On Global Warming
(Softening Global Constraints)

Willem Jan van Hoeve[1], Gilles Pesant[2,3] and Louis-Martin Rousseau[2,3]

[1] CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
W.J.van.Hoeve@cwi.nl
[2] École Polytechnique de Montréal, Montreal, Canada
[3] Centre for Research on Transportation (CRT),
Université de Montréal, C.P. 6128, succ. Centre-ville, Montreal, H3C 3J7, Canada
{pesant,louism}@crt.umontreal.ca

Abstract. We describe soft versions of the global cardinality constraint and the regular constraint, with efficient filtering algorithms maintaining domain consistency. For both constraints, the softening is achieved by augmenting the underlying graph. The softened constraints can be used to extend the meta-constraint framework for over-constrained problems proposed by Petit, Régin and Bessière.

1 Introduction

Constraint Programming (CP) is a widely used and efficient technique to solve combinatorial optimization problems. However in practice many problems are over-constrained (intrinsically or from being badly stated). Several frameworks have been proposed to handle over-constrained problems, mostly by introducing *soft constraints* that are allowed to be (partially) violated. The most well-known framework is the Partial Constraint Satisfaction Problem framework (PCSP [8]), which includes the Max-CSP framework that tries to maximize the number of satisfied constraints. Since in this framework all constraints are either violated or satisfied, this objective is equivalent to minimizing the number of violations. It has been extended to the *Weighted-CSP* [10, 11], associating a degree of violation (not just a boolean value) to each constraint and minimizing the sum of all weighted violations. The *Possibilistic-CSP* [18] associates a preference to each constraint (a real value between 0 and 1) representing its importance. The objective of the framework is the hierarchical satisfaction of the most important constraints, that is, the minimization of the highest preference level for a violated constraint. The *Fuzzy-CSP* [6, 7] is somewhat similar to the Possibilistic-CSP but here a preference is associated to each tuple of each constraint. A preference value of 0 means the constraint is highly violated and 1 stands for satisfaction. The objective is the maximization of the smallest preference value induced by a variable assignment. The last two frameworks are different from the previous ones since the aggregation operator is a *min*/*max* function instead of addition. Max-CSPs are typically encoded and solved with one of two generic paradigms: valued-CSPs [19] and semi-rings [5].

Another approach to model and solve over-constrained problems involves *Meta-Constraints* [13]. The idea behind this technique is to introduce a set of domain variables Z that capture the violation cost of each soft constraint. By correctly constraining these variables it is possible to replicate the previous frameworks and even to extend the modeling capability to capture other types of violation measures. Namely the authors argue that although the Max-CSP family of frameworks is quite efficient to capture local violation measures it is not as adequate to model violation costs involving several soft constraints simultaneously. By defining (possibly global) constraints on Z such a behaviour can be easily achieved. The authors propose to replace each soft constraint S_i present in a model by a disjunctive constraint specifying that either $z_i = 0$ and the constraint S_i is hard or $z_i > 0$ and S_i is violated. This technique allows the resolution of over-constrained problem within traditional CP solvers.

Comparatively few efforts have been invested in developing soft versions of common global constraints [14, 4, 9]. Global constraints are often key elements in successfully modeling real applications and being able to easily and effectively soften such constraints would yield a significant improvement in flexibility. In this paper we study two global constraints: the widely known global cardinality constraint (`gcc`) [15] and the new `regular` [12] constraint. For each of these we propose new violation measures and provide the corresponding filtering algorithms to achieve domain consistency. All the constraint softening is achieved by enriching the underlying graph representation with additional arcs that represent possible relaxations of the constraint. Violation costs are then associated to these new arcs and known graph algorithms are used to achieve domain consistency.

The two constraints studied in this paper are useful to model and solve personnel rostering problems (PRP). The PRP objective is typically to distribute a set of working shifts (or days off) to a set of employees every day over a planning horizon (a set of days). The `gcc` is a perfect tool to restrict the number of work shifts of each type (Day, Evening, and Night for instance) performed by each employee. Other types of constraints involve sequences of shifts over time, typically forbidding non ergonomic schedules. The `regular` constraint has the expressive power necessary to cope with the complex regulations found in many organizations. Since most real rostering applications are over-constrained (due to lack of personnel or over-optimistic scheduling objectives), soft versions of the `gcc` and `regular` constraints promise to significantly improve our modelling flexibility.

This paper is organized as follows. Section 2 presents background information on Constraint Programming and the softening of (global) constraints. In Section 3 and 4 we describe the softening of the `gcc` and the `regular` constraint respectively. Both constraints are softened with respect to two violation measures. We also provide corresponding filtering algorithms achieving domain consistency. Section 5 discusses the aggregation of several soft (global) constraints by meta-constraints. Finally, a conclusion is given in Section 6.

2 Background

We assume familiarity with the basic concepts of constraint programming. For a thorough explanation of constraint programming, see [2].

A constraint satisfaction problem (CSP) consists of a finite set of variables $X = \{x_1, \dots, x_n\}$ with finite domains $\mathcal{D} = \{D_1, \dots, D_n\}$ such that $x_i \in D_i$ for all i, together with a finite set of constraints $\mathcal{C}$, each on a subset of X. A constraint $C \in \mathcal{C}$ is defined as a subset of the Cartesian product of the domains of the variables that are in C. A tuple $(d_1, \dots, d_n) \in D_1 \times \cdots \times D_n$ is a solution to a CSP if for every constraint $C \in \mathcal{C}$ on the variables $x_{i_1}, \dots, x_{i_k}$ we have $(d_{i_1}, \dots, d_{i_k}) \in C$. A constraint optimization problem (COP) is a CSP together with an objective function to be optimized. A solution to a COP is a solution to the corresponding CSP that has an optimal objective function value.

Definition 1 (Domain consistency). *A constraint C on the variables $x_1, \dots, x_k$ is called domain consistent if for each variable x_i and value $d_i \in D_i$, there exist values $d_1, \dots, d_{i-1}, d_{i+1}, \dots, d_k$ in $D_1, \dots, D_{i-1}, D_{i+1}, \dots, D_k$, such that $(d_1, \dots, d_k) \in C$.*

Our definition of domain consistency corresponds to hyper-arc consistency or generalized arc consistency, which are also often used in the literature.

Definition 2 (Consistent CSP). *A CSP is domain consistent if all its constraints are domain consistent. A CSP is inconsistent if it has no solution. Similarly for a COP.*

When a CSP is inconsistent it is also said to be over-constrained. It is then natural to identify soft constraints, that are allowed to be violated, and minimize the total violation according to some criteria. For each soft constraint C, we introduce a function that measures the violation, and has the following form:

$$\text{violation}_C : D_1 \times \cdots \times D_n \to \mathbb{N}.$$

This approach has been introduced in [14] and was developed further in [4]. There may be several natural ways to evaluate the degree to which a global constraint is violated and these are not equivalent usually. A standard measure is the variable-based cost:

Definition 3 (Variable-based cost). *Given a constraint C on the variables $x_1, \dots, x_k$ and an instantiation $d_1, \dots, d_k$ with $d_i \in D_i$, the variable-based cost of violation of C is the minimum number of variables that need to change their value in order to satisfy the constraint.*

Alternative measures exist for specific constraints. For example, if a constraint is expressible as a conjunction of binary constraints, the cost may be defined as the number of these binary constraints that are violated. For the soft `gcc` and the soft `regular` constraint, we will introduce new violation measures, that are likely to be more effective in practical applications.

3 Soft Global Cardinality Constraint

A global cardinality constraint (gcc) on a set of variables specifies the minimum and maximum number of times each value in the union of their domains should be assigned to these variables. Régin developed a domain consistency algorithm for the gcc, making use of network flows [15]. A variant of the gcc is the cost-gcc, which can be seen as a weighted version of the gcc [16, 17]. For the cost-gcc a weight is assigned to each variable-value assignment and the goal is to satisfy the gcc with minimum total cost.

Throughout this section, we will use the following notation (unless specified otherwise). Let X denote a set of variables $\{x_1, \dots, x_n\}$ with respective finite domains $D_1, \dots, D_n$. We define $D_X = \cup_{i \in \{1,\dots,n\}} D_i$ and we assume a fixed but arbitrary ordering on D_X. For $d \in D_X$, let $l_d, u_d \in \mathbb{N}$, with $l_d \leq u_d$. Finally, let z be a variable with finite domain D_z, representing the cost of violation of the gcc.

Definition 4 (Global cardinality constraint).

$$\texttt{gcc}(X, l, u) = \{(d_1, \dots, d_n) \mid d_i \in D_i, l_d \leq |\{d_i \mid d_i = d\}| \leq u_d \ \forall\ d \in D_X\}.$$

We first give a generic definition for a soft version of the gcc.

Definition 5 (Soft global cardinality constraint).

$$\begin{aligned}\texttt{soft\_gcc}[\star](X, l, u, z) = \{(d_1, \dots, d_n, \tilde{d}) \mid d_i \in D_i, \tilde{d} \in D_z,& \\ \text{violation}_{\texttt{soft\_gcc}[\star]}(d_1, \dots, d_n) \leq \tilde{d}\},&\end{aligned}$$

where $\star$ defines a violation measure for the gcc.

In order to define measures of violation for the gcc, it is convenient to introduce the following functions.

Definition 6 (Overflow, underflow). *Given* $\texttt{gcc}(X, l, u)$*, define for all* $d \in D_X$

$$\text{overflow}(X, d) = \begin{cases} |\{x_i \mid x_i = d\}| - u_d & \text{if } |\{x_i \mid x_i = d\}| \geq u_d, \\ 0 & \text{otherwise,} \end{cases}$$

$$\text{underflow}(X, d) = \begin{cases} l_d - |\{x_i \mid x_i = d\}| & \text{if } |\{x_i \mid x_i = d\}| \leq l_d, \\ 0 & \text{otherwise.} \end{cases}$$

Let $\text{violation}_{\texttt{soft\_gcc}[\text{var}]}$ denote the variable-based cost of violation (see Definition 3) of the gcc. The next lemma expresses $\text{violation}_{\texttt{soft\_gcc}[\text{var}]}$ in terms of the above functions.

Lemma 1. *Given* $\texttt{gcc}(X, l, u)$,

$$\text{violation}_{\texttt{soft\_gcc}[\text{var}]}(X) = \max\left(\sum_{d \in D_X} \text{overflow}(X, d), \sum_{d \in D_X} \text{underflow}(X, d)\right)$$

provided that

$$\sum_{d \in D_X} l_d \leq |X| \leq \sum_{d \in D_X} u_d. \qquad (1)$$

Proof. The variable-based cost of violation corresponds to the minimal number of re-assignments of variables until both $\sum_{d \in D_X} \text{overflow}(X, d) = 0$ and $\sum_{d \in D_X} \text{underflow}(X, d) = 0$.

Assume $\sum_{d \in D_X} \text{overflow}(X, d) \geq \sum_{d \in D_X} \text{underflow}(X, d)$. Variables assigned to values $d' \in D_X$ with $\text{overflow}(X, d') > 0$ can be assigned to values $d'' \in D_X$ with $\text{underflow}(X, d'') > 0$, until $\sum_{d \in D_X} \text{underflow}(X, d) = 0$. In order to achieve $\sum_{d \in D_X} \text{overflow}(X, d) = 0$, we still need to re-assign the other variables assigned to values $d' \in D_X$ with $\text{overflow}(X, d') > 0$. Hence, in total we need to re-assign exactly $\sum_{d \in D_X} \text{overflow}(X, d)$ variables.

Similarly when we assume $\sum_{d \in D_X} \text{overflow}(X, d) \leq \sum_{d \in D_X} \text{underflow}(X, d)$.

If (1) does not hold, there is no variable assignment that satisfies the `gcc`. □

Without assumption (1), the variable-based violation measure for the `gcc` cannot be applied. Therefore, we introduce the following value-based violation measure, which can also be applied when assumption (1) does not hold.

Definition 7 (Value-based cost). *For* $\texttt{gcc}(X, l, u)$ *the value-based cost of violation is*

$$\sum_{d \in D_X} \text{overflow}(X, d) + \text{underflow}(X, d).$$

We denote the value-based violation measure for the `gcc` by $\text{violation}_{\texttt{soft\_gcc}[\text{val}]}$.

3.1 Graph Representation

First, we introduce the concept of a flow in a directed graph, following Schrijver [20, pp. 148–150].

A directed graph is a pair $\mathcal{G} = (V, A)$ where V is a finite set of vertices and A is a family[1] of ordered pairs from V, called arcs. For $v \in V$, let $\delta^{\text{in}}(v)$ and $\delta^{\text{out}}(v)$ denote the family of arcs entering and leaving v respectively.

A (directed) walk in $\mathcal{G}$ is a sequence $P = v_0, a_1, v_1, \ldots, a_k, v_k$ where $k \geq 0$, $v_0, v_1, \ldots, v_k \in V$, $a_1, \ldots, a_k \in A$ and $a_i = (v_{i-1}, v_i)$ for $i = 1, \ldots, k$. If there is no confusion, P may be denoted as $P = v_0, v_1, \ldots, v_k$. A (directed) walk is called a (directed) path if $v_0, \ldots, v_k$ are distinct. A closed (directed) walk, i.e. $v_0 = v_k$, is called a (directed) circuit if $v_1, \ldots, v_k$ are distinct.

Let $s, t \in V$. We apply a capacity function $c : A \to \mathbb{R}_+$, a demand function $d : A \to \mathbb{R}_+$ and a cost function $w : A \to \mathbb{R}_+$ on the arcs. A function $f : A \to \mathbb{R}$ is called a *feasible flow* from s to t, or an $s - t$ flow, if

$$d(a) \leq f(a) \leq c(a) \qquad \text{for each } a \in A, \tag{2}$$

$$f(\delta^{\text{out}}(v)) = f(\delta^{\text{in}}(v)) \text{ for each } v \in V \setminus \{s, t\}, \tag{3}$$

where $f(S) = \sum_{a \in S} f(a)$ for all $S \subseteq A$. Property (3) ensures flow conservation, i.e. for a vertex $v \neq s, t$, the amount of flow entering v is equal to the amount of flow leaving v. The value of an $s - t$ flow f is defined as

$$\text{value}(f) = f(\delta^{\text{out}}(s)) - f(\delta^{\text{in}}(s)).$$

[1] A family is a set in which elements may occur more than once.

In other words, the value of a flow is the net amount of flow leaving s, which can be shown to be equal to the net amount of flow entering t. The cost of a flow f is defined as

$$\mathrm{cost}(f) = \sum_{a\in A} w(a)f(a).$$

A *minimum-cost flow* is a feasible $s-t$ flow of minimum cost. The *minimum-cost flow problem* is the problem of finding such a minimum-cost flow.

Theorem 1 ([15]). *A solution to* $\texttt{gcc}(X,l,u)$ *corresponds to a feasible* $s-t$ *flow of value* n *in the graph* $\mathcal{G}=(V,A)$ *with vertex set*

$$V = X \cup D_X \cup \{s,t\}$$

and edge set

$$A = A_{s\to X} \cup A_{X\to D_X} \cup A_{D_X\to t},$$

where

$$\begin{aligned} A_{s\to X} &= \{(s,x_i) \mid i \in \{1,\dots,n\}\}, \\ A_{X\to D_X} &= \{(x_i,d) \mid d\in D_i, i\in\{1,\dots,n\}\}, \\ A_{D_X\to t} &= \{(d,t) \mid d \in D_X\}, \end{aligned}$$

with demand function

$$d(a) = \begin{cases} 1 \text{ if } a \in A_{s\to X}, \\ 0 \text{ if } a \in A_{X\to D_X}, \\ l_d \text{ if } a = (d,t) \in A_{D_X\to t}, \end{cases}$$

and capacity function

$$c(a) = \begin{cases} 1 \ \text{ if } a \in A_{s\to X}, \\ 1 \ \text{ if } a \in A_{X\to D_X}, \\ u_d \text{ if } a = (d,t) \in A_{D_X\to t}. \end{cases}$$

Example 1. Consider the CSP

$$\begin{aligned} &x_1 \in \{1,2\}, x_2 \in \{1\}, x_3 \in \{1,2\}, x_4 \in \{1\}, \\ &\texttt{gcc}(X,l,u) \end{aligned}$$

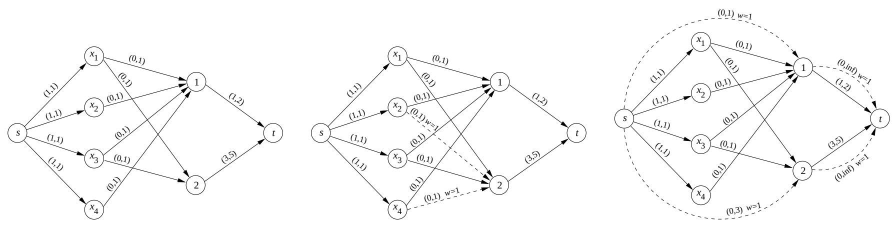
where $X = \{x_1,\dots,x_4\}$, $l_1 = 1$, $l_2 = 3$, $u_1 = 2$ and $u_2 = 5$. In Figure 1.a the corresponding graph $\mathcal{G}$ for the $\texttt{gcc}$ by applying the above procedure is presented.

3.2 Variable-Based Violation

For the variable-based violation measure, we adapt the graph $\mathcal{G}$ in the following way. We add the arc set $\tilde{A}_{X\to D_X} = \{(x_i,d) \mid d \notin D_i, i \in \{1,\dots,n\}\}$, with demand $d(a)=0$, capacity $c(a)=1$ for all arcs $a \in \tilde{A}_{X\to D_X}$. Further, we apply a cost function $w : A \to \mathbb{R}$, where

$$w(a) = \begin{cases} 1 \text{ if } a \in \tilde{A}_{X\to D_X}, \\ 0 \text{ otherwise.} \end{cases}$$

Let the resulting graph be denoted by $\mathcal{G}_{\mathrm{var}}$.



a. original gcc b. soft_gcc[var] c. soft_gcc[val]

Fig. 1. Graph representation for the gcc, the variable-based soft_gcc and the value-based soft_gcc. Demand and capacity are indicated between parentheses for each arc. Dashed arcs indicate the inserted weighted arcs.

Example 2. Consider the CSP

$$
\begin{aligned}
&x_1 \in \{1,2\}, x_2 \in \{1\}, x_3 \in \{1,2\}, x_4 \in \{1\}, z \in \{0,1,\ldots,4\}\\
&\texttt{soft\_gcc}[\text{var}](X,l,u,z)\\
&\texttt{minimize}\ z
\end{aligned}
$$

where $X = \{x_1,\ldots,x_4\}$, $l_1 = 1$, $l_2 = 3$, $u_1 = 2$ and $u_2 = 5$. In Figure 1.b the graph $\mathcal{G}_{\text{var}}$ for the soft_gcc[var] is presented.

Theorem 2. *A minimum-cost flow in the graph $\mathcal{G}_{\text{var}}$ corresponds to a solution to the* soft_gcc[var]*, minimizing the variable-based violation.*

Proof. An assignment $x_i = d$ corresponds to the arc $a = (x_i, d)$ with $f(a) = 1$. By construction, all variables need to be assigned to a value and the cost function exactly measures the variable-based cost of violation. □

The graph $\mathcal{G}_{\text{var}}$ corresponds to a particular instance of the cost-gcc [16, 17]. Hence, we can apply the filtering procedures developed for that constraint directly to the soft_gcc[var]. The soft_gcc[var] also inherits from the cost-gcc the time complexity of achieving domain consistency, being $O(n(m + n \log n))$ where $m = \sum_{i=1}^{n} |D_i|$ and $n = |X|$.

Note that [4] also consider the variable-based cost measure for a different version of the soft gcc. Their version considers the parameters l and u to be variables too. Hence, the variable-based cost evaluation becomes a rather poor measure, as we trivially can change l and u to satisfy the gcc. They fix this by restricting the set of variables to consider to be the set X, which corresponds to our situation. However, they do not provide a filtering algorithm for that case.

3.3 Value-Based Violation

For the value-based violation measure, we adapt the graph $\mathcal{G}$ in the following way. We add arc sets $A_{\text{underflow}} = \{(s,d) \mid d \in D_X\}$ and $A_{\text{overflow}} = \{(d,t) \mid d \in D_X\}$, with demand $d(a) = 0$ for all $a \in A_{\text{underflow}} \cup A_{\text{overflow}}$ and capacity

$$c(a) = \begin{cases} l_d & \text{if } a = (s,d) \in A_{\text{underflow}}, \\ \infty & \text{if } a \in A_{\text{overflow}}. \end{cases}$$

Further, we again apply a cost function $w : A \to \mathbb{R}$, where

$$w(a) = \begin{cases} 1 \text{ if } a \in A_{\text{underflow}} \cup A_{\text{overflow}}, \\ 0 \text{ otherwise.} \end{cases}$$

Let the resulting graph be denoted by $\mathcal{G}_{\text{val}}$.

Example 3. Consider the CSP

$$\begin{array}{l} x_1 \in \{1,2\}, x_2 \in \{1\}, x_3 \in \{1,2\}, x_4 \in \{1\}, z \in \{0,1,\ldots,5\} \\ \texttt{soft\_gcc}[\text{val}](X,l,u,z) \\ \texttt{minimize } z \end{array}$$

where $X = \{x_1, \ldots, x_4\}$, $l_1 = 1$, $l_2 = 2$, $u_1 = 3$ and $u_2 = 2$. In Figure 1.c the graph $\mathcal{G}_{\text{val}}$ for the `soft_gcc` with respect to value-based cost is presented.

Theorem 3. *A minimum-cost flow in the graph $\mathcal{G}_{\text{val}}$ corresponds to a solution to the* `soft_gcc`[val]*, minimizing the value-based violation.*

Proof. An assignment $x_i = d$ corresponds to the arc $a = (x_i, d)$ with $f(a) = 1$. By construction, all variables need to be assigned to a value and the cost function exactly measures the value-based cost of violation. $\square$

Unfortunately, the graph $\mathcal{G}_{\text{val}}$ does not preserve the structure of the cost-`gcc` because of the arcs $A_{\text{underflow}}$. Therefore we cannot blindly apply the same filtering algorithms. However, it is still possible to design an efficient filtering algorithm for the value-based `soft_gcc` (in the same spirit of the filtering algorithm for the cost-`gcc`), based again on flow theory. For this, we need to introduce the residual graph $\mathcal{G}^f = (V, A^f)$ of a flow f on $\mathcal{G} = (V, A)$ (with respect to c and d), where

$$A^f = \{a \mid a \in A, f(a) < c(a)\} \cup \{a^{-1} \mid a \in A, f(a) > d(a)\}.$$

Here $a^{-1} = (v, u)$ if $a = (u, v)$. We extend w to $A^{-1} = \{a^{-1} \mid a \in A\}$ by defining $w(a^{-1}) = -w(a)$ for each $a \in A$.

Theorem 4. *Let f be a minimum-cost flow in $\mathcal{G}_{\text{val}}$. Then* `soft_gcc`[val](X,l,u,z) *is domain consistent if and only if*

$$\min D_z \geq \text{cost}(f)$$

and

$$\text{cost}(f) + \text{cost}(\text{SP}(d, x_i)) \leq \max D_z \ \forall (x_i, d) \in A_{X \to D_X},$$

where $\text{cost}(\text{SP}(d, x_i))$ *denotes the cost of a shortest path from d to x_i in the residual graph $\mathcal{G}^f_{\text{val}}$.*

Proof. From flow theory [1] we know that, given a minimum-cost flow f in $\mathcal{G}_{\text{val}}$, if we enforce arc (x_i, d) to be in a minimum-cost flow $\tilde{f}$ in $\mathcal{G}_{\text{val}}$, $\text{cost}(\tilde{f}) = \text{cost}(f) + \text{cost}(\text{SP}(d, x_i))$ where $\text{SP}(d, x_i)$ is the shortest $d - x_i$ path in $\mathcal{G}^f_{\text{val}}$.

In order for a value $d \in D_i$ to be consistent, the cost of a minimum-cost flow that uses (x_i, d) should be less than or equal to $\max D_z$. By the above fact, we only need to compute a shortest path from d to x_i instead of a new minimum-cost flow. $\square$

A minimum-cost flow f in $\mathcal{G}_{\text{val}}$ can be computed in $O(m(m + n \log n))$ time (see [1]), where again $m = \sum_{i=1}^{n} |D_i|$ and $n = |X|$. Compared to the complexity of the `soft_gcc`[var], we have a factor m instead of n. This is because computing the flow for `soft_gcc`[val] is dependent on the number of arcs m rather than on the number variables n. A shortest $d - x_i$ path in $\mathcal{G}_{\text{val}}$ can be computed in $O(m + n \log n)$ time. Hence the `soft_gcc` with respect to the value-based violation measure can be made domain consistent in $O((m - n)(m + n \log n))$ time as we need to check $m - n$ arcs for consistency.

When $l = \mathbf{0}$ in `soft_gcc`[val](X, l, u, z), the arc set $A_{\text{underflow}}$ is empty. In that case, $\mathcal{G}_{\text{val}}$ has a particular structure, i.e. the only costs appear on arcs from D_X to t. As pointed out in [9] for the `soft_alldifferent` constraint, constraints with this structure can be checked for consistency in $O(nm)$ time, and domain consistency can be achieved in $O(m)$ time. The result is obtained by exploiting the strongly connected components[2] in $\mathcal{G}_{\text{val}}$ restricted to vertex sets X and D_X.

4 Soft Regular Constraint

A `regular` constraint [12] on a fixed-length sequence of finite-domain variables requires that the corresponding sequence of values taken by these variables belong to a given regular language. A *deterministic finite automaton* (DFA) may be described by a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of states, Σ is an alphabet, $\delta : Q \times \Sigma \to Q$ is a partial transition function, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final (or accepting) states. A finite sequence of symbols from an alphabet is called a *string*. Strings processed by M and ending in an accepting state from F are said to belong to the language defined by M, denoted $L(M)$. The languages recognized by DFAs are precisely regular languages.

Given a sequence $\mathbf{x} = \langle x_1, x_2, \ldots, x_n \rangle$ of finite-domain variables with respective domains $D_1, D_2, \ldots, D_n \subseteq \Sigma$, there is a natural interpretation of the set of possible instantiations of $\mathbf{x}$, $D_1 \times D_2 \times \cdots \times D_n$, as a subset of all strings of length n over Σ, Σ^n. We are now ready to state the constraint.

Definition 8 (Regular language membership constraint). *Let* $M = (Q, \Sigma, \delta, q_0, F)$ *denote a deterministic finite automaton and* $\mathbf{x}$ *a sequence of finite-domain variables* $\langle x_1, x_2, \ldots, x_n \rangle$ *with respective domains* D_1, D_2, $\ldots$,

[2] A strongly connected component in a directed graph $\mathcal{G} = (V, A)$ is a subset of vertices $S \subseteq V$ such that there exists a directed $u - v$ path in $\mathcal{G}$ for all $u, v \in S$.

$D_n \subseteq \Sigma$. Under a regular language membership constraint `regular`$(\mathbf{x}, M)$*, any sequence of values taken by the variables of* $\mathbf{x}$ *corresponds to a string in* $L(M)$*.*

In [12], a domain consistency algorithm for the `regular` constraint processed the sequence $\mathbf{x}$ with the automaton M, building a layered directed multi-graph $\mathcal{G} = (N^1, N^2, \ldots, N^{n+1}, A)$ where each layer $N^i = \{q_0^i, q_1^i, \ldots, q_{|Q|-1}^i\}$ contains a different node for each state of M and arcs only appear between consecutive layers. Each arc corresponds to a consistent variable-value pair: there is an arc from q_k^i to q_ℓ^{i+1} if and only if there exists some $v_j \in D_i$ such that $\delta(q_k, v_j) = q_\ell$ and the arc belongs to a path from q_0 in the first layer to a member of F in the last layer. The existence of such an arc, labeled v_j, constitutes a support for variable x_i taking value v_j.

For example, consider a sequence $\mathbf{x}$ of five variables with $D_1 = \{a, b, c, o\}$, $D_2 = \{b, o\}$, $D_3 = \{a, c, o\}$, $D_4 = \{a, b, o\}$, and $D_5 = \{a\}$. Figure 2 gives an automaton M (with its initial state labeled 1) and the resulting graph for constraint `regular`$(\mathbf{x}, M)$. As a result, value b is removed from D_2 and D_4.

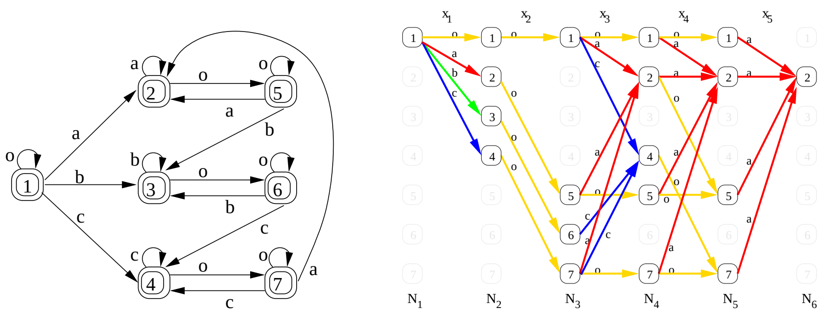


Fig. 2. A DFA (left) and its layered directed graph $\mathcal{G}$ (right).

4.1 Cost Definition

We first give a generic definition for a soft version of the `regular` constraint.

Definition 9 (Soft regular language membership constraint). *Let $M = (Q, \Sigma, \delta, q_0, F)$ denote a deterministic finite automaton and* $\mathbf{x}$ *a sequence of finite-domain variables $\langle x_1, x_2, \ldots, x_n\rangle$ with respective domains D_1, D_2, …, $D_n \subseteq \Sigma$. Let z be a finite-domain variable of domain $D_z \subset \mathbb{N}$ representing the cost of a violation and let $d : \Sigma^\star \times \Sigma^\star \to \mathbb{N}$ be some distance function over strings. Under a* soft regular language membership constraint `soft_regular`$[d](\mathbf{x}, M, z)$*, for any sequence of values σ taken by the variables of* $\mathbf{x}$ *we have* $\min_{\sigma' \in L(M)}\{d(\sigma, \sigma')\} = z$*.*

Our first instantiation of the distance function yields the variable-based cost:

Definition 10 (Hamming distance). *The number of positions in which two strings of same length differ is called their* Hamming distance*.*

Intuitively, such a distance represents the number of symbols we need to change to go from one string to the other, or equivalently the number of variables whose value must change. Using the Hamming distance for d in the previous definition, z becomes the variable-based cost.

Another distance function that is often used with strings is the following:

Definition 11 (Edit distance). *The smallest number of insertions, deletions, and substitutions required to change one string into another is called the* edit distance*.*

It captures the fact that two strings that are identical except for one extra or missing symbol should be considered close to one another. For example, the edit distance between strings "bcdea" and "abcde" is two: insert an 'a' at the front of the first string and delete the 'a' from its end. The Hamming distance between the same strings is five: every symbol must be changed. Edit distance is probably a better way to measure violations of a `regular` constraint. We provide a more natural example in the area of rostering. Given a string, we call *stretch* a maximal substring of identical values. We often need to impose restrictions on the length of stretches of work shifts, and these can be expressed with a `regular` constraint. Suppose stretches of a's and b's must each be of length 2 and consider the string "abbaabbaab": its Hamming distance to a string belonging to the corresponding regular language is 5 since changing either the first a to a b or b to an a has a domino effect on the following stretches; its edit distance is just 2 since we can insert an a at the beginning to make a legal stretch of a's and remove the b at the end. In this case, the edit distance reflects the number of illegal stretches whereas the Hamming distance is proportional to the length of the string.

4.2 Cost Evaluation and Cost-Based Filtering

For both cost measures, we proceed by modifying the layered directed graph $\mathcal{G}$ built for the "hard" version of `regular` into graph $\mathcal{G}_{\text{var}}$. Before, we added an arc from q_k^i to q_ℓ^{i+1} if $\delta(q_k, v_j) = q_\ell$ for some $v_j \in D_i$; now we relax it slightly to any $v_j \in \Sigma$. This only makes a difference if the domains of the variables are not initially full. Arcs are never removed in $\mathcal{G}_{\text{var}}$ but their labels are updated instead. The label of an arc (q_k^i, q_ℓ^{i+1}) is generalized to the invariant $V_{ik\ell} = \{v_j \in D_i \mid \delta(q_k, v_j) = q_\ell\}$; as values are removed from the domain of variable x_i, they are also removed from the corresponding $V_{ik\ell}$'s. The cost of using an arc (q_k^i, q_ℓ^{i+1}) for variable-value pair $\langle x_i, v_j \rangle$ will be zero if v_j belongs to $V_{ik\ell}$ and some positive integer cost otherwise. This cost represents the penalty for an individual violation. In the remainder of the section we will consider unit costs but the framework also makes it possible to use varying costs, e.g. to distinguish between insertions and substitutions when using the edit distance. The graph on

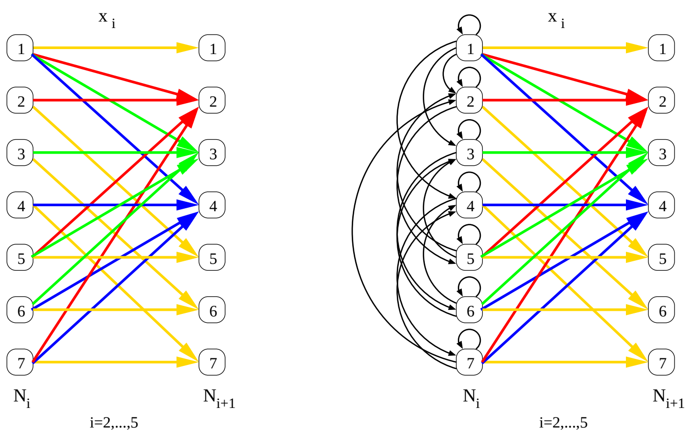


Fig. 3. Shorthand versions of $\mathcal{G}_{\text{var}}$ (left) and $\mathcal{G}_{\text{edit}}$ (right) for the DFA of Figure 2.

the left at Figure 3 is a shorthand version of $\mathcal{G}_{\text{var}}$ for the automaton of Figure 2. Since all values in Σ are considered, the same arcs appear between consecutive layers. What changes from one layer to the other are the $V_{ik\ell}$ labels.

Taking into account *substitutions*, common to both Hamming and edit distances, is immediate from the previous modification. It is not difficult to see that the introduction of costs transforms a supporting path in the domain consistency algorithm for `regular` into a zero-cost path in the modified graph. The cost of a shortest path from q_0 in the first layer to a member of F in the last layer corresponds to the smallest number of variables forced to take a value outside of their domain.

Theorem 5. *A minimum-cost path from $u \in N^1$ to $v \in N^{n+1}$ in $\mathcal{G}_{\text{var}}$ corresponds to a solution to* `soft_regular`[var] *minimizing the variable-based cost (Hamming distance).*

Just as the existence of a path through a given arc representing a variable-value pair constituted a support for that pair in the filtering algorithm for `regular`, the existence of a path whose cost doesn't exceed $\max D_z$ constitutes a support for that variable-value pair in a cost-based filtering algorithm for `soft_regular`.

Theorem 6. `soft_regular`[var]$(\mathbf{x}, M, z)$ *is domain consistent on* $\mathbf{x}$ *and bound consistent on z if and only if*

$$\min_{q_f \in F}\{\text{cost}(\text{SP}(q_0^1, q_f^{n+1}))\} \leq \min D_z$$

and

$$\min_{q_k \in Q, q_f \in F}\{\text{cost}(\text{SP}(q_0^1, q_k^i)) + \text{cost}(\text{SP}(q_\ell^{i+1}, q_f^{n+1}))\} \leq \max D_z, \quad \forall x_i \in \mathbf{x}, v_j \in D_i$$

where $\delta(q_k, v_j) = q_\ell$ and $\text{cost}(\text{SP}(u, v))$ *denotes the cost of a shortest path from u to v in $\mathcal{G}_{\text{var}}$.*

Computing shortest paths from the initial state in the first layer to every other node and from every node to a final state in the last layer can be done in $O(n\,|\delta|)$ time[3] through topological sorts because of the special structure of the graph. That computation can also be made incremental in the same way as in [12]. Recently, that same result was independently obtained in [3]. We however go further by considering edit distance, for which insertions and deletions are allowed as well.

For *deletions* we need to allow "wasting" a value without changing the current state. To this effect, we add to $\mathcal{G}_{\text{var}}$ an arc $(q_k^i, q_k^{i+1})\ \forall\ 1 \leq i \leq n, q_k \in Q$, with $V_{ikk} = \emptyset$, if it isn't already present in the graph. To allow *insertions*, inspired by ϵ-transitions in DFAs, we introduce some special arcs between nodes in the same layer: if $\exists v \in \Sigma$ such that $\delta(q_k, v) = q_\ell$ then we further add an arc $(q_k^i, q_\ell^i)\ \forall\ 1 \leq i \leq n+1$ with fixed positive cost. Figure 3 provides an example of the resulting graph (on the right). Unfortunately, those special arcs modify the structure of the graph since cycles (of strictly positive cost) are introduced. Consequently shortest paths can no longer be computed through topological sorts. An efficient implementation of Dijkstra's algorithm increases the time complexity to $O(n\,|\delta| + n\,|Q| \log(n\,|Q|))$. Regardless of this increase in computational cost, Theorems 5 and 6 can be generalized to hold for `soft_regular`[edit] as well.

5 Aggregating Soft Constraints

The preceding sections have introduced filtering algorithms based on different violation measures for two soft global constraints. If these filtering techniques are to be effective, especially in the presence of soft constraints of a different nature, they must be able to cooperate and communicate. Even though there are many avenues for combining soft constraints, the objective almost always remains to minimize constraint violations. We propose here a small extension to the approach of [13], where meta-constraints on the cost variables of soft constraints are introduced. We illustrate this approach with the newly introduced `soft_gcc`.

Definition 12 (Soft global cardinality aggregator). *Let $\mathcal{S}$ be a set of soft constraints and $z_i \in D_{z_i}$ the variable indicating the violation cost of $S_i \in \mathcal{S}$. The* soft global cardinality aggregator (`sgca`) *is defined as* `soft_gcc`$[\star](Z, l, u, z_{\text{agg}})$ *where $Z = \{z_1, \ldots, z_{|\mathcal{S}|}\}$, l_i, u_i is the interval defining the allowed number of occurrences of each value in the domain of z_i and $z_{\text{agg}} \in D_{z_{\text{agg}}} \subseteq \mathbb{N}$ the cost variable based on the violation measure $\star$.*

When all constraints are either satisfied or violated ($Z \in \{0,1\}^{|\mathcal{S}|}$) the Max-CSP approach can be easily obtained by setting $l_1 = 0$, $u_1 = 0$, $violation(Z) = \sum_{d \in D_Z} overflow(Z, d)$ and reading the number of violations in z_{agg}. The `sgca` could also be used as in [13] to enforce homogeneity (in a soft manner) or to define other violation measures like restricting the number of highly violated

[3] $|\delta|$ refers to the number of transitions in the automaton.

constraint. For instance, we could wish to impose that no more then a certain number of constraints are highly violated, but since we cannot guarantee that this is possible the use of `sgca` allows to state this wish without risking to create an inconsistent problem. More generally, by defining the values of l and u accordingly it is possible to limit (or at least attempt to limit) the number violated constraints by violation cost. Another approach could be to set all u to 0 and adjust the violation function so that higher violation costs are more penalized. The use of soft meta-constraints, when possible, is also an alternative to the introduction of disjunctive constraints since they need not be satisfied for the problem to be consistent.

In the original meta-constraint framework, similar behaviour can be established by applying a cost-`gcc` to Z. For instance, we can define for each pair (z_i, d) $(d \in D_{z_i})$ a cost d which penalizes higher violations more. With the `soft_gcc`, this cost function can be stated as $\text{violation}(Z) = \sum_{d \in D_Z} d \cdot \text{overflow}(Z, d)$. However, as for this variant of the `soft_gcc` we have $l = \mathbf{0}$, the `soft_gcc` will be much more efficient than the cost-`gcc`, as was discussed at the end of Section 3. In fact, the `sgca` can be checked for consistency in $O(nm)$ time and made domain consistent in $O(m)$ time (where $n = |\mathcal{S}|$ and $m = \cup_i |D_{z_i}|$ whenever $l = \mathbf{0}$ and $\text{violation}(Z) = \sum_{d \in D_Z} F(d) \cdot \text{overflow}(Z, d)$ for any cost function $F : D_Z \to \mathbb{R}_+$.

6 Conclusion

We have presented soft versions of two global constraints: the global cardinality constraint and the regular constraint. Different violation measures have been presented and the corresponding filtering algorithms achieving domain consistency have been introduced. These new techniques are based on the addition of “relaxation arcs” in the underlying graph and the use of known graph algorithms. We also have proposed to extend the Meta-Constraint framework for combining constraint violations by using the soft version of `gcc`.

Since these two constraints are very useful to solve Personnel Rostering Problems the next step is thus the implementation of these algorithms in order to model such problems and benchmark these new constraints.

References

1. R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows.* Prentice Hall, 1993.
2. K.R. Apt. *Principles of Constraint Programming.* Cambridge University Press, 2003.
3. N. Beldiceanu, M. Carlsson, and T. Petit. Deriving Filtering Algorithms from Constraint Checkers. In *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP 2004)*, volume 3258 of *LNCS*. Springer, 2004.
4. N. Beldiceanu and T. Petit. Cost Evaluation of Soft Global Constraints. In *CPAIOR 2004: Proceedings of the First International Conference*, volume 3011 of *LNCS*, pages 80–95. Springer, 2004.

5. S. Bistarelli. *Semirings for Soft Constraint Solving and Programming*, volume 2962 of *LNCS*. Springer, 2004.
6. D. Dubois, H. Fargier, and H. Prade. The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction. In *Proceedings of the Second IEEE International Conference on Fuzzy Systems*, volume 2, pages 1131–1136, 1993.
7. H. Fargier, J. Lang, and T. Schiex. Selecting preferred solutions in fuzzy constraint satisfaction problems. In *Proceedings of the first European Congress on Fuzzy and Intelligent Technologies*, 1993.
8. R. Freuder and M. Wallace. Partial Constraint Satisfaction. *Artificial Intelligence*, 58:21–70, 1992.
9. W.J. van Hoeve. A Hyper-Arc Consistency Algorithm for the Soft Alldifferent Constraint. In *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP 2004)*, volume 3258 of *LNCS*. Springer, 2004.
10. J. Larrosa. Node and Arc Consistency in Weighted CSP. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pages 48–53. AAAI Press/The MIT Press, 2002.
11. J. Larrosa and T. Schiex. In the quest of the best form of local consistency for Weighted CSP. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 239–244. Morgan Kaufmann, 2003.
12. G. Pesant. A Regular Language Membership Constraint for Finite Sequences of Variables. In *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP 2004)*, volume 3258 of *LNCS*. Springer, 2004.
13. T. Petit, J.-C. Régin, and C. Bessière. Meta constraints on violations for over constrained problems. In *Proceedings of the 12th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 358–365, 2000.
14. T. Petit, J.-C. Régin, and C. Bessière. Specific Filtering Algorithms for Over Constrained Problems. In *Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming (CP 2001)*, volume 2239 of *LNCS*, pages 451–463. Springer, 2001.
15. J.-C. Régin. Generalized Arc Consistency for Global Cardinality Constraint. In *Proceedings of AAAI/IAAI*, volume 1, pages 209–215. AAAI Press/The MIT Press, 1996.
16. J.-C. Régin. Arc Consistency for Global Cardinality Constraints with Costs. In *Proceedings of the Fifth International Conference on Principles and Practice of Constraint Programming (CP'99)*, volume 1713 of *LNCS*, pages 390–404. Springer, 1999.
17. J.-C. Régin. Cost-Based Arc Consistency for Global Cardinality Constraints. *Constraints*, 7:387–405, 2002.
18. T. Schiex. Possibilistic Constraint Satisfaction Problems or "How to handle soft constraints ?". In *Proceedings of the 8th Annual Conference on Uncertainty in Artificial Intelligence*, pages 268–275. Morgan Kaufmann, 1992.
19. T. Schiex, H. Fargier, and G. Verfaillie. Valued Constraint Satisfaction Problems: Hard and Easy Problems. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 631–639. Morgan Kaufmann, 1995.
20. A. Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency.* Springer, 2003.