Constraints (2007) 12:21–62
DOI 10.1007/s10601-006-9010-8

Global Constraint Catalogue: Past, Present and Future

Nicolas Beldiceanu · Mats Carlsson · Sophie Demassey · Thierry Petit

Published online: 17 February 2007

Abstract The catalogue of global constraints is reviewed, focusing on the graph-based description of global constraints. A number of possible enhancements are proposed as well as several research paths for the development of the area.

Keywords Global constraint · Catalogue · Graph · Meta-data · Bound · Graph invariant

1 Introduction

The term "Global Constraint". The term *global constraint* originally appeared in the late '80s in a thesis [81] dealing with house floor-plan design problems. A global constraint "rooms must fill the house" was incorporated into a generator of house floor-plans. Later on, the term *global constraint* was reused at the European Computer Research Centre, with a specific constraint taking into account the interaction

Part of this article joins and extends the results from two publications [12, 15].

N. Beldiceanu (✉) · S. Demassey · T. Petit
École des Mines de Nantes, LINA FRE CNRS 2729, 44307 Nantes, France
e-mail: nicolas.beldiceanu@emn.fr

S. Demassey
e-mail: sophie.demassey@emn.fr

T. Petit
e-mail: thierry.petit@emn.fr

M. Carlsson
SICS, P.O. Box 1263, SE-164 29 Kista, Sweden
e-mail: mats.carlsson@sics.se

between several `alldifferent` constraints for handling orthogonal Latin squares problems [4].[1]

In [20], Bessière and Van Hentenryck proposed several definitions of the notion of globality. They introduced the notion of *semantic globality* (expressiveness), *operational globality* (quality of filtering), and *algorithmic globality* (computational efficiency of the filtering). As they mention, "globality is often discussed in an operational context with a consistency notion in mind" (i.e., globality refers to some aspect of the filtering).

This article aims to stress the fact that global constraint can be used beyond filtering. For this reason, the term *global constraint* should be understood here as an *expressive and concise condition involving a non-fixed number of variables*. This informal definition *does not make any assumption about the potential use* of a global constraint. Indeed, defining the concept of global constraint in terms of its potential uses simply "dries up" this concept.

The Area of Global Constraints: Current Status. Global constraints were initially introduced in industrial solvers [31, 62] for solving recurring problems in dedicated application areas. From an early time, global constraints have been used for their strong pruning power thanks to efficient specialised filtering algorithms that take advantage of the structure of the constraints they consider (see for instance, the filtering algorithms proposed for the `alldifferent` constraint [24, 29, 52, 53, 56, 63, 68]). More recently, it became clear that global constraints are not only useful for their deductive power, but that they also have a central role to play in *modelling languages* for problem solving [77], in *local search* [78], in *symmetry breaking* [65], in *visualisation tools* for debugging constraint programs [74], in *linear programming* tools [46] or in deriving efficient *heuristics* [41] based on their internal status.

The Essence of Constraint Programming: An Engineering or an Academic Field? Global constraints are recognised as a mean to accelerate the convergence between Constraint Programming and Operations Research [57]. As a consequence of this current convergence, a common opinion is that Constraint Programming is becoming a sub-field of Operations Research: Constraint Programming provides a collection of new techniques that can be added or combined with, for instance, classical techniques from Linear Programming or Metaheuristics. By adopting this point of view, Constraint Programming is reduced more to an engineering field than to a basic science field with a core question to address [45]. However, the *Holy Grail* [38, 76] was quite clear right from the beginning of Constraint Programming:

> *Specify a problem in a declarative way, and let the machine solve it in an efficient way.*

Actually, no paradigm usually attached to Operations Research has addressed such a question. On one hand, many efficient solution methods have been developed in Operations Research, but they often are very specialised to handle one given problem. On the other hand, the declarative modelling aspect distinguishes Constraint

[1] i.e., `alldifferent` constraints, enforcing that all variables be pairwise different, as well as generalised `alldifferent` constraints, enforcing that all pairs of variables be pairwise different.

Programming from systematic solution approaches of Operations Research, such as Linear Programming.

For both practical and theoretical reasons, the core question of Constraint Programming should be pursued:

- From a practical point of view, the question of the applicability of Constraint Programming has been raised by one of the main providers of Constraint Programming tools at the CP'2004 conference [64]: as more and more efficient ad hoc methods are involved, the technique gets more and more difficult to use in practice. In a sense, Constraint Programming is joining established fields (e.g., logic, linear programming, algorithmic, group theory) whose techniques require a strong expertise only to be used.
- From a theoretical point of view, the question is to come up, not just with efficient solution methods dedicated to specific problems, but with a theory allowing to *automatically generate* efficient solution methods from the *explicit description* of the problems.

Systematic description and filtering of global constraints. This article aims to show how one can start to address the previous core question in the context of global constraints. The key idea is that global constraints have two sides: a *declarative* side describing the meaning of the global constraint, and a *procedural* side corresponding to the code that actually performs a typical task as, for example, the filtering of the constrained variable domains. In this context, the questions to address are:

- Find ways that allow to explicitly define the meaning of global constraints.
- Find processes that allow to synthesize code from the description of a global constraint.

One may wonder how one can ever hope to generate any efficient code from a declarative description. This question is even more relevant if one thinks about all the specialized filtering algorithms that were gradually developed over the past 10 years for specific global constraints like `alldifferent` [24, 52, 53, 56, 63, 68], `global_cardinality` [47, 48, 66, 67, 69], `cumulative` [27, 35, 50] or `tree` [13].

Even if we do not have any definitive answer to this difficult question, we provide the first framework that allows to get a filtering scheme from a declarative description of a global constraint. Within this framework, global constraints are described in terms of graphs satisfying given properties. The properties relate satisfiable values for usual graph parameters. Filtering consists then in maintaining bounds on these values and enforcing the graph properties.

Beyond Filtering. As evocated above, the notion of global constraint may be useful in a context where we are not interested at all in any filtering. For instance, in the context of local search [78], we are only interested in evaluating how much the condition associated with a global constraint is violated. In the context of debugging [30], we would like to have an insightful visual representation [25] of the global constraints of the problem during the solution search (i.e. when variables are not all yet fixed).

Actually, the procedural side of a global constraint may be any code for detecting the constraint infeasibility or for filtering the variable domains, but also, for generating a graphical representation [74], for constructing a heuristic based on the global

constraint description, for evaluating the cost violation [7] associated with a ground instance...

The ability of deriving different codes directly from the description of a global constraint, has motivated our choices for such a declarative description. Besides the graph property-based representation, we also present a second description of global constraints in terms of automata.

Aim and Organisation of the Article. This article has two complementary aims regarding the field of global constraints:

- *To review the work already done with the catalogue* [11] and *to provide a first generic filtering scheme* combining existing and new techniques based on graph properties.
- *To make a series of suggestions for advancement of the field*. They are interspersed in the next sections and have one of the following forms:
  - *Proposed Enhancements* (*PE*) suggest ways to improve specific parts of the catalogue of global constraints. Their purpose is to draw attention to general questions related to potential uses of global constraints in different contexts.
  - *Research Paths* (*RP*) draw attention to key issues related to the synthesis of code for global constraints. They typically are long-term goals.

The article is organised as follows:

- Section 2 gives the motivations behind the catalogue of global constraints [11].
- Section 3 presents the description of global constraints in terms of graph properties as well as in terms of automata. It also suggests research paths related to this second representation.
- Section 4 summarises the results obtained so far related to the systematic graph property-based filtering. It shows how to reformulate a global constraint into a satisfaction problem according to its graph description. Then it presents a first filtering for this reformulation, by computing tight bounds on graph parameter values [15] and by using graph invariants [12].
- Section 5 sketches a natural continuation of this work. It illustrates how to filter by enforcing graph properties according to the parameter bounds, and shows how to tighten those bounds for specific graph classes. Then, we present the generic graph-based filtering scheme, which puts together all the elements introduced in Sections 4 and 5. Finally, we show how to retrieve existing filtering algorithms for global constraints such as `alldifferent` and `proper_forest`, with this scheme, and how to adapt it in the context of CP(Graph) [32].
- Section 6 gives several research paths, beyond filtering, related to the graph-based representation of global constraints.
- Section 7 concludes.

2 Motivations

This section presents our motivations for an explicit description of the meaning of global constraints. A current trend in the field of global constraints is to use natural language to describe the meaning of a global constraint, and then, to work out a

specialised filtering algorithm. Describing all potential global constraints with this approach is an endless task, when one considers the many ways of combining global constraints.

This is even worse if, in the long term, we aim at providing other services such as visualisation [74], explanations [73], soft global constraints [7, 60, 80], learning implied global constraints [21], or specialised heuristics for each global constraint.

These considerations motivates the use of a composable mathematical language for explicitly describing the meaning of global constraints. The CP(Graph) framework [32], for instance, proposes a restricted form of second order predicate calculus. However, when it comes to filtering algorithms, this model is not used any more.[2] For such a reason, Prolog was restricted to Horn clauses for which one had a reasonable solving mechanism.

Through this example, we want to stress the fact that a declarative description is really useful only if it also provides some hints about how to deal with that description.

2.1 Graph-based Description

We first proposed a description of global constraints in terms of *graphs* satisfying some graph properties that are given by *formulae*. This choice was influenced by the following observations:

- The concept of graph has its roots in the area of mathematical recreations (see for instance Euler [36], Dudeney [33], Lucas [54] and Kirkman [49]), which was somehow the ancestor of combinatorial problems.
- In one of the first books introducing graph theory [19], Berge presents graph theory as a way of grouping apparently diverse problems and results. This was also the case for global constraints.
- The parameters associated with graphs are concrete and concise (e.g., the number of arcs, the maximum size of connected components,...). Moreover a lot of results about graphs can be expressed in terms of graph invariants involving various graph parameters, that are valid for specific graph classes. In essence, formulae are a kind of declarative statement that is much more suited for creating a knowledge data base in combinatorics than algorithms. In fact, as we will see in Section 5.1, quite often significant parts of filtering algorithms can be reinterpreted as a compiled form of a graph invariant $\mathcal{I}$, which enforces the validity of $\mathcal{I}$.
- Finally, it is well known that graph theory is an important tool [55] with respect to the development of efficient filtering algorithms [9, 47, 56, 67–70, 72, 79].

The question of how to automatically generate efficient code from the description of a problem was already identified as the core question by forerunners in Constraint Programming like Pitrat [61] and Laurière [51]. Both promote an approach based on meta-programming but did not address the question of the relation between the combinatorial aspect of constraints and graph theory. In Operations Research, we

[2] One could perhaps use a system like MONA [43] for getting a constraint checker in the context of CP(Graph).

can cite a work of Hansen related to this article [42]. He addresses the question of the automatic generation of graph invariants that can potentially be used in branch and bound methods.

2.2 Automaton-based Representation

Our second choice of an automaton-based representation has been motivated by the following observation. Writing a constraint checker is a straightforward task. The corresponding program can usually be turned into an automaton. Of course, an automaton is typically used on a fixed sequence of symbols. But, in the context of filtering algorithms, we have to deal with a sequence of variables. For this purpose, we have shown [8] for some automata how to decompose them into a conjunction of smaller constraints. In this context, a global constraint can be seen as a hypergraph corresponding to its decomposition.

Note that, in the context of global constraints on a finite sequence of variables, the `regular` constraint by Pesant [59] also uses a finite automaton for representing the solution set.

3 Summary of the Framework

This section summarises the representation of global constraints as graph properties introduced in [5] or as a hypergraph of constraints associated with an automaton [10]. It illustrates this framework with the `nvalue` [23] and the `group` [12] constraints.

3.1 Graph-based Description

Let $C(V_1, \ldots, V_p, x_1, \ldots, x_n)$ be a global constraint with domain variables[3] $V_1, \ldots, V_p$, and domain or set variables[4] $x_1, \ldots, x_n$. When it exists, a *graph-based description* of C is given by one (or several) network(s) $G_{\mathcal{R}} = (X, E_{\mathcal{R}})$ of binary constraints over $X = \{x_1, \ldots, x_n\}$ in association with a set $\mathcal{GP}_{\mathcal{R}} = \{\mathcal{P}_l \; op_l \; V_l \mid l = 1, \ldots, p\}$ of graph properties and, optionally, a graph class $c_{\mathcal{R}}$, where:

- The constraints defining digraph $G_{\mathcal{R}} = (X, E_{\mathcal{R}})$ share the same semantics (typically it is an equality, an inequality or a disequality). Let $x_j \mathcal{R} x_k$ denote the so-called *arc constraint* between the ordered pair of variables $(x_j, x_k) \in E_{\mathcal{R}}$ (or the unary constraint if $j = k$).
- $\mathcal{P}_l \; op_l \; V_l$ expresses a graph property comparing the value of a graph parameter $\mathcal{P}_l$ to the value of variable V_l. The comparison operator op_l is either $\geq$, $\leq$, $=$, or $\neq$. Among the most usual graph parameters $\mathcal{P}_l$, let **NARC** denote the number of arcs of a graph, **NVERTEX** the number of vertices, **NCC** the number of connected components, **MIN_NCC** and **MAX_NCC** the numbers of vertices of the smallest and the largest connected components respectively.

[3] A *domain variable* D is a variable ranging over a finite set of integers $\mathrm{dom}(D)$. $\min(D)$ and $\max(D)$ respectively denote the *minimum* and *maximum* values in $\mathrm{dom}(D)$.

[4] A *set variable* S is a variable that will be assigned to a finite set of integer values. Its domain is specified by its *lower bound* $\underline{S}$, and its *upper bound* $\overline{S}$, and contains all sets that contain $\underline{S}$ and are contained in $\overline{S}$.

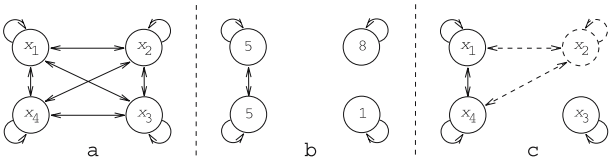

Fig. 1 **a** Initial digraph $G_{\mathcal{R}}$ associated with the nvalue(N, $\{x_1, x_2, x_3, x_4\}$) constraint. **b** Final digraph G_f of the ground solution nvalue(3, {5, 8, 1, 5}). **c** Digraph corresponding to a partial assignment: $\text{dom}(x_1) = \{5\}$, $\text{dom}(x_3) = \{1\}$, $\text{dom}(x_4) = \{5\}$ and $\text{dom}(x_2) = \{5, 8\}$

– $c_{\mathcal{R}}$ corresponds to recurring graph classes that show up for different global constraints. For example, we consider graphs in the classes acyclic, symmetric, bipartite.

$G_{\mathcal{R}}$ is called the *initial digraph*. When all variables x are instantiated, the subgraph of $G_{\mathcal{R}}$, obtained by removing all arcs corresponding to unsatisfied constraints $x_j \mathcal{R} x_k$ as well as all vertices becoming isolated,[5] is called a *final digraph* (associated with the instantiation) and is denoted by $G_f = (X_f, E_f)$.

The relation between C and its graph-based description is stated as follows:

Proposition 1 *A complete assignment of variables $V_1, \ldots, V_p, x_1, \ldots, x_n$ is a solution of C iff the final digraph associated with the assignment of $x_1, \ldots, x_n$, satisfies all graph properties $\mathcal{P}_l$ op_l V_l in $\mathcal{GP}_{\mathcal{R}}$ and belongs to the graph class $c_{\mathcal{R}}$.*

Example 1 Consider the nvalue(N, VARIABLES) constraint, where N and VARIABLES $= \{x_1, ..., x_m\}$ are domain variables. The nvalue constraint holds iff the number of distinct values assigned to the variables in VARIABLES is equal to N. Parts a and b of Fig. 1 respectively show the initial digraph $G_{\mathcal{R}}$ generated for the nvalue constraint with VARIABLES $= \{x_1, x_2, x_3, x_4\}$ and the final digraph G_f associated with the ground solution nvalue(3, {5, 8, 1, 5}). Each vertex of $G_{\mathcal{R}}$ depicts a variable. All arcs corresponding to equality constraints that are not satisfied are removed to obtain G_f from $G_{\mathcal{R}}$. Each vertex of G_f depicts the value assigned to its corresponding variable. The nvalue constraint is defined by the graph property **NSCC** = N, which means that a complete instantiation satisfies constraint nvalue iff the associated final digraph is made up of exactly N strongly connected components. The nvalue(3, {5, 8, 1, 5}) constraint holds since G_f contains three strongly connected components. Part c of Fig. 1 will be referenced in Example 4.

Example 2 Consider the group(NGROUP, MIN_SIZE, MAX_SIZE, MIN_DIST, MAX_DIST, NVAL, VARIABLES, VALUES) constraint, where the first six parameters are domain variables, while VARIABLES is a sequence of domain variables and VALUES a finite set of integers. Let m denote the number of variables of the sequence VARIABLES. Let $x_i, x_{i+1}, \ldots, x_j$ $(1 \leq i \leq j \leq m)$ be consecutive variables

[5] A few constraints of the catalogue like common, same or used_by, which all use the graph parameter **NSINK** or **NSOURCE**, rely on the fact that isolated vertices are removed from the final digraph (i.e., this way, isolated vertices are neither counted as sources nor sinks).

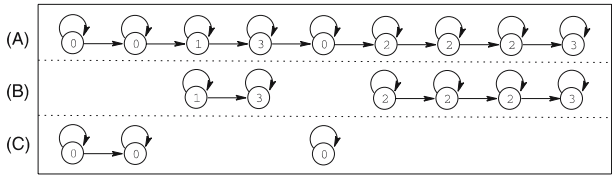

Fig. 2 Initial (**a**) and final digraphs (**b**,**c**) of group(2, 2, 4, 1, 2, 6, ⟨0, 0, 1, 3, 0, 2, 2, 2, 3⟩, {1, 2, 3})

of the sequence VARIABLES such that all the following conditions simultaneously apply: (1) all variables $x_i, \ldots, x_j$ take their value in the set of values VALUES, (2) either $i = 1$, or x_{i-1} does not take a value in VALUES, (3) either $j = m$, or x_{j+1} does not take a value in VALUES. We call such a set of variables a *group*. The constraint group is fulfilled if all the following conditions hold:

- there are exactly NGROUP groups of variables,
- MIN_SIZE and MAX_SIZE are the number of variables of the smallest and largest group,
- MIN_DIST and MAX_DIST are the minimum and maximum number of variables between two consecutive groups or between one border and one group,
- NVAL is the number of variables taking their value in the set VALUES.

group(2, 2, 4, 1, 2, 6, ⟨0, 0, 1, 3, 0, 2, 2, 2, 3⟩, {1, 2, 3}) holds since the sequence ⟨0, 0, 1, 3, 0, 2, 2, 2, 3⟩ contains two groups ⟨1, 3⟩ and ⟨2, 2, 2, 3⟩ of non-zero values of size 2 and 4, two groups ⟨0, 0⟩ and ⟨0⟩ of zeros, and six non-zero values. The graph-based description of the group constraint uses two graph constraints which respectively mention the graph properties **NCC** = NGROUP, **MIN_NCC** = MIN_SIZE, **MAX_NCC** = MAX_SIZE, **NVERTEX** = NVAL and **MIN_NCC** = MIN_DIST, **MAX_NCC** = MAX_DIST. Figure 2 depicts the initial digraph as well as the two final digraphs associated with the two graph constraints of the example given for the group constraint. The group constraint will also be used in Section 3.2 for illustrating the automaton-based description.

3.2 Automaton-based Representation

Currently, the catalogue of global constraints contains more than 100 constraints that can be described with one or several automata. We first illustrate this on the example of the group [12] constraint introduced in the previous section and then present the different constraint network structures we have identified. Finally, we suggest some research paths related to the use of automata in the context of global constraints.

Example 3 Parts a, b, c and d of Fig. 3 respectively depict the automata associated with the graph parameters **NCC**, **MIN_NCC**, **MAX_NCC** and **NVERTEX** of the first graph constraint described in the last paragraph of Example 2. Each automaton is applied to the sequence of variables corresponding to the VARIABLES parameter. A transition with a standard line depicts the fact that a variable takes its value in the set VALUES, while a thick line denotes the fact that a variable does not take its value in VALUES. Finally, a transition with a dashed line indicates the end of the sequence of variables. Since all the four automata use counters, we indicate how these counters are initialised in the initial state s, how a counter is unified to an argument of

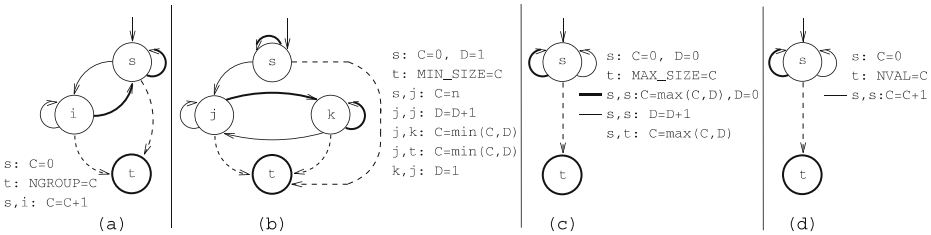

Fig. 3 Automata associated with the graph parameters ofthe `group` constraint

the `group` constraint in the final state `t`, and how they are possibly updated on a given transition. When there are several transitions between a given pair of states, we indicate with a dotted line or a standard line its type (see for instance the two transitions between `s` and `s` of the automaton depicted by Part c). The automata associated with **MIN_NCC** = `MIN_DIST` and with **MAX_NCC** = `MAX_DIST` are similar to the automata depicted by Parts b and c, except that we change a thick line to a standard line and vice versa.

In the catalogue of global constraints we have identified the following recurring classes of constraint network structures, depicted in Fig. 4:

- Berge-acyclic constraint network,
- Alpha-acyclic constraint network,
- Sliding cyclic constraint network,
- Circular sliding cyclic constraint network,
- Centered cyclic constraint network.

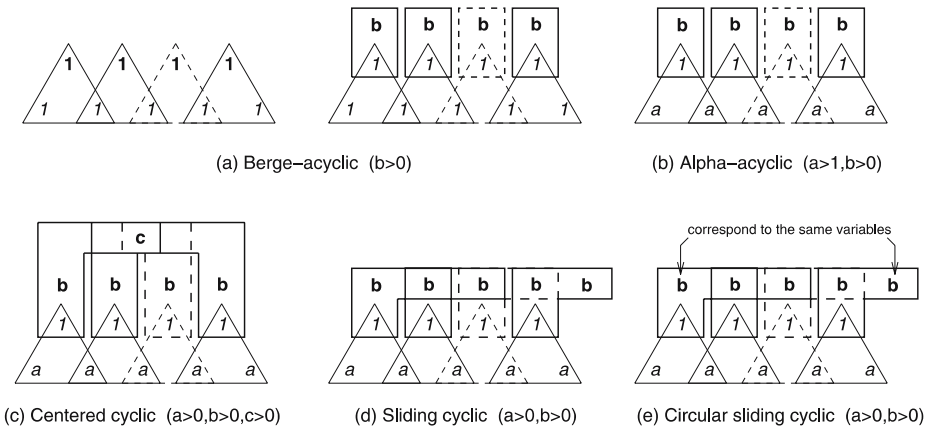

Fig. 4 The different classes of constraint networks; a 1 represents one single variable, while letters a, b, c respectively stand for a, b and c distinct variables; bold and italic styles respectively depict variables that originally occur in the global constraint or that were introduced for modelling the behaviour of the automaton; all constraint networks mention only two distinct constraints $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively outlined with a thick and a dashed line. The constraint networks associated with the four automata of Fig. 3 correspond to Alpha-acyclic constraint networks

Even if the area of constraint network is a well established topic of Constraint Programming, almost nothing is known[6] when the networks are highly structured and non-binary, as is actually the case for global constraints. Moreover we are not interested in pruning all the variables of the network but only the variables that belong to the original global constraint from which the network was derived. This suggests the following research path.

RP 1. Investigate the topic of *structured constraint networks* and the relations between the type of consistency one can achieve and the properties of the hyperedges as well as the overall structure of the constraint network.

A significant number of important global constraints like `alldifferent`, `global_cardinality` or `nvalue` can only be modelled in a compact way by using automata together with arrays of counters, which are modified while triggering some transitions. However, it is not currently known how to perform any filtering for this class of automaton. This suggests the following question.

PE 1. Come up with reformulation and/or filtering algorithms for handling automata with *arrays of counters*.

4 Results to Date

This section gives an overview of the results obtained so far in relation with the use of the graph-based description of global constraints for filtering. The next section presents refinements and further researches on this topic.

All filtering presented in these two sections rely on the automatic reformulation of most global constraints into a satisfaction problem based on the graph representation presented in Section 3.1 and including domain variables P_l $(1 \leq l \leq n)$ associated with each graph parameter $\mathcal{P}_l$. The reformulation is presented in Section 4.1. Tight estimations of the bounds of variables P_l are given in Section 4.2, for the general case, and in Section 5.2 for a specific class of graph. A first way of filtering is to enforce the consistency of the mathematical constraints involving P_l. Sections 4.3 and 4.4 respectively show how to filter from graph property constraints and from graph invariant constraints. Conversely, Section 5.1 shows how to filter back from the bounds of P_l to the variables of the global constraint. A complete filtering algorithm including all these techniques is given in Section 5.3.

4.1 Graph-based Reformulation

According to Definition 1, any global constraint $C(V_1, \ldots, V_p, x_1, \ldots, x_n)$ holding a graph-based description can be reformulated as follows:

Proposition 2 *Define additional variables attached to each constraint network* $G_{\mathcal{R}} = (X, E_{\mathcal{R}})$*: to each vertex* x_j *and to each arc* e_{jk} *of* $G_{\mathcal{R}}$ *correspond* 0-1 *variables*

[6] Beside the results on Berge-acyclic and alpha-acyclic constraint networks.

respectively denoted $vertex_j$ *and* arc_{jk}. *Vertex and Arc denote these sets of variables. Last, let* G_f *denote the subgraph of* $G_{\mathcal{R}}$, *whose vertices and arcs correspond to the variables* $vertex_j$ *and* arc_{jk} *set to 1. Then constraint C holds iff the following constraints hold:*

$$arc_{jk} = 1 \Leftrightarrow x_j \mathcal{R} x_k, \quad \forall e_{jk} \in E_{\mathcal{R}} \tag{1}$$

$$vertex_j = \min\left(1, \sum_{\{k \mid e_{jk} \in E_{\mathcal{R}}\}} arc_{jk} + \sum_{\{k \mid e_{kj} \in E_{\mathcal{R}}\}} arc_{kj}\right), \quad \forall x_j \in X \tag{2}$$

$$ctr_{\mathcal{P}_l}(Vertex, Arc, P_l), \quad \forall (\mathcal{P}_l \; op_l \; V_l) \in \mathcal{GP}_{\mathcal{R}} \tag{3}$$

$$P_l \; op_l \; V_l, \quad \forall (\mathcal{P}_l \; op_l \; V_l) \in \mathcal{GP}_{\mathcal{R}} \tag{4}$$

$$ctr_{c_{\mathcal{R}}}(Vertex, Arc) \tag{5}$$

where Constraint (3) *is satisfied when* P_l *is equal to the value of the corresponding parameter* $\mathcal{P}_l$ *in* G_f *and graph-class constraint* (5) *is satisfied if* G_f *belongs to the graph class* $c_{\mathcal{R}}$.

Filtering domains of variables V and x according to C can be achieved by enforcing alternatively, and for each constraint network $G_{\mathcal{R}}$, the five sets of constraints of this reformulation. Enforcing constraints (1), (2), (4) and (5) is mostly trivial since these constraints are elementary arithmetic constraints. The generic graph-based filtering then mainly relies on maintaining consistency according to constraints (3), from the *arc* and *vertex* variables to the bounds of the graph parameter variables P_l, and conversely. In [15] it was presented, for some usual graph parameters, how to estimate their minimal ($\underline{P_l}$) and maximal ($\overline{P_l}$) values in the final digraphs G_f, given the current state of the arcs and vertices of $G_{\mathcal{R}}$. Section 4.3 shows how in turn, the status of some arcs and vertices can be determined according to a graph parameter variable when it is set to one of its extreme values (i.e. $\text{dom}(P_l) = \{\underline{P_l}\}$ or $\text{dom}(P_l) = \{\overline{P_l}\}$).

Hence, the approach relies on identifying the possible final digraphs in $G_{\mathcal{R}}$ that minimise or maximise a given graph parameter. Any final digraph contains (resp. does not contain) the arcs and vertices corresponding to *arc* and *vertex* variables fixed to 1 (resp. to 0). Since it has no isolated vertices, we assume that the *normalisation constraints* (2) are enforced before estimating a graph parameter.

Since the proposed reformulation applies to many global constraints for which enforcing AC is proven to be NP-hard (e.g. `nvalue`), we cannot expect to get AC in general. Even with a complete characterisation of all feasible values of a graph parameter and of all corresponding unfeasible arcs (arcs that do not belong to final digraphs satisfying a parameter value), we cannot enforce AC in general because of the inter-dependency of constraints (1) : the *arc* variables are not independent of each other.

As for the graph-based description of any global constraint, we aim at providing a catalogue of generic filtering rules related to the bounds of graph parameters. In order to formalise this, we first need to introduce a number of notations.

Let $G_{\mathcal{R}}$ be an initial digraph associated with the graph-based description of a global constraint. The current domains of variables *arc* and *vertex* of the reformulation correspond to a unique partitioning of the arc and vertex sets of $G_{\mathcal{R}}$, denoted as follows:

Notation 1 A vertex x_j or an arc e_{jk} of $G_{\mathcal{R}}$ is either *true* (T), *false* (F), or *undetermined* (U) whether the corresponding variable $vertex_j$ or arc_{jk} is fixed to 1, fixed to 0 or yet unfixed (with domain $\{0, 1\}$). This leads to the partitioning of the vertex set of $G_{\mathcal{R}}$ into $X_T \dot{\cup} X_F \dot{\cup} X_U$ and to the partitioning of the edge set of $G_{\mathcal{R}}$ into $E_T \dot{\cup} E_F \dot{\cup} E_U$.

For two distinct elements Q and R in $\{T, U, F\}$, let X_{QR} denote the vertex subset $X_Q \dot{\cup} X_R$, and E_{QR} denote the arc subset $E_Q \dot{\cup} E_R$.

Once the normalisation constraints are enforced, subgraph (X_T, E_T) is well defined and is included in any final digraph. (X_{TU}, E_{TU}) is also a subgraph of $G_{\mathcal{R}}$, called the *intermediate digraph*, and any final digraph is derived from this by turning each U-arc and U-vertex into T or F.[7] We aim at identifying the final digraphs in which a graph parameter P reaches its lower value $\underline{P}$ or its upper value $\overline{P}$. An estimated bound is said to be *sharp* if for any intermediate digraph, there exists at least one final digraph where the parameter takes this value. To estimate these bounds, we deal with different digraphs derived from the intermediate digraph:

Notation 2 For any subsets Q, R and S of $\{T, U, F\}$, X_Q and X_S are vertex subsets and E_R is an arc subset of the initial digraph, and:

- $X_{Q,R}$ (resp. $X_{Q,\neg R}$) denotes the set of vertices in X_Q that are extremities of at least one arc (resp. no arc) in E_R.
- $X_{Q,R,S}$ (resp. $X_{Q,R,\neg S}$) denotes the set of vertices in $X_{Q,R}$ that are linked to at least one vertex (resp. to no vertex) in X_S by an arc in E_R.
- $X_{Q,\neg R,S}$ (resp. $X_{Q,\neg R,\neg S}$) denotes the set of vertices in $X_{Q,\neg R}$ that are linked to at least one vertex (resp. to no vertex) in X_S by an arc in E_{TU}.
- $E_{R,Q}$ is the set of arcs in E_R that are incident on at least one vertex in X_Q.
- $E_{R,Q,S}$ is the set of arcs in E_R that are incident on one vertex in X_Q and on one vertex in X_S.

Notation 3 Given a digraph G and subsets $\mathcal{X}$ of vertices and $\mathcal{E}$ of arcs:

- $\overrightarrow{G}(\mathcal{X}, \mathcal{E})$ denotes the induced subgraph of G containing all vertices in $\mathcal{X}$ and all arcs of $\mathcal{E}$ having their two extremities in $\mathcal{X}$.
- $\overleftrightarrow{G}(\mathcal{X}, \mathcal{E})$ denotes the corresponding undirected graph: to one edge (u, v) corresponds at least one arc (or loop) (u, v) or (v, u) in $\overrightarrow{G}(\mathcal{X}, \mathcal{E})$.

Example 4 Consider again the `nvalue(N, VARIABLES)` constraint introduced in Example 1, and assume that not all variables of `VARIABLES` $= \{x_1, x_2, x_3, x_4\}$ are fixed: $\text{dom}(x_1) = \{5\}$, $\text{dom}(x_2) = \{5, 8\}$, $\text{dom}(x_3) = \{1\}$, $\text{dom}(x_4) = \{5\}$. Furthermore w.l.o.g. assume that, for an equality constraint *ec* associated with an arc of the initial digraph $G_{\mathcal{R}}$ of `nvalue`, entailment is only detected when the two variables occurring in *ec* are fixed. This leads to partition the arcs of $G_{\mathcal{R}}$ in the following three sets $E_T = \{(x_1, x_1),(x_1, x_4),(x_3, x_3),(x_4, x_1),(x_4, x_4)\}$, $E_U = \{(x_1, x_2),$

[7] In the context of CP(Graph) [32], these two digraphs respectively correspond to the lower and upper bounds of a graph variable. Note that, as a consequence, our approach can easily be adapted to provide a generic filtering for CP(Graph).

$(x_2, x_1),(x_2, x_2),(x_2, x_4),(x_4, x_2)\}$ and $E_F = \{(x_1, x_3),(x_2, x_3),(x_3, x_1),(x_3, x_2), (x_3, x_4), (x_4, x_3)\}$. The status of the vertices is initially set *undetermined* (i.e., $X_U = \{x_1, x_2, x_3, x_4\}$) and Part c of Fig. 1 shows the current status of arcs and vertices of a partial assignment (*intermediate digraph)*. A solid line depicts a T-vertex or a T-arc, while a dashed line indicates a U-vertex or a U-arc. The same style will be used in all other figures of this article in order to depict T-vertices, T-arcs, U-vertices and U-arcs.

Computing lower and upper bounds of graph parameters can essentially be seen as computing some other graph parameters on the graphs introduced above. For this purpose, we define the following notations for a digraph G.

Notation 4 $vertex(G)$, $cc(G)$, $scc(G)$, $sink(G)$ and $source(G)$ respectively denote the set of vertices, connected components, strongly connected components, sinks and sources of G. Similarly, in order to restrict to specific subsets of elements satisfying a given condition *cond*, we use the following notation: $vertex_{[cond]}(G)$, $cc_{[cond]}(G)$, $scc_{[cond]}(G)$, $sink_{[cond]}(G)$ and $source_{[cond]}(G)$.

Example 5 For instance, the set of connected components of $\overrightarrow{G}(X_{TU}, E_{TU})$ containing at least one T-arc is denoted by $cc_{[|E_T|\geq 1]}(\overrightarrow{G}(X_{TU}, E_{TU}))$.

4.2 Bounds of Graph Parameters

In the constraint system of Proposition 2, the domain of a graph parameter variable P must contain the values the graph parameter takes in any final digraph reachable from the current status of arcs and vertices. A way to estimate the bounds of the domain of P is to consider the minimum and the maximum values taken by P after setting all undecided variables arc_{jk} and $vertex_j$ to 0 or 1 according to the normalisation constraints (2). This section shows how to approximate these minimum and maximum values efficiently for the common graph parameters introduced in Section 3. Let $\underline{P}$ and $\overline{P}$ denote such bounds. When the approximation is equal to the effective optimum value, i.e. if for any intermediate digraph, there exists at least one final digraph where the parameter takes this value, the bound is *sharp*.

For computing these bounds, we will deal with digraphs derived from the intermediate digraph with different sets of arcs and vertices. Section 4.2.1 describes the obtained bounds. Section 4.2.2 puts in perspective these results. We previously introduce some notions of graph theory, which are used in the bound computations.

Definition 1 Graph theoretic terms:

– Given a digraph G, a sequence $(a_1, a_2, \ldots, a_k)$ of arcs of G such that, for each arc a_i $(1 \leq i < k)$ the end of a_i is equal to the start of the arc a_{i+1}, is called a *path*. A path where all vertices are distinct is called an *elementary path*. Each equivalence class of the relation "a_i is equal to a_j or there exists a path between a_i and a_j"

is a *strongly connected component* of G. The *reduced digraph* of G is defined as follows: to each strongly connected component of G corresponds a vertex of the reduced digraph. To each arc of G that connects different strongly connected components corresponds an arc in the reduced digraph (multiple arcs between the same pair of vertices are merged).

– Given an undirected graph G, a sequence $(e_1, e_2, \ldots, e_k)$ of edges of G such that each edge has a common vertex with the previous edge, and the other vertex common to the next edge is called a *chain*. A chain where all vertices are distinct is called an *elementary chain*. An *articulation point* (resp. a *bridge*) of G is a vertex (resp. an edge) whose removal increases the number of connected components. A *matching* of G is a set of edges, excluding loops, of G such that no two edges have a vertex in common. A *maximum matching* is a matching of maximum cardinality (i.e., maximum number of edges). $\mu(G)$ denotes the *cardinality of a maximum matching* of G. If loops are allowed in the matching then it is called a *l-matching* and the maximum cardinality of a l-matching in G is denoted by $\mu_l(G)$.
– Given a bipartite graph $G((Y, Z), E)$, a *hitting set* of $G((Y, Z), E)$ is a subset Z' of Z such that for any vertex $y \in Y$ there exists an edge in E connecting y to a vertex in Z'. $h(G)$ denotes the *cardinality of a minimum hitting set* of G.

4.2.1 Bound Computations

The bounds of the graph parameters are briefly presented and illustrated by examples. Proofs are available in [16].

Estimating $\overline{\textbf{NARC}}$ and $\overline{\textbf{NVERTEX}}$ Setting all U-vertices and U-arcs to true leads to a final digraph that does not violate the normalisation constraints (2). Hence, the maximum number of arcs and vertices are given by these formula:

$$\overline{\textbf{NARC}} = |E_{TU}| \qquad \overline{\textbf{NVERTEX}} = |X_{TU}|$$

Estimating $\underline{\textbf{NARC}}$ Because of the normalisation constraints (2), any final digraph G_f has to contain all T-arcs as well as U-arcs in order to connect the isolated T-vertices (i.e., vertices in $X_{T,\neg T}$). The minimum number of such required U-arcs is equal to the number of isolated T-vertices minus the number of arcs that can be saved by connecting these vertices together in a maximum matching.

$$\underline{\textbf{NARC}} = |E_T| + |X_{T,\neg T}| - \mu(\overleftrightarrow{G}(X_{T,\neg T}, E_U))$$

Example 6 Figure 5 shows how to compute the lower bound of **NARC** according to the intermediate digraph $\overrightarrow{G}(X_{TU}, E_{TU})$ given by Part a. Part b shows the corresponding undirected graph $\overleftrightarrow{G}(X_{T,\neg T}, E_U)$ used for computing the cardinality of a maximum matching. We have $E_T = \{(1, 1), (1, 2), (5, 1)\}$, $E_U = \{(1, 5), (2, 6), (3, 4), (4, 4), (5, 6), (6, 7), (7, 3)\}$, $X_{T,\neg T} = \{4, 6, 7\}$, $|E_T| = 3$, $|X_{T,\neg T}| = 3$, $\mu(\overleftrightarrow{G}(X_{T,\neg T}, E_U)) = 1$, thus $\underline{\textbf{NARC}} = 3 + 3 - 1 = 5$. Part c provides a final digraph where this lower bound of five arcs is reached.

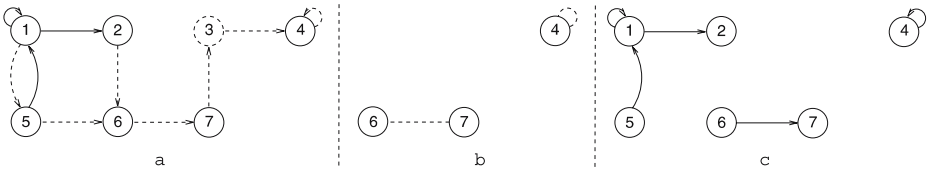

Fig. 5 **a** Intermediate digraph and **b** undirected graph used to compute the cardinality of a maximum matching for estimating **NARC**. **c** Example of a final digraph achieving the lower bound

Estimating NVERTEX Because of the normalisation constraints (2), any final digraph G_f has to contain all T-vertices as well as some U-vertices that are linked to some T-vertices that are not linked to any T-vertices (i.e., vertices in $X_{T,\neg T,\neg T}$). The minimum number of such required U-vertices is equal to the cardinality of a minimum hitting set in an undirected subgraph of the intermediate digraph (see Part b of Fig. 6).

$$\underline{\textbf{NVERTEX}} \;=\; |X_T| + h(\overleftrightarrow{G}((X_{T,\neg T,\neg T}, X_{U,\neg T,T}), E_{U,T}))$$

Example 7 Figure 6 shows how to compute the lower bound of **NVERTEX** according to the intermediate digraph depicted by Part a of Fig. 6. Part b illustrates the corresponding bipartite graph $\overleftrightarrow{G}((X_{T,\neg T,\neg T}, X_{U,\neg T,T}), E_{U,T})$ used for computing the cardinality of the minimum hitting set. We have $X_T = \{1, 3, 6\}$, $E_{U,T} = \{(1, 1), (1, 2), (3, 4), (5, 1), (5, 6), (6, 7), (7, 3)\}$, $X_{T,\neg T,\neg T} = \{3, 6\}$, $X_{U,\neg T,T} = \{2, 4, 5, 7\}$, $|X_T| = 3$ and $h(\overleftrightarrow{G}((X_{T,\neg T,\neg T}, X_{U,\neg T,T}), E_{U,T})) = 1$ (i.e., 7 allows to cover both 6 and 3), thus **NVERTEX** $= 3 + 1 = 4$. Part c provides a final digraph where this lower bound of four vertices is reached.

Estimating NCC The connected components of the intermediate digraph can obviously not be merged in any final subgraph. Furthermore, the connected components containing no T-vertex can be safely removed without violating constraints (2).

$$\underline{\textbf{NCC}} \;=\; |cc_{[|X_T|\geq 1]}(\overrightarrow{G}(X_{TU}, E_{TU}))|$$

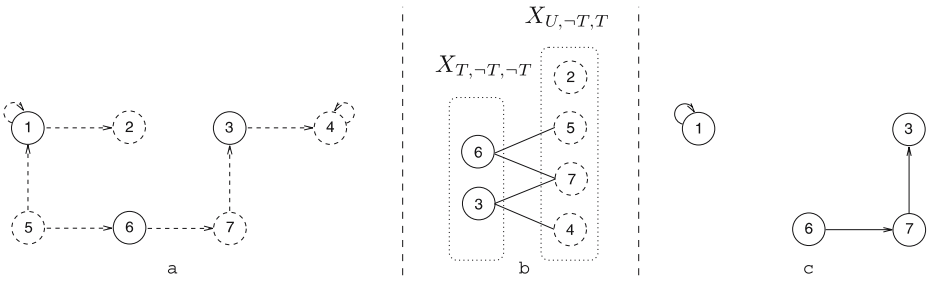

Fig. 6 **a** Intermediate digraph and **b** undirected graph used to compute the cardinality of a minimum hitting set for estimating **NVERTEX**. **c** Example of a final digraph achieving the lower bound

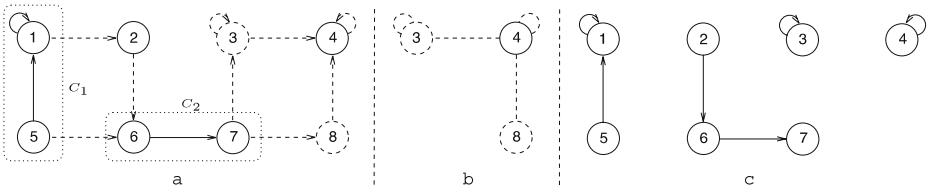

Fig. 7 **a** Intermediate digraph and **b** undirected graph $\overleftrightarrow{G}_{rem}$ used for estimating $\overline{\mathbf{NCC}}$. **c** Example of a final digraph achieving the upper bound

Estimating $\overline{\mathbf{NCC}}$ If we except T-vertices with no incident T-arcs we are able to count in $\overrightarrow{G}(X_T, E_T)$ a certain number of connected components. By definition, augmenting any of these connected components by new arcs cannot increase their number. Therefore, the intuitive idea for estimating $\overline{\mathbf{NCC}}$ is first to count these connected components, to remove them from the intermediate digraph, and then to estimate the maximum number, *rem*, of connected components that may exist in the remaining digraph. For this purpose, we introduce $\overleftrightarrow{G}_{rem}$, which denotes the induced subgraph of the undirected graph $\overleftrightarrow{G}(X_{TU}, E_U)$ obtained by removing all vertices present in $cc_{[|E_T|\geq 1]}(\overrightarrow{G}(X_T, E_T))$ and then by removing all vertices becoming isolated in the remaining undirected graph. Since loops are allowed, *rem* corresponds to the maximum cardinality of a l-matching on this graph.

$$\overline{\mathbf{NCC}} = |cc_{[|E_T|\geq 1]}(\overrightarrow{G}(X_T, E_T))| + \mu_l(\overleftrightarrow{G}_{rem})$$

Example 8 Figure 7 shows how to compute the upper bound of **NCC** according to the intermediate digraph $\overrightarrow{G}(X_{TU}, E_{TU})$ depicted by Part a of Fig. 7. In Part a, C_1 and C_2 correspond to the connected components of $\overrightarrow{G}(X_T, E_T)$ that have at least one T-arc (i.e., $cc_{[|E_T|\geq 1]}(\overrightarrow{G}(X_T, E_T))$). Part b illustrates the corresponding undirected graph $\overleftrightarrow{G}_{rem}$ on which we compute a maximum l-matching. We have $|cc_{[|E_T|\geq 1]}(\overrightarrow{G}(X_T, E_T))| = 2$, $\mu_l(\overleftrightarrow{G}_{rem}) = 2$, thus $\overline{\mathbf{NCC}} = 2 + 2 = 4$. Part c provides a final digraph where this upper bound of four connected components is reached.

Estimating $\underline{\mathbf{NSCC}}$ First observe that two T-vertices that belong to two distinct strongly connected components of the intermediate digraph also belong to distinct strongly connected components of any final digraph G_f. Therefore $\underline{\mathbf{NSCC}} \geq |scc_{[|X_T|\geq 1]}(\overrightarrow{G}(X_{TU}, E_{TU}))|$. Then, because of the normalisation constraints (2), a T-vertex that is not connected to any strongly connected component that includes at least one T-vertex requires the creation of some extra strongly connected component. The total number of such strongly connected components is equal to the cardinality of a minimum hitting set on the following bipartite graph $G_{\underline{\mathbf{NSCC}}}((Y, Z), E)$:

– To each strongly connected component of $scc_{[|X_T|=1\wedge|E_{TU}|=0]}(\overrightarrow{G}(X_{TU}, E_{TU}))$ that does not have as successor or predecessor a vertex belonging to a strongly connected component of $scc_{[|X_T|\geq 1]}(\overrightarrow{G}(X_{TU}, E_{TU}))$, corresponds a vertex in Y.

– To each strongly connected component of $\overrightarrow{G}(X_{TU}, E_{TU})$ that is linked by an arc to a strongly connected component associated with an element of Y, corresponds a vertex of Z.
– Given a vertex $y \in Y$ and a vertex $z \in Z$, $(y, z) \in E$ iff there is an arc between a vertex of the strongly connected component associated with Y and a vertex of the strongly connected component associated with Z in the digraph $\overrightarrow{G}(X_{TU}, E_{TU})$.

$$\underline{\mathbf{NSCC}} = |scc_{[|X_T|\geq 1]}(\overrightarrow{G}(X_{TU}, E_{TU}))| + h(G_{\underline{\mathbf{NSCC}}}((Y, Z), E))$$

Example 9 Figure 8 shows how to compute the lower bound of **NSCC** according to the intermediate digraph $\overrightarrow{G}(X_{TU}, E_{TU})$ depicted by Part a. All the strongly connected components $C_1, C_2, \ldots, C_9$ of this intermediate digraph are enclosed in a dotted rectangle. Part b shows the corresponding bipartite graph $G_{\underline{\mathbf{NSCC}}}((Y, Z), E)$ and outlines one of its minimum hitting set with thick lines. Since five strongly connected components of $\overrightarrow{G}(X_{TU}, E_{TU})$ contain at least one T-vertex (i.e., C_3, C_5, C_6, C_7, C_8) and since the cardinality of a minimum hitting set on the bipartite graph $G_{\underline{\mathbf{NSCC}}}((Y, Z), E)$ is equal to 1 (i.e., we need one extra strongly connected component in order to connect the vertices of C_6 and C_7) we have $\underline{\mathbf{NSCC}} = 5 + 1 = 6$. Part c provides a final digraph where this lower bound of six strongly connected components is reached.

Estimating $\overline{\mathbf{NSCC}}$ The maximum number of strongly connected components is obtained by counting all strongly connected components of $\overrightarrow{G}(X_T, E_T)$ and all U-vertices (since U-vertices can be incorporated without creating any circuit by constructing a spanning forest).

$$\overline{\mathbf{NSCC}} = |scc(\overrightarrow{G}(X_{TU}, E_T))|$$

Example 10 Parts a and b of Fig. 9 illustrate how to compute the upper bound of **NSCC** according to the intermediate digraph $\overrightarrow{G}(X_{TU}, E_{TU})$ depicted by Part a.

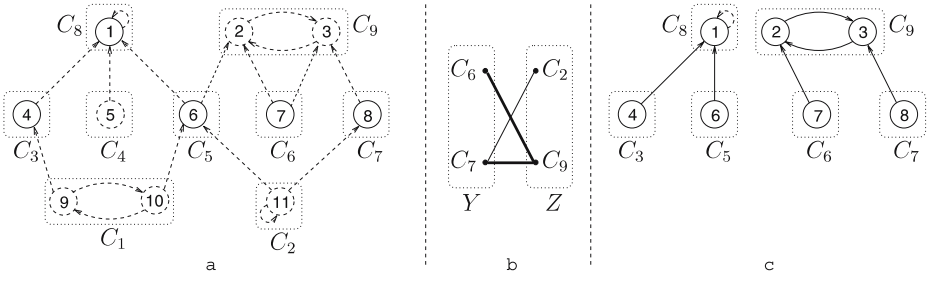

Fig. 8 **a** Intermediate digraph and **b** corresponding bipartite graph used to compute the cardinality of a minimum hitting set. **c** Example of a final digraph achieving the lower bound

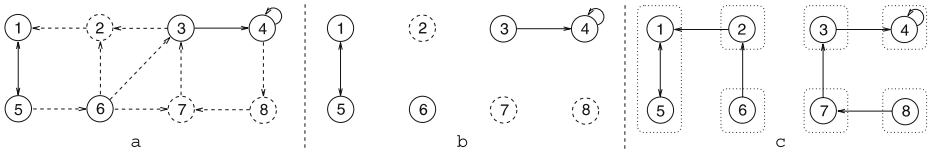

Fig. 9 **a** Intermediate digraph and **b** graph used for estimating $\overline{\mathbf{NSCC}}$. **c** Example of a final digraph achieving the upper bound

Part b gives the corresponding digraph $\overrightarrow{G}(X_{TU}, E_T)$, which has seven strongly connected components, thus $\overline{\mathbf{NSCC}} = 7$. Part c provides a final digraph where this upper bound of seven strongly connected components is reached.

Estimating NSINK The minimum number of sinks of any final digraph is equal to the number of T-vertices that are sinks in the intermediate digraph plus the minimum number of U-sink vertices necessary to turn into T-vertices in order to hinder non-sink T-vertices. This last quantity is equal to the cardinality of a minimum hitting set on the following bipartite graph $G'_r((Y, Z), E)$ stemming from the reduced digraph of $\overrightarrow{G}(X_{TU}, E_{TU})$ in the following way:

- Y denotes the set of strongly connected components $scc_{[|X_T|=1 \wedge |E_{TU}|=0]}(\overrightarrow{G}(X_{TU}, E_{TU}))$ such that:
  - For all $y \in Y$, y is not a sink in the reduced digraph.
  - All descendants in the reduced digraph of a vertex $y \in Y$ correspond to strongly connected components of $\overrightarrow{G}(X_{TU}, E_{TU})$ reduced to one single U-vertex with no incident arc, that is, belonging to $scc_{[|X_T|=0 \wedge |X_U|=1 \wedge |E_{TU}|=0]}(\overrightarrow{G}(X_{TU}, E_{TU}))$.
- Z denotes the set of strongly connected components in $scc_{[|X_T|=0 \wedge |X_U|=1 \wedge |E_{TU}|=0]}(\overrightarrow{G}(X_{TU}, E_{TU}))$ that are sinks in the reduced digraph.
- An edge $e = (y, z)$, $y \in Y$, $z \in Z$ belongs to E iff there is a path from y to z in the reduced digraph.

$$\underline{\mathbf{NSINK}} = |sink_{[|X_T|=1]}(\overrightarrow{G}(X_{TU}, E_{TU}))| + h(G'_r((Y, Z), E))$$

Example 11 Figure 10 illustrates how to compute the lower bound of **NSINK** according to the intermediate digraph $\overrightarrow{G}(X_{TU}, E_{TU})$ depicted by Part a. Part b gives the corresponding bipartite graph $G'_r((Y, Z), E)$, where $Y = \{6, 7\}$, $Z = \{14, 15\}$ and $E = \{(6, 14), (7, 14), (7, 15)\}$. Since $\overrightarrow{G}(X_{TU}, E_{TU})$ contains a single T-vertex that is a sink, $sink_{[|X_T|=1]}(\overrightarrow{G}(X_{TU}, E_{TU})) = \{\{13\}\}$ and thus $|sink_{[|X_T|=1]}(\overrightarrow{G}(X_{TU}, E_{TU}))| = 1$. The cardinality of the minimum hitting set, $h(G'_r((Y, Z), E))$, is equal to 1. As a consequence $\underline{\mathbf{NSINK}} = |sink_{[|X_T|=1]}(\overrightarrow{G}(X_{TU}, E_{TU}))| + h(G'_r((Y, Z), E)) = 2$. Part c provides a final digraph where this lower bound of two sinks is reached.

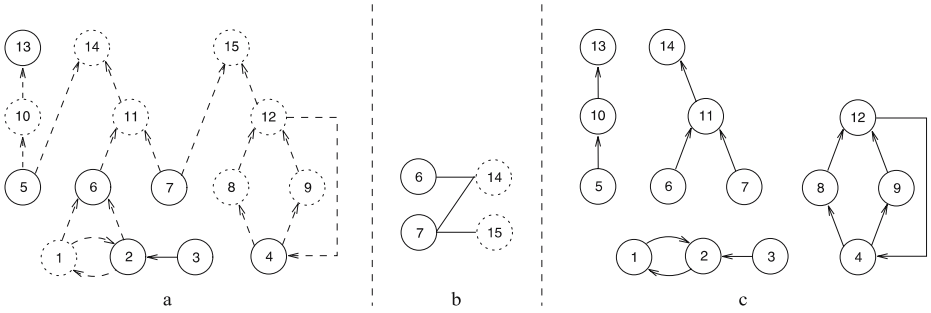

Fig. 10 **a** Intermediate digraph, **b** corresponding bipartite graph used to compute the cardinality of a minimum hitting set. **c** Example of a final digraph achieving the lower bound

Estimating NSINK In $\overrightarrow{G}(X_{TU}, E_{TU})$, let XP denote the set of U-vertices that are not sources and such that at least one successor is a sink having one single predecessor. We can remove from $|sink(\overrightarrow{G}(X_T, E_T))| + |X_U|$ the number of U-vertices that are sources in the intermediate digraph. Moreover, a pending vertex in $\overrightarrow{G}(X_{TU}, E_{TU})$ and its predecessor cannot both be sinks in a final digraph. Therefore we can also remove $|XP|$.

$$\overline{\textbf{NSINK}} = |sink(\overrightarrow{G}(X_T, E_T))| + |X_U| - |source_{[|X_U|=1]}(\overrightarrow{G}(X_{TU}, E_{TU}))| - |XP|$$

Example 12 Figure 11 illustrates how to compute the upper bound of **NSINK** according to an intermediate digraph. Since we do not have any T-vertex, $|sink(\overrightarrow{G}(X_T, E_T))| = 0$. Since the number of sources is equal to 3 and since $XP = \{3\}$ we have $|sink(\overrightarrow{G}(X_T, E_T))| + |X_U| - |source_{[|X_U|=1]}(\overrightarrow{G}(X_{TU}, E_{TU}))| - |XP| = 0 + 9 - 3 - 1 = 5$. But we can only get a maximum of four sinks since the leftmost and rightmost connected components can respectively generate at most two and two (and not three as suggested by the upper bound) sinks.

4.2.2 Synthesis

Table 1 recapitulates the different bounds. Except the upper bound on **NSINK**, all bounds are sharp [15]. Regarding complexity, the three non-polynomial bounds are the lower bounds of **NVERTEX**, **NSCC** and **NSINK** since they all involve the minimum hitting set [40] problem.[8] Note that many of the digraphs that express

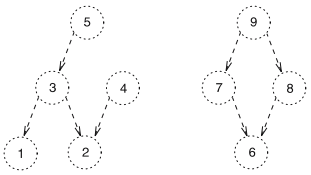

Fig. 11 An example of intermediate digraph for which the upper bound is not sharp

[8] As a consequence a lower bound of the minimum hitting set should be considered in practice.

Table 1 Bounds of the different graph parameters

Graph parameter	Bound	Sharpness	Complexity
NARC	$\lvert E_T\rvert + \lvert X_{T,\neg T}\rvert - \mu(\overleftrightarrow{G}(X_{T,\neg T}, E_U))$	yes	P
$\overline{\textbf{NARC}}$	$\lvert E_{TU}\rvert$	yes	P
NVERTEX	$\lvert X_T\rvert + h(\overleftrightarrow{G}((X_{T,\neg T,\neg T}, X_{U,\neg T,T}), E_{U,T}))$	yes	NP-complete
$\overline{\textbf{NVERTEX}}$	$\lvert X_{TU}\rvert$	yes	P
NCC	$\lvert cc_{[\lvert X_T\rvert\geq 1]}(\overrightarrow{G}(X_{TU}, E_{TU}))\rvert$	yes	P
$\overline{\textbf{NCC}}$	$\lvert cc_{[\lvert E_T\rvert\geq 1]}(\overrightarrow{G}(X_T, E_T))\rvert + \mu_l(\overleftrightarrow{G}_{rem})$	yes	P
NSCC	$\lvert scc_{[\lvert X_T\rvert\geq 1]}(\overrightarrow{G}(X_{TU}, E_{TU}))\rvert + h(G_{\underline{\textbf{NSCC}}}((Y, Z), E))$	yes	NP-complete
$\overline{\textbf{NSCC}}$	$\lvert scc(\overrightarrow{G}(X_{TU}, E_T))\rvert$	yes	P
NSINK	$\lvert sink_{[\lvert X_T\rvert=1]}(\overrightarrow{G}(X_{TU}, E_{TU}))\rvert + h(G'_r((Y, Z), E))$	yes	NP-complete
$\overline{\textbf{NSINK}}$	$\lvert sink(\overrightarrow{G}(X_T, E_T))\rvert + \lvert X_U\rvert$ $- \lvert source_{[\lvert X_U\rvert=1]}(\overrightarrow{G}(X_{TU}, E_{TU}))\rvert - \lvert XP\rvert$	no	P

a global constraint belong to specific graph classes for which such a problem, NP-hard in the general case, becomes polynomial. Section 5.2 shows how to simplify the formula in the context of a typical graph class that arises in practice.

The bounds $\min(P)$ and $\max(P)$ of the variable P of a graph parameter $\mathcal{P}$ involved in the description of a global constraint are respectively initialised to $\underline{P}$ and $\overline{P}$.

4.3 Filtering on Graph Properties

Given a graph property $\mathcal{P}\ op\ V$ occurring in the description of a global constraint, Table 2 gives the standard propagation rules for reducing the domains of V and P in order to enforce constraint $P\ op\ V$ (4) of the reformulation.

4.4 Filtering on Graph Invariants

Quite often, it happens that one wants to enforce the final digraph G_f of a global constraint C to satisfy more than one graph property. In this context, these graph

Table 2 Rules for reducing the domain of variables V and P involved in a graph property $P\ op\ V$

$P \leq V$	$\min(V) \leftarrow \max(\min(P), \min(V))$ $\max(P) \leftarrow \min(\max(V), \max(P))$
$P \geq V$	$\max(V) \leftarrow \min(\max(P), \max(V))$ $\min(P) \leftarrow \max(\min(V), \min(P))$
$P = V$	$\min(V) \leftarrow \max(\min(P), \min(V)) \wedge \max(V) \leftarrow \min(\max(P), \max(V))$ $\min(P) \leftarrow \max(\min(V), \min(P)) \wedge \max(P) \leftarrow \min(\max(V), \max(P))$
$P \neq V$	$\min(P) = \max(P) \Rightarrow \text{dom}(V) \leftarrow \text{dom}(V) \setminus \{\min(P)\}$ $\min(V) = \max(V) \Rightarrow \text{dom}(P) \leftarrow \text{dom}(P) \setminus \{\min(V)\}$

properties involve several graph parameters that cannot vary independently. So, in order to get stronger necessary conditions for the feasibility of C we can search for graph invariants that link the different graph parameters. These graph invariants are typically inequalities between one graph parameter and an arithmetic expression mentioning other graph parameters.[9] In this context, each graph invariant will be used in order to adjust the minimum $\min(P)$ and the maximum $\max(P)$ values of its graph parameter. Remember that these values were initialised by using the formula of Section 4.2.

Furthermore, it happens quite often that the final digraph G_f associated with a global constraint has a regular structure, which comes from its initial digraph or from a property of its arc constraint *ctr*. According to a given graph class, initial graph parameters bounds can be refined, as shown in Section 5.2. Similarly, tighter graph invariants holding for a graph class can be used as additional constraints in the reformulation of Section 4.1, in order to improve the filtering.

Example 13 Consider again the `nvalue` constraint introduced in Example 1. Bessière et al. [23] give a necessary condition based on a result by Turán [75]. For the final digraph G_f of the `nvalue` constraint, which is symmetric, reflexive and transitive,[10] this necessary condition[11] links the *number of strongly connected components*, the *number of vertices* and the *number of arcs* of G_f: $\mathbf{NSCC} \geq \left\lceil \frac{\mathbf{NVERTEX}^2}{\mathbf{NARC}} \right\rceil$. This allows to evaluate the minimum number of distinct values according to the number of variables of the `nvalue` constraint (i.e., **NVERTEX**, which is fixed to the number of variables of the `nvalue` constraint since each vertex of the initial digraph belongs to the final digraph) and to the maximum number of arcs of its intermediate digraph (i.e., **NARC**).

To conclude this section, we propose the following research path, which is closely related to the third research path proposed in [42].

RP 2. Organise a site for interactive addition to and consultation of a *database of graph invariants as well as bounds of graph parameters*. These invariants and bounds might be valid for all digraphs,[12] digraphs with no isolated vertex[13] or specific graph classes that show up in the context of global constraints.

Such a data base could constitute an important resource for dealing with the combinatorial aspect of global constraints.

[9] See Chapter 3 of [11] for a collection of about 200 graph invariants.

[10] G_f consists of one or several cliques.

[11] We recast the original condition given in [23] to the context of the intermediate digraph of the `nvalue` constraint.

[12] This is the case of the CP(Graph) framework [32].

[13] This is actually the case for the global constraint catalogue [11].

5 Toward Graph-based Filtering

The work presented in the previous section constitutes a first attempt in the direction of generic filtering algorithms. The goal of this section is to highlight natural enhancements and to show how they could be combined with previous work: Section 5.1 first shows how to filter according to the lower and upper bounds of a graph parameter; We then discuss in Section 5.2 the need for considering specific graph classes in order to simplify the lower and upper bounds presented in the previous section; Finally, Section 5.3 presents the structure of a generic filtering scheme that combines all the elements introduced in Sections 4 and 5. All this will be done by considering small concrete examples.

5.1 Filtering According to Bounds of Graph Parameters

Maintaining constraints (3) of the reformulation leads to filtering arc and vertex variables (arc_{jk} and $vertex_j$) according to the minimum and maximum values of a graph parameter. In other words, we aim at determining the status of U-vertices and U-arcs of the intermediate digraph so that no final digraph contains more than (resp. less than) a given fixed number of arcs, vertices, connected components, strongly connected components, or sinks. In principle one could use shaving, i.e., fix the status of each U-vertex and each U-arc and check if the formulae of Section 4.2 do not lead to a contradiction. Since this is very costly in practice, one should rather try to characterise the propagation that can be achieved when a graph parameter variable P is fixed to one of its extreme values $\underline{P}$ or $\overline{P}$. In order to illustrate this idea we consider the formula giving the lower and upper bounds of **NCC**. Filtering for the other usual graph parameters **NARC** and **NVERTEX** are presented in [17]. Propositions 3 and 4 correspond to the case where the number of connected components is equal to the values, respectively, $\overline{\mathbf{NCC}}$ and $\underline{\mathbf{NCC}}$ given in Table 1. The proofs of these propositions are available in Part A of the Appendix. Proposition 3 reuses the graph $\overleftrightarrow{G}_{rem}$ introduced in Section 4.2.1 for computing $\overline{\mathbf{NCC}}$.

Proposition 3 *If* $\mathrm{dom}(\mathbf{NCC}) = \{\ \overline{\mathbf{NCC}}\ \}$, *with* $\overline{\mathbf{NCC}} = |cc_{[|E_T|\geq 1]}(\overrightarrow{G}(X_T, E_T))| + \mu_l(\overleftrightarrow{G}_{rem})$, *then:*

1. *All U-arcs joining two T-vertices belonging to two distinct connected components of $cc_{[|E_T|\geq 1]}(\overrightarrow{G}(X_T, E_T))$ can be turned into F-arcs.*
2. *For all edges in $\overleftrightarrow{G}_{rem}$ that do not belong to any maximum l-matching of $\overleftrightarrow{G}_{rem}$, the corresponding U-arcs can be turned into F-arcs.*
3. *All arcs $e = (u, v) \in E_U$ such that u belongs to a connected component of $cc_{[|E_T|\geq 1]}(\overrightarrow{G}(X_T, E_T))$ and v is saturated in all maximum l-matchings of $\overleftrightarrow{G}_{rem}$ can be turned into F-arcs.*
4. *All vertices v of $\overleftrightarrow{G}_{rem}$ that belong to all maximum l-matchings of $\overleftrightarrow{G}_{rem}$ can be turned into T-vertices.*
5. *For all edges e belonging to all maximum l-matchings of $\overleftrightarrow{G}_{rem}$, if a unique U-arc in $\overrightarrow{G}(X_{TU}, E_U)$ corresponds to e then this arc can be turned into a T-arc.*

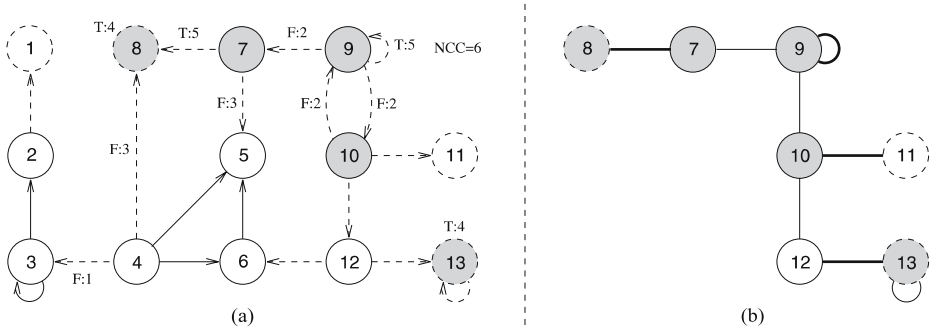

Fig. 12 Illustration of Proposition 3

Proposition 4 *If* dom(**NCC**) = { **NCC** }, *with* **NCC** = $|cc_{[|X_T|\geq 1]}(\overrightarrow{G}(X_{TU}, E_{TU}))|$, *then:*

1. *All U-arcs and U-vertices in* $\overrightarrow{G}(X_{TU}, E_{TU}))$ *that belong to a connected component involving neither T-vertices nor T-arcs, can be turned into F-arcs and F-vertices.*
2. *All U-vertices that are articulation points of* $\overleftrightarrow{G}(X_{TU}, E_{TU})$ *and that belong to an elementary chain between two T-vertices can be turned into T-vertices.*
3. *For all edges e of* $\overleftrightarrow{G}(X_{TU}, E_{TU})$ *that are bridges belonging to an elementary chain between two T-vertices, if a unique U-arc in* $\overrightarrow{G}(X_{TU}, E_{TU})$ *corresponds to e then this U-arc can be turned into a T-arc.*

We illustrate these propositions on two examples. In illustrative figures, U-vertices and U-arcs for which the status is determined are systematically depicted with the number of the corresponding item and their new status. As for the previous section, we assume that the intermediate digraph $\overrightarrow{G}(X_{TU}, E_{TU})$ is normalised according to constraints (2).

Example 14 Part a of Fig. 12 illustrates Proposition 3 according to the hypothesis that the final digraph should contain at least six connected components. $cc_{[|E_T|\geq 1]}(\overrightarrow{G}(X_T, E_T))$ consists of the two connected components {2, 3} and {4, 5, 6}. Part b illustrates the undirected graph $\overleftrightarrow{G}_{rem}$, where thick lines correspond to a maximum l-matching of cardinality 4, and grey vertices are vertices that are saturated in all maximum l-matchings. Since the precondition of Proposition 3 holds, Items 1, 2 and 3 respectively turn the U-arcs in {(4, 3)}, {(9, 7), (9, 10), (10, 9)} and {(4, 8), (7, 5)} into F-arcs. Item 4 turns the U-vertices {8, 13} into T-vertices. Finally, Item 5 turns the U-arcs {(7, 8), (9, 9)} into T-arcs.

Example 15 Part a of Fig. 13 illustrates Proposition 4 according to the hypothesis that the final digraph should contain no more than one connected component. Part b represents the undirected graph $\overleftrightarrow{G}(X_{TU}, E_{TU})$, where grey vertices correspond to articulation points and thick lines correspond to bridges. Since $\overrightarrow{G}(X_{TU}, E_{TU})$

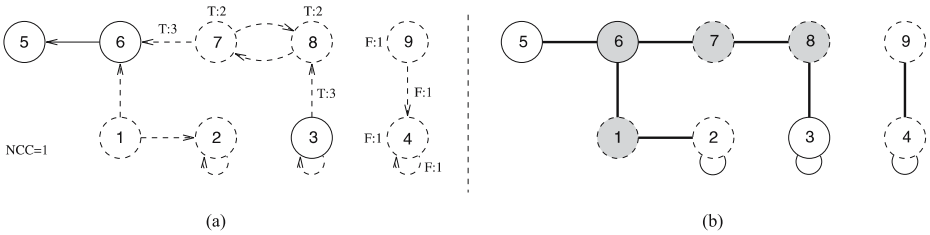

Fig. 13 Illustration of Proposition 4

contains one single connected component involving at least one T-vertex the precondition of Proposition 4 holds. Since the connected component of $\overrightarrow{G}(X_{TU}, E_{TU})$ corresponding to U-vertices 4 and 9 does not contain any T-vertex, then from Item 1, U-vertices 4 and 9 as well as U-arcs $(4, 4)$ and $(9, 4)$ are turned into F-vertices and F-arcs. From Item 2, the two U-vertices 7 and 8, which are articulation points of $\overleftrightarrow{G}(X_{TU}, E_{TU})$ belonging to an elementary chain between two T-vertices (3 and 6), are turned into T-vertices. From Item 3, among the three bridges of $\overleftrightarrow{G}(X_{TU}, E_{TU})$ belonging to an elementary chain between two T-vertices (3 and 6), the arcs $(3, 8)$ and $(7, 6)$ are turned into T-arcs since their respective counterpart $(8, 3)$ and $(6, 7)$ does not belong to $\overrightarrow{G}(X_{TU}, E_{TU})$.

To conclude, we present a research path that considers the problem of a modular implementation of such bound-based filtering algorithms. Unlike graph invariants, which can be directly modelled by arithmetic constraints, the filtering rules associated with each bound of each graph parameter seem to require ad hoc developments. This would be contradictory to the goal of synthesising filtering algorithms. However, most existing graph-based filtering algorithms rely on:

- A restricted number of graph transformations (e.g., union of graphs, induced graph, reduced digraph, ...),
- And a limited set of basic graph algorithms to apply to these transformations. They typically compute a graph parameter (e.g., cardinality of a maximum matching, maximum flow, ...) or identify subsets of vertices and arcs with given properties (e.g., bridges, edges and/or vertices saturated in every maximum matching, ...).

This suggests the following research path.

RP 3. Design a *compositional language* and find out the set of basic graph transformations and algorithms that are required in order to describe a lot of graph-based filtering algorithms. Create an *abstract machine* that can evaluate such a language.

5.2 Bounds for Special Graph Classes

For specific graph classes that arise in practice the lower bound or the upper bound of a given graph parameter can usually be greatly simplified. This can typically lead

Table 3 Simplified bounds of the different graph parameters in the context of the *PATH* arc-generator

Graph parameter	Bound	Sharpness	Complexity
$\underline{\mathbf{NARC}}_{PATH}$	$\lvert E_T\rvert + \sum_{i \in cc(\overrightarrow{G}(X_{T,\neg T}, E_U))} \lfloor \frac{\lvert vertex(i)\rvert+1}{2} \rfloor$	yes	P
$\underline{\mathbf{NVERTEX}}_{PATH}$ $\underline{\mathbf{NSCC}}_{PATH}$	$\lvert X_T\rvert + \sum_{i \in cc(\overrightarrow{G}(X_{T,\neg T,\neg T \cup X_{U,U,\neg U}}, E_U))} \lfloor \frac{\lvert vertex_{[\lvert X_T\rvert=1]}(i)\rvert+1}{2} \rfloor$	yes	P
$\overline{\mathbf{NCC}}_{PATH}$ $\overline{\mathbf{NSINK}}_{PATH}$	$\lvert cc_{[\lvert E_T\rvert \geq 1]}(\overrightarrow{G}(X_T, E_T))\rvert + \sum_{i \in cc(\overleftrightarrow{G}_{rem})} \lfloor \frac{\lvert vertex(i)\rvert}{2} \rfloor$	yes	P
$\overline{\mathbf{NSCC}}_{PATH}$	$\lvert X_{TU}\rvert$	yes	P
$\underline{\mathbf{NSINK}}_{PATH}$	$\lvert cc_{[\lvert X_T\rvert \geq 1]}(\overrightarrow{G}(X_{TU}, E_{TU}))\rvert$	yes	P

to more efficient algorithms for evaluating these bounds. We illustrate this point in the following context:

- When the final digraph corresponds to a set of cliques. This is the case for constraints such as `alldifferent` and `nvalue`.
- When the final digraph has to be a subgraph of an elementary path.[14] In Table 3, we revisit the bounds of Table 1 that cannot be evaluated in linear time (i.e., all bounds except $\overline{\mathbf{NARC}}$, $\overline{\mathbf{NVERTEX}}$ and $\underline{\mathbf{NCC}}$).

5.2.1 A Set of Cliques

When the initial digraph is a clique and when the arc constraint defines an equivalence relation $\mathcal{R}$ (e.g. an equality constraint), then a final digraph is a set of cliques, where each clique represents an equivalence class of $\mathcal{R}$. This is for instance the case for the `alldifferent`, the `not_all_equal` or the `nvalue` constraints, which all involve the graph parameter **NCC**. Assume now that you want to evaluate the maximum number of connected components within the context of this class of graphs. The upper bound of **NCC** we gave early ($\lvert cc_{[\lvert E_T\rvert \geq 1]}(\overrightarrow{G}(X_T, E_T))\rvert + \mu_l(\overleftrightarrow{G}_{rem})$) is still valid but not sharp anymore. But we can get back to a sharp bound by computing the cardinality of a maximum matching of the following bipartite graph:

- To each variable associated with a vertex of the initial digraph we create a vertex. To each equivalence class defined by $\mathcal{R}$ we also create a vertex.
- There is an edge between a variable and an equivalence class if and only if the variable can be assigned to a value that belongs to that equivalence class.

[14] This is actually the case when we use the *PATH* arc-generator described in [11] for generating the initial digraph associated with a global constraint.

5.2.2 A Set of Elementary Paths

Given a graph parameter P, the new lower (resp. upper) bound of P is denoted by $\underline{P}_{PATH}$ (resp. $\overline{P}_{PATH}$). Proofs are available in Part B of the Appendix. Graph parameters $\mathbf{NSCC}_{PATH}$ and $\mathbf{NVERTEX}_{PATH}$ are identical in the context of a set of elementary paths, as well as $\mathbf{NSINK}_{PATH}$ and $\mathbf{NCC}_{PATH}$. Observe that all these bounds are sharp and can be evaluated in $O(n)$ where n is the number of vertices of the initial digraph. This is a big improvement over the general bounds given by Table 1 since:

- The three non-polynomial bounds involving the minimum hitting set problem and the two bounds involving a matching have been simplified.
- The bound on the maximum number of sinks is now sharp.

Example 16 Figure 14 illustrates the formula introduced in Table 3 for computing the minimum number of arcs, the minimum number of vertices and the maximum number of connected components according to the intermediate digraph depicted by Part a, where the status of each vertex is given on top or below it:

- Part b1 shows the graph $\overrightarrow{G}(X_{T,\neg T}, E_U)$, which has four connected components respectively involving four, two, one and one vertices. Since the intermediate digraph contains one T-arc we get a minimum number of arcs equal to $1 + \lfloor \frac{4+1}{2} \rfloor + \lfloor \frac{2+1}{2} \rfloor + \lfloor \frac{1+1}{2} \rfloor + \lfloor \frac{1+1}{2} \rfloor = 6$. Part b2 provides a final digraph where this lower bound of six arcs is reached.
- Part c1 shows the graph $\overrightarrow{G}(X_{T,\neg T,\neg T} \cup X_{U,U,\neg U}, E_U)$ which has two connected components respectively involving zero and two T-vertices. Since the intermediate digraph contains ten T-vertices we get a minimum number of vertices equal to $10 + \lfloor \frac{0+1}{2} \rfloor + \lfloor \frac{2+1}{2} \rfloor = 11$. Part c2 gives a final digraph where this lower bound of 11 vertices is reached.

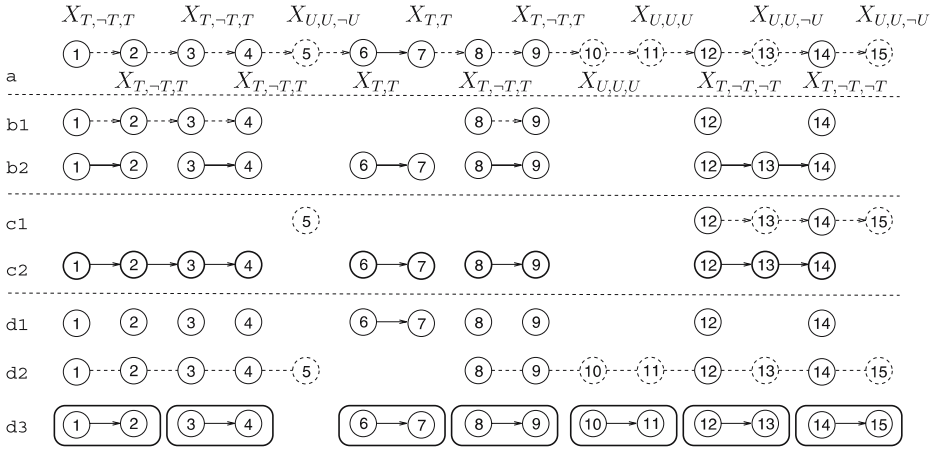

Fig. 14 **a** Intermediate digraph and corresponding graphs **b1** $\overrightarrow{G}(X_{T,\neg T}, E_U)$, **c1** $\overrightarrow{G}(X_{T,\neg T,\neg T} \cup X_{U,U,\neg U}, E_U)$, **d1** $\overrightarrow{G}(X_T, E_T)$, **d2** $\overleftrightarrow{G}_{rem}$. Examples of final digraphs achieving a minimum number of arcs **b2**, a minimum number of vertices **c2**, and a maximum number of connected components **d3**

– Parts d1 and d2 respectively depict the graphs $\overrightarrow{G}(X_T, E_T)$ and $\overleftrightarrow{G}_{rem}$. Since $\overrightarrow{G}(X_T, E_T)$ contains one single connected component containing at least one T-arc, and since $\overleftrightarrow{G}_{rem}$ has two connected components respectively involving five and eight vertices we get a maximum number of connected components equal to $1 + \lfloor \frac{5}{2} \rfloor + \lfloor \frac{8}{2} \rfloor = 7$. Part d3 gives a final digraph where this upper bound of seven connected components is reached.

5.2.3 Research Improvement

To conclude this section we suggest the following research improvement.

PE 2. Similarly to what we just did for the *PATH* arc-generator, come up with simplified bounds for other graph classes, described in Section 3.2.1 of the global constraint catalogue [11].

5.3 Putting Everything Together

In the context of graph-based filtering, this section shows how to put together the different pieces that were previously introduced in order to get a first generic filtering scheme from the graph-based representation of a global constraint. W.l.o.g. we assume that the global constraint $C(V_1, \ldots, V_n, x_1, \ldots, x_m)$ under consideration is defined as a conjunction of graph properties of the form $\mathcal{P}_l\ op_l\ V_l$ $(1 \leq l \leq n)$ on a single initial digraph $G_{\mathcal{R}}$. c denotes the graph class associated with the final digraph of the constraint C, and *ctr* represents the binary constraint associated with any arcs in E_i. The generic filtering scheme given below implements the reformulation stated by Proposition 2.

Algorithm 1 Implementing Proposition 2.

for the initial digraph $G_i = (X_i, E_i)$ of C **do**
  for each arc $e_{jk} \in E_i$ **do**
    Create a 0-1 variable arc_{jk};
    Post the *arc-constraint* $arc_{jk} = 1 \Leftrightarrow ctr(x_j, x_k)$;
  for each vertex $v_j \in X_i$ **do**
    Create a 0-1 variable $vertex_j$;
    Post the *normalisation* constraint
      $vertex_j = \min(1, \sum_{\{k \mid e_{jk} \in E_i\}} arc_{jk} + \sum_{\{k \mid e_{kj} \in E_i\}} arc_{kj})$;
  for each graph parameter $\mathcal{P}_l$ $(1 \leq l \leq n)$ **do**
    Create the variable P_l;
    Post the *graph property constraint* $P_l\ op_l\ V_l$;
Renormalises according to the possible graph-class c of C;
for each graph parameter $\mathcal{P}_l$ $(1 \leq l \leq n)$ **do**
    Post the *graph parameter constraint* $ctr_{\mathcal{P}_l}(Vertex, Arc, P_l)$;
Post all *graph invariants* involving parameters from $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n\}$;

Algorithm 1 introduces new domain variables and posts various propagators on those variables. It consists of the following parts:

- An *initialisation phase* (lines 01–10) creates the vertex and arc variables (respectively denoted by *Vertex* and *Arc*) associated with the initial graph and states the arc constraints (1) as well as the normalisation constraints (2). It also posts all graph property constraints (4) $P_l\ op_l\ V_l$, which link a graph parameter variable P_l to a variable V_l of the global constraint C (see Table 2).
- A *renormalisation phase* (line 11): For each specific graph class c, an additional renormalisation constraint depending of c is stated. For instance, if the final digraph has to be symmetric, we post for each arc $e_{jk} \in E_i$ constraint $arc_{jk} = arc_{kj}$.
- A *local propagation phase*[15] (lines 12,13) posts for each graph parameter $\mathcal{P}_l$ ($1 \leq l \leq n$) a constraint (3) corresponding to $ctr_{\mathcal{P}_l}(Vertex, Arc, P_l)$. Algorithm 2 shows how such a propagator is implemented: we first evaluate the lower and upper bounds $\underline{P}$ and $\overline{P}$ of the graph parameter $\mathcal{P}$, update the minimum and maximum value of P and possibly identify vertex and arc variables to fix in order to enforce that $P \in [\underline{P}, \overline{P}]$.
- A *global propagation phase*[16] (line 14) posts all graph invariants involving several graph parameters from $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n\}$.

Algorithm 2 Propagator associated with a graph parameter constraint $ctr_{\mathcal{P}}(Vertex, Arc, P)$.

```
01 Evaluate P and P̄ according to the intermediate digraph;
02 min(P) ← max(P, min(P));
03 max(P) ← min(P̄, max(P));
04 if min(P) = max(P) = P̄ then
05    Identify some vertex and arc-variables to fix in order to avoid P < P̄;
06 if min(P) = max(P) = P then
07    Identify some vertex and arc-variables to fix in order to avoid P > P;
```

5.4 Linking Up with Ad hoc Filtering

One may wonder whether the generic graph-based filtering is not too theoretical in order to have any practical interest. We show that we can in fact retrieve several ad hoc algorithms that were constructed for specific global constraints. For this purpose, we first consider the `proper_forest` constraint for which a specialised filtering algorithm was recently proposed in [18]. After recalling the different steps of this algorithm, we "deconstruct" this algorithm and reinterpret its parts in terms of the generic graph-based filtering. Similarly, we consider the `alldifferent` constraint. Note that in [17], we also considered the `among` constraint introduced in [6]: we then retrieved the specialised filtering algorithm that was proposed by Bessière et al. in [22].

5.4.1 Retrieving the Filtering Algorithm of `proper_forest`

Example 17 Consider the `proper_forest(NTREE, VER)` constraint introduced in [18], which takes a domain variable `NTREE` and an initial digraph $G_{\mathcal{R}}$ described

[15] We call it *local* since it involves one single graph parameter.

[16] We call it *global* since it involves several graph parameters.

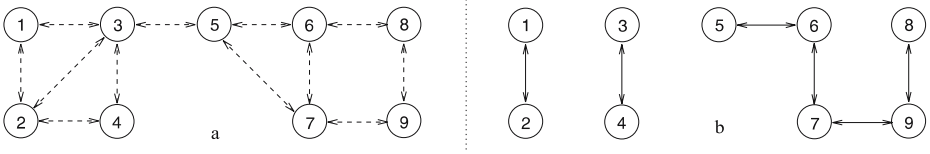

Fig. 15 **a** A digraph and **b** a solution with three proper trees for the `proper_forest` constraint corresponding to parts **a** and **b** of Table 4

by a collection of vertices `VER`. The `proper_forest(NTREE, VER)` constraint partitions the vertices of $G_{\mathcal{R}}$ into a set of vertex-disjoint proper trees (i.e., trees that have at least two vertices each). Each item $v \in$ `VER` has the following attributes:

- `I` is an integer between 1 and m (i.e., the total number of vertices of $G_{\mathcal{R}}$), which can be interpreted as the *label* of v.
- `N` is a set variable whose elements are integers (vertex labels) between 1 and m. The lower and upper bounds of `N` can respectively be interpreted as the *set of mandatory neighbours* and the *set of mandatory or potential neighbours* of v.

Part a of Fig. 15 shows the initial digraph $G_{\mathcal{R}}$. Part b of the figure shows a possible solution on this digraph with three proper trees.

We now describe the `proper_forest` constraint in terms of graph properties. A final digraph G_f of the `proper_forest` constraint belongs to the class of symmetric graphs[17] and has no loop. Moreover G_f has to fulfill the following graph properties on **NVERTEX**, **NARC** and on **NCC**: **NVERTEX** $= m$ (since we have a vertex portioning problem, all vertices of the initial digraph should belong to the final digraph[18]), **NARC** $= 2 \cdot (m -$ `NTREE`$)$ (2 since G_f is symmetric, and $m -$ `NTREE` since we have `NTREE` acyclic connected digraphs) and **NCC** $=$ `NTREE`.

Table 4 Domains of the variables for the `proper_forest` constraint corresponding to parts a and b of Fig. 15

i	dom(`VER`[i].`N`) (A)	dom(`VER`[i].`N`) (B)	i	dom(`VER`[i].`N`) (A)	dom(`VER`[i].`N`) (B)
1	{2, 3}	{2}	6	{5, 7, 8}	{5, 7}
2	{1, 3, 4}	{1}	7	{5, 6, 9}	{6}
3	{1, 2, 4, 5}	{4}	8	{6, 9}	{9}
4	{2, 3}	{3}	9	{7, 8}	{7, 8}
5	{3, 6, 7}	{6}			

[17] A digraph is *symmetric* iff, if there is an arc from u to v, there is also an arc from v to u.

[18] Since the initial digraph does not contains any loops and since isolated vertices are forbidden in a final digraph, each vertex of the initial graph belongs to a connected component of the final digraph involving at least two vertices.

The specialised filtering algorithm of the proper_forest constraint is made up from the following steps:

1. Renormalise $\overrightarrow{G}(X_{TU}, E_{TU})$ according to the fact that the final digraph has to be symmetric.
2. Check the feasibility of the proper_forest constraint:
   (a) The intermediate digraph has no isolated vertex.
   (b) There is no cycle made up from T-arcs.
   (c) NTREE has at least one value in [MINTREE, MAXTREE] where MINTREE is the number of connected components of the intermediate digraph and MAXTREE is the number of connected components of $\overrightarrow{G}(X_T, E_T)$ plus the cardinality of a maximum cardinality matching in the subgraph induced by the vertices that are not incident on any T-vertices.
3. Every U-arc that would create a cycle of T-vertices is turned into an F-arc.
4. The minimum and maximum values of NTREE are respectively adjusted to MINTREE and MAXTREE.
5. When NTREE is fixed to MINTREE all U-arcs corresponding to bridges of $\overrightarrow{G}(X_{TU}, E_{TU})$ are turned into T-arcs.
6. When NTREE is fixed to MAXTREE each U-arc (u, v) satisfying one of the following conditions is turned into an F-arc:
   (a) Both u and v belong to two distinct connected components of $\overrightarrow{G}(X_T, E_T)$ involving more than one vertex.
   (b) (u, v) does not belong to any maximum matching in the subgraph induced by the vertices that are not incident on any T-vertices.
   (c) u is the extremity of a T-arc and v is saturated in every maximum matching in the subgraph induced by the vertices that are not incident on any T-vertices.
7. Every U-arc involving a source or a sink is turned into a T-arc.

By considering the generic graph-based reformulation of Proposition 2 on the graph property NTREE $=$ **NCC**, we retrieve almost all the steps of the previous algorithm (except steps 2(b) and 3, which come from the invariant linking two graph parameters **NARC** $= 2 \cdot (m -$ **NCC**$)$):

- Item (1) corresponds to posting the renormalisation constraints (5) according to the possible graph-class c of C (in the context of proper_forest, the final digraph has to be symmetric).
- Item (2.a) corresponds to posting the normalisation constraints (2).
- Item (2.c) corresponds to checking the graph property constraint **NCC** $=$ NTREE (4). MINTREE and MAXTREE respectively correspond to the general lower and upper bounds of **NCC** given in Table 1 and deduced from constraint (3).
- Item (4) corresponds to the propagation induced by the graph property constraint (4).
- Item (5) and Item (6) correspond to propagating constraint (3) on the graph parameter **NCC**: Item (5) is the specialisation of Theorem 4, namely its third item (since the first two items of Theorem 4 are irrelevant because $\overrightarrow{G}(X_{TU}, E_{TU})$ does not contain any U-vertex). Item (6) corresponds to Theorem 3.

– Finally, Item (7) corresponds to the propagation obtained by the graph-class constraint (5), which avoids the creation of any isolated vertex in the symmetric graph.

5.4.2 Retrieving the Filtering Algorithm of `alldifferent`

The `alldifferent`($\mathtt{V}_1, \mathtt{V}_2, \ldots, \mathtt{V}_n$) enforces all variables of $V_1, V_2, \ldots, V_n$ to be assigned to distinct values. Its initial digraph is a clique where each vertex corresponds to a variable of $V_1, V_2, \ldots, V_n$. An equality constraint is associated to each arc and consequently the final digraph is a set of cliques where each clique corresponds to the variables that are assigned to the same value. The graph property $\mathbf{NCC} = \mathtt{n}$ is used in order to have n distinct values. Since **NCC** cannot exceed n it remain to enforce that $\mathbf{NCC} \geq \mathtt{n}$. For this purpose, we have to evaluate an upper bound of **NCC**. We just saw in Section 5.2.1 how this turned out to evaluate the cardinality of a maximum matching within a specific bipartite graph.

5.5 Adaptation to CP(Graph)

The CP(Graph) framework [32] introduces graph variables as well as constraints on these variables for representing some constraints related to graphs. Similarly to what was presented for the graph-based representation, the CP(Graph) framework represents a global constraint as the search for a subgraph of an initial digraph, so that this subgraph satisfies some graph properties. In this context, the lower and upper bounds of a graph variable of the CP(Graph) framework can be respectively reinterpreted as the graphs $\overrightarrow{G}(X_T, E_T)$ and $\overrightarrow{G}(X_{TU}, E_{TU})$.

A first minor difference between the two approaches is that the graph-based representation forbids isolated vertices in the final digraph, which is not the case for CP(Graph). A second difference is that no binary constraint is associated with each arc, as is currently the case for the graph-representation (see constraints (1) of Proposition 2).

From a filtering perspective, both the CP(Graph) framework and the graph-based representation have to find out which arcs and vertices will effectively belong to the final digraph. We now show how one can easily adapt the results of Sections 4 and 5.3 to the context of CP(Graph). Assume that a graph variable is represented by a set of set variables (one set variable S_j for each vertex v_j of the initial digraph). The lower and upper bounds of such a variable S_j represents the mandatory successors as well as the potential successors of v_j. Within Proposition 2, we have to replace constraints (1) by $arc_{jk} = 1 \Leftrightarrow k \in S_j$. Since isolated vertices are now allowed and since if an arc exists, then its endpoints must also exist, we have also to replace the symbol $=$ of constraints (2) by $\geq$. We keep unchanged the other items of Proposition 2. Regarding the bound computations, we have now to consider that isolated vertices may belong to the final digraph. All bounds of Section 4.2.1 can be easily adapted to this requirement. This leads to simplified bounds summarised in Table 5. All previous bounds are sharp. Regarding complexity, all the bounds are polynomial but **NSINK**.

Graph invariants can also be used, provided that one revisits the data base of graph invariants in order to sort out which invariants are still valid and which invariants need to be adapted to the context of isolated vertices. Finally, the generic filtering

Table 5 Bounds of the different graph parameters in the context of CP(Graph)

Graph parameter	Bound	Sharpness	Complexity
NARC	$\lvert E_T \rvert$	yes	P
$\overline{\textbf{NARC}}$	$\lvert E_{TU} \rvert$	yes	P
NVERTEX	$\lvert X_T \rvert$	yes	P
$\overline{\textbf{NVERTEX}}$	$\lvert X_{TU} \rvert$	yes	P
NCC	$\lvert cc_{[\lvert X_T \rvert \geq 1]}(\overrightarrow{G}(X_{TU}, E_{TU})) \rvert$	yes	P
$\overline{\textbf{NCC}}$	$\lvert cc(\overrightarrow{G}(X_T, E_T)) \rvert + \lvert X_U \rvert$	yes	P
NSCC	$\lvert scc_{[\lvert X_T \rvert \geq 1]}(\overrightarrow{G}(X_{TU}, E_{TU})) \rvert$	yes	P
$\overline{\textbf{NSCC}}$	$\lvert scc(\overrightarrow{G}(X_{TU}, E_T)) \rvert$	yes	P
NSINK	$\lvert sink_{[\lvert X_T \rvert = 1]}(\overrightarrow{G}(X_{TU}, E_{TU})) \rvert + h(G'_r((Y, Z), E))$	yes	NP-complete
$\overline{\textbf{NSINK}}$	$\lvert sink(\overrightarrow{G}(X_T, E_T)) \rvert + \lvert X_U \rvert$	yes	P

scheme introduced in Section 5.3 still applies, modulo some minor modifications of Algorithm 1 (i.e., replace line 04 by $arc_{jk} = 1 \Leftrightarrow k \in S_j$, and remove line 07).

6 Other Research Lines Closely Related to the Catalogue

This section presents different lines of research that could be built on top of the existing catalogue of global constraints. These lines of research deal respectively with *constraint reformulation*, *constraint visualisation*, *constraint-oriented heuristics*, and *constraint seeker*. We now successively go through these four topics.

6.1 Constraint Reformulation

Constraint reformulation has been identified as a key technique that is required in order to ease the use of Constraint Programming technology. Modelling a specific problem is not an easy task and can dramatically affect the performance. An additional difficulty stems from the fact that modelling is not independent of the targeted constraint system. As a consequence a lot of effort was devoted to develop various modelling techniques for specific problems.[19] As a matter of fact these techniques are currently disseminated in the head of constraint experts and/or in articles and are only used on a case by case basis. Only recently some effort was devoted into the development of automatic reformulation techniques and concrete system as well as to the automatic compilation from high-level modelling languages to a given constraint system [21, 28, 39, 44]. But the use of global constraints was not yet considered in such a system. This is quite surprising since the concept of global constraint seems to be a good support for automatic constraint reformulation: in fact, quite a lot of global constraints have been introduced in order to handle

[19] In fact an annual workshop is devoted to this topic (i.e., see http://4c.ucc.ie/~brahim/cp05ws/).

a conjunction of constraints that occur in a recurring way.[20] This was for instance the case of the `alldifferent` constraint, which replaces a clique of disequality constraints. But this was also the case of the `lex_chain_less` [26], the `diffn` [6] or the `colored_matrix` [71] constraints, which can respectively be decomposed as a conjunction of lexicographic ordering constraints, of non-overlapping constraints or of `global_cardinality` [69] constraints. In fact lexicographic ordering, non-overlapping or `global_cardinality` constraints are much more known than their global counterparts `lex_chain_less`, `diffn` or `colored_matrix`.

The concept of global constraint can also catch well-known necessary conditions that are widespread all over scientific articles. We illustrate this by the following examples:

- Consider the `cumulative` [1] constraint. A well-know necessary condition is obtained by extracting all disjunctive tasks and stating explicitly that they are in disjunction. A first simple extraction method is to consider all tasks that use more than the half of the resource limit. More sophisticated methods are available when we have more than one cumulative resource [2, 3].
- Consider the two-dimensional case of the `diffn` constraint, which allows for expressing the fact that a set of rectangles do not pairwise overlap. A well-known necessary condition based on the `cumulative` constraint is depicted in [1].
- Consider systems of `alldifferent` constraints [34] (i.e., several `alldifferent` constraints sharing some variables). It is sometimes possible to get stronger propagation, which catches the interaction of several `alldifferent` constraints by using other global constraints such as `nvalue` [58] or `same_and_global_cardinality` [14].

In order to integrate global constraints into an automatic reformulation process, a possibility is to associate to each global constraint a set of transformation rules. These rules would allow for expressing a given global constraint in terms of others, or would permit generating necessary conditions expressed as a conjunction of constraints for the feasibility of a given global constraint. Then transformation rules could be gradually incorporated into the catalogue of global constraints. This suggests the following research path.

RP 4. Investigate in which form transformation rules should be given: ideally, they should both allow for writing down programs that derive implied global constraints as well as be used in two directions.[21] Finally, come up with compilers that use these rules in order to automate the process of moving from a model to a description designed for efficient solution and/or specific constraint system target. These compilers should also deal with the problem of extracting global constraints from matrix models [37].

[20] This is usually done in order to get more pruning and/or to use less memory.

[21] From a set of disequality constraints one should be able to extract cliques of disequalities, and from an `alldifferent` constraint it should be possible to generate a clique of disequalities.

6.2 Constraint Visualisation

Constraint visualisation has been recognised as a key topic for debugging constraint programs [30]. One typical aspect of Constraint Programming is that, quite often, we initially have inefficient correct programs. This leads to the notion of *performance debugging* (i.e., try to enhance the performance of a constraint program that is already correct). Global constraints have been found useful in this context since:

- Programs that use global constraints usually involve much fewer constraints.
- Quite often, it turns out that global constraints are much more application oriented than primitive constraints.[22] For instance associating a Gantt chart to a `cumulative` constraint allows to have a first graphical interface that can be immediately understood.

Until now, specific visualisation tools have been independently developed for each global constraint. This was for instance the case in the DISCIPL project where visualisation tools were developed for the `cumulative`, the `cycle` and the `diffn` constraints [74]. However, in the context of the global constraint catalogue, this is clearly not a feasible approach since we have too many global constraints. Currently, the global constraint catalogue contains a generic graphical representation, which is systematically used. This representation displays the final digraph associated with a ground instance of a global constraint: according to the graph parameters used in the graph-based representation of the global constraint we give the values actually taken by these graph parameters and outline these graph parameters (for instance for **MIN_NCC** we outline the smallest connected component of the final digraph). This representation is automatically synthesised from the graph-based description.[23] Even if this constitutes a generic approach, this graph-based visualisation is far from being satisfactory. This can be observed from the fact that, for quite a lot of global constraints, the catalogue provides additional figures that represent the constraint in a more appropriate graphical form (see for instance the `cumulative`, the `cumulative_two_d`, the `nvalue` or the `orchard` constraints). This suggests the following line of research.

RP 5. From the graph-based representation of global constraints, investigate how to derive more appropriate graphical representations related to frequent graph classes. Moreover these graphical representations should cope with large data sets.

6.3 Constraint-oriented Heuristics

When it comes to the point of solving concrete problems, global constraints require to use a heuristic that explores the search space by gradually adding bactrackable

[22] Especially in the context of commercial solvers that have global constraints dedicated to certain application areas.

[23] For this purpose, we use the open source graph drawing software Graphviz available from AT&T (http://www.research.att.com/sw/tools/graphviz).

constraints[24] until all variables of the problem get fixed. Constraint systems usually offer three ways for expressing heuristics:

- *General purpose heuristics* mostly based on the domain of the variables (i.e., the number of values in the domain, the smallest or largest values, the difference between the second smallest value and the smallest value, the number of holes, ...) as well as on the number of constraints mentioning a given variable.
- *Basic primitives* for creating a choice point or for assigning a value to a variable. These primitives are provided for programming specific heuristics.
- *Predefined constraint-oriented heuristics* that are based on the internal state of a global constraint. For instance, in the context of a `cumulative` constraint, one may want to base a heuristics on the cumulated profile of compulsory parts[25] already built. But, in the context of an `alldifferent` or a `cycle` constraints, one may want to build a heuristics based on the perfect matching computed by the filtering algorithm in order to check feasibility.

Using heuristics that are related to the structure of a problem is of great importance and, in fact, quite common in Operations Research. But currently, in the context of Constraint Programming, the situation is difficult since:

- By definition, predefined constraint-oriented heuristics are ad hoc. This concretely means that it is almost sure that available predefined constraint-oriented heuristics will not fit the exact requirement of a user dealing with a specific problem. Unless he has access to the source code of the filtering algorithm he will not be able to take advantage of the information available inside the global constraint. But even if the source code is available, understanding all the subtleties of the internal implementation of a global constraint will usually be far too difficult.
- Since they typically are available in only one solver, predefined constraint-oriented heuristics hinder the compatibility between solvers. This concretely means that, even when two constraint systems offer the same set of constraints, a source to source translation of a model from a first system to a second system will be quite impossible when predefined constraint-oriented heuristics are used. It is also very unlikely that reformulation tools could take advantage of such predefined constraint-oriented heuristics in a systematic way.
- Finally, the declarative semantics of predefined constraint-oriented heuristics is usually not fully documented. This make such extensions quite dirty from a language perspective: hooks to the internal data structure have to be provided.

This suggests the following research path.

RP 6. Use the graph-based description of global constraints in order to come up with a clean and concise way to express graph-based heuristics. The key point would be to allow for expressing a graph property depicting what we want to select first (and not how to compute the things we want to select first).

[24] Usually an equality constraint between a variable and a constant.

[25] The *compulsory part* of a task corresponds to the intersection of all feasible instances of a task.

6.4 Constraint Seeker

The catalogue of global constraints contains more than 250 global constraints. Retrieving easily a global constraint in the catalogue or finding out that a global constraint does not exist may be quite complex:

- The user may not be aware of the name of the global constraint he is looking for. In fact, the name of a global constraint usually reflects a particular interpretation of the utilisation of that constraint, which makes it hard to come up with names that reflect all potential usage of a global constraint.
- The user may also not know in which order he has to provide the different arguments of a global constraint. This gets even worse as the number of arguments of a global constraint increases (e.g., the `group` constraint has 8 arguments).
- Some global constraints have an argument that corresponds to a collection of items where each item consists of several attributes. For instance, in the context of the `cumulative` constraint, the first argument is a collection of tasks, where each task has an `origin`, a `duration`, a `height` and an `end` attribute. Again it is unrealistic to expect that the user knows the name of the attributes and the order in which they are defined. Things get even worse when we consider the fact that some attributes may not need to be explicitly defined.
- An additional problem is that the user may look for a global constraint that is very close to an existing global constraint of the catalogue.

All these points suggest the following enhancement.

PE 3. Organise a site for interactive consultation of the catalogue of global constraints where global constraints can be searched (without knowing neither their name nor the syntax of their parameters) by providing ground instances corresponding to solutions and/or non-solutions.

7 Conclusion

In Operations Research it is a well-known fact that good bounds are a key element for efficient problem solving. In Operations Research as well as Constraint Programming, it is also known that necessary conditions (that can be checked efficiently) for the realisability of a problem are extremely important. Despite these facts, bounds and necessary conditions are currently used in an ad hoc way in both Operations Research and Constraint Programming: given a problem or a global constraint, one investigates how to find bounds and necessary conditions for this problem or that constraint.

In this article we have proposed the first systematic approach where a constraint is described in terms of graph properties that are used for searching in a data base of bounds and graph invariants the relevant formula and components for synthesising a filtering algorithm. Two points are critical in this approach:

- In the short term, the approach relies mostly on the availability of a large data base of bounds and invariants for a significant number of graph classes that arise in practice. Even if building such a huge data base requires a significant amount of work beyond the capability of one single person, this kind of work is typically

incremental and could be done in parallel. Such a data base could surely benefit the Operation Research[26] as well as the Constraint Programming communities.
– In the long term, one would also like to address the research path *RP 3* corresponding to the design of a *compositional language* for expressing most graph algorithms and the creation of an *abstract machine* for evaluating expressions of such a language. Again this would benefit both the Operations Research and the Constraint Programming communities.

Finally, we also hope that we have raised the interest of using global constraints in a systematic way for dealing with such topics as *constraint reformulation*, *constraint visualisation*, *constraint-oriented heuristics*, and *constraint seeker*.

Appendix

A Omitted Proofs of Propositions 3 and 4

Proof of Proposition 3 $\mu_l(\overleftrightarrow{G}_{rem})$ is the maximum possible number of new connected components that could be present in G_f in addition to the current number of connected components involving at least one T-arc. If one arc joins two connected components in $cc_{[|E_T|\geq 1]}(\overrightarrow{G}(X_T, E_T))$ then the number of connected components of the final digraph will be strictly less than $|cc_{[|E_T|\geq 1]}(\overrightarrow{G}(X_T, E_T))| + \mu_l(\overleftrightarrow{G}_{rem})$, a contradiction. Therefore Item 1 holds.

Similarly, U-arcs corresponding to edges in $\overleftrightarrow{G}_{rem}$ that do not belong to at least one maximum l-matching of $\overleftrightarrow{G}_{rem}$ cannot belong to the final digraph because their presence would decrease the number of new connected components. Therefore Item 2 holds.

Furthermore, no U-arc that joins a connected component in $cc_{[|E_T|\geq 1]}(\overrightarrow{G}(X_T, E_T))$ with a vertex saturated in all maximum matchings of $\overleftrightarrow{G}_{rem}$ can be turned into a T-arc. Therefore Item 3 holds.

On the contrary, U-vertices saturated in all maximum matchings of $\overleftrightarrow{G}_{rem}$ and U-arcs belonging to all such maximum matchings must necessarily belong to G_f. Items 4 and 5 hold. □

Proof of Proposition 4 A subpart of each connected component in $cc_{[|X_T|\geq 1]}(\overrightarrow{G}(X_{TU}, E_{TU}))$ will necessarily belong to a distinct connected component of G_f. By definition, no additional connected component of $\overrightarrow{G}(X_{TU}, E_{TU}))$ can belong to G_f. As a consequence Item 1 holds. Moreover, existing connected components containing a T-arc should not be split. Therefore some U-vertices and U-arcs should necessarily belong to G_f. Items 2 and 3 hold. □

B Omitted Proofs of Table 3

Property 1 $\underline{\mathbf{NARC}}_{PATH} \geq |E_T| + \sum_{i\in cc(\overrightarrow{G}(X_{T,\neg T}, E_U))} \lfloor \frac{|vertex(i)|+1}{2} \rfloor$

[26] In fact P. Hansen is calling for such a development in one recent article [42].

Proof Since no T-vertex is isolated in $\overrightarrow{G}(X_{TU}, E_{TU})$, any T-vertex in $\overrightarrow{G}(X_{T,\neg T}, E_U)$ is the extremity of one or two arcs in E_U. Therefore, the corresponding undirected graph $\overleftrightarrow{G}(X_{T,\neg T}, E_U)$ is a set of mutually disjoint elementary chains. For a given elementary chain i (that is, a connected component), the cardinality of a minimum edge cover is $\lfloor \frac{|vertex(i)|+1}{2} \rfloor$. By Table 1 the property holds. □

Property 2 The lower bound provided by Property 1 is sharp.

Proof When $G_{\mathcal{R}}$ is generated by using PATH, $\overleftrightarrow{G}(X_{TU}, E_{TU})$ is a set of mutually disjoint elementary chains. $\sum_{i \in cc(\overrightarrow{G}(X_{T,\neg T}, E_U))} \lfloor \frac{|vertex(i)|+1}{2} \rfloor = |X_{T,\neg T}| - \mu(\overleftrightarrow{G}(X_{T,\neg T}, E_U))$. By Table 1 the property holds. □

Property 3 $\underline{\mathbf{NVERTEX}}_{PATH} \geq |X_T| + \sum_{i \in cc(\overrightarrow{G}(X_{T,\neg T,\neg T} \cup X_{U,U,\neg U}, E_{U,T}))} \lfloor \frac{|vertex_{[|X_T|=1]}(i)|+1}{2} \rfloor$

Proof All T-vertices will belong to the final digraph G_f. Therefore, $\underline{\mathbf{NVERTEX}}_{PATH} \geq |X_T|$. Some of them are linked to U-vertices only. Since there is no loop and no isolated vertices in G_f, a certain number of these U-vertices must belong to G_f. Consider the connected components that are exclusively composed by T-vertices not linked to any other T-vertex, the U-vertices that are linked to them, and all the U-arcs incident on a T-vertex. Each of these connected components, which we denote by i, is such that by definition neither two T-vertices nor two U-vertices are consecutive. Thus, for each i, the minimum number of U-vertices required to prevent all T-vertices from being isolated is exactly $\lfloor \frac{|vertex_{[|X_T|=1]}(i)|+1}{2} \rfloor$. The property holds. □

Property 4 The lower bound provided by Property 3 is sharp.

Proof For the arc-generator $PATH$, the quantity $\sum_{i \in cc(\overrightarrow{G}(X_{T,\neg T,\neg T} \cup X_{U,U,\neg U}, E_U))} \lfloor \frac{|vertex_{[|X_T|=1]}(i)|+1}{2} \rfloor$ is equal to the cardinality of the minimum hitting set involved in the general bound of Table 1. It is possible to derive from $\overrightarrow{G}(X_{TU}, E_{TU})$ a final digraph where $\mathbf{NVERTEX}_{PATH} = |X_T| + \sum_{i \in cc(\overrightarrow{G}(X_{T,\neg T,\neg T} \cup X_{U,U,\neg U}, E_U))} \lfloor \frac{|vertex_{[|X_T|=1]}(i)|+1}{2} \rfloor$ by turning exactly one U-arc to a T-arc for each isolated T-vertex in each connected component of $cc(\overrightarrow{G}(X_{T,\neg T,\neg T} \cup X_{U,U,\neg U}, E_U))$. The property holds. □

Property 5 $\overline{\mathbf{NCC}}_{PATH} \leq |cc_{[|E_T|\geq 1]}(\overrightarrow{G}(X_T, E_T))| + \sum_{i \in cc(\overleftrightarrow{G}_{rem})} \lfloor \frac{|vertex(i)|}{2} \rfloor$

Proof $\overleftrightarrow{G}_{rem}$ is a set of elementary chains disjoint from one another. For a subgraph consisting of one chain, the cardinality of a maximum matching is equal to the floor of the number of vertices divided by two. $\mu_l(\overleftrightarrow{G}_{rem}) = \sum_{i \in cc(\overleftrightarrow{G}_{rem})} \lfloor \frac{|vertex(i)|}{2} \rfloor$. From the general bound of Table 1 the property holds. □

Property 6 The lower bound provided by Property 5 is sharp.

Proof By proof of Property 5 and by Table 1. □

Property 7 $\mathbf{NVERTEX}_{PATH} = \mathbf{NSCC}_{PATH}$

Proof Given an arc (u, v) in $\overrightarrow{G}(X_{TU}, E_{TU})$, by definition of the *PATH* arc-generator, (v, u) does not belong to $\overrightarrow{G}(X_{TU}, E_{TU})$. Therefore the set of strongly connected components is the set of vertices. □

Property 8 $\mathbf{NCC}_{PATH} = \mathbf{NSINK}_{PATH}$

Proof By definition of the *PATH* arc-generator all connected components are elementary paths. □

References

1. Aggoun, A., & Beldiceanu, N. (1993). Extending CHIP in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling*, *17*(7), 57–73.
2. Baptiste, P., & Le Pape, C. (2000). Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. *Constraints*, *5*(1/2), 119–139.
3. Baptiste, P., & Demassey, S. (2004). Tight LP bounds for resource constrained project scheduling. *OR-Spektrum*, *26*, 251–262.
4. Beldiceanu, N. (1990). *An example of introduction of global constraints in CHIP: Application to block theory problems*. Technical report TR-LP-49, ECRC, Munich.
5. Beldiceanu, N. (2000). Global constraints as graph properties on a structured network of elementary constraints of the same type. In R. Dechter (Ed.), *Principles and practice of Constraint Programming (CP'2000)*, volume 1894 of *LNCS* (pp. 52–66). Berlin Heidelberg New York: Springer. Preprint available as SICS Tech Report T2000-01.
6. Beldiceanu, N., & Contejean, E. (1994). Introducing global constraints in CHIP. *Mathematical and Computer Modelling*, *20*(12), 97–123.
7. Beldiceanu, N., & Petit, T. (2004). Cost evaluation of soft global constraints. In J.-C. Régin & M. Rueher (Eds.), *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 3011 of *LNCS* (pp. 80–95). Berlin Heidelberg New York: Springer.
8. Beldiceanu, N., Carlsson, M., & Petit, T. (2004). Deriving filtering algorithms from constraint checkers. In M. Wallace (Ed.), *Principles and Practice of Constraint Programming (CP'2004)*, volume 3258 of *LNCS* (pp 107–122). Berlin Heidelberg New York: Springer. Preprint available as SICS Tech Report T2004-08.
9. Beldiceanu, N., Katriel, I., & Thiel, S. (2004). Filtering algorithms for the *same* constraint. In J.-C. Régin & M. Rueher (Eds.), *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR 2004)*, volume 3011 of *LNCS* (pp 65–79). Berlin Heidelberg New York: Springer.
10. Beldiceanu, N., Carlsson, M., Debruyne, R., & Petit, T. (2005). Reformulation of global constraints based on constraint checkers. *Constraints*, *10*(3), 339–362.
11. Beldiceanu, N., Carlsson, M., & Rampon, J.-X. (2005). *Global constraint catalog.* Technical Report T2005-06, Swedish Institute of Computer Science, Kista.
12. Beldiceanu, N., Carlsson, M., Rampon, J.-X., & Truchet, C. (2005). Graph invariants as necessary conditions for global constraints. In P. van Beek (Ed.), *Principles and Practice of Constraint Programming (CP'2005)*, volume 3709 of *LNCS* (pp 92–106). Berlin Heidelberg New York: Springer. Preprint available as SICS Tech Report T2005-07.
13. Beldiceanu, N., Flener, P., & Lorca, X. (May 2005) The *tree* constraint. In R. Barták & M. Milano (Eds.), *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'05)*, volume 3524 of *LNCS* (pp. 64–78). Prague, Czech Republic. Berlin Heidelberg New York: Springer.
14. Beldiceanu, N., Katriel, I., & Thiel, S. (2005). GCC-like restrictions on the *same* constraint. In B. Faltings, A. Petcu, F. Fages, & F. Rossi (Eds.), *Springer LNAI volume based on the 2004 edition*

of the ERCIM/Colognet workshop on Constraint Solving and Constraint Logic Programming (CSCLP04), volume 3419 of *LNAI* (pp. 1–11). Berlin Heidelberg New York: Springer.
15. Beldiceanu, N., Petit, T., & Rochart, G. (2005). Bounds of graph characteristics. In P. van Beek (Ed.), *Principles and Practice of Constraint Programming (CP'2005)*, volume 3709 of *LNCS* (pp. 742–746). Berlin Heidelberg New York: Springer.
16. Beldiceanu, N., Petit, T., & Rochart, G. (2005). *Bounds of graph characteristics.* Technical Report 05/2/INFO. Ecole des Mines, Paris.
17. Beldiceanu, N., Carlsson, M., Demassey, S., & Petit, T. (2006). *Graph properties based filtering.* Technical Report T2006-10, Swedish Institute of Computer Science, Kista, Sweden.
18. Beldiceanu, N., Katriel, I., & Lorca, X. (2006). Undirected forest constraints. In CP-AI-OR'06, volume 3990 of *LNCS*. Berlin Heidelberg New York: Springer.
19. Berge, C. (1970). *Graphes*. Dunod (in French).
20. Bessière, C., & Van Hentenryck, P. (2003). To be or not to be ... a global constraint. In F. Rossi (Ed.), *Principles and Practice of Constraint Programming (CP'2003)*, volume 2833 of *LNCS* (pp. 789–794). Berlin Heidelberg New York: Springer.
21. Bessière, C., Coletta, R., & Petit, T. (2005). Acquiring parameters of implied global constraints. In P. van Beek (Ed.), *Principles and Practice of Constraint Programming (CP'2005)*, volume 3709 of *LNCS* (pp. 747–751). Berlin Heidelberg New York: Springer.
22. Bessière, C., Hebrard, E., Hnich, B., Kızıltan, Z., & Walsh, T. (2005). *Among*, *common* and *disjoint* constraints. In *CSCLP 2005*, pp. 223–235. Uppsala, Sweden. Berlin Heidelberg New York: Springer.
23. Bessière, C., Hebrard, E., Hnich, B., Kızıltan, Z., & Walsh, T. (May 2005). Filtering algorithms for the *nvalue* constraint. In R. Barták & M. Milano (Eds.), *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'05)*, volume 3524 of *LNCS*, (pp. 79–93). Prague, Czech Republic. Berlin Heidelberg New York: Springer.
24. Bleuzen-Guernalec, N., & Colmerauer, A. (1997). Narrowing a block of sortings in quadratic time. In G. Smolka (Ed.), *Principles and Practice of Constraint Programming (CP'97)*, volume 1330 of *LNCS* (pp. 2–16). Berlin Heidelberg New York: Springer.
25. Card, S. K., Mackinlay, J. D., & Shneiderman, B. (1999). *Readings in information visualization using vision to think*. Morgan Kaufmann.
26. Carlsson, M., & Beldiceanu, N. (2002). *Arc-consistency for a chain of lexicographic ordering constraints*. Technical Report T2002-18, Swedish Institute of Computer Science, Kista, Sweden.
27. Caseau, Y., & Laburthe, F. (1996). Cumulative scheduling with task intervals. In *Joint International Conference and Symposium on Logic Programming (JICSLP'96)*. MIT Press.
28. Colton, S., & Miguel, I. (2001). Constraint generation via automated theory formation. In T. Walsh (Ed.), *Principles and Practice of Constraint Programming (CP'2001)*, volume 2239 of *LNCS* (pp. 575–579). Berlin Heidelberg New York: Springer.
29. Costa, M.-C. (1994). Persistency in maximum cardinality bipartite matchings. *Operation Research Letters*, *15*, 143–149.
30. Deransart, P., Hermenegildo, M. V., & Maluszynski, J. (Eds.) (2000). *Analysis and Visualization Tools for Constraint Programming, Constraint Debugging (DiSCiPl project)*, volume 1870 of *LNCS*. Berlin Heidelberg New York: Springer.
31. Dincbas, M., Van Hentenryck, P., Simonis, H., Graf, T., Aggoun, A., & Berthier, F. (1988). The constraint logic programming language CHIP. In *International Conference on Fifth Generation Computer Systems (FGCS'88)* (pp. 693–702). Tokyo, Japan: ICOT.
32. Dooms, G., Deville, Y., & Dupont, P. (2005). CP(Graph): Introducing a graph computation domain in constraint. In P. van Beek (Ed.), *Principles and Practice of Constraint Programming (CP'2005)*, volume 3709 of *LNCS* (pp. 211–225). Berlin Heidelberg New York: Springer.
33. Dudeney, H. E. (1919). *The Canterbury puzzles*. Nelson.
34. Elbassioni, K. M., Katriel, I., Kutz, M., & Mahajan, M. (2005). Simultaneous matchings. In *Algorithms and Computation, 16th International Symposium (ISAAC 2005)*, volume 3827 of *LNCS* (pp. 106–115). Berlin Heidelberg New York: Springer.
35. Erschler, J., & Lopez, P. (June 1990). Energy-based approach for task scheduling under time and resources constraints. In *2nd International Workshop on Project Management and Scheduling* (pp. 115–121). France: Compiégne.
36. Euler, L. (1759). Solution d'une question curieuse qui ne parait soumise à aucune analyse. *Mm. Acad. Sci. Berlin*, *15*, 310–337.

37. Flener, P., Frisch, A. M., Hnich, B., Kızıltan, Z., Miguel, I., & Walsh, T. (2002). Matrix modelling: Exploiting common patterns in constraint programming. In A. M. Frisch (Ed.), *Proceedings of the International Workshop on Reformulating CSPs, held at CP'02*.
38. Freuder, E. C. (1997). In pursuit of the holy grail. *Constraints*, *2*(1), 57–61.
39. Frisch, A. M., Jefferson, C., Martinez-Hernandez, B., & Miguel, I. (2005). The rules of constraint modelling. In *19th International Joint Conference on Artificial Intelligence (IJCAI-05)* (pp. 109–116).
40. Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability. A guide to the theory of NP-completeness*. Freeman.
41. Guéret, C., Jussien, N., Boizumault, P., & Prins, C. (August 1995). Building university timetables using constraint logic programming. In *ICPTAT'95: First International Conference on the Practice and Theory of Automated Time Tabling* (pp. 393–408). Edinburgh, United Kingdom.
42. Hansen, P. (2005). How far is, should and could be conjecture-making in graph theory an automated process? In S. Fajtlowicz, P. W. Fowler, P. Hansen, M. F. Janowitz, & F. S. Roberts (Eds.), *Graphs and Discovery*, volume 69 of *DIMACS: Series in discrete mathematics and theoretical computer science* (pp. 189–230). American Mathematical Society, DIMACS.
43. Henriksen, J. G., Jensen, J. L., Jørgensen, M. E., Klarlund, N., Paige, R., Rauhe, T., & Sandholm, A. (May 1995). Mona: Monadic second-order logic in practice. In E. Brinksma, R. Cleaveland, K. G. Larsen, T. Margaria, & B. Steffen (Eds.), *Tools and Algorithms for Construction and Analysis of Systems, First International Workshop, TACAS'95, Aarhus, Denmark*, volume 1019 of *LNCS* (pp. 89–110). Berlin Heidelberg New York: Springer.
44. Hnich, B. (January 2003). Function variables for constraint programming. Ph.D. Thesis, Department of Information Science, Uppsala University.
45. Hooker, J. (October 2005). Past and future of CP. Panel, "The Past and Future of Constraint Programming" at the Eleventh International Conference on Principles and Practice of Constraint Programming. Available at http://www.iiia.csic.es/cp2005/CP05-panel-hooker.pdf.
46. Hooker, J. N., & Yan, H. (2002). A relaxation for the *cumulative* constraint. In P. Van Hentenryck (Ed.), *Principles and Practice of Constraint Programming (CP'2002)*, volume 2470 of *LNCS* (pp. 686–690). Berlin Heidelberg New York: Springer.
47. Katriel, I., & Thiel, S. (2003). Fast bound consistency for the *global cardinality* constraint. In F. Rossi (Ed.), *Principles and Practice of Constraint Programming (CP'2003)*, volume 2833 of *LNCS* (pp. 437–451). Berlin Heidelberg New York: Springer.
48. Katriel, I., & Thiel, S. (2005). Complete bound consistency for the global cardinality constraint. *Constraints*, *10*(3), 191–217.
49. Kirkman, T. P. (1847). On a problem in combinatorics. *Cambridge and Dublin Math. J.*, *2*, 191–204.
50. Lahrichi, A. (February 1982). Scheduling: The notions of hump, compulsory parts and their use in cumulative poblems. *C.R. Acad. Sci., Paris* 294: 209–211.
51. Laurière, J.-L. (1996). *Constraint propagation or automatic programming?* Technical Report Laforia, number 19, Institut Blaise Pascal (in French).
52. Leconte, M. (1996). A bounds-based reduction scheme for constraints of difference. In *CP'96, Second International Workshop on Constraint-based Reasoning* (pp. 19–28). Key West, FL.
53. Lopez-Ortiz, A., Quimper, C.-G., Tromp, J., & van Beek, P. (2003). A fast and simple algorithm for bounds consistency of the *alldifferent* constraint. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'2003)* (pp. 245–250).
54. Lucas, E. (1882). *Récréations Mathématiques*, volume 1-2. Gauthier-Villars.
55. Mehlhorn, K. (2000). Constraint programming and graph algorithms. In U. Montanari, J. D. P. Rolim, & E. Welzl (Eds.), *27th International Colloquium on Automata, Languages and Programming (ICALP'2000)*, volume 1853 of *LNCS* (pp. 571–575). Berlin Heidelberg New York: Springer.
56. Mehlhorn, K., & Thiel, S. (2000). Faster algorithms for bound-consistency of the *sortedness* and the *alldifferent* constraint. In *Principles and Practice of Constraint Programming (CP'2000)*, volume 1894 of *LNCS* (pp. 306–319). Berlin Heidelberg New York: Springer.
57. Milano, M., Ottoson, G., Refalo, P., & Thorsteinsson, E. (2002). The role of integer programming techniques in constraint programming's global constraints. *INFORMS Journal on Computing*, *14*(4), 387–402.
58. Pachet, F., & Roy, P. (1999). Automatic generation of music programs. In *Principles and Practice of Constraint Programming (CP'99)*, volume 1713 of *LNCS* (pp. 331–345). Berlin Heidelberg New York: Springer.

59. Pesant, G. (2004). A regular language membership constraint for finite sequences of variables. In M. Wallace (Ed.), *Principles and Practice of Constraint Programming (CP'2004)*, volume 3258 of *LNCS* (pp. 482–495). Berlin Heidelberg New York: Springer.
60. Petit, T., Régin, J.-C., & Bessière, C. (2001). Specific filtering algorithms for over-constrained problems. In T. Walsh (Ed.), *Principles and Practice of Constraint Programming (CP'2001)*, volume 2239 of *LNCS* (pp. 451–463). Berlin Heidelberg New York: Springer.
61. Pitrat, J. (September 2001). MALICE, notre collègue. In *Colloque Métaconnaissance de Berder* (pp. 4–19). In French.
62. Puget, J.-F. (November 1994). A C++ implementation of CLP. In *Second Singapore International Conference on Intelligent Systems (SPICIS)*, pp. 256–261. Singapore.
63. Puget, J.-F. (1998). A fast algorithm for the bound consistency of *alldiff* constraints. In *15th National Conference on Artificial Intelligence (AAAI-98)* (pp. 359–366). AAAI Press.
64. Puget, J.-F. (2004). Constraint programming next challenge: Simplicity of use. In M. Wallace (Ed.), *Principles and Practice of Constraint Programming (CP'2004)*, volume 3258 of *LNCS* (pp. 5–8). Berlin Heidelberg New York: Springer.
65. Puget, J.-F. (2005). Automatic detection of variable and value symmetries. In P. van Beek (Ed.), *Principles and Practice of Constraint Programming (CP'2005)*, volume 3709 of *LNCS* (pp. 475–489). Berlin Heidelberg New York: Springer.
66. Quimper, C.-G., van Beek, P., López-Ortiz, A., Golynski, A., & Sadjad, S. B. (2003). An efficient bounds consistency algorithm for the *global cardinality* constraint. In F. Rossi (Ed.), *Principles and Practice of Constraint Programming (CP'2003)*, volume 2833 of *LNCS* (pp. 600–614). Berlin Heidelberg New York: Springer.
67. Quimper, C.-G., López-Ortiz, A., van Beek, P., & Golynski, A. (2004). Improved algorithms for the *global cardinality* constraint. In M. Wallace (Ed.), *Principles and Practice of Constraint Programming (CP'2004)*, volume 3258 of *LNCS* (pp. 542–556). Berlin Heidelberg New York: Springer.
68. Régin, J.-C. (1994). A filtering algorithm for constraints of difference in CSP. In *12th National Conference on Artificial Intelligence (AAAI-94)*, pp. 362–367.
69. Régin, J.-C. (1996). Generalized arc consistency for *global cardinality* constraint. In *14th National Conference on Artificial Intelligence (AAAI-96)*, pp. 209–215.
70. Régin, J.-C. (1999). The symmetric *alldiff* constraint. In *16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pp. 420–425.
71. Régin, J.-C., & Gomes, C. (2004). The *cardinality matrix* constraint. In M. Wallace (Ed.), *Principles and Practice of Constraint Programming (CP'2004)*, volume 3258 of *LNCS* (pp. 572–587). Berlin Heidelberg New York: Springer.
72. Régin, J.-C., & Rueher, M. (1999). A global constraint combining a *sum* constraint and binary inequalities. In *IJCAI-99 Workshop on Non Binary Constraints*.
73. Rochart, G., & Jussien, N. (2003). *Explanations for Global Constraints: Instrumenting the Stretch Constraint.* Technical report 03-01-INFO, École des Mines de Nantes.
74. Simonis, H., Aggoun, A., Beldiceanu, N., & Bourreau, E. (2000). Complex constraint abstraction: Global constraint visualization. In P. Deransart, M. V. Hermenegildo, & J. Małuszyński (Eds.), *Analysis and Vizualisation Tools for Constraint Programming*, volume 1870 of *LNCS* (pp. 299–317). Berlin Heidelberg New York: Springer.
75. Turán, P. (1941). On an extremal problem in graph theory. *Matematikai es' Fizikai Lapok*, *48*, 436–452. In Hungarian.
76. Van Hentenryck, P. (1997). Constraint programming for combinatorial search problems. *Constraints*, *2*(1), 99–101.
77. Van Hentenryck, P. (1999). *The OPL optimization programming language*. MIT Press.
78. Van Hentenryck, P., & Michel, L. (2005). *Constraint-based local search*. MIT Press.
79. van Hoeve, W.-J. (2004). A hyper-arc consistency algorithm for the *soft alldifferent* constraint. In M. Wallace (Ed.), *Principles and Practice of Constraint Programming (CP'2004)*, volume 3258 of *LNCS* (pp. 679–689). Berlin Heidelberg New York: Springer.
80. van Hoeve, W.-J., Pesant, G., & Rousseau, L.-M. (September 2004). On global warming (softening global constraints). In *Workshop on soft constraints*. Toronto, Canada.
81. Voigt, K. (1988). *Incorporating global constraints.* Technical Report LCSR-TR-114, Laboratory for Computer Science Research, Hill Center for the Mathematical Sciences, Busch Campus, Rutgers University, New Brunswick, NJ.