Global Constraints

Jean-Charles Regin
ILOG, Sophia Antipolis
regin@ilog.fr
Plan

- General Principles of Constraint Programming
- Global Constraints: advantages
- How to write a filtering algorithm?
- Examples: sports scheduling and car sequencing
- Over-constrained problems
- Discussion: quality of a FA, incrementality, closure, incomplete algorithms, power of a FA
- Conclusion
Plan

- General Principles of Constraint Programming
  - Global Constraints: advantages
  - How to write a filtering algorithm?
  - Examples: sports scheduling and car sequencing
  - Over-constrained problems
  - Discussion: quality of a FA, incrementality, closure, incomplete algorithms, power of a FA
- Conclusion
Constraint Programming

- 3 notions:
  - constraint network: variables, domains constraints
  + filtering (domain reduction)
  - propagation
  - search procedure (assignments + backtrack)
Problem = conjunction of sub-problems

- In CP a problem can be viewed as a conjunction of sub-problems that we are able to solve
- A sub-problem can be trivial: $x < y$ or complex: search for a feasible flow
- A sub-problem = a constraint
Filtering

- We are able to solve a sub-problem: a method is available
- CP uses this method to remove values from domain that do not belong to a solution of this sub-problem: filtering
- E.g: \( x < y \) and \( D(x) = [10,20] \), \( D(y) = [5,15] \)
  \[ \Rightarrow D(x) = [10,14], \quad D(y) = [11,15] \]
Filtering

- A filtering algorithm is associated with each constraint (sub-problem).
- Can be simple \((x < y)\) or complex \((\text{alldiff})\).
Alldiff and GCC Constraints

- **Alldiff(X):** the variables of X must be pairwise different (i.e. \( \forall x, y \in X: x \neq y \))
- **GCC(X,\{li\},\{ui\}):** the number of times each value \( v_i \) can be taken must be in a given interval \([li, ui]\)
- **Example:** \( D(x1) = \{a,b,c,d\} \), \( D(x2) = \{a,b,c,d\} \), \( D(x3) = \{b,c\} \), \( D(x4) = \{c,d\} \). Values b and c must be taken at most 2, values a and d must be taken at least 1.
Arc consistency

- All the values which do not belong to any solution of the constraint are deleted.

- Example: Alldiff({x,y,z}) with D(x)=D(y)={0,1}, D(z)={0,1,2}
  the two variables x and y take the values 0 and 1, thus z cannot take these values.
  FA by AC => 0 and 1 are removed from D(z)
Propagation

- Domain Reduction due to one constraint can lead to new domain reduction of other variables
- When a domain is modified all the constraints involving this variable are studied and so on ...
Why Propagation?

- A problem = conjunction of easy sub-problems.
- Sub-problems: local point of view. Propagation tries to obtain a global point of view from independent local point of view.
- The conjunction is stronger than the union of independent resolutions.
Why Propagation?

- A problem = conjunction of easy sub-problems.
- Sub-problems: local point of view. Propagation tries to obtain a global point of view from independent local point of view.
- The conjunction is stronger than the union of independent resolution.
- To help the propagation to have a global point of view: use global constraints!
Global Constraint

- A global constraint is equal to a conjunction of constraints
- Example: alldiff and Gcc constraints
- \( G = \land \{ C_1, C_2, \ldots, C_k \} \)
  The set of tuples of \( G \) is equal to the set of solutions of the problem: \( (U_i \times (C_i), DX(G), \{ C_1, C_2, \ldots, C_k \}) \)
Plan

- General Principles of Constraint Programming
- **Global Constraints: advantages**
  - How to write a filtering algorithm?
  - Examples: sports scheduling and car sequencing
  - Over-constrained problems
  - Discussion: quality of a FA, incrementality, closure, incomplete algorithms, power of a FA
- Conclusion
Global Constraints: advantages

- **Expressiveness:** It is more convenient to define one constraint corresponding to a set of constraints than to define independently each constraint.

- **Better understanding of the problem structure:** Some part of the structure is immediately identified.

- **Powerful filtering algorithms:** The set of constraint can be taken into account as a whole.
Global constraint: expressiveness

- Example Rostering Problem
### Rostering (G. Pesant)

<table>
<thead>
<tr>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
<th>Sat</th>
<th>Sun</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>D</td>
<td>E</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M. Green</td>
<td>M. Red</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mrs. Blue</td>
<td>M. Yellow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Rostering

<table>
<thead>
<tr>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
<th>Sat</th>
<th>Sun</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>M. Green</td>
<td>M. Red</td>
<td>Mrs. Blue</td>
<td>M. Yellow</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Each works at most one shift per day
### Rostering

<table>
<thead>
<tr>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
<th>Sat</th>
<th>Sun</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- M. Green
- Mrs. Blue
- M. Red
- M. Yellow

$x_{ij} \in \{g, b, r, y\}$

$x_{iD} \neq x_{iE}$, $x_{iD} \neq x_{iN}$, $x_{iE} \neq x_{iN}$, Mon $\leq i \leq$ Sun
enum Days = \{\text{mon, tue, wed, thu, fri, sat, sun}\}
enum Shifts = \{D, E, N\}
enum Workers = \{\text{green, white, red, yellow}\}

var Workers onDuty[Days, Shifts]
forall( i in Days )
forall( j, k in Shifts: j < k )
onDuty[i, j] \neq onDuty[i, k]
enum Days = {mon,tue,wed,thu,fri,sat,sun}
enum Shifts = {D,E,N}
enum Workers = {green,white,red,yellow}

var Workers onDuty[Days,Shifts]
forall( i in Days )
  forall( j,k in Shifts: j < k )
    onDuty[i,j] ≠ onDuty[i,k]
Mutual exclusion

- A set of variables must take on distinct values.
- \( \text{forall( i in Days )} \)
  \( \text{forall( j,k in Shifts: j < k )} \)
  \( \text{onDuty}[i,j] \neq \text{onDuty}[i,k] \)
Mutual exclusion

- A set of variables must take on distinct values.
- \[
    \forall i \in \text{Days} \quad \forall j,k \in \text{Shifts: } j < k 
    \quad \text{onDuty}[i,j] \neq \text{onDuty}[i,k]
\]
- Can be replaced by
  \[
    \forall i \in \text{Days} 
    \quad \text{alldifferent(onDuty}[i])
  \]
Global constraint: underlined structure

- A global cardinality constraint is equivalent to a set of atmost/atleast constraint.
- From the simultaneous presence of these constraints new deductions can be made (detailed in this talk)
Global constraint: powerful filtering algorithms

- Color the graph with cliques:
  - $c_0 = \{0, 1, 2, 3, 4\}$
  - $c_1 = \{0, 5, 6, 7, 8\}$
  - $c_2 = \{1, 5, 9, 10, 11\}$
  - $c_3 = \{2, 6, 9, 12, 13\}$
  - $c_4 = \{3, 7, 10, 12, 14\}$
  - $c_5 = \{4, 8, 11, 13, 14\}$

- Clique size: 27  Global: #fails: 0  cpu time: 1.212 s
  - Local:  #fails: 1  cpu time: 0.171 s
- Clique size: 31  Global: #fails: 4  cpu time: 2.263 s
  - Local:  #fails: 65  cpu time: 0.37 s
- Clique size: 51  Global: #fails: 501  cpu time: 25.947 s
  - Local:  #fails: 24512  cpu time: 66.485 s
- Clique size: 61  Global: #fails: 5  cpu time: 58.223 s
  - Local:  ??????????????
Some Global Constraints

- Cumulative, diff-n, cycle, sort, alldiff and permutation, symmetric alldiff, global cardinality, global cardinality with costs, sum and scalar product of alldiff variables, sequence, stretch, minimum global distance, k-diff, number of distinct values, lexicographic ordering, regular.
Cumulative constraint

- Scheduling problems:
- Activities having a start time (S var), a duration (D var), a consumption (H var)
- At each time the summation of the consumption must be less than a var u.
- Non preemptive scheduling (interruption is forbidden)
Sort

- Two types of variables: X var and Y var
- The Y var represents the X var when there are sorted.
- Example: $x_1=[0,5]$, $x_2=[0,5]$, $x_3=[0,5]$  
- We want to have a strict ordering: 
  $y_1=[0,3]$, $y_2=[1,4]$, $y_3=[2,5]$
Symmetric alldiff

- Goal: group by pair
- Alldiff + If i is assigned to j then j is assigned to i
- Vars= entities and Values =entities
- 3 entities: a, b, c: 3 vars xa, xb, xc and 3 values a, b, c
- If xa=c then xc=a
- No solution here (the alldiff does not find this result).
Nvalue

- **Nvalue(X,k):** the number of distinct values assigned to X must be equal to k.
- \((x_1=a,x_2=b,x_3=c,x_4=b,x_5=a,x_6=a): k=3 \ (a,b,c)\)
- NP-Complete constraint equivalent to set-cover problem.
Stretch

- Run length coding
- Consecutives values
- Defines by size of group of values (with min and max)
- Idea: var are ordered and if $x_i$ is blue then $x_i$ belongs to a group of $k$ consecutive var taken blue as value.
Regular

- Defined by automata
- Kind of table constraints whose list of tuples is defined by an automata (instead of being explicitly given).
Plan

- General Principles of Constraint Programming
- Global Constraints: advantages
- **How to write a filtering algorithm?**
- Examples: sports scheduling and car sequencing
- Over-constrained problems
- Discussion: quality of a FA, incrementality, closure, incomplete algorithms, power of a FA
- Conclusion
Filtering Algorithm

- based on constraints addition (from the simultaneous presence of constraints, new constraints are added)
- general (GAC-Schema)
- ad-hoc (integration of OR algorithm)
Constraints addition

- 5 variables: X={x1,x2,x3,x4,x5}
- domains: [0..4]
- constraints: atleast(X, 1,1) , atleast(X,1,2)
  atleast(X,1,3), atleast(X, 1,4)
- What will happen?
Constraints addition

- 5 variables: X={x1,x2,x3,x4,x5}
- domains: [0..4]
- constraints: atleast(X, 1,1) , atleast(X,1,2) 
  atleast(X,1,3), atleast(X, 1,4)
- atleast(X,1,val): local view: while val belongs to 1+1=2 domains of variable, nothing can be deduced
Constraints addition

- 5 variables: $X = \{x_1, x_2, x_3, x_4, x_5\}$
- domains: [0..4]
- constraints: \(\text{atleast}(X, 1, 1)\), \(\text{atleast}(X, 1, 2)\), \(\text{atleast}(X, 1, 3)\), \(\text{atleast}(X, 1, 4)\)

- \(\text{atleast}(X, 1, \text{val})\): local view: while \text{val} belongs to $1+1=2$ domains of variable, nothing can be deduced

- $x_1=0$, $x_2=0$, $x_3=0$ is ok, any combination of 3 variables containing 2 times \text{val} 0 is ok. This is stupid
Constraints addition

- 5 variables: \( X = \{x_1, x_2, x_3, x_4, x_5\} \)
- domains: \([0..4]\)
- constraints: \( \text{atleast}(X, 1, 1) , \text{atleast}(X, 1, 2) \)
  \( \text{atleast}(X, 1, 3), \text{atleast}(X, 1, 4) \)

- From the simultaneous presence of constraints we can deduce other constraints:
  if (4 values must be taken at least 1) then the other values can be taken at most \( n-4=5-4=1 \)
  New constraint: \( \text{atmost}(X, 1, 0) \)
Constraints addition

- Done for the global cardinality constraint (the unconstrained values becomes constrained)
- Therefore, done for the alldiff constraint and permutation (each value has to be taken exactly once)
Constraints addition

- Another example [Roy and Pachet CP’99]:
  Union of set variables:
  \( E = A \cup B \)
  Good implementation:
Another example [Roy and Pachet CP’99]:
Union of set variables:
\[ E = A \cup B \]
Good implementation: \textbf{think to the intersection}
\[ I = A \cap B, \text{ and} \]
\[ \text{card}(E) = \text{card}(A) + \text{card}(B) - \text{card}(I) \]
Constraints addition

- There is no new filtering algorithm
- Only implicit constraints are added
- The previous problem is solved!
- Easy and really interesting: a kind of presolve.
Filtering Algorithm

- **based on constraints addition** *(from the simultaneous presence of constraints, new constraints are added)*
- **general (GAC-Schema)**
- **ad-hoc** *(integration of OR algorithm)*
GAC-Schema

- A generic framework for achieving AC for any kind of constraint (can be non binary).
  Bessiere and Regin, IJCAI’97, CP’99
- You just have to say how to compute a solution.
- Manages the incrementality for you (notion of support).
GAC-Schema: instantiation

- List of allowed tuples
- List of forbidden tuples
- Predicates
- Any OR algorithm
- Solver reentrance
GAC-Schema

- **Idea:**
  - `tuple` = solution of the constraint
  - `support` = valid tuple
  - while the tuple is valid: do nothing
  - if the tuple is no longer valid, then search for a new support for the values it contains

- a solution (support) can be computed by any OR algorithm
Example

- $X(C) = \{x_1, x_2, x_3\}$  $D(x_i) = \{a, b\}$
- $T(C) = \{(a, a, a), (a, b, b), (b, b, a), (b, b, b)\}$
Example

- $X(C) = \{x_1, x_2, x_3\}$  $D(x_i) = \{a, b\}$
- $T(C) = \{(a, a, a), (a, b, b), (b, b, a), (b, b, b)\}$
- Support for $(x_1,a)$: $(a,a,a)$ is computed and $(a,a,a)$ is added to $S(x_2,a)$ and $S(x_3,a)$, $(x_1,a)$ in $(a,a,a)$ is marked as supported.
Example

- $X(C) = \{x_1, x_2, x_3\}$  $D(x_i) = \{a, b\}$
- $T(C) = \{(a, a, a), (a, b, b), (b, b, a), (b, b, b)\}$
- Support for $(x_1, a)$: $(a, a, a)$ is computed and $(a, a, a)$ is added to $S(x_2, a)$ and $S(x_3, a)$, $(x_1, a)$ in $(a, a, a)$ is marked as supported.
- Support for $(x_2, a)$: $(a, a, a)$ is in $S(x_2, a)$ it is valid, therefore it is a support. (Multidirectionnality). **No need to compute a solution**
Example

- $X(C) = \{x_1, x_2, x_3\}$  $D(x_i) = \{a, b\}$
- $T(C) = \{(a,a,a),(a,b,b),(b,b,a),(b,b,b)\}$
- Support for $(x_1,a)$: $(a,a,a)$ is computed and $(a,a,a)$ is added to $S(x_2,a)$ and $S(x_3,a)$, $(x_1,a)$ in $(a,a,a)$ is marked as supported.
- Value $a$ is removed from $x_2$, then all the tuple in $S(x_2,a)$ are no longer valid: $(a,a,a)$ for instance. The validity of the values supported by this tuple must be reconsidered.
Example

- $X(C) = \{x_1, x_2, x_3\}$, $D(x_i) = \{a, b\}$
- $T(C) = \{(a,a,a), (a,b,b), (b,b,a), (b,b,b)\}$
- Support for $(x_1,a)$: $(a,a,a)$ is computed and $(a,a,a)$ is added to $S(x_2,a)$ and $S(x_3,a)$, $(x_1,a)$ in $(a,a,a)$ is marked as supported.
- Support for $(x_1,b)$: $(b,b,a)$ is computed, and update ...
GAC-Schema: complexity

- CC complexity to check consistency (seek in table, call to OR algorithm): seek for a Support costs CC
- n variables, d values:
  - for each value: CC
  - for all values: $O(ndCC)$
- For any OR algorithm which is able to compute a solution, Arc consistency can be achieved in $O(ndCC)$. 
GAC-Schema: complexity

- After 1 modification:
  - consistency in $O(CC)$
  - arc consistency in $O(ndCC)$

- After $k$ modifications
  - consistency in $O(CC)$
  - arc consistency in $O(ndCC)$
Configuration problem:
5 types of components: \{glass, plastic, steel, wood, copper\}
3 types of bins: \{red, blue, green\} whose capacity is red 5, blue 5, green 6

Constraints:
- red can contain glass, cooper, wood
- blue can contain glass, steel, cooper
- green can contain plastic, copper, wood
- wood require plastic; glass exclusive copper
- red contains at most 1 of wood
- green contains at most 2 of wood

For all the bins there is either no plastic or at least 2 plastic

Given an initial supply of 12 of glass, 10 of plastic, 8 of steel, 12 of wood and 8 of copper; what is the minimum total number of bins?
Table Constraint: results

<table>
<thead>
<tr>
<th></th>
<th>#bk</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>standard model</td>
<td>1,361,709</td>
<td>430</td>
</tr>
<tr>
<td>Table Constraint</td>
<td>12,659</td>
<td>9.7</td>
</tr>
</tbody>
</table>
Filtering Algorithm

- based on constraints addition (from the simultaneous presence of constraints, new constraints are added)
- general (GAC-Schema)
- ad-hoc (integration of OR algorithm but not only)
Structure of a constraint

- Speed-up the search for a support (solution which contain a value \((x,a)\))
Structure of a constraint

- Speed-up the search for a support (solution which contain a value $(x,a)$):
  $x < y$, $D(x)=[0..10000]$, $D(y)=[0..10000]$
  support for $(x,9000)$: immediate any value greater than 9000 in $D(y)$
Structure of a constraint

- Design of ad-hoc filtering algorithm:
  - $x < y$:
    - (a) $\max(x) = \max(y) - 1$
    - (b) $\min(y) = \min(x) + 1$
Structure of a constraint

- Design of ad-hoc filtering algorithm:
  \[ x < y : \]
  (a) \[ \text{max}(x) = \text{max}(y) - 1 \]
  (b) \[ \text{min}(y) = \text{min}(x) + 1 \]

- Triggering of the filtering algorithm:
  no possible pruning of \( D(x) \) while \( \text{max}(y) \) is not modified
  no possible pruning of \( D(y) \) while \( \text{min}(x) \) is not modified
Structure of a constraint

- Speed-up the search for a support (solution which contain a value (x,a))
- Design of specific algorithm
- Incrementality
The value graph:

D(x1) = \{1,2\}
D(x2) = \{2,3\}
D(x3) = \{1,3\}
D(x4) = \{3,4\}
D(x5) = \{2,4,5,6\}
D(x6) = \{5,6,7\}
Alldiff constraint

Default orientation
Alldiff constraint

Default orientation
Alldiff constraint

Default orientation

x1 -- x2 -- x3 -- x4 -- x5 -- x6

s

1 -- 2 -- 3 -- 4 -- 5 -- 6 -- 7

t
Default orientation

Value network

(1,1)

(0,1)

(0,1)

(6,6)
A feasible flow

Default orientation

(1,1)

(t)

(6,6)

x1

x2

x3

x4

x5

x6

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)

(0,1)
Residual graph
Residual graph

orientation
Arc consistency

The value graph:

\[ \begin{align*}
D(x_1) &= \{1, 2\} \\
D(x_2) &= \{2, 3\} \\
D(x_3) &= \{1, 3\} \\
D(x_4) &= \{4\} \\
D(x_5) &= \{5, 6\} \\
D(x_6) &= \{5, 6, 7\}
\end{align*} \]
Alldiff results

- Color the graph with cliques:
  - $c_0 = \{0, 1, 2, 3, 4\}$
  - $c_1 = \{0, 5, 6, 7, 8\}$
  - $c_2 = \{1, 5, 9, 10, 11\}$
  - $c_3 = \{2, 6, 9, 12, 13\}$
  - $c_4 = \{3, 7, 10, 12, 14\}$
  - $c_5 = \{4, 8, 11, 13, 14\}$

- clique size: 27  Global: #fails: 0  cpu time: 1.212 s
  - Local: #fails: 1  cpu time: 0.171 s
- clique size: 31  Global: #fails: 4  cpu time: 2.263 s
  - Local: #fails: 65  cpu time: 0.37 s
- clique size: 51  Global: #fails: 501  cpu time: 25.947 s
  - Local: #fails: 24512  cpu time: 66.485 s
- clique size: 61  Global: #fails: 5  cpu time: 58.223 s
  - Local: ????????????????
Alldiff constraint

- Compute a feasible flow
- Compute the strongly connected components
- Remove every arc of flow value 0 for which the ends belong to two different components
- Linear algorithm achieving arc consistency
- Idem for global cardinality constraints
- work well due to (0,1) arcs
Alldiff constraint: complexity

- After 1 modification:
  - consistency computed in $O(nd)$
  - arc consistency computed in $O(nd)$

- After $k$ modifications:
  - consistency in $O(nd \sqrt{k})$
  - arc consistency in $O(nd \sqrt{k} + nd)$
Alldiff constraint

- Relations between GAC, AC, etc…:

Arc Consistency for the global constraints corresponds to the arc consistency of a CN with an exponential number of constraints:
- for k=1..n: for every group G of k variables: we must have:
  \[ |D(G)| \geq k \]
and if \( |D(G)| = k \) then \( D(X) \leftarrow D(X) - D(G) \)
Ad-hoc algorithm: N-queens problems

- variables: a var = possible columns for a row
  Rule: [Regin]
  If the domain of a var contains more than 3 values, this var cannot cause any deletion

3 directions for every value of y
if x contains 4 values: no problem
Ad-hoc algorithm: N-queens problems

- variables: a var = possible columns for a row

Rule: [Regin]
If the domain of a var contains exactly 3 values, this var can cause only specific deletions

The red value of y is deleted only if x contains yellows values
Ad-hoc algorithm: N-queens problems

- variables: a var = possible columns for a row
  - Rule: [Regin]
    - If the domain of a var contains exactly 2 values, this var can cause only specific deletions

The red values of y are deleted only if x contains yellows values
Plan

- General Principles of Constraint Programming
- Global Constraints: advantages
- How to write a filtering algorithm?
- Examples: sports scheduling and car sequencing
- Over-constrained problems
- Discussion: quality of a FA, incrementality, closure, incomplete algorithms, power of a FA
- Conclusion
Examples

- Sports scheduling
- Car sequencing
The problem

- n teams and n-1 weeks and n/2 periods
- every two teams play each other exactly once
- every team plays one game in each week
- no team plays more than twice in the same period

<table>
<thead>
<tr>
<th></th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period 1</td>
<td>0 vs 1</td>
<td>0 vs 2</td>
<td>4 vs 7</td>
<td>3 vs 6</td>
<td>3 vs 7</td>
<td>1 vs 5</td>
<td>2 vs 4</td>
</tr>
<tr>
<td>Period 2</td>
<td>2 vs 3</td>
<td>1 vs 7</td>
<td>0 vs 3</td>
<td>5 vs 7</td>
<td>1 vs 4</td>
<td>0 vs 6</td>
<td>5 vs 6</td>
</tr>
<tr>
<td>Period 3</td>
<td>4 vs 5</td>
<td>3 vs 5</td>
<td>1 vs 6</td>
<td>0 vs 4</td>
<td>2 vs 6</td>
<td>2 vs 7</td>
<td>0 vs 7</td>
</tr>
<tr>
<td>Period 4</td>
<td>6 vs 7</td>
<td>4 vs 6</td>
<td>2 vs 5</td>
<td>1 vs 2</td>
<td>0 vs 5</td>
<td>3 vs 4</td>
<td>1 vs 3</td>
</tr>
</tbody>
</table>
The problem

- n teams and n-1 weeks and n/2 periods
- every two teams play each other exactly once
- every team plays one game in each week
- no team plays more than twice in the same period

<table>
<thead>
<tr>
<th></th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period 1</td>
<td>0 vs 1</td>
<td>0 vs 2</td>
<td>4 vs 7</td>
<td>3 vs 6</td>
<td>3 vs 7</td>
<td>1 vs 5</td>
<td>2 vs 4</td>
</tr>
<tr>
<td>Period 2</td>
<td>2 vs 3</td>
<td>1 vs 7</td>
<td>0 vs 3</td>
<td>5 vs 7</td>
<td>1 vs 4</td>
<td>0 vs 6</td>
<td>5 vs 6</td>
</tr>
<tr>
<td>Period 3</td>
<td>4 vs 5</td>
<td>3 vs 5</td>
<td>1 vs 6</td>
<td>0 vs 4</td>
<td>2 vs 6</td>
<td>2 vs 7</td>
<td>0 vs 7</td>
</tr>
<tr>
<td>Period 4</td>
<td>6 vs 7</td>
<td>4 vs 6</td>
<td>2 vs 5</td>
<td>1 vs 2</td>
<td>0 vs 5</td>
<td>3 vs 4</td>
<td>1 vs 3</td>
</tr>
</tbody>
</table>

- Problem 10 teams of the MIPLIB
  (n=10 and the objective function is dummy)
- MIP is not able to find a solution for n=14
- CP finds a solution for n=10 in 0.06s, n=14 in 0.2, n=40 in 6h
The problem

- $n$ teams and $n-1$ weeks and $n/2$ periods
- every two teams play each other exactly once
- every team plays one game in each week
- no team plays more than twice in the same period

<table>
<thead>
<tr>
<th></th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period 1</td>
<td>0 vs 1</td>
<td>0 vs 2</td>
<td>4 vs 7</td>
<td>3 vs 6</td>
<td>3 vs 7</td>
<td>1 vs 5</td>
<td>2 vs 4</td>
</tr>
<tr>
<td>Period 2</td>
<td>2 vs 3</td>
<td>1 vs 7</td>
<td>0 vs 3</td>
<td>5 vs 7</td>
<td>1 vs 4</td>
<td>0 vs 6</td>
<td>5 vs 6</td>
</tr>
<tr>
<td>Period 3</td>
<td>4 vs 5</td>
<td>3 vs 5</td>
<td>1 vs 6</td>
<td>0 vs 4</td>
<td>2 vs 6</td>
<td>2 vs 7</td>
<td>0 vs 7</td>
</tr>
<tr>
<td>Period 4</td>
<td>6 vs 7</td>
<td>4 vs 6</td>
<td>2 vs 5</td>
<td>1 vs 2</td>
<td>0 vs 5</td>
<td>3 vs 4</td>
<td>1 vs 3</td>
</tr>
</tbody>
</table>

For 40 teams: 800 variables with 39 possible values for each variable.
CP model: variables

For each slot: 2 variables represent the teams and 1 variable represents the match are defined

<table>
<thead>
<tr>
<th></th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period 1</td>
<td>0 vs 1</td>
<td>0 vs 2</td>
<td>4 vs 7</td>
<td>3 vs 6</td>
<td>3 vs 7</td>
<td>1 vs 5</td>
<td>2 vs 4</td>
</tr>
<tr>
<td>Period 2</td>
<td>2 vs 3</td>
<td>1 vs 7</td>
<td>0 vs 3</td>
<td>5 vs 7</td>
<td>1 vs 4</td>
<td>0 vs 6</td>
<td>5 vs 6</td>
</tr>
<tr>
<td>Period 3</td>
<td>4 vs 5</td>
<td>3 vs 5</td>
<td>1 vs 6</td>
<td>0 vs 4</td>
<td>2 vs 6</td>
<td>2 vs 7</td>
<td>0 vs 7</td>
</tr>
<tr>
<td>Period 4</td>
<td>6 vs 7</td>
<td>4 vs 6</td>
<td>2 vs 5</td>
<td>1 vs 2</td>
<td>0 vs 5</td>
<td>3 vs 4</td>
<td>1 vs 3</td>
</tr>
</tbody>
</table>

Mij=1 \iff 0 vs 1 or 1 vs 0
Mij=12 \iff 1 vs 6 or 6 vs 1

1 vs 6
M33 variable (M33=12)

T33a variable (T33a=6)

T33h variable (T33h=1)
### CP model: T variables

<table>
<thead>
<tr>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Period 1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T_{11}h$ vs</td>
<td>$T_{12}h$ vs</td>
<td>$T_{13}h$ vs</td>
<td>$T_{14}h$ vs</td>
<td>$T_{15}h$ vs</td>
<td>$T_{16}h$ vs</td>
<td>$T_{17}h$ vs</td>
</tr>
<tr>
<td>$T_{11}a$</td>
<td>$T_{12}a$</td>
<td>$T_{13}a$</td>
<td>$T_{14}a$</td>
<td>$T_{15}a$</td>
<td>$T_{16}a$</td>
<td>$T_{17}a$</td>
</tr>
<tr>
<td><strong>Period 2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T_{21}h$ vs</td>
<td>$T_{22}h$ vs</td>
<td>$T_{23}h$ vs</td>
<td>$T_{24}h$ vs</td>
<td>$T_{25}h$ vs</td>
<td>$T_{26}h$ vs</td>
<td>$T_{27}h$ vs</td>
</tr>
<tr>
<td>$T_{21}a$</td>
<td>$T_{22}a$</td>
<td>$T_{23}a$</td>
<td>$T_{24}a$</td>
<td>$T_{25}a$</td>
<td>$T_{26}a$</td>
<td>$T_{27}a$</td>
</tr>
<tr>
<td><strong>Period 3</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T_{31}h$ vs</td>
<td>$T_{32}h$ vs</td>
<td>$T_{33}h$ vs</td>
<td>$T_{34}h$ vs</td>
<td>$T_{35}h$ vs</td>
<td>$T_{36}h$ vs</td>
<td>$T_{37}h$ vs</td>
</tr>
<tr>
<td>$T_{31}a$</td>
<td>$T_{32}a$</td>
<td>$T_{33}a$</td>
<td>$T_{34}a$</td>
<td>$T_{35}a$</td>
<td>$T_{36}a$</td>
<td>$T_{37}a$</td>
</tr>
<tr>
<td><strong>Period 4</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T_{41}h$ vs</td>
<td>$T_{42}h$ vs</td>
<td>$T_{43}h$ vs</td>
<td>$T_{44}h$ vs</td>
<td>$T_{45}h$ vs</td>
<td>$T_{46}h$ vs</td>
<td>$T_{47}h$ vs</td>
</tr>
<tr>
<td>$T_{41}a$</td>
<td>$T_{42}a$</td>
<td>$T_{43}a$</td>
<td>$T_{44}a$</td>
<td>$T_{45}a$</td>
<td>$T_{46}a$</td>
<td>$T_{47}a$</td>
</tr>
</tbody>
</table>

$D(T_{ija}) = [1, n-1]$  
$D(T_{ijh}) = [0, n-2]$  
$T_{ijh} < T_{ija}$
CP model: M variables

<table>
<thead>
<tr>
<th></th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period 1</td>
<td>M11</td>
<td>M12</td>
<td>M13</td>
<td>M14</td>
<td>M15</td>
<td>M16</td>
<td>M17</td>
</tr>
<tr>
<td>Period 2</td>
<td>M21</td>
<td>M22</td>
<td>M23</td>
<td>M24</td>
<td>M25</td>
<td>M26</td>
<td>M27</td>
</tr>
<tr>
<td>Period 3</td>
<td>M31</td>
<td>M32</td>
<td>M33</td>
<td>M34</td>
<td>M35</td>
<td>M36</td>
<td>M37</td>
</tr>
<tr>
<td>Period 4</td>
<td>M41</td>
<td>M42</td>
<td>M43</td>
<td>M44</td>
<td>M45</td>
<td>M46</td>
<td>M47</td>
</tr>
</tbody>
</table>

\[ D(M_{ij}) = [1, n(n-1)/2] \]
CP model: constraints

- \( n \) teams and \( n-1 \) weeks and \( n/2 \) periods
- every two teams play each other exactly once
- every team plays one game in each week
- no team plays more than twice in the same period

<table>
<thead>
<tr>
<th></th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period 1</td>
<td>M11</td>
<td>M12</td>
<td>M13</td>
<td>M14</td>
<td>M15</td>
<td>M16</td>
<td>M17</td>
</tr>
<tr>
<td>Period 2</td>
<td>M21</td>
<td>M22</td>
<td>M23</td>
<td>M24</td>
<td>M25</td>
<td>M26</td>
<td>M27</td>
</tr>
<tr>
<td>Period 3</td>
<td>M31</td>
<td>M32</td>
<td>M33</td>
<td>M34</td>
<td>M35</td>
<td>M36</td>
<td>M37</td>
</tr>
<tr>
<td>Period 4</td>
<td>M41</td>
<td>M42</td>
<td>M43</td>
<td>M44</td>
<td>M45</td>
<td>M46</td>
<td>M47</td>
</tr>
</tbody>
</table>
**CP model: constraints**

- **n teams and n-1 weeks and n/2 periods**
- every two teams play each other exactly once
- every team plays one game in each week
- no team plays more than twice in the same period

<table>
<thead>
<tr>
<th></th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period 1</td>
<td>M11</td>
<td>M12</td>
<td>M13</td>
<td>M14</td>
<td>M15</td>
<td>M16</td>
<td>M17</td>
</tr>
<tr>
<td>Period 2</td>
<td>M21</td>
<td>M22</td>
<td>M23</td>
<td>M24</td>
<td>M25</td>
<td>M26</td>
<td>M27</td>
</tr>
<tr>
<td>Period 3</td>
<td>M31</td>
<td>M32</td>
<td>M33</td>
<td>M34</td>
<td>M35</td>
<td>M36</td>
<td>M37</td>
</tr>
<tr>
<td>Period 4</td>
<td>M41</td>
<td>M42</td>
<td>M43</td>
<td>M44</td>
<td>M45</td>
<td>M46</td>
<td>M47</td>
</tr>
</tbody>
</table>

Alldiff constraints defined on M variables
### CP model: constraints

- \( n \) teams and \( n-1 \) weeks and \( n/2 \) periods
- every two teams play each other exactly once
- every team plays one game in each week
- no team plays more than twice in the same period

<table>
<thead>
<tr>
<th>Period 1</th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T11h vs T11a</td>
<td>T12h vs T12a</td>
<td>T13h vs T13a</td>
<td>T14h vs T14a</td>
<td>T15h vs T15a</td>
<td>T16h vs T16a</td>
<td>T17h vs T17a</td>
</tr>
<tr>
<td>Period 2</td>
<td>T21h vs T21a</td>
<td>T22h vs T22a</td>
<td>T23h vs T23a</td>
<td>T24h vs T24a</td>
<td>T25h vs T25a</td>
<td>T26h vs T26a</td>
<td>T27h vs T27a</td>
</tr>
<tr>
<td>Period 3</td>
<td>T31h vs T31a</td>
<td>T32h vs T32a</td>
<td>T33h vs T33a</td>
<td>T34h vs T34a</td>
<td>T35h vs T35a</td>
<td>T36h vs T36a</td>
<td>T37h vs T37a</td>
</tr>
<tr>
<td>Period 4</td>
<td>T41h vs T41a</td>
<td>T42h vs T42a</td>
<td>T43h vs T43a</td>
<td>T44h vs T44a</td>
<td>T45h vs T45a</td>
<td>T46h vs T46a</td>
<td>T47h vs T47a</td>
</tr>
</tbody>
</table>
CP model: constraints

- n teams and n-1 weeks and n/2 periods
- every two teams play each other exactly once
- every team plays one game in each week
- no team plays more than twice in the same period

<table>
<thead>
<tr>
<th>Period 1</th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T11h vs</td>
<td>T12h vs</td>
<td>T13h vs</td>
<td>T14h vs</td>
<td>T15h vs</td>
<td>T16h vs</td>
<td>T17h vs</td>
</tr>
<tr>
<td></td>
<td>T11a</td>
<td>T12a</td>
<td>T13a</td>
<td>T14a</td>
<td>T15a</td>
<td>T16a</td>
<td>T17a</td>
</tr>
<tr>
<td>Period 2</td>
<td>T21h vs</td>
<td>T22h vs</td>
<td>T23h vs</td>
<td>T24h vs</td>
<td>T25h vs</td>
<td>T26h vs</td>
<td>T27h vs</td>
</tr>
<tr>
<td></td>
<td>T21a</td>
<td>T22a</td>
<td>T23a</td>
<td>T24a</td>
<td>T25a</td>
<td>T26a</td>
<td>T27a</td>
</tr>
<tr>
<td>Period 3</td>
<td>T31h vs</td>
<td>T32h vs</td>
<td>T33h vs</td>
<td>T34h vs</td>
<td>T35h vs</td>
<td>T36h vs</td>
<td>T37h vs</td>
</tr>
<tr>
<td></td>
<td>T31a</td>
<td>T32a</td>
<td>T33a</td>
<td>T34a</td>
<td>T35a</td>
<td>T36a</td>
<td>T37a</td>
</tr>
<tr>
<td>Period 4</td>
<td>T41h vs</td>
<td>T42h vs</td>
<td>T43h vs</td>
<td>T44h vs</td>
<td>T45h vs</td>
<td>T46h vs</td>
<td>T47h vs</td>
</tr>
<tr>
<td></td>
<td>T41a</td>
<td>T42a</td>
<td>T43a</td>
<td>T44a</td>
<td>T45a</td>
<td>T46a</td>
<td>T47a</td>
</tr>
</tbody>
</table>

For each week w:
Alldiff constraint defined on \{Tpwh, p=1..4\} U \{Tpwa, p=1..4\}
CP model: constraints

- n teams and n-1 weeks and n/2 periods
- every two teams play each other exactly once
- every team plays one game in each week
- no team plays more than twice in the same period

<table>
<thead>
<tr>
<th>Period 1</th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T11h vs T11a</td>
<td>T12h vs T12a</td>
<td>T13h vs T13a</td>
<td>T14h vs T14a</td>
<td>T15h vs T15a</td>
<td>T16h vs T16a</td>
<td>T17h vs T17a</td>
</tr>
<tr>
<td>Period 2</td>
<td>T21h vs T21a</td>
<td>T22h vs T22a</td>
<td>T23h vs T23a</td>
<td>T24h vs T24a</td>
<td>T25h vs T25a</td>
<td>T26h vs T26a</td>
<td>T27h vs T27a</td>
</tr>
<tr>
<td>Period 3</td>
<td>T31h vs T31a</td>
<td>T32h vs T32a</td>
<td>T33h vs T33a</td>
<td>T34h vs T34a</td>
<td>T35h vs T35a</td>
<td>T36h vs T36a</td>
<td>T37h vs T37a</td>
</tr>
<tr>
<td>Period 4</td>
<td>T41h vs T41a</td>
<td>T42h vs T42a</td>
<td>T43h vs T43a</td>
<td>T44h vs T44a</td>
<td>T45h vs T45a</td>
<td>T46h vs T46a</td>
<td>T47h vs T47a</td>
</tr>
</tbody>
</table>
CP model: constraints

- n teams and n-1 weeks and n/2 periods
- every two teams play each other exactly once
- every team plays one game in each week
- no team plays more than twice in the same period

<table>
<thead>
<tr>
<th>Period 1</th>
<th>Period 2</th>
<th>Period 3</th>
<th>Period 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Week 1</td>
<td>Week 2</td>
<td>Week 3</td>
<td>Week 4</td>
</tr>
<tr>
<td>T11h vs</td>
<td>T12h vs</td>
<td>T13h vs</td>
<td>T14h vs</td>
</tr>
<tr>
<td>T11a</td>
<td>T12a</td>
<td>T13a</td>
<td>T14a</td>
</tr>
<tr>
<td></td>
<td>T21h vs</td>
<td>T23h vs</td>
<td>T25h vs</td>
</tr>
<tr>
<td></td>
<td>T21a</td>
<td>T23a</td>
<td>T25a</td>
</tr>
<tr>
<td></td>
<td>T22h vs</td>
<td>T24h vs</td>
<td>T26h vs</td>
</tr>
<tr>
<td></td>
<td>T22a</td>
<td>T24a</td>
<td>T26a</td>
</tr>
<tr>
<td></td>
<td>T31h vs</td>
<td>T33h vs</td>
<td>T35h vs</td>
</tr>
<tr>
<td></td>
<td>T31a</td>
<td>T33a</td>
<td>T35a</td>
</tr>
<tr>
<td></td>
<td>T32h vs</td>
<td>T34h vs</td>
<td>T36h vs</td>
</tr>
<tr>
<td></td>
<td>T32a</td>
<td>T34a</td>
<td>T36a</td>
</tr>
<tr>
<td></td>
<td>T32h vs</td>
<td>T34h vs</td>
<td>T36h vs</td>
</tr>
<tr>
<td></td>
<td>T32a</td>
<td>T34a</td>
<td>T36a</td>
</tr>
<tr>
<td></td>
<td>T41h vs</td>
<td>T43h vs</td>
<td>T45h vs</td>
</tr>
<tr>
<td></td>
<td>T41a</td>
<td>T43a</td>
<td>T45a</td>
</tr>
<tr>
<td></td>
<td>T42h vs</td>
<td>T44h vs</td>
<td>T46h vs</td>
</tr>
<tr>
<td></td>
<td>T42a</td>
<td>T44a</td>
<td>T46a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T43h vs</td>
<td>T45h vs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T43a</td>
<td>T45a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T44h vs</td>
<td>T46h vs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T44a</td>
<td>T46a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T47h vs</td>
<td>T47a</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Global cardinality constraint defined on
\{Tpwh, w=1..7\} U \{Tpwa, w=1..7\}
every team t is taken at most 2
CP model: constraints

- For each slot the two T variables and the M variable must be linked together; example:
  M12 = game T12h vs T12a
CP model: constraints

- For each slot the two T variables and the M variable must be linked together; example:
  M12 = game T12h vs T12a

- For each slot we add Cij a ternary constraint defined on the two T variables and the M variable; example:
  C12 defined on \{T12h, T12a, M12\}
CP model: constraints

- For each slot the two T variables and the M variable must be linked together; example:
  \[ M_{12} = \text{game } T_{12h} \text{ vs } T_{12a} \]

- For each slot we add \( C_{ij} \) a ternary constraint defined on the two T variables and the M variable; example:
  \( C_{12} \) defined on \( \{T_{12h}, T_{12a}, M_{12}\} \)

- \( C_{ij} \) are defined by the list of allowed tuples:
  for \( n=4 \): \( \{(0,1,1),(0,2,2),(0,3,3),(1,2,4),(1,3,5),(2,3,6)\} \)
  \( (1,2,4) \) means game 1 vs 2 is the game number 4
CP model: constraints

- For each slot the two T variables and the M variable must be linked together; example:
  \[ M_{12} = \text{game } T_{12h} \text{ vs } T_{12a} \]

- For each slot we add \( C_{ij} \) a ternary constraint defined on the two T variables and the M variable; example:
  \( C_{12} \) defined on \( \{T_{12h}, T_{12a}, M_{12}\} \)

- \( C_{ij} \) are defined by the list of allowed tuples:
  for \( n=4 \): \( \{(0,1,1),(0,2,2),(0,3,3),(1,2,4),(1,3,5),(2,3,6)\} \)
  \( (1,2,4) \) means game 1 vs 2 is the game number 4

- All these constraints have the same list of allowed tuples

- Efficient arc consistency algorithm for this kind of constraint is known
First model

Introduction of a dummy column

<table>
<thead>
<tr>
<th></th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
<th>Dummy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period 1</td>
<td>0 vs 1</td>
<td>0 vs 2</td>
<td>4 vs 7</td>
<td>3 vs 6</td>
<td>3 vs 7</td>
<td>1 vs 5</td>
<td>2 vs 4</td>
<td>. vs .</td>
</tr>
<tr>
<td>Period 2</td>
<td>2 vs 3</td>
<td>1 vs 7</td>
<td>0 vs 3</td>
<td>5 vs 7</td>
<td>1 vs 4</td>
<td>0 vs 6</td>
<td>5 vs 6</td>
<td>. vs .</td>
</tr>
<tr>
<td>Period 3</td>
<td>4 vs 5</td>
<td>3 vs 5</td>
<td>1 vs 6</td>
<td>0 vs 4</td>
<td>2 vs 6</td>
<td>2 vs 7</td>
<td>0 vs 7</td>
<td>. vs .</td>
</tr>
<tr>
<td>Period 4</td>
<td>6 vs 7</td>
<td>4 vs 6</td>
<td>2 vs 5</td>
<td>1 vs 2</td>
<td>0 vs 5</td>
<td>3 vs 4</td>
<td>1 vs 3</td>
<td>. vs .</td>
</tr>
</tbody>
</table>
First model

Introduction of a dummy column

<table>
<thead>
<tr>
<th></th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
<th>Dummy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period 1</td>
<td>0 vs 1</td>
<td>0 vs 2</td>
<td>4 vs 7</td>
<td>3 vs 6</td>
<td>3 vs 7</td>
<td>1 vs 5</td>
<td>2 vs 4</td>
<td>. vs .</td>
</tr>
<tr>
<td>Period 2</td>
<td>2 vs 3</td>
<td>1 vs 7</td>
<td>0 vs 3</td>
<td>5 vs 7</td>
<td>1 vs 4</td>
<td>0 vs 6</td>
<td>5 vs 6</td>
<td>. vs .</td>
</tr>
<tr>
<td>Period 3</td>
<td>4 vs 5</td>
<td>3 vs 5</td>
<td>1 vs 6</td>
<td>0 vs 4</td>
<td>2 vs 6</td>
<td>2 vs 7</td>
<td>0 vs 7</td>
<td>. vs .</td>
</tr>
<tr>
<td>Period 4</td>
<td>6 vs 7</td>
<td>4 vs 6</td>
<td>2 vs 5</td>
<td>1 vs 2</td>
<td>0 vs 5</td>
<td>3 vs 4</td>
<td>1 vs 3</td>
<td>. vs .</td>
</tr>
</tbody>
</table>

We can prove that:
• each team occurs exactly twice for each period
First model

Introduction of a dummy column

We can prove that:
• each team occurs exactly twice for each period

<table>
<thead>
<tr>
<th></th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
<th>Dummy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period 1</td>
<td>0 vs 1</td>
<td>0 vs 2</td>
<td>4 vs 7</td>
<td>3 vs 6</td>
<td>3 vs 7</td>
<td>1 vs 5</td>
<td>2 vs 4</td>
<td>5 vs 6</td>
</tr>
<tr>
<td>Period 2</td>
<td>2 vs 3</td>
<td>1 vs 7</td>
<td>0 vs 3</td>
<td>5 vs 7</td>
<td>1 vs 4</td>
<td>0 vs 6</td>
<td>5 vs 6</td>
<td>. vs .</td>
</tr>
<tr>
<td>Period 3</td>
<td>4 vs 5</td>
<td>3 vs 5</td>
<td>1 vs 6</td>
<td>0 vs 4</td>
<td>2 vs 6</td>
<td>2 vs 7</td>
<td>0 vs 7</td>
<td>. vs .</td>
</tr>
<tr>
<td>Period 4</td>
<td>6 vs 7</td>
<td>4 vs 6</td>
<td>2 vs 5</td>
<td>1 vs 2</td>
<td>0 vs 5</td>
<td>3 vs 4</td>
<td>1 vs 3</td>
<td>. vs .</td>
</tr>
</tbody>
</table>
First model

Introduction of a dummy column

<table>
<thead>
<tr>
<th></th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
<th>Dummy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period 1</td>
<td>0 vs 1</td>
<td>0 vs 2</td>
<td>4 vs 7</td>
<td>3 vs 6</td>
<td>3 vs 7</td>
<td>1 vs 5</td>
<td>2 vs 4</td>
<td>5 vs 6</td>
</tr>
<tr>
<td>Period 2</td>
<td>2 vs 3</td>
<td>1 vs 7</td>
<td>0 vs 3</td>
<td>5 vs 7</td>
<td>1 vs 4</td>
<td>0 vs 6</td>
<td>5 vs 6</td>
<td>2 vs 4</td>
</tr>
<tr>
<td>Period 3</td>
<td>4 vs 5</td>
<td>3 vs 5</td>
<td>1 vs 6</td>
<td>0 vs 4</td>
<td>2 vs 6</td>
<td>2 vs 7</td>
<td>0 vs 7</td>
<td>1 vs 3</td>
</tr>
<tr>
<td>Period 4</td>
<td>6 vs 7</td>
<td>4 vs 6</td>
<td>2 vs 5</td>
<td>1 vs 2</td>
<td>0 vs 5</td>
<td>3 vs 4</td>
<td>1 vs 3</td>
<td>0 vs 7</td>
</tr>
</tbody>
</table>

We can prove that:

- each team occurs exactly twice for each period
**First model**

**Introduction of a dummy column**

<table>
<thead>
<tr>
<th></th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
<th>Dummy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period 1</td>
<td>0 vs 1</td>
<td>0 vs 2</td>
<td>4 vs 7</td>
<td>3 vs 6</td>
<td>3 vs 7</td>
<td>1 vs 5</td>
<td>2 vs 4</td>
<td>5 vs 6</td>
</tr>
<tr>
<td>Period 2</td>
<td>2 vs 3</td>
<td>1 vs 7</td>
<td>0 vs 3</td>
<td>5 vs 7</td>
<td>1 vs 4</td>
<td>0 vs 6</td>
<td>5 vs 6</td>
<td>2 vs 4</td>
</tr>
<tr>
<td>Period 3</td>
<td>4 vs 5</td>
<td>3 vs 5</td>
<td>1 vs 6</td>
<td>0 vs 4</td>
<td>2 vs 6</td>
<td>2 vs 7</td>
<td>0 vs 7</td>
<td>1 vs 3</td>
</tr>
<tr>
<td>Period 4</td>
<td>6 vs 7</td>
<td>4 vs 6</td>
<td>2 vs 5</td>
<td>1 vs 2</td>
<td>0 vs 5</td>
<td>3 vs 4</td>
<td>1 vs 3</td>
<td>0 vs 7</td>
</tr>
</tbody>
</table>

We can prove that:
- each team occurs exactly twice for each period
- each team occurs exactly once in the dummy column
First model

Introduction of a dummy column

<table>
<thead>
<tr>
<th></th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
<th>Dummy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period 1</td>
<td>0 vs 1</td>
<td>0 vs 2</td>
<td>4 vs 7</td>
<td>3 vs 6</td>
<td>3 vs 7</td>
<td>1 vs 5</td>
<td>2 vs 4</td>
<td>5 vs 6</td>
</tr>
<tr>
<td>Period 2</td>
<td>2 vs 3</td>
<td>1 vs 7</td>
<td>0 vs 3</td>
<td>5 vs 7</td>
<td>1 vs 4</td>
<td>0 vs 6</td>
<td>5 vs 6</td>
<td>2 vs 4</td>
</tr>
<tr>
<td>Period 3</td>
<td>4 vs 5</td>
<td>3 vs 5</td>
<td>1 vs 6</td>
<td>0 vs 4</td>
<td>2 vs 6</td>
<td>2 vs 7</td>
<td>0 vs 7</td>
<td>1 vs 3</td>
</tr>
<tr>
<td>Period 4</td>
<td>6 vs 7</td>
<td>4 vs 6</td>
<td>2 vs 5</td>
<td>1 vs 2</td>
<td>0 vs 5</td>
<td>3 vs 4</td>
<td>1 vs 3</td>
<td>0 vs 7</td>
</tr>
</tbody>
</table>

• The problem is exactly the same
• The solver is helped by such constraint. It can deduce some inconsistencies more quickly
First model: strategies

- Break symmetries: 0 vs w appears in week w
First model: strategies

- Break symmetries: 0 vs w appears in week w
- Teams are instantiated:
  - the most instantiated team is chosen
  - the slots that has the less remaining possibilities (Tijh or Tija is minimal) is instantiated with that team
First model: results

<table>
<thead>
<tr>
<th># teams</th>
<th># fails</th>
<th>Time (in s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
<td>0.01</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>0.03</td>
</tr>
<tr>
<td>8</td>
<td>32</td>
<td>0.08</td>
</tr>
<tr>
<td>10</td>
<td>417</td>
<td>0.8</td>
</tr>
<tr>
<td>12</td>
<td>41</td>
<td>0.2</td>
</tr>
<tr>
<td>14</td>
<td>3,514</td>
<td>9.2</td>
</tr>
<tr>
<td>16</td>
<td>1,112</td>
<td>4.2</td>
</tr>
<tr>
<td>18</td>
<td>8,756</td>
<td>36</td>
</tr>
<tr>
<td>20</td>
<td>72,095</td>
<td>338</td>
</tr>
<tr>
<td>22</td>
<td>6,172,672</td>
<td>10h</td>
</tr>
<tr>
<td>24</td>
<td>6,391,470</td>
<td>12h</td>
</tr>
</tbody>
</table>

MIPLIB

MIP solver limit
Car sequencing

- Car sequencing problems arise on assembly lines in factories in the automotive industry.
- Many different types of cars can be built on an assembly line.
- A car = a basic car + options (color, motor, telephone, seats, …)
- A car = a configuration of options
Capacity of an option

- For practical reasons: a given option cannot be installed on every vehicle on the line
- **Capacity of an option**: ratio $p/q$, for any sequence of $q$ cars on the line, at most $p$ of them can have the option
The problem

- Determine in which order cars should be assembled, while:
  - building a certain number of cars per configuration
  - satisfying the capacity of each option.
Example

<table>
<thead>
<tr>
<th>opt cap</th>
<th>configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>0</td>
<td>1/2X</td>
</tr>
<tr>
<td>1</td>
<td>2/3 X</td>
</tr>
<tr>
<td>2</td>
<td>1/3X X</td>
</tr>
<tr>
<td>3</td>
<td>2/5X X</td>
</tr>
<tr>
<td>4</td>
<td>1/5 X</td>
</tr>
<tr>
<td>#cars</td>
<td>1 1 2 2 2 2</td>
</tr>
</tbody>
</table>
### Example

<table>
<thead>
<tr>
<th>opt cap</th>
<th>configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>0</td>
<td>1/2X</td>
</tr>
<tr>
<td>1</td>
<td>2/3</td>
</tr>
<tr>
<td>2</td>
<td>1/3X</td>
</tr>
<tr>
<td>3</td>
<td>2/5X</td>
</tr>
<tr>
<td>4</td>
<td>1/5</td>
</tr>
<tr>
<td>#cars</td>
<td>1 1 2 2 2 2</td>
</tr>
</tbody>
</table>

- Sequences 4,4 or 4,5 or 0,4 or 0,5 are forbidden
### Example

<table>
<thead>
<tr>
<th>opt cap</th>
<th>configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>0</td>
<td>1/2X</td>
</tr>
<tr>
<td>1</td>
<td>2/3</td>
</tr>
<tr>
<td>2</td>
<td>1/3X</td>
</tr>
<tr>
<td>3</td>
<td>2/5X</td>
</tr>
<tr>
<td>4</td>
<td>1/5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#cars</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 2 2 2 2</td>
</tr>
</tbody>
</table>

- Sequences 2,2,1 or 2,3,0 are allowed
- Sequences 2,2,3 or 5,3,2 are forbidden
Car sequencing: model

- A variable for each car
- The domain of a variable = the set of all possible configurations
- Constraints:
  - 1 global cardinality constraint which defines the number of time each configuration has to be built
  - 1 global sequencing constraint for each option
Car sequencing with CP

- **Global Sequencing Constraint:**
  \[ \text{GSC}(X,V,\text{min},\text{max},q,\{l_i\},\{u_i\}) \]

- A GCC (Global cardinality constraint) for the values of V + constraints stating that for each sequence S of q consecutive variables, at least min and at most max variables of S takes their values in V.
Global Sequencing Constraint

- GSC(\(X,V,\text{min, max, q, }\{\text{li}\}, \{\text{ui}\}\))
- Idea of the filtering algorithm: represent a GSC by a set of GCC + an ad hoc constraint that will link these GCCs
Global Cardinality Constraint

- GCC(X,\{li\},\{ui\})
- Defined on a set X of variables, the number of times each value \(v_i\) can be taken must be in a given interval \([l_i, u_i]\)
- Example: \(D(x_1) = \{a, b, c, d\}\), \(D(x_2) = \{a, b, c, d\}\), \(D(x_3) = \{b, c\}\), \(D(x_4) = \{c, d\}\). Values b and c must be taken at most 2, values a and d must be taken at least 1.
Filtering algorithm for GCC

- Can be represented by a flow problem:

![Flow Problem Diagram]

Arc orientation

Flow value: $|X| = 4$
Filtering algorithm for GCC

A solution:

Arc orientation

Flow value: $|X| = 4$
Filtering algorithm for GCC

- Arc from variable to value that does not belong to any solution with a flow value = 4 can be removed.

Arc orientation

Flow value: $|X| = 4$
Filtering algorithm for GCC

- Arc from variable to value that does not belong to any solution with a flow value = 4 can be removed.

Arc orientation

Flow value: $|X| = 4$
Filtering algorithm for GCC

- Compute a feasible flow
- Compute the strongly connected components
- Remove every arc with flow value = 0 for which the ends belong to two different components
- Linear algorithm achieving arc consistency
- Work well due to (0,1) arcs
Global Sequencing Constraint

- GSC(X,V,min,max,q,\{l_i\},\{u_i\})
- A GCC (Global cardinality constraint) for the values of V + constraint stating that for each sequence S of q consecutive variables, at least min and at most max variables of S takes their values in V.
Abstract Values

- GSC(X,V,…): the values of D(X) - V are not constrained individually. For each sequence S they can be replaced (inside the constraint) by e(S) an abstract value.
Abstract Value

- $GSC(X, V=\{a, b\}, \text{min}=0, \text{max}=1, q=2, \ldots)$

Constraints on values not in $V$ are no longer considered
Abstract Value

- $GSC(X, V=\{a, b\}, \text{min}=0, \text{max}=1, q=2, \ldots)$

Values c and d does not belong to V, they are replaced by $e(S1)$, for the sequence
Abstract Value

- GSC($X, V=\{a, b\}, \text{min}=0, \text{max}=1, q=2, \ldots$)

Value $e(S1)$ must be taken at least $|S| - \text{max} = 2 - 1 = 1$ and at most $q - \text{min} = 2 - 0 = 2$
Split of X into a partition of Sequence

- GSC(X, V={a,b}, min=0, max=1, q=2, …)

Red and Yellow arcs represent the constraints on sequences
Black arcs represent the global constraints on values of V
Split of X into partition of sequences

- Problem: an exponential number of partitions exist
- Solution: what is needed is just to have each sequence represented at least once. We propose to have $|X|$ partitions simultaneously.
- For our example: $P_1=\{(x_1,x_2),(x_3,x_4)\}$ and $P_2=\{(x_1),(x_2,x_3),(x_4)\}$
Partitions of sequences

<table>
<thead>
<tr>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
<th>x5</th>
<th>x6</th>
<th>x7</th>
<th>x8</th>
<th>x9</th>
<th>x10</th>
</tr>
</thead>
<tbody>
<tr>
<td>S11</td>
<td>S12</td>
<td>S13</td>
<td>S14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S21</td>
<td>S22</td>
<td>S23</td>
<td>S24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S31</td>
<td>S32</td>
<td>S33</td>
<td>S34</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Partitions of sequences

- Communication between sequences is necessary to solve the problem
Partitions of sequences

$S_i \rightarrow S_j$ means $S_j$ is a successor of $S_i$
Successor of a sequence

- The successor of a sequence is the same sequence translated by one variable.

  S1: x1 x2 x3
  S2: x2 x3 x4

- \( \#e(S1) \) and \( \#e(S2) \) can be linked:
  \[ |\#e(S1) - \#e(S2)| \leq 1 \]
Constraints between sequences

- \(|#e(S1) - #e(S2)| \leq 1\) must be implemented carefully:
  S1: x1 x2 x3
  S2: x2 x3 x4

- \(x1 = e(S1)\) and \(x4 \neq e(S2)\) \(\iff\) \(#e(S1) = #e(S2) + 1\)

- \((x1 = e(S1)\) and \(x4 = e(S2)\)) or \((x1 \neq e(S1)\) and \(x4 \neq e(S2)\)) \(\iff\) \(#e(S1) = #e(S2)\)

- \(x1 \neq e(S1)\) and \(x4 = e(S2)\) \(\iff\) \(#e(S1) = #e(S2) - 1\)
Car sequencing

- For each option a Global Sequencing Constraint is defined
- The filtering algorithm previously presented is used
Variable-Value Strategy

- How can we choose the next variable to instantiate and the value to assign to this variable?
Variable-Value Strategy

- How can we choose the next variable to instantiate and the value to assign to this variable?
  1) Choose the most constraint option (e.g. 50 cars needs option 0 with a capacity 1/2)
  2) Choose the configurations (i.e. values) that requires this option
  3) Begin by the cars (i.e. variable) in the middle of the assembly line
Results

- Instances provided by Barbara Smith: 100 cars, 25 configurations, 5 options
- We proved (in 1997) that:
  - one instance has no solution in 3.5s
  - one instance has no solution in 422s. As far as we know, this is currently the only one method which is able to obtain this result
- We solve some other open problems
Plan

- General Principles of Constraint Programming
- Global Constraints: advantages
- How to write a filtering algorithm?
- Examples: sports scheduling and car sequencing
- Over-constrained problems
- Discussion: quality of a FA, incrementality, closure, incomplete algorithms, power of a FA
- Conclusion
A soft constraint is a constraint that can be violated. The violation can be associated with a cost that can be:

- The same for any violation
- Depends on the violation

Example: \( x < y \), if \( x \geq y \) we can have:

- A fixed cost: \( \text{cost} = c \)
- A cost depending on the violation: \( \text{cost} = x - y \) or \( \text{cost} = (x-y)^2 \)
Soft constraint and Filtering algorithm

- When the violation is accepted this means that we accept that any combination of values satisfies the constraint.
Soft constraint and Filtering algorithm

- When the violation is accepted this means that we accept that any combination of values satisfies the constraint.
- Roughly, the constraint become an universal constraint associating a cost with any tuple, so we loose the structure of the constraint.
Soft constraint and Filtering algorithm

- When the violation is accepted this means that we accept that any combination of values satisfies the constraint.
- Roughly, the constraint become an universal constraint associating a cost with any tuple, so we loose the structure of the constraint
- Problem with filtering algorithm (FA):
  - FA exploits the structure of the constraints
  - FA are not efficient when everything is possible!
Soft constraint and Filtering algorithm

- When the violation is accepted this means that we accept that any combination of values satisfies the constraint.
- Roughly, the constraint become an universal constraint associating a cost with any tuple, so we lose the structure of the constraint
- Problem with filtering algorithm (FA):
  - FA exploits the structure of the constraints
  - FA are not efficient when everything is possible!
- Filtering for soft depends mainly on back propagation. Problem with global constraints
Meta Constraint

- $s_i > 0$ expresses that $C_i$ is violated (distance to satisfaction)
- $s_i = 0$ expresses that $C_i$ is satisfied
- $D(s_i)$ is an integer domain
- Each “soft” constraint is replaced by the disjunction:

$$[ (s = 0) \land C ] \lor [ (s > 0) \land \neg C ]$$
Since valuations are expressed through variables, constraints on these variables can be added in order to express “global rules” on violations.
Max-SAT = Satisfiability Sum Constraint

- In the ssc, each constraint $C_i$ is replaced by:
  \[ (C_i \land (u_i = 0)) \lor (\neg C_i \land (u_i = 1)) \]

- A variable $\text{unsat}$ is used to express the objective:

\[ \sum_{i=1}^{\# C_i} u_i \]

\[ \text{unsat} = \sum_{i=1}^{\# C_i} u_i \]
Advantages of This Model

- Classical constraint optimization problem
  - Direct integration into a solver
  - Any search algorithm can be used, not only a Branch and Bound based one.
- When a value is assigned to $u_i \in U$, the filtering algorithm associated with $C_i$ (resp. $\neg C_i$) can be used
- No hypothesis is made on constraints (arity)
Advantages of This Model

- **Integration of cost within the constraint**
  Costs as a variable:
  - the costs of violations have a structure:
    if \(x \leq y\) is violated then cost = \(x - y\)
    We can use this information.

- General definitions of cost of violations
- Global soft constraints
- Constraints on violations can be easily defined
Use of the structure of the violation:

\[ x \leq y \]

- **Structure**
  - If the constraint is satisfied then \( cost = 0 \)
  - If the constraint is violated then \( cost = x - y \)

- **Filtering Algorithm:**
  - \( D(x) = [90000,100000], \ D(y) = [99990,200000] \)
  - We deduce immediately \( \max(cost) = \max(x) - \min(y) = 10 \)
General definition of the cost of violation

- Two different general costs:
  - Variables based violation cost
  - Primal Graph Based violation cost

- Some others see papers at CP-AI-OR’04 (Beldiceanu and Petit) and papers at workshop on soft constraints at CP’04.
Variable based violation cost

- How many variables must be removed to satisfy the constraint?
- \text{Alldiff}\{x_1, x_2, x_3, x_4, x_5\)
  - \((a, a, a, b, b)\) cost = 3
  - \((a, a, a, a, b)\) cost = 3
For a global constraint corresponding to a conjunction of constraints. Number of the constraints in the conjunction that are violated

- \text{Alldiff}\{x1,x2,x3,x4,x5\})
- \text{(a,a,a,b,b) cost} = \text{triangle}(a,a,a) + \text{pair} (b,b)
  \quad = 3 + 2 = 5
- \text{(a,a,a,a,b) cost} = \text{quadrangle} (a,a,a,a)
  \quad = 6
All Different constraint

The same value assigned to 2 variables $\rightarrow$ 1 violation

The same value assigned to 3 variables $\rightarrow$ 3 violations

The same value assigned to 4 variables $\rightarrow$ 6 violations

$n$ variables $\rightarrow n(n-1)/2$ violations
Soft global constraints

- For the alldiff, GCC, stretch, and regular constraints specific algorithms have been designed. These FA are able to take into account a cost variable w.r.t. the defined cost (see papers at CP conferences)
Global constraints with costs

- Integration of the costs within the constraint is quite important
- Alldiff with costs: quite important
  - [Caseau & Laburthe CP98] only consistency checking
  - [Focacci & Milano CP-AI-OR 99] filtering based on reduced costs
  - [Regin CP99] arc consistency
Plan

- General Principles of Constraint Programming
- Global Constraints: advantages
- How to write a filtering algorithm?
- Examples: sports scheduling and car sequencing
- Over-constrained problems
- Discussion: quality of a FA, incrementality, closure, incomplete algorithms, power of a FA
- Conclusion
Good Filtering Algorithm

- What is a good filtering algorithm? Or what is a poor one? *(The FA is considered and not the idea of using an OR algorithm)*

- GAC-Schema:
  - After 1 modification:
    - consistency in $O(CC)$
    - arc consistency in $O(ndCC)$
  - After $k$ modifications:
    - consistency in $O(CC)$
    - arc consistency in $O(ndCC)$
Good or Poor?

- If $O(CC)$ for consistency then arc consistency:
  - $O(ndCC)$ poor
  (reached by GAC- Schema)
Good or Poor?

- If $O(CC)$ for consistency then arc consistency:
  - $O(ndCC)$ poor
  (reached by GAC-Schema)
  - $O(CC)$ good!
Good or Poor?

- If $O(CC)$ for consistency then arc consistency:
  - $O(ndCC)$ poor
  (reached by GAC-Schema)
  - $O(CC)$ good!
  - and between? Not so bad...
Good FA

- Alldiff,
- global cardinality constraint
Medium FA

- Algorithm such that arc consistency is in: \( O(nCC) \) (a factor of \( d \) is gained):
  - Global cardinality with cost
  - Symmetric alldiff (alldiff U \( \{x_i = j \iff x_j = i\} \))
  - sum with binary inequalities
  - edge-finder
Poor FA?

- I hope there is none :-)
- Be careful with algorithms that successively try all values of variables (or ranges)
Perfect FA?

- Idea: FA has no cost.
- Complexity of the FA is always the same as the complexity of the consistency checking algorithm.
- All the possible cases are considered and not only the worst case.
- Another possibility $O(1)$ per deletion.
- AllDiff: 1 modification: if the deleted arc does not belong to the current maximum matching, then consistency in $O(1)$, AC in $O(nd)$: not perfect.
- No perfect FA is known? (maybe only $x < y$)
Incremental algorithms

- An incremental approach is not always the best. (cf IJCAI-2001 paper on AC-2001)
- The consequence of the deletions is a good approach if the number of modifications is less than the number of remaining things. Otherwise it is not good.
- The incremental aspect is quite important for a FA
AC-2001 vs AC-6

If $\Delta(j) = \{a, b, c\}$ and $\Delta(i) = \{a, b, c\}$ then:
- recomputation from scratch: 4 operations
- study of the consequences of the deletions: 6 operations
Adaptive Algorithm: Adaptive AC

- $A = \sum_{v \in \Delta(j)} (|Svj| + 1)$
- $B = |D(i)|$
- If $A < B$ then run AC-6
- If $B < A$ then run AC-2001

If $2|\Delta(j)| < |D(i)|$ then run AC-6
else run AC-2001
Closure or not?

- Is the FA closed w.r.t a property?
- Consider the values deleted by the FA. The consequence of these new deletions can be:
  1) taken into account by the same pass of the FA (alldiff)
  2) ignored by the same pass of the FA (Table)
- 1) no need to call again the FA
  2) need to call again the FA
- It is a choice.
Amortized Complexity

- It is possible to define the complexity in regards to the number of deletions (ex: O(CC) per deletion)
- Symmetric alldiff:
  AC: from every var, run algorithm A. Algorithm A can remove some values.
  AC Complexity: nO(A). Pb systematic.
- Other FA: pick one variable, run A, and set k=#deletions. You gain k runs!
  Complexity O(A) per deletion
Incomplete algorithms

- The constraint is an NP-Hard problem
- Try to characterize what is done
- useful in practice but sometimes difficult to handle:
  - no fixpoint (largest clique depends on the way the graph is defined):
  - less constraints can lead to more pruning
  - debug is difficult
- global sequencing constraint (NP-Hard with fixpoint)
Power of Filtering Algorithms

- Arc consistency is a strong property but it is sometimes costly
- Weaker consistencies exist: range consistency, bound consistency (see nice papers of C-G Quimper, A. Lopez-Ortiz et al about alldiff and GCC)
- However, arc consistency has some advantages
Advantages of Arc Consistency

- AC is much more robust. During the modeling phase it is useful to use strong FA. It is rare to be able to solve a problem with weaker consistency and not with AC.
- There is a room for improvements of AC algorithms.
- For binary constraints old story FC vs MAC.
- We should study more strong properties like Singleton Arc Consistency.
Plan

- General Principles of Constraint Programming
- Global Constraints: advantages
- How to write a filtering algorithm?
- Examples: sports scheduling and car sequencing
- Over-constrained problems
- Discussion: quality of a FA, incrementality, closure, incomplete algorithms, power of a FA

- Conclusion
Conclusion

- Filtering algorithm are one of the main strength of CP. Define your model by using them.
- If you write an FA:
  - try to write a good or a medium one. Do not forget that GAC-Schema exists
  - take care of the semantics of the constraint and especially the triggering of the FA
Conclusion

- Incremental algorithms are not always the best.
- General filtering algorithms are efficient in lack of other algorithms, when some predefined FA exist, use them.
- Over-constrained problems: use the constraint structure