

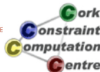
A Constraint Seeker: Finding and Ranking Global Constraints from Examples

N. Beldiceanu (TASC, Nantes) and H. Simonis (4C, Cork)

Ecole des Mines de Nantes, TASC (CNRS/INRIA), France

Cork Constraint Computation Centre, University College Cork, Ireland

CP 2011, Perugia



Points to Remember

- Set of tools for constraint catalog
- Search for constraints by example
- **Positive** or **negative** ground instances
- Produces **ranked list** of candidates
- Uses **catalog description** to build constraint models
- Also generates tests/counts solutions



Dealing with many different global constraints is not a problem, since meta data and meta programming help producing tools and automating their use!

A Constraint Seeker

- **Web tool** to search for global constraints by example
(*without knowing name and arguments*)
- **Guide user** to relevant entries in on-line catalog
- Does not need to identify constraints completely
- Uses multiple on-line and off-line constraint models
- On-line service at EMN:

`http://seeker.mines-nantes.fr/`

Additional Motivation

- Suggest **redundant constraints** (e.g., amazon puzzle)
- Address the **ranking** problem (*crucial when considering many alternatives*)
- Experiment with a system containing a **large number of constraints** (*address scalability in number of constraints handled*)
- Building block for constraint acquisition tool (ModRef talk yesterday)

Wider Context: Demonstrate the advantages of using meta data and meta programming

- **Meta data for describing different aspects of global constraints** (e.g., *argument type, restrictions, typical use, symmetries, links between constraints*)
- Not only useful for the constraint seeker, **but also** for:
 - 1 Generating the constraint catalog
 - 2 Estimating **solution density** of a constraint
 - 3 Extracting from meta data information used for the ranking
 - 4 Generating **typical examples** with some degree of diversity
 - 5 Generating **discriminating examples**
 - 6 Simulating the effect of various **consistencies**
 - 7 Getting interesting examples of **missing propagation**
 - 8 Providing **certificates** for testing constraints



LABORATOIRE D'INFORMATIQUE
DE NANTES ATLANTIQUE



Wider Context (*end*)

- All this is done **without** providing ad-hoc code that depends on a specific constraint (*only meta-data and evaluator required*)
- Currently **every** constraint system **repeats** some of these tasks in an **ad-hoc way** (*code/documentation has to be modified each time a constraint is updated or added in the system*).

Background

- Global Constraint Catalog (1999-2011)
- Work mainly at **SICS** and **EMN**
- 350+ constraints
- 2800+ pages (pdf)
- Website (<http://www.emn.fr/z-info/sdemasse/gccat>)
- Formal description of constraints with meta data
- **Everything else is generated**

Searching in Catalog

- Search by **name**
- Search by **keyword**
- Search by **type**
- **Links** between constraints

Searching in Other Catalogs

- **Integer sequence** search tool
- Numbers, math (**Wolfram Alpha**)
- Consistent naming schemes (Chemistry)
- Search by type in software libraries (**CamLight**)
- Electronic symbol library
- Music search (**Tunatic, Midomi**)

Outline

- 1 User Experience
 - Single Example
 - Adding More Examples
 - Transformations
- 2 How does it work?
- 3 Evaluation

Web Interface



Global Constraint Seeker

Examples

+ ▾ p(2,[4,2,4,5],4)

+ ▾

+ ▾

+ ▾

+ ▾

+ ▾

Submit Query

Example Query

- $p(2, [4,2,4,5], 4)$
- Everything is ground
- Coloring by argument
- Literal $p()$ or $n()$ for positive or negative
- Lists [...]
- Basic types: int, atom
- No distinction between collections and objects

Result

Constraint	Rank	Density	Links	UnTyp	ArgOrder	Crisp	Func
exactly	0	3125	9	0	0	n/a	graph
Pattern:	exactly(2 , [[var-4],[var-2],[var-4],[var-5]] , 4)						
Relations:	exactly implies atmost exactly implies atleast						
atleast	5	5625	11	0	0	0	n/a
Pattern:	atleast(2 , [[var-4],[var-2],[var-4],[var-5]] , 4)						
Relations:	atleast implied by exactly						
atmost	5	9188	12	0	0	0	n/a
Pattern:	atmost(2 , [[var-4],[var-2],[var-4],[var-5]] , 4)						
Relations:	atmost implied by exactly						
minimum_greater_than	30	2146	10	0	3	n/a	n/a
Pattern:	minimum_greater_than(4 , 2 , [[var-4],[var-2],[var-4],[var-5]])						
int_value_precede	30	7060	11	0	3	n/a	n/a
Pattern:	int_value_precede(4 , 2 , [[var-4],[var-2],[var-4],[var-5]])						
atmost	30	9188	12	1	2	3	n/a
Pattern:	atmost(4 , [[var-4],[var-2],[var-4],[var-5]] , 2)						
Relations:	atmost implied by exactly						

Wider Result

Constraint	Rank	Density	Links	UnTyp	ArgOrder	Crisp	Func	Tr
element	0	2500	35	0	0	n/a	manual	T11
Pattern:	<i>element</i> (3 , [[value-4],[value-2],[value-4],[value-5]] , 4)							
count	30	3125	16	0	3	n/a	n/a	T1
Pattern:	<i>count</i> (4 , [[var-4],[var-2],[var-4],[var-5]] , = , 2)							

Adding More Examples

$p(2, [4, 2, 4, 5], 4)$

$p(1, [1, 6, 3, 8, 8], 8)$

Adding Instance (Result)

Constraint	Rank	Density	Links	UnTyp	ArgOrder	Crisp	Func	Tr
atleast	5	5625	11	0	0	0	n/a	-
Pattern:	atleast(2 , [[var-4],[var-2],[var-4],[var-5]] , 4)							
atmost	30	9188	12	1	2	3	n/a	-
Pattern:	atmost(4 , [[var-4],[var-2],[var-4],[var-5]] , 2)							

Rule Based Modification of Examples

- Some constraints could be specified in **multiple ways**
- Catalog picks one representation only
- User may give example in different form
- Transform input into form recognized by catalog
- Apply Pre- and/or post-conditions to make transformation meaningful

Cumulative Constraint (CHIP Syntax)

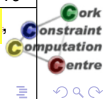
$p(8, [1,2,3,6,7], [3,9,10,6,2], [4,11,13,12,9], [1,2,1,1,3])$

Transformation Results

Constraint	Rank	Density	Links	Un typ	ArgOrder	Crisp	Func	Tr
cumulative	10	35561	34	1	1		n/a	T6
Patterns		cumulative	[origin-1.duration-3.and-4.height-1] [origin-2.duration-9.and-11.height-2] [origin-3.duration-10.and-13.height-1] [origin-6.duration-6.and-12.height-1] [origin-7.duration-2.and-9.height-3] 8					
Heuristics		cumulative	implies coloured cumulative					
cumulative_product	20	5457	8	1	1	2	n/a	T6
Patterns		cumulative_product	[origin-1.duration-3.and-4.height-1] [origin-2.duration-9.and-11.height-2] [origin-3.duration-10.and-13.height-1] [origin-6.duration-6.and-12.height-1] [origin-7.duration-2.and-9.height-3] 8					
almost_invector	30	59684	11	1	2	3	n/a	T6
Patterns		almost_invector	8 , [vec-1][vec-4][vec-3][vec-13] [vec-2][vec-11][vec-9][vec-20] [vec-7][vec-13][vec-10][vec-33] [vec-10][vec-12][vec-6][vec-43] [vec-16][vec-9][vec-26][vec-78] 3					
cumulative	40	35561	34	1	1	4	n/a	T6
Patterns		cumulative	[origin-3.duration-1.and-4.height-1] [origin-9.duration-2.and-11.height-2] [origin-10.duration-3.and-13.height-1] [origin-6.duration-6.and-12.height-1] [origin-5.duration-7.and-9.height-3] 8					
Heuristics		cumulative	implies coloured cumulative					
coloured_cumulative	50	3342	14	1	1	5	n/a	T6
Patterns		coloured_cumulative	[origin-1.duration-3.and-4.colour-1] [origin-2.duration-9.and-11.colour-2] [origin-3.duration-10.and-13.colour-1] [origin-6.duration-6.and-12.colour-1] [origin-7.duration-2.and-9.colour-3] 8					
Heuristics		coloured_cumulative	implies by cumulative					
cumulative_product	50	5457	8	1	1	5	n/a	T6
Patterns		cumulative_product	[origin-3.duration-1.and-4.height-1] [origin-9.duration-2.and-11.height-2] [origin-10.duration-3.and-13.height-1] [origin-6.duration-6.and-12.height-1] [origin-2.duration-7.and-9.height-3] 8					
coloured_cumulative	60	3342	14	1	1	6	n/a	T6
Patterns		coloured_cumulative	[origin-3.duration-1.and-4.colour-1] [origin-9.duration-2.and-11.colour-2] [origin-10.duration-3.and-13.colour-1] [origin-6.duration-6.and-12.colour-1] [origin-2.duration-7.and-9.colour-3] 8					
Heuristics		coloured_cumulative	implies by cumulative					

Transformation Results (Enlarged)

Constraint	Rank	Density	Links	UnTyp	ArgOrder	Crisp	Func	Tr	
cumulative	10	35561	34	1	1	1	n/a	T6	
Pattern:	<i>cumulative</i> ([[origin-1,duration-3,end-4,height-1] , [origin-2,duration-9,end-11,height-2] , [origin-3,duration-10,end-13,height-1] , [origin-6,duration-6,end-12,height-1] , [origin-7,duration-2,end-9,height-3]] , 8)								
Relations:	cumulative	implies	coloured_cumulative						
cumulative_product	20	5457	8	1	1	2	n/a	T6	
Pattern:	<i>cumulative_product</i> ([[origin-1,duration-3,end-4,height-1] , [origin-2,duration-9,end-11,height-2] , [origin-3,duration-10,end-13,height-1] , [origin-6,duration-6,end-12,height-1] , [origin-7,duration-2,end-9,height-3]] , 8)								
atmost_nvector	30	59684	11	1	2	3	n/a	T6	
Pattern:	<i>atmost_nvector</i> (8 , [[vec-[[var-1],[var-4],[var-3],[var-1]]] , [vec-[[var-2],[var-11],[var-9],[var-2]]] , [vec-[[var-1],[var-13],[var-10],[var-3]]] ;								



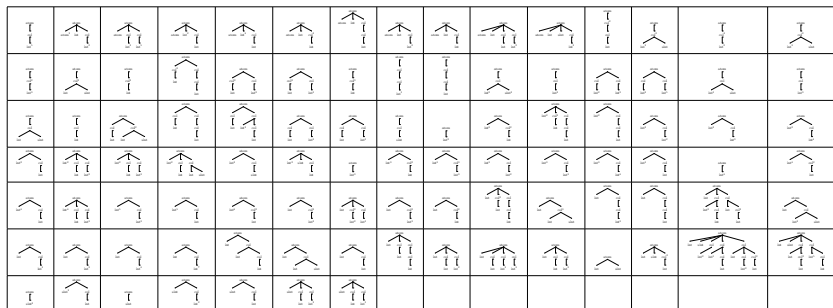
Outline

- 1 User Experience
- 2 How does it work?
 - Candidate Generation
 - Ranking
 - Instance Generation
- 3 Evaluation

Find Candidates from Sample Instances

- Find **compressed type** of sample
- **Select** all constraints with correct compressed type
- **Evaluate** samples (positive/negative)
- **Filter** candidates
- Also allow **transformations** of samples

All Signatures (97 from 350+ constraints)



Evaluator Types

- built-in provided by **SICStus** solver
- reformulation **rewritten** using other constraints
- automaton **automata with counters**
- logic **geost** rule language
- checker checks **ground instances** only
- none no evaluator given; no finite set solver in SICStus

Evaluator Stats

Evaluator	Nr Constraints
reformulation	137
automaton	49
reformulation + automaton	40
builtin	26
logic	9
builtin + automaton	8
checker	3
reformulation + reformulation	1
builtin + reformulation	1
none	80
Total	354

Filtering Candidates

- Candidate list may contain duplicates (depends on input)
- Some explanations are symmetrical
- Can be removed by **testing for symmetry definitions**
- Interaction with ranking (keep highest ranking)

Generating Candidates is not enough

- Queries can return **multiple candidates**
- Some are more meaningful than others
- User will not look up more than 2 (or perhaps 3) entries
- Present most likely candidate at top
- Include human expertise
- How to evaluate?

Ordering Criteria

rank **Composite value** derived from **structural information**

density **Pre-computed** number of solutions for small instances

links Number of **cross-references** to constraint in catalog

Structural Information

- **Functional dependency**
- **Crispness**
- **Argument reordering**
- **Typical restriction violations**

Typical Restrictions for $\text{atmost}(N, \text{VARIABLES}, \text{VALUE})$

- $N > 0$ (otherwise **value forbidden**)
- $N < |\text{VARIABLES}|$ (otherwise **no constraint**)
- $|\text{VARIABLES}| > 1$ (otherwise **unary**)
- $\text{atleast}(1, \text{VARIABLES}, \text{VALUE})$ (otherwise **unused value**)

Alternative Approaches to Density Computation

- Solution counting
 - Depends on background **theory on counting**
(*few constraints, sometimes #P-complete*)
- Complete enumeration
 - Only works for small collection/domain sizes
 - Tough to get completely right
- Random sampling
 - Can be hard to find any solutions

Use Cases

- Density computation
- Random samples
- Test sets for constraints
- Comparing reformulation and automata

Two Step Approach

- Sizing solver
- Variants of instance generator

Outline

- 1 User Experience
- 2 How does it work?
- 3 Evaluation

Counting Candidates

Nr	Catalog Examples			Gen. Ex.	Rep. Ex.	Single	Combined
	all	restricted	first				
1	66	63	63	155	140	96	107
2	35	28	28	48	60	59	45
3	27	25	20	23	24	33	29
4	32	31	28	11	13	22	27
5	26	24	22	8	9	11	11
6	30	30	32	6	5	12	11
7	15	15	16	1	2	7	10
8	11	9	9	2	2	7	7
9	12	12	12	1	-	4	3
10	7	7	7	-	-	2	3
11	7	7	7	-	-	-	-
12	2	2	2	-	-	-	-
13	2	2	2	-	-	-	-
14	-	-	7	-	-	2	lina 2

Quality of Ranking

Nr	Catalog Examples			Gen. Ex.	Rep. Ex.	Single	Combined
	all	restricted	first				
only	66	63	63	155	140	96	107
first	149	142	135	89	99	124	121
second	33	27	31	10	14	25	17
third	12	11	13	0	0	6	6
other	13	12	13	1	2	5	4
total	273	255	255	255	255	255	255

The Constraint Seeker as a CP Application

- Check positive and negative examples for satisfaction (on-line)
- Rank candidate lists by estimated relevance (on-line)
- Determine the argument order used for output coloring (on-line)
- Compute all call patterns up to given size (off-line)
- Count all (typical) solutions for small problem sizes (off-line)
- Sample solution space to generate positive (typical) instances (off-line)

Some Metrics

- SICStus Prolog + Apache server
- Generates HTML+CSS
- 12 years to collect the meta-data
- 3 man*months to develop code
- 6,400 lines of SICStus Prolog
- 2,000 lines of solver code
- 50,000 lines of meta-data for constraints

Points to Remember

- Set of tools for constraint catalog
- Search for constraints by example
- **Positive** or **negative** ground instances
- Produces **ranked list** of candidates
- Uses **catalog description** to build constraint models
- Also generates tests/counts solutions



Dealing with many different global constraints is not a problem, since meta data and meta programming help producing tools and automating their use!

CPAIOR 2012

CPAIOR 2012

Nantes, France
May 28 - June 1, 2012



*9th International Conference on Integration of Artificial Intelligence
and Operations Research Techniques in Constraint Programming
for Combinatorial Optimization Problems*