A Model Seeker: Extracting Global Constraint Models From Positive Examples

N. Beldiceanu[*] and **H. Simonis**[+]

[*]TASC team (INRIA/CNRS), Mines de Nantes, France

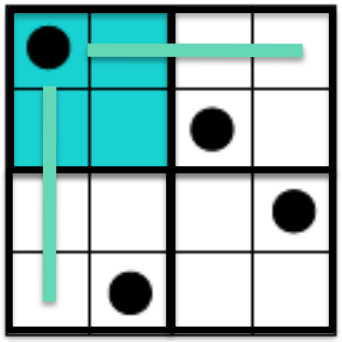
[+]4C, University College Cork, Ireland

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 ?

problem

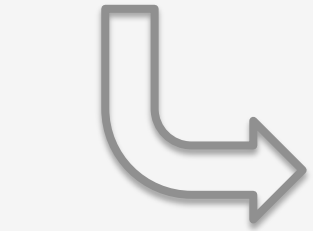


C. Durr (exam)

From sample

3 0 2 1

to model



LEARNING

alldifferent(<V_1,V_2,V_3,V_4>)

alldifferent_interval(<V_1,V_2>, 2)

alldifferent_interval(<V_3,V_4>, 2)

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 12 22 32 35 41 55 61

(index of cells, *start from 1, ordered*)

22 12 55 61 32 35 2 41

(index of cells, *start from 1, not ordered*)

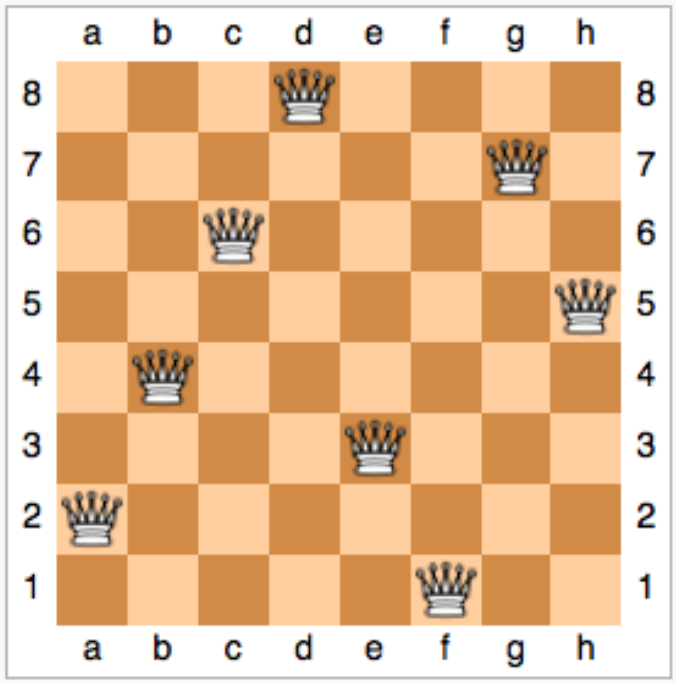
1 2 2 4 3 6 4 8 5 3 6 1 7 7 8 5

(coordinates of cells, *start from 1, ordered*)

0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0

(flat 0/1 matrix, *1 for occupied cells*)

different representations for the same solution



Source: wikipedia

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

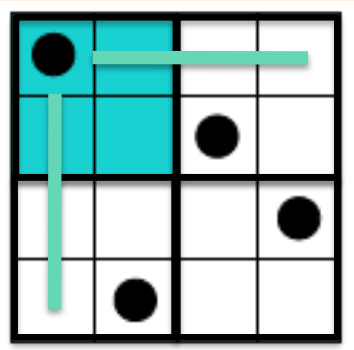
Constraint exam (*Polytechnique 2011*)

ORIGINAL QUESTION (*in French*)

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.

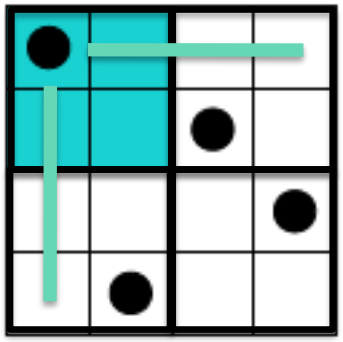
http://www.enseignement.polytechnique.fr/informatique/INF580/exams/ C. Durr

AN EXAMPLE



n Samuraïs: model

sample



3 0 2 1

model

J	Scheme	Ref	Trans	Constraint
1	vector(4)	2241	id	alldifferent_consecutive_values*1
2	scheme(4,2,2,1,2)	2240	id	alldifferent_interval(2)*2
3	pan_diagonal(4,2,0)	2239	id	alldifferent_interval(2)*2

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

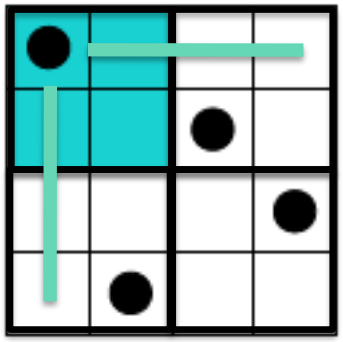
alldifferent_interval(2)*2

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

alldifferent_interval(2)*2

n Samuraïs: model

samples



3 0 2 1

0 2 1 3

.....

model

J	Scheme	Ref	Trans	Constraint
1	vector(4)	2241	id	alldifferent_consecutive_values*1
2	scheme(4,2,2,1,2)	2240	id	alldifferent_interval(2)*2
3	pan_diagonal(4,2,0)	2239	id	alldifferent_interval(2)*2

Constraints for Problem 4 Samurai

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

alldifferent_consecutive_values*1

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

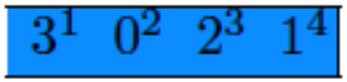
alldifferent_interval(2)*2

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

alldifferent_interval(2)*2

Eliminated if we provide more samples

n Samuraïs model
(*two conjunctions of similar constraints*)



alldifferent_consecutive_values(<V_1,V_2,V_3,V_4>)

alldifferent_interval(<V_1,V_2>, 2)

alldifferent_interval(<V_3,V_4>, 2)

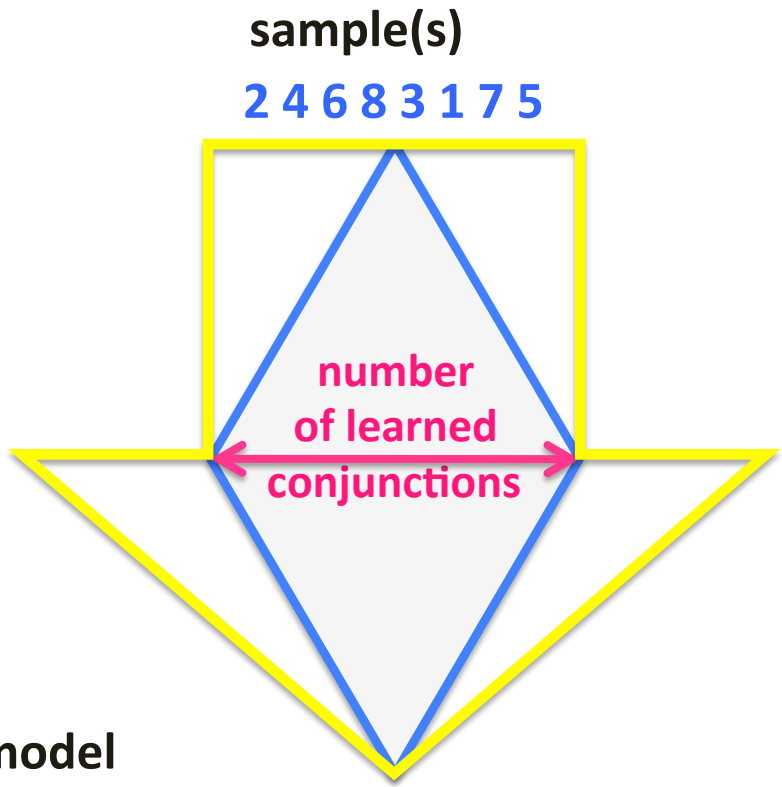
reformulation

$V_1=2*Q_1+R_1$ $(0 \leq R_1 < 2)$
$V_2=2*Q_2+R_2$ $(0 \leq R_2 < 2)$
alldifferent(<Q_1,Q_2>)

- 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*)



model

J	Scheme	Ref	Trans	Constraint
1	vector(7)	2048	id	permutation*1
2	vector(7)	2048	id	alldifferent_cst([7..1])*1
3	vector(7)	2048	id	alldifferent_cst([1..7])*1
4	vector(7)	2048	id	smooth(6,1)*1

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

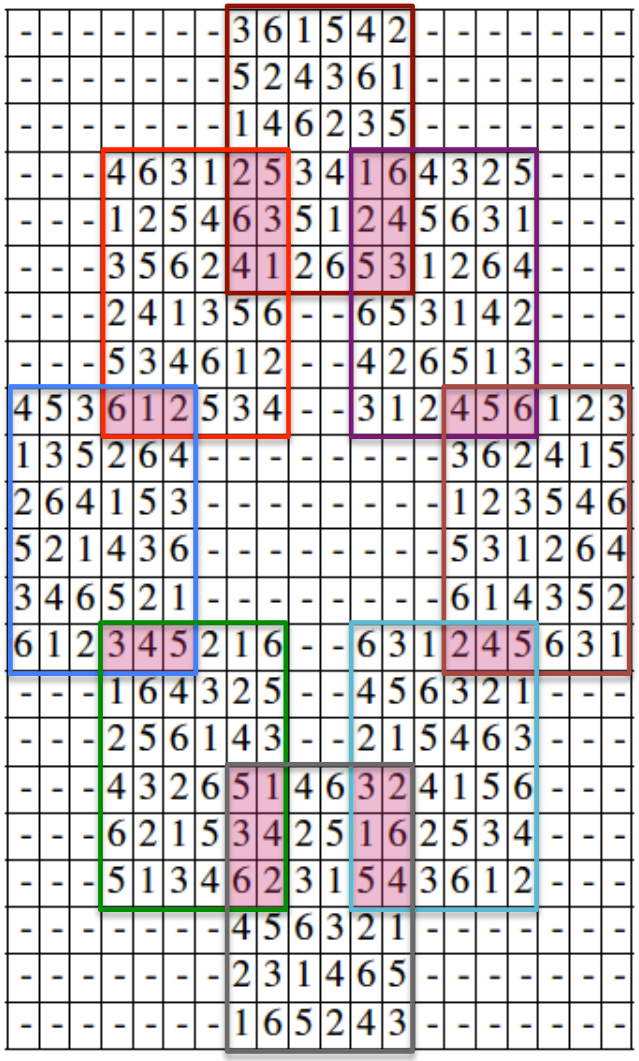
(*Extracting overlapping grids from irregular shapes*)

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

Flower Sudoku



Transformations, Example 2
(*tours/paths*)

Euler **first example** on open **knight's tour**;
the numbers mark the order of the cells the knight visit

32	13	54	27	56	23	…
63	52	31	24	29	26	…
14	33	2	51	16	35	…
1	64	15	34	3	50	…

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

0	272	220	43	53	333	363 …
270	222	41	55	212	51	331 …
224	38	57	210	277	214	48 …
36	60	207	280	386	275	216 …
334	362	105	182	84	388	273 …
52	332	364	103	1	271	221 …

Number links (**Nikoli**)
all cells belonging to a same path are labelled by the same number

2	2	2	2	2	2	2	2	2	2
2	7	7	7	7	7	7	**7**	**1**	2
2	7	**1**	1	1	1	1	1	1	2
2	7	**6**	6	6	6	6	6	**2**	2
2	7	7	5	5	5	5	6	6	6

Convert to successor representation and check that the underlying graph is regular

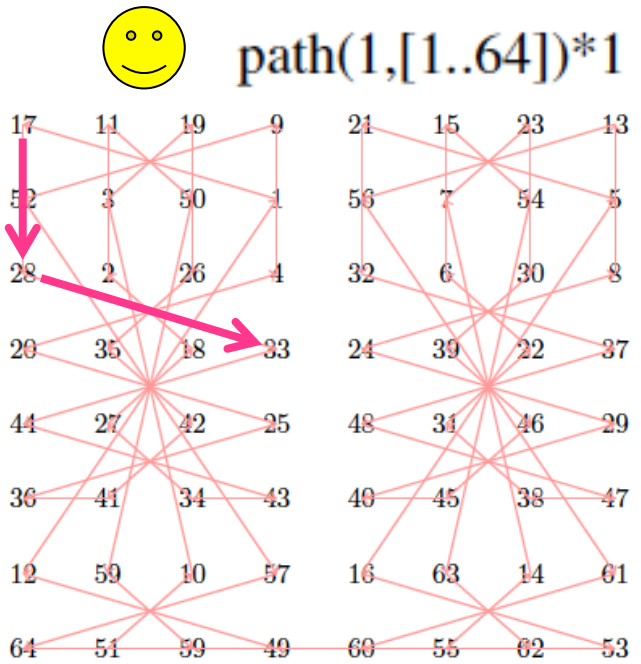
Pattern found in an old problem

(*B. Franklin 1750 semi magic square*)

sum_ctr(130)*16

52^{1}	61^{2}	4^{3}	13^{4}	20^{5}	29^{6}	36^{7}	45^{8}
14^{9}	3^{10}	62^{11}	51^{12}	46^{13}	35^{14}	30^{15}	19^{16}
53^{17}	60^{18}	5^{19}	12^{20}	21^{21}	28^{22}	37^{23}	44^{24}
11^{25}	6^{26}	59^{27}	54^{28}	43^{29}	38^{30}	27^{31}	22^{32}
55^{33}	58^{34}	7^{35}	10^{36}	23^{37}	26^{38}	39^{39}	42^{40}
9^{41}	8^{42}	57^{43}	56^{44}	41^{45}	40^{46}	25^{47}	24^{48}
50^{49}	63^{50}	2^{51}	15^{52}	18^{53}	31^{54}	34^{55}	47^{56}
16^{57}	1^{58}	64^{59}	49^{60}	48^{61}	33^{62}	32^{63}	17^{64}

Sum of half row and half columns=130



(transformation 7: from path to successor representation)

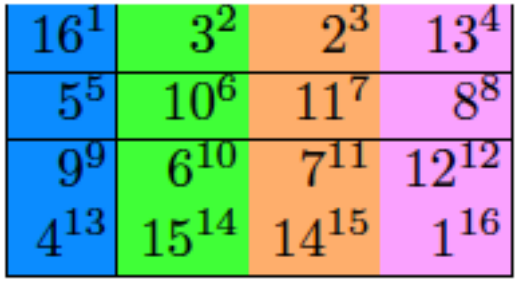
Partition generators

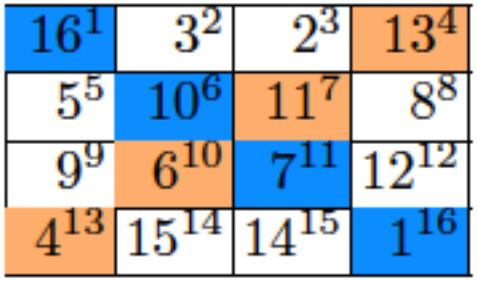
Structured groups of variables passed to a conjunction of **identical** constraints

sample

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

16^{1}	3^{2}	2^{3}	13^{4}
5^{5}	10^{6}	11^{7}	8^{8}
9^{9}	6^{10}	7^{11}	12^{12}
4^{13}	15^{14}	14^{15}	1^{16}





Partitions generators

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

sample

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

16^{1}	3^{2}	2^{3}	13^{4}
5^{5}	10^{6}	11^{7}	8^{8}
9^{9}	6^{10}	7^{11}	12^{12}
4^{13}	15^{14}	14^{15}	1^{16}

sum_ctr(34)*4

16^{1}	3^{2}	2^{3}	13^{4}
5^{5}	10^{6}	11^{7}	8^{8}
9^{9}	6^{10}	7^{11}	12^{12}
4^{13}	15^{14}	14^{15}	1^{16}

sum_ctr(34)*4

16^{1}	3^{2}	2^{3}	13^{4}
5^{5}	10^{6}	11^{7}	8^{8}
9^{9}	6^{10}	7^{11}	12^{12}
4^{13}	15^{14}	14^{15}	1^{16}

strictly_decreasing*2
sum_ctr(34)*2

Partition generators

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

sample

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

16^1	3^2	2^3	13^4
5^5	10^6	11^7	8^8
9^9	6^{10}	7^{11}	12^{12}
4^{13}	15^{14}	14^{15}	1^{16}

16^1	3^2	2^3	13^4
5^5	10^6	11^7	8^8
9^9	6^{10}	7^{11}	12^{12}
4^{13}	15^{14}	14^{15}	1^{16}

16^1	3^2	2^3	13^4
5^5	10^6	11^7	8^8
9^9	6^{10}	7^{11}	12^{12}
4^{13}	15^{14}	14^{15}	1^{16}

Partitions generators

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

sample

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

16^{1}	3^{2}	2^{3}	13^{4}
5^{5}	10^{6}	11^{7}	8^{8}
9^{9}	6^{10}	7^{11}	12^{12}
4^{13}	15^{14}	14^{15}	1^{16}

sum_ctr(34)*4

surprise

16^{1}	3^{2}	2^{3}	13^{4}
5^{5}	10^{6}	11^{7}	8^{8}
9^{9}	6^{10}	7^{11}	12^{12}
4^{13}	15^{14}	14^{15}	1^{16}

sum_squares_ctr(358)*2

surprise

16^{1}	3^{2}	2^{3}	13^{4}
5^{5}	10^{6}	11^{7}	8^{8}
9^{9}	6^{10}	7^{11}	12^{12}
4^{13}	15^{14}	14^{15}	1^{16}

sum_squares_ctr(390)*2

surprise

Partitions generators

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

sample

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

16^1	3^2	2^3	13^4
5^5	10^6	11^7	8^8
9^9	6^{10}	7^{11}	12^{12}
4^{13}	15^{14}	14^{15}	1^{16}

16^1	3^2	2^3	13^4
5^5	10^6	11^7	8^8
9^9	6^{10}	7^{11}	12^{12}
4^{13}	15^{14}	14^{15}	1^{16}

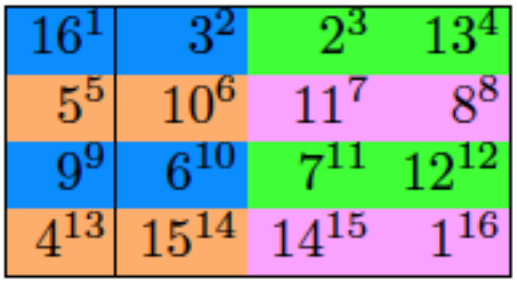
16^1	3^2	2^3	13^4
5^5	10^6	11^7	8^8
9^9	6^{10}	7^{11}	12^{12}
4^{13}	15^{14}	14^{15}	1^{16}

Partitions generators

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

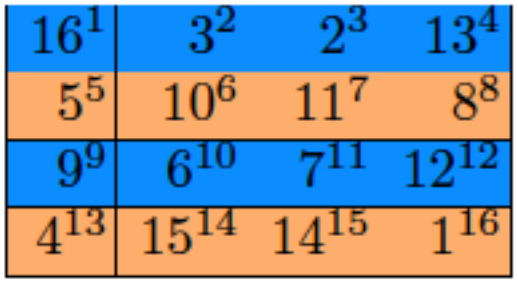
sample

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1



sum_ctr(34)*4

surprise



sum_squares_ctr(748)*2

surprise

alldifferent_interval(2)*8

Partition generators (end)

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

sample

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

16^{1}	3^{2}	2^{3}	13^{4}
5^{5}	10^{6}	11^{7}	8^{8}
9^{9}	6^{10}	7^{11}	12^{12}
4^{13}	15^{14}	14^{15}	1^{16}

symmetric_alldifferent_loop([1..16])*1

surprise

symmetric_alldiff_loop(<x_1,x_2,…,x_n>)

x_1,x_2,…,x_n is a permutation of order 2 (*an involution*)

x_i=j ⇔ x_j=i (*i may be equal to j*)

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**

EXAMPLE

$\texttt{atleast}(\texttt{N}, \texttt{VARIABLES}, \texttt{VALUE})$

Typical

$\texttt{N} > 0$

$\texttt{N} < |\texttt{VARIABLES}|$

$|\texttt{VARIABLES}| > 1$

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 (*from 4 up to **6551***)
  - Number of samples (*from 1 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, samuraï*)

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

Time	: from 20 ms up to 5 min.
Calls to the seeker	: up to 5,044 calls
Calls to Constraints	: up to 1,100,000 calls
Found conjunctions	: up to 2207 (*before dominance check*)
# constraints used	: 69 out of 399 constraints in the catalog

- 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*)