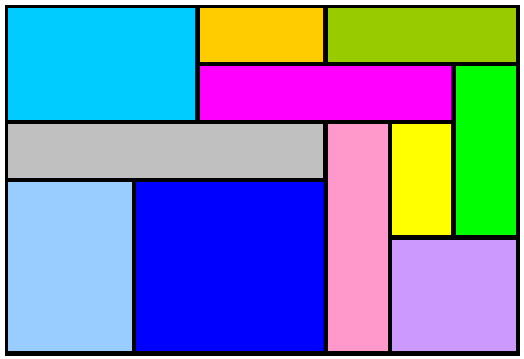
- As explained before, rather than commiting to one of these conditions, and change the commitment by backtracking, a better option is to adopt a least commitment approach, for implementing such disjunctive constraint.

Placement Problems

- The results obtained show well the importance of using the redundant cumulative/ 4 constraints. Testing the program presented with and without these constraints, the following results are obtained

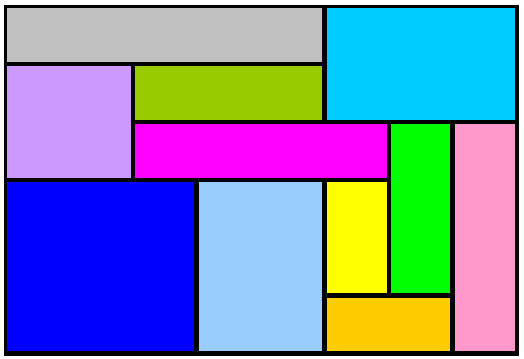
in 16 ms

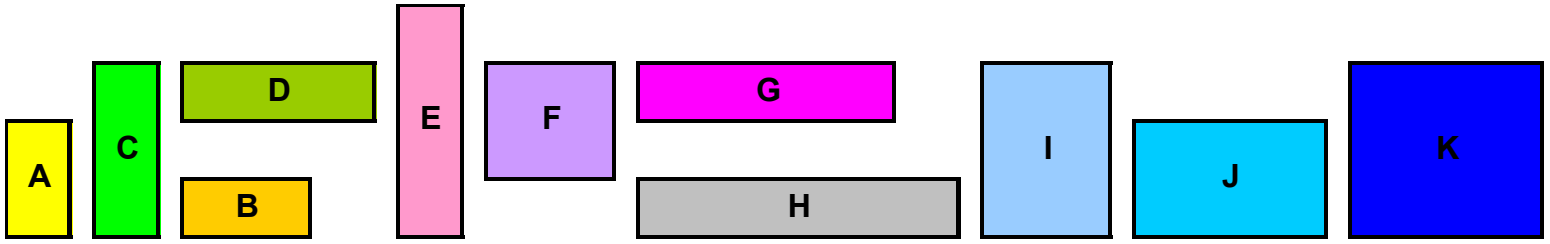
with cumulative



in 5.407 s

without cumulative





Global Constraints – Global Sequence

More about Global Constraints

- A Catalogue of Global constraints is maintained by Beldiceani at Nantes.

  (URL: **http://www.emn.fr/x-info/sdemasse/gccat/** - but see next page)

- The pdf verson of the catalogue – download from

  http://www.emn.fr/x-info/sdemasse/gccat/doc/catalog.pdf

Global Constraint Catalog

Corresponding author: Nicolas Beldiceanu nicolas.beldiceanu@emn.fr
Online version: Sophie Demassey sophie.demassy@emn.fr

html / 2009-12-16

Search by: name Keyword Meta-Keyword Argument Pattern ...

About the catalogue

The catalogue presents a list of 348 global constraints issued from the literature in constraint programming and from popular constraint systems. The semantic of each constraint is given together with a description in terms of graph properties and/or automata.

The catalogue is periodically updated by Nicolas Beldiceanu, Mats Carlsson and Jean-Xavier Rampon. Feel free to contact the first author for any questions about the content of the catalogue.

Download the Global Constraint Catalog in pdf format:

- the last working version (2010-01-29) (about 15 Mo)
- the edited version (2005-08) (Sicstus technical report, about 7 Mo)

Example: all-different

Global Constraints in SICStus

- A number of global constraints are available in SICStus, namely

count(+Val,+List,+RelOp,?Count)

where Val is an integer, List is a list of integers or domain variables, Count an integer or a domain variable, and RelOp is a relational symbol.

True if N is the number of elements of List that are equal to Val and N RelOp Count.

global_cardinality(+Xs,+Vals)

global_cardinality(+Xs,+Vals,+Options)

where Xs = [X1, . . . ,Xd] is a list of integers or domain variables, and Vals = [K1 − V1, . . . ,Kn − Vn] is a list of pairs where each key Ki is a unique integer and Vi is a domain variable or an integer. True if every element of Xs is equal to some key and for each pair Ki − Vi, exactly Vi elements of Xs are equal to Ki.

If either Xs or Vals is ground, and in many other special cases, global_cardinality/[2,3] maintains domain-consistency, but generally, interval-consistency cannot be guaranteed. A domain-consistency algorithm [Regin 96] is used, roughly linear in the total size of the domains.

Global Constraints in SICStus

- **table(+Tuples,+Extension, [+Options])**
- **case(+Template, +Tuples, +Dag, [+Options])**
- **all_different(+Variables [, +Options])**
- **all_distinct(+Variables [, +Options])**
- **nvalue(?N, +Variables)**
- **assignment(+Xs, +Ys [, +Options])**
- **circuit(+Succ)**
- **circuit(+Succ, +Pred)**
- **sorting(+Xs,+Ps,+Ys)**
- **cumulative(+Tasks)**
- **cumulative(+Tasks,+Options)**
- **disjoint1(+Lines, [+Options])**
- **disjoint2(+Rectangles, [+Options])**
- **lex_chain(+Vectors,[+Options])**
- **geost(+Objects,+Shapes [,+Options,+Rules])**
- **automaton(Signature, SourcesSinks, Arcs)**

Global Constraints in Comet (1)

A number of global constraints are also available in **Comet**. Two examples below:

- `Constraint<CP> alldifferent(var<CP>{int}[])`
  - This function creates an alldifferent constraint which requires an array of expressions be given different values.
  - Possible consistency levels are onValues (default), onDomains.
  - The relaxed version of this constraint is the atLeastNValue
  - **x:** an array of expressions (not containing aggregate operators).
  - **Module:** *CP*

- `Constraint<CP> circuit(var<CP>{int}[])`
  - This function creates a circuit constraint which holds if all the variables are assigned values representing an Hamiltonian circuit.
  - Let x be an array x[l],x[l+1],... ,x[u] and R be l..u. Each element in R denotes a vertex and x[i] denotes the successor of vertex i. The constraint succeeds if the graph so formed is an hamiltonian circuit, i.e. , if it visits every vertex exactly once and connects all vertices.
  - **x:** an array
  - **Module:** *CP*

Global Constraints in Comet (2)

Many more global constraints are available in Comet, namely

- **Constraint<CP> allDisjoint(var<CP>{int}[][])**
- **Constraint<CP> alldifferent(var<CP>{int}[])**
- **Constraint<CP> atleast(int[], var<CP>{int}[])**
- **Constraint<CP> atmost(int[], var<CP>{int}[])**
- **Constraint<CP> bijection(var<CP>{int}[])**
- **Constraint<CP> binaryKnapsackAtleast(var<CP>{int}[], int[], int)**
- **Constraint<CP> binpackingLB(var<CP>{int}[], int[], var<CP>{int}[])**
- **Constraint<CP> cardinality(int[], var<CP>{int}[])**
- **Constraint<CP> circuit(var<CP>{int}[])**
- **Constraint<CP> deviation(var<CP>{int}[], int, var<CP>{int})**
- **Constraint<CP> exactly(int[], var<CP>{int}[])**

Global Constraints in Comet (3)

Many more global constraints are available in Comet, namely (cont)

- **Constraint<CP> costregular(var<CP>{int}[], Automaton<CP>, var<CP>{int}, int[,,])**
- **Constraint<CP> cumulative<CP>(int, int, int, var<CP>{int}[], var<CP>{int}[], var<CP>{int}[])**
- **Constraint<CP> lexleq(var<CP>{int}[], var<CP>{int}[])**
- **Constraint<CP> lightBinaryKnapsack(var<CP>{int}[], int[], var<CP>{int})**
- **Constraint<CP> minAssignment(var<CP>{int}[], float[,], var<CP>{float})**
- **Constraint<CP> multiknapsack(var<CP>{int}[], int[], var<CP>{int}[])**
- **Constraint<CP> noCycle(var<CP>{int}[])**
- **Constraint<CP> operator ~>(expr<CP>{boolean}, Constraint<CP>)**
- **Constraint<CP> operator ~>(expr<CP>{boolean}, expr<CP>{boolean})**

Global Constraints in Comet (4)

Many more global constraints are available in Comet, namely (cont)

- **Constraint<CP> regular(var<CP>{int}[], Automaton<CP>)**
- **Constraint<CP> sequence(var<CP>{int}[], int[], int, int, set{int})**
- **SoftAtLeast<CP> softAtLeast(int[], var<CP>{int}[], var<CP>{int})**
- **SoftAtMost<CP> softAtMost(int[], var<CP>{int}[], var<CP>{int})**
- **SoftCardinality<CP> softCardinality(int[], var<CP>{int}[], int[], var<CP>{int})**
- **Constraint<CP> spread(var<CP>{int}[], int, var<CP>{int})**
- **Constraint<CP> stretch(int[], var<CP>{int}[], int[])**
- **Constraint<CP> softStretch(int[], var<CP>{int}[], int[], var<CP>{int})**
- **Constraint<CP> table(var<CP>{int}, var<CP>{int}, var<CP>{int}, Table<CP>)**