Global Constraints in Constraint Programming

Willem-Jan van Hoeve

Tepper School of Business, Carnegie Mellon University

Pittsburgh, PA

Optimization Days 2010
Outline

• Constraint Programming
  – Central concepts, motivation, applications
  – Domain filtering algorithms

• Global Constraints
  – Classical (alldifferent)
  – Over-Constrained Problems (soft-alldifferent)
  – Sequencing and Scheduling (sequence)

• Recent Developments
  – Open constraints (open-alldifferent)
  – Constraint-based (local) search

(With apologies for the bias towards my own work...)
Constraint Programming
## Comparison with Integer Programming

<table>
<thead>
<tr>
<th><strong>Integer Linear Programming</strong> (branch-and-bound/branch-and-cut)</th>
<th><strong>Constraint Programming</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• systematic search</td>
<td>• systematic search</td>
</tr>
<tr>
<td>• at each search state, solve continuous relaxation of problem (expensive)</td>
<td>• at each search state, reason on individual constraints (cheap)</td>
</tr>
<tr>
<td>• add cuts to reduce search space</td>
<td>• filter variable domains to reduce search space</td>
</tr>
<tr>
<td>• domains are intervals</td>
<td>• domains may contain holes</td>
</tr>
</tbody>
</table>

- very suitable for optimization problems
- very suitable for highly combinatorial problems, e.g., scheduling, timetabling
Constraint Programming History

1970s: Artificial Intelligence
- image processing applications
- search + qualitative inference

1980s: Logic Programming
- logic programming languages (e.g., Prolog)
- search + logical inference

1989: CHIP system (Constraint Handling In Prolog)
- constraint logic programming

1990s: Constraint Programming
- combines artificial intelligence, logic programming, and operations research
- industrial solvers (e.g., ILOG, Eclipse, Xpress-Kalis) and industrial applications

1994: filtering for alldifferent and resource scheduling (edge finding)

2000s: Various developments
- efficient algorithms for special constraints
- integrated methods (with OR techniques)
- modeling languages (e.g., OPL, Comet, Zinc)
Successful applications
An 8 Team Round Robin Timetable

<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>

Schedule of 1997/1998 ACC basketball league (9 teams)

- various complicated side constraints
- all 179 solutions were found in 24h using enumeration and integer linear programming [Nemhauser & Trick, 1998]
- all 179 solutions were found in less than a minute using constraint programming [Henz, 1999, 2001]
• Gate allocation at the new (1998) Hong Kong airport
• System was implemented in only four months, including constraint programming technology (ILOG)
• Schedules ~800 flights a day
  (47 million passengers in 2007)

Port of Singapore

- One of the world’s largest container transshipment hubs
- Links shippers to a network of 200 shipping lines with connections to 600 ports in 123 countries
- Problem: Assign yard locations and loading plans under various operational and safety requirements
- Solution: Yard planning system, based on constraint programming
Railroad Optimization

- Netherlands Railways has among the densest rail networks in the world, with 5,500 trains per day.
- Constraint programming is one of the components in their railway planning software, which was used to design a new timetable from scratch (2009).
- Much more robust and effective schedule, and $75M additional annual profit.
- INFORMS Edelman Award winner (2009).
A Constraint Satisfaction Problem, or CSP, consists of
- a set of variables $X$,
- variable domains $D(x)$ (for all $x \in X$),
- and a set of constraints on subsets of the variables

A solution to a CSP is:
assign to each variable a single element from its domain
such that all constraints are satisfied

Example:
variables $x_1, x_2, x_3$
domains $D(x_1) = \{1,2\}$, $D(x_2) = \{0,1,2,3\}$, $D(x_3) = \{2,3\}$
constraints $x_1 > x_2$
$x_1 + x_2 = x_3$
$\text{alldifferent}(x_1,x_2,x_3)$
solution: $x_1 = 2$, $x_2 = 1$, $x_3 = 3$
A Constraint Optimization Problem, or COP, consists of

• a set of variables \( X \),
• variable domains \( D(x) \) (for all \( x \in X \)),
• a set of constraints on subsets of the variables,
• and an objective function \( f(X) \rightarrow \mathbb{R} \) to be optimized

A solution to a COP is:

assign to each variable a single element from its domain
such that all constraints are satisfied, and the objective function is a global optimum

Example:

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

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

objective function maximize \( x_2 + x_3 \)

solution: \( x_1 = 2, x_2 = 1, x_3 = 3 \)
More modeling examples

• variables range over finite or continuous domain:
  \( v \in \{a,b,c,d\}, \ start \in \{0,1,2,3,4,5\}, \ z \in [2.18, 4.33], \ S \in [ \{b,c\}, \{a,b,c,d,e\} ] \)

• algebraic expressions:
  \( x^3(y^2 - z) \geq 25 + x^2 \cdot \max(x,y,z) \)

• variables as subscripts:
  \( y = \text{cost}[x] \)  (here \( y \) and \( x \) are variables, ‘cost’ is an array of parameters)

• logical relations in which constraints can be mixed:
  \( ((x < y) \ \text{OR} \ (y < z)) \ \Rightarrow \ (c = \min(x,y)) \)

• ‘global’ constraints (a.k.a. symbolic constraints):
  \( \text{alldifferent}(x_1,x_2, ...,x_n) \)
  \( \text{UnaryResource}( \ [\text{start}_1,..., \text{start}_n], \ [\text{duration}_1,...,\text{duration}_n] \ ) \)
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

\[ x_1 > x_2 \]
\[ x_1 + x_2 = x_3 \]
\[ \text{alldifferent}(x_1,x_2,x_3) \]
The solution process of CP interleaves

- **domain filtering**
  - remove inconsistent values from the domains of the variables, based on individual constraints

- **constraint propagation**
  - propagate the filtered domains through the constraints, by re-evaluating them until there are no more changes in the domains

- **search**
  - implicitly all possible variable-value combinations are enumerated, but the search tree is kept small due to the domain filtering and constraint propagation

Because all variable-value combinations are (implicitly) enumerated, this solution method is complete
Domain Filtering Algorithms
Example:

\[ \text{alldifferent}(x_1, x_2, \ldots, x_n) \text{ semantically equivalent to } \{ x_i \neq x_j \text{ for all } i \neq j \} \]

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

\[ \rightarrow \text{no filtering for individual not-equal constraints} \]

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

\[ \rightarrow \text{global view of alldifferent: no solution} \]

Observation: conjunction of constraints allows more filtering!
More filtering: just 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 (e.g., alldifferent)

(Alternative: keep NP-hard subproblem, but don’t require to filter all inconsistent values)
**Overview of global constraints**

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Structure/technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>alldifferent</td>
<td>bipartite matching [Régis, 1994]</td>
</tr>
<tr>
<td>symmetric-alldifferent</td>
<td>general matching [Régis, 1999]</td>
</tr>
<tr>
<td>open-alldifferent</td>
<td>network flow [v.H. &amp; Régis, 2006]</td>
</tr>
<tr>
<td>cardinality</td>
<td>network flow [Régis, 1999, 2002]</td>
</tr>
<tr>
<td>open-cardinality</td>
<td>network flow [v.H. &amp; Régis, 2006]</td>
</tr>
<tr>
<td>knapsack/sum</td>
<td>dynamic programming [Trick, 2003]</td>
</tr>
<tr>
<td>regular</td>
<td>directed acyclic graph [Pesant, 2004]</td>
</tr>
<tr>
<td>soft-regular</td>
<td>shortest paths [v.H., Pesant &amp; Rousseau, 2006]</td>
</tr>
<tr>
<td>circuit</td>
<td>network flow [Genc Kaya &amp; Hooker, 2006]</td>
</tr>
<tr>
<td>disjunctive/cumulative</td>
<td>dedicated algorithm [Carlier &amp; Pinson, 1994] [Vilim, 2009]</td>
</tr>
<tr>
<td>inter-distance</td>
<td>dedicated algorithm [Quimper, Lopez-Ortiz &amp; Pesant, 2006]</td>
</tr>
<tr>
<td>. . .</td>
<td>. . .</td>
</tr>
</tbody>
</table>

...
Filtering algorithm for *alldifferent*

Filtering for \textit{alldifferent}

\textbf{Observation} [Régis, 1994]:

solution to \textit{alldifferent} \iff matching in bipartite graph covering all variables

\textbf{Example}:

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

\textit{alldifferent}(x_1,x_2,x_3)

\textbf{Filtering}: remove all edges (and corresponding domain values) that are not in any matching covering the variables

\textbf{Find initial matching}: \(O(m\sqrt{n})\) time\(^1\) [Hopcroft and Karp, 1973]

\textbf{How to filter all inconsistent edges?}

\(^1\) for \(n\) variables and \(m\) edges
Filtering Edges

• Naive approach is to fix each edge and test for consistency
  – Time complexity too high: $O(m^2 \sqrt{n})$
• Instead, the following can be done
  – compute one maximum matching $M$: is it covering all variables $X$?
  – orient the edges in $M$ ‘forward’, and edges not in $M$ ‘backward’
  – compute the strongly connected components (SCCs)
  – edges in $M$, and edges on even $M$-alternating path are consistent
    (i.e., edges within SCC and edges on path starting from $M$-free vertex)
  – all other edges are not consistent and can be removed

![Graph example](image)

Filtering in $O(m)$ time
Important aspects

• Separation of consistency check ( \( O(m\sqrt{n}) \) ) and domain filtering ( \( O(m) \) )

• Incremental algorithm
  – When \( k \) domain values have been removed, we can repair the matching in \( O(km) \) time

Note that these algorithms are typically invoked many times during constraint propagation

• We can apply/embed efficient algorithms from graph theory, computer science, and operations research in global constraints
Soft Global 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
Cost-based approach [Petit, Régin, 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!
  
  if \( z_1 > 0 \) then \( z_2 = 0 \)

- we can apply classical constraint programming solvers
- we can apply (cost-based) domain filtering algorithms!
**Soft-alldifferent**

*Example:* \( x_1 \in \{1,2\}, x_2 \in \{1,2\}, x_3 \in \{1,2\} \)

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

\[ x_1 \in \{1,2\}, x_2 \in \{1,2\}, x_3 \in \{1,2\}, z \in \{0,1,2,3\} \]

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

**minimize** \( z \)

Let \( z \) represent the total number of violated not-equal constraints

Solution: \( x_1=1, x_2=2, x_3=1, z=1 \) with only \( x_1 \neq x_3 \) violated

**Filter** *soft-alldifferent*:

remove domain values for which minimum violation > \( \text{max}(z) \)

**Note:** Typically we have many more constraints in our model
Filtering algorithm for soft-alldifferent


**Filtering soft-alldifferent**

**Observation:** solution to *soft-alldifferent* with minimum violation
\[ \iff \] integer minimum-cost flow

**Example:**
\[
x_1 \in \{a,b\}, \ x_2 \in \{a,b\}, \\
x_3 \in \{a,b\}, \ x_4 \in \{b,c\}, \ z \in \{0,1\}
\]
\[
\text{soft-alldifferent}(x_1,x_2,x_3,x_4,z) \\
\text{minimize } z
\]

**Filtering:** remove all edges (and corresponding domain values) that are not in any flow \( f \) with \( \text{cost}(f) \leq \max(z) \)
Filtering soft-alldifferent

- Naive approach: fix each edge and test for consistency by computing a minimum-cost network flow
  - Time complexity: $O(m^2n)$

Drawbacks:
- no separation between consistency check and filtering
- time complexity too high
- algorithm not incremental: start from scratch every time
Improved Algorithm

To improve algorithm: use residual graph $G_f$

for all arcs $a$
  - if $f(a) = 1$: reverse $a$ and weight($a$)
  - if $f(a) = 0$: leave unchanged
Theorem (e.g., Ahuja et al. 1993)

\[ f \text{ minimum-cost flow in } G \]

\[ P \text{ shortest } d-x_i \text{ path in } G_f \]

\[ \Leftrightarrow \]

minimum-cost flow \( f' \) in \( G \) with \( f'(x_i,d) = 1 \) has

cost(\( f' \)) = cost(\( f \)) + cost(\( P \))
Improved Algorithm

compute minimum-cost flow $f$ in $G$

if $\text{cost}(f) > \max(D(z))$ return inconsistent

for all arcs $(x_i,d)$ {

    compute minimum-cost $d$-$x_i$ path $P$ in $G_f$

    if $\text{cost}(f) + \text{cost}(P) > \max(D(z))$ remove $d$ from $D(x_i)$

    if $D(x_i)$ is empty return inconsistent

}

update $\min(D(z)) \geq \min(\text{cost}(f))$

if $D(z)$ is empty return inconsistent
else return consistent
how to compute minimum-cost $d-x_i$ path $P$ in $G_f$: 

if $f(x_i,d)=1$ then $P = d,x_i$ and cost$(P) = 0$

else if $x_i$ and $d$ in same SCC\(^1\) then cost$(P) = 0$

else $P$ must visit $t$ (once) and

\[
\text{cost}(P) = \text{min-cost SCC}(d)-t \text{ path} + \text{min-cost t-SCC}(x_i) \text{ path}
\]

\(^1\) strongly connected component in $G_f - \{s,t\}$. 
Improved algorithm in detail

compute minimum-cost flow \( f \) in \( G \) \( \quad \text{O}(mn) \)

\textbf{if} \( \text{cost}(f) > \max(z) \) \textbf{return} inconsistent

compute SCCs in \( G_f\{-s,t\} \) \( \quad \text{O}(m) \)

compute minimum-cost paths from all nodes to \( t \) and reverse

\textbf{for all arcs (}x_i, d\text{)} \{  
  compute minimum-cost \( d-x_i \) path \( P \) in \( G_f \) \( \quad \text{O}(1) \)
  \textbf{if} \( \text{cost}(f) + \text{cost}(P) > \max(z) \) remove \( d \) from \( D(x_i) \)
  \textbf{if} \( D(x_i) \) is empty \textbf{return} inconsistent

\}  

data update \( \min(D(z)) \geq \min(\text{cost}(f)) \)
\textbf{if} \( D(z) \) is empty \textbf{return} inconsistent
\textbf{else} \textbf{return} consistent

- Consistency check: \( \text{O}(mn) \)
- Filtering all inconsistent values: \( \text{O}(m) \)
- Incremental: after \( k \) changes initial flow can be repaired in \( \text{O}(km) \) time
Many other soft global constraints

- Soft cumulative constraint [Baptiste et al., 1998], [Petit and Poder 2008]
- Soft global cardinality constraint [v.H. et al., 2006] [Zanarini et al. 2006, 2010]
- Soft regular constraint, soft same constraint [v.H. et al., 2006]
- Soft slide constraint [Bessiere et al., 2007]
- Sigma-alldifferent, Sigma-Gcc, Sigma-regular [Métivier et al., 2007, 2009]
- Soft sequence constraint [Maher et al., 2008]
- Soft context-free grammar constraint [Katsirelos et al., 2008]
- Soft constraints for timetabling application [Cambazard et al., 2008]
- Soft all-equal constraint [Hebrard et al., 2008], [Hebrard et al., 2009]
- Soft precedence constraint [Lesaint et al., 2009]
- Soft open global constraints [Maher, 2009]
- Soft global constraints for Weighted CSPs [Lee and Leung, 2009]

Filtering algorithm for sequence


Nurse rostering

• find feasible working pattern for each employee
• restrictions:
  – every calendar-week 4 or 5 working days
  – every 9 consecutive days at most 7 working days
  – every 30 consecutive days at least 20 working days

<table>
<thead>
<tr>
<th>week</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun</td>
<td>1</td>
<td>8</td>
<td>15</td>
<td>22</td>
<td>29</td>
</tr>
<tr>
<td>Mon</td>
<td>2</td>
<td>9</td>
<td>16</td>
<td>23</td>
<td>30</td>
</tr>
<tr>
<td>Tue</td>
<td>3</td>
<td>10</td>
<td>17</td>
<td>24</td>
<td>1</td>
</tr>
<tr>
<td>Wed</td>
<td>4</td>
<td>11</td>
<td>18</td>
<td>25</td>
<td>2</td>
</tr>
<tr>
<td>Thu</td>
<td>5</td>
<td>12</td>
<td>19</td>
<td>26</td>
<td>3</td>
</tr>
<tr>
<td>Fri</td>
<td>6</td>
<td>13</td>
<td>20</td>
<td>27</td>
<td>4</td>
</tr>
<tr>
<td>Sat</td>
<td>7</td>
<td>14</td>
<td>21</td>
<td>28</td>
<td>5</td>
</tr>
</tbody>
</table>

• additional constraints
  – demand, union requirements, night shift restrictions, etcetera
**Sequence constraint**

**Example:** every 9 consecutive days at most 7 working days

variable $x_i \in \{0, 1\}$ for each day $i$

<table>
<thead>
<tr>
<th></th>
<th>sun</th>
<th>mon</th>
<th>tue</th>
<th>wed</th>
<th>thu</th>
<th>fri</th>
<th>sat</th>
<th>sun</th>
<th>mon</th>
<th>tue</th>
<th>wed</th>
<th>thu</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>$x_2$</td>
<td>$x_3$</td>
<td>$x_4$</td>
<td>$x_5$</td>
<td>$x_6$</td>
<td>$x_7$</td>
<td>$x_8$</td>
<td>$x_9$</td>
<td>$x_{10}$</td>
<td>$x_{11}$</td>
<td>$x_{12}$</td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{align*}
0 \leq x_1 + x_2 + \ldots + x_9 & \leq 7 \\
0 \leq x_2 + x_3 + \ldots + x_{10} & \leq 7 \\
0 \leq x_3 + x_4 + \ldots + x_{11} & \leq 7 \\
0 \leq x_4 + x_5 + \ldots + x_{12} & \leq 7 \\
\end{align*}
\]

\[
\text{sequence}(x_1, x_2, \ldots, x_{12}, q=9, \text{min}=0, \text{max}=7)
\]

**sequence**($x_1, x_2, \ldots, x_n, q, \text{min}, \text{max}$):

the sum of every $q$ consecutive variables is between $\text{min}$ and $\text{max}$

A sequence constraint **groups together** the individual constraints
Sequence constraint

sequence is more powerful than individual constraints filtered separately

Example: sequence($x_1, x_2, \ldots, x_7$, $q=5$, $min=2$, $max=3$)

$x_1 = 1$, $x_2 = 1$, $x_6 = 0$

\[
\begin{array}{ccccccc}
  x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \\
  1 & 1 & 0/1 & 0/1 & 0/1 & 0 & 0/1 \\
\end{array}
\]

\[
2 \leq x_1 + x_2 + x_3 + x_4 + x_5 \leq 3 \\
2 \leq x_2 + x_3 + x_4 + x_5 + x_6 \leq 3 \\
2 \leq x_3 + x_4 + x_5 + x_6 + x_7 \leq 3
\]

= 1
**Sequence constraint**

**History:**

1988: car sequencing (Dincbas, Simonis & Van Hentenryck, 1988)

1994: sequence introduced (Beldiceanu & Contejean, 1994) as conjunction of overlapping cardinality constraints

1997: filtering algorithm (Régin & Puget, 1997) tailored to car sequencing, no complete filtering

2001: filtering algorithm (Beldiceanu & Carlsson, 2001) instance of generic class of *cardinality-path* constraints, no complete filtering

**Goal:** efficient (polynomial-time) complete filtering for sequence
Accumulative solutions

Accumulate variables: $y[i] = x_1 + x_2 + \ldots + x_i$

Example: sequence($x_1, x_2, x_4, x_5, x_6$, $q=3$, $min=1$, $max=2$)

$1 \leq y[3] - y[0] \leq 2$
$1 \leq y[4] - y[1] \leq 2$
$1 \leq y[5] - y[2] \leq 2$
$1 \leq y[6] - y[3] \leq 2$

$y_{blue} = 0 \ 1 \ 1 \ 2 \ 2 \ 2 \ 3$
Accumulative solutions

Accumulate variables: \( y[i] = x_1 + x_2 + ... + x_i \)

Example: sequence\((x_1, x_2, x_3, x_4, x_5, x_6, q=3, \text{ min}=1, \text{ max}=2)\)

\[ y_\text{blue} = 0 \ 1 \ 1 \ 2 \ 2 \ 2 \ 3 \]
\[ y_\text{red} = 0 \ 0 \ 1 \ 1 \ 2 \ 3 \ 3 \]

Observation: for any two accumulate solutions, their pointwise minimum and maximum are also solutions

This is \textbf{not} true for binary \( x \) representation!

\[ x_\text{blue} = 1 \ 0 \ 1 \ 0 \ 0 \ 1 \]
\[ x_\text{red} = 0 \ 1 \ 0 \ 1 \ 1 \ 0 \]
Accumulative solutions

Accumulate variables: \( y[i] = x_1 + x_2 + ... + x_i \)

Example: sequence\((x_1, x_2, x_3, x_4, x_5, x_6, q=3, \text{min}=1, \text{max}=2)\)

Observation: for any two accumulate solutions, their pointwise minimum and maximum are also solutions

Corollary: absolute minimum and maximum solutions envelope all solutions
Find minimum solution

Algorithm:

initialize y
while some subsequence violated
  push-up endpoint minimally
  repair on left and right (using push-ups)

invariant: \( y[i+1] - y[i] \) is 0 or 1

Example:
sequence(\( x_1, x_2, \ldots, x_6 \), \( q=3, \min=2, \max=2 \))
\( D(x_i) = \{0,1\} \) for all \( i \neq 5 \)
\( D(x_5) = \{1\} \)

2 \leq y[3] - y[0] \leq 2
Find minimum solution

Algorithm:
initialize \( y \)
while some subsequence violated
  push-up endpoint minimally
  repair on left and right (using push-ups)

\textbf{Invariant:} \( y[i+1] - y[i] \) is 0 or 1

\textbf{Example:}
sequence(\( x_1, x_2, ..., x_6 \); \( q = 3 \); \( \min = 2 \); \( \max = 2 \))
\( D(x_i) = \{0, 1\} \) for all \( i \neq 5 \)
\( D(x_5) = \{1\} \)
Find minimum solution

Algorithm:
initialize $y$
while some subsequence violated
  push-up endpoint minimally
  repair on left and right (using push-ups)

invariant: $y[i+1] - y[i]$ is 0 or 1

Example:
sequence($x_1, x_2, ..., x_6$, $q=3$, min=2, max=2)
$D(x_i) = \{0, 1\}$ for all $i \neq 5$
$D(x_5) = \{1\}$
Find minimum solution

**Properties:**

- repair keeps $y[i] \leq y_{\text{min}}[i]$ for all $i$ (by induction)
  hence, if minimum solution exists, algorithm finds it
  otherwise, $y[i] > i$ leads to unsatisfiability

- total number of push-ups bounded by $n^2$
  algorithm runs in $O(n^2)$ time
Filtering algorithm

Basic algorithm:
for every domain value:
    compute minimum solution (using this value)
    if no solution, remove value

Filtering: we remove all inconsistent values

Time complexity: $O(n^3)$

Improvements:
- maintain supports for domain values
  - each solution provides support for $n$ values
- for each value, restart from $y_{\text{min}}$
- also compute maximum solution $y_{\text{max}}$
  - detect violation if $y[i] > y_{\text{max}}[i]$
- maintain $y_{\text{min}}$ and $y_{\text{max}}$ during search (both are monotone)
  amortize complexity: $O(n^3)$ on any path from root to a leaf
• filtering algorithm also applies to \textit{generalized sequence}
  \(q, \min, \text{and} \ max\) vary per subsequence

\textbf{Example}: nurse rostering problem
– every calendar-week 4 or 5 working days
– every 9 consecutive days at most 7 working days
– every 30 consecutive days at least 20 working days

• \textit{sequence} on non-consecutive subsequences is NP-hard
  [Régin, 2005]
## Single sequence constraint

<table>
<thead>
<tr>
<th>$n = 100$</th>
<th>ILOG Basic (max – min)</th>
<th>ILOG Extended (max – min)</th>
<th>our algorithm (max – min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q$</td>
<td>back-&lt;br&gt;tracks</td>
<td>CPU</td>
<td>back-&lt;br&gt;tracks</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>limit</td>
<td>limit</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>362K</td>
<td>54</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>381K</td>
<td>55</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>265K</td>
<td>54</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>287K</td>
<td>48</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>limit</td>
<td>limit</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>195K</td>
<td>43</td>
</tr>
</tbody>
</table>
**Single sequence constraint**

<table>
<thead>
<tr>
<th>max – min = 1</th>
<th>ILOG Basic</th>
<th>ILOG Extended</th>
<th>our algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>q</td>
<td>n</td>
<td>back-tracks</td>
<td>CPU</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>459K</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>192K</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>500</td>
<td>48K</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>1000</td>
<td>1K</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>50</td>
<td>210K</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>100</td>
<td>221K</td>
<td>18</td>
</tr>
<tr>
<td>7</td>
<td>500</td>
<td>80K</td>
<td>21</td>
</tr>
<tr>
<td>7</td>
<td>1000</td>
<td>30K</td>
<td>28</td>
</tr>
<tr>
<td>9</td>
<td>50</td>
<td>18K</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>100</td>
<td>3K</td>
<td>0.3</td>
</tr>
<tr>
<td>9</td>
<td>500</td>
<td>49K</td>
<td>18</td>
</tr>
<tr>
<td>9</td>
<td>1000</td>
<td>17K</td>
<td>20</td>
</tr>
</tbody>
</table>
Generalized sequence constraint

Instances:
• inspired by nurse rostering problems
• two sequence constraints
• find all solutions

<table>
<thead>
<tr>
<th>instance type</th>
<th>horizon</th>
<th>#solutions</th>
<th>our individual sequence constraints</th>
<th>our generalized sequence constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>backtracks</td>
<td>time</td>
</tr>
<tr>
<td>max6/8-min22/30</td>
<td>40</td>
<td>2248</td>
<td>185k</td>
<td>4 min</td>
</tr>
<tr>
<td>max6/8-min22/30</td>
<td>80</td>
<td>730</td>
<td>198k</td>
<td>18 min</td>
</tr>
<tr>
<td>max6/9-min20/30</td>
<td>40</td>
<td>3</td>
<td>394k</td>
<td>7 min</td>
</tr>
<tr>
<td>max6/9-min20/30</td>
<td>80</td>
<td>3</td>
<td>394k</td>
<td>30 min</td>
</tr>
<tr>
<td>max7/9-min22/30</td>
<td>40</td>
<td>138k</td>
<td>328k</td>
<td>7 min</td>
</tr>
<tr>
<td>max7/9-min22/30</td>
<td>80</td>
<td>23k</td>
<td>1847k</td>
<td>2 hours</td>
</tr>
</tbody>
</table>
Further Improvements

• Brand et al. [2007] have shown that our algorithm can be interpreted as a ‘Singleton Bounds Consistency’ algorithm on the cumulative decomposition:

\[
y_{i+1} = y_i + x_i \\
y_{i+q} - y_i \geq l \\
y_{i+q} - y_i \leq u
\]

This decomposition has the same filtering power and the same complexity, but runs faster in practice.

• Using a different decomposition Brand et al. show that complete filtering can be done in \(O(n^2 \log n)\) time.

• Maher et al. [2008] present an \(O(n^2)\) algorithm, by representing the problem as an integer program and then converting it into a network flow.
Other Recent Developments
Open constraints

Traditional CSPs:
- all variables and constraints are fixed from the beginning
- “closed-world scenarios”

Open CSPs:
- variables and constraints are revealed over time

Example:
- process set of activities over different factory lines: each factory line has predefined set of constraints, but paths of the activities are unknown initially

Open constraints: defined on a-priori unknown set of variables

Open constraints in a closed world: all potential variables and domains are known

[v.H. and Regin, 2006]
Example

variables:
- \( \text{start}(j_1), \ldots, \text{start}(j_n) \)

constraints:
- each task must be processed on one production line
- on each production line, the start times of the tasks are different:
  - open-alldifferent\((S_i)\) for production line \(i=1,2,3\)

where \(S_i\) is a set-variable representing the start time variables of the jobs on line \(i\)
Open constraints

Traditional CSP:

\[ \text{alldifferent}(x_1, x_2, \ldots, x_n) \]
\[ x_i \in D(x_i) \text{ for } i=1,\ldots,n \]

Open CSP (in a closed world):

\[ \text{alldifferent}(S) \]
\[ S \in [\emptyset, \{x_1, x_2, \ldots, x_n\}] \quad (S \text{ is a set variable}) \]
\[ x_i \in D(x_i) \text{ for } i=1,\ldots,n \]

Goals: given an open constraint, we want to filter all inconsistent values from

- \( D(x_i) \) for all \( i \),
- and \( D(S) \)
  - add mandatory elements to lower bound,
  - remove impossible elements from upper bound
  - compute tight lower and upper bound on cardinality of \( S \)

Efficient filtering algorithms for open \textit{alldifferent} (and gcc), and \textit{conjunctions} of them, can be designed using specific network flow representation [v.H. and Regin, 2006]
• The combinatorial structure embedded by global constraints can also be used for other purposes than only filtering, for example to guide the search

• Examples
  – Constraint Based Local Search
  – Counting Based Search
**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 \textit{alldifferent}, \textit{atmost}, \textit{atleast}, \textit{multi-knapsack}, \textit{sequence}, systems of not-equal constraints, and weighted constraint systems
Counting-Based Search

- **Aim**: Guide the search to ‘promising’ search space containing many solutions
- Branching decision defined by selecting a variable-value pair
- So we need to associate to each variable-value pair a measure indicating to how many solutions it belongs
- Counting number of solution is #P-complete in general
- However, we can efficiently find approximations for individual *global constraints* and then aggregate the results [Pesant 2005], e.g.,
  - *alldifferent* and *regular* constraints [Zanarini and Pesant, 2007, 2009]
  - *knapsack* constraints [Pesant and Quimper, 2008]
Summary

• Global constraints are driving force of successful application of constraint programming
• This talk: efficient domain filtering algorithms for
  – *alldifferent* (matchings)
  – *soft-alldifferent* (minimum-cost network flow)
  – *sequence* (dedicated algorithm)

They provide of flavor of what can be done with global constraints

• Many more research opportunities