

Global Constraints: Graph-Based Representation

cumulatives

global cardinality

graph_crossing

binary_tree

alldifferent_same_value

bin-packing

polyonimo

cardinality_path

connect_points

tree

Nicolas Beldiceanu

EMN, LINA

Alfred Kastler

44000 Nantes, FRANCE

Nicolas.Beldiceanu@emn.fr

element

cumulative

soft_alldifferent
nvalue

same

map

orchad

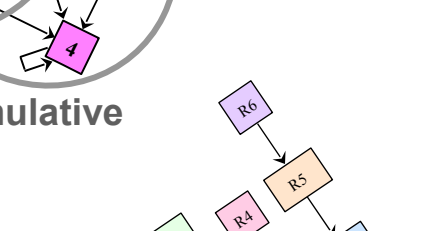
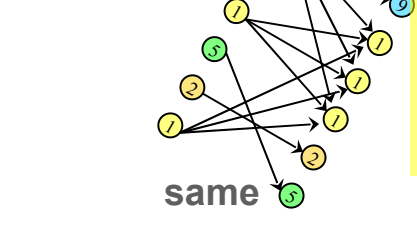
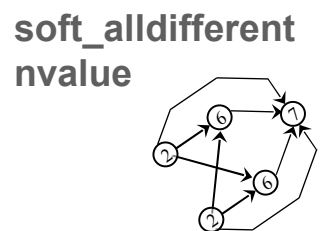
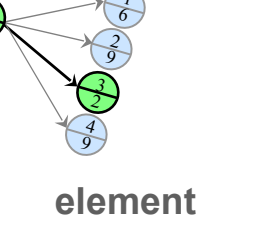
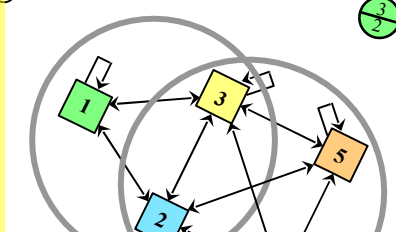
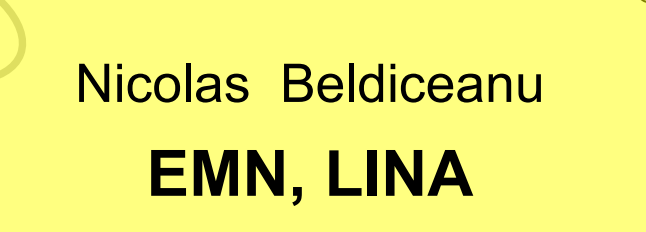
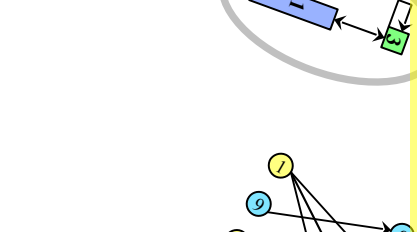
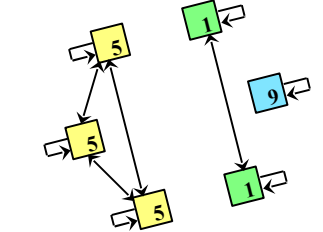
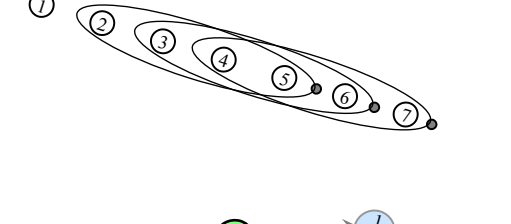
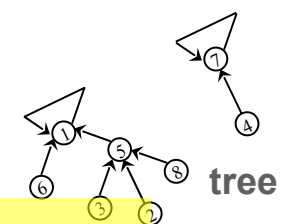
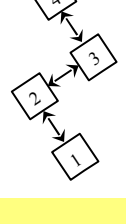
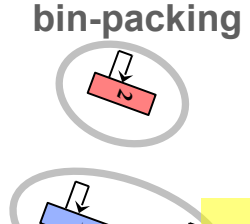
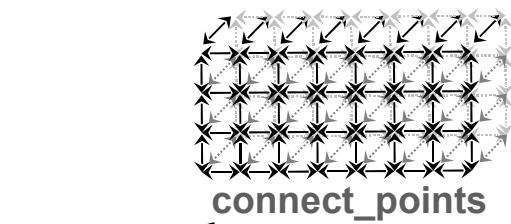
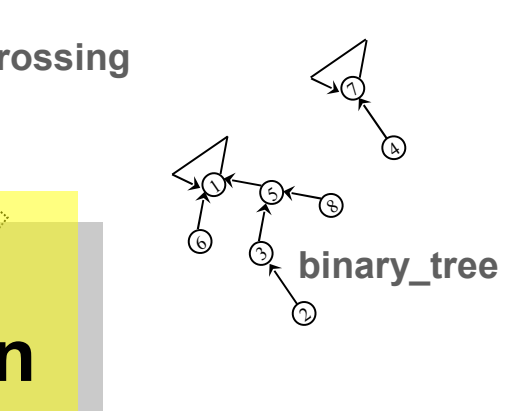
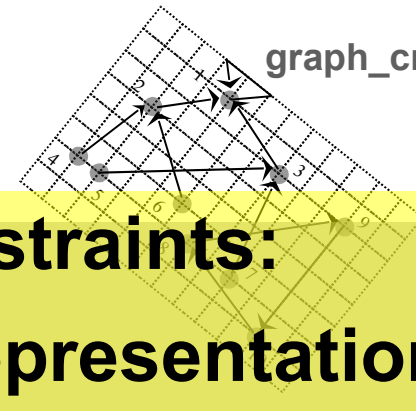
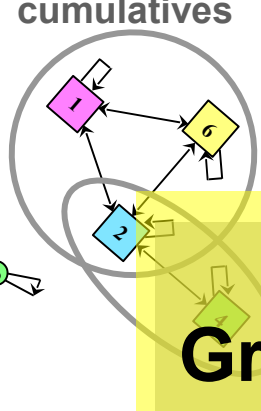
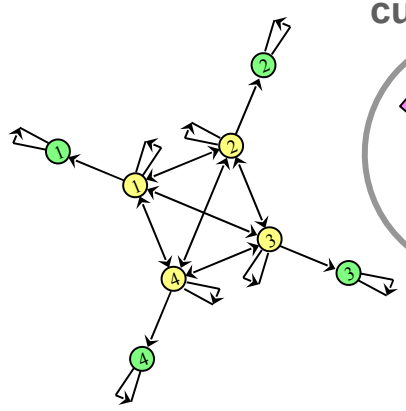
place_in_pyramid

minimum

group

symmetric_alldiff

Samos Summer



Goal of the Course

- Overview of **different ways** to look at global constraints,
- Present the **graph-based representation** (declarative/procedural).

The topic of graph-based representation is an ongoing work in collaboration with Mats Carlsson, Sophie Demasse, Thierry Petit and Jean-Xavier Rampon.

Outline of the Course

PREREQUISITE

GENERAL INTRODUCTION

GRAPH BASED DESCRIPTION

CONCLUSION

PREREQUISITE



- **Constraint program**
- Constraint kernel
- Filtering algorithm

GENERAL INTRODUCTION

GRAPH BASED DESCRIPTION

CONCLUSION

The **Three Parts** of a Typical Constraint Program

$[X1, X2, X3]$ in 1..4

$X1 + X2 < X3$

labeling($[X1, X2, X3]$)

Part 1: the variables

[X1,X2,X3] in 1..4

domain declaration before stating any constraint
(what are the variables, what are the values ?)

$X1 + X2 < X3$

labeling([X1,X2,X3])

X1:1..4
X2:1..4
X3:1..4

Part 2: the constraints

$[X1, X2, X3]$ in 1..4

$X1 + X2 < X3$

labeling($[X1, X2, X3]$)

setting up a constraint
(how to express a given condition ?)

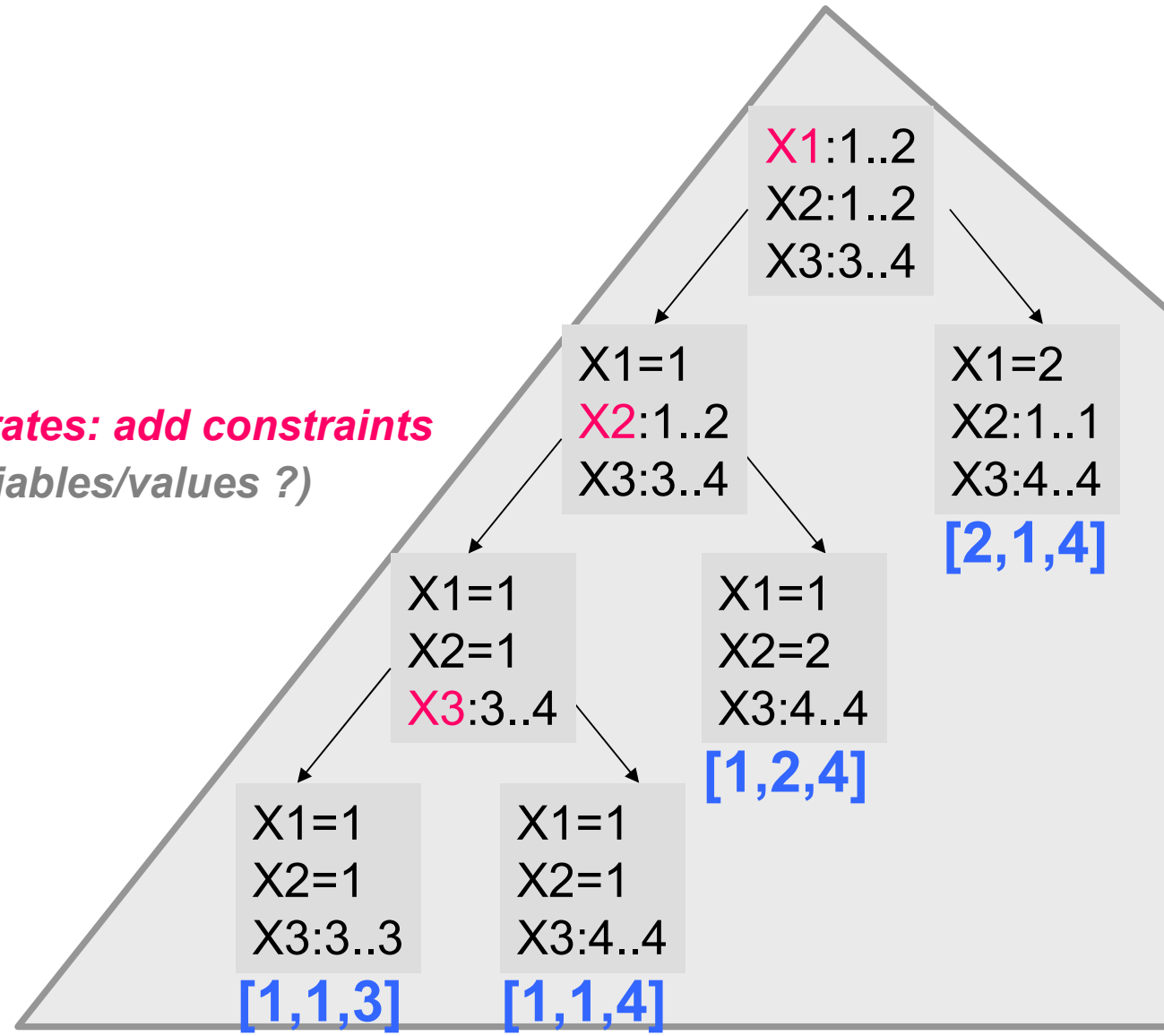
X1:1..**2**
X2:1..**2**
X3:**3**..4

Part 3: the enumeration

$[X1, X2, X3]$ in $1..4$

$X1 + X2 < X3$

labeling([X1,X2,X3]) *enumerates: add constraints*
(in which order to assign variables/values ?)



PREREQUISITE

- Constraint program
- **Constraint kernel**
- Filtering algorithm



GENERAL INTRODUCTION

TEN VIEWS OF GLOBAL CONSTRAINTS

GRAPH BASED DESCRIPTION

CONCLUSION

Operational View of a Constraint

Piece of code (usually polynomial algorithm) used as a **coroutine** for

(1) Checking if a condition is **for sure false**

⇒ **backtrack**

(2) Checking if a condition is **for sure true**

⇒ **entailment**

Communication channel between
two constraints sharing variables

(3) **Removing** values from variables which

lead the condition to be for sure false

⇒ **suspend** : until all constraints are not entailed you

are **not sure** if there exist a solution

⇒ you **must** assign values to **all** variables

- *allows to backtrack as soon as possible,*
- *guides search heuristics.*

Knowledge About the Constraint Kernel

- **Domain variables**
- **Context of a filtering algorithm**
- **Services of the kernel**

Domain variables (low level)

Filtering algorithms need:

- low level queries to domain variables:
 - get the minimum or maximum value
 - check if a given value belongs or not to a domain
 - check if two domain variables are the same or not
- low level modifications of domain variables:
 - remove a value
 - adjust the minimum or maximum value
 - fix to a value

Domain variables (continued)

Filtering algorithms sometimes need:

- higher level queries to domain variables:
 - get the next/previous value in a domain,
 - check if a domain intersects / is included in a given set of values,
 - check if two domain variables are the same or not.
- modifications of domain variables:
 - remove an interval of values,
 - remove all values in/not in a given set of values,
 - unify two domain variables (make them equal once for all).

Domain variables

In order to design efficient filtering algorithms you should know :

- **which** queries are available,
- their respective **complexity**.

Context of a filtering algorithm

Reason for a call :

- Creation of the constraint
- Specific domain modification on a given variable
- Domain modification within a set of variables

Additional information (for optimization purpose):

- No backtrack since last call to that constraint, ...

Different levels of granularity

- (1) One propagator for each variable of the constraint
- (2) Only one propagator for all the variables of the constraint

Services of the kernel

- Scanning the parameters of the constraint
 - Adding a new propagator to the system (with a waking condition)
 - Memory allocation
 - Memory management for backtracking
-
- Creating a new constraint (rewriting)
 - Making some hypothesis and calling recursively the propagation

PREREQUISITE

- Constraint program
- Constraint kernel
- **Filtering algorithm**



GENERAL INTRODUCTION

GRAPH BASED DESCRIPTION

CONCLUSION

An Illustrative Example: the *cycle* Constraint

Design a basic filtering algorithm for handling the following constraint:

$\text{cycle}(N, [S1, \dots, Sn])$ N is the **number of cycles of the permutation** $S1, \dots, Sn$
 N and $S1, \dots, Sn$ are domain variables

**EXAMPLES OF
SOLUTIONS :**

$\text{cycle}(1, [2, 3, 4, 1])$
 $\text{cycle}(2, [2, 1, 4, 3])$
 $\text{cycle}(4, [1, 2, 3, 4])$

State of the *cycle* Constraint

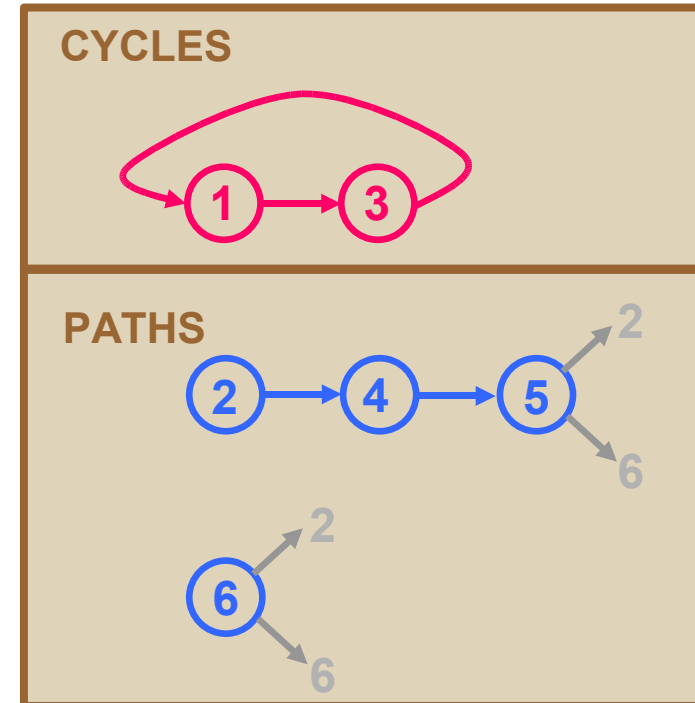
State can be represented by :

- (1) The **cycles (closed)** which were already built,
- (2) The **paths (partial)** that were constructed.

ILLUSTRATION

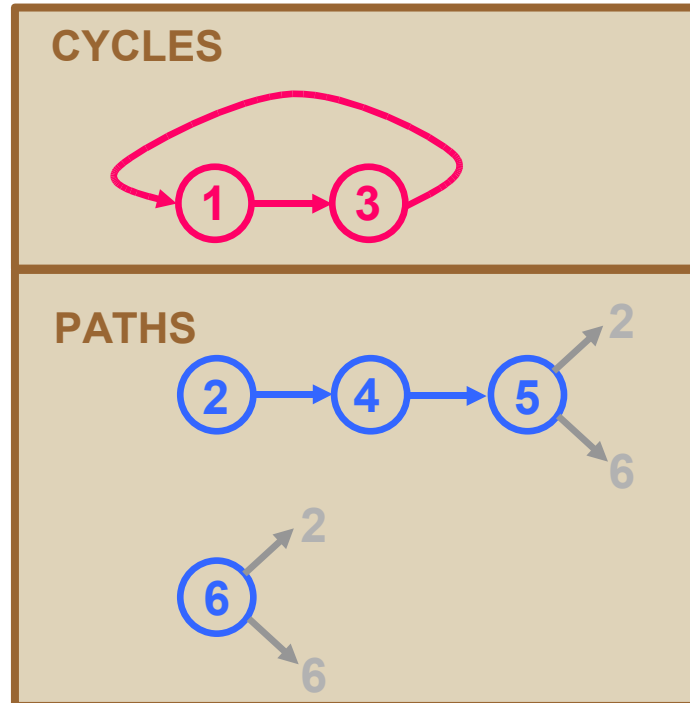
S1=3 S2=4 S3=1 S4=5 S5 in {2,6} S6 in {2,6}

cycle(N,[S1,S2,S3,S4,S5,S6])



Invariants Associated to the State

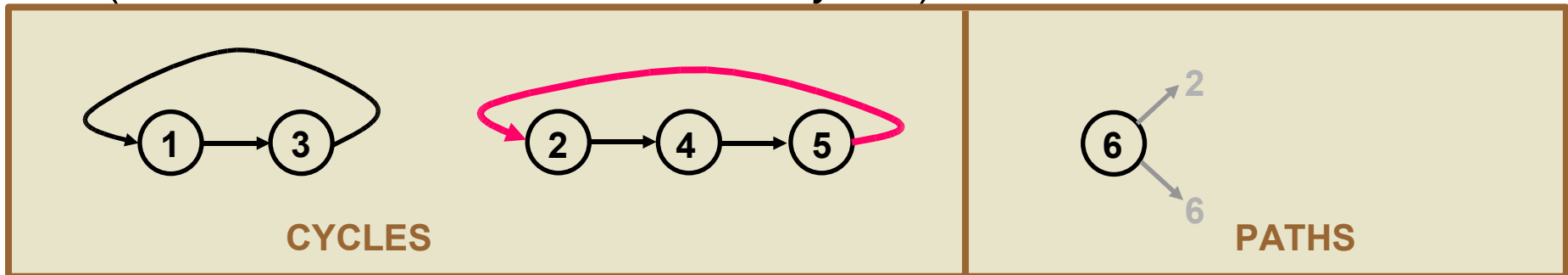
- (1) The potential successors of an end of path **are only path starts** (*alldifferent*)
- (2) An end of path **has more than one successor** (*else has to merge paths*)



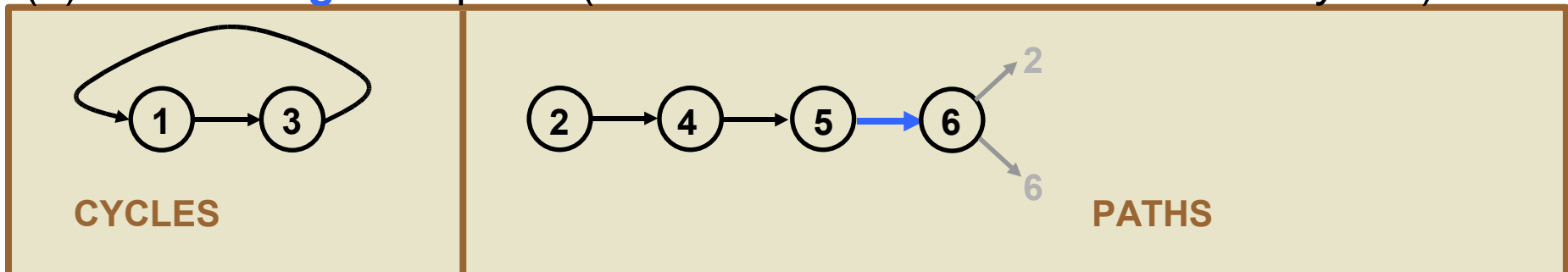
Updating the State of cycle

Each time we fix a variable S_i to j :

- (1) We either **close** a path and create a new cycle
(*increase the minimum number of cycles*)



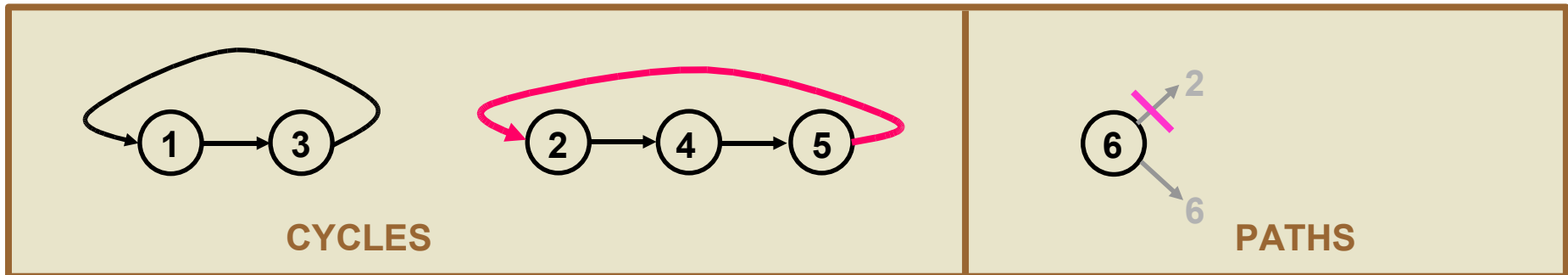
- (2) Or we **merge** two paths (*decrease the maximum number of cycles*)



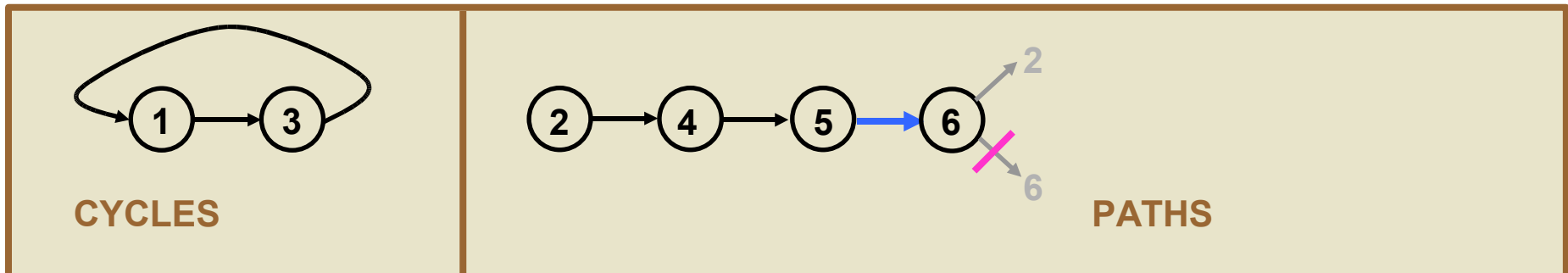
Maintaining the Invariant

Each time we fix a variable S_i to j :

(1) We either **close** a path and create a new cycle **and maintain the invariant**



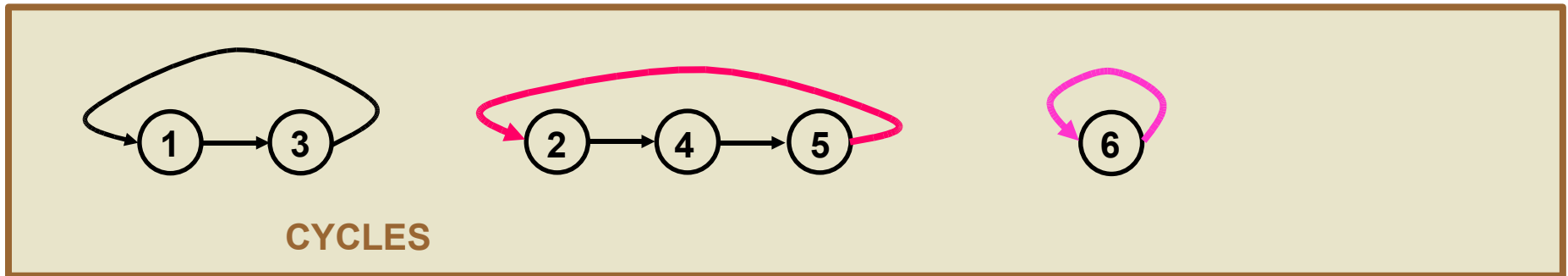
(2) Or we **merge** two paths **and maintain the invariant**



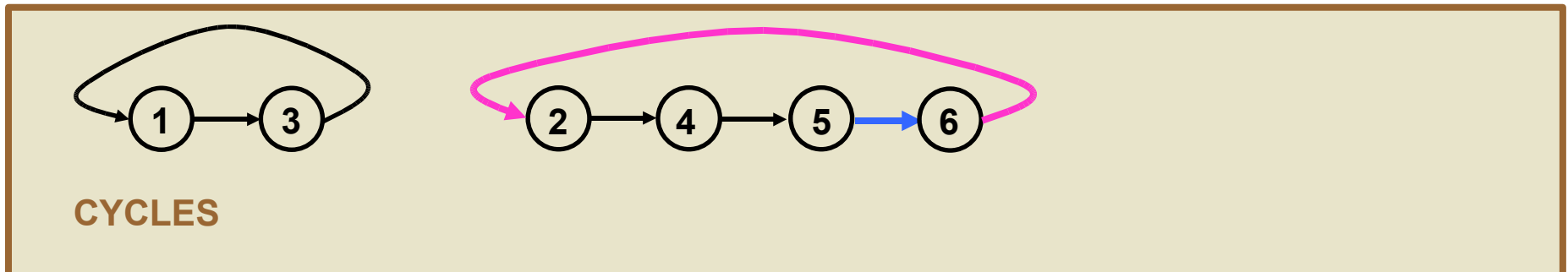
Maintaining the Invariant

Each time we fix a variable S_i to j :

(1) We either **close** a path and create a new cycle **and maintain the invariant**



(2) Or we **merge** two paths **and maintain the invariant**



Computing **Bounds** for N

RULE 1 : each time we **close a cycle** we should update
the **minimum of N** to:
the number of cycles

RULE 2 : each time we **merge two paths** we should update
the **maximum of N** to:
the number of cycles + the number of paths

From the Bounds of N to the Successor Variables

RULE 3 : **if** **$\max(N) = \text{number of cycles} + 1$**
 and more than one path (*allow to close last path*)
 then *forbid to close all the paths*

RULE 4 : **if** **$\min(N) = \text{number of cycles} + \text{number of paths}$**
 then *force to close all the paths*

More about *cycle*

Question: think about additional pruning methods for *cycle*.

Typical Filtering Algorithm

constraint(**Var**, **Variables**)

FILTERING ALGORITHMS

find **BOUNDS**
for **Var**

Lower bound LB

Upper bound UB

PROPAGATE from bounds
of **Var** to **Variables**

Remove *val* from *var* iff:
 $LB + \text{lower_regret}(var, val) > \max(\mathbf{Var})$

Remove *val* from *var* iff:
 $UB - \text{upper_regret}(var, val) < \min(\mathbf{Var})$

Typical Filtering Algorithm

constraint(**Var**, **Variables**)

FILTERING ALGORITHMS

find **BOUNDS**
for **Var**

Lower bound LB

Upper bound UB

PROPAGATE from bounds
of **Var** to **Variables**

Remove *val* from *var* iff:
 $LB + \text{lower_regret}(var, val) > \max(\mathbf{Var})$

Remove *val* from *var* iff:
 $UB - \text{upper_regret}(var, val) < \min(\mathbf{Var})$

Focus first on invariants

Structure of a Typical Filtering Algorithm

constraint(**Var**, **Variables**)

FILTERING ALGORITHMS

find **BOUNDS**
for **Var**

Lower bound LB

Upper bound UB

Focus first on invariants

PROPAGATE from bounds
of **Var** to **Variables**

Remove *val* from *var* iff:
 $LB + \text{lower_regret}(var, val) > \max(\mathbf{Var})$

Remove *val* from *var* iff:
 $UB - \text{upper_regret}(var, val) < \min(\mathbf{Var})$

and then on how to enforce these invariants

PREREQUISITE

GENERAL INTRODUCTION



- What is a global constraint?
- A Brief History
- Current Status
- Reccuring debates
- So many global constraints?
- Types of global constraints

GRAPH BASED DESCRIPTION

CONCLUSION

Separate **Definition** from Usage

Because a same definition can have **different** usages [Pitrat 93]

DEFINING GLOBAL CONSTRAINTS

Useful for some purpose

Not an automaton of exponential size

An **expressive** and **concise** condition
involving a **non-fixed** number of variables

Don't go directly to the usage (**and skip the definition**)

Separate Definition from Usage

POTENTIAL USAGES

- **filtering** algorithms for constraint solving,
- generation of **explanations**,
- **visualization** algorithms for constraint debugging,
- generation of **cuts** for linear programming,
- incremental **moves** for local search.

main usage



Remark about Constraints Definitions

Allways gives the defintion of a constraint according to the **ground case**.

Provide a checker (polynomial) which takes a ground instance and returns:

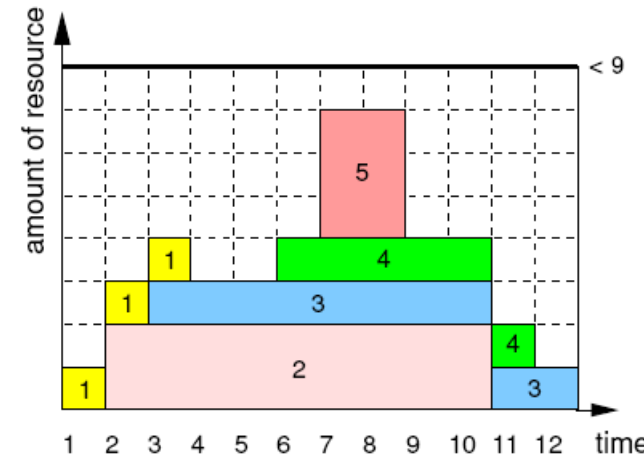
- yes if the constraint holds,
- no otherwise.

Example: *cumulative* Constraint

Given a **limit** and a set of tasks,
where each task t has an **origin** $[t]$, an **end** $[t]$ and a **height** $[t]$:

$$\forall i: \sum_{t: \text{origin}[t] \leq i < \text{end}[t]} \text{height}[t] \leq \text{limit}$$

$$\text{cumulative} \left(\left\{ \begin{array}{llll} \text{origin} - 1 & \text{duration} - 3 & \text{end} - 4 & \text{height} - 1, \\ \text{origin} - 2 & \text{duration} - 9 & \text{end} - 11 & \text{height} - 2, \\ \text{origin} - 3 & \text{duration} - 10 & \text{end} - 13 & \text{height} - 1, \\ \text{origin} - 6 & \text{duration} - 6 & \text{end} - 12 & \text{height} - 1, \\ \text{origin} - 7 & \text{duration} - 2 & \text{end} - 9 & \text{height} - 3 \end{array} \right\}, 8 \right)$$



Not Just Filtering !

In many real-world applications **the solving process is left to the human**.

In this context you **only** need for each global constraint:

- a visualizer that shows in an appropriate way a ground instance and points out the violation (if it exists) in a specific way.

And for most global constraints you potentially can derive a graphical representation from the definition of the constraint (*for instance you represent the corresponding graph and outline the properties you are looking for*).

*In the context of cumulative you want to visualize the cumulative profile and identify the contribution of each task within this profile.
(need to specialize the visualisation according to specific graph class)*

Not Just Filtering !

In the context of local search, given a ground instance of a global constraint you are interested to **quantify the violation**.

And again, for some global constraints the violation cost can be directly derived from the description of the constraint
(for instance in the case of **decomposition-based violation measure**).

In the context of cumulative you could consider the surface over the limit.

Nature of Globality in the Context of Filtering

[Bessière, Van Hentenryck]

- Globality from a **semantic** point of view
- Globality from a **deductive** point of view
- Globality from a **operational** point of view

Semantic Globality

A constraint **can't be expressed** as a conjunction of elementary constraints (**without** introducing new variables).

EXAMPLE: the *period* constraint

$\text{period}(4, [2, 1, 4, 1, 2, 1, 4, 1, 2, 1, 4])$

COUNTER-EXAMPLE: the *alldifferent* constraint

$\text{alldifferent}([2, 1, 4])$

$2 \neq 1 \wedge 2 \neq 4 \wedge 1 \neq 4$

Deductive Globality

A constraint can be expressed as a conjunction of elementary constraints but this leads to **weak pruning**.

EXAMPLE: the *alldifferent* constraint

$v_1, v_2, v_3 \text{ in } 1..2, \text{ alldifferent}([v_1, v_2, v_3]) \Rightarrow \text{no}$

$v_1, v_2, v_3 \text{ in } 1..2, v_1 \neq v_2 \wedge v_1 \neq v_3 \wedge v_2 \neq v_3 \Rightarrow \text{maybe}$

The *alldifferent* constraint is global from a deductive point of view.

Operational Globality

Expressing a constraint as a conjunction of elementary constraints does not lead to weak pruning but leads to **poor performance** (time, memory).

EXAMPLE: a set of inequalities

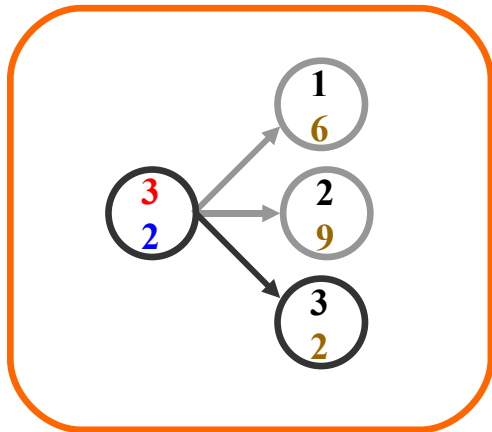
inequalities([**V1-V2**, **V2-V3**, **V3-V4**, **V4-V1**]) \Rightarrow no

$$\left\{ \begin{array}{l} \text{V1} < \text{V2} \wedge \\ \text{V2} < \text{V3} \wedge \\ \text{V3} < \text{V4} \wedge \\ \text{V4} < \text{V1} \end{array} \right.$$

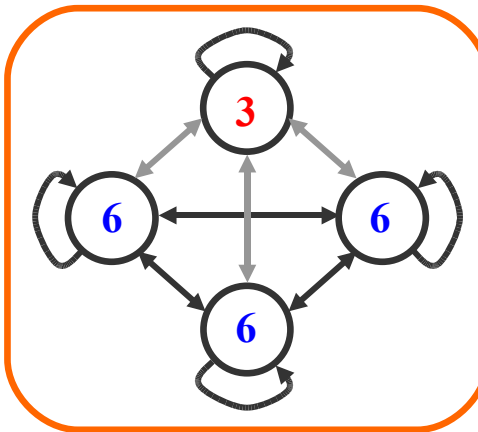
\Rightarrow no, but complexity depends of the **sizes of the domains** (and not of the number of variables)

Declarative Aspect

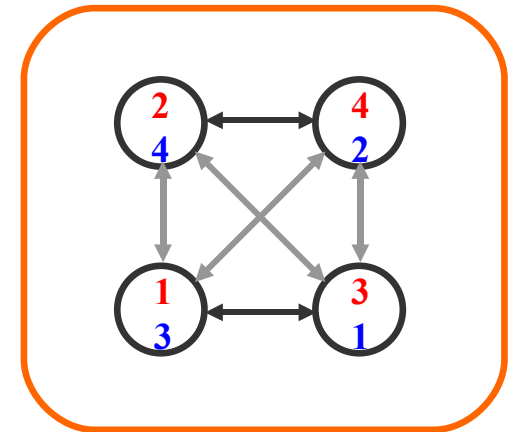
GOAL: Identify common structures in discrete combinatorial problems.



`element(3, {6, 9, 2}, 2)`



`nvalue(2, {6, 3, 6, 6})`



`symetric_alldiff({3, 4, 1, 2})`

Declarative Aspect: Key Ideas

- A common structure is **not an entire problem** !
⇒ first difference from catalog of problems.
*(allows to come up with **components** which can be **reused** across different problems)*
- There should be an **explicit description** of these structures.
⇒ second difference from catalog of problems.
*(since different programs [that consider a **specific usage**] will exploit these structures describing things with **natural language is useless**)*

Declarative Aspect: Describe your Problem

A problem should be **described** as
a combination of common structures
regardless of the solving paradigm (CP,LP,LS)

***n*-queens:**

- ***n*** variables,
- **three** *alldifferent* constraints.

Warehouse location:

- ***c+3*** variables,
- **one** *gcc* constraint,
- **one** *swdv* constraint.

Traveling salesman:

- ***c+1*** variables,
- **one** *circuit* constraint,
- **one** *mwa* constraint.

Declarative Aspect: Describe your Problem

n-queens:

- *n* variables,
- **three** *alldifferent* constraints.

Warehouse location:

- **c+3** variables,
- **one** *gcc* constraint,
- **one** *swdv* constraint.

Traveling salesman:

- **c+1** variables,
- **one** *circuit* constraint,
- **one** *mwa* constraint.

```
sum_of_weights_of_distinct_values(VARIABLES, VALUES, COST)
```

```
VARIABLES : collection(var - dvar)
VALUES     : collection(val - int, weight - int)
COST       : dvar
```

```
sum_of_weights_of_distinct_values ( ( { var - 1,
                                         var - 6,
                                         var - 1
                                         } ,
                                       { val - 1 weight - 5,
                                         val - 2 weight - 3,
                                         val - 6 weight - 7
                                       } ) ) 12
```

n-queens:

- *n* variables,
- **three** *alldifferent* constraints.

Warehouse location:

- **c+3** variables,
- **one** *gcc* constraint,
- **one** *swdv* constraint.

Traveling salesman:

- **c+1** variables,
- **one** *circuit* constraint,
- **one** *mwa* constraint.

```
minimum_weight_alldifferent(VARIABLES, MATRIX, COST)
```

```
VARIABLES : collection(var - dvar)
MATRIX    : collection(i - int, j - int, c - int)
COST      : dvar
```

```
minimum_weight_alldifferent
```

$$\left(\begin{array}{l} \left\{ \begin{array}{l} \text{var} - 2, \\ \text{var} - 3, \\ \text{var} - 1, \\ \text{var} - 4 \end{array} \right\}, \\ \left\{ \begin{array}{lll} i - 1 & j - 1 & c - 4, \\ i - 1 & j - 2 & c - 1, \\ i - 1 & j - 3 & c - 7, \\ i - 1 & j - 4 & c - 0, \\ i - 2 & j - 1 & c - 1, \\ i - 2 & j - 2 & c - 0, \\ i - 2 & j - 3 & c - 8, \\ i - 2 & j - 4 & c - 2, \\ i - 3 & j - 1 & c - 3, \\ i - 3 & j - 2 & c - 2, \\ i - 3 & j - 3 & c - 1, \\ i - 3 & j - 4 & c - 6, \\ i - 4 & j - 1 & c - 0, \\ i - 4 & j - 2 & c - 0, \\ i - 4 & j - 3 & c - 6, \\ i - 4 & j - 4 & c - 5 \end{array} \right\}, \end{array} \right) 17$$

A Short Overview of the Area: Procedural Aspect

Produce **efficient** algorithms within **various** contexts:

- **filtering** algorithms for constraint solving,
- generation of **explanations**,
- **visualization** algorithms for constraint debugging,
- generation of **cuts** for linear programming,
- incremental **moves** for local search.

Try to use the description of constraints to produce these algorithms (**PROCEDURALIZE the declarative representation**)

Procedural Aspect: Key Idea

- Don't reformulate the problem in a **unique formalism** (*k-sat, linear programming, ...*) with a general purpose solver,

BUT RATHER

- Describe your problem in terms of **common structures** for which you have the best known algorithms.

Within a unique formalism you will **try to catch** some problem structure soon or later !

PREREQUISITE

GENERAL INTRODUCTION

- What is a global constraint?
- **A Brief History**
- Current Status
- Reccuring debates
- So many global constraints?
- Types of global constraints

GRAPH BASED DESCRIPTION

CONCLUSION

A Brief History: Key Phases

1976 (pioneer)	<ul style="list-style-type: none"> ALICE J.-L. Laurière 	<i>alldifferent, circuit, tree</i>
1987-1990 (research-lab)	<ul style="list-style-type: none"> CHIP 	<i>element, atleast, atmost, between, lex_ordering, diff_pair, ...</i>
1990-1992 (industry)	<ul style="list-style-type: none"> Bull, Cosytec, Ilog 	<i>cumulative, cycle, diffn, distribute, ...</i>
1993-2006 (academy)	<ul style="list-style-type: none"> BProlog, CHOCO, <i>increasing, global cardinality,</i> ECLAIR, ECLiPSe, <i>sequence, cumulatives, ...</i> FaCiLe, Figaro, Gecode, ICEBERG, IF/Prolog, Mozart, SICStus, ... 	

A Brief History: Key Points

1988	• Non-numerical constraint	<i>element: warehouse location</i>
1990	• Global constraint	<i>diff_pair: orthogonal latin squares</i>
1993	• Encapsulating OR	<i>scheduling, routing, metaheuristics</i>
2000	• Classification	<i>explicit description of global constraints</i>
2001	• Combinatorial structures for local search	<i>incremental moves for global constraints</i>
2007	• Convergence between CP and OR ?	

PREREQUISITE

GENERAL INTRODUCTION

- What is a global constraint?
- A Brief History
- ➔ • **Current Status**
- Reccuring debates
- So many global constraints?

GRAPH BASED DESCRIPTION

CONCLUSION

Current status

INDUSTRY

- **Librairies**

(Artelys-Dash, Bouygues, Cosytec, Ilog, Koalog, Thalès, ...)

- **Application oriented**

(scheduling, logistics, configuration)

- **Reused in other contexts**

(preprocessing in linear programming)

ACADEMY

- **Librairies**

(Choco, Gecode, ...)

- **Convergence between CP and OR**

- **Hybridation**

PREREQUISITE

GENERAL INTRODUCTION

- What is a global constraint?
- A Brief History
- Current Status
- ➔ • **Reccuring debates**
- So many global constraints?
- Types of global constraints

GRAPH BASED DESCRIPTION

CONCLUSION

Reccuring Debates (1975)

- **Theoretical computer science versus AI (and constraint) ?**

Marcel-Paul Schützenberger - Jean-Louis Laurière


“Fuyons les théories qui n'ont point de théorèmes”

Reccuring Debates (1975)

- **Theoretical computer science versus AI (and constraint) ?**

Marcel-Paul Schützenberger - Jean-Louis Laurière

“Fuyons les théories qui n'ont point de théorèmes”

- 
- **Constraint propagation**
 - **Branching**
 - **Greedy Heuristics**
 - **Meta-heuristics**

Reccuring Debates (1975)

- **Theoretical computer science versus AI (and constraint) ?**

Marcel-Paul Schützenberger - Jean-Louis Laurière

“Fuyons les théories qui n'ont point de théorèmes”

- ◆ Automata is a core topic of computer science, but not AI.
- ◆ Well, 30 years later, see work about automata+constraints:
Amilhastre, Beldiceanu, Carlsson, van Hoeve, Pesant, Walsh,

Reccuring Debates (1976)

- **Constraints versus algorithms ?**

Since the goal of ALICE was to show that general systems could do as well as specialized algorithms, matching algorithms were not considered.

- ◆ See conclusion of Habilitation thesis of Jean-Louis Laurière
(matching [Hopcroft and Karp 73] and Hungarian assignment algorithms are explicitly mentionned)
- ◆ Well, 20 years later, the *alldifferent* constraint (and its filtering algorithm) were recognized as one core global constraint.

Reccuring Debates (1988)

- **Constraints versus algorithms ?**

Jean-Louis Laurière - OR peoples

*For specific problems (e.g. job shop 10x10) you need to develop algorithms which **take advantage of the structure of the problem**.*

- ◆ This seems to have been accepted (hopefully).
- ◆ But 20 years later, OR peoples are also involved in CP
see **CP-AI-OR** conference.

But still nowadays (parts of AI, logic, constraint) tends to ignore that no general mechanism (e.g., resolution) can yet take advantage (in a systematic way) from the structure of all potential existing problems.

Reccuring Debates (1993)

- **clp(FD): Glax box versus black box [Diaz, Codognet, ICLP 93] ?**

Philippe Codognet

"The uniform treatment of a single primitive constraint leads to a better understanding of the overall constraint solving process and makes possible three main global optimizations which encompass many previous particular optimizations of "black-box" finite domains solvers."

- ◆ well 15 years later, clp(FD) is almost not used any more, but most constraint libraries use global constraints.
- ◆ but clp(FD) has been very influential, and one is **still searching** for a clp(FD)-like concept for global constraints.

Reccuring Debates (1998)

- **Automatic and meta programming versus constraint programming ?**
 - Automatic Programming (last paper of Jean-Louis Laurière),
 - MALICE (a declarative version of ALICE by Jacques Pitrat).
- ♦ The key problems:
 - When you start to build a constraint system you start to make a set of decisions about your constraint kernel **without knowing the future and all the potential applications and extensions**
 - You optimize a lot your constraint kernel (which make it almost impossible to modify),
 - The system relies on a few set of key peoples and, as a result, you can not automatically migrate from one system to the next.

Reccuring Debates (1998)

- **Automatic and meta programming versus constraint programming ?**
 - Automatic Programming (last paper of Jean-Louis Laurière),
 - MALICE (a declarative version of ALICE by Jacques Pitrat).
- ♦ The point: **too much** knowledge/code is needed to have any chance to come up with an efficient constraint programming system.
- ♦ Their view:
 - make things **explicit**,
 - **distinguish the meaning from the usage**,
 - use **meta programming** to proceduralize knowledge,
 - provide **bootstrapping**, **introspection** and **learning** capabilities.

Reccuring Debates (2006)

- **Do we have too many global constraints ?**
- ◆ You retrieve a certain number of constraints in every system (*alldifferent, element, ...*).
- ◆ In the academy certain constraints are related to specific techniques (**breaking symmetry**: *lex_lesseq*, expressing **multi-objective criteria**: *choquet*, expressing **preferences**, ...).
- ◆ In the industry one finds some global constraints behind the scene (**embedded** within a library for a specific area: **scheduling, logistics, configuration**).

Reccuring Debates (2006)

- **Do we have too many global constraints ?**

- ◆ Global constraints are in general **not independant** from a specific domain:

scheduling : you have to handle **resource** constraints,

planning : you have to consider **optional activities**,

logistics : you have to consider **graph partitionning** constraints,

configuration : you have to be able to **add constraints in a dynamic way**,

geometry : you have to choose **continuous versus discrete**.

Reccuring Debates (conclusion)

Adapted by NB from François Pachet [2004]

“La programmation par objets (logique, contraintes) nous fait croire que l'on représente les objets (le monde) d'une manière essentielle, alors que toute l'activité de modélisation nous montre que les objets n'ont (qu'un paradigme n'a) de sens que pour une certaine classe d'application et d'usages.”

“Préférer l'étude d'un réel ancré, tel qu'il est et qu'il résiste, à une contemplation idéaliste, trop vite universelle, dont le risque est de conduire à des positions de principe rigides et des dérives idéologiques (par exemple conduisant à décréter que certaines choses sont à bannir irrévocablement, par principe).”

But we (i.e., constraint peoples) still hope/search for general paradigms (CLP(FD) for global constraints).

PREREQUISITE

GENERAL INTRODUCTION

- What is a global constraint?
- A Brief History
- Current Status
- Reccuring debates
- **So many global constraints?**
- Types of global constraints



GRAPH BASED DESCRIPTION

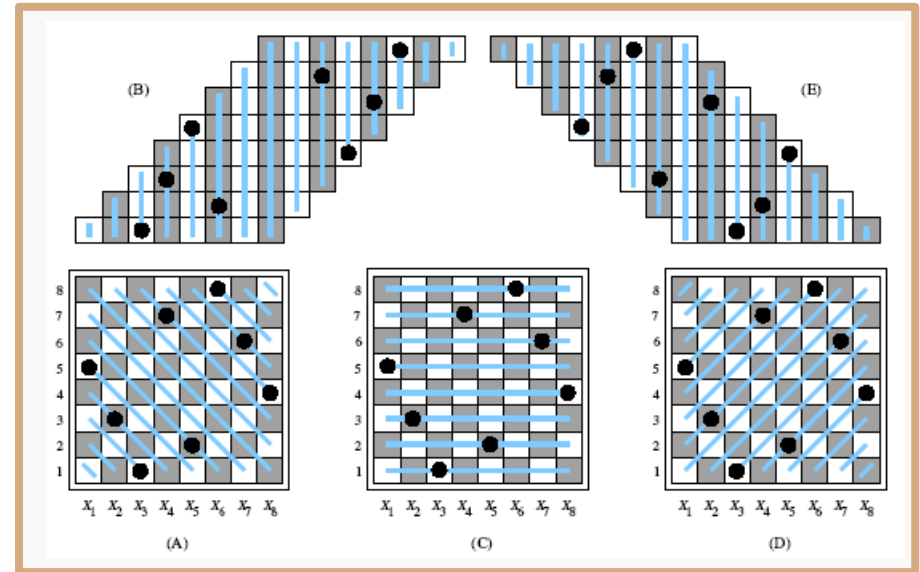
CONCLUSION

Shift From a Constant : n-queens

Consider the model of the n-queens problem with 3

alldifferent:

$\left\{ \begin{array}{l} \text{alldifferent}(X_1, X_2 + 1, \dots, X_n + n - 1) \\ \text{alldifferent}(X_1, X_2, \dots, X_n) \\ \text{alldifferent}(X_1 + n - 1, X_2 + n - 2, \dots, X_n) \end{array} \right.$



If you don't want to introduce extra variables and equality constraints you:

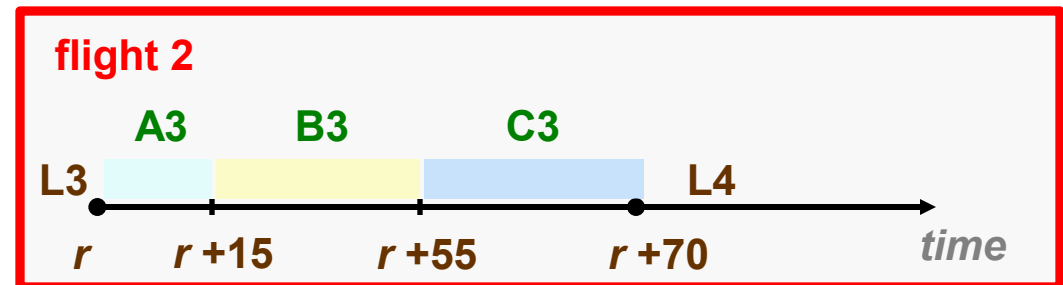
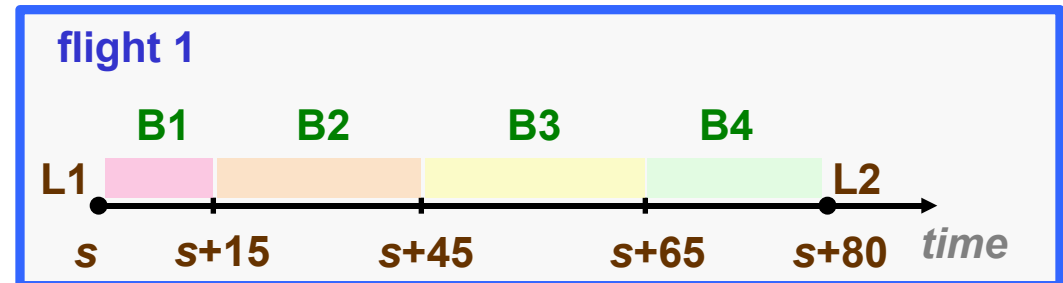
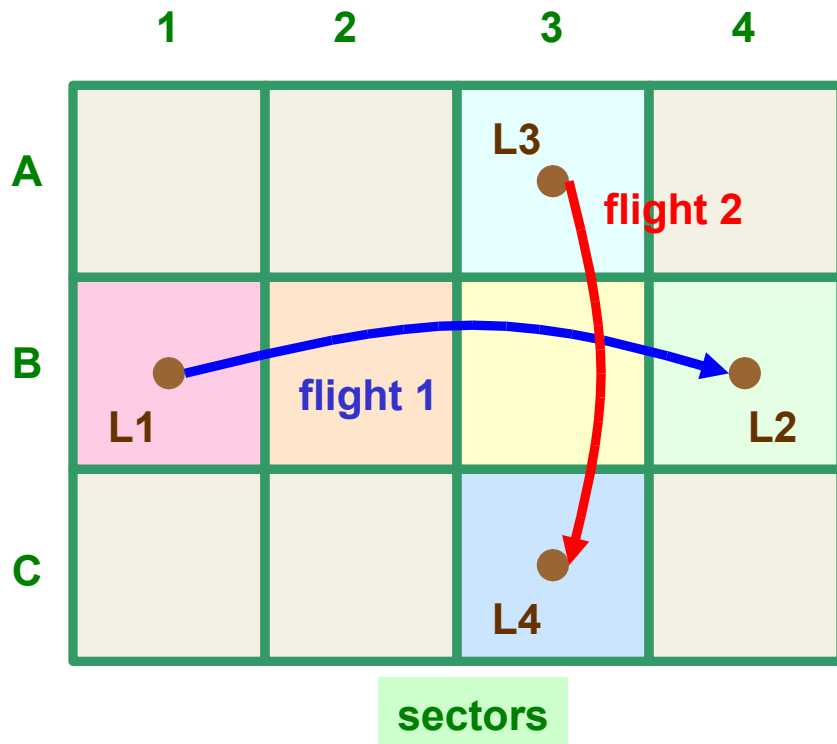
- either need to extend your *alldifferent* constraint (CHIP),
- either need to **extend your constraint engine** (Gecode).

Easy when you design it, much more difficult after !

Shift From a Constant : Air Traffic Control

Fix the exact starting time of each airplane so that:

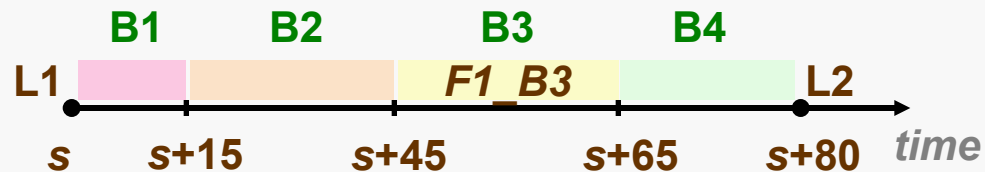
- each sector traversed by an airplane is not overcrowded.



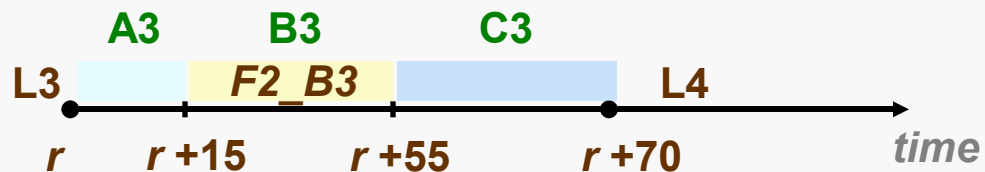
Shift From a Constant : Air Traffic Control

Create one *cumulative* constraint for each sector, and collect all tasks that go through one sector.

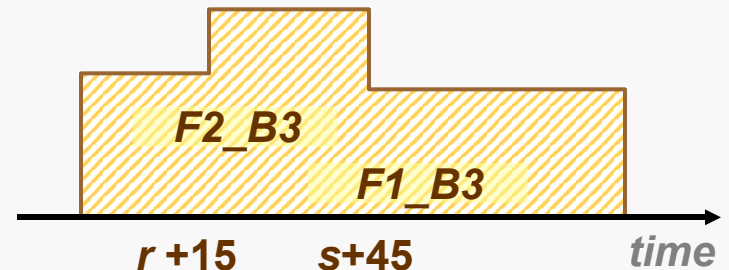
flight 1



flight 2



Use of sector B3



If your *cumulative* constraint does not allow adding a constant to the start then:
for each flight f needs to introduce a variable for each sector that is traversed by f
(as well as equality constraints) instead of having **one single** start variable.
But in practice **a constant factor matters** (for scalability)!

Dummy Values for Modelling Relaxation

OBSERVATION

Quite often you need to model the fact that you can **relax** some part of a problem.

POSSIBLE SOLUTION

A ***dummy value*** is a value that is only assigned to variables that you ignore, (i.e., variables taking a dummy value are simply ignored from the constraints where they occur).

PROBLEM

But standard constraints consider **all** their variables !

Almost Periodic

CONTEXT

Produce for a person its planning over several months:
For each day decide what kind of shift (morning, afternoon, night) he is doing or if he is off.

Moreover the planning produced should be periodic (possibly with a small period) so that you can repeat it over and over.

A PERIODIC SCHEDULE

M A A O O N N A A O O M A A O O N N A A O O M A A O O N

← period = 11 →

SOLUTION

Use the **period** constraint [ICLP'04] $\text{period}(P, [S_1, S_2, \dots, S_n])$, which enforce that P is the smallest period of the sequence $S_1 S_2 \dots S_n$.

Almost Periodic

CRITICAL POINT

In practice using the **period** constraint is not going to work since usually a person can reserve some days off (standard vacations + extra days) which by definition are not placed in a periodic way.

A PERIODIC SCHEDULE ???

M A A O O N N A A O O M A A O O N O A A O O M A A O O N

preassigned day off

SOLUTION

Use the **period_except_0** constraint
where equality **$x=y$** is replaced by **$x=y$ or $x=0$ or $y=0$** .

0 plays the role of a dummy value

Almost a Bijection

You want to match two sequences of genes (so *that you minimise some cost*).

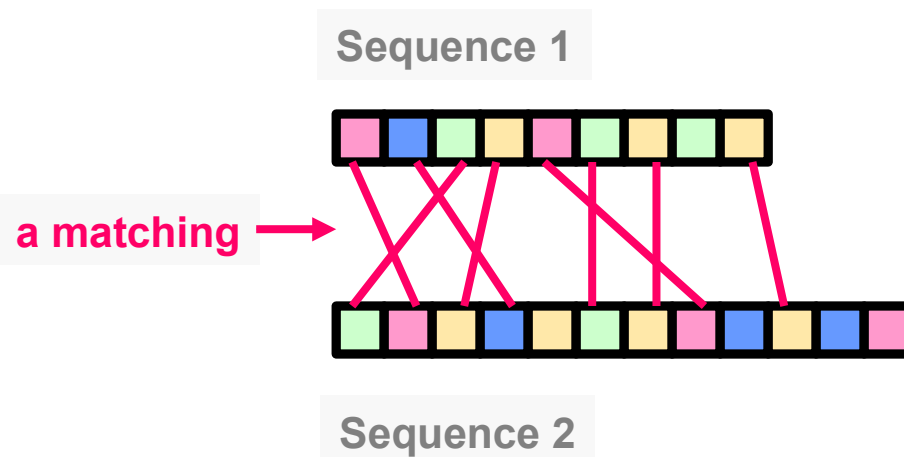
Sequence 1



Sequence 2

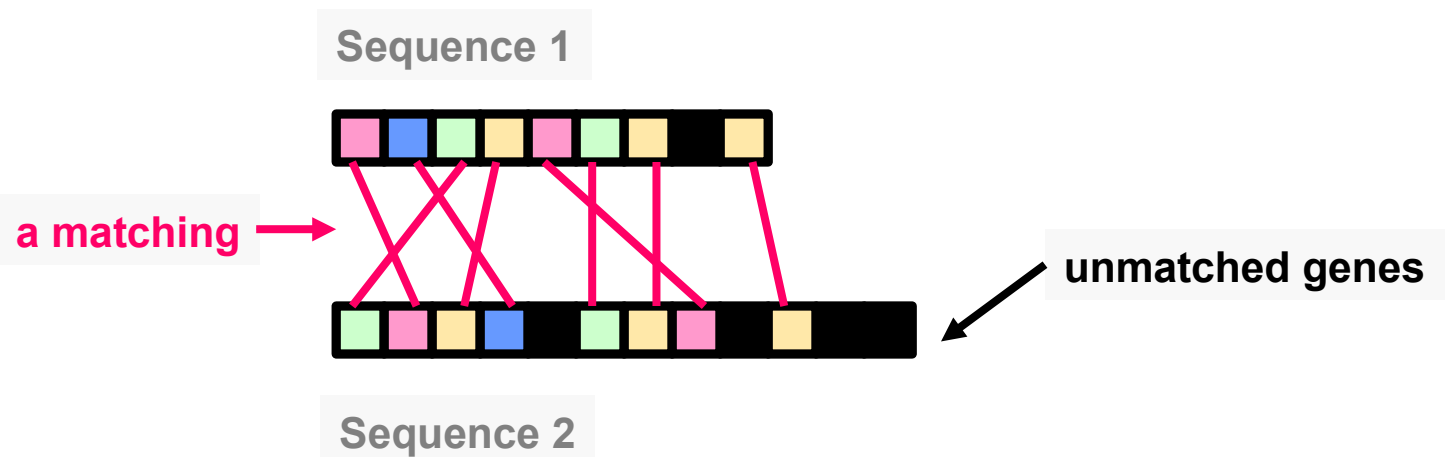
Almost a Bijection

You want to match two sequences of genes (so *that you minimise some cost*).



Almost a Bijection

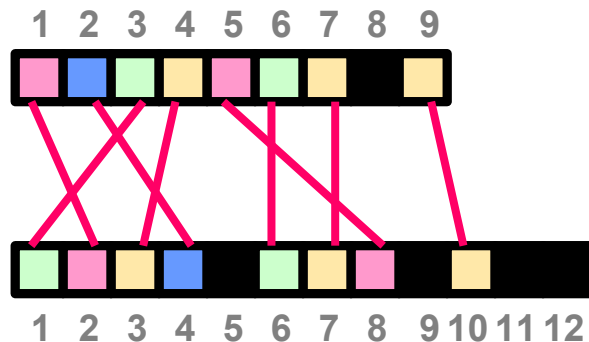
You want to match two sequences of genes (so *that you minimise some cost*).



Almost a Bijection: Dual Model with a Channelling Constraint

You want to match two sequences of genes (so *that you minimise some cost*).

Sequence 1



Sequence 2

Sequence 1	Sequence 2
A1 = 2	B1 = 3
A2 = 4	B2 = 1
A3 = 1	B3 = 4
A4 = 3	B4 = 2
A5 = 8	B5 = 0
A6 = 6	B6 = 6
A7 = 7	B7 = 7
A8 = 0	B8 = 5
A9 = 10	B9 = 0
	B10 = 9
	B11 = 0
	B12 = 0

Dual model

Question: how do you link these two sets of variables ?

The standard channelling constraint (*inverse*) is not working !
Which leads to *inverse_within_range* (*inverse*+dummy value)

Almost *cumulative* (example 1)

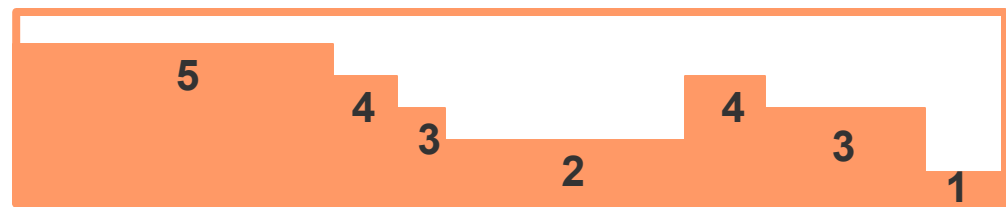
THE PROBLEM

You have a printer that can print 5 orders in parallel.
Each order can use at most 4 colours from a predefined set of 6 colours.
Schedule a set of given printing orders.

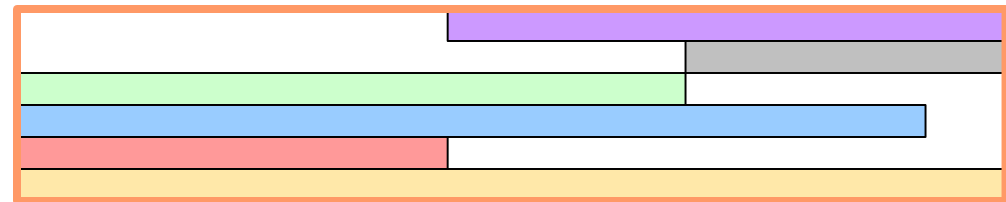
OBSERVATION

It sounds like a cumulative problem (**well, not completely**).

Almost *cumulative* (example 1)

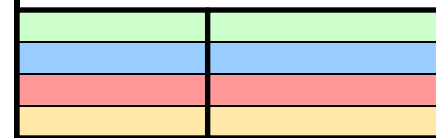


cumulative profile

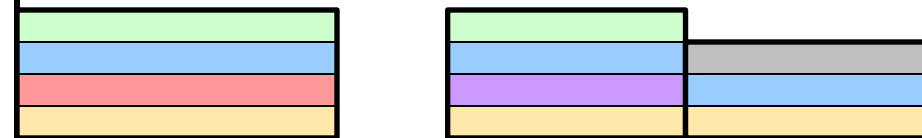


Used colours

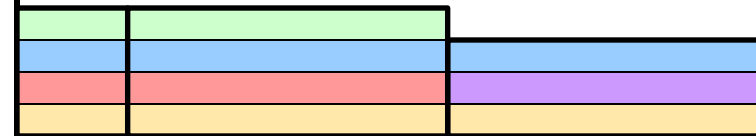
Output 5



Output 4



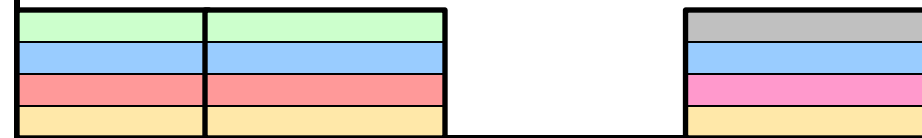
Output 3



Output 2



Output 1

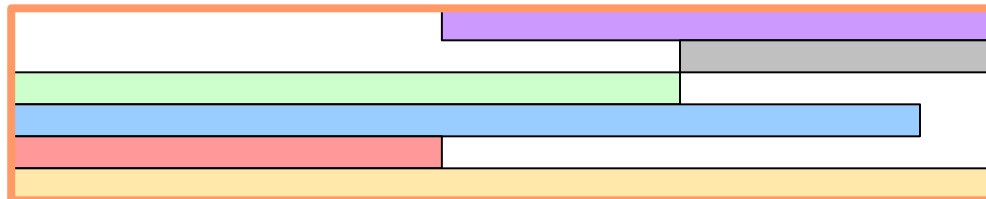


time

Almost *cumulative* (example 1)

The constraint on the maximum number of colours in parallel cannot be modelled as a *cumulative* constraint:

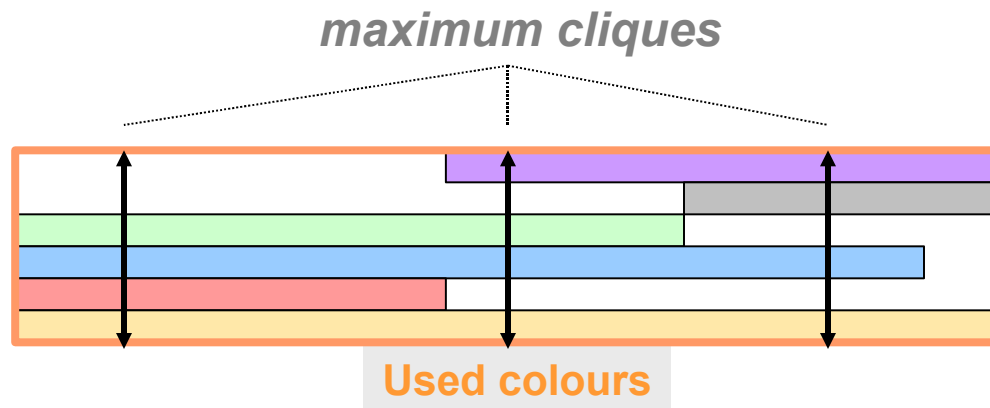
the *sum* constraint of the heights of the tasks of the *cumulative* constraint has to be replaced by the *nvalue* constraint (*coloured_cumulative*)



Used colours

Almost *cumulative* (example 1)

You get an *nvalue* constraint on each maximal clique of your interval graph.



Almost *cumulative* (example 2)

THE PROBLEM

On one side you have a **set of orders** to deliver, where each order has a weight and a destination town.

On the other side you have a **fleet of vehicles**, where each vehicle has a give capacity.

Question: assign the orders to the vehicles in order to:

- (1) not exceed the capacity of each vehicle,
- (2) each vehicle should visit **at most 3 distinct towns**.

OBSERVATION

Again, you need a **coloured_cumulative** constraint for expressing (2) in a direct way.

Almost *cumulative* (example 3)

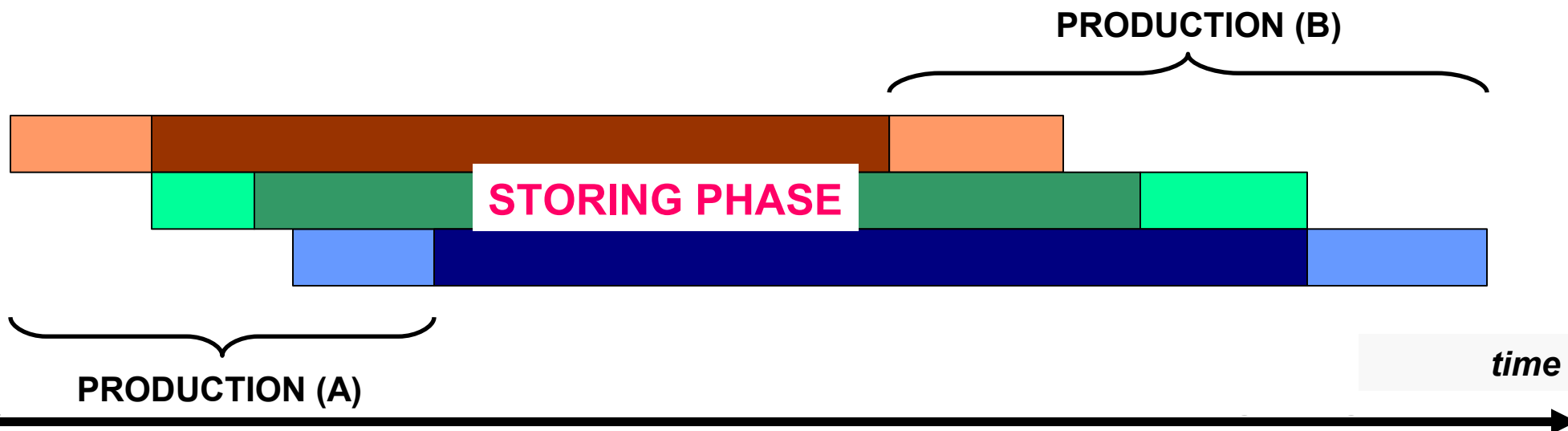
THE PROBLEM

A first part (A) of a plant produces steel in batches.

A second part (B) of a plant uses each batch to produce something.

The steel should be maintained at a minimum temperature and therefore stays in a special wagon (i.e., a “couveuse”).

Question: given a maximum storage capacity for storing the steel
encode this constraint.



Almost *cumulative* (example 3)

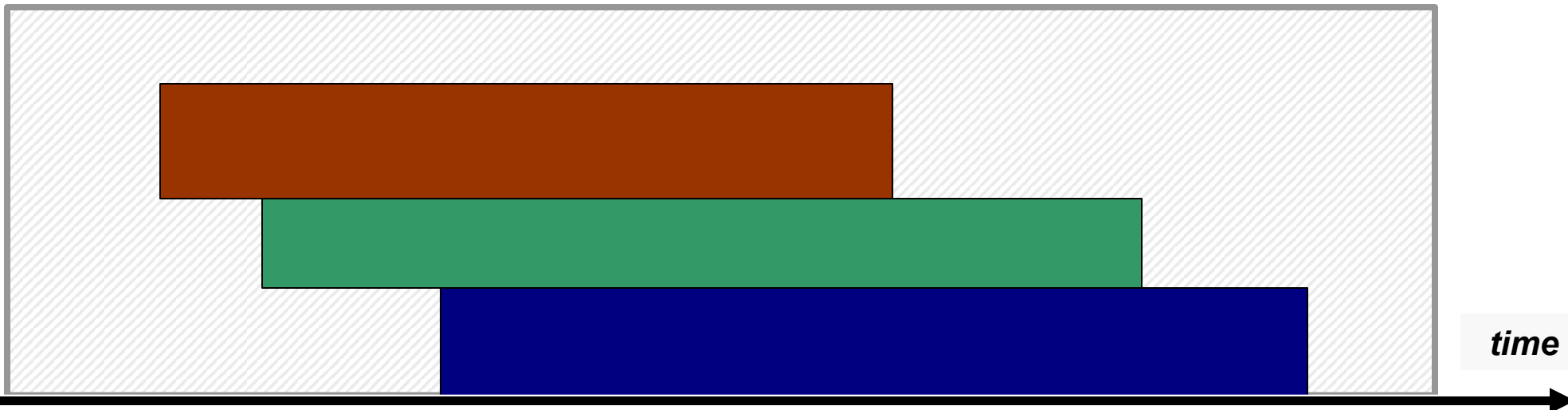
SOLUTION

For each batch creates a storing task, with:

- a **start** corresponding to end of production (A)
- an **end** corresponding to the start of production (B)
- a **height** corresponding to the quantity associated to the batch.

And put all the storing tasks in a *cumulative* constraint.

If your *cumulative* constraint describes each task by a triple (**start**, **duration**, **height**) then creates an extra **end** and states the constraint **start+duration=end**.



Almost *cumulative* (example 3)

PROBLEM

If your ***cumulative*** constraint does not allow to directly use an **end** variable for describing a task then propagation will be very poor.

ILLUSTRATION OF THE PROBLEM

Assume you have a task such that:

its **origin** is within [1,2] and its **end** is within [7,8].

If your ***cumulative*** constraint does not accept ends then you create a **duration** variable that is within [5,8] and you state the constraint **start+duration=end**
(which does not trigger any further reduction)

But, since your ***cumulative*** constraint knows only the **start** and the **duration**, it estimates the **earliest end** to 1+5 and the **latest end** to 2+8 !!!

These **under and over estimations** will kill any clever filtering algorithm of the *cumulative* constraint (*compulsory part, edge finding, ...*).

In this context you need a cumulative constraint that accepts end variables

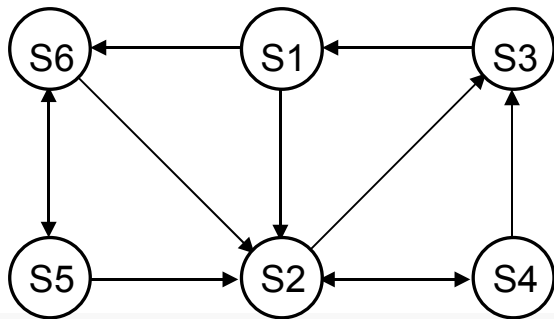
Relaxation again (# of proper patterns)

Consider the context of graph covering constraints (*cycle*, *map*, *path*, *tree*) where **each vertex should belong to one single pattern**.

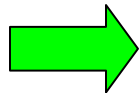
AN INSTANCE OF GRAPH COVERING CONSTRAINT

cycle

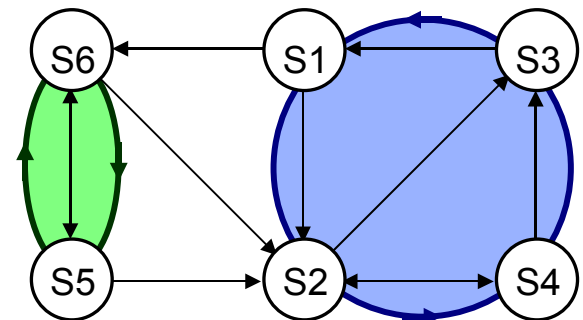
Cover a directed graph by a set of circuits in such a way that each vertex belongs to exactly one circuit (or # of cycles of a permutation).



cycle(N, {S1,S2,S3,S4,S5,S6})



S1 :: {2,6}
S2 :: {3,4}
S3 :: {1}
S4 :: {2,3}
S5 :: {2,6}
S6 :: {2,5}



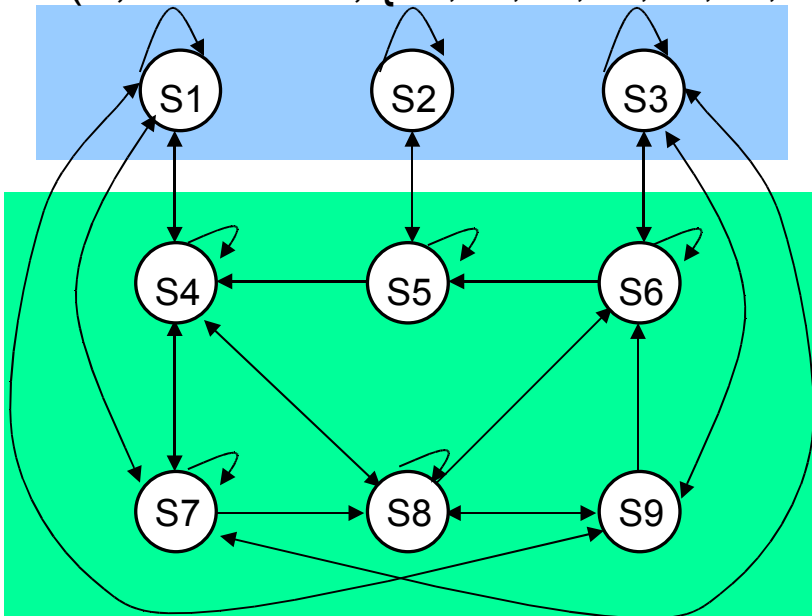
cycle(2, {2,4,1,3,6,5})

Relaxation again (# proper patterns)

Quite often we have two kind of vertices: – **resource** vertices, – **task** vertices.
A pattern corresponds to a combination of a resource and several tasks.
Self-loops corresponds to **idle resources** or **relaxed tasks**.

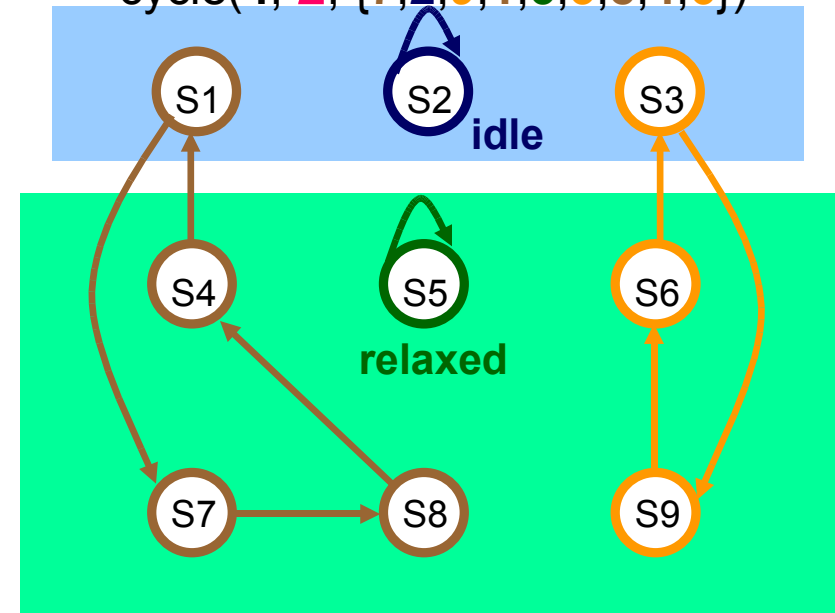
In this context you are interested by the **number of proper patterns** (**≠self-loops**):
 $\text{cycle}(N, \text{NPROPER}, \text{SUCCESSORS})$

$\text{cycle}(N, \text{NPROPER}, \{\text{S1}, \text{S2}, \text{S3}, \text{S4}, \text{S5}, \text{S6}, \text{S7}, \text{S8}, \text{S9}\})$



initial graph

$\text{cycle}(4, 2, \{7, 2, 9, 1, 5, 3, 8, 4, 6\})$



a solution

ps Summer

Lack of Communication Between Constraints

SLOGAN OF CONSTRAINT PROGRAMMING

Constraints communicates **only** via the domain of the variables, so constraints are **independent** from each other.

- **GOOD NEWS:** can add constraint **regardless** of existing constraints
(not as with classical algorithms)
- **BAD NEWS:** yes you can, but how it propagates is an other story; you have to have the faith in constraint programming
(not as with classical algorithms)



CONSEQUENCES

obvious propagation missing,
creates artificial global constraints
or
extend your constraint kernel.

Integrating Functional Dependency Constraints into Existing Global Constraints

FUNCTIONAL DEPENDENCY CONSTRAINTS

A constraint that allows for representing a *functional dependency* between two domain variables.

A variable X is said to *functionnaly determine* another variable Y if and only if each potential value of X is associated with exactly one potential value of Y .

EXAMPLE

element(**Index**, Table, **Value**)

element(**4**, [6, 9, 2, 9], **9**)

1 2 3 4

Integrating Functional Dependency Constraints into Existing Global Constraints

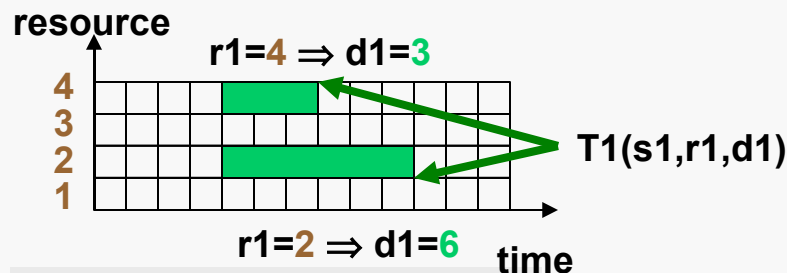
RECCURING CONSTRAINT PATTERN

Very often you have:

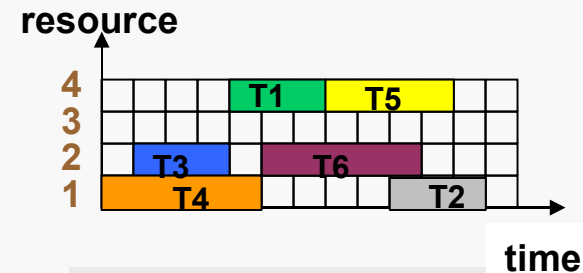
- (1) a **collection of objects** on which you impose a global constraint,
- (2) these objects have **several** attributes,
- (3) there is a **functional dependency constraint** between two attributes.

EXAMPLE

- (1) a collection of tasks so that tasks assigned to the same resource **should not overlap** (*diffn* constraint).
- (2) each task has a **resource** (dvar), a **start** (dvar) and a **duration** (dvar).
- (3) the duration of a task **depends** of the resource to which it is assigned.



functional dependency

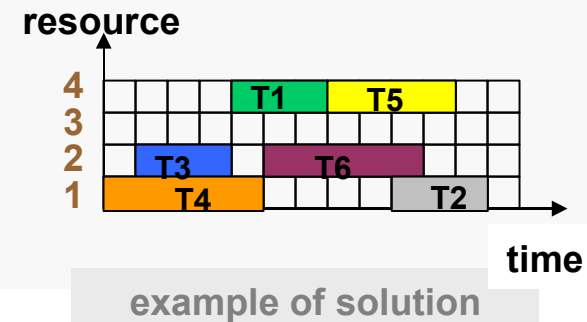
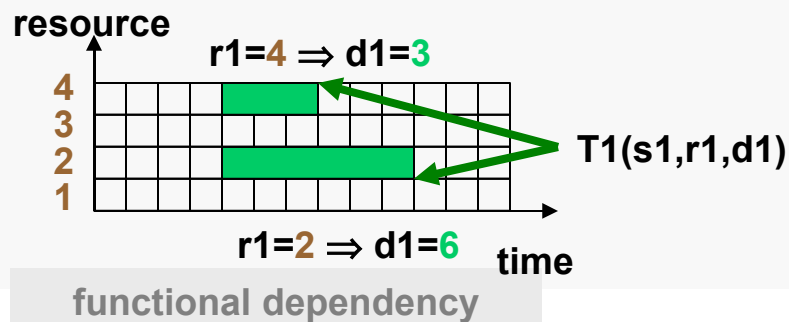


example of solution

Integrating Functional Dependency Constraints into Existing Global Constraints

EXAMPLE

- (1) a collection of tasks so that tasks assigned to the same resource **should not overlap** (*diffn* constraint).
- (2) each task has a **resource** (dvar), a **start** (dvar) and a **duration** (dvar).
- (3) the duration of a task **depends** of the resource to which it is assigned.



MODEL

non-overlapping constraint between all the tasks

`diffn([t(s1,r1,d1,1), t(s2,r2,d2,1), t(s3,r3,d3,1), t(s4,r4,d4,1), t(s5,r5,d5,1), t(s6,r6,d6,1)])`

functional dependency constraints (relation between the resource and the duration)

`element(r1,[9,6,9,3],d1)`

`element(r3,[9,3,1,3],d3)`

`element(r5,[2,6,4,4],d5)`

`element(r2,[3,2,2,3],d2)`

`element(r4,[5,6,6,9],d4)`

`element(r6,[7,5,7,5],d6)`

Integrating Functional Dependency Constraints into Existing Global Constraints

MODEL

$\text{diffn}([t(s1,r1,d1,1), t(s2,r2,d2,1), t(s3,r3,d3,1), t(s4,r4,d4,1), t(s5,r5,d5,1), t(s6,r6,d6,1)])$

$\text{element}(r1,[9,6,9,3],d1)$

$\text{element}(r2,[3,2,2,3],d2)$

$\text{element}(r3,[9,3,1,3],d3)$

$\text{element}(r4,[5,6,6,9],d4)$

$\text{element}(r5,[2,6,4,4],d5)$

$\text{element}(r6,[7,5,7,5],d6)$

TYPICAL MISSING PROPAGATION

INITIAL ASSUMPTIONS:

$s1 \in [1,9]$, $r6=2$ (by *element* $d2=5$), $s6=6$.

QUESTION: can **T1** be assigned to resource **2** ?

T1 should not overlap $[6, 6+5-1]$

$s1 \notin [6 - \min(d1), 6+5-1]$

since $\min(d1)=3$ (for *diffn*) we get $s1 \notin [3, 6+5-1] \not\subset [1,9]$ (no deduction !!!)

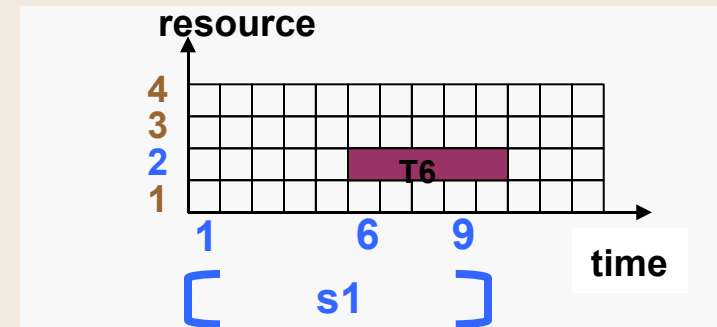
PROBLEM: lack of communication between *diffn* and *element*

T1 should not overlap $[6, 6+5-1]$ (implicitly assume $r1=2$)

$s1 \notin [6 - \min(d1), 6+5-1]$

since $\min(d1)=6$ ($\text{element}(r1,[9,6,9,3],d1)$)

we get $s1 \notin [0, 6+5-1] \subseteq [1,9]$ (a contradiction $\Rightarrow r1 \neq 2$)



No clever filtering algorithm is missing,
but still it kills propagation !

Integrating Functional Dependency Constraints into Existing Global Constraints

PREVIOUS MODEL

```
diffn([t(s1,r1,d1,1), t(s2,r2,d2,1), t(s3,r3,d3,1), t(s4,r4,d4,1), t(s5,r5,d5,1), t(s6,r6,d6,1)])
```

```
element(r1,[9,6,9,3],d1)
```

```
element(r2,[3,2,2,3],d2)
```

```
element(r3,[9,3,1,3],d3)
```

```
element(r4,[5,6,6,9],d4)
```

```
element(r5,[2,6,4,4],d5)
```

```
element(r6,[7,5,7,5],d6)
```

FIRST SOLUTION *(exists also for other constraints)*

(A) create an new global constraint
combining *diffn* and a set of *element* constraints

```
diffn([t(s1,r1,d1,1), t(s2,r2,d2,1), t(s3,r3,d3,1), t(s4,r4,d4,1), t(s5,r5,d5,1), t(s6,r6,d6,1)],  
      dependency(resource,duration,Matrix))
```

where: Matrix = [[9,6,9,3], [3,2,2,3], [9,3,1,3], [5,6,6,9], [2,6,4,4], [7,5,7,5]]

(B) revise the filtering algorithm of *diffn*

each time you access to the minimum duration **get the correct minimum value**

Integrating Functional Dependency Constraints into Existing Global Constraints

PREVIOUS MODEL

`diffn([t(s1,r1,d1,1), t(s2,r2,d2,1), t(s3,r3,d3,1), t(s4,r4,d4,1), t(s5,r5,d5,1), t(s6,r6,d6,1)])`

`element(r1,[9,6,9,3],d1)`

`element(r2,[3,2,2,3],d2)`

`element(r3,[9,3,1,3],d3)`

`element(r4,[5,6,6,9],d4)`

`element(r5,[2,6,4,4],d5)`

`element(r6,[7,5,7,5],d6)`

SECOND SUGGESTED SOLUTION [BELD 00]

A new communication primitive between constraints

ask(Information, Expression, Restriction)

- **Information** : min, max
- **Expression** : X, X+Y, X.Y
- **Restriction** : X=val, X>val, X<val

Ask about the minimum or maximum value of an expression according to a restriction on one other variable.

Example of use of ask : Interaction of Arithmetic Constraints

PROBLEM

$$\left\{ \begin{array}{l} (1) \text{ } Low \leq \sum_{i=1}^n X_i \cdot Y_i \leq Up \\ (2) \forall i \in 1..n : Low_i \leq X_i \cdot Y_i \leq Up_i \end{array} \right.$$

PROPAGATORS FOR (1)

if $\max(Y_i) > 0$

$$\text{then } \min(X_i) \geq \left\lfloor \frac{Low - \sum_{j=1, j \neq i}^n \max(X_j \cdot Y_j)}{\max(Y_i)} \right\rfloor$$

if $\min(Y_i) > 0$

$$\text{then } \max(X_i) \leq \left\lfloor \frac{Up - \sum_{j=1, j \neq i}^n \min(X_j \cdot Y_j)}{\min(Y_i)} \right\rfloor$$

WITHIN PROPAGATORS FOR (1) USE

```
ask(max, Xj.Yj, [])
ask(min, Xj.Yj, [])
```

INSTEAD OF

```
max(Xj) * max(Yj)
min(Xj) * min(Yj)
```

So that (2) can provide accurate bounds to (1) !

Example of use of ask : Interaction of *cumulative* and *element*

PROBLEM

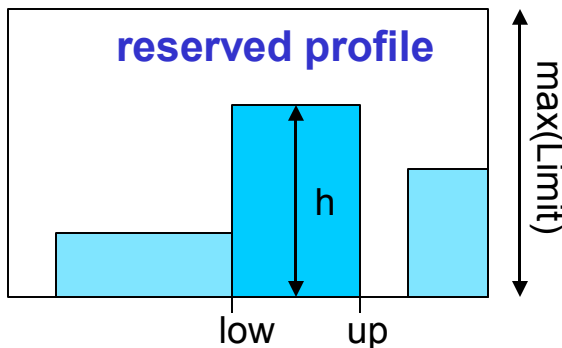
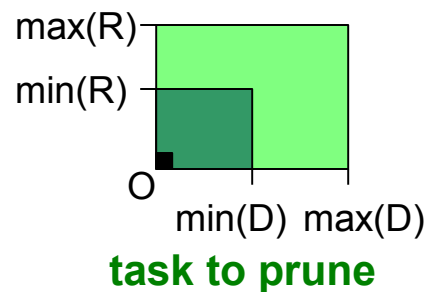
Prune task origin

- origin: O
- duration: D
- resource: R

According to interval

- low
- up
- h

And a given maximum Limit



And the fact that they may be some dependencies between O, D and R.

CLASSICAL PROPAGATOR

IF $\min(R) > \max(\text{Limit}) - h$

THEN remove interval $[\text{low} - \min(D) + 1, \text{up}]$ from O

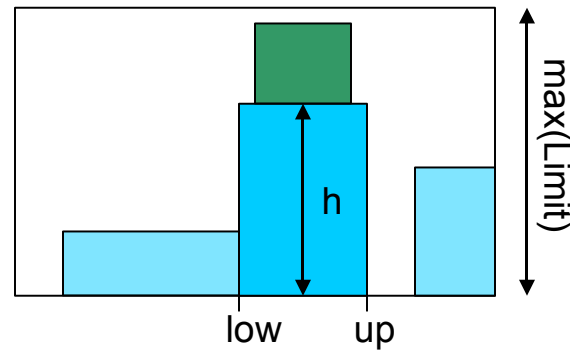
Interaction of *cumulative* and *element*: Identifying the Two Problems

POTENTIAL PROBLEM

Weak pruning because considers that:

- duration takes $\min(D)$
- resource takes $\min(R)$

And so task can overlap interval $[low, up]$



CLASSICAL PROPAGATOR

IF $\min(R) > \max(Limit) - h$

THEN remove interval $[low - \min(D) + 1, up]$ from O

Interaction of *cumulative* and *element*: Making **Explicit** the Implicit Hypothesis

Assumes that the task overlaps interval $[low, up]$.

Assumes that the task overlaps interval $[low, up]$
and that $\min(R) > \max(Limit) - h$.

```
IF min(R) > max(Limit) - h  
THEN remove interval  $[low - \text{min}(D) + 1, up]$  from O
```

Example of use of **ask** : Interaction of *cumulative* and *element*

A BETTER PROPAGATOR

```
IF    ask(min,R, [O+D-1≥low, O≤up]) > max(Limit) - h  
THEN remove [low - ask(min,D, [R > max(Limit) - h, O+D-1≥low, O≤up]) + 1, up] from O;
```

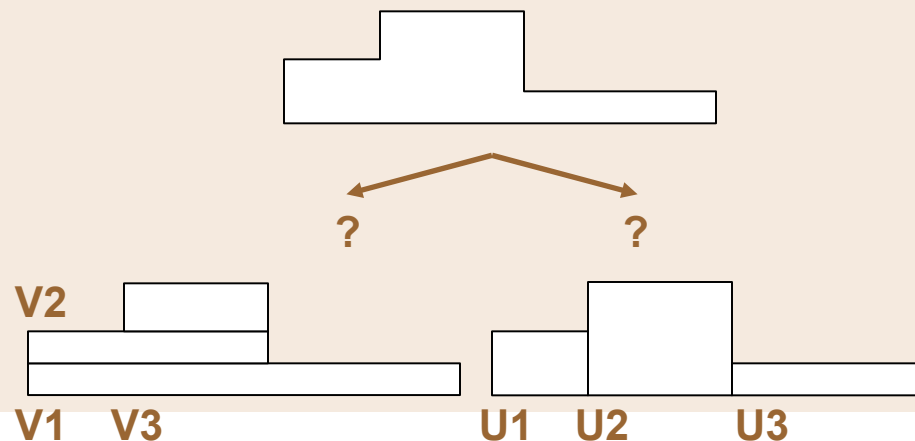
When Breaking Things into Peaces Kills Propagation

RECCURING CONSTRAINT PATTERN

Very often you have:

- (1) a **collection of shapes** on which you impose a global constraint (cumulative, non-overlapping, ...),
- (2) these shapes are not just rectangles,
- (3) you model it by a set of rectangles (and you link the coordinates of each rectangle by a constraint of the type $\text{Var1} = \text{Var2} + \text{Cst}$).

cumulative (variable resource consumption)



Both decompositions
hinder propagation
(*in the context of compulsory parts*)

$$V1, V2, V3, V1 = V2, V2 + C2 = V3$$

$$U1, U2, U3, U1 + D1 = U2, U2 + D2 = U3$$

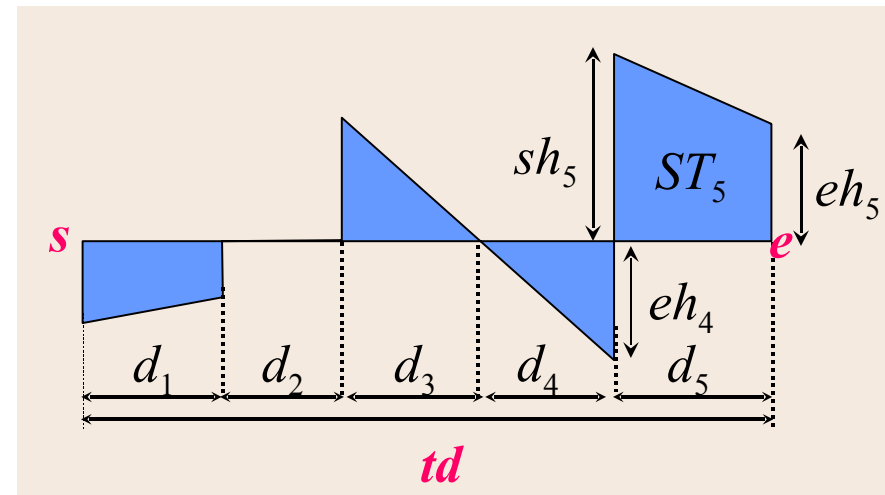
When Breaking Things into Peaces Kills Propagation

Consequence

You may want to have global constraints handling directly complex shape
without breaking things into peaces and without introducing extra variables.

A task T is defined by [BELDICEANU PODER 2004]:

- a **start** $s \in \mathbb{R}$, an **end** $f \in \mathbb{R}$, a total **duration** $td \in \mathbb{R}^+$ and a set $r \in \mathbb{N}$ of possible resource assignments
- a **positive or negative piecewise linear** resource function represented by a **sequence of** p consecutive **trapezoid** sub-tasks ST_1, ST_2, \dots, ST_p where ST_i has a start height $sh_i \in \mathbb{R}$, an end height $eh_i \in \mathbb{R}$ and a duration $d_i \in \mathbb{R}^+$.



Conclusion (so many global constraints?)

In the early 90, even a few global constraints (i.e., 5) have started to irritate a few Logic and AI people,

In the early 90, CP (BULL with CHARME) promised to model very easily a lot of real problems (which has irritated most OR people)

The number of global constraints increased in the 90 (motivated by specific applications).

Ideally you would like to be able to just describe the semantics of your constraint and get a visualisation algorithm, a filtering algorithm, ... from that description: which was the motivation for the graph-based description.

PREREQUISITE

GENERAL INTRODUCTION

- What is a global constraint?
- A Brief History
- Current Status
- Reccuring debates
- So many global constraints?
- ➔ • **Types of global constraints**

GRAPH BASED DESCRIPTION

CONCLUSION

Overview of the Different Types of Constraints

- **Data** constraints
- **Value** constraints (**one** set of variables)
- **Value** constraints (**two** sets of variables)
- **Ordering** constraints
- **Channeling** constraints
- **Resource scheduling** constraints
- **Geometrical** constraints
- **Graph covering** constraints
- **Graph** constraints
- **Relaxation** and **proximity** constraints
- **Cost filtering** constraints

Data Constraints

DEFINED IN EXTENSION OR ACCESS TO A DATA STRUCTURE

element [VanHentenryckCarillon88]

```
element(3, [[value-6], [value-9], [value-2], [value-9]], 2)
```

- **case** [SICStus00]
- **element_greater_eq, element_lesseq** [OttossonThorsteinssonHooker99]
- **element_sparse** [CHIP]
- **in_relation**
- **sum** [Yunes02]
- **graph & map variables** [Deville et al. 05]

You also maintain invariants (constraints) on your data structure

Data Constraints (case)

Allows to represent a constraint defined in extension in a **compact way** by providing a **dag** (with a unique source) representing implicitly a set of tuples (*work also for the non-binary case*).

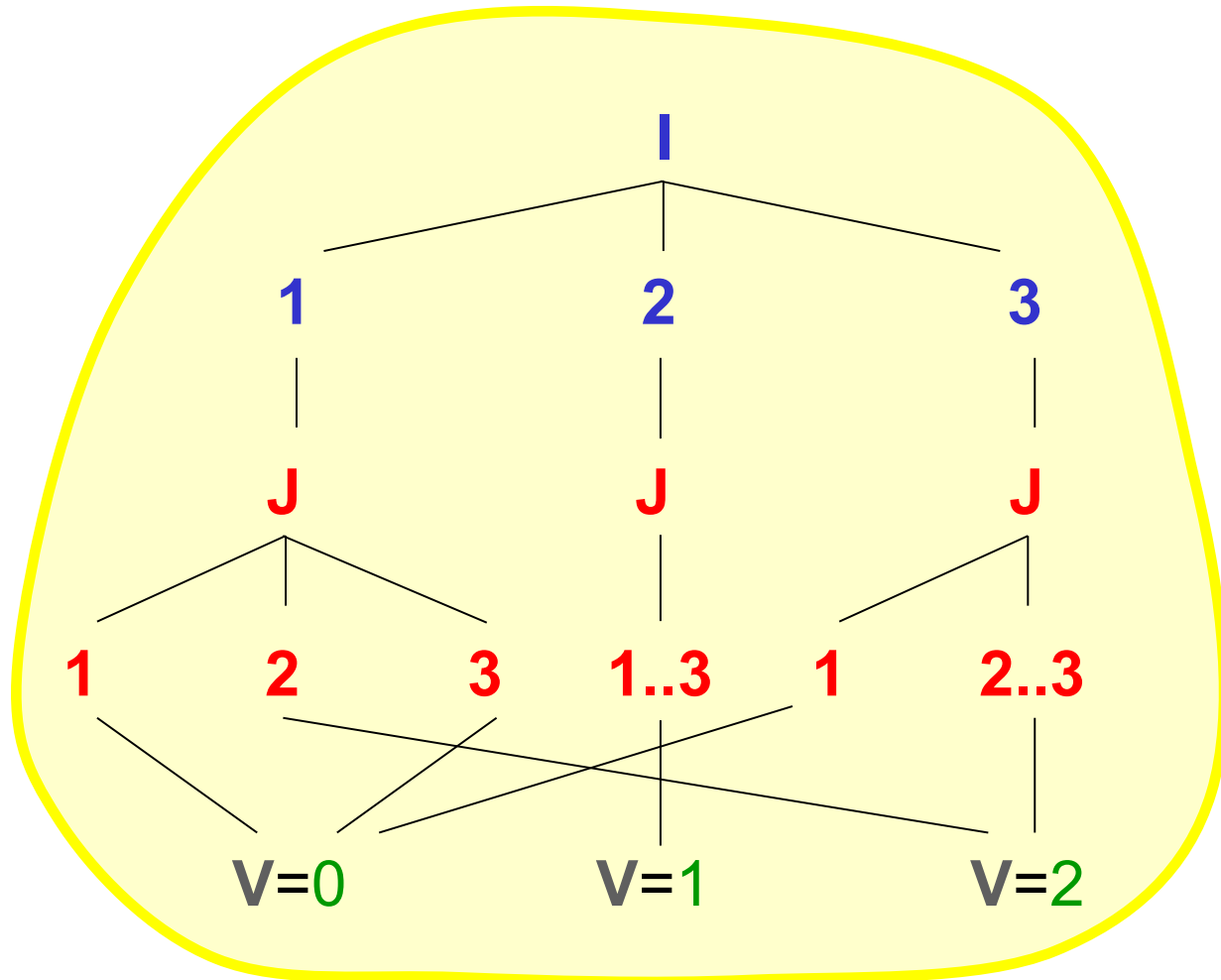
- Generalise element and is used for encoding the transition constraint of an automaton (within the context of the reformulation).
- From a graph perspective enforces a path between the source and a sink.

Data Constraints (case)

Want a constraint of the form $\text{elem}(\mathbf{I}, \mathbf{J}, \text{Matrix}, \mathbf{Val})$

$\mathbf{I}=1$	0	2	0
$\mathbf{I}=2$	1	1	1
$\mathbf{I}=3$	0	2	2

2-dim.matrix



corresponding dag for case

mer

Data Constraints (case)

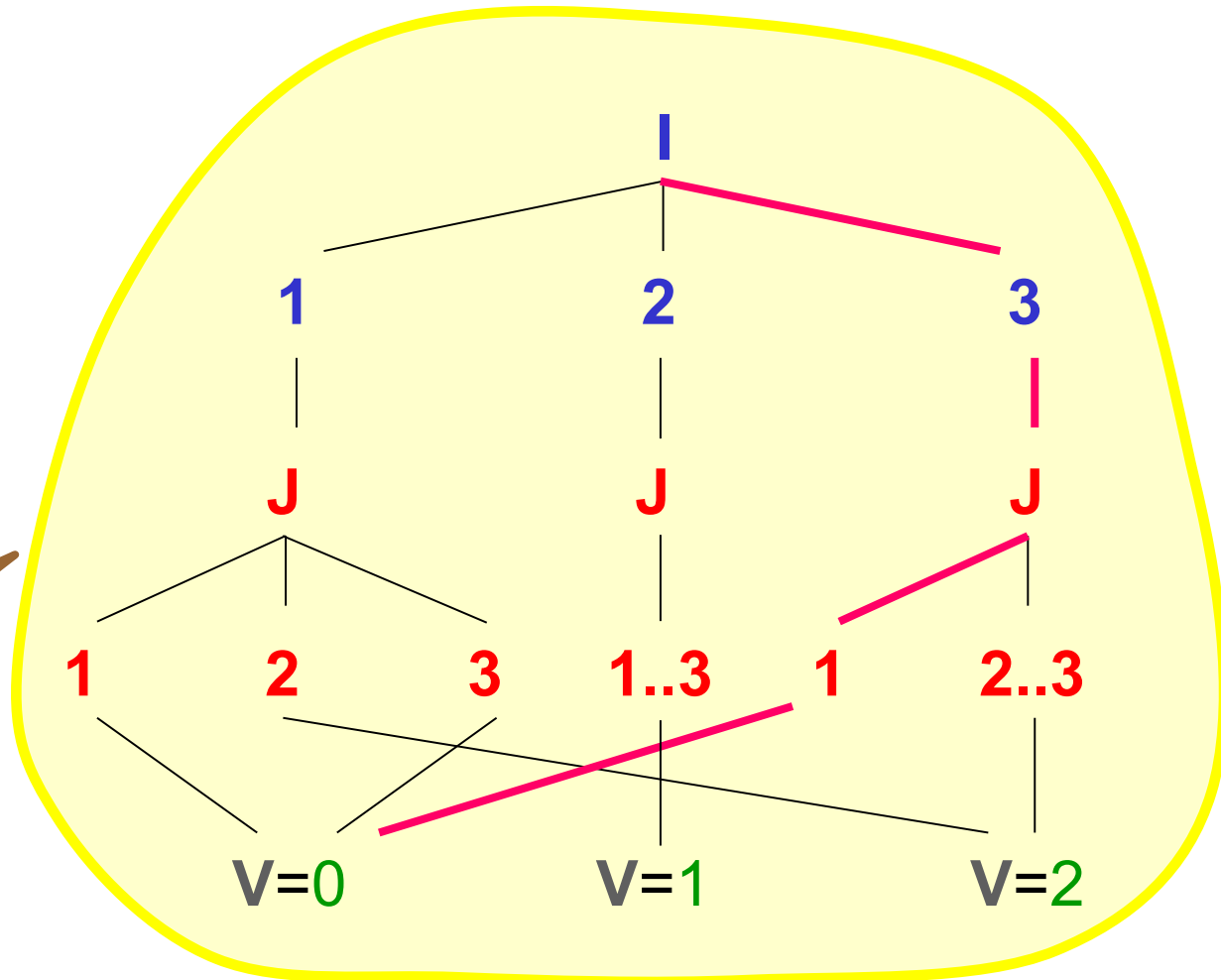
Want a constraint of the form $\text{elem}(\mathbf{I}, \mathbf{J}, \text{Matrix}, \mathbf{Val})$

$\mathbf{I}=1$	0	2	0
$\mathbf{I}=2$	1	1	1
$\mathbf{I}=3$	0	2	2

\mathbf{J}

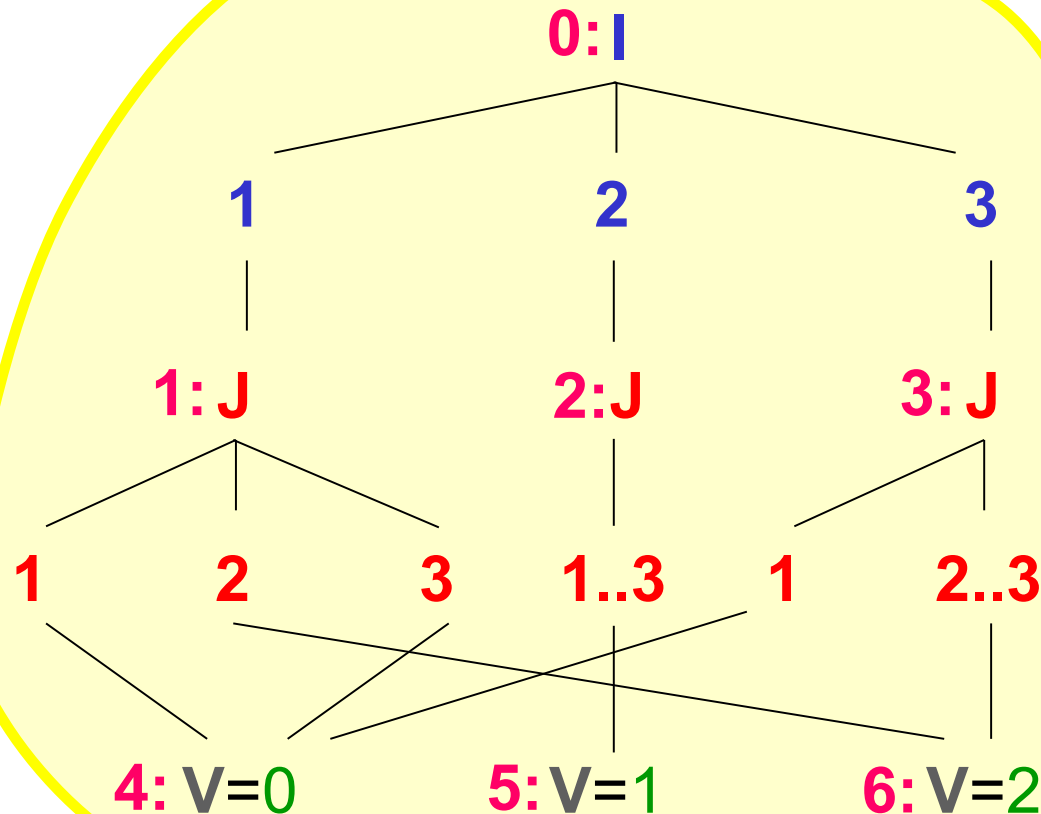
2-dim.matrix

$\text{elem}(\mathbf{3}, \mathbf{1}, \text{Matrix}, \mathbf{0})$



corresponding dag for case mer

Data Constraints (case)



corresponding dag for case

```
case ( f(I,J,V) ,
      [ f(X,Y,Z) ] ,
      [ node(0,I, [(1..1)-1,
                    (2..2)-2,
                    (3..3)-3]) ,
        node(1,J, [(1..1)-4,
                    (2..2)-6,
                    (3..3)-4]) ,
        node(2,J, [(1..3)-5]) ,
        node(3,J, [(1..1)-4,
                    (2..3)-6]) ,
        node(4,V, [(0..0)]) ,
        node(5,V, [(1..1)]) ,
        node(6,V, [(2..2)]) ] )
```

Value Constraints (one set of variables)

CONSTRAINT ON ONE SET OF VARIABLES

alldifferent [Laurière78] [Costa94] [Régis94] [MehlhornThiel00]

```
alldifferent([var-5], [var-1], [var-9], [var-3])
```

- **among** [BeldiceanuContejean94]
- **balance** [Beldiceanu00]
- **change** [CHIP] [BeldiceanuCarlsson01]
- **global_cardinality** [CHARME] [Régis96] [KatrielThiel03] [QuimperBeek03]
- **group** [CHIP]
- **stretch** [Pesant01]
- **symmetric_alldifferent** [Régis99]

Value Constraints (two sets of variables)

CONSTRAINT ON TWO SETS OF VARIABLES

sort [OlderSwinkelsEmden95] [GuernalecColmerauer97] [MehlhornThiel00]
`sort([var-1], [var-9], [var-1], [var-5], [var-2], [var-1]),
 [var-1], [var-1], [var-1], [var-2], [var-5], [var-9]))`

- **disjoint** [Beldiceanu00]
- **same** [BeldiceanuKatrielThiel04]
- **sort_permutation** [Zhou97]
- **symmetric_gcc** [KocjanKruger04]
- **used_by** [BeldiceanuKatrielThiel04]

Ordering Constraints

CONSTRAINT ON TWO SETS OF VARIABLES

minimum [CHIP] [Beldiceanu01]

```
minimum(2, [[var-3], [var-2], [var-7], [var-2], [var-6]])
```

- **between** [BeldiceanuCarlsson04]
- **lex_lesseq** [CHIP] [FrischHnichKiziltanMiguelWalsh02]
- **lex_chain_lesseq** [BeldiceanuCarlsson04]
- **maximum** [CHIP] [Beldiceanu01]
- **mset_lesseq** [FrischHnichKiziltanMiguelWalsh03]

Channeling Constraints

MAKE THE LINK BETWEEN TWO MODELS

domain_constraint [Refalo00]

```
domain_constraint(5, [[var01-0, value-9], [var01-1, value-5],  
                     [var01-0, value-2], [var01-0, value-7]])
```

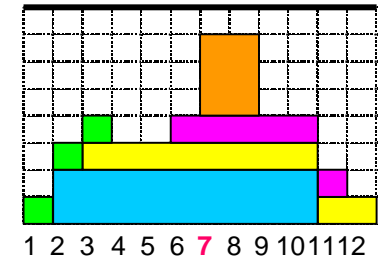
- inverse [CHIP]
- link_set_to_booleans

Resource Scheduling Constraints

SCHEDULING UNDER VARIOUS RESOURCE CONSTRAINTS

cumulative [Beldiceanu93] [Lahrichi82] [ErschlerLopez90] [CaseauLaburthe96]

```
cumulative([ [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)
```



- **bin_packing**
- **coloured_cumulative** [Beldiceanu00]
- **cumulatives** [BeldiceanuCarlsson02]
- **disjoint_tasks** [Beldiceanu00]
- **track** [Marte01]

Geometrical Constraints

NON-OVERLAPPING AND DISTANCE CONSTRAINTS

diffn [BeldiceanuContejean94]

```
diffn([[quad-[[ori-2, siz-2, end-4 ],[ori-1, siz-3, end-4]]],  
      [quad-[[ori-4, siz-4, end-8 ],[ori-3, siz-3, end-3]]],  
      [quad-[[ori-9, siz-2, end-11],[ori-4, siz-3, end-7]]]])
```

- non-overlapping between rectangles [BeldiceanuCarlsson01]
- non-overlapping between convex polytopes [BeldiceanuGuoThiel01]
- non-overlapping between non-convex polygon [RibeiroCarravilla04]

Graph Covering Constraints

VERTEX-PARTITIONING OF A DIRECTED GRAPH

cycle [BeldiceanuContejean94] [Bourreau99]

```
cycle(2, [[index-1,succ-2],[index-2,succ-1],[index-3,succ-5],  
          [index-4,succ-3],[index-5,succ-4]])
```

- **temporal_path** [ILOG]
- **tree** [Beldiceanu00]
- **map** [Beldiceanu00]

Graph Constraints

ENFORCING A GIVEN GRAPH PROPERTY

clique [Fahle02] [Régin03]

```
clique(3, [[index-1, succ-{} ], [index-2, succ-{3,5}],  
          [index-3, succ-{2,5}], [index-4, succ-{} ], [index-5, succ-{2,3}]])
```

- **cutset** [FagesLal03]
- **k_cut** [Althaus]
- **path_from_to** [AlthausBockmayrElfKasperJungerMehlhorn02]

Relaxation and Proximity Constraints

Relaxation

soft_alldifferent_var [PetitRégineBessière01]

```
soft_alldifferent_var(3, [[var-5], [var-1], [var-9], [var-1], [var-5], [var-5]])
```

- **soft_alldifferent_ctr** [PetitRégineBessière01]
- **soft_global_cardinality** [BeldiceanuPetit04]
- **distance_between** [Beldiceanu00]

Cost Filtering Constraints

Decision variables + a cost variable

global_cardinality_with_costs [Régis99]

```
global_cardinality_with_costs([ [var-3], [var-3], [var-3], [var-6]],  
    [[val-3, noccurrence-3], [val-5, noccurrence-0], [val-6, noccurrence-1]],  
    [[i-1, j-1, c-4], [i-1, j-2, c-1], [i-1, j-3, c-7],  
     [i-2, j-1, c-1], [i-2, j-2, c-0], [i-2, j-3, c-8],  
     [i-3, j-1, c-3], [i-3, j-2, c-2], [i-3, j-3, c-1],  
     [i-4, j-1, c-0], [i-4, j-2, c-0], [i-4, j-3, c-6]], 14)
```

- `minimum_weight_alldifferent` [FocacciLodiMilano99] [Sellman02]
- `sum_of_weight_of_distinct_values` [BeldiceanuCarlssonThiel02]

PREREQUISITE

GENERAL INTRODUCTION

GRAPH BASED DESCRIPTION



– Introduction

- Describing things
- Graph-Based Reformulation
- Normalisation according a given graph-class
- Bounds of graph parameters
- Filtering back from the bounds to the arcs
- Invariants linking several graph parameters
- Taking advantage from the graph structure
- Putting everything together

Goal

Derive in a **systematic** way a filtering algorithm **just** from the graph-based description

⇒ once you can describe the meaning of your constraint in term of graph properties you get a first filtering algorithm for free.

The Thesis

Constraint Propagation
=
Maintaining Graph Invariants

Focus first on formula (invariants)
and then on how to interpret these formula (algorithm)

The Whole Idea

- (1) Describe global constraints in terms of graph properties
- (2) Maintain graph invariants involving those graph properties
- (3) Build a data base of graph invariants
and bounds of graph parameters
- (4) Extract relevant invariants and bounds and interpret them

PREREQUISITE

GENERAL INTRODUCTION

GRAPH BASED DESCRIPTION

- Introduction
- **Describing things**
- Graph-Based Reformulation
- Normalisation according a given graph-class
- Bounds of graph parameters
- Filtering back from the bounds to the arcs
- Invariants linking several graph parameters
- Taking advantage from the graph structure
- Putting everything together



Motivations for a Classification of Global Constraints

- Find out the basic **constituents** of the global constraint,
- Classify the **properties** of each basic constituent,
- Understand how properties **interact**.

Main Idea of the Classification

Global Constraints as:

Graph Properties

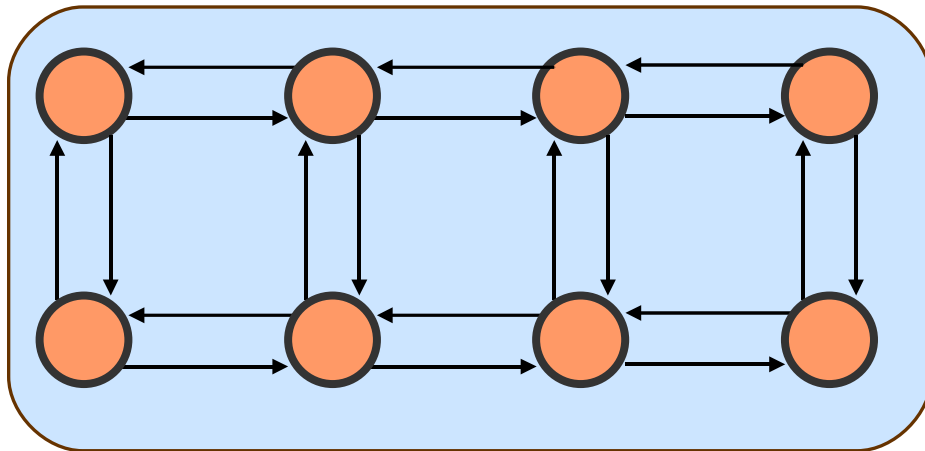
on **Structured Network**

of **Elementary Constraints**

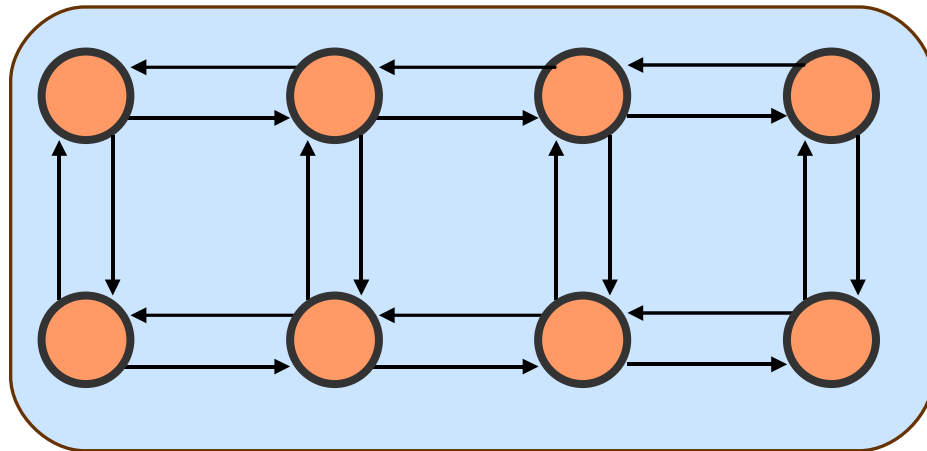
of the **Same Type**

Modelling a Global Constraint (1)

- Reformulation as a constraint network of primitive constraints

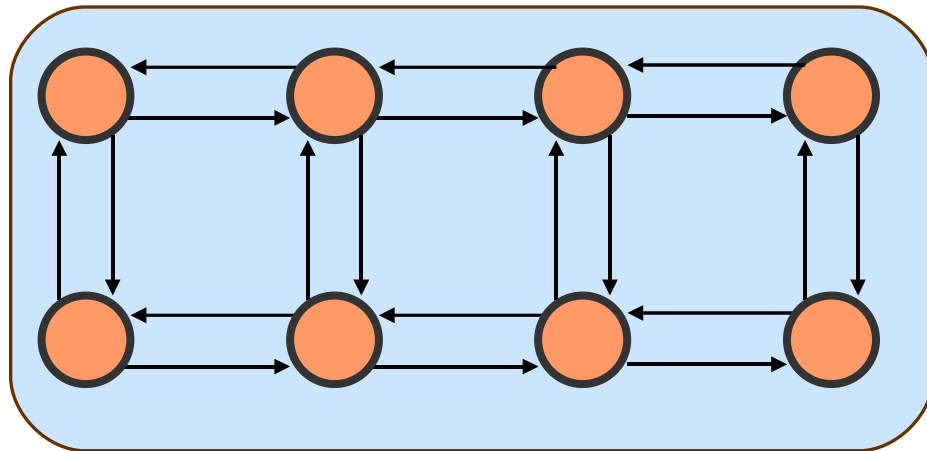


Modelling a Global Constraint (1)



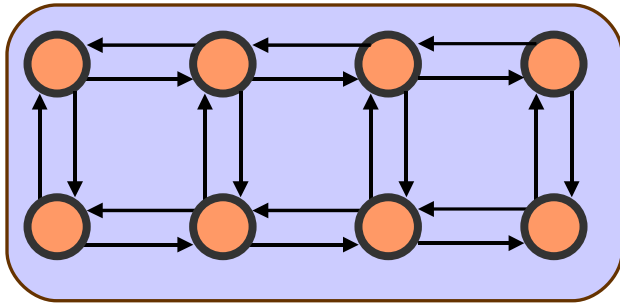
- Reformulation as a constraint network of primitive constraints
- Vertices correspond to the variables of the global constraint

Modelling a Global Constraint (1)



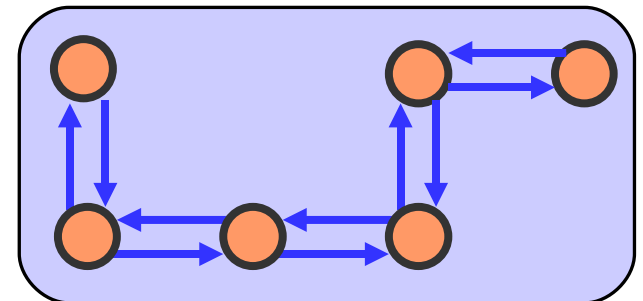
- Reformulation as a constraint network of primitive constraints
- Vertices correspond to the variables of the global constraint
- **During search some variables get fixed, and as a consequence, some primitive constraints get satisfied and some other get violated**

Modelling a Global Constraint (2)



Initial graph

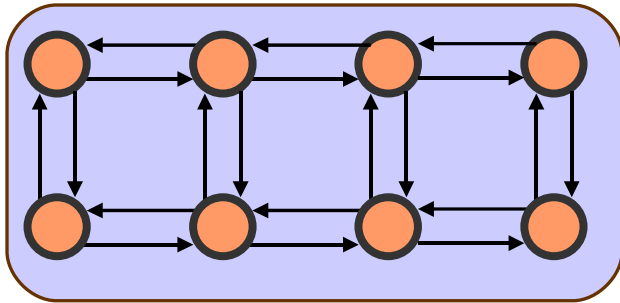
- When variables get fixed:
- some constraints are **violated**
 - some other are **satisfied**



Final graph

Samos Summer

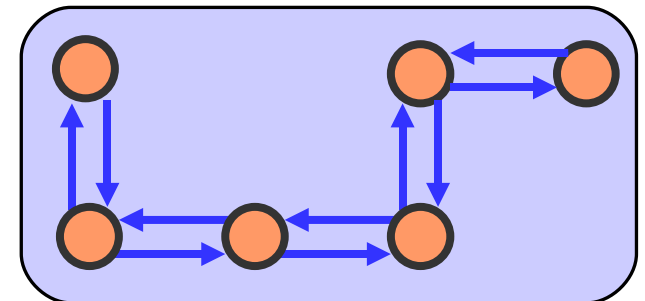
Modelling a Global Constraint (2)



Initial graph

- When variables get fixed:
- some constraints are **violated**
 - some other are **satisfied**

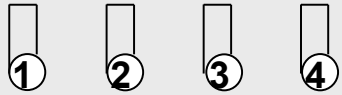
Semantic of a global constraint :
the **final graph** has to verify a set
of **properties**



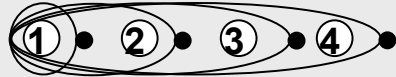
Final graph

Samos Summer

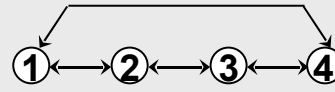
Generating the Initial Graph: Examples of Graph Generator



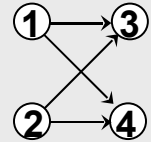
SELF



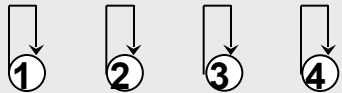
PATH_1



CYCLE



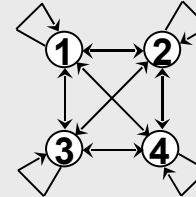
PRODUCT



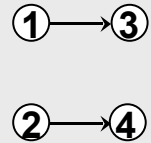
LOOP



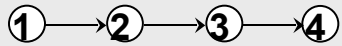
PATH_N



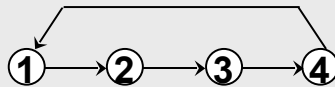
CLIQUE



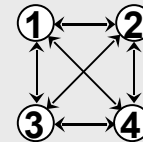
PRODUCT(=)



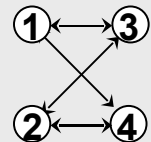
PATH



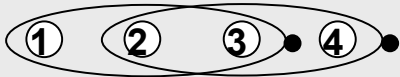
CIRCUIT



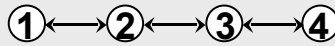
CLIQUE(≠)



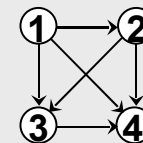
SYMMETRIC_PRODUCT



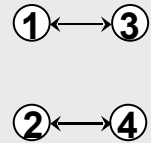
PATH



CHAIN



CLIQUE(<)



SYMMETRIC_PRODUCT(=)

Graph Parameters Involved in the Properties to Check

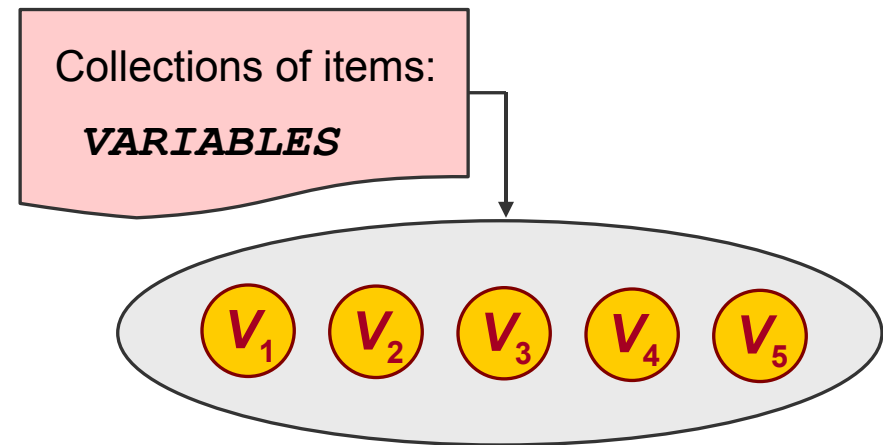
- **NVERTEX** : cardinality of $V(G)$
- **NEDGE** : cardinality of $E(G)$
- **NSOURCE** : number of vertices without any predecessor
- **NSINK** : number of vertices without any successor
- **NCC** : number of connected components of graph G
- **MIN_NCC** : number of vertices of the smallest c.c. of graph G
- **MAX_NCC** : number of vertices of the largest c.c. of graph G
- **NSCC** : number of strongly connected components of graph G
- **MIN_NSCC** : number of vertices of the smallest s.c.c. of graph G
- **MAX_NSCC** : number of vertices of the largest s.c.c. of graph G
- **MAX_IN_DEGREE** : number of predecessors of the vertex of G that has the maximum number of predecessors (without considering loops)

nvalue(NVAL, VARIABLES)

• ARGUMENT	:	NVAL	:	dvar
		VARIABLES: collection(var-dvar)		
• RESTRICTION (S)	:	NVAL \geq 0		
		NVAL \leq VARIABLES		
		required(VARIABLES.var)		
• VERTEX GENERATOR	:	VARIABLES		
• EDGE GENERATOR	:	CLIQUE		
• EDGE ARITY	:	2		
• EDGE CONSTRAINT	:	VARIABLES.var[1] = VARIABLES.var[2]		
• GRAPH PROPERTY	:	NSCC = NVAL		
• GRAPH CLASS	:	equivalence		

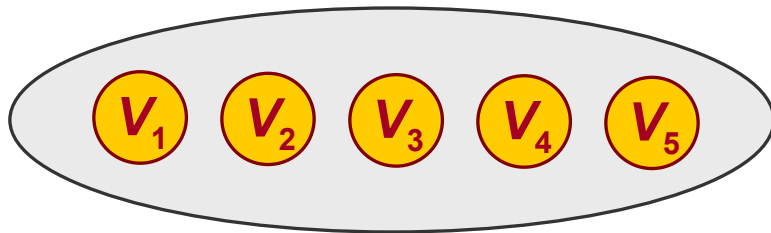
nvalue(4, { var-3, var-1, var-7, var-1, var-6 })

•	ARGUMENT	:	NVAL	:	dvar
			VARIABLES: collection(var-dvar)		
•	RESTRICTION (S)	:	NVAL \geq 0		
			NVAL \leq VARIABLES		
			required(VARIABLES.var)		
•	VERTEX GENERATOR	:	VARIABLES		
•	EDGE GENERATOR	:	CLIQUE		
•	EDGE ARITY	:	2		
•	EDGE CONSTRAINT	:	VARIABLES.var[1] = VARIABLES.var[2]		
•	GRAPH PROPERTY	:	NSCC = NVAL		
•	GRAPH CLASS	:	equivalence		

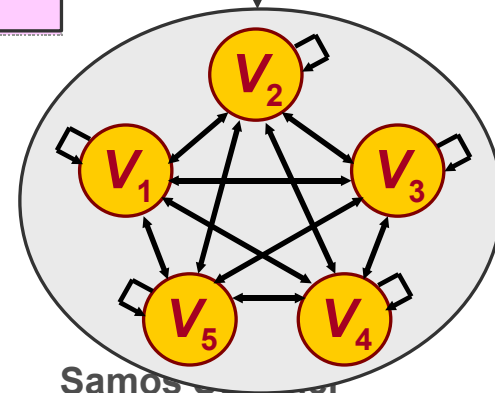


• ARGUMENT	:	NVAL	:	dvar
				VARIABLES: collection(var-dvar)
• RESTRICTION (S)	:	NVAL ≥ 0		
		NVAL ≤ VARIABLES		
		required(VARIABLES.var)		
• VERTEX GENERATOR	:	VARIABLES		
• EDGE GENERATOR	:	CLIQUE		
• EDGE ARITY	:	2		
• EDGE CONSTRAINT	:	VARIABLES.var[1] = VARIABLES.var[2]		
• GRAPH PROPERTY	:	NSCC = NVAL		
• GRAPH CLASS	:	equivalence		

Collections of items:
VARIABLES

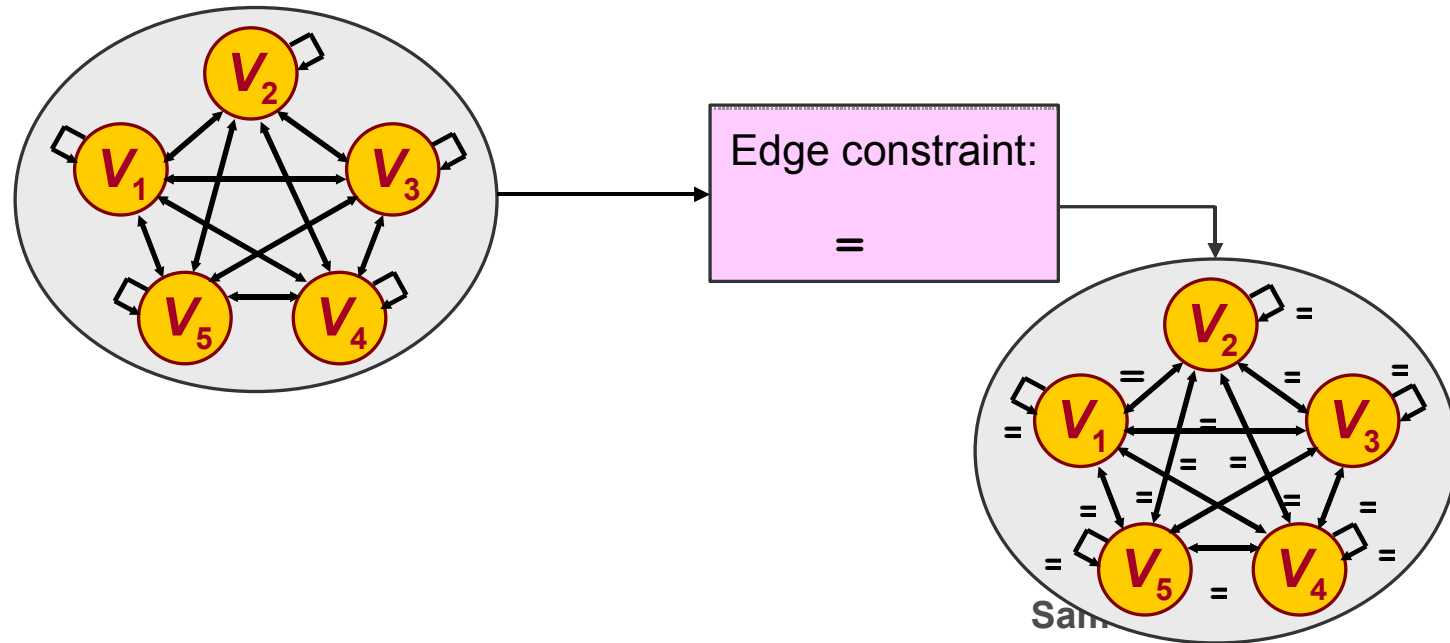


Edge generator:
CLIQUE



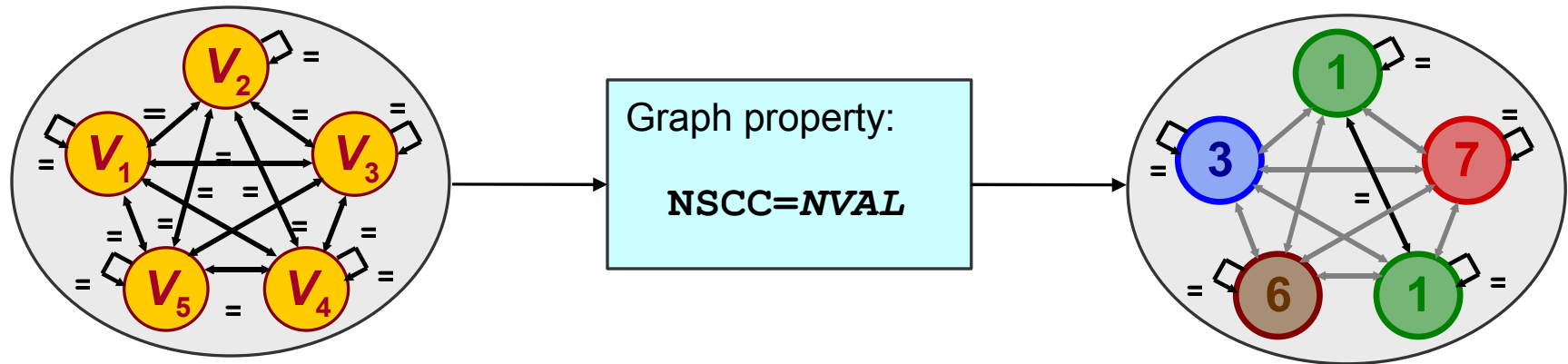
Samos C. et al.

- **ARGUMENT** : *NVAL* : *dvar*
VARIABLES: *collection(var-dvar)*
- **RESTRICTION (S)** : *NVAL* ≥ 0
NVAL $\leq |VARIABLES|$
required(VARIABLES.var)
- **VERTEX GENERATOR** : *VARIABLES*
- **EDGE GENERATOR** : *CLIQUE*
- **EDGE ARITY** : 2
- **EDGE CONSTRAINT** : *VARIABLES.var[1] = VARIABLES.var[2]*
- **GRAPH PROPERTY** : *NSCC = NVAL*
- **GRAPH CLASS** : *equivalence*



Sam

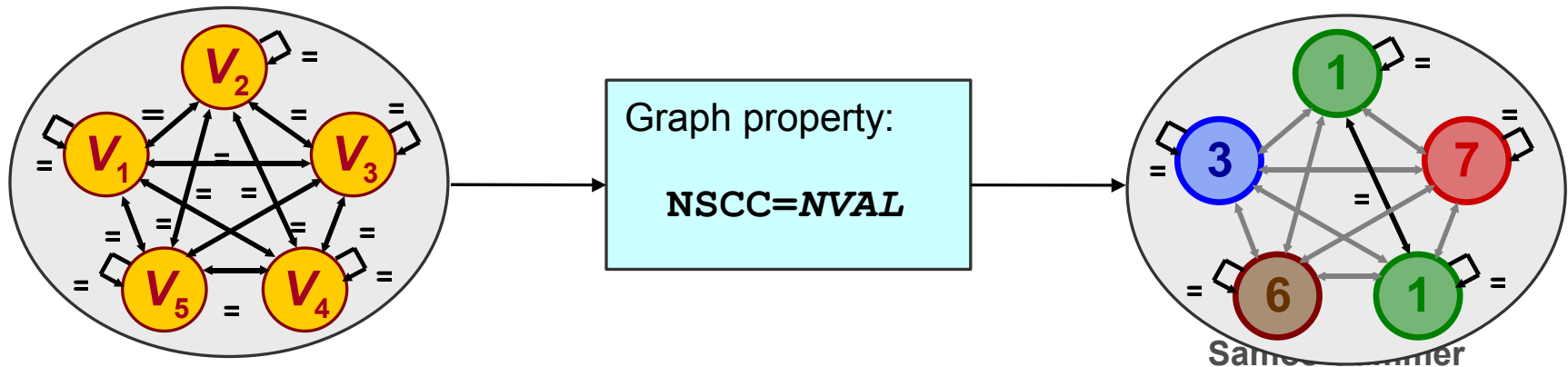
- **ARGUMENT** : *NVAL* : *dvar*
VARIABLES: *collection(var-dvar)*
- **RESTRICTION (S)** : *NVAL* ≥ 0
NVAL $\leq |VARIABLES|$
required(VARIABLES.var)
- **VERTEX GENERATOR** : *VARIABLES*
- **EDGE GENERATOR** : *CLIQUE*
- **EDGE ARITY** : 2
- **EDGE CONSTRAINT** : *VARIABLES.var[1] = VARIABLES.var[2]*
- **GRAPH PROPERTY** : *NSCC = NVAL*
- **GRAPH CLASS** : *equivalence*



nvalue(4, { *var-3*, *var-1*, *var-7*, *var-1*, *var-6* })

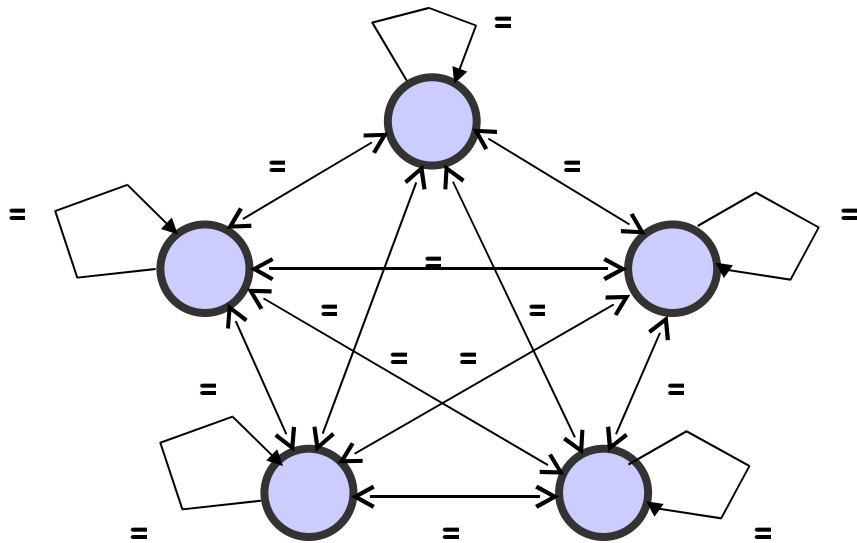
- **ARGUMENT** : *NVAL* : *dvar*
VARIABLES: *collection(var-dvar)*
- **RESTRICTION (S)** : *NVAL* ≥ 0
NVAL $\leq |VARIABLES|$
required(VARIABLES.var)
- **VERTEX GENERATOR** : *VARIABLES*
- **EDGE GENERATOR** : *CLIQUE*
- **EDGE ARITY** : 2
- **EDGE CONSTRAINT** : *VARIABLES.var[1] = VARIABLES.var[2]*
- **GRAPH PROPERTY** : *NSCC = NVAL*
- **GRAPH CLASS** : *equivalence*

Induced by the edge generator (CLIQUE)
and the edge constraint (=)

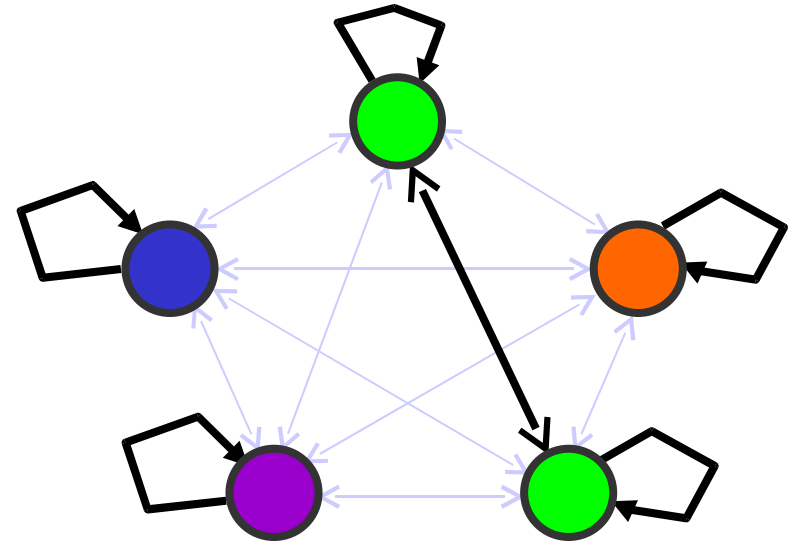


nvalue example (end)

Solution : `nvalue(4, { var-3, var-1, var-7, var-1, var-6 })`



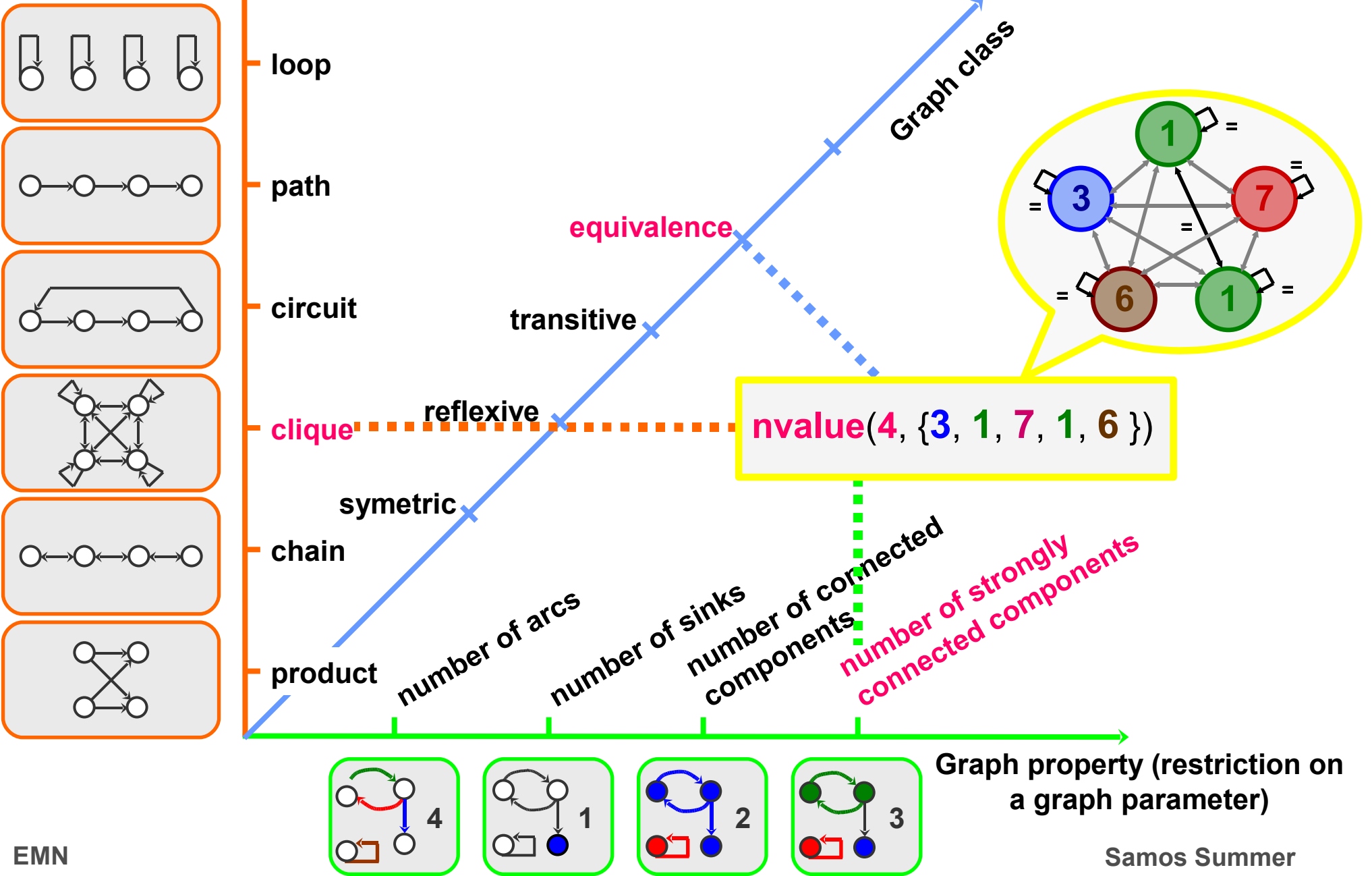
Initial Graph



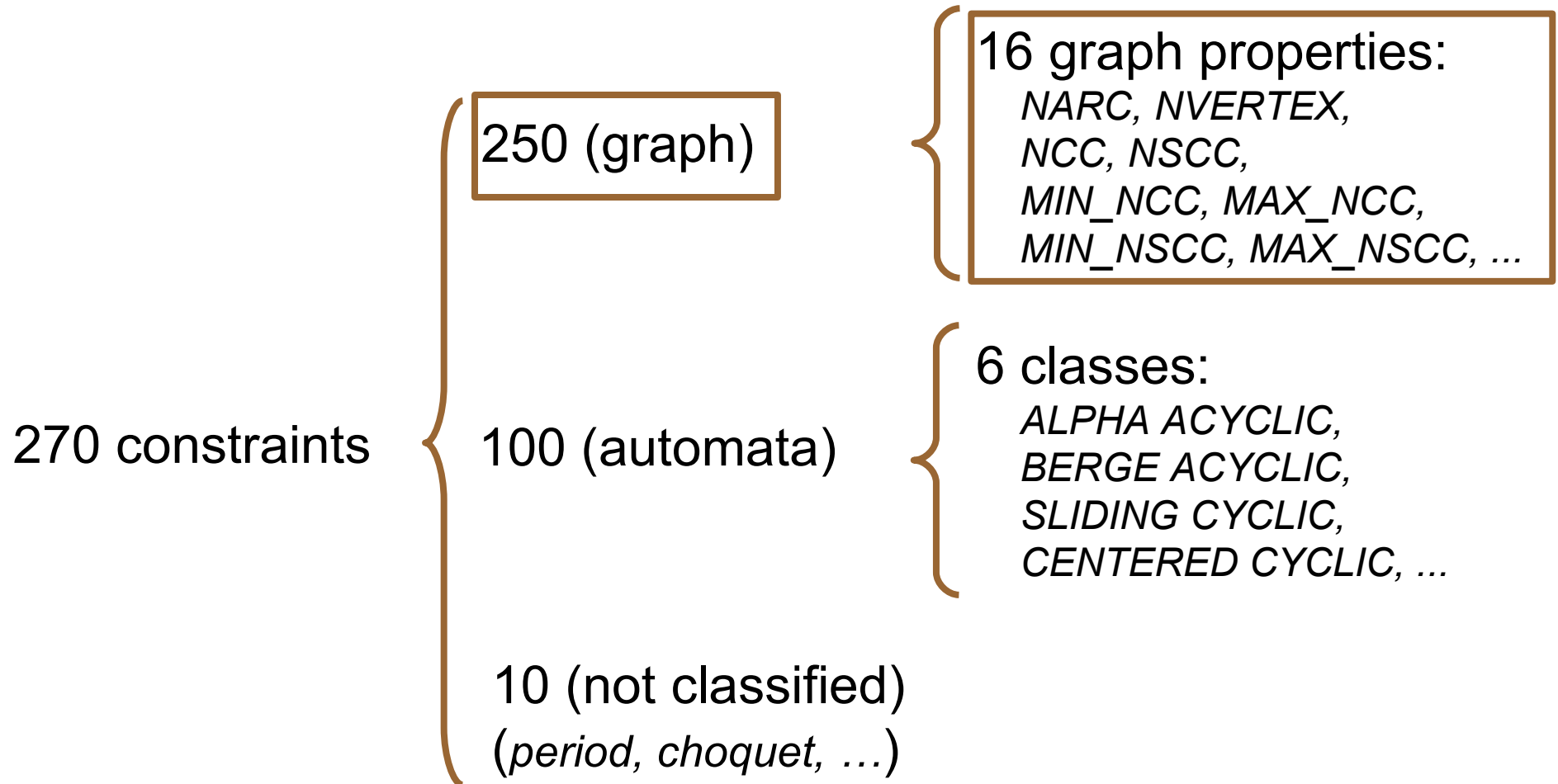
Final Graph

Structure of the initial graph

Global Constraint "Space"



A Catalog of Global Constraints



Link: <http://www.emn.fr/x-info/sdemasse/gccat/>

Other Features for Describing the Meaning of Global Constraints

- Set variables
- Constraints on several (related) graphs
- Set generators
- Derived collections

Using Set Variables (finite set of integers)

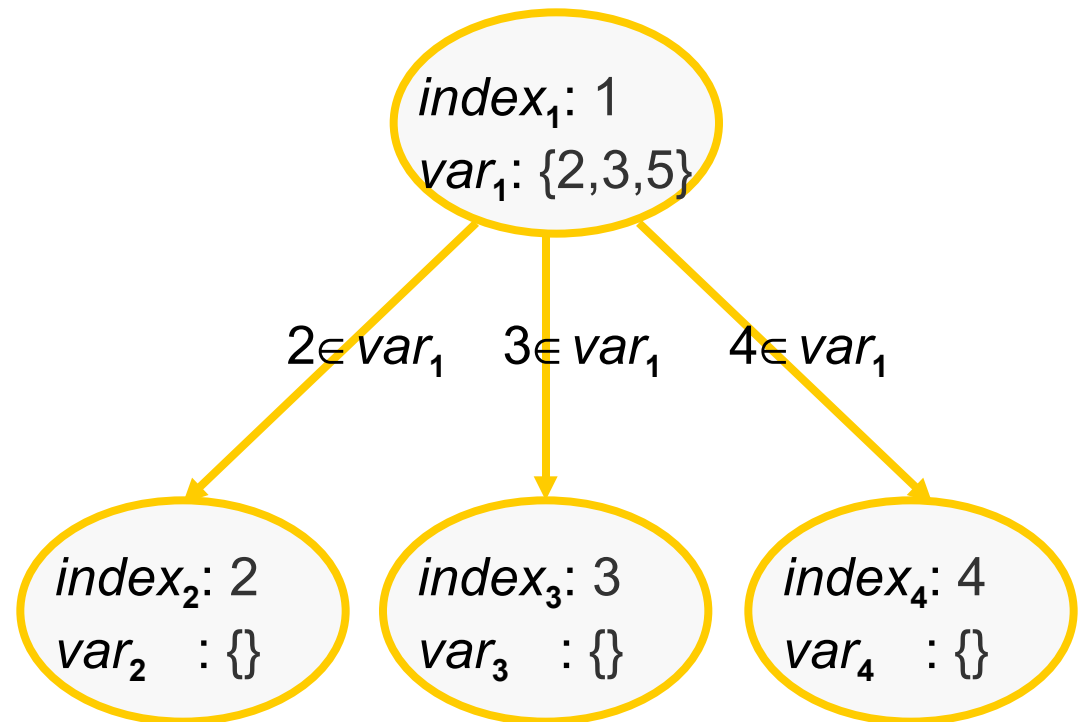
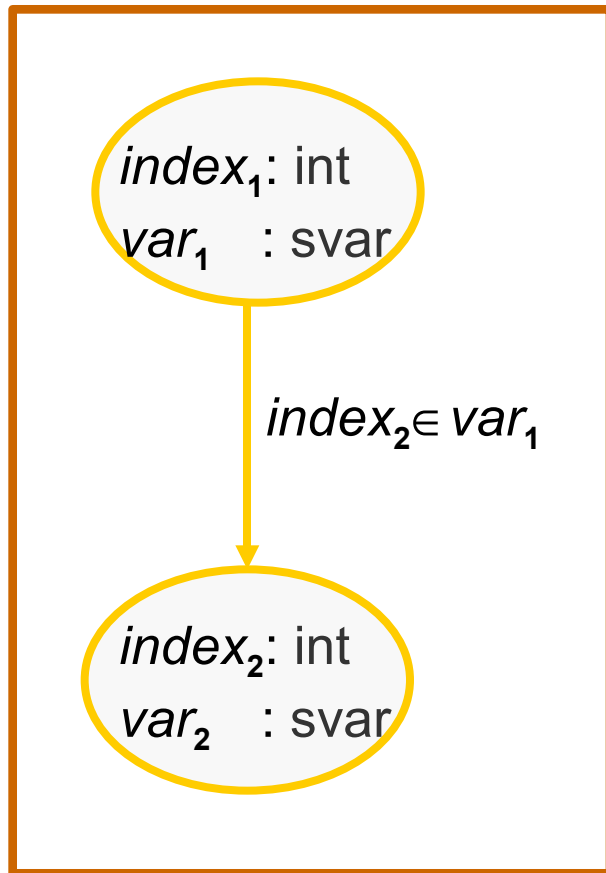
- 1) Exists since a **long** time (both in academy and in industrial solvers)
- 2) Has a **discrete** nature (as standard domain variables)
- 3) **Expressive** (avoid models with artificial variables)
- 4) Recently some suggestions of **global constraints with set variables**
- 5) Within linear programming and constraint programming a **typical** constraint is: *for a given graph selects a subset of arcs so that a given graph property holds*

Inserting Set Variables

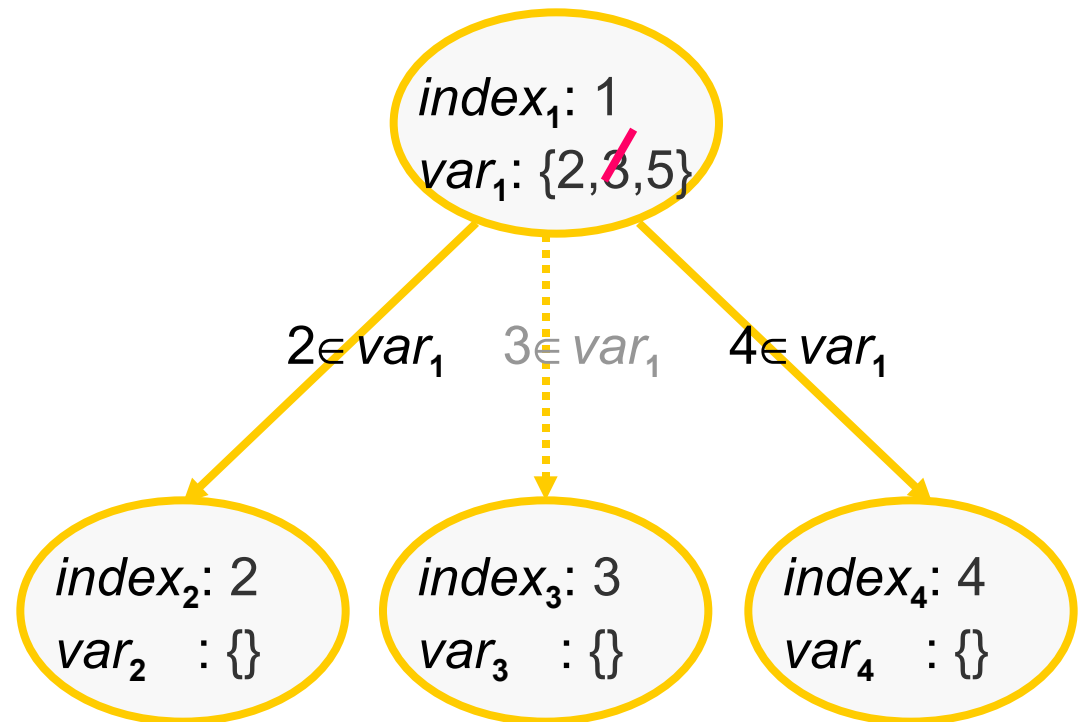
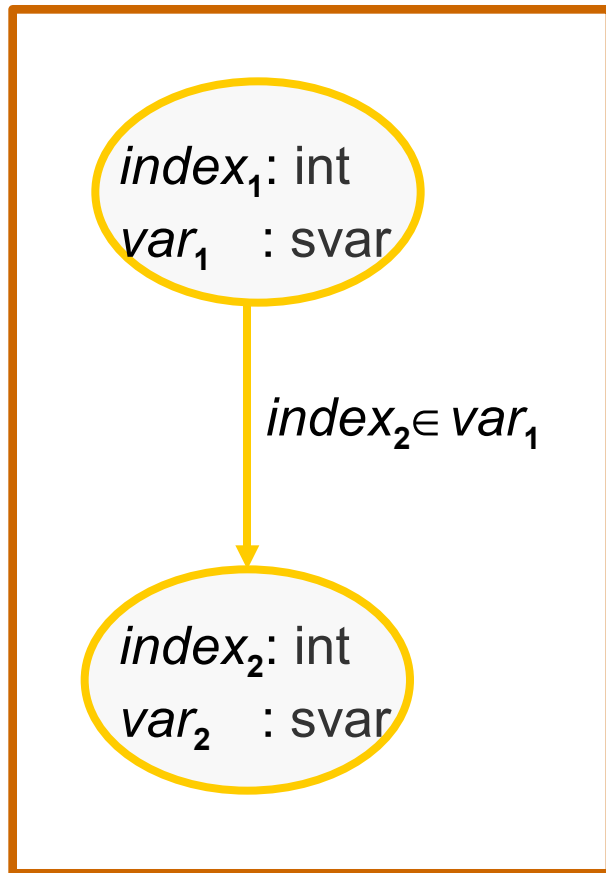
Has to introduce:

- 1) A new basic data type: finite set of integers
- 2) Set variables
- 3) Elementary constraints over set variables

Typical Arc Constraint



Typical Arc Constraint



Example of Global Constraint with Set Variables

Given a directed graph select a subset of vertices so that the corresponding sub-graph does not contain any circuit.
(F.Fages)

Example of Global Constraint with Set Variables

Symbolic **C**onstraints in **I**nteger **L**inear **P**rogramming
[E.Althaus,A.Bockmayr,M.Elf,M.Jünger,T.Kasper,K.Mehlhorn]
<http://www.mpi-sb.mpg.de/SCIL/>

StronglyConnected

"This symbolic constraints takes as arguments an directed graph G and a $\text{var_map}\langle\text{edge}\rangle X$. X has to map every edge of the graph to a binary variable.

The feasible assignments of the symbolic constraint are those where the vector of the variables associated with the edges of the graph is an incidence vector of a strongly connected subgraph of G ."

Constraints Over **Several** Final Graphs

PROBLEM Want to check graph properties on **several** final graphs

EXAMPLES

- **global cardinality** [RÉGIN]: impose restriction on number of occurrences of a value (depend on the value).
- **stretch** [PESANT]: impose minimum and maximum time that a value can occur in a consecutive way (depend on the value)

SOLUTION Introduce an **iterator** over the items of a collection for specifying in a **generic** way a set of:

- elementary constraints which are **pairwise incompatible** (before CTR & \neg CTR)
- graph properties.

Constraints Over Several Sets of Variables

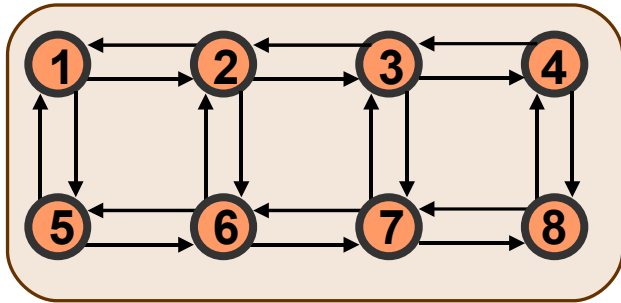
PROBLEM Very often we want to impose constraints on set of variables that are not originally known.

EXAMPLE

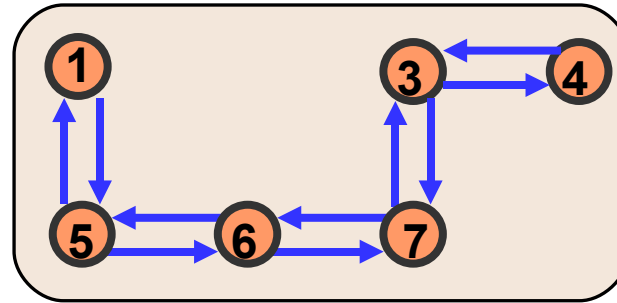
- **cumulative**: impose a **sum** constraint on the heights of the tasks that overlap (but since the tasks are not initially fixed **we cannot state directly a set of sum** constraints).

SOLUTION Introduce generators of set of vertices (of the final graph).

Example of Set Generator: Succ



Initial graph



Final graph

Succ: for each vertex of the final graph generates the set of its successors (as a collection of items) and imposes a constraint on this set.

5

1

6

5

7

3

6

4

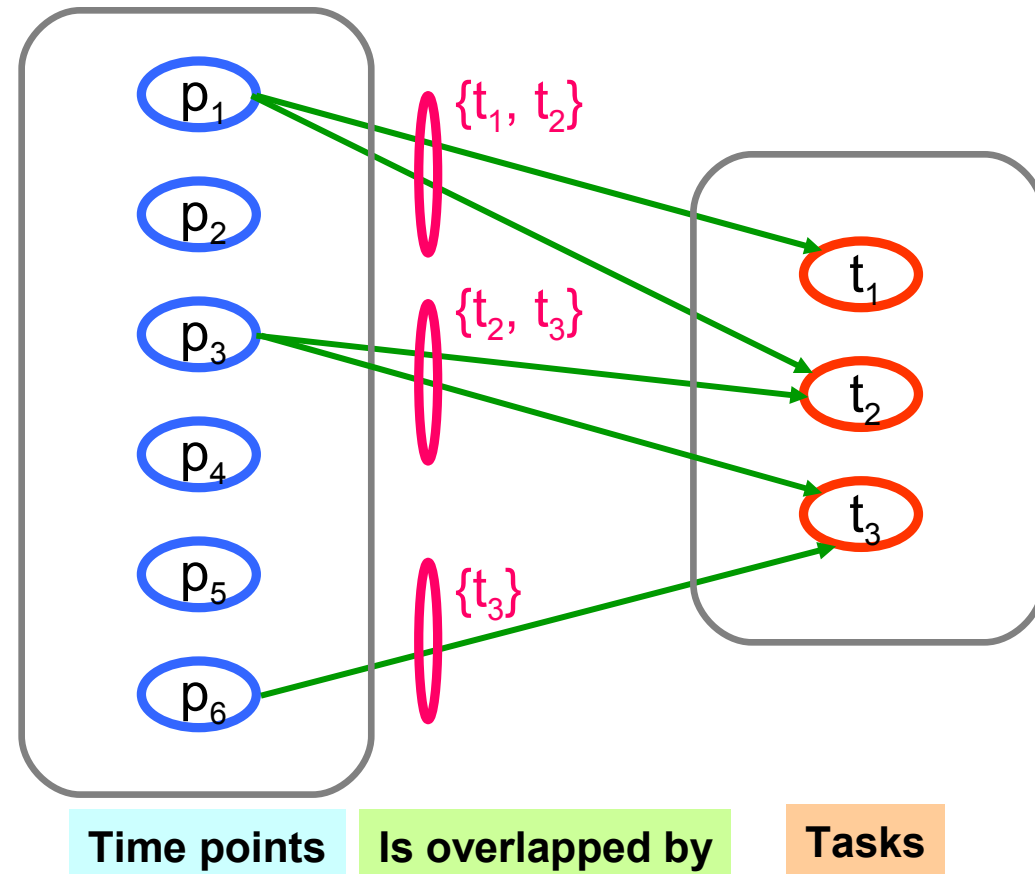
7

4

Force the total number of generated sets to be **polynomial**
in the number of vertices (tractability)

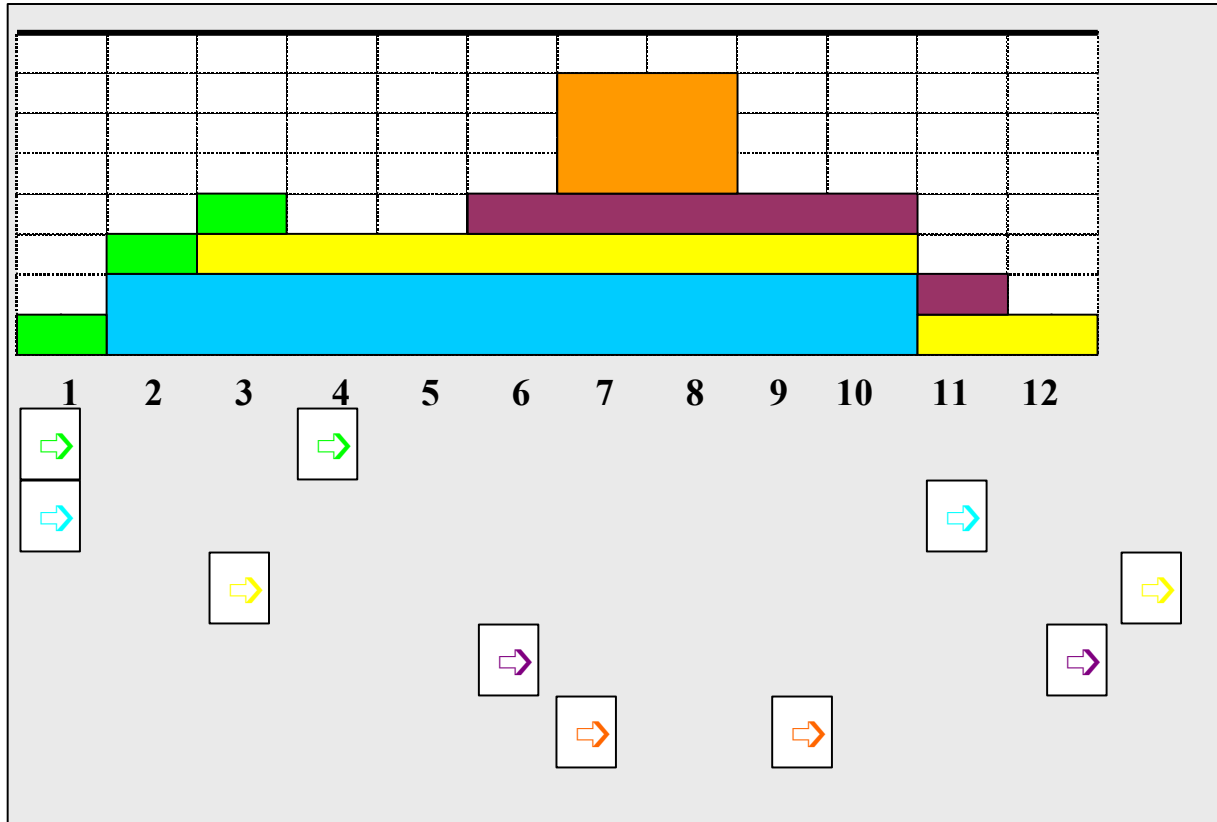
Model for cumulative

- ARC INPUT : Time points, Tasks
- ARC GENERATOR : product
- ARC ARITY : 2
- ARC CONSTRAINT : overlap
- SETS : **succ**



Which Time Points ?

The task origins and ends:



But the time points are not explicitly mentioned in the constraint?

Derived Collections

An optional extra field for describing a global constraint.

Purpose : **generates collections** from the parameters of the constraint.

Motivation: allows to simplify parameters of a constraint.

FROM element(**INDEX**,**TABLE**,**VALUE**)

INDEX : dvar

TABLE : collection(index-dvar,value-dvar)

VALUE: dvar

TO element(**ITEM**,**TABLE**)

ITEM : collection(index-dvar,value-dvar)

TABLE: collection(index-int ,value-int)

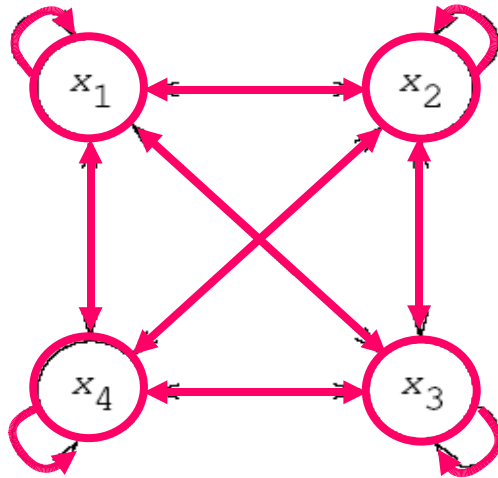
PREREQUISITE

GENERAL INTRODUCTION

GRAPH BASED DESCRIPTION

- Introduction
- Describing things
- **– Graph-Based Reformulation**
- Normalisation according a given graph-class
- Bounds of graph parameters
- Filtering back from the bounds to the arcs
- Invariants linking several graph parameters
- Taking advantage from the graph structure
- Putting everything together

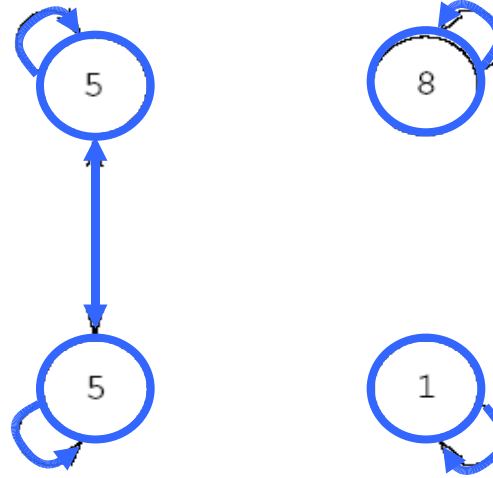
Intermediate Graph



(A)

Initial graph

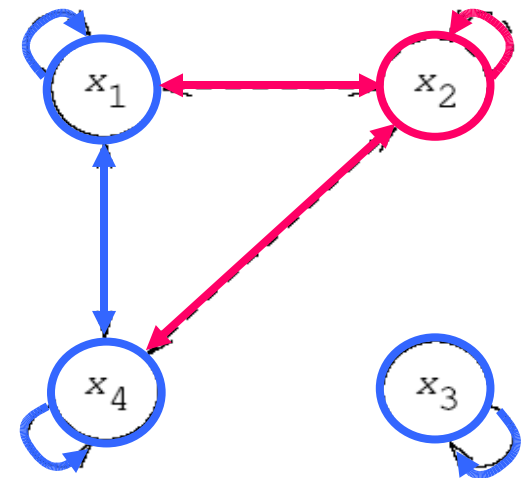
$nvalue(N, \{x_1, x_2, x_3, x_4\})$



(B)

Final graph

$nvalue(3, \{5, 8, 1, 5\})$



(C)

Intermediate graph

$nvalue(3, \{x_1, x_2, x_3, x_4\})$

$x_1 = x_4$

$x_1 \neq x_3$ $x_2 \neq x_3$ $x_3 \neq x_4$

Some vertices/arcs **belong for sure** to the final graph

Some vertices/arcs **may belong** to the final graph

(**T**-vertices, **T**-arcs)

(**U**-vertices, **U**-arcs)

Graph Based Filtering

Reasoning on **one** single graph property:

- (1) From the status of the vertices/arcs **get bounds on that property**
- (2) From limits on the bounds of that property **propagate back to status**

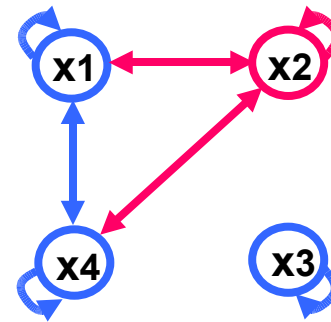
Reasoning on **several** graph properties on the same graph:

- (3) Use invariants linking several graph parameters

$$(1) \min(\mathbf{NSCC}) \geq 2 \quad \max(\mathbf{NSCC}) \leq 3$$

$$(2) \mathbf{NSCC} \geq 3 \Rightarrow \mathbf{x}_1 \neq \mathbf{x}_2, \quad \mathbf{x}_2 \neq \mathbf{x}_4$$

$$(3) \mathbf{NSCC} \geq \left\lceil \frac{\mathbf{NVERTEX}^2}{\mathbf{NARC}} \right\rceil \quad (\text{Turán, 1941})$$



From the Description of the Meaning of a Global Constraint to the Filtering

Key Idea: reformulate the definition into a system of constraints

Consider a global constraint $C(V_1, \dots, V_n, x_1, \dots, x_m)$ defined by:

- an initial graph $G_i(X_i, E_i)$,
- n graph properties P_k on V_k ($k \in [1, n]$) on any final graph G_f of C ,
- and eventually a graph class of G_f .

We will describe step by step the reformulation on an example

Consider the Example of *proper_forest* [CPAIOR 06]

PROBLEM

Cover an undirected graph G by a set of *NTREES* trees in such a way that each vertex of G belongs to one distinct tree (a *tree* is a connected graph without cycles that contains at least two vertices)

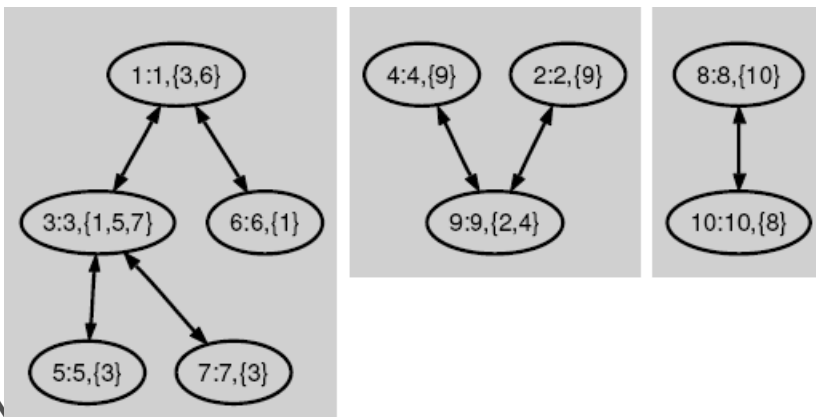
CONSTRAINT

```
proper_forest(NTREES, NODES)
```

```
NTREES : dvar
```

```
NODES : collection(index - int, neighbour - svar)
```

EXAMPLE



```
proper_forest 3, {
    {
        index - 1   neighbour - {3, 6},
        index - 2   neighbour - {9},
        index - 3   neighbour - {1, 5, 7},
        index - 4   neighbour - {9},
        index - 5   neighbour - {3},
        index - 6   neighbour - {1},
        index - 7   neighbour - {3},
        index - 8   neighbour - {10},
        index - 9   neighbour - {2, 4},
        index - 10  neighbour - {8}
    }
}
```

Initial graph of proper_forest

Arc input(s)	NODES
Arc generator	$CLIQUE(\neq) \mapsto \text{collection}(\text{nodes1}, \text{nodes2})$
Arc arity	2
Arc constraint(s)	$\text{in_set}(\text{nodes2.index}, \text{nodes1.neighbour})$

Reformulation (graph-property)

Introduces a domain variable for each graph parameter involved in the different graph properties and:

- (1) for each graph parameter states $ctr_{\mathcal{P}_l}(Vertex, Arc, P_l)$
- (2) for each graph property states the corresponding constraint.

EXAMPLE

Graph properties of *proper_forest*

- $NVERTEX = (NARC + 2 * NTREES) / 2$
- $NCC = NTREES$
- $NVERTEX = |NODES|$

(1) Creating variables

$NVERTEX \rightarrow V, ctr_{NVERTEX}(Vertex, Arc, V)$

$NARC \rightarrow A, ctr_{NVERTEX}(Vertex, Arc, A)$

$NCC \rightarrow C, ctr_{NVERTEX}(Vertex, Arc, C)$

(2) Stating constraints

$V = (A + 2 * NTREES) / 2$

$C = NTREES$

$V = |NODES|$

Reformulation: Representing the Intermediate Graph

To each vertex v_j of the initial graph G_i corresponds a 0-1 variable *vertex* _{j} (1 if the vertex belongs to G_f , 0 otherwise).

To each arc e_{jk} corresponds a 0-1 variable *arc* _{jk} as well as the constraint

$$arc_{jk} = 1 \Leftrightarrow \text{ctr}(x_j, x_k)$$

EXAMPLE

Arc input(s)	NODES
Arc generator	$CLIQUE(\neq) \mapsto \text{collection}(\text{nodes1}, \text{nodes2})$
Arc arity	2
Arc constraint(s)	$\text{in_set}(\text{nodes2.index}, \text{nodes1.neighbour})$

For each distinct pair of vertices of the initial graph G_i :

$$\text{Index}_2 \in \text{Neighbour}_1$$

Reformulation: Normalisation Constraint (*no isolated vertex in the final graph*)

$$vertex_j = \min(1, \sum_{\{k \mid e_{jk} \in E_{\mathcal{R}}\}} arc_{jk} + \sum_{\{k \mid e_{kj} \in E_{\mathcal{R}}\}} arc_{kj})$$

PREREQUISITE

GENERAL INTRODUCTION

GRAPH BASED DESCRIPTION

- Introduction
- Describing things
- Graph-Based Reformulation
- ➔ **– Normalisation according a given graph-class**
- Bounds of graph parameters
- Filtering back from the bounds to the arcs
- Invariants linking several graph parameters
- Taking advantage from the graph structure
- Putting everything together

Reformulation: Graph Class Constraints

The following constraint is satisfied if G_f belongs to the graph class $c_{\mathcal{R}}$:

$$ctr_{c_{\mathcal{R}}}(Vertex, Arc)$$

EXAMPLE

SYMMETRIC

For each arc e_{jk}

$$arc_{jk} = arc_{kj},$$

For each vertex x_j

$$vertex_j = \min(1, 2 \cdot \sum_{\{k \mid e_{jk} \in E_{\mathcal{R}}\}} arc_{jk}).$$

PREREQUISITE

GENERAL INTRODUCTION

GRAPH BASED DESCRIPTION

- Introduction
- Describing things
- Graph-Based Reformulation
- Normalisation according a given graph-class
- **– Bounds of graph parameters**
- Filtering back from the bounds to the arcs
- Invariants linking several graph parameters
- Taking advantage from the graph structure
- Putting everything together

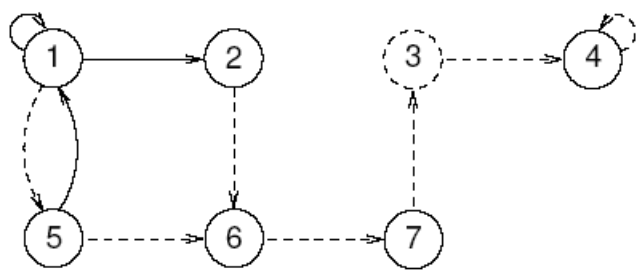
Bounds (no isolated vertices in the final graph)

Graph parameter	Bound
<u>NARC</u>	$ E_T + X_{T,-T} - \mu(\overrightarrow{G}(X_{T,-T}, E_U))$
<u>NARC</u>	$ E_{TU} $
<u>NVERTEX</u>	$ X_T + h(\overrightarrow{G}((X_{T,-T,-T}, X_{U,-T,T}), E_{U,T}))$
<u>NVERTEX</u>	$ X_{TU} $
<u>NCC</u>	$ cc_{[X_T \geq 1]}(\overrightarrow{G}(X_{TU}, E_{TU})) $
<u>NCC</u>	$ cc_{[E_T \geq 1]}(\overrightarrow{G}(X_T, E_T)) + \mu_l(\overrightarrow{G}_{rem})$
<u>NSCC</u>	$ scc_{[X_T \geq 1]}(\overrightarrow{G}(X_{TU}, E_{TU})) + h(G_{\underline{NSCC}}((Y, Z), E))$
<u>NSCC</u>	$ scc(\overrightarrow{G}(X_{TU}, E_T)) $
<u>NSINK</u>	$ sink_{[X_T =1]}(\overrightarrow{G}(X_{TU}, E_{TU})) + h(G'_r((Y, Z), E))$
<u>NSINK</u>	$ sink(\overrightarrow{G}(X_T, E_T)) + X_U - source_{[X_U =1]}(\overrightarrow{G}(X_{TU}, E_{TU})) - XP $

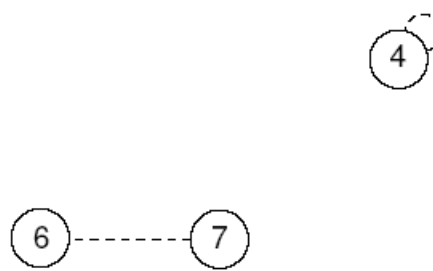
Minimum Number of Arcs

$$\underline{\text{NARC}} \geq \underbrace{|E_T|}_{3} + \underbrace{|X_{T,\neg T}|}_{3} - \underbrace{\mu(\overleftrightarrow{G}(X_{T,\neg T}, E_U))}_{1}$$

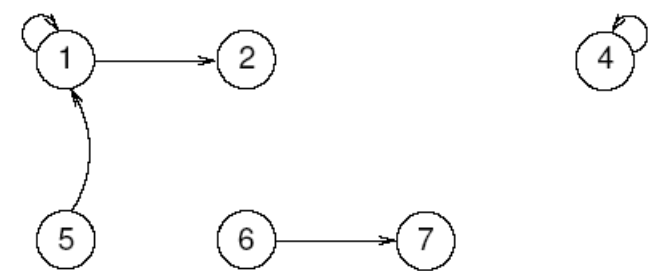
\swarrow cardinality of a maximum matching



Intermediate digraph



$\overleftrightarrow{G}(X_{T,\neg T}, E_U)$



A solution reaching 5 arcs

E_U : Arcs to be undetermined

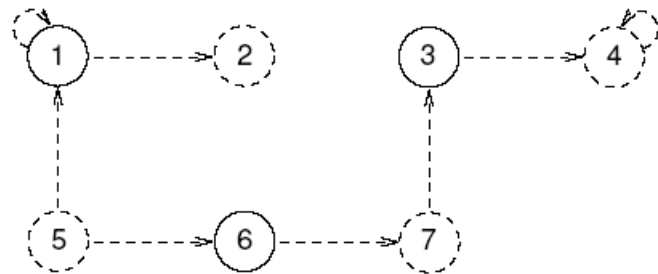
E_T : Arcs to be true

$X_{T,\neg T}$: Vertices to be true with all adjacent arcs to be undetermined

Minimum Number of Vertices

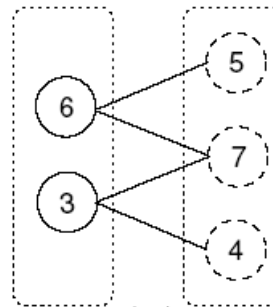
cardinality of a minimum hitting set

$$\underline{\text{NVERTEX}} \geq \underbrace{|X_T|}_3 + \underbrace{h(\overleftrightarrow{G}((X_{T,-T,-T}, X_{U,-T,T}), E_{U,T}))}_1$$

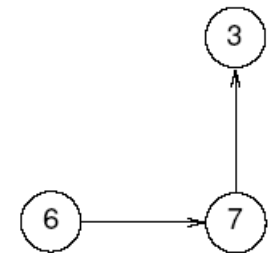


Intermediate digraph

$X_{T,-T,-T}$ $X_{U,-T,T}$



$\overleftrightarrow{G}((X_{T,-T,-T}, X_{U,-T,T}), E_{U,T})$



A solution reaching 4 vertices

$X_{T,-T,-T}$: True vertices that are not the extremity of any true arcs and that have not a true vertex as neighbour

$X_{U,-T,T}$: Undetermined vertices that has at least one true vertex as neighbour

$E_{U,T}$: Undetermined arcs such that at least one of their extremities is a true vertex

PREREQUISITE

GENERAL INTRODUCTION

GRAPH BASED DESCRIPTION

- Introduction
- Describing things
- Graph-Based Reformulation
- Normalisation according a given graph-class
- Bounds of graph parameters
- ➔ **– Filtering back from the bounds to the arcs**
- Invariants linking several graph parameters
- Taking advantage from the graph structure
- Putting everything together

Filtering Back From the Maximum Number of Arcs

If $\overline{\text{NARC}} = |E_T| + |X_{T,-T}| - \mu(\overleftrightarrow{G}(X_{T,-T}, E_U))$ then:

1. All arcs $(u, v) \in E_U$ such that u or v belongs to two, not necessarily distinct, connected components in $cc_{[|E_T| \geq 1]}(\overleftrightarrow{G}(X_T, E_T))$ can be turned into F -arcs.
2. All arcs in $E_{U,U,U}$ can be turned into F -arcs.
3. All U -vertices in $X_{U,-T,-T}$ (i.e., not connected to any T -vertex) can be turned into F -vertices.
4. For all edges $e = (u, v)$, $u \neq v$, of $\overleftrightarrow{G}(X_{T,-T}, E_U)$ such that $u, v \in X_{T,-T}$ and e does not belong to any maximum matching of $\overleftrightarrow{G}(X_{T,-T}, E_U)$, the corresponding arcs (u, v) and (v, u) of $\overleftrightarrow{G}(X_{T,-T}, E_U)$ can be turned into F -arcs.
5. For all edges $e = (u, v)$, $u \neq v$, of $\overleftrightarrow{G}(X_{T,-T}, E_U)$ such that $u, v \in X_{T,-T}$ and e belongs to all maximum matchings of $\overleftrightarrow{G}(X_{T,-T}, E_U)$, if a unique arc (u, v) or (v, u) in $\overleftrightarrow{G}(X_{T,-T}, E_U)$ corresponds to e then this arc can be turned to a T -arc.
6. All arcs $e = (u, v) \in E_U$ such that $u \in X_{T,-T}$ is saturated in all maximum matchings of $\overleftrightarrow{G}(X_{T,-T}, E_U)$ and $v \in X_U$ can be turned into F -arcs.
7. All loops $e = (u, u) \in E_U$ such that $u \in X_{T,-T}$ is saturated in all maximum matchings of $\overleftrightarrow{G}(X_{T,-T}, E_U)$ can be turned into F -arcs.

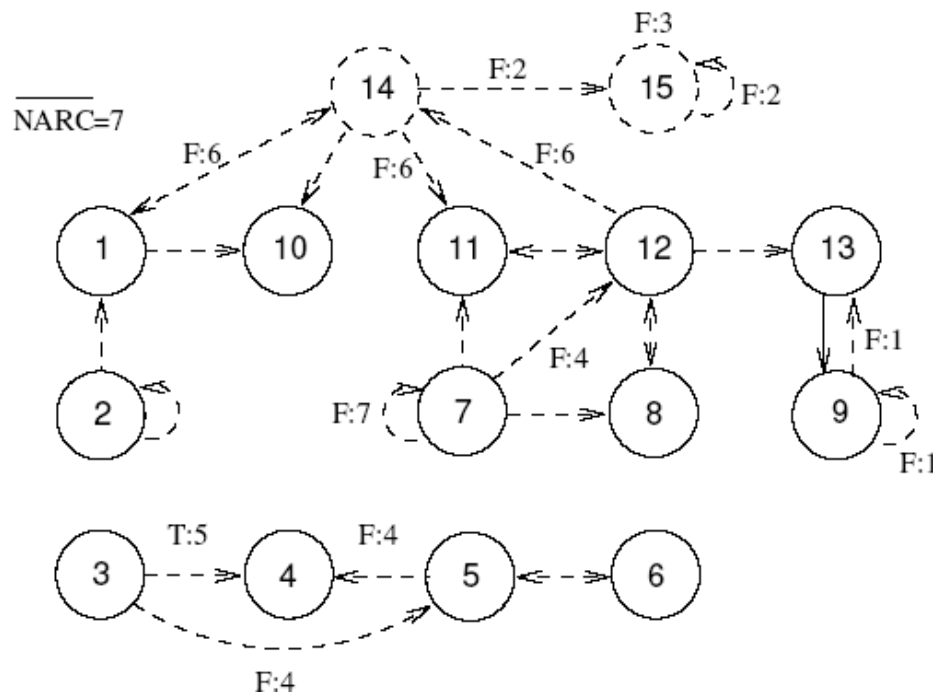
Filtering Back From the Maximum Number of Arcs

If $\overline{\text{NARC}} = |E_T| + |X_{T,-T}| - \mu(\overleftrightarrow{G}(X_{T,-T}, E_U))$ then:

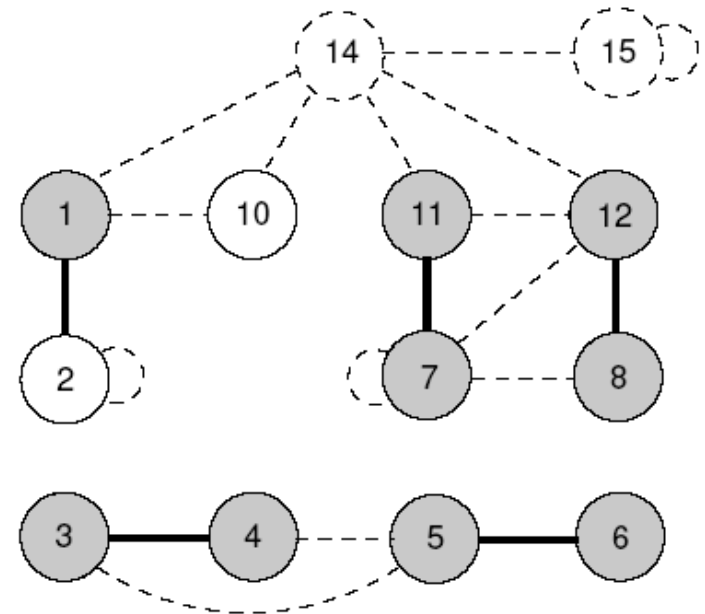
1

11

5



Intermediate digraph



$\overleftrightarrow{G}(X_{T,-T}, E_U)$

PREREQUISITE

GENERAL INTRODUCTION

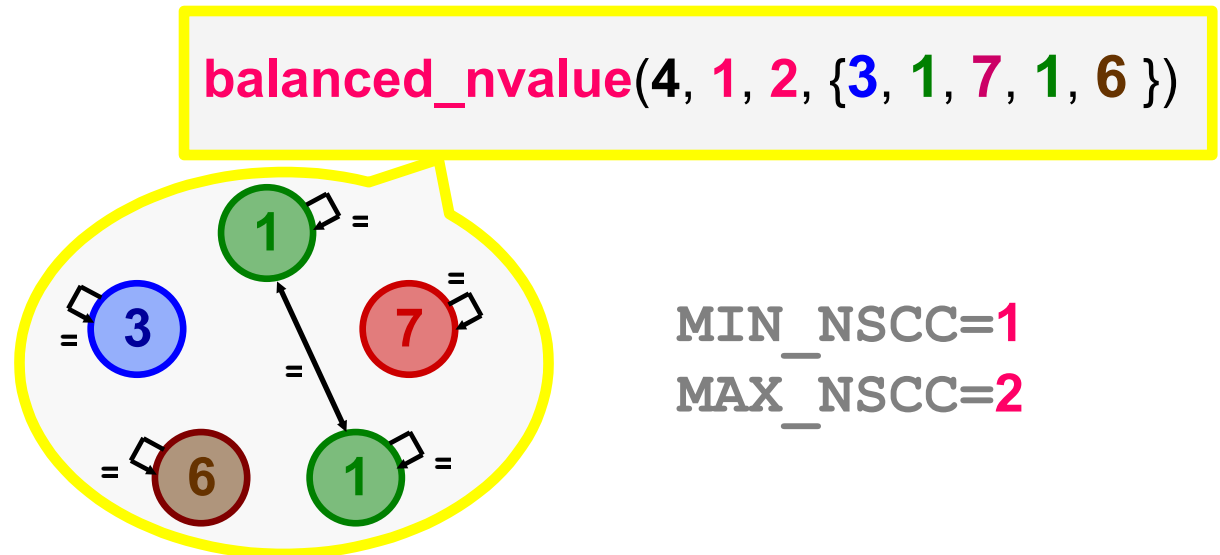
GRAPH BASED DESCRIPTION

- Introduction
- Describing things
- Graph-Based Reformulation
- Normalisation according a given graph-class
- Bounds of graph parameters
- Filtering back from the bounds to the arcs
- ➔ **– Invariants linking several graph parameters**
- Taking advantage from the graph structure
- Putting everything together

Motivation

A lot of global constraints use **more than one** graph parameter.

Example: consider the *nvalue* constraint and suppose we want a *balanced assignment* by restricting the min. and max. numbers of occurrences of any value effectively used.



This involves constraining the **MIN_NSCC** and **MAX_NSCC** graph parameters.

Idea

A lot of global constraints use more than one graph parameters.

But graph parameters are not independent,
they are related by graph invariants.

Graph invariants have been collected in a database.

Given a global constraint C specified in terms of graph parameters,
the relevant graph invariants form necessary conditions for C .

Example: in the context of *balanced_nvalue* you have:

$$\begin{aligned} \text{NVERTEX} &\leq \max(\text{NSCC} - 1, 0) * \text{MAX_NSCC} + \text{MIN_NSCC} \\ \text{NVERTEX} &\geq \max(\text{NSCC} - 1, 0) * \text{MIN_NSCC} + \text{MAX_NSCC} \end{aligned}$$

A data base of graph invariants

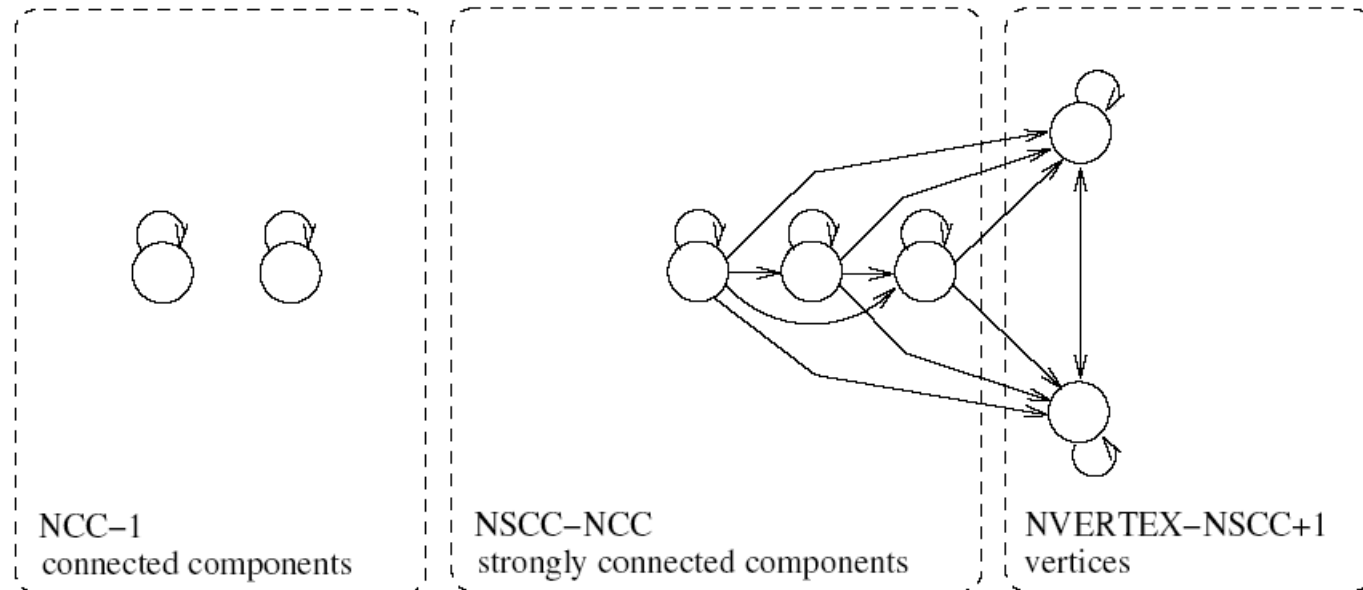
Indexed on: graph class (see later) and graph parameters mentioned by the constraint of interest.

#graphs	#GC	#invariants
1	1	13
1	2	50
1	3	34
1	4	12
1	5	2
2	2	10
2	3	10
2	4	6
2	5	16
2	6	4

Example of Graph Invariant

$$\text{NARC} \leq \text{NCC} - 1 + (\text{NVERTEX} - \text{NSCC} + 1) \cdot (\text{NVERTEX} - \text{NCC} + 1) + \frac{(\text{NSCC} - \text{NCC} + 1) \cdot (\text{NSCC} - \text{NCC})}{2}$$

Pattern that achieves the maximum number of **arcs** for a fixed number of **vertices**, a fixed number of **connected components** and a fixed number of **strongly connected components**



Formula and family of extreme graphs

Graph invariants are nice since:

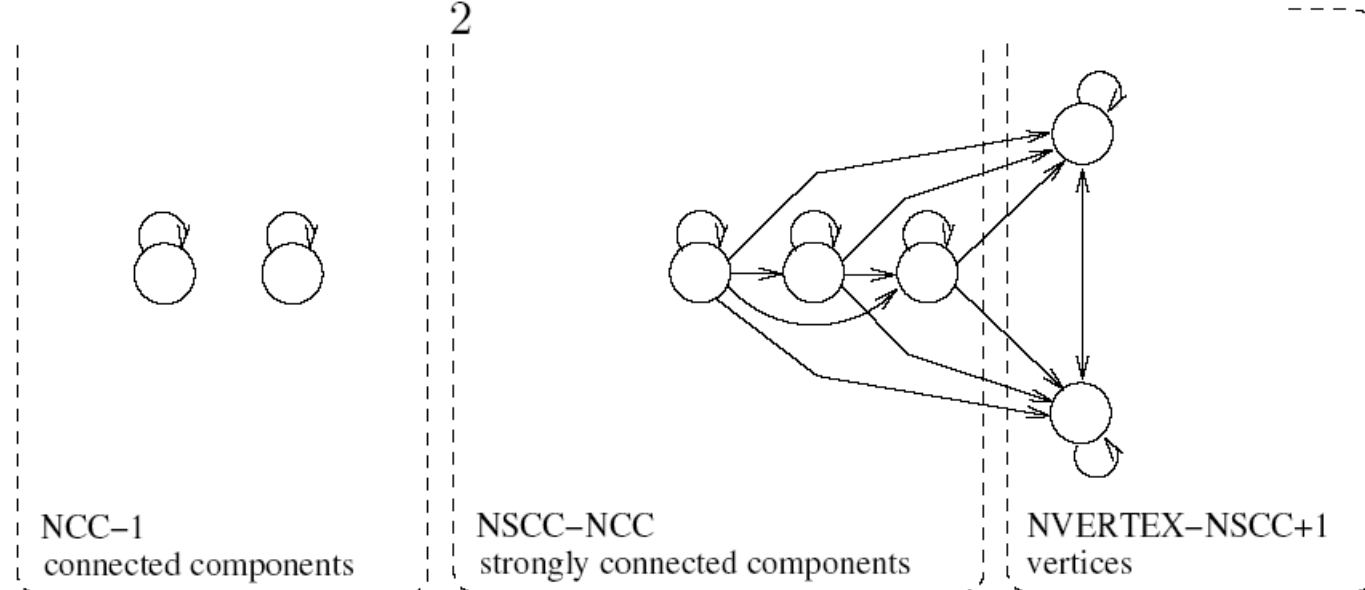
- they express a clean and universal piece of knowledge,
- they are somehow much more declarative than an algorithm.

But still the proof of a formula contains more information than the formula itself (very often the proof characterises a family of extreme graphs that would be useful for performing some constraint propagation)

$$\text{NARC} \leq \text{NCC} - 1 + (\text{NVERTEX} - \text{NSCC} + 1) \cdot (\text{NVERTEX} - \text{NCC} + 1) + \frac{(\text{NSCC} - \text{NCC} + 1) \cdot (\text{NSCC} - \text{NCC})}{2}$$

QUESTION:

Would it be possible to get an explicit description of families of extreme graphs that are behind some formula?



Invariants Constructed from a Disjunction

Consider the graph parameters P_1, P_2, \dots, P_n , a constant CST and assume we have:

$$(1) P_1 \leq CST \Rightarrow P_2 \leq f_1(P_3, \dots, P_n)$$

$$(2) P_1 > CST \Rightarrow P_2 \geq f_2(P_3, \dots, P_n)$$

then by eliminating P_1 and combining (1) and (2) we get:

$$P_2 \notin [f_1(P_3, \dots, P_n), f_2(P_3, \dots, P_n)]$$

EXAMPLE

- $NCC \leq 1 \Rightarrow NVERTEX \leq MIN_NCC$
- $NCC > 1 \Rightarrow NVERTEX \geq MIN_NCC + MAX_NCC$



$$NVERTEX \notin [MIN_NCC + 1, MIN_NCC + MAX_NCC - 1]$$

PREREQUISITE

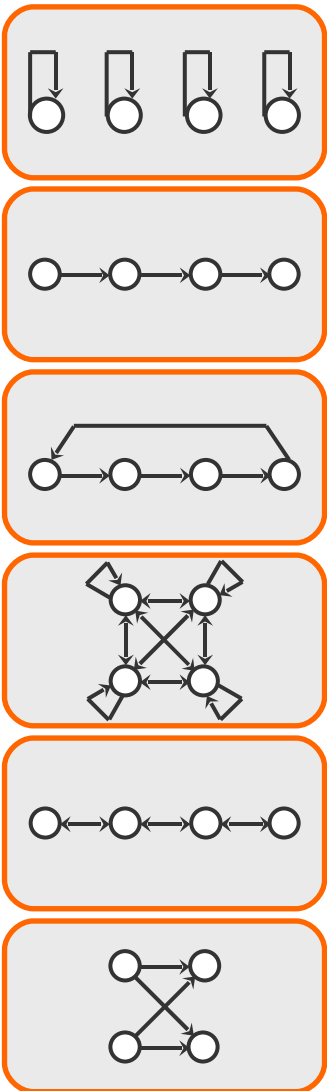
GENERAL INTRODUCTION

GRAPH BASED DESCRIPTION

- Introduction
- Describing things
- Graph-Based Reformulation
- Normalisation according a given graph-class
- Bounds of graph parameters
- Filtering back from the bounds to the arcs
- Invariants linking several graph parameters
- **– Taking advantage from the graph structure**
- Putting everything together

Structure of the initial graph

Invariants on Specific Graph Classes



loop

path

circuit

clique

chain

product

asymetric

equivalence

transitive

reflexive

symetric

$$\text{NARC} \leq \text{NVERTEX} - 1$$

number of arcs

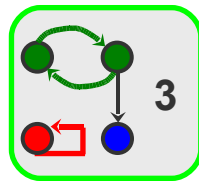
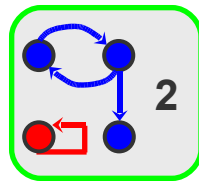
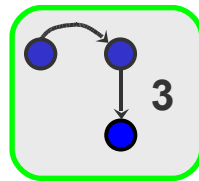
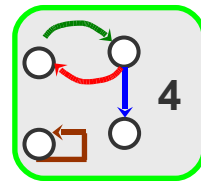
number of vertices

number of connected components

number of strongly connected components

Property of the final graph

Graph property



Idea

By considering the **structure of the final graph** we can get **tighter** bounds as well as **tighter** graph invariants.

- ▶ A general graph invariant:

$$\text{NARC} \leq \text{NVERTEX}^2$$

- ▶ A tighter graph invariant that holds for arc generator PATH:

$$\text{NARC} \leq \text{NVERTEX} - 1$$

Graph Class

binary constraint	arc generator
<ul style="list-style-type: none"> • acyclic • bipartite • equivalence • no_loop • one_succ 	<ul style="list-style-type: none"> • CHAIN • CIRCUIT • PATH • PRODUCT • SYMMETRIC_PRODUCT

nvalue

• ARGUMENT	:	NVAL	:	dvar
				<i>VARIABLES: collection(var-dvar)</i>
• RESTRICTION (S)	:	NVAL ≥ 0		
				<i>NVAL ≤ VARIABLES </i>
				<i>required(VARIABLES.var)</i>
• VERTEX GENERATOR	:	VARIABLES		
• EDGE GENERATOR	:	CLIQUE		
• EDGE ARITY	:	2		
• EDGE CONSTRAINT	:	<i>VARIABLES.var[1]=VARIABLES.var[2]</i>		
• GRAPH PROPERTY	:	NSCC = NVAL		

FINAL GRAPH
of *nvalue* :

a set of cliques

Examples of Tighter Graph Invariants

$$\text{MIN_NCC} > 0 \Rightarrow \text{NARC} \geq \max(1, \text{MIN_NCC} - 1)$$

$$\text{symmetric}: \text{MIN_NCC} > 0 \Rightarrow \text{NARC} \geq \max(1, 2 \cdot \text{MIN_NCC} - 2)$$

$$\text{equivalence}: \text{NARC} \geq \text{MIN_NCC}^2$$

$$\text{arc_gen} = \text{PATH}: \text{NARC} \geq \text{MIN_NCC} - 1$$

Specializing bounds

General bounds

Graph parameters	Bound
$\underline{\text{NARC}}$	$ E_T + X_{T,\neg T} - \mu(\overrightarrow{G}(X_{T,\neg T}, E_U))$
$\overline{\text{NARC}}$	$ E_{TU} $
$\underline{\text{NVERTEX}}$	$ X_T + h(\overrightarrow{G}((X_{T,\neg T,\neg T}, X_{U,\neg T,T}), E_{U,T}))$
$\overline{\text{NVERTEX}}$	$ X_{TU} $
$\underline{\text{NCC}}$	$ cc_{[X_T \geq 1]}(\overrightarrow{G}(X_{TU}, E_{TU})) $
$\overline{\text{NCC}}$	$ cc_{[E_T \geq 1]}(\overrightarrow{G}(X_T, E_T)) + \mu_l(\overrightarrow{G}_{rem})$
$\underline{\text{NSCC}}$	$ scc_{[X_T \geq 1]}(\overrightarrow{G}(X_{TU}, E_{TU})) + h(G_{\underline{\text{NSCC}}}((Y, Z), E))$
$\overline{\text{NSCC}}$	$ scc(\overrightarrow{G}(X_{TU}, E_T)) $
$\underline{\text{NSINK}}$	$ sink_{[X_T =1]}(\overrightarrow{G}(X_{TU}, E_{TU})) + h(G'_r((Y, Z), E))$
$\overline{\text{NSINK}}$	$ sink(\overrightarrow{G}(X_T, E_T)) + X_U - source_{[X_U =1]}(\overrightarrow{G}(X_{TU}, E_{TU})) - XP $

According to the initial graph: current bounds remain tight but can simplify them.

According to the final graph (symmetric, equivalence): current bounds may not be tight any more.

PREREQUISITE

GENERAL INTRODUCTION

GRAPH BASED DESCRIPTION

- Introduction
- Describing things
- Graph-Based Reformulation
- Normalisation according a given graph-class
- Bounds of graph parameters
- Filtering back from the bounds to the arcs
- Invariants linking several graph parameters
- Taking advantage from the graph structure
- **Putting everything together**



Sketch of a Generic Graph Based Filtering Algorithm

INPUT: A global constraint **C** defined by: $P_1 = V_1 \wedge P_2 = V_2 \wedge \dots \wedge P_n = V_n$

01. Creates the intermediate graph **G** of **C** and evaluates the status of each arc of **C**
02. **Normalises** it according to the graph class of **C**
03. **For** each graph characteristics P_i ($1 \leq i \leq n$) **do**
04. Pmin = **evaluate lower bound of** P_i on **G** according to the graph class of **C**
05. adjust minimum value of V_i to Pmin
06. Pmax = **evaluate upper bound of** P_i on **G** according to the graph class of **C**
07. adjust maximum value of V_i to Pmax
08. **If** $\min(V_i) = P_{\max}$ **then**
09. Turn the status of some arc-constraints to True or False to **ensure** $\min(V_i) = P_{\max}$ and propagate the corresponding binary constraints
10. **If** $\max(V_i) = P_{\min}$ **then**
11. Turn the status of some arc-constraints to True or False to **ensure** $\max(V_i) = P_{\min}$ and propagate the corresponding binary constraints
12. **Propagate all graph invariants** (associated to the graph class **C**) mentioning
EMN the graph parameters P_1, P_2, \dots, P_n

Conclusion

There is still a lot of bounds and graph invariants to be found for **specific graph class** (necessary if want to be efficient)

But this is a task that can be done in an incremental way
(much more easy than building a huge library of algorithms)

Synthesize { checkers
filtering algorithms
incremental moves
visualization
specialized heuristics } from the description of constraints

Understand globals constraints in term of the
properties of their basic **constituents**

CC(FD) for graphs

A. Design a graph-based language that allow to express formula

$$\underline{\text{NVERTEX}} \geq |X_T| + h(\overleftrightarrow{G}((X_{T,-T,-T}, X_{U,-T,T}), E_{U,T}))$$

The language should be **compositional**

(i.e., you **build** intermediate graphs and **compute** something on them)
see **Comet** (P.Van Hentenryck)

B. Design an efficient evaluator for this language