A Model Seeker: Extracting Global Constraint Models From Positive Examples

N. Beldiceanu* and H. Simonis+

* TASC team (INRIA/CNRS), Mines de Nantes, France
+ 4C, University College Cork, Ireland
In Pursuit of the Holy Grail

• “Constraint Programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it.” [E. Freuder]

• This is boring: we don’t want to state the problem, the computer should do this as well!
What is it all about?

From sample
3 0 2 1 to model
alldifferent(\langle V_1, V_2, V_3, V_4 \rangle)
alldifferent_interval(\langle V_1, V_2 \rangle, 2)
alldifferent_interval(\langle V_3, V_4 \rangle, 2)

C. Durr (exam)
Points to remember

• Learning constraint models from positive examples
• Start with vector of values
• Group into regular pattern
• Find constraint pattern that apply to group elements
• Using Constraint Seeker for Global Constraint Catalog
• Works for highly structured problems
Overview

- Motivation
- Input Format
- An Example
- Learning Algorithm
  - Transformations
  - Partition Generators
  - Constraint Seeker
  - Relevance Optimizer
  - Dominance Check
- Evaluation
- Conclusion
Motivation

- Constraint models can be hard to write
- Can we generate them automatically?
- User gives example solutions
- Systems suggest **compact conjunctions** of **similar** constraints
- Users accepts/rejects constraints and/or gives more samples
Global Constraints

• Define abstractions occurring in real-world situations
  – Useful in multiple problems
  – As general as possible (complexity)
  – Perform better propagation than naïve models
  – Sometimes: Balance propagation/effort

• Classical Example: alldifferent
  – Works on sets of variables, they must be pairwise different
  – Algorithmic choices/amount of propagation
Global Constraint Catalog

• Systematic description of published global constraints
• Started from 1999 by Beldiceanu, Carlsson and Rampon (SICS, EMN)
• 400 constraints on 3000 pages
• Formal definitions, meta-data and links
• Motivation

• **Input Format**
  • An Example
  • Learning Algorithm
    – Transformations
    – Partitions Generators
    – Constraint Seeker
    – Relevance Optimizer
    – Dominance Check
  • Evaluation
  • Conclusion
User oriented **input** format

Ideally, starts from the format used in books, on the web for presenting the solution of a problem.  
*(there may be *more than one* way)*

Very often solutions are represented as one (or several) **tables, boards, grids, ...**, with (sometime) extra information (**hints**, **parameters**).
**Input format:** flat sequence of integers

2 4 6 8 3 1 7 5
(positions in the different columns, *start from 1*)

1 3 5 7 2 0 6 4
(positions in the different columns, *start from 0*)

2 1 2 2 2 3 2 3 5 4 1 5 5 6 1
(index of cells, *start from 1, ordered*)

2 2 1 2 5 5 6 1 3 2 3 5 2 4 1
(index of cells, *start from 1, not ordered*)

1 2 2 4 3 6 4 8 5 3 6 1 7 7 8 5
(coordinates of cells, *start from 1, ordered*)

0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0
(flattened 0/1 matrix, 1 for occupied cells)
• Motivation
• Input Format
• **An Example**
  • Learning Algorithm
    – Transformations
    – Partitions Generators
    – Constraint Seeker
    – Relevance Optimizer
    – Dominance Check
• Evaluation
• Conclusion
On veut placer $n$ samouraïs sur une grille $n \times n$, de sorte qu’ils ne puissent pas s’attaquer. La situation est un peu différente de celle des $n$ reines. En effet, nous avons la promesse que $n = m^2$ pour un entier $m \geq 2$, et que la grille consiste en $n$ carrés élémentaires de taille $m \times m$, voir figure 1. Deux samouraïs peuvent s’attaquer s’ils sont placés soit dans la même colonne, soit dans la même ligne, soit dans le même carré élémentaire.
$n$ Samuraïs: model

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Ref</th>
<th>Trans</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>vector(4)</td>
<td>2241</td>
<td>id</td>
<td>alldifferent_consecutive_values*1</td>
</tr>
<tr>
<td>scheme(4,2,2,1,2)</td>
<td>2240</td>
<td>id</td>
<td>alldifferent_interval(2)*2</td>
</tr>
<tr>
<td>pan_diagonal(4,2,0)</td>
<td>2239</td>
<td>id</td>
<td>alldifferent_interval(2)*2</td>
</tr>
</tbody>
</table>

Constraints for Problem 4 Samurai

$3^1 0^2 2^3 1^4$

alldifferent_consecutive_values*1

$3^1 0^2 2^3 1^4$

alldifferent_interval(2)*2

$3^1 0^2 2^3 1^4$

alldifferent_interval(2)*2
$n$ Samuraïs: model

<table>
<thead>
<tr>
<th>J</th>
<th>Scheme</th>
<th>Ref</th>
<th>Trans</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>vector(4)</td>
<td>2241</td>
<td>id</td>
<td>alldifferent_consecutive_values*1</td>
</tr>
<tr>
<td>2</td>
<td>scheme(4,2,2,1,2)</td>
<td>2240</td>
<td>id</td>
<td>alldifferent_interval(2)*2</td>
</tr>
<tr>
<td>3</td>
<td>pan_diagonal(4,2,0)</td>
<td>2239</td>
<td>id</td>
<td>alldifferent_interval(2)*2</td>
</tr>
</tbody>
</table>

Constraints for Problem 4 Samurai

\[
\begin{array}{cccc}
3^1 & 0^2 & 2^3 & 1^4 \\
\end{array}
\]

alldifferent_consecutive_values*1

\[
\begin{array}{cccc}
3^1 & 0^2 & 2^3 & 1^4 \\
\end{array}
\]

alldifferent_interval(2)*2

\[
\begin{array}{cccc}
3^1 & 0^2 & 2^3 & 1^4 \\
\end{array}
\]

alldifferent_interval(2)*2

Eliminated if we provide more samples
n Samurais model
(two conjunctions of similar constraints)

\[
\begin{align*}
V_1 &= 2Q_1 + R_1 \quad (0 \leq R_1 < 2) \\
V_2 &= 2Q_2 + R_2 \quad (0 \leq R_2 < 2) \\
\text{alldifferent_consecutive_values}(V_1, V_2, V_3, V_4) \\
\text{alldifferent_interval}(V_1, V_2, 2) \\
\text{alldifferent_interval}(V_3, V_4, 2)
\end{align*}
\]
• Motivation
• Input Format
• An Example
  • **Learning Algorithm**
    – Transformations
    – Partition Generators
    – Constraint Seeker
    – Relevance Optimizer
    – Dominance Check
• Evaluation
• Conclusion
Workflow of the learning procedure
(from samples to program)

- Transformations
- Partition generators
- Arguments creation
- Constraint seeker
- Relevance optimizer
- Domain creation
- Link between objects attributes
- Dominance check (crucial)
- Trivial suppression
- Code generation (catalog syntax, FlatZinc)
Transformations

• Extract **substructures** from samples
  – Extracting **overlapping grids** from **irregular shapes**
  – Distinguish **main grid** from **hints on column and/or rows**

• Derive **new samples** from samples
  – Build **triangular differences table**
  – Take **sign** and/or **absolute value**

• Handle **multiple input formats** (*in a transparent way*)
  – **Bijection**
  – **Tour/Path**
  – **Domination in graphs**
Transformations, example 1
(Extracting overlapping grids from irregular shapes)

**IDEA**

Cover the non-empty space by the **minimum** number of rectangles in such a way that the **maximum** intersection between any pairs of rectangles is minimized.

*use a constraint program*
Euler first example on open knight’s tour; the numbers mark the order of the cells the knight visit.

Leaper graphs in Selected Papers on Fun and Games [Knuth 2010] the tour is given in base 9 (in order to highlight symmetries).

Number links (Nikoli) all cells belonging to a same path are labelled by the same number.
Pattern found in an old problem
(B. Franklin 1750 semi magic square)

Sum of half row and half columns=130

(transformation 7: from path to successor representation)
Partition generators

**Structured** groups of variables passed to a conjunction of **identical** constraints

![Sample Table]

<table>
<thead>
<tr>
<th>16</th>
<th>3</th>
<th>2</th>
<th>3</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>10</td>
<td>11</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>7</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>14</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

![Second Table]

<table>
<thead>
<tr>
<th>16</th>
<th>3</th>
<th>2</th>
<th>3</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>10</td>
<td>11</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>7</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>14</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

![Third Table]

<table>
<thead>
<tr>
<th>16</th>
<th>3</th>
<th>2</th>
<th>3</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>10</td>
<td>11</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>7</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>14</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Partitions generators

**Structured** groups of variables passed to a conjunction of **identical** constraints

```
<table>
<thead>
<tr>
<th>16^1</th>
<th>3^2</th>
<th>2^3</th>
<th>13^4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5^5</td>
<td>10^6</td>
<td>11^7</td>
<td>8^8</td>
</tr>
<tr>
<td>9^9</td>
<td>6^{10}</td>
<td>7^{11}</td>
<td>12^{12}</td>
</tr>
<tr>
<td>4^{13}</td>
<td>15^{14}</td>
<td>14^{15}</td>
<td>1^{16}</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>16^1</th>
<th>3^2</th>
<th>2^3</th>
<th>13^4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5^5</td>
<td>10^6</td>
<td>11^7</td>
<td>8^8</td>
</tr>
<tr>
<td>9^9</td>
<td>6^{10}</td>
<td>7^{11}</td>
<td>12^{12}</td>
</tr>
<tr>
<td>4^{13}</td>
<td>15^{14}</td>
<td>14^{15}</td>
<td>1^{16}</td>
</tr>
</tbody>
</table>
```

```
sum_ctr(34)*4
```
Partition generators

Structured groups of variables passed to a conjunction of identical constraints
Structured groups of variables passed to a conjunction of identical constraints.
Partitions generators

**Structured** groups of variables passed to a conjunction of **identical** constraints
**Partitions generators**

**Structured** groups of variables passed to a conjunction of **identical** constraints

```
<table>
<thead>
<tr>
<th>sum_ctr(34)*4</th>
</tr>
</thead>
<tbody>
<tr>
<td>16^1</td>
</tr>
<tr>
<td>3^2</td>
</tr>
<tr>
<td>2^3</td>
</tr>
<tr>
<td>13^4</td>
</tr>
<tr>
<td>5^5</td>
</tr>
<tr>
<td>10^6</td>
</tr>
<tr>
<td>11^7</td>
</tr>
<tr>
<td>8^8</td>
</tr>
<tr>
<td>9^9</td>
</tr>
<tr>
<td>6^10</td>
</tr>
<tr>
<td>7^11</td>
</tr>
<tr>
<td>12^12</td>
</tr>
<tr>
<td>4^13</td>
</tr>
<tr>
<td>15^14</td>
</tr>
<tr>
<td>14^15</td>
</tr>
<tr>
<td>1^16</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>sum_squares_ctr(748)*2</th>
</tr>
</thead>
<tbody>
<tr>
<td>16^1</td>
</tr>
<tr>
<td>3^2</td>
</tr>
<tr>
<td>2^3</td>
</tr>
<tr>
<td>13^4</td>
</tr>
<tr>
<td>5^5</td>
</tr>
<tr>
<td>10^6</td>
</tr>
<tr>
<td>11^7</td>
</tr>
<tr>
<td>8^8</td>
</tr>
<tr>
<td>9^9</td>
</tr>
<tr>
<td>6^10</td>
</tr>
<tr>
<td>7^11</td>
</tr>
<tr>
<td>12^12</td>
</tr>
<tr>
<td>4^13</td>
</tr>
<tr>
<td>15^14</td>
</tr>
<tr>
<td>14^15</td>
</tr>
<tr>
<td>1^16</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>alldifferent_interval(2)*8</th>
</tr>
</thead>
<tbody>
<tr>
<td>16^1</td>
</tr>
<tr>
<td>3^2</td>
</tr>
<tr>
<td>2^3</td>
</tr>
<tr>
<td>13^4</td>
</tr>
<tr>
<td>5^5</td>
</tr>
<tr>
<td>10^6</td>
</tr>
<tr>
<td>11^7</td>
</tr>
<tr>
<td>8^8</td>
</tr>
<tr>
<td>9^9</td>
</tr>
<tr>
<td>6^10</td>
</tr>
<tr>
<td>7^11</td>
</tr>
<tr>
<td>12^12</td>
</tr>
<tr>
<td>4^13</td>
</tr>
<tr>
<td>15^14</td>
</tr>
<tr>
<td>14^15</td>
</tr>
<tr>
<td>1^16</td>
</tr>
</tbody>
</table>
```
Structured groups of variables passed to a conjunction of identical constraints

\[
symmetric\_alldifferent\_loop([1..16])*1
\]

\[
surprise
\]

\[
symmetric\_alldiff\_loop(<x_1,x_2,...,x_n>)
\]

\[
x_1,x_2,...,x_n\text{ is a permutation of order 2 (an involution)}
\]

\[
x_i=j \iff x_j=i \text{ (} i \text{ may be equal to} j \text{)}
\]
Arguments creation + Constraint seeker

• Arguments creation
  – Use partition generators
  – Add arguments
    • as parameters (*extracted from sample*)
    • through **functional dependency**

• Constraint seeker (CP 2011)
  – Only **typical use**
Dominance check

- Certain conjunctions of constraints are dominated by others (*crucial to eliminate them to restrict output*)
- Weaker than full implication
- Use:
  - Implication and conditional implication (*given in the catalog*)
    - Sum of squares constraint equivalent to sum (*if 0/1 variables*)
  - Properties of constraints arguments (*given in the catalog*)
    - Contractible (*alldifferent*)
    - Extensible (*atleast*)
    - Aggregation (*among*)
  - Ad hoc conditional implication (*about 10 currently*)
• Motivation
• Input Format
• An Example
• Learning Algorithm

• Evaluation
• Conclusion
Evaluation: Problem Sizes

- **335** instances considered
  - Sample sizes  \((\text{from } 4 \text{ up to } 6551)\)
  - Number of samples  \((\text{from } 1 \text{ up to } 7040)\)

*Usually one single sample is enough (crucial point for a realistic use)*
A fair variety of problem types

No attack on a board  (e.g. queen, amazon, samurai)
Domination on a graph (e.g. queen, knight on a board, on a cube)
Tour/path on a graph (e.g. knight, leaper, number link)
Balanced block design (e.g. BIBD, Steiner, Kirkman)
Latin squares (e.g. standard, self-symmetric, orthogonal)
Sudoku (e.g. consecutive, samurai, anti diagonal, twin)
Sport scheduling (e.g. ACC Basketball, Bundesliga, Whist)
Scheduling (e.g. Job Shop)
Packing (e.g. squared squares, pallet loading, Conway 3d)
Magic/bimagic (e.g. sequence, squares, cubes)
Miscellaneous (e.g. tomography, progressive party, car sequencing)
## Results: Some Stats

<table>
<thead>
<tr>
<th>Category</th>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>from 20 ms up to 5 min.</td>
</tr>
<tr>
<td>Calls to the seeker</td>
<td>up to 5,044 calls</td>
</tr>
<tr>
<td>Calls to Constraints</td>
<td>up to 1,100,000 calls</td>
</tr>
<tr>
<td>Found conjunctions</td>
<td>up to 2207 <em>(before dominance check)</em></td>
</tr>
<tr>
<td># constraints used</td>
<td>69 out of 399 constraints in the catalog</td>
</tr>
</tbody>
</table>
• Motivation
• Input Format
• An Example
• Learning Algorithm
• Evaluation

• Conclusion
Conclusion

• Learning constraint models from very small sets of positive examples
• Start with vector of values
• Group into regular pattern
• Find constraint pattern that apply on group elements
• Using Constraint Seeker for Global Constraint Catalog
• Works for highly structured problems
Remarks

• Having many constraints allows to get **precise** models
• Filtering not used at all (*but need efficient checkers*)
• AI approach to learning (*knowledge base/no statistics*)
• Master student level (*maybe*)
• Of course the program does not invent new constraints, new generators, new transformations, ...
• Should provide an interface for presenting global constraint to normal users (*natural language + first order logical formulae for many constraints*).

*Extensive use of meta data describing constraints (e.g., typical case, functional dependency, imply, contractibility, checker, ...)*
Why does it work at all?

• Searching for conjunction of similar global constraints is the correct level of abstraction (find structured models)

• Learning at the level of a modelling language (OPL, Zinc, Essence) is too hard, as the language is too expressive

Global constraints were introduced for filtering, but they are key modelling constructs, and allow effective learning of models
What next?

- More examples (first real-world examples)
- More transformations
- More work on dominance check
- **Generic models** *(from different sample sizes)*
- Handle negative samples
- Include reformulation
- Include generation of implied constraints
- Uniform format for problem description and models for each problem and visualisation
- Learning automaton constraints for specific problems types *(e.g., still life)*