Constraint Technology for Solving Combinatorial Problems: Overview, Results, and Applications

Pierre Flener

Computing Science Division
Department of Information Technology
Uppsala University, Sweden

Acknowledgements: Co-authors & sponsors (STINT, VINNOVA, VR)
1. Constraint Technology in a Nutshell

**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:** $A + B < C$, where $A, B, C \in 1\ldots4$.

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

**Example:** Find the smallest number of colours that solve a given map colouring problem.

**Constraint Technology** offers:

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

The focus is here on finite, discrete domains.
Definition: A constraint program (or constraint model) usually consists of, in this sequence:

1. **Domain declarations** for 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?

Example: The following is a constraint program:

\[
A, B, C \in 1\ldots4 \\
A + B < C \\
labelling([A, B, C])
\]
Example: The constraint program

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

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 \textit{definitely true}: if so, then \textit{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 \textit{not} definitely true.

- Testing if a constraint is \textit{definitely false}: if so, then \textit{backtrack}!

  Example: The minimum 2 of \( A + B \) is not larger than the maximum 4 of \( C \), so the constraint \( A + B < C \) is \textit{not} definitely false.
• **Pruning** values that make the constraint false:
if not definitely true or false, then *suspend* it!

One may have to *search* later on!

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

**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

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

Initially:

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

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

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

Posting \( A + B = 9 \):

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

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

\[
\begin{array}{c|cccccccccc}
A & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
B & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\end{array}
\]
Propagating to $A + B = 9$:

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

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

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

Propagating to $A + B = 9$:

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

**Fixpoint**: Both constraints are now definitely true, and are deactivated!
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

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

**Example:** The basic constraint $B \neq C$ operates on 2 decision variables.

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

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

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

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.

Operationally, it prunes 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 \(\text{allDifferent}([A,B,C,D])\).
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)
- Libraries (1993 – …):
  - C++: ILOG Solver, CHIP, Figaro
  - Java: Koalog, JCL, Minerva
  - Prolog: SICStus, ECLiPSe, IF, GNU
  - Oz: Mozart
The Meze Tasting Party

A *meze tasting party* is an assignment of subsets of a set of $v$ mezes to $b$ diners, such that:

(1) Each diner tastes exactly $k$ mezes.

(2) Each meze is tasted by exactly $r$ diners.

(3) Each pair of distinct mezes is tasted by exactly $\lambda$ diners.

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 \(\langle 7,7,3,3,1 \rangle\) meze tasting party:

<table>
<thead>
<tr>
<th></th>
<th>Alkım</th>
<th>Bedriye</th>
<th>Cem</th>
<th>Deniz</th>
<th>Esra</th>
<th>Ferit</th>
<th>Gökhan</th>
</tr>
</thead>
<tbody>
<tr>
<td>börek</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dolma</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ezme</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>humus</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>kısır</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>sarma</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>tarator</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
4. Applications

Bioinformatics: The Tree of Life (Phylogeny)

In collaboration with the teams of Prof. Vincent Moulton at the Linnaeus Centre for Bioinformatics at Uppsala University, Sweden, and Prof. Nicolas Beldiceanu at the École des Mines de Nantes, France

```
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
- Buiweria 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 amsterdamiensis
  - DIOMEDEA EPOmorphora

- Pygoscelis adeliae
- Eudyptula minor
- Megadyptes antipodes
- Eudyptes pacificus
- Pelagodroma marina
- DIOMEDEA EPOmorphora
- THALASSARCHE BULLERI
  - Daption capense
  - Pelecanoides georgicus
  - Pachyptila vittata
  - Pachyptila turtur
  - Procellaria westlandica
  - Puffinus griseus
  - Puffinus huttoni
  - Pterodroma inexpectata
  - Pterodroma cookii
Financial Mathematics: Portfolio Selection

In collaboration with Dr Luis Reyna at Merrill Lynch, in New York, USA

In the meze tasting party, replace the $\nu$ mezes by sub-portfolios and the $b$ diners by bonds:

1. Each bond appears in $??$ sub-portfolios.
2. Each sub-portfolio contains exactly $r$ bonds.
3. Each pair of distinct sub-portfolios contains at most $\lambda$ bonds.

Constraint optimisation problem (COP): What is the minimal value of $\lambda$?
**Example:** A typical portfolio is $\langle 10,350,100 \rangle$, for which we can calculate that $\lambda \geq 21$:

<table>
<thead>
<tr>
<th>Sub-portfolio 1</th>
<th>Bond 1</th>
<th>Bond 2</th>
<th>Bond 3</th>
<th>...</th>
<th>Bond 348</th>
<th>Bond 349</th>
<th>Bond 350</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sub-portfolio 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sub-portfolio 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>Sub-portfolio 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sub-portfolio 9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sub-portfolio 10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Selecting this portfolio for a given $\lambda$ is way beyond the capabilities of the best available solvers…
Air Traffic Management: Flight Planning

In collaboration with the team of Dr Mete Çeliktin 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.

Pilots are allowed some flexibility for following the route and for matching each time of passage.
**Input:** Tactical flight plans for a geographic area.

**Output:** Minimally revised flight plans that meet safety regulations (space and time separation), some 20 to 60 minutes in advance.

A *revision* may consist of:

- Delaying / Advancing the passage of a flight over a point.
- Assigning a new route.

The impact of the current flexibility on the potential optimisation will be studied on traffic samples.

In a first step, we aim only at en-route flights. A later generalisation may extend the scope to take-offs and landings.