Centre for
Data Analytics

Insight

The ModelSeeker - Learning Structured Constraint Models from Example Solutions

Helmut Simonis

Progress Towards the Holy Grail Workshop, CP 2017

DCU NUI Galway OÉ Gaillimh UCC UCD Dublin SFI

Joint work with...

- Nicolas Beldiceanu, TASC team (CNRS/INRIA), IMT Atlantique, France
- Contributions by
  - Georgiana Ifrim, Insight UCD, Ireland
  - Arnaud Lenoir, EDF Research, France
  - Jean-Yves Lucas, EDF Research, France
  - Mats Carlsson, SICS, Sweden
  - Naina Razakarison, ENS Cachan, France
- Special thanks for examples due to
  - Hakan Kjellerstrand, Sweden

Outline

Background

Part I: Learning global constraint Models from Sample Solutions

Part II: Generalizing Problem Parameters

Part III: Industrial Case Study

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]
- Why do we have to specify the problem? The computer should at least help us to do this.

What is New?

- Exploit regular structure of many constraint problems
- Global Constraint Catalog provides repository of constraints used in systems
- Provides appropriate bias for learning models
- Use meta-data describing key properties of global constraints
- Use logic programming to provide flexible environment

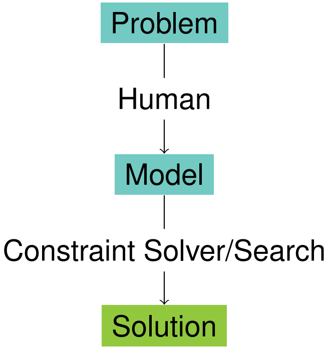
Is this different?

- Constraint Aquisition
    - Version space learning
    - Learning binary constraints
    - Asking many, many questions
    - Even if query complexity is optimal
- Inductive Logic Programming
    - Generate size independent models
    - Does not understand global constraints

Structure of Talk

- Learning models from solutions of fixed size
- How to generalize models by learning size parameters
- Industrial case study (EDF generator profiles)

Basic Process



Dual Role of Model

- Allows Human to Express Problem
  - Close to Problem Domain
  - Constraints as Abstractions
- Allows Solver to Execute
  - Variables as Communication Mechanism
  - Constraints as Algorithms

Global Constraint Catalog

- Collection of global constraints described in systems (Beldiceanu, Carlsson from 1999)
- Human and machine readable format
- Describe properties and relations between constraints
- Currently 443 constraints on 2712 pages
- 50000 lines of Prolog description

Outline

Background

Part I: Learning global constraint Models from Sample Solutions

Part II: Generalizing Problem Parameters

Part III: Industrial Case Study

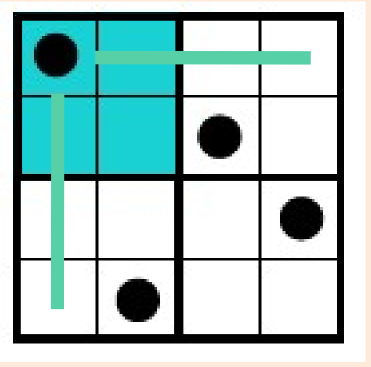
Constraint exam (*Polytechnique 2011*)

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.

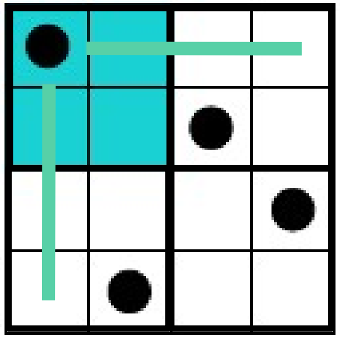
http://www.enseignement.polytechnique.fr/informatique/INF580/exams/ C. Durr

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

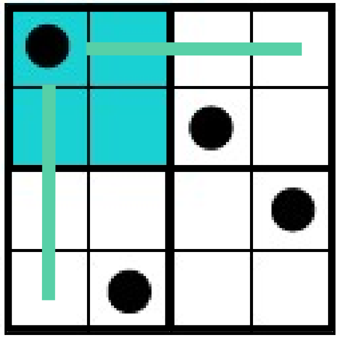
alldifferent_interval(2)*2

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

alldifferent_interval(2)*2

n Samuraïs: model

samples



3 0 2 1

0 2 1 3

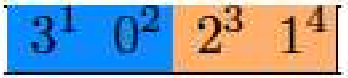
....

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

alldifferent_consecutive_values*1



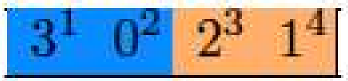
alldifferent_interval(2)*2

~~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)

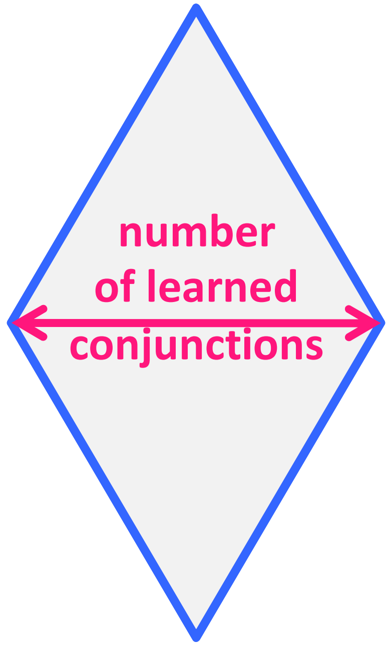
alldifferent_interval($<V_3,V_4>$, 2)

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($<Q_1,Q_2>$)

Workflow of the learning procedure
(*from samples to program*)

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



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

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

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*)

We start from that idea

Input format: flat sequence of integers

different representations for the same solution

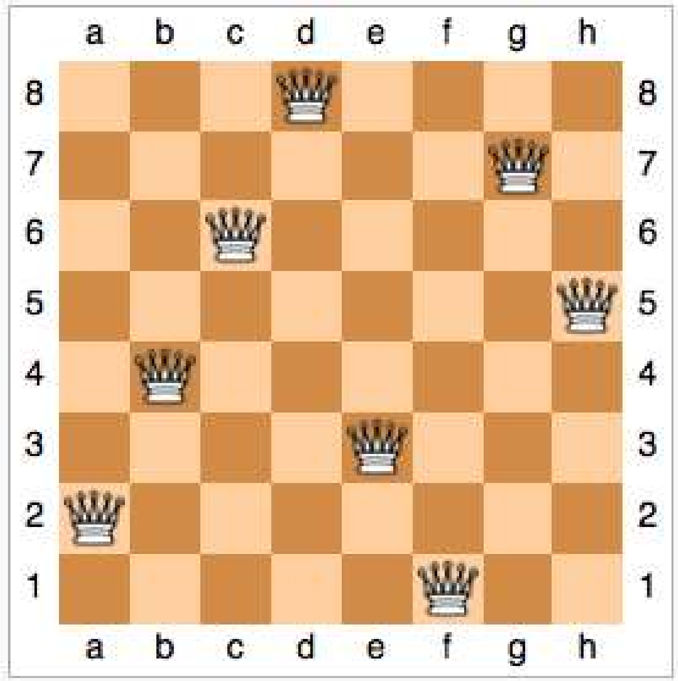
2 4 6 8 3 1 7 5
(positions in the different columns, *start from 1*)

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*)



Source: wikipedia

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

input format: parameters

Integrated in the sample: automatically extracted by transformations

car sequencing

0	1	0	1	1	0
1	0	0	0	1	0
5	1	1	0	0	0
2	0	1	0	0	1
4	1	0	1	0	0
3	0	1	0	1	0
3	0	1	0	1	0
4	1	0	1	0	0
2	0	1	0	0	1
5	1	1	0	0	0
-	[1,2]	[2,3]	[1,3]	[2,5]	[1,5]

logigraphe

0	1	0	1	0	0	0	1	0	1	[1,1,1,1]
1	1	0	1	1	1	1	0	0	0	[2,4]
0	1	1	1	1	1	1	1	0	0	[7]
0	0	0	1	1	1	1	1	1	0	[6]
0	1	1	0	0	0	0	1	1	1	[2,3]
1	0	0	0	1	0	1	0	1	0	[1,1,1,1]
1	0	1	1	1	0	1	0	0	0	[1,3,1]
1	0	0	0	1	0	0	1	0	1	[1,1,1,1]
1	0	0	1	0	1	0	0	1	0	[1,1,1,1]
0	0	0	0	0	1	1	0	0	0	[2]
[1,4]	[3,1]	[1,1,1]	[4,1,1]	[3,3]	[3,2]	[3,2,1]	[1,3,1]	[3,1]	[1,1,1]	-

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

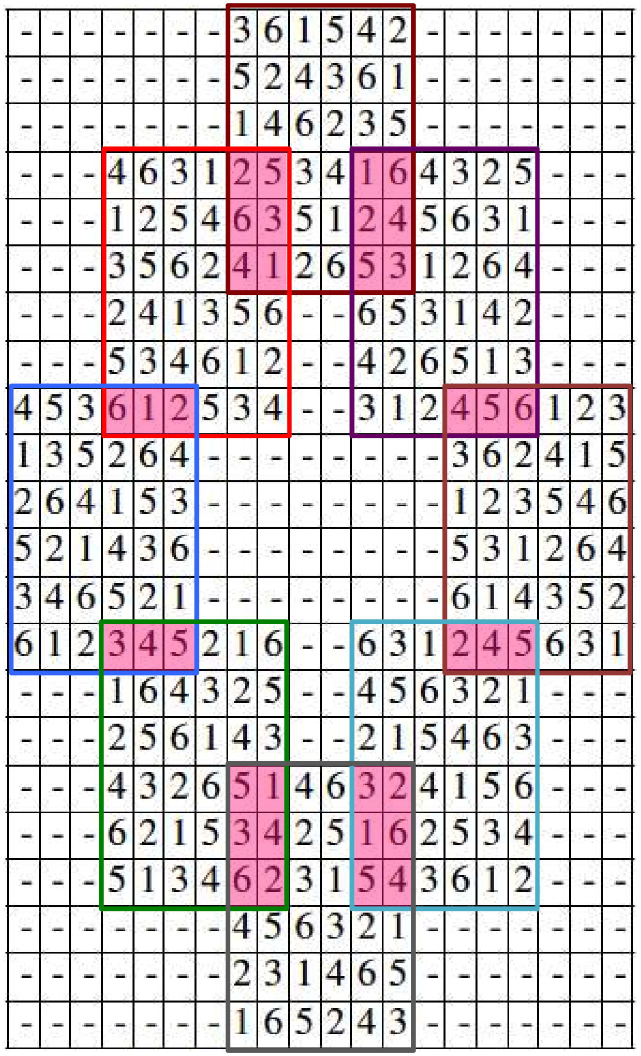
(*Extracting overlapping grids from irregular shapes*)

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

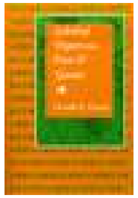


Transformations, Example 2

(*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*)

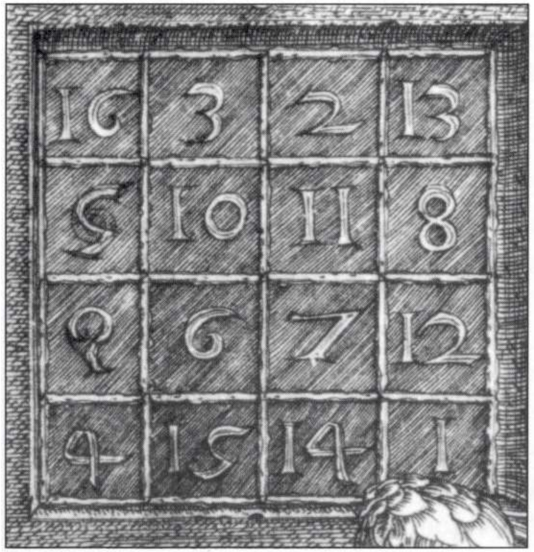
0	272	220	43	53	333	363	…
270	222	41	55	212	51	331	
224	38	57	210	277	214	48	…
36	60	207	280	386	275	216	…
334	362	105	182	84	388	273	
52	332	364	103	1	271	221	…

Numberlink (`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	**2**	2	2
2	7	7	5	5	5	5	6	6	6

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

Magic Square Example



Albrecht Dürer: Melencolia I (1514)

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}

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}

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}

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}

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

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}

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}

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(748)*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}

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($<x_1,x_2,\dots,x_n>$)

$x_1,x_2,\dots,x_n$ is a permutation of order 2 (*an involution*)

$x_i=j \Leftrightarrow x_j=i$ (*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

atleast(N, VARIABLES, VALUE)

Typical
$N > 0$
$N < |VARIABLES|$
$|VARIABLES| > 1$

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*)

Evaluation: Problem Sizes

- **350** instances considered
  - Special thanks to **Håkan Kjellerstrand**
  - 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(130) out of 399 constraints in the catalog

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 (finding structured models)
- Learning at the level of a modelling language (OPL, Zinc, Essence) is too hard, as the **language is too expressive**
- Learning inequalities (MIP) or clauses (SAT) is **too generic**

Global constraints are usually introduced for filtering, but they are key modelling constructs, and allow effective learning of models

Outline

Background

Part I: Learning global constraint Models from Sample Solutions

Part II: Generalizing Problem Parameters

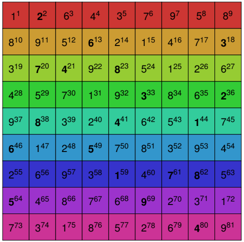
Part III: Industrial Case Study

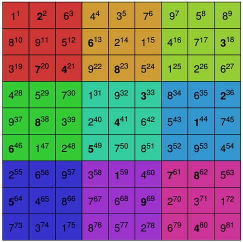
From Fixed Size Samples to Generic Models

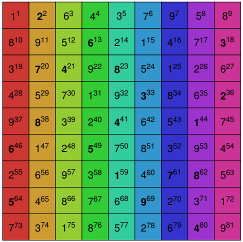
- Work in progress
- Combine models found for multiple problem sizes
- Replace size specific parameters with size dependent functions
- Suggest potential solutions from (very) few samples

Example: Sudoku 9x9

Partition sample in different ways:







Generated Model (compact representation):

scheme(81,9,9,9,1) `permutation`*9
scheme(81,9,9,3,3) `permutation`*9
scheme(81,9,9,1,9) `permutation`*9

Example: Sudoku 9x9,16x16,25x25

	scheme(81,9,9,9,1)	`permutation`*9
Sudoku 9x9	scheme(81,9,9,3,3)	`permutation`*9
	scheme(81,9,9,1,9)	`permutation`*9
	scheme(256,16,16,16,1)	`permutation`*16
Sudoku 16x16	scheme(256,16,16,4,4)	`permutation`*16
	scheme(256,16,16,1,16)	`permutation`*16
	scheme(625,25,25,25,1)	`permutation`*25
Sudoku 25x25	scheme(625,25,25,5,5)	`permutation`*25
	scheme(625,25,25,1,25)	`permutation`*25

Next Step: Generalize Models

- Find parametrized models
- One model for all problem sizes
- Parameters expressed as polynomials of one or multiple parameters
- Assumptions
    - Very few (1-3) samples
    - Highly structured problems lead to simple polynomials
- Learning polynomials is expressed as a constraint problem

Generic Model: Sudoku $n \times n$

From multiple specific models:

	scheme(81,9,9,9,1)	permutation*9
Sudoku 9x9	scheme(81,9,9,3,3)	permutation*9
	scheme(81,9,9,1,9)	permutation*9
	scheme(256,16,16,16,1)	permutation*16
Sudoku 16x16	scheme(256,16,16,4,4)	permutation*16
	scheme(256,16,16,1,16)	permutation*16
	scheme(625,25,25,25,1)	permutation*25
Sudoku 25x25	scheme(625,25,25,5,5)	permutation*25
	scheme(625,25,25,1,25)	permutation*25

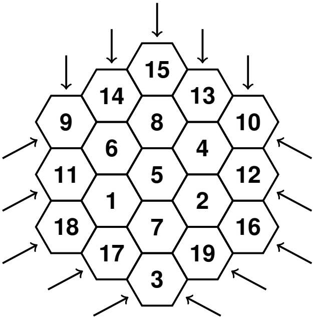
To one generic model:

scheme(n^4,n^2,n^2,n^2,1) permutation*n^2

scheme(n^4,n^2,n^2,n,n) permutation*n^2

scheme(n^4,n^2,n^2,1,n^2) permutation*n^2

More Complex Example: Magic Hexagon



5 samples, independent parameters $x_1 = [3, 4, 5, 6, 7]$, $x_2 = [1, 3, 6, 21, 2]$, dependent parameter $y_1 = [190, 777, 2196, 6006, 8255]$

$$2y_{1k} = 9x_{1k}^4 + 6x_{1k}^2 x_{2k} - 18x_{1k}^3 - 6x_{1k}x_{2k} + 12x_{1k}^2 + 2x_{2k} - 3x_{1k}$$

Outline

Background

Part I: Learning global constraint Models from Sample Solutions

Part II: Generalizing Problem Parameters

Part III: Industrial Case Study

Take away message

- Apply ModelSeeker Approach to problem from EDF
- Find constraints in Unit Commitment Problem
- Modelled with constraints having functional dependencies
- Generate sample output similar to input data

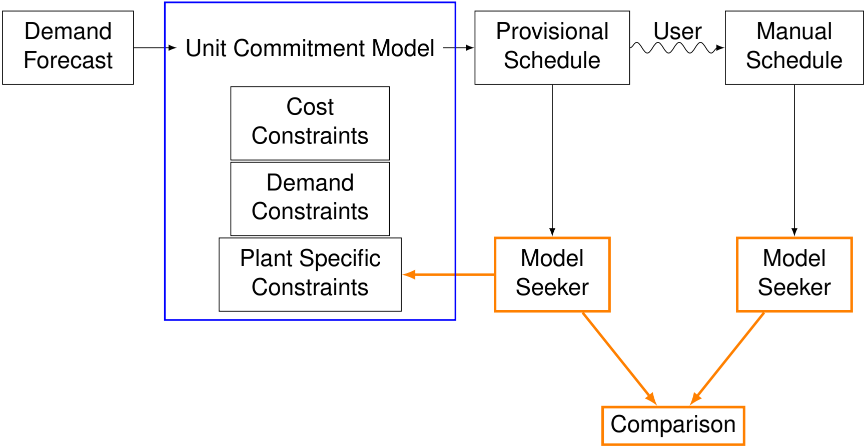
EDF Unit Commitment Model (UCM)

- Planning the use of all power stations in France for next two days
  - Run every day
- Based on demand prediction, minimizing production cost
- Each plant is defined by its own constraints
- Very large optimization model (MIP/Lagrangian Relaxation)
- Execution time critical to process

Problem: Identify Plant Specific Constraints

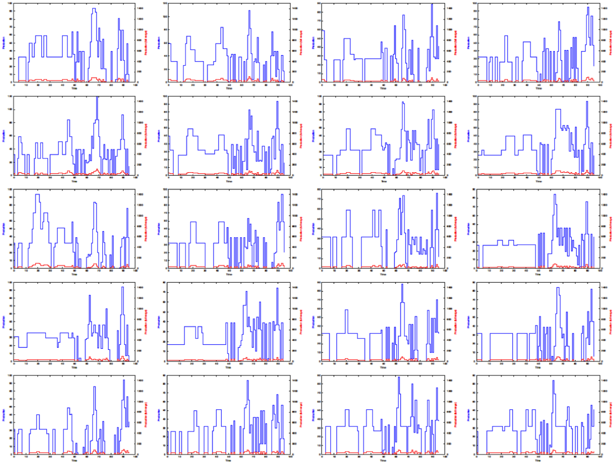
- Large part of model are plant specific constraints
- This determines how a plant can be scheduled
- Big differences between different types of plants (nuclear, thermal, hydro)
- Different parameter values for each plant (even if same type)
- Ignore at the moment:
  - Matching demand (+ handling of reserves)
  - Minimizing cost
  - Seasonal/weather effects (especially hydro)

Schema

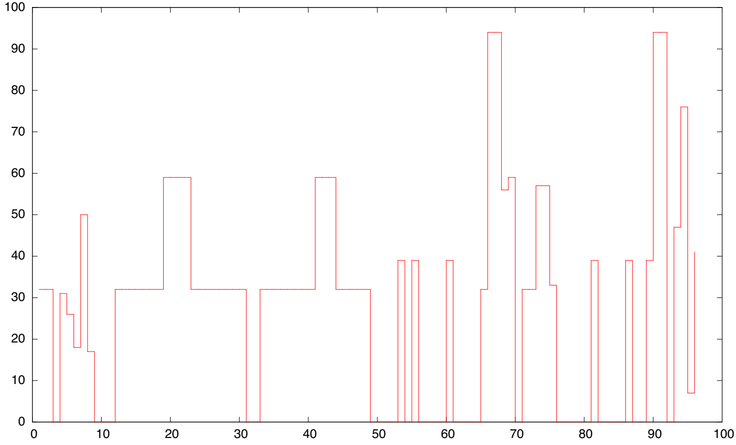


Demand Forecast
Unit Commitment Model
Cost Constraints
Demand Constraints
Plant Specific Constraints
Provisional Schedule
User
Manual Schedule
Model Seeker
Model Seeker
Comparison

Example: From this...



... to this (Generated Profile)



Summary

- Learning constraint models from few, positive examples
- Generalize models to arbitrary size (work in progress)
- Specific problem domain leads to more specific model generator

Bibliography

- N. Beldiceanu, H. Simonis: A Constraint Seeker: Finding and Ranking Global Constraints from Examples. CP 2011: 12-26
- N. Beldiceanu, H. Simonis: A Model Seeker: Extracting Global Constraint Models from Positive Examples. CP 2012: 141-157
- N. Razakarison, M. Carlsson, N. Beldiceanu, H. Simonis: GAC for a Linear Inequality and an Atleast Constraint with an Application to Learning Simple Polynomials. SOCS 2013
- N. Beldiceanu, G. Ifrim, A. Lenoir, H. Simonis: Describing and Generating Solutions for the EDF Unit Commitment Problem with the ModelSeeker. CP 2013: 733-748