Constraint Technology for Solving Combinatorial Problems: Overview and Applications

Pierre Flener

Computing Science Division
Department of Information Technology
Uppsala University, Sweden

Acknowledgments: Co-authors & sponsors (STINT, VR, EuroControl, ...)

1. Constraint Technology in a Nutshell

Example: $A + B < C$, where $A, B, C \in 1…4$.

Definition: A constraint is a logical relationship between unknowns, called (decision) variables, each of which has a set of possible values, called its domain.

Example: Colour the countries of a map such that no two neighbour countries have the same colour.

Definition: A constraint satisfaction problem (CSP) is about labelling its decision variables with values from their domains, such that its set of constraints on these decision variables is satisfied.
**Example:** Find the smallest number of colours that solve a given map colouring problem.

**Definition:** A *constraint optimisation problem (COP)* is a CSP plus a cost expression on its decision variables, whose value has to be minimised (or maximised).

**Constraint Technology** offers:

- A programming *language* for modelling CSPs and COPs.
- A programming *language* for (globally) *searching* for their solutions.
- A set of *solvers* for pruning the domains of the decision variables.

The focus is here on finite, discrete domains.
Constraint Programming

Example: The following is a constraint program:

\[ A, B, C \in 1\ldots4 \]
\[ A + B < C \]
\[ \text{labelling}([A, B, C]) \]

Definition: A constraint program usually consists of, in this sequence:

1. **Domain declarations** for the decision variables.
   What are the variables and values of my problem?
2. **Posted constraints** on these variables.
   What is the best way of formulating the constraints of my problem?
3. A **search procedure**. (There is a default search procedure.)
   What is the best way of searching for solutions to my problem?

The first two parts are declarative and together form the constraint model.
The search procedure part is necessarily non-declarative.
Example: The constraint program

\[ A, B, C \in 1\ldots4 \]
\[ A + B < C \]

*labelling*([A, B, C])

executes as follows:

1. After the unique domain declaration, the domains trivially are: \( A, B, C \in 1\ldots4 \).
2. After posting the unique constraint, the domains have become: \( A, B \in 1\ldots2 \) and \( C \in 3\ldots4 \).
3. Labelling searches and finds the following 4 solutions:
   \[ [1,1,3], [1,1,4], [1,2,4], [2,1,4] \].
**Propagation**

\[ A + B < C, \text{ where } A, B, C \in 1 \ldots 4 \]

*Operationally*, posting a constraint invokes a co-routine for:

- Testing if a constraint is *definitely true*: if so, then *deactivate* it!

  **Example**: The maximum 8 of \( A + B \) is not smaller than the minimum 1 of \( C \), so the constraint \( A + B < C \) is *not* definitely true.

- Testing if a constraint is *definitely false*: if so, then *backtrack*!

  **Example**: The minimum 2 of \( A + B \) is not larger than the maximum 4 of \( C \), so the constraint \( A + B < C \) is *not* definitely false.
• *Pruning* values that make the constraint false: if it is then neither definitely true nor definitely false, then *suspend* it!

One may have to *search* later on!

\[ A + B < C, \quad \text{where } A, B, C \in 1\ldots4 \]

**Example:** \[ \max(A) = \max(C) - \min(B) - 1 = 4 - 1 - 1 = 2 \]

**Example:** \[ \max(B) = \max(C) - \min(A) - 1 = 4 - 1 - 1 = 2 \]

**Example:** \[ \min(C) = \min(A) + \min(B) + 1 = 1 + 1 + 1 = 3 \]

Usually, *polynomial-time* but *incomplete* algorithms are used for all this.
**Example:** Establishing **bounds consistency** on

\[2 \cdot A + 4 \cdot B = 24 \text{ and } A + B = 9, \text{ where } A, B \in 0\ldots9.\]

Initially:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

Posting \( 2 \cdot A + 4 \cdot B = 24 \) (**arc consistency** also prunes the **blue** values):

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

Posting \( A + B = 9 \):

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

Propagating to \( 2 \cdot A + 4 \cdot B = 24 \):

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>
Propagating to $A + B = 9$:

\[
\begin{array}{ccccccccc}
A & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
B & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\end{array}
\]

Propagating to $2 \cdot A + 4 \cdot B = 24$ and deactivating it (as definitely true):

\[
\begin{array}{ccccccccc}
A & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
B & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\end{array}
\]

Propagating to $A + B = 9$ and deactivating it (as it is definitely true):

\[
\begin{array}{ccccccccc}
A & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
B & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\end{array}
\]

*Fixpoint* reached!

In this particular case: no search is needed, as no constraint is suspended. The (only) solution is $A = 6$ and $B = 3$. 
Search

Suppose all the constraints become either deactivated or suspended:

If at least one constraint is suspended,
then one cannot know for sure whether there is any solution or not,
so one must search for values for all the decision variables.

A classical search procedure:

**While** there is at least one suspended constraint **do**:

**Pick** a decision variable $x$ whose domain $D$ has at least 2 elements.

**Pick** a value $d \in D$.

**Post** an additional constraint, called a decision,
say $x = d$, or $x \neq d$, or $x > d$, or $x \leq d$. (Propagation!)
Global Constraints

Example: The basic constraint \( B \neq C \) operates on 2 decision variables.

Definition: A basic constraint operates on a fixed number of arguments.

Example: The global constraint \( \text{allDifferent}([A,B,C,D]) \) operates here on a list of \( n = 4 \) decision variables, which have to take distinct values.

Example: The global constraint \( \text{atMost}(N,E,[A,B,C,D]) \) requires that there are at most \( N \) occurrences of \( E \) in the list of 4 decision variables.

Definition: A global constraint operates on any number of arguments.

Many other global constraints are necessary in practice, covering interesting problems from operations research, flow theory, graph theory, geometry, and so on.
Example: The global constraint \textit{allDifferent}([A,B,C,D]) operates here on a list of \( n = 4 \) decision variables, which have to take distinct values.

\textit{Declaratively}, it is equivalent to the \( n \cdot (n - 1) / 2 = 6 \) basic constraints

\[ A \neq B, \ A \neq C, \ A \neq D, \ B \neq C, \ B \neq D, \ \text{and} \ C \neq D. \]

It provides necessary and convenient genericity in constraint programs.

\textit{Operationally}, it prunes \textit{much stronger} than its basic constraints.

Example: Consider the domain declarations

\[ A \in \{2,3\}, \ B \in \{2,3\}, \ C \in \{1,3\}, \ \text{and} \ D \in \{1,2,3,4\} \]

for \textit{allDifferent}([A,B,C,D]).
Contributors to Constraint Technology

- **Artificial Intelligence:** Constraint networks, data-driven computation.
- **Logic Programming:** Non-determinism, backtracking.
- **Discrete Mathematics:** Combinatorics, graph theory, group theory.
- **Operations Research:** Flow analysis, modelling languages.
- **Algorithms and Data Structures:** Incrementality.
History of Constraint Technology

• **ALICE** (Jean-Louis Laurière, Paris, 1976)

• **CHIP** (ECRC Munich: 1987 – 1990)

• Industry (Bull, Cosytec, ILOG: 1990 – 1992)

• Libraries (1993 – …):
  
  C++: **ILOG Solver, CHIP, Gecode, Figaro**
  
  Java: **JChoco, Koalog, JCL, Minerva**
  
  Prolog: **SICStus Prolog, ECLiPSe, IF Prolog, GNU Prolog**
  
  OCaml: **FaCiLe**
  
  Oz: **Mozart**
Example: Scheduling

```
enum Tasks {masonry, carpentry, plumbing, ceiling, ..., garden, moving};
int duration[Tasks] = [7,3,8,3,1,2,1,2,1,1];
Activity a[t in Tasks](duration[t]);
DiscreteResource budget(29000); // UnaryResource, Reservoir, ...
minimize a[moving].end
subject to {
    a[masonry] precedes a[carpentry]; a[masonry] precedes a[plumbing];
    a[masonry] precedes a[ceiling]; a[carpentry] precedes a[roofing];
    a[ceiling] precedes a[painting]; a[roofing] precedes a[windows]; ...;
    a[garden] precedes a[moving]; a[painting] precedes a[moving];
    forall(t in Tasks) a[t] consumes(1000*duration[t]) budget; // requires capacityMax(budget,0,15,20000);
}; // P. van Hentenryck: The OPL Optimization Programming Language
```
Example: The Beer Tasting Party

A *beer tasting party* is an assignment of subsets of a set of \( v \) beers to \( b \) students, such that:

1. Each student tastes exactly \( k \) beers.
2. Each beer is tasted by exactly \( r \) students.
3. Each pair of distinct beers is tasted by exactly \( \lambda \) students.

This is a *balanced incomplete block design* (BIBD) and can be specified by a 5-tuple \( \langle v, b, r, k, \lambda \rangle \).

Usage areas include the design of statistical experiments, cryptography, …
**Example:** A solution to the $\langle v=7, b=7, r=3, k=3, \lambda=1 \rangle$ tasting party:

<table>
<thead>
<tr>
<th></th>
<th>Anton</th>
<th>Britt</th>
<th>Carl</th>
<th>Daniel</th>
<th>Emma</th>
<th>Fanny</th>
<th>Gustav</th>
</tr>
</thead>
<tbody>
<tr>
<td>Affligem</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Chimay</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Duvel</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Grimbergen</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Orval</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Rochefort</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Westmalle</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

The beers are *not distinguished*, nor are the students, hence the rows and the columns can be freely *permuted*, without affecting whether any given assignment is a solution or not. Breaking *a few* of these symmetries:

1. Every row is lexicographically larger than the next, if any.
2. Every col. is lexicographically larger than or equal to the next, if any.

For best efficiency, search row by row, left to right, trying the value ✔ first.
2. Real-Life Applications

Financial Mathematics: CDO Portfolio Design

In collaboration with Dr Luis Reyna at Swiss Re, in New York, USA

In the beer tasting party, replace the $v$ beers by baskets and the $b$ students by credits:

(1) Each credit appears in $???$ baskets.

(2) Each basket contains exactly $r$ credits.

(3) Each pair of distinct baskets contains at most $\lambda$ credits.

Constraint optimisation problem: What is the minimal value of $\lambda$?
**Example:** A typical portfolio is \( \langle 10, 350, 100 \rangle \), with \( >10^{746} \) symmetries, for which we can calculate that \( \lambda \geq 22 \):

<table>
<thead>
<tr>
<th>credit</th>
<th>credit</th>
<th>credit</th>
<th>3</th>
<th>…</th>
<th>credit</th>
<th>credit</th>
<th>credit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>348</td>
<td>349</td>
<td>350</td>
</tr>
<tr>
<td>basket 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>basket 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>basket 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>…</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>basket 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>basket 9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>basket 10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Designing this portfolio even for a given \( \lambda \) is way beyond the capabilities of the best available solvers…
Observation: As the credits seem indistinguishable and $350 = 11 \cdot 30 + 20$ and $100 = 11 \cdot 9 + 1$, a not necessarily optimal solution to $\langle 10,350,100 \rangle$ can be obtained by:

1. Making 11 copies of every column in a solution to $\langle 10,30,9 \rangle$.

2. Concatenating that solution with a solution to $\langle 10,20,1 \rangle$.

Let $\lambda_1$ be the minimal $\lambda$ for $\langle 10,30,9 \rangle$. We can calculate that $\lambda_1 \geq 2$. Let $\lambda_2$ be the minimal $\lambda$ for $\langle 10,20,1 \rangle$. We can calculate that $\lambda_2 \geq 0$.

Solving instances like $\langle 10,30,9 \rangle$ and $\langle 10,20,1 \rangle$ for a given $\lambda$ is a matter of a CPU second, and it turns out that $\lambda_1 = 2$ and $\lambda_2 = 0$.

Hence we have a solution with $\lambda = 11 \cdot \lambda_1 + \lambda_2 = 22$!
Air Traffic Control: Flight Scheduling

In collaboration with Carlos Garcia Avello at EuroControl, the European Organisation for the Safety of Air Navigation, in Brussels, Belgium

Definition: A flight plan is a sequence of 4D points that are to be connected by straight flight.

Definition: A revision of a flight plan is any combination of:

- Advancing / Postponing the take-off of the flight by $-5$…$+10$ minutes.
- Changing the altitude of its passage over a 2D way-point.
The complexity of a sector \( s \) at time \( t \) (and hence the stress of the controller) depends on:

- The number of flights in \( s \) at \( t \).
- The number of flights near the boundary of \( s \) at \( t \).
- The number of flights in climb or descent in \( s \) at \( t \).

**Input:** Flight plans for a set of adjacent sectors.

**Output:** *Revised* flight plans, some 20 to 90 minutes in advance, such that:

1. The maximum among the complexities of the sectors is minimised.
2. All the originally scheduled flights are considered.
3. Safety regulations (space and time separation) are met.
4. …
Bioinformatics: The Tree of Life (Phylogeny)

In collaboration with
Prof. Vincent Moulton at the University of East Anglia, UK,
Prof. Nicolas Beldiceanu at the École des Mines de Nantes, France,
and Dr Patrick Prosser at Glasgow University, UK

Tree: \((f, ((d, e), ((c, (a, b)), g)))\)
Triples: \(((a, b), c), ((d, e), c), ((c, b), e), ((e, b), f), ((a, g), f)\)
**Example:** Two published trees with sea birds, sharing two species:

- Oceanodroma castro
- Hydrobates pelagicus
- Macronectes giganteus
- Fulmarus glacialoides
- Fulmarus glacialis
- Buizeria bulwerii
- Procellaria cinerea
- Calonectris diomedea
- Puffinus assimilis
- Puffinus puffinus
- Puffinus yelkouan
- Puffinus mauretanicus
- Thalassarche bulleri
- Thalassarche chrysostoma
- Phoebetria fusca
- Phoebetria palpebrata
- Phoebastria albatrus
- Phoebastria immutabilis
- Diomedea amsterdamsis
- Diomedea epomophora

- Pygoscelis adeliae
- Eudyptula minor
- Megadytes antipodes
- Eudyptes pachyrhynchus
- Pelagodroma marina
- Diomedea epomophora
- Daption capense
- Pelecanoides georgicus
- Pachyptila vittata
- Pachyptila turtur
- Procellaria westlandica
- Puffinus griseus
- Puffinus huttoni
- Pterodroma inexpectata
- Pterodroma cookii