Constraint Programming:
Global Constraints and Reified Constraints

Université de Nantes

Objectives

- reified constraints
  - why are they useful ?
  - important modelling and solving efficiency
  - some well-known examples
- global constraints :
  - why are they useful ?
  - important modelling and solving efficiency
  - some well-known examples

Reified constraints

Reified constraints (1)

form of the constraint :

$$C_1 \leftrightarrow C_2$$

where C_1 and C_2 are two constraints.

Semantics : the constraint C_1 is equivalent to the constraint C_2, i.e., C_1 and C_2 have the same truth value

- C_1 is violated iff C_2 is violated
- C_1 is satisfiable iff C_2 is
- otherwise, $C_1 \leftrightarrow C_2$ is suspended and woken-up when one of the variables of C_1 or C_2 is modified.

Reified constraints (2)

- implementation : difficult
- must be able to test whether a constraint is “implied” by the *store* of constraints
  $\rightarrow$ notion of *entailment*
- in practice :
  reification limited to some “primitives” constraints
- also exists as : $C_1 \rightarrow C_2$

Reified constraints (ECLiPSe)

constraint of the form :

$$C_1 \quad \# <=> \quad C_2$$

where C_1 and C_2 are two arithmetic constraints.

semantics :

- C_1 is violated iff C_2 is
- C_1 is satisfiable iff C_2 is
- otherwise, $C_1 \quad \# <=> \quad C_2$ is suspended and woken-up as soons as the domain of one of the variables of C_1 or C_2 is modified.

reified constraints (GNU Prolog)

constraint of the form :

$$B \ \# <=> \ \ C$$

where B is a Boolean variable (domain $[0, 1]$) and C is a constraint

semantics : verified if the *equivalence* is verified (the constraint C can be violated)

- B is 0 iff C is false
- B is 1 iff C is true
- if C is unknown, $B \in \{0, 1\}$

Reified constraints : example

Example of use : either

```
% C1 true or C2 true, but not both
either(C1,C2):-
          B1 #<=> C1,
          B2 #<=> C2,
          B1 + B2 #=1.
```

Reified constraints : example

Example of use : absolute value

```
% AbsT is either +T or -T
abs(T,AbsT):-
        T #>= 0 #<=> AbsT #= T,
        T #< 0  #<=> AbsT #= -T.
```

Reified constraints : example

Example of use : absolute value 2

```
% AbsT is either +T or -T
abs(T,AbsT):-
          T #>= 0 #<=> B,
          AbsT #= 2*B*T -T.
```

Global constraints

Global constraints

Motivations :

- reduce the gap between constraints issued from modelling, and constraints available in the language
- to ease formulating complexe global conditions that are not easily formulated with the structures of the language
- to increase domain reduction capacity (stronger consistency, problem of (n,k)-consistencies)

Global constraints

Setting up :

- constraints that appear often in practice
  (all_diff, cycle, . . .)
- constraints that are the key-point of a type of application
  (specific flow constraint, max-flow, . . .)
- need specific algorithm for domain reduction
  $\rightarrow$ efficiency, and thus usefulness depending on the algorithm

alldiff/2 : example (1)

- sequencing problem :

speaker	beginning	end
John	3	6
Mary	3	4
Gregory	2	5
Suzan	2	4
Paul	3	4
Helen	1	6

- a single room
- each talk last one hour

→ sequencing of presentations ?

alldiff/2 : example (2)

Modelling

- a variable = the “hour” of a speach
- no overlapping of speaches = not 2 talks at the same time
- $J \in [3,6], M \in [3,4], G \in [2,5],$
  $S \in [2,4], P \in [3,4], H \in [1,6],$
  alldiff$([J, M, G, S, P, H])$

alldiff/2

formulation by conjunction of disequations :

- costly ($\frac{n(n-1)}{2}$ constraints)
- inefficient

$$x_1 \in \{1,2\}, x_2 \in \{1,2\}, x_3 \in \{1,2\}, \text{alldiff}([x_1, x_2, x_3])$$

enforcing arc consistency ➔ generally, no reduction

in the previous example :

- arc consistency (binary) : no reduction
- bound consistency (n-ary) : not consistent

  $M = 4$ and $P = 3$ (or vice-versa), 3 must be deleted from J

  ➔ reduction

`alldiff/2` : Hall

let K be a set of variables, and $|K|$ the cardinamity of K. Consider :

$$\text{dom}(K) = \bigcup_{x_i \in K} D_i$$

Theorem[from Hall, 1935] the constraint $\texttt{alldiff}(x_1, \ldots, x_n)$ over the variables $x_1, \ldots, x_n$ with domains $D_1, \ldots, D_n$ has a solution iff there does not exists a sub-set $K \subseteq \{x_1, \ldots, x_n\}$ s.t. :

$$|K| > |\text{dom}(K)|$$

Idea : if there exists a set K s.t. $|K| = |\text{dom}(K)|$, we know that the variables of K will use all the values from $\text{dom}(K)$

➔ these values can be removed from variables not in K

Examples (previous example) : $K = \{M, S, P\}$ and $K = \{M, P\}$

`alldiff/2` : Hall interval

Hall interval Given the variables $x_1, \ldots, x_n$ with domains $D_1, \ldots, D_n$ and an interval I, let vars$(I) = \{x_i \mid D_i \subseteq I\}$. The interval I is a *Hall interval* iff $|I| = |$vars$(I)|$.

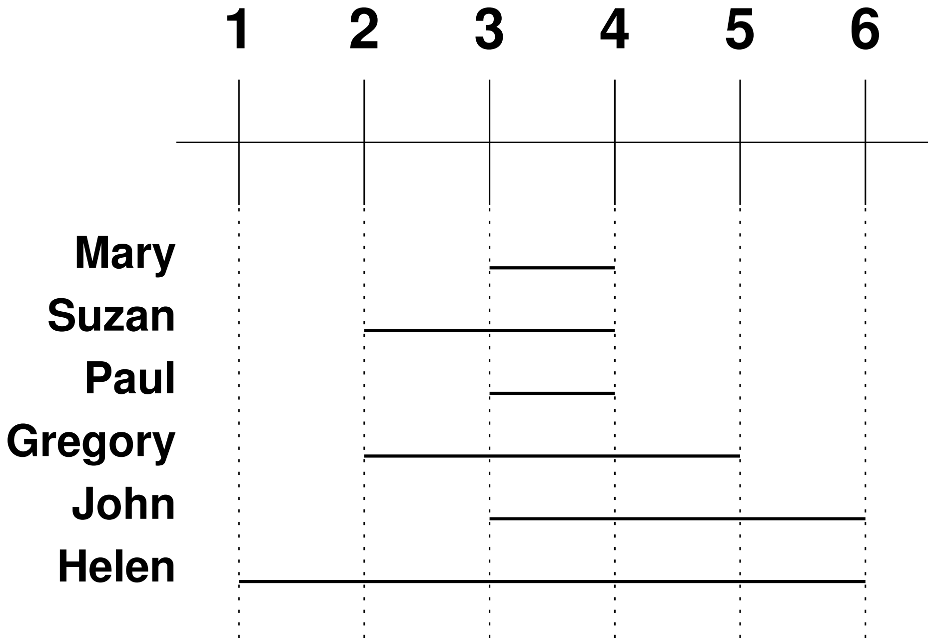
Proposition the constraint $\texttt{alldiff}(x_1, \ldots, x_n)$ is *bound consistent* w.r.t. $D_1, \ldots, D_n$ iff

- for each interval I, $|$vars$(I)| \leqslant |I|$,
- and if for each Hall interval J and each variable x_i, we have : either $D_i \subseteq J$, or $\{\min D_i, \max D_i\} \cap J = \varnothing$

alldiff/2 : mechanism

Process in 2 phases : update of left bounds and update of right bounds

- ordering of variables : increasing ordering on right bounds
- determining Hall intervals
- modification of right bounds



`alldiff/2` : algorithm (based on Hall)

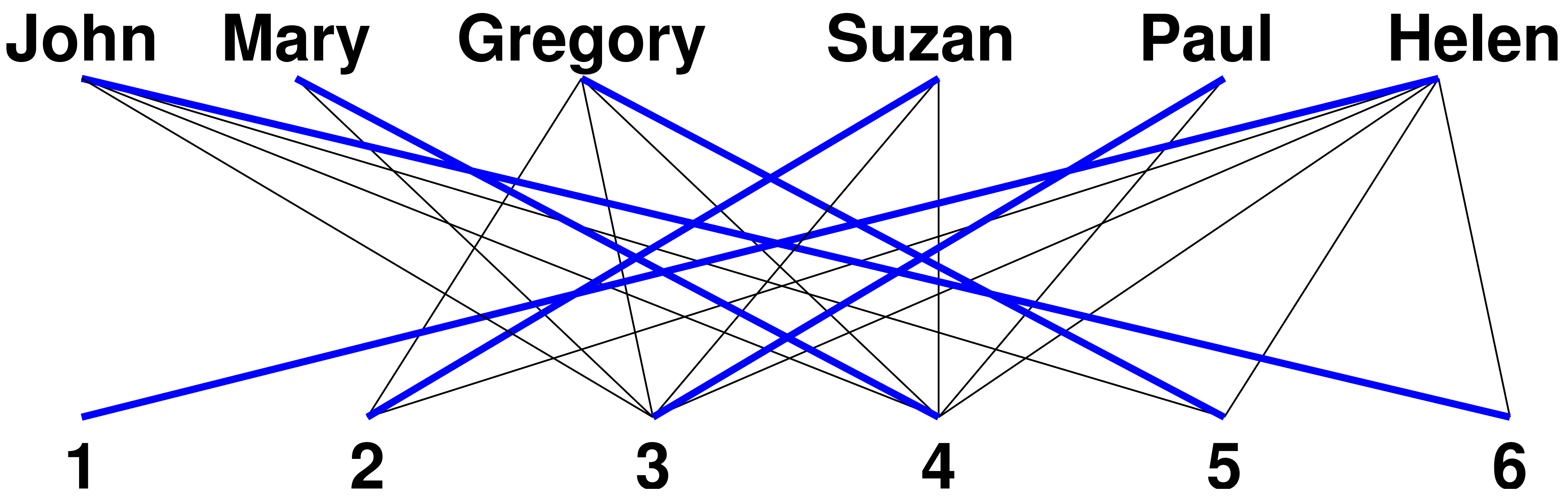
```
update_min(x=x_1...x_n)
begin
  sort(x)
  for i=1 to n do
    min[i]=min(x[i])
    max[i]=max(x[i])
  done
  for i=1 to n do
    Insert(i)
  done
end

IncrMin(a,b,i)
% [a,b] Intervalle de Hall
begin
  for j=i+1 to n do
      if min[j] >= a then
          x[j] #>= b+1
      fi
  done
end
```

```
Insert(i)
begin
  u[i]=min[i]
  for j=1 to i-1 do
    if min[j]<min[i] then
        u[j]++
        if u[j]>max[i] then Fail
        if u[j]=max[i] then
          IncrMin(min[j],max[j],i)
        fi
    else
        u[i]++
    fi
  done
  if u[i]>max[i] then Fail
  if u[i]=max[i] then
      IncrMin(min[i],max[i],i)
  fi
end
```

primitive algorithm in $\mathcal{O}(n^3)$. a refined version in $\mathcal{O}(n \log n)$

alldiff/2 : graph

possibility to enforce a stronger consistency (hyper-arc consistency) by searching a maximum coupling in the graph of the values of the problem



complexity : $\mathcal{O}(m\sqrt{n})$, m the number of arcs in the graph

`alldiff/2` : idea of algorithm (graph)

- graph : bipartite (values, variables)
- coupling : not two arcs on the same node
- maximum : the coupling cannot be extended
- if a variable is not connected : insatisfiable constraint
- if a value is not connected : several solutions

Other global constraints

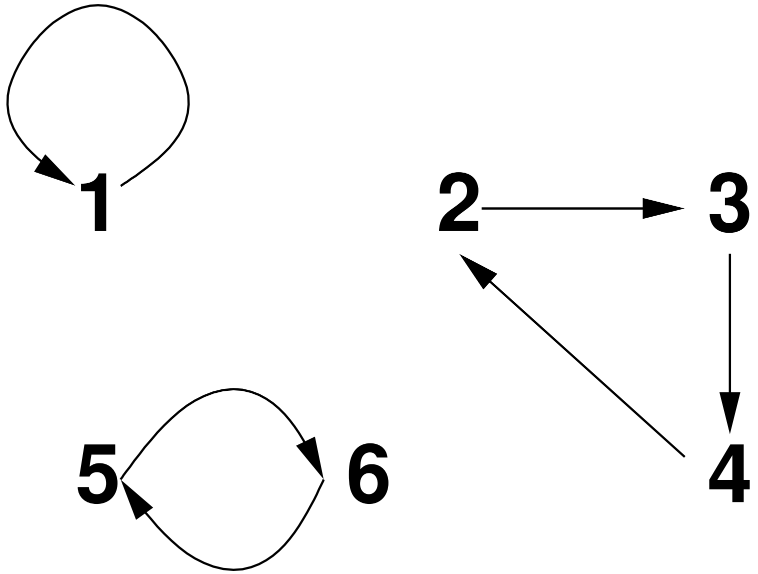
- `element`$(k, [c_1, \ldots, c_n], x)$.
  the variable x must be equal to c_k
- `atmost(N,List,V)`
  at most `N` variables of `List` must be equal to the value `V`
- `gcc`$([x_1, \ldots, x_n], [v_1, \ldots, v_k], [q_1, \ldots, q_k])$
  the number of variables from $[x_1, \ldots, x_n]$ that have the value v_i must be equal to q_i
  (generalization of `alldiff`)

cycle/2

- $\texttt{cycle}(n, [s_1, \ldots, s_m])$.
  the list $[s_1, \ldots, s_m]$ must be a permutation of $\{1, \ldots, m\}$ constituting n distinct cycles :
  - $\forall i \in [1, m]$: $1 \leqslant s_i \leqslant m$
  - $s_i \neq s_j \quad \forall i \neq j$
  - let C_i be a set of integers defined by :
    - $i \in C_i$
    - if $j \in C_i$ then $s_j \in C_i$
      (so, n distinct sets are defined)

cycle/2

- Example : `cycle(3,[1,3,4,2,6,5]).`
  4 in 3rd position, thus an arc from 3 to 4, . . .



Example (1)

Travelling salesman problem :

- n sites must be visited exactly once
- there are k travelling salesmen
- distances c_{ij} between sites i and j are known

$\rightarrow$ find the round of each salesman which minimizes the total covered distance

Example (1)

Modelling : x_i is the site to visit after the site i, y_i the cost (distance) from i.

$$
\begin{array}{ll}
 & \min \sum_{i=1}^{n} y_i \\
s.t. & x_i \in \{1, \ldots, n\}, \text{ for } i \in \{1, \ldots, n\} \\
 & y_i \in \{c_{i1}, \ldots, c_{in}\}, \text{ for } i \in \{1, \ldots, n\} \\
 & \text{element}(x_i, [c_{i1}, \ldots, c_{in}], y_i), \text{ for } i \in \{1, \ldots, n\} \\
 & \text{cycle}(k, [x_1, \ldots, x_n])
\end{array}
$$

Exemple (2)

- $\{c_{i1}, \ldots, c_{in}\}$: cost from city i to the n other cities
- k : number of cycles needed (number of travelling salesmen)
- $x_1, \ldots, x_n$: set of cities
- element$(x_i, [c_{i1}, \ldots, c_{in}], y_i)$: the cost from city i to city x_i (i.e., y_i) is an element of the list of costs from city i to another city
- cycle$(k, [x_1, \ldots, x_n])$: all cities must be visited in k distinct cycles
- $\min \sum_{i=1}^{n} y_i$: money, money ! ! ! the total cost to visit all cities must be minimized

cumulative/4

$\texttt{cumulative}([O_1, \ldots, O_m], [D_1, \ldots, D_m], [R_1, \ldots, R_m], L)$
the constraint is verified iff

$$\forall i \in \mathbb{N}: \sum_{j | O_j \leqslant i \leqslant O_j + D_j - 1} R_j \leqslant L$$

interprétation : allocation of a single resource

- $[O_1, \ldots, O_m]$: starting date of the m tasks
- $[D_1, \ldots, D_m]$: duration of the m tasks
- $[R_1, \ldots, R_m]$: number of resource units required for each task
- L : total number of resource units available at each moment

Example

there are 13 resource units available at each moment

we have the following tasks :

task	t_1	t_2	t_3	t_4	t_5	t_6	t_7
duration	16	6	13	7	5	18	4
resource units	2	9	3	7	10	1	11

Question : for all the tasks, find starting and ending dates that minimize the total time of resource utilization

Program (GNU Prolog)

```
schedule(LO,End):-
   LO = [O1,O2,O3,O4,
         O5,O6,O7],
   LD = [16,6,13,7,
         5,18,4],
   LR = [2,9,3,7,10,
         1,11],
   LE = [E1,E2,E3,E4,
         E5,E6,E7],
   End in 1..30,
   domain(LO,1,30),
   domain(LE,1,30),
   O1 + 16 #= E1,
   O2 + 6 #= E2,
   O3 + 13 #= E3,
```

```
   O4 + 7 #= E4,
   O5 + 5 #= E5,
   O6 + 18 #= E6,
   O7 + 4 #= E7,
   maximum(End,LE),
   cumulative(LO,LD,LR,13),
   minimize(labeling(LO),End).
```

Program (ECL^{i}PS^e) (1)

```
:-lib(fd),lib(fd_global),lib(cumulative).

schedule(LO,End):-
  % starting time
  LO = [O1,O2,O3,O4,O5,O6,O7],

  %duration of tasks
  LD = [16,6,13,7,5,18,4],

  % resources needed by each task
  LR = [2,9,3,7,10,1,11],

  % ending times
  LE = [E1,E2,E3,E4,E5,E6,E7],

  % time allowed
  End:: [1..30],
  LO:: [1..30],
  LE:: [1..30],
```

Program (ECL^{i}PS^e) (2)

```
% ending time is starting time + duration
O1 + 16 #= E1,
O2 + 6 #= E2,
O3 + 13 #= E3,
O4 + 7 #= E4,
O5 + 5 #= E5,
O6 + 18 #= E6,
O7 + 4 #= E7,

% constraint End to be the maximum element in the list L
 maxlist(LE,End),

% start, duration, resource units, resource limits
cumulative(LO,LD,LR,13),

% find the values that minize LO
minimize(labeling(LO),End).
```

Solution

```
[eclipse 22]: schedule(LO,E).
Found a solution with cost 28
Found a solution with cost 27
Found a solution with cost 23

LO = [1, 17, 10, 10, 5, 5, 1]
E = 23
Yes (0.07s cpu)
```