Soft Global Constraints

Tutorial - CP 2009

Willem-Jan van Hoeve

Tepper School of Business
Carnegie Mellon University
Outline

• Constraint programming in brief
• Soft constraints
• Soft `alldifferent` constraint
• Soft `global cardinality` constraint
• Soft `regular` constraint
• Other soft global constraints
• Constraint-based local search
• Conclusions and perspectives
Main references for this tutorial


(contains many more references)
Constraint Programming
Example:

variables/domains \( x_1 \in \{1,2\}, x_2 \in \{0,1,2,3\}, x_3 \in \{2,3\} \)

constraints
- \( x_1 > x_2 \)
- \( x_1 + x_2 = x_3 \)
- \text{alldifferent}(x_1,x_2,x_3)
Example:

variables/domains  \( x_1 \in \{1,2\}, x_2 \in \{0,1,2,3\}, x_3 \in \{2,3\} \)

constraints

\( x_1 > x_2 \)
\( x_1 + x_2 = x_3 \)
\( \text{alldifferent}(x_1, x_2, x_3) \)
Example:

variables/domains  \( x_1 \in \{1\}, \ x_2 \in \{0,1\}, \ x_3 \in \{2,3\} \)

constraints

\( x_1 > x_2 \)
\( x_1 + x_2 = x_3 \)
\( \text{alldifferent}(x_1,x_2,x_3) \)
Example:

variables/domains  \( x_1 \in \{2\}, \ x_2 \in \{0,1\}, \ x_3 \in \{2,3\} \)

constraints

\( x_1 > x_2 \)

\( x_1 + x_2 = x_3 \)

\( \text{alldifferent}(x_1, x_2, x_3) \)
Goal: Remove as many inconsistent values as efficiently possible for each constraint individually

More filtering: Group constraints together?

Problem: Solving arbitrary conjunction of constraints is NP-hard

Solution:

- group constraints together that occur frequently in applications, and capture tractable structure
- result is called a global constraint
- Examples: alldifferent, cumulative, global-cardinality, among, sequence, circuit, element, regular, ...
Soft Constraints
Over-constrained problems

- Assign seats for overbooked airplane; no solution that carries all passengers
- Create roster for employees with conflicting preferences
- Factory wants to satisfy demands of all customers, but has limited resources
  (Many industrial problems are essentially over-constrained)

A CP solver will report that no solution exists. How to find acceptable ‘solution’?
- Soften (some of) the constraints of the problem
- Compute solution that minimizes conflicts or maximizes satisfaction
A **hard** constraint must always be satisfied.  
A **soft** constraint does not need to be satisfied, but we would like it to be.

**Example:**

Over-constrained CSP with two soft constraints:

\[ x_1 \in \{1,2\}, \ x_2 \in \{2,3\}, \ x_3 \in \{2,3\} \]

(i) \[ x_1 > x_2 \]

(ii) \[ x_1 + x_2 = x_3 \]

\[
\begin{align*}
(x_1,x_2,x_3) &= (1,2,2) \quad \text{both constraints violated} \\
(x_1,x_2,x_3) &= (1,2,3) \quad \text{constraint (i) violated} \\
(x_1,x_2,x_3) &= (1,3,2) \quad \text{both constraints violated} \\
(x_1,x_2,x_3) &= (1,3,3) \quad \text{both constraints violated} \\
(x_1,x_2,x_3) &= (2,2,2) \quad \text{both constraints violated} \\
(x_1,x_2,x_3) &= (2,2,3) \quad \text{both constraints violated} \\
(x_1,x_2,x_3) &= (2,3,2) \quad \text{both constraints violated} \\
(x_1,x_2,x_3) &= (2,3,3) \quad \text{both constraints violated}
\end{align*}
\]

minimum violation or maximum satisfaction
‘Traditional’ approaches to soft CSPs

- **Partial CSP** [Freuder & Wallace, 1992]
  maximize number of satisfied constraints (previous example)

- **Weighted CSP** [Larossa, 2002]
  associate a weight to each constraint
  maximize weighted sum of satisfied constraints

- **Possibilistic CSP** [Schiex, 1992]
  associate weight to each constraint representing its importance
  hierarchical satisfaction of most important constraints, i.e. maximize
  smallest weight over all satisfied constraints

- **Fuzzy CSP** [Dubois et al., 1993] [Ruttkay, 1994]
  associate weight to each tuple of every constraint
  maximize smallest preference level (over all constraints)

These approaches can be modeled using valued CSPs [Schiex, Fargier, Verfaillie, 1995] or semi-rings [Bistarelli, Montanari, Rossi, 1997]
Some drawbacks

Previous approaches typically use a functional representation or an explicit representation (tuples), and usually are limited to constraints of small arity (unary, binary, sometimes ternary)

- If constraints are represented explicitly (as in e.g., Fuzzy CSPs), domain filtering may become computationally very expensive, or not very effective
- Difficult to exploit domain structure using global constraints
- Standard CP solvers (ILOG, Comet, Eclipse, Gecode, Minion,...) are not implemented to apply the previous methods

Ideally, we would like to apply soft constraints in a CP solver just as any other constraint; using an implicit representation where possible (for example using global constraints), with efficient and effective domain filtering algorithms
Cost-based approach [Petit, Régis, and Bessiere, 2000] (see also [Baptiste et al., 1998]):

- Introduce a cost variable for each soft constraint
- This variable represents some violation measure of the constraint
- Optimize aggregation of all cost variables (e.g., take their sum, or max)
- Use upper bound on cost variable to apply cost-based filtering (with back-propagation)

In this way

- soft global constraints become hard optimization constraints
- soft CSPs become hard COPs
- the cost variables can be used in other (meta-)constraints!
  
  \[
  \text{if } (z_1 > 0) \text{ then } (z_2 = 0)
  \]
- we can apply classical constraint programming solvers

Note: In this framework, we can also encode the previous approaches [Petit et al., 2000]
From soft constraints to hard optimization constraints

Example: \( x \in [9000, 10000] \) \( (x, y \text{ integer}) \)
\( y \in [0, 20000] \)
\( x \leq y \)

Let’s make the constraint ‘\( x \leq y \)’ soft by introducing a ‘cost’ variable \( z \) representing the amount of violation, as the gap between \( x \) and \( y \).

Suppose that we impose \( z \in [0, 5] \). By looking at the bounds of \( x \) and \( y \), we can immediately deduce that \( y \in [8995, 20000] \).

Observations:

• We can exploit the semantics of this soft constraint for efficient domain filtering algorithm (looking at the values one by one would have take at least 8995 steps)
• We can perform back-propagation of the cost variable to the other variables
Softening global constraints

This approach also works for \textit{global} constraints

**General recipe** for global constraint $C(X)$, where $X = \{x_1, x_2, \ldots, x_n\}$:

1. define violation measure $\mu$: $D(x_1) \times D(x_2) \times \cdots \times D(x_n) \rightarrow \mathbb{R}_{\geq}$ such that $\mu = 0$ iff $C$ is satisfied and $\mu > 0$ otherwise
2. define cost variable $z$
3. then \textit{soften} the constraint $C$ as:
   
   $\text{soft}_C(X, z, \mu) := (\mu(X) \leq z)$

**Previous example:**

define violation measure $\mu_{\leq}(x, y) = \max\{x - y, 0\}$

$\text{soft}_{\leq}(x, y, z, \mu) := (\mu(x, y) \leq z)$, in other words, $x - y \leq z$

**Remark:** All information is now embedded in $\mu$
Basic cost-based filtering algorithm for soft\_\_C(\_X,\_z,\_\mu) = ( \_\mu(\_X) \leq \_z ) :

\begin{verbatim}
for all x\in X and d\in D(x) {
    compute minimum of \_\mu with x=d
    if minimum > max(D(z)) remove d from D(x)
    if D(x) empty return inconsistent
}
update min(D(z)) ≥ min(all minima of \_\mu)  
if D(z) empty return inconsistent  
else return consistent
\end{verbatim}

Notes:
• Similar to classical filtering for ‘optimization constraints’
• Filtering establishes domain consistency on soft\_\_C, i.e., we remove all domain values That are not part of a solution to the constraint (this is the best possible)
• Different violation measures define different soft constraints, and thus different filtering algorithms
• Time complexity depends on computation of minimum of \_\mu

Always try to improve this: Separate consistency check from filtering
Remark: We define soft constraint as
\[
\text{soft}_C(X,z,\mu) := (\mu(X) \leq z)
\]

Why not
\[
\text{soft}_C(X,z,\mu) := (\mu(X) = z)
\]

Because then domain consistency becomes often NP-hard (for ‘reasonable’ violation measure \(\mu\)), while inequality makes it often tractable

Alternative view:

• use equality in definition, but
• establish domain consistency w.r.t. \(X\) and ‘bound consistency’ w.r.t. \(z\) (effectively the same approach as with inequality)
Our **goal** is to design efficient filtering algorithms for soft global constraints, that establish domain consistency, using cost-based approach.

Methodology:
1. Choose a global constraint
2. Define useful violation measure(s)
3. Design efficient filtering algorithm

Note: We will omit the objective ‘minimize $z$’ in our examples, and let it depend on the problem at hand what to do with $z$.
Violation measures

General violation measures [Petit et al., 2001]:

- **variable-based** $\mu_{\text{var}}$
  - minimum number of variables that need to change their value in order to satisfy the constraint (minimum Hamming distance to any solution)
  - Example for alldifferent($x_1, x_2, x_3, x_4$): $\mu_{\text{var}}(1,2,2,2) = 2$

- **decomposition-based** $\mu_{\text{dec}}$
  - number of violated constraints in binary decomposition
  - e.g., $x_i \neq x_j$ for all $i < j$ is binary decomposition of alldifferent($x_1, ..., x_n$)
  - Example for alldifferent ($x_1, x_2, x_3, x_4$): $\mu_{\text{dec}}(1,2,2,2) = 3$

Other, more problem-specific violation measures [Beldiceanu and Petit, 2004]:

- **object-based**
  - for example, number of late activities in scheduling problem

- **graph-based**
  - for example, number of SCCs
Outlook

<table>
<thead>
<tr>
<th>constraint</th>
<th>violation measure</th>
<th>consistency check</th>
<th>domain consistency</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>alldifferent</td>
<td></td>
<td>$O(m\sqrt{n})$</td>
<td>$O(m)$</td>
<td>[Régisn, 1994]</td>
</tr>
<tr>
<td>soft_alldifferent</td>
<td>variable-based</td>
<td>$O(m\sqrt{n})$</td>
<td>$O(m)$</td>
<td>[Petit et al., 2001]</td>
</tr>
<tr>
<td>soft_alldifferent</td>
<td>decomposition-based</td>
<td>$O(mn)$</td>
<td>$O(m)$</td>
<td>[van Hoeve, 2004]</td>
</tr>
<tr>
<td>gcc</td>
<td></td>
<td>$O(m\sqrt{n})$</td>
<td>$O(m)$</td>
<td>[Quimper et al., 2004]</td>
</tr>
<tr>
<td>soft_gcc</td>
<td>variable-based</td>
<td>$O(m\sqrt{n})$</td>
<td>$O(m)$</td>
<td>[Zanarini et al., 2006]</td>
</tr>
<tr>
<td>soft_gcc</td>
<td>value-based</td>
<td>$O(m\sqrt{n})$</td>
<td>$O(m)$</td>
<td>[Zanarini et al., 2006]</td>
</tr>
<tr>
<td>regular</td>
<td></td>
<td>$O(m)$</td>
<td>$O(m)$</td>
<td>[Pesant, 2004]</td>
</tr>
<tr>
<td>soft_regular</td>
<td>variable-based</td>
<td>$O(m)$</td>
<td>$O(m)$</td>
<td>[van Hoeve et al., 2006a]</td>
</tr>
<tr>
<td>soft_regular</td>
<td>edit-based</td>
<td>$O(m)$</td>
<td>$O(m)$</td>
<td>[van Hoeve et al., 2006a]</td>
</tr>
</tbody>
</table>

Here $n$ is the number of variables, $m$ is proportional to the sum of the domain sizes.

Time complexity of soft versions similar to hard versions!
Filtering algorithms for the soft alldifferent constraint

1. variable-based violation measure
2. decomposition-based violation measure
**Softening alldifferent**

Example:  
\[ x_1 \in \{a,b\}, \ x_2 \in \{a,b\}, \]  
\[ x_3 \in \{a,b\}, \ x_4 \in \{b,c\}, \]  
\[ \text{alldifferent}(x_1,x_2,x_3,x_4) \]

Over-constrained CSP; we need to **soften the alldifferent** constraint:  
introduce cost variable \( z \)  
define some **violation measure**, say \( \mu \)

Result:  
\[ x_1 \in \{a,b\}, \ x_2 \in \{a,b\}, \]  
\[ x_3 \in \{a,b\}, \ x_4 \in \{b,c\}, \]  
\[ z \in \{0,1,2,3,...\} \]  
\[ \text{soft\_alldifferent}(x_1,x_2,x_3,x_4,z,\mu) \]

We will consider the variable-based and graph-based violation measures
**Violation measures**

**variable-based** violation measure $\mu_{\text{var}}$:

“minimum number of variables that need to change value in order to satisfy the constraint”

$x_1 \in \{a,b\}$, $x_2 \in \{a,b\}$,

$x_3 \in \{a,b\}$, $x_4 \in \{b,c\}$,

$\text{alldifferent}(x_1,x_2,x_3,x_4)$

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$\mu_{\text{var}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>a</td>
<td>c</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Equally bad?
**Violation measures**

**decomposition-based violation measure** $\mu_{\text{dec}}$:
“number of violated constraints in binary decomposition”

$x_1 \in \{a,b\}$, $x_2 \in \{a,b\}$,

$x_3 \in \{a,b\}$, $x_4 \in \{b,c\}$,

$\text{alldifferent}(x_1,x_2,x_3,x_4)$

$x_1 \neq x_2, x_1 \neq x_3, x_1 \neq x_4,

x_2 \neq x_3, x_2 \neq x_4, x_3 \neq x_4$

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$\mu_{\text{var}}$</th>
<th>$\mu_{\text{dec}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>a</td>
<td>c</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**distinction**
Filtering algorithm for soft alldifferent using variable-based violation measure
Filtering for variable-based violation

**Theorem** [Régin, 1994]:

solution to *hard* alldifferent ⇔ maximum matching in value graph

\[
x_1 \in \{a, b\}, \ x_2 \in \{a, b\}, \ x_3 \in \{b, c\}
\]

\[\text{alldifferent}(x_1, x_2, x_3)\]

Filtering for *hard* alldifferent: remove all edges (and corresponding domain values) that are not in any maximum matching

But what can we do if there is no solution?

\[
x_1 \in \{a, b\}, \ x_2 \in \{a, b\}, \ x_3 \in \{a, b\}, \ x_4 \in \{b, c\},
\]

\[\text{alldifferent}(x_1, x_2, x_3, x_4)\]
Filtering for variable-based violation

Fortunately, we can re-use these techniques

**Theorem** [Petit et al., 2001]: for alldifferent

\[
\min \mu_{\text{var}}(x_1, x_2, \ldots, x_n) = n - \text{value of maximum matching in value graph}
\]

**Example:**

\[
x_1 \in \{a, b\}, \ x_2 \in \{a, b\}, \ x_3 \in \{a, b\}, \ x_4 \in \{b, c\},
\]

\[\text{alldifferent}(x_1, x_2, x_3, x_4)\]

\[n - \text{value of maximum matching} = 4 - 3 = 1, \]

minimum value for \(\mu_{\text{var}}\) is 1 (see before)
Filtering rules for $x_1, x_2, \ldots, x_n$ in \texttt{soft\_alldifferent}(x_1, x_2, \ldots, x_n, z, \mu_{\text{var}}):

Let $M$ be maximum matching in value graph

- if $(n - |M|) > \max(D(z))$ then constraint is \textit{inconsistent}
- if $(n - |M|) < \max(D(z))$ then all domain values are \textit{consistent}
  (namely, if we change the value of one variable $\mu_{\text{var}}$ can increase at most one)
- if $(n - |M|) = \max(D(z))$ then domain value $d \in D(x)$ is \textit{consistent} iff edge $(x, d)$ belongs to some maximum matching in value graph

Example:

$x_1 \in \{a, b\}$, $x_2 \in \{a, b\}$,
$x_3 \in \{a, b\}$, $x_4 \in \{b, c\}$, $z \in \{0, 1\}$
\texttt{soft\_alldifferent}(x_1, x_2, x_3, x_4, z, \mu_{\text{var}})

$|M| = 3$, minimum $\mu_{\text{var}} = 4 - 3 = 1$, which is equal to $\max(D(z))$
edge $(x_4, b)$ not in maximum matching
note that also value 0 in $D(z)$ is inconsistent
Filtering for variable-based violation

To find consistent edges (same as hard alldifferent):

**Theorem** [Petersen, 1891]:

Given maximum matching $M$, any edge $e$ is in some maximum matching iff

- $e$ in $M$, or
- $e$ on even-length $M$-alternating path starting from $M$-free vertex, or
- $e$ on even-length $M$-alternating circuit

$x_1 \in \{a,b\}$, $x_2 \in \{a,b\}$,

$x_3 \in \{a,b\}$, $x_4 \in \{b,c\}$, $z \in \{0,1\}$

soft\_alldifferent($x_1,x_2,x_3,x_4,z,\mu_{\text{var}}$)

even-length $M$-alternating path starting from $M$-free vertex: $x_3 \rightarrow b \rightarrow x_2 \rightarrow a \rightarrow x_1$

even-length $M$-alternating circuit: $x_1 \rightarrow a \rightarrow x_2 \rightarrow b \rightarrow x_1$

again, edge $(x_4,b)$ not in maximum matching
Algorithm to make soft_alldifferent\((x_1, x_2, ..., x_n, z, \mu_{\text{var}})\) domain consistent

[Petit et al., 2001]:

compute maximum matching \(M\) in value graph \(\mathcal{O}(m\sqrt{n})\) [Hopcroft/Karp, 1973]

\(\text{if } (n - |M| > \max(D(z)) \text{ return inconsistent}
\)

\(\text{else if } (n - |M| < \max(D(z)) \text{ return consistent}
\)

\(\text{else } \{
\)

remove all edges (and corresponding domain values) not in any maximum size matching

\(\text{if some domain is empty return inconsistent}
\)

\(\}
\)

update \(\min(D(z)) \geq n - |M|\)

\(\text{if } D(z) \text{ is empty return inconsistent}
\)

\(\text{else return consistent}
\)

**Remarks:** Separation between constraint check and filtering; incrementality
Intermezzo: Network Flows
Network Flows

Let $G=(V,A)$ be a directed graph with vertex set $V$ and arc set $A$. To each arc $a \in A$ we assign a capacity function $[d(a),c(a)]$ and a weight function $w(a)$.

Let $s,t \in V$. A function $f: A \rightarrow \mathbb{R}$ is called an s-t flow (or a flow) if

- $f(a) \geq 0$ for all $a \in A$
- $\sum_{a \text{ enters } v} f(a) = \sum_{a \text{ leaves } v} f(a)$ for all $v \in V$ (flow conservation)
- $d(a) \leq f(a) \leq c(a)$ for all $a \in A$

Define the cost of flow $f$ as $\sum_{a \in A} w(a)f(a)$. A minimum-cost flow is a flow with minimum cost.

![Network Flow Diagram]

flow (in blue) with cost 10

special arc to ensure flow conservation
Alternative graph representation for alldifferent

**Fact:** matching in bipartite graph $\iff$ integer flow in directed bipartite graph

Step 1: direct edges from $X$ to $D(X)$
Step 2: add a source $s$ and sink $t$
Step 3: connect $s$ to $X$, and $D(X)$ to $t$
Step 4: add special arc $(t,s)$

all arcs have capacity $[0,1]$ and weight 0
except arc $(t,s)$ with capacity $[0, \min\{|X|,|D(X)|\}]$
Start from basic network:

**Example:**

\[
\begin{align*}
  x_1 &\in \{a,b\}, \\
  x_2 &\in \{a,b\}, \\
  x_3 &\in \{a,b\}, \\
  x_4 &\in \{b,c\}, \\
  z &\in \{0,1\}
\end{align*}
\]

\[\text{soft\_alldifferent}(x_1, x_2, x_3, x_4, z, \mu_{\text{dec}})\]

E.g. \(x_1 = x_2 = x_3 = x_4 = b\) should give violation cost 6

What can we do?

**Options:**

- manipulate capacities
- add/remove arcs
- add weights to some arcs
Graph representation

We can do the following [v.H., 2004]:
• for all \( d \in D(X) \) with \( k \) incoming arcs: add arcs \((d,t)_1, (d,t)_2, \ldots, (d,t)_{k-1}\), with capacity \([0,1]\), and weight \( i \) for arc \((d,t)_i\) (\( i = 1, \ldots, k-1 \))
• for all \( x \in X \): set capacity of arc \((s,x)\) to \([1,1]\), i.e. force \( x \) into solution
• minimum-cost flow \( \iff \) solution to soft_alldifferent minimizing \( \mu_{\text{dec}} \)

Example:
\[
\begin{align*}
x_1 &\in \{a,b\}, \ x_2 \in \{a,b\}, \\
x_3 &\in \{a,b\}, \ x_4 \in \{b,c\}, \ z \in \{0,1\}
\end{align*}
\]
soft_alldifferent(\(x_1,x_2,x_3,x_4,z,\mu_{\text{dec}}\))

Note: minimum-cost flow in this graph can be computed in \( O(nm) \) time
Algorithm to make soft_alldifferent($x_1, x_2, ..., x_n, z, \mu_{dec}$) domain consistent:

for all arcs ($x_i, d$) {
    set capacity of ($x_i, d$) to $[1, 1]$  // force flow to represent $x_i = d$
    compute min-cost flow $f$
    if cost($f$) > max($D(z)$) remove $d$ from $D(x_i)$
    if $D(x_i)$ is empty return inconsistent
}
update min($D(z)$) $\geq$ min(cost($f$))
if $D(z)$ is empty return inconsistent
else return consistent

Improvements: Can use flow theory to make incremental, and to separate consistency check ($O(mn)$ time), and filtering ($O(m)$ time)
Filtering algorithms for the soft global cardinality constraint

1. variable-based violation measure
2. value-based violation measure
A global cardinality constraint is defined as \( \text{gcc}(X, l, u) \)
where \( X \) is set of variables \( \{x_1, x_2, \ldots, x_n\} \)
\( l \) and \( u \) are arrays of (constant) integers, such that
domain value \( d \) is used between \( l_d \) and \( u_d \) times.

\[
x_1 \in \{0,1\}, x_2 \in \{0,1,2\}, x_3 \in \{1,2\}, x_4 \in \{1,2,3\}
\]
\[
\text{gcc}({x_1, x_2, x_3, x_4}, [0,1,0,1], [1,2,2,1])
\]
a solution: \( x_1=0, x_2=1, x_3=2, x_4=3 \)

Efficient domain consistency filtering algorithm for gcc in \( O(mn) \) time [Régin, 1996],
improved to \( O(m\sqrt{n}) \) time [Quimper et al., 2004]

Special case: alldifferent = gcc with \( l=[0,0,\ldots,0] \) and \( u=[1,1,\ldots,1] \)
Example:

\[ x_1 \in \{1,2\}, x_2 \in \{1\}, x_3 \in \{1,2\}, x_4 \in \{1\} \]
\[ \text{gcc}\{x_1,x_2,x_3,x_4\}, [1,3], [2,5] \]

Over-constrained CSP; we need to soften the gcc:
- introduce cost variable \( z \)
- define some violation measure, say \( \mu \)

Result:

\[ x_1 \in \{1,2\}, x_2 \in \{1\}, x_3 \in \{1,2\}, x_4 \in \{1\} \]
\[ z \in \{0,1,2,3,\ldots\} \]
\[ \text{soft\_gcc}\{x_1,x_2,x_3,x_4\}, [1,3], [2,5], z, \mu \]

What is a good violation measure?
Violation measures for gcc

General violation measures:
• **variable-based** $\mu_{\text{var}}$:
  – minimum number of variables that need to change their value in order to satisfy the constraint
• **decomposition-based** $\mu_{\text{dec}}$:
  – minimum number of violated constraints in binary decomposition

We can apply $\mu_{\text{var}}$, but what is binary decomposition of gcc?

**Example:**
\[
x_1 \in \{1,2\}, \ x_2 \in \{1\}, \ x_3 \in \{1,2\}, \ x_4 \in \{1\}
gcc(\{x_1,x_2,x_3,x_4\}, [1,3], [2,5])
\]

$\mu_{\text{var}}(1,1,1,1) = 3$ (e.g. change $x_1$, $x_2$ and $x_3$ to 2)
$\mu_{\text{var}}(2,1,2,1) = 1$ (e.g. change $x_2$ to 2)
Violation measures for gcc

Rewrite $\mu_{\text{var}}$ in terms of ‘shortage’ and ‘excess’ functions [v.H. et al., 2006].

Let $X = \{x_1, x_2, ..., x_n\}$ and $d \in D(X)$. Then

\[
\begin{align*}
  s(X, d) &= \begin{cases} 
    l_d - |\{x_i \mid x_i = d\}| & \text{if } |\{x_i \mid x_i = d\}| \leq l_d \\
    0 & \text{otherwise}
  \end{cases} \quad \text{‘shortage’ of value } d \\
  e(X, d) &= \begin{cases} 
    |\{x_i \mid x_i = d\}| - u_d & \text{if } |\{x_i \mid x_i = d\}| \geq u_d \\
    0 & \text{otherwise}
  \end{cases} \quad \text{‘excess’ of value } d
\end{align*}
\]

We have $\mu_{\text{var}}(X) = \max \left( \sum_d s(X, d), \sum_d e(X, d) \right)$

**Example:**

\[x_1 \in \{1,2\}, x_2 \in \{1\}, x_3 \in \{1,2\}, x_4 \in \{1\}\]

gcc($\{x_1, x_2, x_3, x_4\}$, $[1,3]$, $[2,5]$)

for $X=(1,1,1,1)$: $s(X,1)=0$, $s(X,2)=3$, $e(X,1)=2$, $e(X,2)=0$, hence $\mu_{\text{var}}(1,1,1,1) = \max(3,2)=3$
Violation measures for gcc

Example: \( x_1 \in \{1,2\}, x_2 \in \{1\}, x_3 \in \{1,2\}, x_4 \in \{1\} \)
\( gcc(\{x_1, x_2, x_3, x_4\}, [2,3], [3,5]) \)

for \( X=(1,1,1,1) \): \( s(X,1)=0, s(X,2)=3, e(X,1)=1, e(X,2)=0, \)
hence \( \mu_{\text{var}}(1,1,1,1) = \max(3,1)=3 \)
but this is wrong! we can never assign 2+3=5 values to only 4 variables!

Problem: \( \mu_{\text{var}} \) only applicable if \( \sum_d l_d \leq |X| \leq \sum_d u_d \)

Remedy: Define new ‘value-based’ violation measure

\[
\mu_{\text{val}}(X) = \sum_d (s(X, d) + e(X, d))
\]

Example (cont’d): \( \mu_{\text{val}}(1,1,1,1) = 0+3+1+0=4 \)
Filtering algorithm for soft gcc using variable-based violation measure
Filtering for variable-based violation

**Theorem** [Régin, 1996]:

solution to hard gcc ⇔ flow in particular network

**Example:**

\[ x_1 \in \{1,2\}, x_2 \in \{1\}, x_3 \in \{1,2\}, x_4 \in \{1\} \]

\[ \text{gcc}\{x_1,x_2,x_3,x_4\}, [1,3], [2,5] \]

for each arc \( a \), capacity indicated as ( \( d(a) \), \( c(a) \) )

**over-constrained gcc:**

no flow exists

what can we do?

insert weighted arcs?
Filtering for variable-based violation

We can do the following [v.H., Pesant, and Rousseau, 2006]:

- for all pairs $u,v \in D(X)$: add arcs $(u,v)$ with capacity $[0,|X|]$, and weight 1
- minimum-cost flow $\iff$ solution to soft_gcc minimizing $\mu_{\text{var}}$

Example:

$x_1 \in \{1,2\}$, $x_2 \in \{1\}$, $x_3 \in \{1,2\}$, $x_4 \in \{1\}$

$\text{soft}_\text{gcc}(\{x_1,x_2,x_3,x_4\}, [1,3], [2,5], z, \mu_{\text{var}})$

flow with cost 1
Domain consistency filtering algorithm for \texttt{soft\_gcc}(X,l,u,z,\mu_{\text{var}}) :

- Similar to network-based \texttt{soft\_alldifferent}
- First compute minimum flow to check for consistency, in \(O(n(m+n \log n))\) time
- Filter all inconsistent arcs, in \(O(n(m+n \log n))\) time (using flow theory)

This can be improved to \(O(m\sqrt{n})\) for consistency check and \(O(m)\) for filtering (will show later)
Filtering algorithm for soft gcc using value-based violation measure
Filtering for value-based violation (gcc)

Again, start from basic network

Example: \( x_1 \in \{1,2\}, x_2 \in \{1\}, x_3 \in \{1,2\}, x_4 \in \{1\} \)
\[ \text{gcc}\{x_1,x_2,x_3,x_4\}, [1,3], [2,5] \]

e.g. when all variables are assigned to 1, \( \mu_{\text{val}} = 2 + 3 = 5 \)

where to insert weighted arcs?

\[ \mu_{\text{val}}(X) = \sum_d (s(X,d) + c(X,d)) \]
Filtering for value-based violation (gcc)

We can do the following [v.H. et al., 2006]:
• for all \( d \in D(X) \): add arcs \((s,d)\) with capacity \([0,l_d]\), and weight 1
• for all \( d \in D(X) \): add arcs \((d,t)\) with capacity \([0,\max\{|X|-u_d,0\}]\), and weight 1
• minimum-cost flow \( \iff \) solution to soft_gcc minimizing \( \mu_{val} \)

flow with cost 5
Filtering for value-based violation

Domain consistency filtering algorithm for soft_gcc(X,l,u,z,µ_val) :

• Again, similar to network-based soft_alldifferent and var-based soft_gcc
• First compute minimum flow to check for consistency, in \( O((n+k)(m+n \log n)) \) time, where \( k = |D(X)| \)
• Filter all inconsistent arcs, in \( O(n(m+n \log n)) \) time (using flow theory)

Also this can be improved to \( O(m\sqrt{n}) \) for consistency check and \( O(m) \) for filtering
Zanarini, Milano and Pesant [2006] present an alternative approach that improves the running time to $O(m\sqrt{n})$. Their approach extends the algorithm of Petit et al. [2001] for the variable-based soft allDifferent:

- Apply result of Quimper et al. [2004] to compute two maximum flows in the value graph (they correspond to ‘capacitated matchings’):
  - one flow w.r.t. lower bounds $l$: $f_l$ (i.e., capacity of node $i$ is $l_i$)
  - one flow w.r.t. upper bounds $u$: $f_u$ (i.e., capacity of node $i$ is $u_i$)
- Minimum shortage is
  - $0$ if $\sum_d l_d = f_l$ i.e., all lower bounds are respected
  - $\sum_d l_d - f_l$ otherwise
- Minimum excess is
  - $0$ if $|X| = f_u$ i.e., all upper bounds are respected
  - $|X| - f_u$ otherwise
Alternative approach

• **Theorem:** minimum shortage and minimum excess can be combined to form $\mu_{val}$ and $\mu_{val}$

• **Filtering** does case analysis, and apply rules:
  - forcing $x=d$ increases $\mu_{var}$ at most 1
  - forcing $x=d$ increases $\mu_{val}$ at most 2
Filtering algorithms for the soft regular constraint

1. variable-based violation measure
2. edit-based violation measure
A **regular language** can be represented by a **deterministic finite automaton** (DFA):

automaton accepts string ⇔ string belongs to regular language

**Example:**

$q_0$ is start state, $q_3$ and $q_4$ are end states

arrow is transition from one state to another

each transition has a label

e.g. string aabbaa and ccc accepted, string caabbac not accepted

Given a DFA, the constraint \texttt{regular}(x_1,x_2,...,x_n,DFA) imposes that the 'string' $x_1x_2\cdots x_n$ is accepted by DFA \cite{Pesant2004}
Consider the problem to roster nurses in a hospital

- each nurse works at most one shift a day
- each shift contains 8 consecutive hours
  - day shift: 8am-4pm
  - evening shift: 4pm-12am
  - night shift: 12am-8am
- after a night shift, nurse needs to take one day rest
- after an evening shift, nurse may not work a day shift

Feasible roster (7 days) for a nurse: day - day - evening - night - rest - day – day

For each nurse, introduce variables $X = \{x_1, x_2, \ldots, x_7\}$ representing shift on day 1,2,...,7 with domains $D(x) = \{r,d,e,n\}$ for all $x \in X$

Model the pattern with regular($X$, DFA)
Filtering hard regular constraint

Theorem [Pesant, 2004]:

solution to regular $\iff$ path from $q_0$ to ‘end vertex’ in layered graph

$x_1 \in \{a, b, c\}$, $x_2 \in \{a, b, c\}$, $x_3 \in \{a, b, c\}$, $x_4 \in \{a, b, c\}$

$\text{regular}(x_1, x_2, x_3, x_4, \text{DFA})$

Filtering: remove all arcs whose label is not supported by domain value and vice versa (linear time in size of graph)
**Soft regular constraint**

**Example:** over-constrained CSP

\[
\begin{align*}
    x_1 &\in \{a,c\}, \ x_2 \in \{a,b,c\}, \ x_3 \in \{a,b,c\}, \ x_4 \in \{b,d\} \\
\text{regular}(x_1,x_2,x_3,x_4,\text{DFA})
\end{align*}
\]

**Soften** the regular constraint, as usual:

\[
\begin{align*}
    x_1 &\in \{a,c\}, \ x_2 \in \{a,b,c\}, \ x_3 \in \{a,b,c\}, \ x_4 \in \{b,d\}, \ z \in \{0,1,2,\ldots\} \\
\text{soft\_regular}(x_1,x_2,x_3,x_4,\text{DFA},z,\mu)
\end{align*}
\]

Again, what violation measure \(\mu\) can we apply?

Variable-based violation measure? Yes, can work: This corresponds to the smallest Hamming distance of the string to any string in the language
**Violation measures for regular**

**Example**: Consider the regular language of strings that consist of alternating pairs of $a$ and $b$, for example $aabbaabb$.

The string $abbaabbaab$ does not belong to the regular language, and the corresponding ‘variable-based violation’ would be 5 (the length of the string divided by 2).

For such cases, the Hamming distance is a bad measure, as it depends on the size of the string rather than its structure.

Alternative: **Edit distance**. Smallest number of insertions, deletions, and substitutions to change one string in another.

**Edit-based violation measure** [v.H. et al. 2006]: Minimum edit distance of given string to any string in the regular language

Example above: Edit-based violation is 2.
Filtering algorithm for soft regular constraint using variable-based violation measure
Filtering for variable-based violation

\[ x_1 \in \{a,b,c\}, \ x_2 \in \{a,b,c\}, \ x_3 \in \{a,b,c\}, \ x_4 \in \{b,d\} \]

regular\((x_1,x_2,x_3,x_4,\text{DFA})\)

Can we add weighted arcs to the unfolded graph?

- e.g., we want to use value b or d for \(x_4\)
- we must start from \(q_0\)
- we must end in end vertex \(q_3\) or \(q_4\)
- we want to capture \(\mu_{\text{var}}\):
  - allowed path must have cost of \(\mu_{\text{var}}\)

For clarity, remove clearly infeasible arcs without touching the domains

Can we insert weighted arcs now?
Filtering for variable-based violation

More formally [v.H. et al., 2006]:

• for every arc in the unfolded graph, add parallel arc with ‘void’ label and weight 1

• minimum-cost path from $q_0$ to end vertex $\iff$ solution to soft_regular minimizing $\mu_{\text{var}}$

Filtering (domain consistency) same time complexity as hard regular
Filtering algorithm for soft regular constraint using edit-based violation measure
Filtering for edit-based violation

Add following arcs with unit weight [v.H. et al. 2006]
- substitution: parallel arcs to existing ones (as before)
- insertion: arcs connecting state with itself in next layer (if no substitution yet)
- deletion: for all arcs in DFA, add arc within layer
- minimum-cost path from $q_0$ to end vertex $\Leftrightarrow$ solution to soft_regular minimizing $\mu_{edit}$

Note: Directed circuits (with positive weight) may have been formed. However, filtering (domain consistency) again same time complexity as hard regular.
Other soft global constraints
Cumulative constraint can be used to model resource constraints in scheduling and packing (see Petr Vilim’s talk yesterday)

Given set of activities, each with processing time, resource consumption, earliest start time and latest end time, assign an execution time to each activity so that a given resource does not exceed its capacity

Violation measures:

- Number of late activities [Baptiste, Le Pape, and Péridy, 1998]
  First soft global constraint, ‘heuristic’ filtering to enforce on time/late.
- Variable-based, on overcapacity of resource [Petit and Poder 2008]
  Provide implementation and compare against value-based approach.
**Other scheduling-related soft global constraints**

Soft precedence constraint [Lesaint, Mehta, O’Sullivan, Quesada, and Wilson, 2009]
- Groups together hard and soft precedence constraint for telecommunications application
- Violation based on total weight of violated soft precedence constraints.
- NP-hard, but provide filtering rules for upper and lower bound

Soft constraints for timetabling application [Cambazard, Hebrard, O’Sullivan, and Papadopoulos, 2008]
- Three problem-specific soft constraints
- Soft constraints are used to exploit good bounds for this problem class
More soft global constraints

Soft sequence constraint [Maher, Narodytska, Quimper, and Walsh, 2008]
- Given ordered set of variables, each subsequence of q consecutive variables must take at least $l$ and at most $u$ values from a special set $V$
- Violation measure sums the deviation from $l$ or $u$ for each subsequence
- Domain consistency filtering using minimum-cost network flow

Soft slide constraint [Bessiere, Hebrard, Hnich, Kiziltan, Quimper, Walsh, 2007]
- Slide is extension of sequence, special case of ‘cardinality-path’ constraint
- Edit-based and variable-based violation measures
- Reformulated as hard slide constraints using additional variables
Still more soft global constraints

Soft context-free grammar constraint [Katsirelos, Narodytska, Walsh, 2008]
• Extension of regular constraint [Sellmann 2006], [Quimper & Walsh, 2006]
• Special case of weighted context free grammar constraint
• Variable-based (Hamming distance) and edit-based violation measures

Soft all-equal constraint [Hebrard, O’Sullivan, and Razgon, 2008]
• ‘Inverse’ of decomposition-based soft alldifferent
• Extended work presented this CP [Hebrard, Marx, O’Sullivan, Razgon, 2009]

Soft open global constraints
• Open constraints have a scope that will be defined during search for a solution
• Presented this CP [Maher, 2009]
Soft global constraint for weighted CSP

Sigma-alldifferent, Sigma-Gcc, Sigma-regular [Métivier, Boizumault, Loundi, 2007, 2009]
• Model preferences among variables and constraints
• Weight associated to each variable, and each constraint (for example, to each not-equal constraint in the alldifferent)
• Domain consistency algorithm for variable-based sigma-alldifferent, using network flow
• For decomposition-based sigma-alldifferent, domain consistency is NP-hard; filtering based on relaxations is proposed
• Filtering for decomposition-based sigma-gcc and a distance-based sigma-regular, similar to the network flow approach of [v.H. et al., 2006]

Weighted CSPs [Lee and Leung, 2009]
• Costs are associated to tuples of the constraint
• Apply flow-based filtering algorithms of [v.H. et al., 2006] in this context
**Constraint-Based Local Search**

**Aim:** Model the problem using variables and constraints (as in CP), and apply an automatically-derived Local Search method to solve the model


Essential to CBLS is that the solution method can be derived from the constraints

- Local Search evaluates current assignment and then moves to an (improving) assignment in its neighborhood
- Neighborhoods as well as evaluation functions can be based on combinatorial properties of the constraints
- Global constraints can be particularly useful for this purpose
  [Nareyek, 2001]
Soft global constraints for CBLS [Van Hentenryck and Michel 2005]

- Instead of domain filtering, the task is to measure the additional amount of violation (gradient) if we were to assign a variable to a certain value
- Violation measures are given for all-different, atmost, atleast, multi-knapsack, sequence, systems of not-equal constraints, and weighted constraint systems
Conclusions and perspectives
Conclusions

• Since the introduction of the first soft global constraint in 1998, and the cost-based framework for soft global constraints, many soft global constraints have been introduced.
• For many of these constraints, domain filtering is as efficient for the soft version as for its hard counterpart.
• It has become an ‘established’ approach
  – for many global constraints, a soft version is presented at the same time as the hard version (e.g., sequence, slide, weighted-grammar, etc.)
  – soft global constraints are being implemented and designed specifically for applications
  – it influences other approaches (weighted CSPs, Local Search)
Is the topic closed?

• No! It is as essential to CP as global constraints are in general
• Number of new soft global constraints and new applications grows rapidly

Many (most) industrial problems are essentially over-constrained

• Soft global constraints allow to model and solve these problems using CP
• Huge potential

Task for (industrial) CP solvers to adopt soft global constraints?

• Present in Comet 2.0, but what about IBM ILOG, Eclipse, Gecode?