Modelling with global constraints

Global Constraint

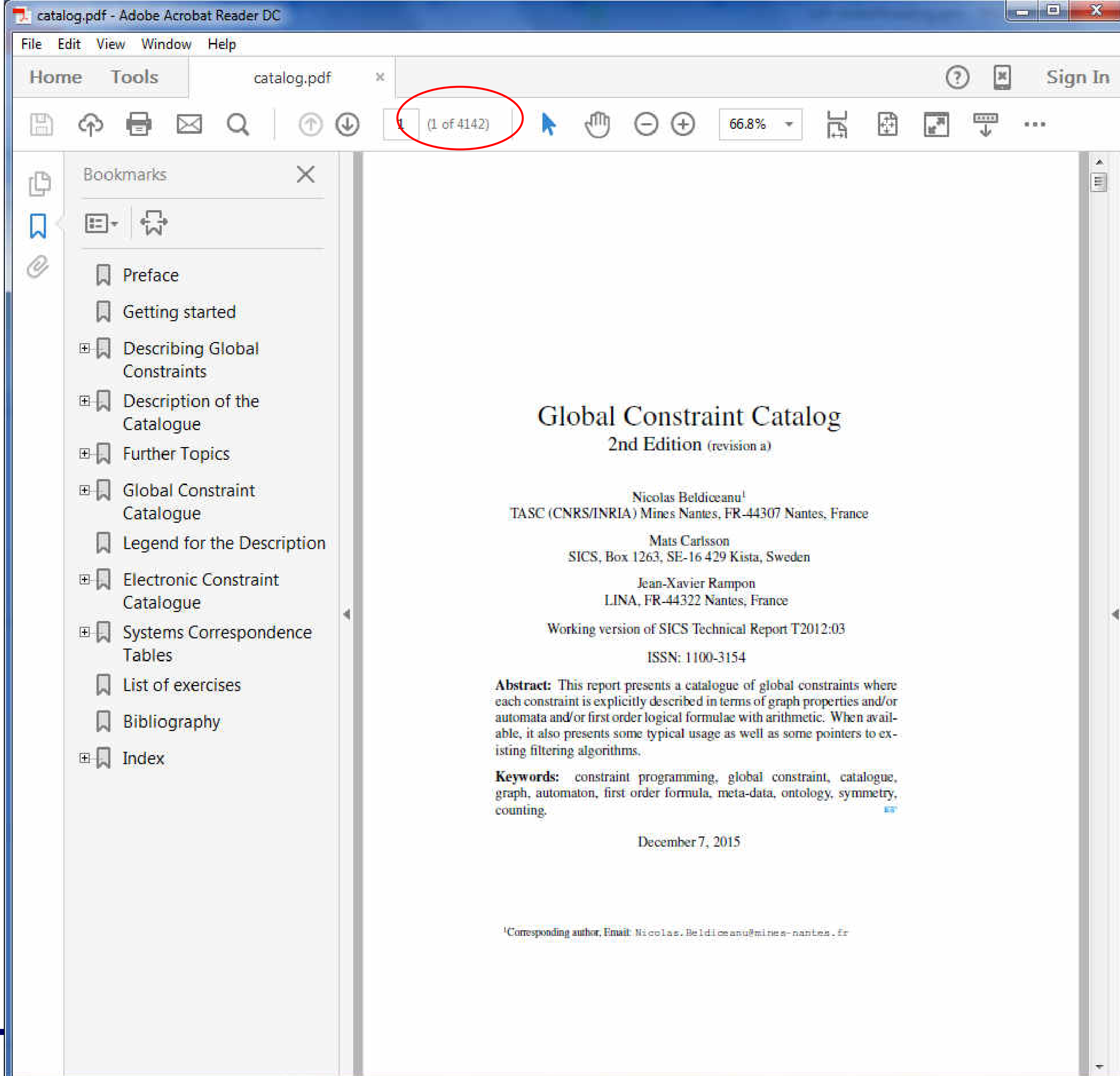
- a *global constraint* is a relation over an unlimited number of variables

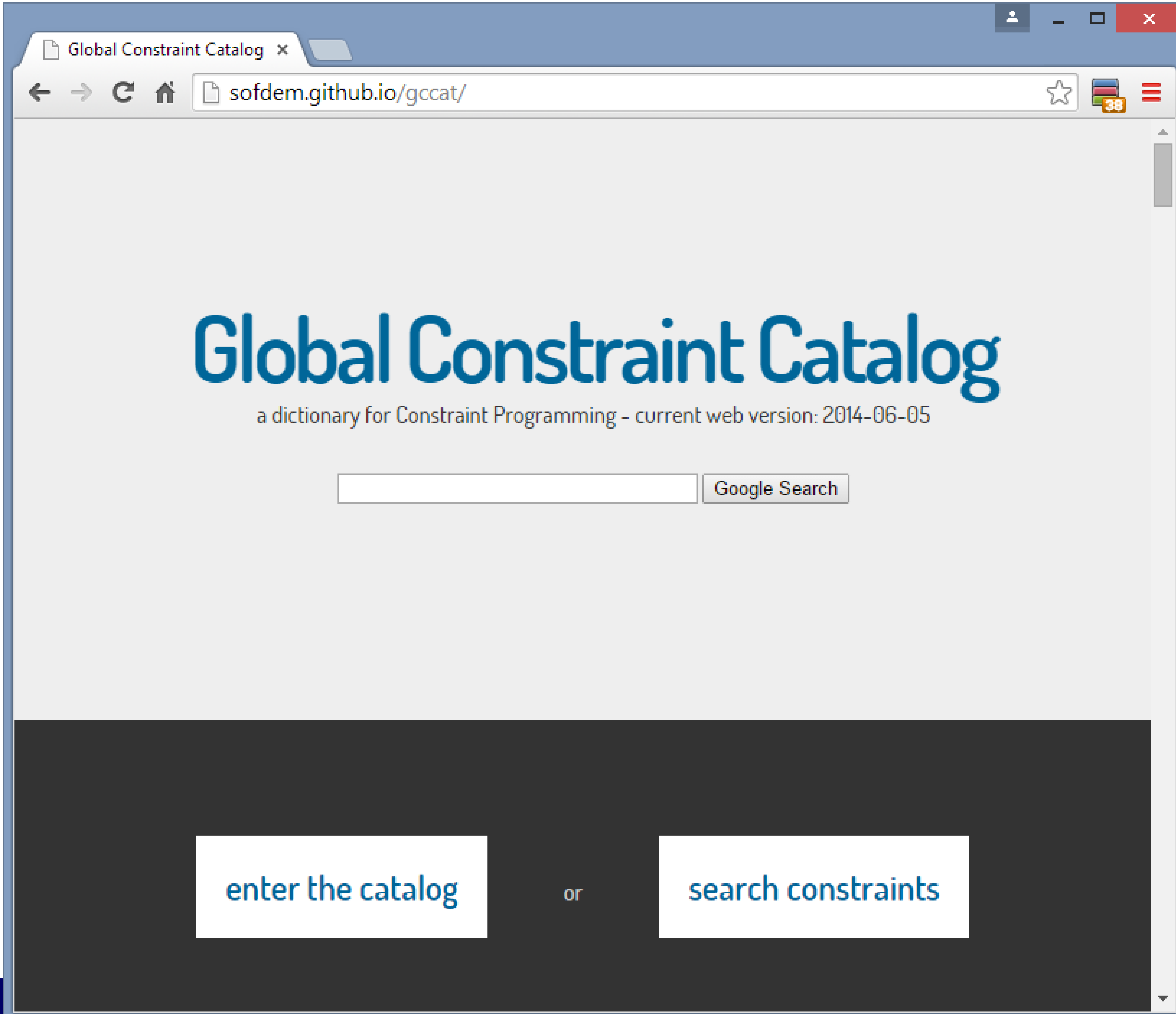
- a global constraint represents a meaningful sub-problem
  - known problems may have efficient algorithms to handle them

- Global constraints are the key to why constraint programming is successful:
  - they simplify problem modelling
  - they can speed up solving significantly

Global Constraints we have seen ...

- alldifferent (crystal maze)
- scalar(pin number, truck loading)
- sum (magic square)
- knapsack (truck loading)
- bin_packing
- cumulative (scheduling)


catalog.pdf - Adobe Acrobat Reader DC
File Edit View Window Help
Home Tools catalog.pdf
Sign In
1 (1 of 4142)
66.8%
Bookmarks
Preface
Getting started
Describing Global Constraints
Description of the Catalogue
Further Topics
Global Constraint Catalogue
Legend for the Description
Electronic Constraint Catalogue
Systems Correspondence Tables
List of exercises
Bibliography
Index
Global Constraint Catalog
2nd Edition (revision a)
Nicolas Beldiceanu[1]
TASC (CNRS/INRIA) Mines Nantes, FR-44307 Nantes, France
Mats Carlsson
SICS, Box 1263, SE-16 429 Kista, Sweden
Jean-Xavier Rampon
LINA, FR-44322 Nantes, France
Working version of SICS Technical Report T2012:03
ISSN: 1100-3154
Abstract: This report presents a catalogue of global constraints where each constraint is explicitly described in terms of graph properties and/or automata and/or first order logical formulae with arithmetic. When available, it also presents some typical usage as well as some pointers to existing filtering algorithms.
Keywords: constraint programming, global constraint, catalogue, graph, automaton, first order formula, meta-data, ontology, symmetry, counting.
December 7, 2015
[1]Corresponding author, Email: Nicolas.Beldiceanu@mines-nantes.fr


Global Constraint Catalog
sofdem.github.io/gccat/
Global Constraint Catalog
a dictionary for Constraint Programming - current web version: 2014-06-05
Google Search
enter the catalog
or
search constraints

The Global Constraint Catalogue

- lists all global constraints implemented in one or more solvers, or described in research papers

- currently over 400 listed

- each one described in a standard format
  - underlying theory
  - sometimes a description of the algorithm underneath
  - examples of its use
  - a list of which solvers include it
    - (but references to Choco appear to be out of date)

count(int VALUE, IntVar[] VARS, IntVar LIMIT)

Exactly LIMIT of the variables in VARS are assigned VALUE

count(3, [X1:{3}, X2:{5}, X3:{3}, X4:{1}], 2)
is satisfied, since exactly 2 of the variables have value 3

count(3, [X1:{3}, X2{5}, X3:{1}, X4:{1}], 2)

is violated, since only 1 of the variables has value 3

count(3, [X1:{3}, X2:{4,5,6}, X3:{1,2}, X4: {1,5,7}], 2)

is violated, since only 1 of the vars has the value 3 in its domain

count(3, [X1:{3}, X2:{4,5,6}, X3:{1,2}, X4:{1,5,7}], {2,3,4})

is violated, since only 1 of the vars has the value 3 in its domain, and so none of 2, 3 or 4 can be reached

global_cardinality(IntVar[] VARS, int[] VALUES,
IntVar[] OCCUR, boolean CLOSED)

For each *i*, the number of times VALUES[i] is assigned to variables in VARS should be equal to OCCUR[i]

- VALUES and OCCUR must be the same length
- If CLOSED is true, then the domains of VARS are restricted to the values in VALUES

global_cardinality([X1:{1}, X2:{3}, X3:{1}, X4:{2}],
[1,2,3], [2,1,1], true)

is satisfied, since '1' appears twice, '2' appears once, and '3' appears once.

Note: if VALUES contains all values in the domains of the variables in VARS, and OCCUR is an array of IntVars fixed to the value '1', then

global_cardinality(VARS, VALUES, OCCUR, true)
is the same as
alldifferent(VARS)

Note: if VALUES is an array of length 1, then
global_cardinality(VARS, VALUES, OCCUR, true)
is the same as
count(VALUES[0], VARS, OCCUR[0])

element(IntVar VALUE, int[] TABLE, IntVar INDEX)

element(IntVar VALUE, IntVar[] TABLE, IntVar INDEX, int OFF)

VALUE = TABLE[INDEX]
VALUE = TABLE[INDEX-OFF]
(in Choco, the index of an array starts at 0)

element(7, [3,7,2,7,8,1], 3) is satisfied, since TABLE[3] == 7

element(7, [3,7,2,7,8,1], {1,2,3}) is satisfiable, since TABLE[1] == 7 and TABLE[3] == 7

element(7, [3,7,2,7,8,1], 4) is violated, since TABLE[4] =/= 7

element(7, [3,7,2,7,8,1], 4, 1) is satisfied, since TABLE[4-1] == 7

regular(IntVar[] VARS, IAutomaton AUTO)

The sequence of values taken by the variables in VARS must be a word accepted by the finite state automaton AUTO (for example, AUTO could be given as a regular expression)

regular([2,4,4,5], "((2|3)(4*)5)") is satisfied,
since 2445 is accepted by the regular expression

regular([2,3,4,5], "((2|3)(4*)5)") is violated,
since 2345 is not accepted (can only have 2 or 3 at the start, not both in sequence)

Modeling example: Warehouse Location

A retail outlet supplies its shops from a number of central warehouses. Each shop must be supplied by only one warehouse, but a warehouse can supply many shops.

For each (warehouse,shop) pair, there is a supply cost based on the travel distance between them. There is a fixed cost to maintain a warehouse. Each warehouse has a maximum number of shops it can supply.

Find the subset of warehouses that minimises the total cost while supplying all the shops.

Example

E.g. if there are 3 warehouses, and 4 shops, such that the supply costs are:

	w0	w1	w2
s0	4	6	8
s1	7	7	5
s2	4	5	6
s3	9	6	5

and the warehouse maintenance cost is 5

and the warehouse capacities are 2,4,3

then opening warehouses 0 and 2, and supplying shops 0:0,2 2:1,3 gives the minimum cost of 28

The model

shopSupply[] has a cell for each shop, assigning a warehouse
whActive[] has a cell for each warehouse, = 1 if active
supplyCount[] has a cell for each wh, giving the number of shops it supplies
supply[][] is the array of supply costs for each shop, each warehouse
shopCost has a cell for each shop, with the cost of its supply

For each shop,   whActive[shopSupply[s]] == 1
i.e. *element*(1, whActive, shopSupply[s])

For each wh, *count*(wh, shopSupply, supplyCount[wh])

For each shop, *element*(shopCost[s], supply[s], shopSupply[s])

count(1, whActive, numberActive)

Add all shop costs and numberActive*maintenance

Global Constraints: summary

- global constraints can make constraint programming very efficient for a wide range of problems
- being successful as a constraint modeller requires you to understand the global constraints, and use them appropriately
- new global constraints are being developed every year

Next lecture ...

Symmetry