On Global Warming: Flow-Based Soft Global Constraints

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

[1] Cornell University, Department of Computer Science,
4130 Upson Hall, Ithaca, NY 14853, USA
vanhoeve@cs.cornell.edu
[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. In case a CSP is over-constrained, it is natural to allow some constraints, called *soft constraints*, to be violated. We propose a generic method to soften global constraints that can be represented by a flow in a graph. Such constraints are softened by adding *violation arcs* to the graph and then computing a minimum-weight flow in the extended graph to measure the violation. We present efficient propagation algorithms, based on different violation measures, achieving domain consistency for the *alldifferent* constraint, the *global cardinality* constraint, the *regular* constraint and the *same* constraint.

1 Introduction

Many real-life problems are over-constrained. In personnel rostering problems for example, people often have conflicting preferences. To such problems there does not exist a feasible solution that respects all preferences. However, we still want to find *some* solution, preferably one that keeps conflicts to a minimum. In the case of the personnel rostering example, we may want to construct a roster in which the number of respected preferences is spread equally among the employees.

In constraint programming, we seek an (optimal) feasible solution to a given problem. Hence, we cannot apply constraint programming directly to over-constrained problems because it finds no solution. As a remedy several methods have been proposed. Most of these methods introduce so-called *soft constraints* that are allowed to be violated. Constraints that are not allowed to be violated are called *hard constraints*. Most methods then try to find a solution that minimizes the number of violated constraints, or some other measure of constraint violation.

Global constraints are often key elements in successfully modeling and solving real-life applications with constraint programming. For many *soft* global constraints, however, no efficient propagation algorithm was available, up to very

* This work was for a large part carried out while the author was at the Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands.

recently. In this paper we distinguish two main objectives with respect to soft global constraints: *useful violation measures* and *efficient propagation algorithms.*

In many cases we can represent a solution to a global constraint as a property in some graph representation of the constraint. For example, a solution to the `alldifferent` constraint corresponds to a matching in a particular graph; see [22]. There exists a large class of such global constraints, see for example [4] for a collection. In this paper, we focus on global constraints for which a solution can be represented by a flow in a graph.

Our method adds *violation arcs* to the graph representation of a global constraint. To these arcs we assign a cost, corresponding to some violation measure of the constraint. Each tuple in the constraint has an associated cost of violation. If the tuple satisfies the constraint, the corresponding flow does not use any violation arc, and the cost is 0. If the tuple does not satisfy the constraint, the corresponding flow must use violation arcs, whose costs sum up to the violation cost of this tuple.

This approach allows us to define and implement useful violation measures for soft global constraints. Moreover, we present an efficient generic propagation algorithm for soft global constraints, making use of flow theory. We apply our method to several global constraints that are well-known to the constraint programming community: the `alldifferent`, the `gcc`, the `regular`, and the `same` constraints, which will be recalled in turn. To each of these global constraints we apply several violation measures, some of which are new.

This paper is organized as follows. In Section 2 we give an overview of related literature. Section 3 provides some background on constraint programming and flow theory. Then our method to soften global constraints is presented in Section 4. We first discuss the general concepts of constraint softening and violation measures. Then we describe the addition of violation arcs to the graph representation and present the generic domain consistency propagation algorithm. The next four sections apply our method to the four global constraints mentioned above. For each constraint we present new or existing violation measures and the corresponding graph representations. We also analyze the corresponding propagation algorithms to achieve domain consistency. Finally, in Section 9 a conclusion is given.

2 Related Literature

The best-known framework to handle soft constraints is the *Partial-CSP* framework [14]. This framework 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, the objective is equivalent to minimizing the number of violated constraints. It has been extended to the *Weighted-CSP* framework in [17] and [18], associating a degree of violation (not just a boolean value) to each constraint and minimizing the sum of all weighted violations. The *Possibilistic-CSP* framework in [29] 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, i.e. the minimization of the highest preference level for a violated constraint. The *Fuzzy-CSP* framework in [11], [12] and [28] 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. With valued-CSPs [30] and semi-rings [8] it is possible to encode Max-CSP, weighted CSPs, Fuzzy CSPs, and Possibilistic CSPs.

Another approach to model and solve over-constrained problems was proposed in [26] and refined in [7]. The idea is to identify with each soft constraint S a "cost" variable z, and replace the constraint S by the disjunction

$$(S \wedge (z = 0)) \ \vee \ (\overline{S} \wedge (z > 0))$$

where $\overline{S}$ is a constraint of the type $z = \mu(S)$ for some violation measure $\mu(S)$ depending on S. The newly defined problem is not over-constrained anymore. If we ask to minimize the (weighted) sum of violation costs, we can solve the problem with a traditional constraint programming solver. A similar approach, specifically designed for over-constrained scheduling problems, was introduced by [3].

This approach also allows us to design specialized filtering algorithms for soft global constraints. Namely, if we treat the soft constraints as "optimization constraints" (see for example [13, 25, 32]), we can apply cost-based propagation algorithms. Constraint propagation algorithms for soft constraints based on this method were given in [21] and [15]. We follow the same method in this paper. Note that in this way we don't need to introduce new theory as we interpret certain optimization constraints as being soft constraints.

3 Background

3.1 Constraint Programming

We first introduce basic constraint programming concepts. For more information on constraint programming we refer to [2] and [10].

Let x be a variable. The *domain* of x is a set of values that can be assigned to x. In this paper we only consider variables with *finite* domains.

Let $X = x_1, x_2, \ldots, x_k$ be a sequence of variables with respective domains $D_1, D_2, \ldots, D_k$. We denote $D_X = \bigcup_{1 \leq i \leq n} D_i$. A *constraint* C on X is defined as a subset of the Cartesian product of the domains of the variables in X, i.e. $C \subseteq D_1 \times D_2 \times \cdots \times D_k$. A tuple $(d_1, \ldots, d_k) \in C$ is called a *solution* to C. We also say that the tuple *satisfies* C. A value $d \in D_i$ for some $i = 1, \ldots, k$ is *inconsistent* with respect to C if it does not belong to a tuple of C, otherwise it is *consistent*. C is *inconsistent* if it does not contain a solution. Otherwise, C is called *consistent*. A constraint is called a *binary constraint* if it is defined

on two variables. If it is defined on more than two variables, we call C a *global constraint.*

Sometimes a constraint C is defined on variables X together with a certain set of parameters p, for example a set of cost values. In such cases, we denote the constraint as $C(X, p)$ for syntactical convenience.

A *constraint satisfaction problem*, or a *CSP*, is defined by a finite sequence of variables $\mathcal{X} = x_1, x_2, \ldots, x_n$ with respective domains $\mathcal{D} = D_1, D_2, \ldots, D_n$, together with a finite set of constraints $\mathcal{C}$, each on a subsequence of $\mathcal{X}$. This is written as $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$. The goal is to find an assignment $x_i = d_i$ with $d_i \in D_i$ for $i = 1, \ldots, n$, such that all constraints are satisfied. This assignment is called a *solution to the CSP.* A *constraint optimization problem* problem is a CSP together with an objective function on $\mathcal{X}$ that has to be optimized.

The solution process of constraint programming interleaves *constraint propagation*, or *propagation* in short, and *search.* The search process essentially consists of enumerating all possible variable-value combinations, until we find a solution or prove that none exists. We say that this process constructs a *search tree.* To reduce the exponential number of combinations, *constraint propagation* is applied to each node of the search tree: Given the current domains and a constraint C, remove domain values that do not belong to a solution to C. This is repeated for all constraints until no more domain value can be removed.

In order to be effective, constraint propagation algorithms should be efficient, because they are applied many times during the solution process. Further, they should remove as many inconsistent values as possible. If a constraint propagation algorithm for a constraint C removes *all* inconsistent values from the domains with respect to C, we say that it makes C *domain consistent.* More formally:

Definition 1 (Domain consistency, [19]). *A constraint C on the variables $x_1, \ldots, x_k$ with respective domains $D_1, \ldots, D_k$ is called* domain consistent *if for each variable x_i and each value $d_i \in D_i$ $(i = 1, \ldots, k)$, there exist a value $d_j \in D_j$ for all $j \neq i$ such that $(d_1, \ldots, d_k) \in C$.*

In the literature, domain consistency is also referred to as *hyper-arc consistency* or *generalized-arc consistency.*

In this paper our goal is to find efficient propagation algorithms for soft global constraints with useful violation measures, that achieve domain consistency.

3.2 Flow Theory

We present some concepts of flow theory that are necessary to understand this paper. For more information on flow theory we refer to [1] and [31, Chapters 6–15].

Let $G = (V, A)$ be a directed graph, or *digraph*, with vertex set V and arc set A. We denote $m = |A|$. Let $s, t \in V$. A function $f : A \to \mathbb{R}$ is called a *flow from s to t*, or an *$s-t$ flow*, if (i) $f(a) \geq 0$ for each $a \in A$, and (ii) $f(\delta^{\text{out}}(v)) = f(\delta^{\text{in}}(v))$ for each $v \in V \setminus \{s, t\}$, where $\delta^{\text{in}}(v)$ and $\delta^{\text{out}}(v)$ denote the multiset of

arcs entering and leaving v, respectively. Here $f(S) = \sum_{a \in S} f(a)$ for all $S \subseteq A$. Property (ii) 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 $\text{value}(f) = f(\delta^{\text{out}}(s)) - f(\delta^{\text{in}}(s))$. In other words, the value of a flow is the net amount of flow leaving s. This is equal to the net amount of flow entering t.

Let $d : A \to \mathbb{R}_+$ and $c : A \to \mathbb{R}_+$ be a "demand" function and a "capacity" function, respectively[4]. We say that a flow f is *feasible* if $d(a) \leq f(a) \leq c(a)$ for each $a \in A$. Let $w : A \to \mathbb{R}$ be a "weight" function. We often also refer to such function as a "cost" function. For a directed path P in G we define $w(P) = \sum_{a \in P} w(a)$. Similarly for a directed circuit. The *weight* of any function $f : A \to \mathbb{R}$ is defined as $\text{weight}(f) = \sum_{a \in A} w(a) f(a)$. A feasible flow is called a *minimum-weight flow* if it has minimum weight among all feasible flows with the same value. Given a graph $G = (V, A)$ with $s, t \in V$ and a number $\phi \in \mathbb{R}_+$, the *minimum-weight flow problem* is: find a minimum-weight $s-t$ flow with value ϕ.

A feasible $s-t$ flow in G with value ϕ and minimum weight can be found using the *successive shortest path algorithm* [31, p. 175–176]. It can be proved that for integer demand and capacity functions it finds an integer $s-t$ flow with minimum weight. The time complexity of the algorithm is $O(\phi \cdot \text{SP})$, where SP is the time to compute a shortest directed path in G. Although faster algorithms exist for general minimum-weight flow problems, this algorithm suffices when applied to our problems. This is because in our case the value of any flow is bounded above by the number of variables in the constraint.

Given a minimum-weight $s-t$ flow, we want to compute the additional weight when an unused arc is forced to be used. In order to do so we need the following notation. Let f be an $s-t$ flow in G. The *residual graph* of f (with respect to d and c) is defined as $G_f = (V, A_f)$ 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 $a \in A$. Any directed circuit C in G_f gives an undirected circuit in $G = (V, A)$. We define $\chi^C \in \mathbb{R}^A$ by

$$\chi^C(a) = \begin{cases} 1 \text{ if } C \text{ traverses } a, \\ -1 \text{ if } C \text{ traverses } a^{-1}, \\ 0 \text{ if } C \text{ traverses neither } a \text{ nor } a^{-1}, \end{cases}$$

for $a \in A$. We have the following result.

Theorem 1. *Let f be a minimum-weight $s-t$ flow of value ϕ in $G = (V, A)$ with $f(a) = 0$ for some $a \in A$. Let C be a directed circuit in G_f with $a \in C$, minimizing $w(C)$. Then $f' = f + \varepsilon\chi^C$, where ε is subject to $d \leq f + \varepsilon\chi^C \leq c$, has minimum weight among all $s-t$ flows g in G with* $\text{value}(g) = \phi$ *and $g(a) = \varepsilon$. If C does not exist, f' does not exist. Otherwise,* $\text{weight}(f') = \text{weight}(f) + \varepsilon \cdot w(C)$.

[4] Here $\mathbb{R}_+$ denotes $\{x \in \mathbb{R} \mid x \geq 0\}$. Similarly for $\mathbb{Q}_+$.

The proof of Theorem 1 relies on the fact that for a minimum-weight flow f in G, the residual graph G_f does not contain directed circuits with negative weight; see also [1, p. 337–339].

Finally a digraph $G = (V, E)$ is *strongly connected* if for any two vertices $u, v \in V$ there is a directed path from u to v. A maximally strongly connected non-empty subgraph of a digraph G is called a *strongly connected component* of G.

4 Outline of Method

In this section we first define how we soften global constraints, and define some general violation measures. Then we present a generic constraint propagation algorithm for a class of soft global constraints: those that can be represented by a flow in a graph.

4.1 Constraint Softening and Violation Measures

As stated before, the idea of the scheme in [26] is as follows. To each soft constraint are associated a violation measure and a cost variable that measures this violation. Then the CSP (or COP) is transformed into a COP where all constraints are hard, and the (weighted) sum of cost variables is minimized.

Let $X = x_1, \ldots, x_n$ be variables with respective finite domains $D_1, \ldots, D_n$.

Definition 2 (Violation measure). *A* violation measure *of a constraint* $C(x_1, \ldots, x_n)$ *is a function* $\mu : D_1 \times \cdots \times D_n \to \mathbb{Q}_+$ *defined over the possible tuples of* C *with the property that* $\mu(d_1, \ldots, d_n) = 0$ *if and only if* $(d_1, \ldots, d_n) \in C$.

Definition 3 (Constraint softening). *Let* z *be a variable with finite domain* D_z *and* $C(x_1, \ldots, x_n)$ *a constraint with a violation measure* μ. *Then*

$$\texttt{soft\_}C(x_1, \ldots, x_n, z, \mu) = \\ \{(d_1, \ldots, d_n, d) \mid d_i \in D_i, d \in D_z,\ \mu(d_1, \ldots, d_n) \leq d\}$$

is the soft version of C *with respect to* μ.

Thus z is a "cost" variable that represents the measure of violation of C — $\max D_z$ and $\min D_z$ respectively represent the maximum and minimum value of violation that is allowed, given the current state of the solution process.[5] It should be noted that z does not *equal* the measure of violation, because we only consider its bounds. The reason for this is that the current definitions allows us to establish domain consistency on soft constraints efficiently. If we would define $\mu(d_1, \ldots, d_n) = d$ instead, it would make that task NP-complete.

There may be several natural ways to evaluate the degree to which a global constraint is violated and these are usually not equivalent. Two general measures are the *variable-based* violation measure and the *decomposition-based* violation measure.

[5] We assume that cost variable z is being minimized.

Definition 4 (Variable-based violation measure, [21]). *Let C be a constraint on the variables $x_1, \ldots, x_n$ and let $d_1, \ldots, d_n$ be an instantiation of variables such that $d_i \in D_i$ for $i = 1, \ldots, n$. The* variable-based violation measure μ_{var} *of C is the minimum number of variables that need to change their value in order to satisfy C.*

For the decomposition-based violation measure we make use of the binary decomposition of a constraint.

Definition 5 (Binary decomposition, [9]). *Let C be a constraint on the variables $x_1, \ldots, x_n$. A* binary decomposition *of C is a minimal set of binary constraints $C_{\text{dec}} = \{C_1, \ldots, C_k\}$ (for integer $k > 0$) on the variables $x_1, \ldots, x_n$ such that the solution set of C equals the solution set of $\bigcap_{i=1}^{k} C_i$.*

Note that we can extend the definition of binary decomposition by defining the constraints in C_{dec} on arbitrary variables, such that the solution set of $\bigwedge_{i=1}^{k} C_i$ is mapped to the solution set of C and vice versa, as proposed in [27]. In this paper this extension is not necessary, however.

Definition 6 (Decomposition-based violation measure, [21]). *Let C be a constraint on the variables $x_1, \ldots, x_n$ for which a binary decomposition C_{dec} exists and let $d_1, \ldots, d_n$ be an instantiation of variables such that $d_i \in D_i$ for $i = 1, \ldots, n$. The* decomposition-based violation measure μ_{dec} *of C is the number of violated constraints in C_{dec}.*

In [21], the decomposition-based violation measure is referred to as *primal graph based violation cost.*

Alternative measures exist for specific constraints. For the soft `gcc` and the soft `regular` constraint, we introduce new violation measures that are likely to be more effective in practical applications.

After we have assigned a violation measure to each soft constraint, we can recast our problem. Consider a CSP (or COP) of the form $P = (X, D, C_{\text{hard}} \cup C_{\text{soft}})$, where C_{hard} and C_{soft} denote the set of hard and soft constraints of P, respectively. We soften each constraint $C_i \in C_{\text{soft}}$ using the violation measure it has been assigned and a cost variable z_i with domain D_{z_i} $(i = 1, \ldots, |C_{\text{soft}}|)$ that represents this measure. Then we transform P into the COP $\tilde{P} = (\tilde{X}, \tilde{D}, \tilde{C})$ where $\tilde{X} = X \cup \{z_1, \ldots, z_{|C_{\text{soft}}|}\}$, $\tilde{D}$ contains their corresponding domains, and $\tilde{C}$ contains C_{hard} and the soft version of each constraint in C_{soft}.

4.2 Propagation of Soft Constraints

We assume from now on that constraint $C(x_1, \ldots, x_n)$ can be represented by a directed graph $G = (V, A)$ with capacity function $c : A \to \mathbb{N}$ that has the following properties:

- a pair (x_i, d) is represented by at least one arc $a \in A$ for $i = 1, \ldots, n$ and all $d \in D_i$, and $c(a) = 1$,

– a tuple $(d_1, \ldots, d_n) \in C$ is represented by a feasible flow f of value $\phi \in \mathbb{R}_+$ in G such that $f(a) = 1$ for an arc a representing the pair (x_i, d_i) for $i = 1, \ldots, n$, and $f(a) = 0$ for all arcs a representing the pair (x_i, d) with $d \neq d_i$ for $i = 1, \ldots, n$.

In case the constraint C is violated, it is impossible to find a flow with the above mentioned properties in the corresponding digraph G. We propose to extend G with certain arcs, such that it becomes possible to find a feasible flow corresponding to a solution. We call these arcs *violation arcs*, and they are denoted by $\tilde{A}$. Violation arcs may appear anywhere in the graph. The only restriction we impose on them is that after their addition, there exists a feasible flow in $\tilde{G} = (V, A \cup \tilde{A})$ that represents a solution to C.

There is no set recipe for the way violation arcs should be added, as their introduction greatly varies from one constraint to another and even between violation measures of the same constraint. The intuition is to consider all the properties imposed by a given graph and decide which ones should be relaxed and which ones should still be enforced. Then, violation arcs are added so that solutions exhibiting a relaxed property now constitute feasible flows.

The next step is to make a connection with the violation measures for a constraint. This is done by applying a "cost" function $w : A \cup \tilde{A} \to \mathbb{Q}$ to $\tilde{G}$ in the following way. For all arcs $a \in A$ we define $w(a) = 0$, while $w(a) \geq 0$ for all arcs $a \in \tilde{A}$. Then each flow f in $\tilde{G}$ has an associated cost

$$\sum_{a \in A \cup \tilde{A}} w(a) f(a) = \sum_{a \in \tilde{A}} w(a) f(a).$$

After the addition of violation arcs, a solution to C corresponds to a feasible flow in $\tilde{G}$ with an associated cost. If the flow does not use any violation arcs, this cost is 0. Otherwise, the cost of the flow depends on the costs we impose on the violation arcs. Hence we can define a violation measure as follows. For each solution to C we define its cost of violation as the minimum-weight flow in $\tilde{G}$ that represents this solution. Conversely, for many existing violation measures it is possible to choose a particular set of violation arcs and associated costs, such that a minimum-weight flow in $\tilde{G}$ representing a solution is exactly the cost of violation of that solution. In the following sections we provide several examples for different constraints and different violation measures. We often denote the extended digraph $\tilde{G}$ as G_μ to indicate its dependence on some violation measure μ.

In other words, if C is represented by the digraph $G = (V, A)$ and we can find violation arcs $\tilde{A}$ and a cost function $w : A \cup \tilde{A} \to \mathbb{Q}_+$ that represent violation measure μ, then the soft version of C with respect to μ, i.e. $\texttt{soft\_}C(x_1, \ldots, x_n, z, \mu)$, is represented by the digraph $G_\mu = (V, A \cup \tilde{A})$ with cost function w. By construction, we have the following result:

Theorem 2. *The constraint* $\texttt{soft\_}C(x_1, \ldots, x_n, z, \mu)$ *is domain consistent if and only if*

Algorithm 1 domain consistency for $\texttt{soft\_}C(X, z, \mu)$

```
set minimum = ∞
construct G_μ = (V, A ∪ Ã)
for x_i ∈ X do
  for d ∈ D_i do
    compute a minimum-weight flow f in G_μ that represents a solution to soft_C,
    with f(a) = 1 for some a ∈ A that represents (x_i, d)
    if cost(f) > max D_z then
      remove d from D_i
    if cost(f) < minimum then
      set minimum = cost(f)
if min D_z < minimum then
  set min D_z = minimum
```

i) for all $i \in \{1, \ldots, n\}$ *and all* $d \in D_i$ *there is an arc* $a \in A$ *representing* (x_i, d) *such that there exists a feasible flow* f *in* G_μ *representing a solution to* $\texttt{soft\_}C$ *with* $f(a) = 1$ *and* $\text{cost}(f) \leq \max D_z$,

ii) the minimum cost of all such flows f *is not larger than* $\min D_z$.

Theorem 2 gives rise to the following propagation algorithm, presented as Algorithm 1. For a sequence of variables $X = x_1, \ldots, x_n$, and a constraint $\texttt{soft\_}C(X, z, \mu)$ the algorithm first builds the digraph G_μ that represents the constraint. Then, for all variable-value pairs (x_i, d) we check whether the pair belongs to a solution, i.e. whether there exists a flow in G_μ that represents a solution containing $x_i = d$, with cost at most $\max D_z$. If this is not the case, we can remove d from D_i. Finally, we update $\min D_z$, if necessary.

The time complexity of this algorithm is $O(ndK)$ where d is the maximum domain size and K is the time complexity to compute a flow in G_μ corresponding to a solution to $\texttt{soft\_}C$. However, we can improve the efficiency by applying Theorem 1.

The resulting, more efficient, algorithm is as follows. We first compute an initial minimum-weight flow f in G_μ representing a solution. Then for all arcs $a = (u, v)$ representing (x_i, d) with $f(a) = 0$, we compute a minimum-weight directed path P from v to u in the residual graph $(G_\mu)_f$. Together with a, P forms a directed circuit. Provided that $c(b) \geq 1$ for all arcs $b \in P$, we reroute the flow over the circuit and obtain a flow f'. Then $\text{cost}(f') = \text{cost}(f) + \text{cost}(P)$, because $w(a) = 0$ for all $a \in A$. If $\text{cost}(f') > \max D_z$ we remove d from the domain of x_i.

This reduces the time complexity of the algorithm to $O(K + nd \cdot \text{SP})$ where SP denotes the time complexity to compute a minimum-weight directed path in G_μ. It should be noted that a similar algorithm was first applied in [24, 25] to make the "`cost-gcc`" domain consistent.

5 Soft Alldifferent Constraint

5.1 Definitions

The `alldifferent` constraint on a sequence of variables specifies that all variables should be pairwise different. To the `alldifferent` constraint we apply two measures of violation: the variable-based violation measure μ_{var} and the decomposition-based violation measure μ_{dec}.

Let $X = x_1, \ldots, x_n$ be a sequence of variables with respective finite domains $D_1, \ldots, D_n$. For $\texttt{alldifferent}(x_1, \ldots, x_n)$ we have

$$\mu_{\text{var}}(x_1, \ldots, x_n) = \textstyle\sum_{d \in D_X} \max(|\{i \mid x_i = d\}| - 1, 0),$$
$$\mu_{\text{dec}}(x_1, \ldots, x_n) = |\{(i,j) \mid x_i = x_j, \text{ for } i < j\}|$$

Example 1. Consider the following over-constrained CSP:

$$x_1 \in \{a,b\}, x_2 \in \{a,b\}, x_3 \in \{a,b\}, x_4 \in \{b,c\},$$
$$\texttt{alldifferent}(x_1, x_2, x_3, x_4).$$

We have $\mu_{\text{var}}(b,b,b,b) = 3$ and $\mu_{\text{dec}}(b,b,b,b) = 6$. □

If we apply Definition 3 to the `alldifferent` constraint using the measures μ_{var} and μ_{dec}, we obtain $\texttt{soft\_alldifferent}(x_1, \ldots, x_n, z, \mu_{\text{var}})$ and $\texttt{soft\_alldifferent}(x_1, \ldots, x_n, z, \mu_{\text{dec}})$. Each of the violation measures μ_{var} and μ_{dec} gives rise to a different domain consistency propagation algorithm. Before we present them, we introduce the graph representation of the `alldifferent` constraint in terms of flows.

5.2 Graph Representation

A solution to the `alldifferent` constraint corresponds to a flow in a particular graph:

Theorem 3. [22] *A solution to* $\texttt{alldifferent}(x_1, \ldots, x_n)$ *corresponds to an integer feasible* $s-t$ *flow of value* n *in the digraph* $\mathcal{A} = (V, A)$ *with vertex set*

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

and arc set

$$A = A_s \cup A_X \cup A_t,$$

where

$$A_s = \{(s, x_i) \mid i \in \{1, \ldots, n\}\},$$
$$A_X = \{(x_i, d) \mid d \in D_i, i \in \{1, \ldots, n\}\},$$
$$A_t = \{(d, t) \mid d \in D_X\},$$

with capacity function $c(a) = 1$ *for all* $a \in A$.

Proof. With an integer feasible $s-t$ flow f of value n in $\mathcal{A}$ we associate the assignment $x_i = d$ for all arcs $a = (x_i, d) \in A_X$ with $f(a) = 1$. Because $c(a) = 1$ for all $a \in A_s \cup A_t$, this is indeed a solution to the `alldifferent` constraint. As $\text{value}(f) = n$, all variables have been assigned a value. Similarly, each solution to the `alldifferent` gives rise to a corresponding appropriate flow in $\mathcal{A}$. □

Figure 1 gives the corresponding graph representation for the CSP of Example 1.

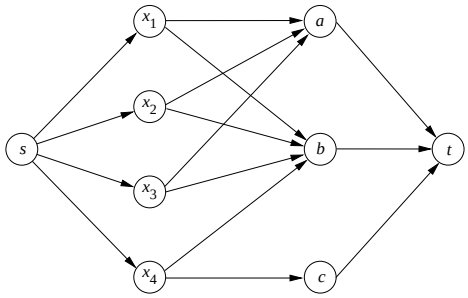


Fig. 1. Graph representation for the `alldifferent` constraint. For all arcs the capacity is 1.

5.3 Variable-Based Violation Measure

The results in this section are originally due to [21]. We state their result in terms of our method, by adding violation arcs to the graph representing the `alldifferent` constraint.

To graph $\mathcal{A}$ of Theorem 3 we add the violation arcs $\tilde{A}_t = \{(d,t) \mid d \in D_X\}\}$ (in fact, $\tilde{A}_t$ is a copy of A_t), with capacity $c(a) = \left|\delta^{\text{in}}(d)\right| - 1$ for all arcs $a = (d,t) \in \tilde{A}_t$. Further, we apply a cost function w as described in Section 4.2, with a uniform cost of 1 for violation arcs. Let the resulting digraph be denoted by $\mathcal{A}_{\text{var}}$ (see Figure 2 for an illustration on Example 1).

The intuition is to add violation arcs that capture the fact that more than one variable can be assigned to the same value. For example, in Figure 2.b four variables are assigned to the same value b. To make this possible, there are three units of flow on the corresponding violation arc, while one unit of flow can use the arc without violation cost. Indeed, we should change the value of at least three variables in order to satisfy the `alldifferent` constraint.

Corollary 1. *The constraint* `soft_alldifferent`$(x_1, \ldots, x_n, z, \mu_{\text{var}})$ *is domain consistent if and only if*

i) for every arc $a \in A_X$ *there exists an integer feasible* $s-t$ *flow* f *of value* n *in* $\mathcal{A}_{\text{var}}$ *with* $f(a) = 1$ *and* $\text{weight}(f) \leq \max D_z$*, and*
ii) $\min D_z \geq \text{weight}(f)$ *for a feasible minimum-weight* $s-t$ *flow* f *of value* n *in* $\mathcal{A}_{\text{var}}$*.*

Proof. The weights on the arcs in $\tilde{A}_t$ are chosen such that the weight of a minimum-weight flow of value n is exactly μ_{var} for the corresponding solution. The result follows from Theorem 2. □

The constraint `soft_alldifferent`$(x_1, \ldots, x_n, z, \mu_{\text{var}})$ can now be made domain consistent in the following way. First we compute a minimum-weight flow f in $\mathcal{A}_{\text{var}}$ in $O(m\sqrt{n})$ time, using the algorithm in [16]. If $\text{weight}(f) > \max D_z$

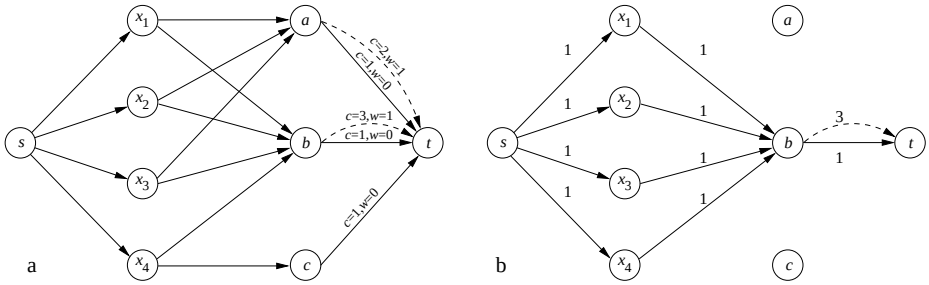


Fig. 2. a) Graph representation for the variable-based `soft_alldifferent` constraint. For all arcs the capacity $c = 1$, unless specified otherwise. Dashed arcs indicate the inserted weighted arcs with weight $w = 1$. b) Example: arcs and associated flows used in solution $x_1 = x_2 = x_3 = x_4 = b$ of weight 3.

or $\min D_z > \text{weight}(f)$ the constraint is inconsistent. Otherwise, we distinguish two situations: either $\text{weight}(f) < \max D_z$ or $\text{weight}(f) = \max D_z$.

Forcing a flow to use an unused arc in A_X can only increase $\text{weight}(f)$ by 1. Hence, if $\min D_z \leq \text{weight}(f) < \max D_z$, all arcs in A_X are consistent.

If $\min D_z \leq \text{weight}(f) = \max D_z$, an unused arc $a = (x_i, d)$ in A_X is consistent if and only if there exists a $d - x_i$ path in $(\mathcal{A}_{\text{var}})_f$ with weight 0. We can find these paths in $O(m)$ time by breadth-first-search. Finally, we update $\min D_z = n - |M|$ if $\min D_z < n - |M|$. Then, by Corollary 1, the `soft_alldifferent`$(x_1, \ldots, x_n, z, \mu_{\text{var}})$ is domain consistent.

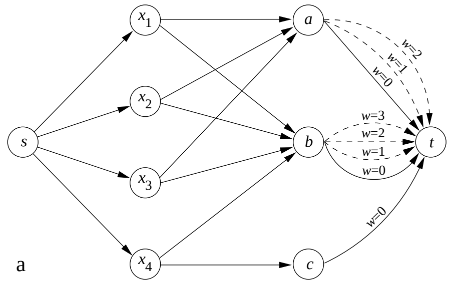
5.4 Decomposition-Based Violation Measure

The results in this section are originally presented in [15]. For the decomposition-based `soft_alldifferent` constraint, we add the following violation arcs to the graph representing the `alldifferent` constraint.

In the graph $\mathcal{A}$ of Theorem 3 we replace the arc set A_t by $\tilde{A}_t = \{(d,t) \mid d \in D_i, i = 1, \ldots, n\}$, with capacity $c(a) = 1$ for all arcs $a \in \tilde{A}_t$. Note that $\tilde{A}_t$ contains parallel arcs if two or more variables share a domain value. If there are k parallel arcs (d,t) between some $d \in D_X$ and t, we distinguish them by numbering the arcs as $(d,t)_0, (d,t)_1, \ldots, (d,t)_{k-1}$ in a fixed but arbitrary way. One can view the arcs $(d,t)_0$ to be the original arc set A_t.

We apply a cost function $w : A \cup \tilde{A}_t \to \mathbb{N}$ as follows. If $a \in \tilde{A}_t$, so $a = (d,t)_i$ for some $d \in D_X$ and integer i, the value of $w(a) = i$. Otherwise $w(a) = 0$. Let the resulting digraph be denoted by $\mathcal{A}_{\text{dec}}$ (see Figure 3 for an illustration on Example 1).

The decomposition-based violation measure captures the amount of violation a solution presents with respect to the binary decomposition of the constraint. For example, in Figure 3.b four variables are assigned to the same value b. To make this possible, there are three units of flow that need to use a violation arc, while one unit of flow can use the arc without violation cost. Indeed, for the



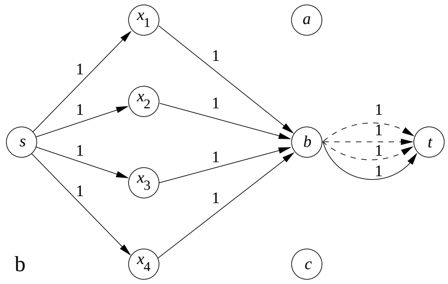


Fig. 3. a) Graph representation for the decomposition-based `soft_alldifferent` constraint. For all arcs the capacity $c = 1$. Dashed arcs indicate the inserted weighted arcs with weight w as specified. b) Example: arcs and associated flows used in solution $x_1 = x_2 = x_3 = x_4 = b$ of weight 6.

first variable assigned to b, say x_1, there is no violated binary constraint and the corresponding unit of flow may use the arc without violation cost. The second variable assigned to b, say x_2, violates one binary constraint, namely $x_1 \neq x_2$. Indeed it uses the violation arc with the next lowest possible cost, i.e. 1. The following variable assigned to b, say x_3, violates two binary constraints (involving x_1 and x_2), which corresponds to using the violation arc with cost 2. Finally, the fourth variable assigned to b, x_4, violates three binary constraints and uses the violation arc with cost 3.

Corollary 2. *The constraint* `soft_alldifferent`$(x_1, \ldots, x_n, z, \mu_{\text{dec}})$ *is domain consistent if and only if*

i) for every arc $a \in A_X$ there exists an integer feasible $s - t$ flow f of value n in $\mathcal{A}_{\text{dec}}$ with $f(a) = 1$ and $\text{weight}(f) \leq \max D_z$*, and*

ii) $\min D_z \geq \text{weight}(f)$ *for a feasible minimum-weight $s - t$ flow f of value n in $\mathcal{A}_{\text{dec}}$.*

Proof. The weights on the arcs in $\tilde{A}_t$ are chosen such that the weight of a minimum-weight flow of value n is exactly μ_{dec}. Namely, the first arc entering a value $d \in D_X$ causes no violation and chooses outgoing arc with weight 0. The k-th arc that enters d causes $k - 1$ violations and chooses outgoing arc with weight $k - 1$. The result follows from Theorem 2. □

The constraint `soft_alldifferent`$(x_1, \ldots, x_n, z, \mu_{\text{dec}})$ can be made domain consistent by applying Algorithm 1. We first compute a minimum-weight flow f in $\mathcal{A}_{\text{dec}}$. We do this by computing n shortest $s - t$ paths in the residual graph. Because there are only weights on arcs in A_t, each shortest path takes $O(m)$ time to compute. Hence we can compute f in $O(nm)$ time. If $\text{weight}(f) > \max D_z$ we know that the constraint is inconsistent.

To identify the arcs $a = (x_i, d) \in A_X$ that belong to a flow g with $\text{value}(g) = n$ and $\text{weight}(g) \leq \max D_z$ we apply Theorem 1. Thus, we search for a shortest

$d - x_i$ path in $(\mathcal{A}_{\text{dec}})_f$ that together with a forms a directed circuit C. We can compute all such shortest paths in $O(m)$ time, using the following result:

Theorem 4. [15] *Let* $\texttt{soft\_alldifferent}(x_1, \ldots, x_n, z, \mu_{\text{dec}})$ *be consistent and let* f *be an integer feasible minimum-weight flow in* $\mathcal{A}_{\text{dec}}$ *of value* n*. Then* $\texttt{soft\_alldifferent}(x_1, \ldots, x_n, z, \mu_{\text{dec}})$ *can be made domain consistent in* $O(m)$ *time.*

Proof. The complexity of the filtering algorithm depends on the computation of the minimum-weight $d - x_i$ paths in $(\mathcal{A}_{\text{dec}})_f$ for arcs $(x_i, d) \in A_X$. We make use of the fact that only arcs $a \in \tilde{A}_t$ contribute to the cost of such path.

Consider the strongly connected components of the graph $(\tilde{\mathcal{A}}_{\text{dec}})_f$ which is a copy of $(\mathcal{A}_{\text{dec}})_f$ where s and t and all their incident arcs are removed. Let P be a minimum-weight $d - x_i$ path P in $\mathcal{A}_f$. If P is equal to d, x_i then $f(x_i, d) = 1$ and $\text{cost}(P) = 0$. Otherwise, either x_i and d are in the same strongly connected component of $(\tilde{\mathcal{A}}_{\text{dec}})_f$, or not. In case they are in the same strongly connected component, P can avoid t in $\mathcal{A}_f$, and $\text{cost}(P) = 0$. In case x_i and d are in different strongly connected components, P must visit t, and we do the following.

Split t into two vertices t^{in} and t^{out} such that $\delta^{\text{in}}(t^{\text{in}}) = \delta^{\text{in}}(t)$, $\delta^{\text{out}}(t^{\text{in}}) = \varnothing$, and $\delta^{\text{in}}(t^{\text{out}}) = \varnothing$, $\delta^{\text{out}}(t^{\text{out}}) = \delta^{\text{out}}(t)$. For every vertex $v \in X \cup D_X$ we can compute the minimum-weight path from v to t^{in} and from t^{out} to v in total $O(m)$ time.

The strongly connected components of $(\tilde{\mathcal{A}}_{\text{dec}})_f$ can be computed in $O(n+m)$ time, following [33]. Hence the total time complexity of achieving domain consistency is $O(m)$, as $n < m$. □

Hence, we update $D_i = D_i \setminus \{d\}$ if $\text{weight}(f) + \text{weight}(C) > \max D_z$. Finally, we update $\min D_z = \text{weight}(f)$ if $\min D_z < \text{weight}(f)$. Then, by Corollary 2, the $\texttt{soft\_alldifferent}(x_1, \ldots, x_n, z, \mu_{\text{dec}})$ is domain consistent.

6 Soft Global Cardinality Constraint

6.1 Definitions

A global cardinality constraint (gcc) on a sequence of variables specifies for each value in the union of their domains an upper and lower bound to the number of variables that are assigned to this value. A domain consistency propagation algorithm for the gcc was developed in [23], making use of network flows.

Let $l_d, u_d \in \mathbb{N}$ with $l_d \leq u_d$ for all $d \in D_X$.

Definition 7 (Global cardinality constraint, [23]).

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

Note that the gcc is a generalization of the alldifferent constraint. If we set $l_d = 0$ and $u_d = 1$ for all $d \in D_X$, the gcc is equal to the alldifferent constraint.

In order to define measures of violation for the gcc, it is convenient to introduce for each domain value a "shortage" function $s : D_1 \times \cdots \times D_n \times D_X \to \mathbb{N}$ and an "excess" function $e : D_1 \times \cdots \times D_n \times D_X \to \mathbb{N}$ as follows:

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

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

To the gcc we apply two measures of violation: the variable-based violation measure μ_{var} and the *value-based* violation measure μ_{val} that we will define in this section. The next lemma expresses μ_{var} in terms of the shortage and excess functions.

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

$$\mu_{\text{var}}(X) = \max\left(\sum_{d \in D_X} s(X,d), \sum_{d \in D_X} e(X,d)\right)$$

provided that

$$\sum_{d \in D_X} l_d \leq |X| \leq \sum_{d \in D_X} u_d. \tag{1}$$

Proof. Note that if (1) does not hold, there is no variable assignment that satisfies the gcc, and μ_{var} cannot be applied.

Applying μ_{var} corresponds to the minimal number of re-assignments of variables until both $\sum_{d \in D_X} s(X,d) = 0$ and $\sum_{d \in D_X} e(X,d) = 0$.

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

Similarly when we assume $\sum_{d \in D_X} s(X,d) \leq \sum_{d \in D_X} e(X,d)$. Then we need to re-assign exactly $\sum_{d \in D_X} e(X,d)$ variables. □

We introduce the following violation measure for the gcc, which can also be applied when assumption (1) does not hold.

Definition 8 (Value-based violation measure). *For* $\texttt{gcc}(X,l,u)$ *the* value-based violation measure *is*

$$\mu_{\text{val}}(X) = \sum_{d \in D_X} \left(s(X,d) + e(X,d)\right).$$

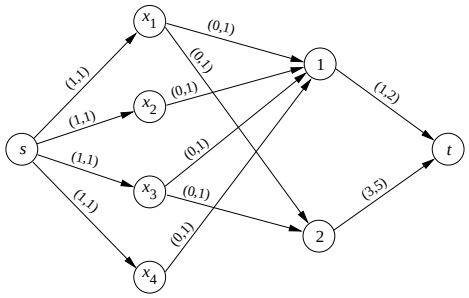


Fig. 4. Graph representation for the gcc. Demand and capacity are indicated between parentheses for each arc a as $(d(a), c(a))$.

6.2 Graph Representation

The gcc has the following graph representation:

Theorem 5. [**23**] *A solution to* $\mathtt{gcc}(X, l, u)$ *corresponds to an integer feasible* $s-t$ *flow of value* n *in the digraph* $\mathcal{G} = (V, A)$ *with vertex set*

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

$$\textit{and arc set} \quad A = A_s \cup A_X \cup A_t,$$

$$\begin{array}{rl} & A_s = \{(s, x_i) \mid i \in \{1, \ldots, n\}\}, \\ \textit{where} & A_X = \{(x_i, d) \mid d \in D_i, i \in \{1, \ldots, n\}\}, \\ & A_t = \{(d, t) \mid d \in D_X\}, \end{array}$$

$$\textit{with demand function} \quad d(a) = \begin{cases} 1 \text{ if } a \in A_s, \\ 0 \text{ if } a \in A_X, \\ l_d \text{ if } a = (d, t) \in A_t, \end{cases}$$

$$\textit{and capacity function} \quad c(a) = \begin{cases} 1 \text{ if } a \in A_s, \\ 1 \text{ if } a \in A_X, \\ u_d \text{ if } a = (d, t) \in A_t. \end{cases}$$

Example 2. Consider the following CSP:

$$\begin{array}{l} x_1 \in \{1, 2\}, x_2 \in \{1\}, x_3 \in \{1, 2\}, x_4 \in \{1\}, \\ \mathtt{gcc}(x_1, x_2, x_3, x_4, [1, 3], [2, 5]). \end{array}$$

In Figure 4 the corresponding graph representation of the gcc is presented. □

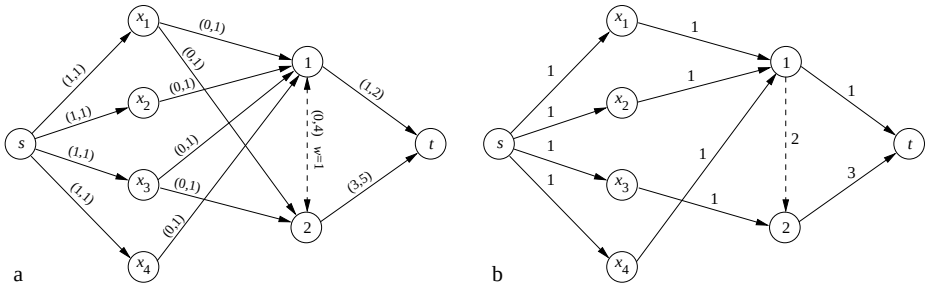


Fig. 5. a) Graph representation for the variable-based `soft_gcc`. Demand and capacity are indicated between parentheses for each arc a as $(d(a), c(a))$. Dashed arcs indicate the inserted weighted arcs with weight $w = 1$. b) Example: arcs and associated flows used in solution $x_1 = x_2 = x_4 = 1, x_3 = 2$ of weight 2.

6.3 Variable-Based Violation Measure

For the variable-based violation measure, we adapt the graph $\mathcal{G}$ of Theorem 5 in the following way. We add the violation arcs

$$\tilde{A} = \{(d_i, d_j) \mid d_i, d_j \in D_X, i \neq j\},$$

with demand $d(a) = 0$, capacity $c(a) = n$ and weight $w(a) = 1$ for all arcs $a \in \tilde{A}$. Let the resulting digraph be denoted by $\mathcal{G}_{\text{var}}$ (see Figure 5 for an illustration on Example 2). The objective of the variable-based violation measure is to count the variables which need to change value in order for a solution to be feasible with respect to the original `gcc`. Here the intuition is immediate: if a variable needs to change its value, it uses the violation arc from its current value to its desired value. Each such change induces an additional weight of 1 to the variable-based violation measure.

Corollary 3. *The constraint* $\texttt{soft\_gcc}(X, l, u, z, \mu_{\text{var}})$ *is domain consistent if and only if*

i) for every arc $a \in A_X$ *there exists an integer feasible* $s - t$ *flow* f *of value* n *in* $\mathcal{G}_{\text{var}}$ *with* $f(a) = 1$ *and* $\text{weight}(f) \leq \max D_z$, *and*
ii) $\min D_z \geq \text{weight}(f)$ *for a feasible minimum-weight* $s - t$ *flow* f *of value* n *in* $\mathcal{G}_{\text{var}}$.

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 result follows from Theorem 2. □

The constraint $\texttt{soft\_gcc}(X, l, u, z, \mu_{\text{var}})$ can be made domain consistent by applying Algorithm 1. We first compute a minimum-weight flow f in $\mathcal{G}_{\text{var}}$ by

computing n shortest $s-t$ paths in the residual graph in $O(n(m+n\log n))$ time. In order to make the `soft_gcc` with respect to μ_{var} domain consistent, we need to check $m-n$ arcs for consistency. Instead of computing $m-n$ shortest paths in the residual graph, we do the following; see [24, 25]. We compute for each (variable) vertex in X the distance to all other vertices in $O(m+n\log n)$ time. Alternatively, this may be done for all (value) vertices in D_X instead. Hence we can check all arcs for consistency in $O(\Delta(m+n\log n))$ time, where $\Delta = \min(n, |D_X|)$.

In [7] the variable-based violation measure is considered 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 propagation algorithm for that case.

6.4 Value-Based Violation Measure

For the value-based violation measure, we adapt the graph $\mathcal{G}$ of Theorem 5 in the following way. We add the violation arcs

$$A_{\text{shortage}} = \{(s,d) \mid d \in D_X\} \text{ and } A_{\text{excess}} = \{(d,t) \mid d \in D_X\},$$

with demand $d(a) = 0$ for all $a \in A_{\text{shortage}} \cup A_{\text{excess}}$ and capacity

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

We again apply a cost function w assigning unit cost to violation arcs. Let the resulting digraph be denoted by $\mathcal{G}_{\text{val}}$ (see Figure 6 for an illustration on Example 2).

This value-based violation measure is not concerned with variables but rather tries to evaluate the amount of flow missing or exceeding the requirement of the original `gcc`. As we are only interested with the demands and capacities stated on values, we allow the flow to completely bypass the layer of nodes corresponding to the variables.

Corollary 4. *The constraint* $\texttt{soft\_gcc}(X, l, u, z, \mu_{\text{val}})$ *is domain consistent if and only if*

i) for every arc $a \in A_X$ *there exists an integer feasible* $s-t$ *flow* f *of value* n *in* $\mathcal{G}_{\text{val}}$ *with* $f(a) = 1$ *and* $\text{weight}(f) \leq \max D_z$, *and*

ii) $\min D_z \geq \text{weight}(f)$ *for a feasible minimum-weight* $s-t$ *flow* f *of value* n *in* $\mathcal{G}_{\text{val}}$.

Proof. Similar to the proof of Theorem 3. □

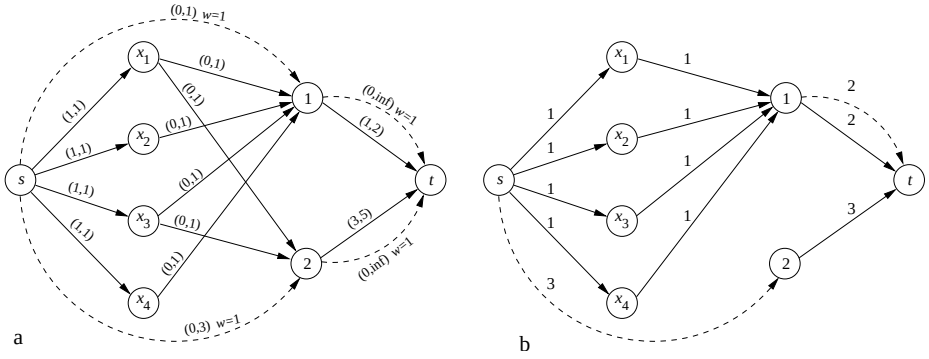


Fig. 6. a) Graph representation for the value-based `soft_gcc`. Demand and capacity are indicated between parentheses for each arc a as $(d(a), c(a))$. Dashed arcs indicate the inserted weighted arcs with weight $w = 1$. b) b) Example: arcs and associated flows used in solution $x_1 = x_2 = x_3 = x_4 = 1$ of weight 5.

We design an efficient propagation algorithm for the value-based `soft_gcc`, by applying again Algorithm 1 and Theorem 1. First, we compute a minimum-weight feasible flow in $\mathcal{G}_{\text{val}}$. For this we first need to compute n shortest paths to satisfy the demand of the arcs in A_s. In order to meet the demand of the arcs in A_t, we need to compute at most another k shortest paths, where $k = |D_X|$. Hence the total time complexity is $O((n+k)(m + n \log n))$.

In order to make the `soft_gcc` with respect to μ_{val} domain consistent, we need to check $m - n$ arcs for consistency. Similar to the variable-based `soft_gcc`, this can be done in $O(\Delta(m + n \log n))$ time, where $\Delta = \min(n, |D_X|)$.

When $l_d = 0$ for all $d \in D_X$, the arc set A_{shortage} is empty. In that case, $\mathcal{G}_{\text{val}}$ has a particular structure, i.e. the costs only appear on arcs from D_X to t. Then, similar to the reasoning for the `soft_alldifferent` constraint with respect to μ_{dec}, we can compute a minimum-weight flow of value n in $\mathcal{G}_{\text{val}}$ in $O(mn)$ time and achieve domain consistency in $O(m)$ time.

7 Soft Regular Constraint

7.1 Definitions

The `regular` constraint was introduced in [20]. It is defined on a fixed-length sequence of finite-domain variables and it states that the corresponding sequence of values taken by these variables belongs to a given regular language. A domain consistency propagation algorithm for this constraint was also provided in [20]. Particular instances of the `regular` constraint can for example be applied in rostering problems or sequencing problems.

Before we introduce the `regular` constraint we need the following definitions. A *deterministic finite automaton* (DFA) is described by a 5-tuple $M =$

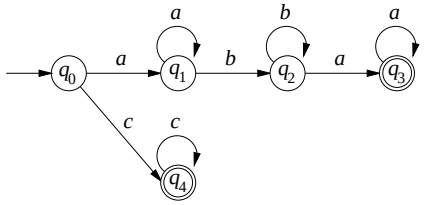


Fig. 7. A representation of a DFA with each state shown as a circle, final states as a double circle, and transitions as arcs.

$(Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of states, Σ is an alphabet, $\delta : Q \times \Sigma \to Q$ is a transition function, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final (or accepting) states. Given an input string, the automaton starts in the initial state q_0 and processes the string one symbol at the time, applying the transition function δ at each step to update the current state. The string is *accepted* if and only if the last state reached belongs to the set of final states F. Strings processed by M that are accepted are said to belong to the language defined by M, denoted by $L(M)$. For example with M depicted in Figure 7, strings *aaabaa* and *cc* belong to $L(M)$ but not *aacbba*. The languages recognized by DFAs are precisely regular languages.

Given a sequence of variables $X = x_1, x_2, \ldots, x_n$ with respective finite domains $D_1, D_2, \ldots, D_n \subseteq \Sigma$, there is a natural interpretation of the set of possible instantiations of X, i.e. $D_1 \times D_2 \times \cdots \times D_n$, as a subset of all strings of length n over Σ.

Definition 9 (Regular language membership constraint, [20]). *Let $M = (Q, \Sigma, \delta, q_0, F)$ denote a DFA and let $X = x_1, x_2, \ldots, x_n$ be a sequence of variables with respective finite domains $D_1, D_2, \ldots, D_n \subseteq \Sigma$. Then*

$$\texttt{regular}(X, M) = \{(d_1, \ldots, d_n) \mid d_i \in D_i, d_1 d_2 \cdots d_n \in L(M)\}.$$

To the `regular` constraint we apply two measures of violation: the variable-based violation measure μ_{var} and the *edit-based* violation measure μ_{edit} that we will define in this section.

Let s_1 and s_2 be two strings of the same length. The *Hamming distance* $H(s_1, s_2)$ is the number of positions in which they differ. Associating with a tuple $(d_1, d_2, \ldots, d_n)$ the string $d_1 d_2 \cdots d_n$, the variable-based violation measure can be expressed in terms of the Hamming distance:

$$\mu_{\text{var}}(X) = \min\{H(D, X) \mid D = d_1 \cdots d_n \in L(M)\}.$$

Another distance function that is often used for two strings is the following. Let s_1 and s_2 be two strings of the same length. The *edit distance* $E(s_1, s_2)$ is the smallest number of insertions, deletions, and substitutions required to change one string into another. It captures the fact that two strings that are identical except

for one extra or missing symbol should be considered close to one another. Edit distance is probably a better way to measure violations of a `regular` constraint. Consider for example a regular language in which strings alternate between pairs of a's and b's: the Hamming distance of string "*abbaabbaab*" is 5 (that is, $n/2$) since changing either the first a to a b or the first b to an a has a domino effect; the edit distance of the same string is 2 since we can insert an a at the beginning and remove a b at the end. In this case, the edit distance reflects the number of incomplete pairs whereas the Hamming distance is proportional to the length of the string rather than to the amount of violation.

Definition 10 (Edit-based violation measure). *For* $\texttt{regular}(X, M)$ *the* edit-based violation measure *is*

$$\mu_{\text{edit}}(X) = \min\{E(D, X) \mid D = d_1 \cdots d_n \in L(M)\}.$$

Example 3. Consider the CSP

$$x_1 \in \{a, b, c\}, x_2 \in \{a, b, c\}, x_3 \in \{a, b, c\}, x_4 \in \{a, b, c\},$$
$$\texttt{regular}(x_1, x_2, x_3, x_4, M)$$

with M as in Figure 7. We have $\mu_{\text{var}}(c, a, a, b) = 3$, because we have to change at least 3 variables. A corresponding valid string with Hamming distance 3 is for example *aaba*. On the other hand, we have $\mu_{\text{edit}}(c, a, a, b) = 2$, because we can delete the value c at the front and add the value a at the end, thus obtaining the valid string *aaba*. □

7.2 Graph Representation

A graph representation for the `regular` constraint was presented in [20]. Recall that $M = (Q, \Sigma, \delta, q_0, F)$.

Theorem 6. [**20**] *A solution to* $\texttt{regular}(X, M)$ *corresponds to an integer feasible* $s - t$ *flow of value* 1 *in the digraph* $\mathcal{R} = (V, A)$ *with vertex set*

$$V = V_1 \cup V_2 \cup \cdots \cup V_{n+1} \cup \{s, t\}$$

and arc set $A = A_s \cup A_1 \cup A_2 \cup \cdots \cup A_n \cup A_t$,

where $V_i = \{q_k^i \mid q_k \in Q\}$ *for* $i = 1, \ldots, n+1$,

and
$$\begin{aligned} A_s &= \{(s, q_0^1\}, \\ A_i &= \{(q_k^i, q_l^{i+1}) \mid \delta(q_k, d) = q_l \text{ for } d \in D_i\} \text{ for } i = 1, \ldots, n, \\ A_t &= \{(q_k^{n+1}, t) \mid q_k \in F\}, \end{aligned}$$

with capacity function $c(a) = 1$ *for all* $a \in A$.

Proof. Each arc in A_i corresponds to a variable-value pair: there is an arc from q_k^i to q_l^{i+1} if and only if there exists some $d \in D_i$ such that $\delta(q_k, d) = q_l$. If an arc belongs to an integer $s - t$ flow of value 1, it belongs to a path from q_0^1 to a member of F in the last V_{n+1}. Hence the assignment $x_i = d$ belongs to a solution to the `regular` constraint. □

Figure 8 gives the corresponding graph representation of the `regular` constraint from Example 3.

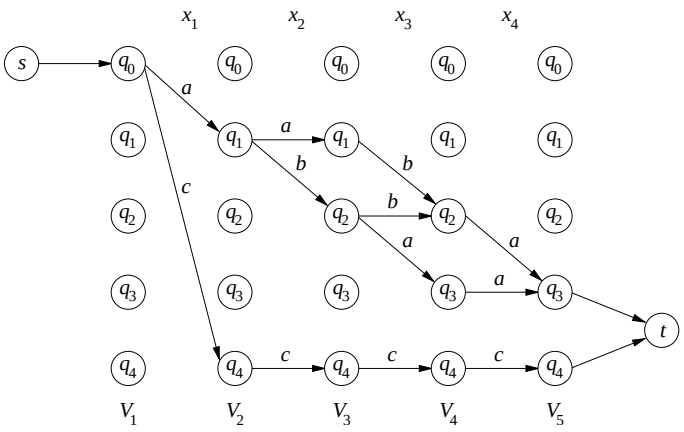


Fig. 8. Graph representation for the `regular` constraint. For all arcs the capacity is 1.

7.3 Variable-Based Violation Measure

For the variable-based `soft_regular` constraint, we add the following violation arcs to the graph representing the `regular` constraint.

To the graph $\mathcal{R}$ of Theorem 6 we add violation arcs

$$A_{\text{sub}} = \{(q_k^i, q_l^{i+1}) \mid \delta(q_k, d) = q_l \text{ for some } d \in \Sigma,\ i = 1, \ldots, n\},$$

with capacity $c(a) = 1$ for all arcs $a \in A_{\text{sub}}$. We apply the usual cost function w with unit cost. Let the resulting digraph be denoted by $\mathcal{R}_{\text{var}}$ (see Figure 9 for an illustration on Example 3).

The input automaton of this constraint specifies the allowed transitions from state to state according to different values. The objective here, in counting the minimum number of substitutions, is to make these transitions value independent. To do so, we add a violation arc between to states (q_k^i, q_l^{i+1}) if there already exists at least one valid arc between them. This means that a flow using a violation arc is in fact a solution where a variable takes a value outside of its domain. The number of such variables thus constitutes a minimum on the number of variables which need to change value.

Corollary 5. *The constraint* `soft_regular`$(X, M, z, \mu_{\text{var}})$ *is domain consistent if and only if*

i) for every arc $a \in A_1 \cup \cdots \cup A_n$ *there exists an integer feasible* $s - t$ *flow* f *of value* 1 *in* $\mathcal{R}_{\text{var}}$ *with* $f(a) = 1$ *and* $\text{weight}(f) \leq \max D_z$*, and*
ii) $\min D_z \geq \text{weight}(f)$ *for a minimum-weight* $s - t$ *flow* f *of value* 1 *in* $\mathcal{R}_{\text{var}}$.

Proof. The weight function measures exactly μ_{var}. The result follows from Theorem 2. □

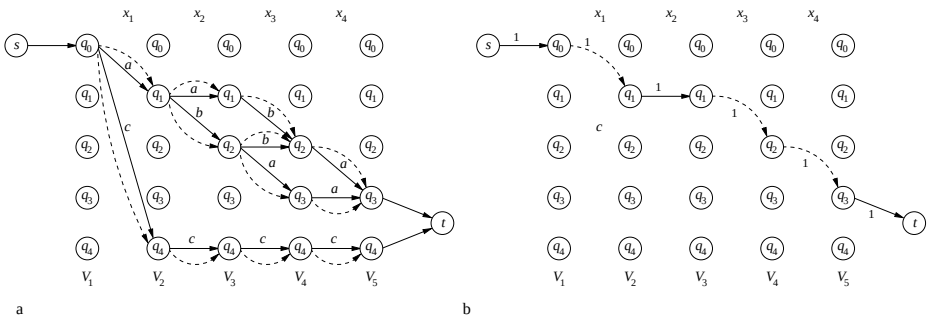


Fig. 9. Graph representation for the variable-based `soft_regular` constraint. For all arcs the capacity is 1. Dashed arcs indicate the inserted weighted arcs with weight 1. b) Example: arcs and associated flows used in solution $x_1 = c, x_2 = a, x_3 = a, x_4 = b$ of weight 3, corresponding to three substitutions from valid string $aaba$.

Note that a minimum-weight $s - t$ flow of value 1 is in fact a shortest $s - t$ path with respect to w. The constraint propagation algorithm thus must ensure that all arcs corresponding to a variable-value assignment are on an $s - t$ path with cost smaller than $\max D_z$. 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 through topological sorts because of the special structure of the graph, as observed in [20]. Here $|\delta|$ denotes the number of transitions in the corresponding DFA. Hence, the algorithm runs in $O(m)$ time, where m is the number of arcs in the graph. The computation can also be made incremental in the same way as in [20]. Note that this result has been obtained independently in [5].

7.4 Edit-Based Violation Measure

For the edit-based `soft_regular` constraint, we add the following violation arcs to the graph $\mathcal{R}$ representing the `regular` constraint. As in the previous section, we add A_{sub} to allow the substitution of a value. To allow deletions and insertions, we add violation arcs

$$A_{\text{del}} = \{(q_k^i, q_k^{i+1}) \mid i = 1, \ldots, n\}\} \setminus A$$

and $\quad A_{\text{ins}} = \{(q_k^i, q_l^i) \mid \delta(q_k, d) = q_l \text{ for some } d \in \Sigma, k \neq l, i = 1, \ldots, n+1\}.$

We set $c(a) = 1$ for each arc $a \in A_{\text{sub}} \cup A_{\text{del}} \cup A_{\text{ins}}$. We apply the usual cost function w with unit cost for violation arcs.

Let the resulting digraph be denoted by $\mathcal{R}_{\text{edit}}$ (see Figure 10 for an illustration on Example 3).

In this version we introduce three sets of violation arcs. The first one (A_{sub}) corresponding to substitutions has already been explained in section 7.3. Deletions are modeled with the arcs introduced in A_{del} which link equivalent states

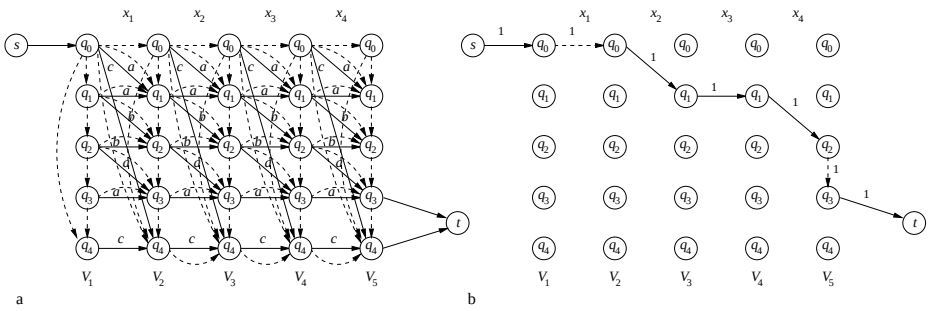


Fig. 10. a) Graph representation for the edit-based `soft_regular` constraint. For all arcs the capacity is 1. Dashed arcs indicate the inserted weighted arcs with weight 1. b) Example: arcs and associated flows used in solution $x_1 = c, x_2 = a, x_3 = a, x_4 = b$ of weight 2, corresponding to one deletion (c in position 1) and one addition (a in position 4) from valid string $aaba$.

of successive layers. The intuition is that by using such an arc it is possible to remain at a given state and simply ignore the value taken by the corresponding variable. The arcs in A_{ins} allow a flow to make more than one transition at any given layer. Since a layer corresponds to a variable and a transition is made on a symbol of the string, this behavior is equivalent to *inserting* one or more symbols. Of course one has to make sure only to allow transitions defined by the automaton.

Corollary 6. *The constraint* $\texttt{soft\_regular}(X, M, z, \mu_{\text{edit}})$ *is domain consistent if and only if*

i) for every arc $a \in A_1 \cup \cdots \cup A_n$ *there exists an integer feasible* $s - t$ *flow* f *of value* 1 *in* $\mathcal{R}_{\text{edit}}$ *with* $f(a) = 1$ *and* $\text{weight}(f) \leq \max D_z$*, and*

ii) $\min D_z \geq \text{weight}(f)$ *for a minimum-weight* $s - t$ *flow* f *of value* 1 *in* $\mathcal{R}_{\text{edit}}$.

Proof. The weight function measures exactly μ_{edit}. The result follows from Theorem 2. □

For the propagation algorithm, we proceed slightly differently from the variable-based `soft_regular` constraint because the structure of the graph is not the same: arcs within a layer may form (positive weight) directed circuits. We compute once and for all the smallest cumulative weight to go from q_k^i to q_l^i for every pair of nodes and record it in a table. This can be done through breadth-first-search from each node since every arc considered has unit weight. Notice that every layer has the same "insertion" arcs — we may preprocess one layer and use the result for all of them. In all, this initial step requires $\Theta(|Q|\,|\delta|)$ time. Then we can proceed as before through topological sort with table lookups, in $O(n\,|\delta|)$ time. The overall time complexity is therefore $O((n + |Q|)\,|\delta|) = O(m)$, where m is the number of arcs in the graph. The last step follows from $|Q| \leq n$, because otherwise some states would be unreachable.

8 Soft Same Constraint

8.1 Definitions

The same constraint is defined on two sequences of variables and states that the variables in one sequence use the same values as the variables in the other sequence. The constraint was introduced in [4]. One can also view the same constraint as demanding that one sequence is a permutation of the other. A domain consistency algorithm for the same constraint was presented in [6], making use of flow theory.

Definition 11 (Same constraint, [4]). *Let $X = x_1, \ldots, x_n$ and $Y = y_1, \ldots, y_n$ be sequences of variables with respective finite domains $D_1, \ldots, D_n$ and $D'_1, \ldots, D'_n$. Then*

$$\texttt{same}(X,Y) = \left\{ (d_1, \ldots, d_n, d'_1, \ldots, d'_n) \mid d_i \in D_i, d'_i \in D'_i, \bigcup_{i=1}^{n} \{d_i\} = \bigcup_{i=1}^{n} \{d'_i\} \right\}.$$

Note that in the above definition $\bigcup_{i=1}^{n} \{d_i\}$ and $\bigcup_{i=1}^{n} \{d'_i\}$ are multisets, in which elements may occur more than once.

To the same constraint we apply the variable-based violation measure μ_{var}. Denote the *symmetric difference* of two multisets S and T by $S \Delta T$, i.e. $S \Delta T = (S \setminus T) \cup (T \setminus S)$. For $\texttt{same}(X,Y)$ we have

$$\mu_{\text{var}}(X,Y) = \left| \left(\bigcup_{i=1}^{n} \{x_i\} \right) \Delta \left(\bigcup_{i=1}^{n} \{y_i\} \right) \right| / 2.$$

Example 4. Consider the following over-constrained CSP:

$$\begin{array}{l} x_1 \in \{a,b,c\}, x_2 \in \{c,d,e\}, x_3 \in \{c,d,e\}, \\ y_1 \in \{a,b\}, y_2 \in \{a,b\}, y_3 \in \{c,d\}, \\ \texttt{same}(x_1, x_2, x_3, y_1, y_2, y_3). \end{array}$$

We have $\mu_{\text{var}}(a,c,c,a,b,c) = 1$ because $\{a,c,c\} \Delta \{a,b,c\} = \{b,c\}$. This gives $|\{b,c\}| / 2 = 1$. □

8.2 Graph Representation

The same constraint has the following graph representation:

Theorem 7. [6] *A solution to $\texttt{same}(X,Y)$ corresponds to an integer feasible $s-t$ flow of value n in the digraph $\mathcal{S} = (V,A)$ with vertex set*

$$V = X \cup (D_X \cap D'_Y) \cup Y \cup \{s,t\}$$

and arc set $A = A_s \cup A_X \cup A_Y \cup A_t$,

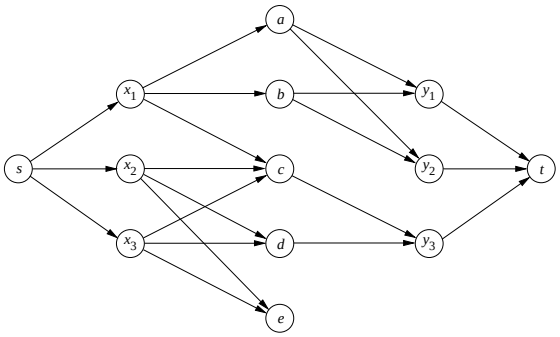


Fig. 11. Graph representation for the `same` constraint. For all arcs the capacity is 1.

$$\begin{aligned} \textit{where} \quad A_s &= \{(s, x_i) \mid i \in \{1, \ldots, n\}\}, \\ A_X &= \{(x_i, d) \mid d \in D_i \cap D_Y, i \in \{1, \ldots, n\}\}, \\ A_Y &= \{(d, y_i) \mid d \in D'_i \cap D_X, i \in \{1, \ldots, n\}\}, \\ A_t &= \{(y_i, t) \mid i \in \{1, \ldots, n\}\}, \end{aligned}$$

with capacity function $c(a) = 1$ *for all* $a \in A$.

Figure 11 gives the corresponding graph representation of the `same` constraint from Example 4.

8.3 Variable-Based Violation Measure

To the graph $\mathcal{S}$ of Theorem 7 we add the arc sets

$$\tilde{A} = \{(d_i, d_j) \mid d_i, d_j \in D_X \cup D_Y, i \neq j\},$$

with capacity $c(a) = n$ and weight $w(a) = 1$ for all arcs $a \in \tilde{A}$. Let the resulting digraph be denoted by $\mathcal{S}_{\text{var}}$ (see Figure 12 for an illustration on Example 4). The inserted violation arcs are similar to the violation arcs of the variable-based `soft_gcc`. Again, if a variable needs to change its value, it uses the violation arc from its current value to its desired value. The total induced weight corresponds exactly to the variable-based violation measure.

Corollary 7. *The constraint* $\texttt{soft\_same}(X, Y, z, \mu_{\text{var}})$ *is domain consistent if and only if*

i) *for every arc* $a \in A_X \cup A_Y$ *there exists a feasible* $s - t$ *flow* f *of value* n *in* $\mathcal{S}_{\text{var}}$ *with* $f(a) = 1$ *and* $\text{weight}(f) \leq \max D_z$, *and*
ii) $\min D_z \geq \text{weight}(f)$ *for a minimum-weight* $s - t$ *flow* f *of value* n *in* $\mathcal{S}_{\text{var}}$.

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 result follows

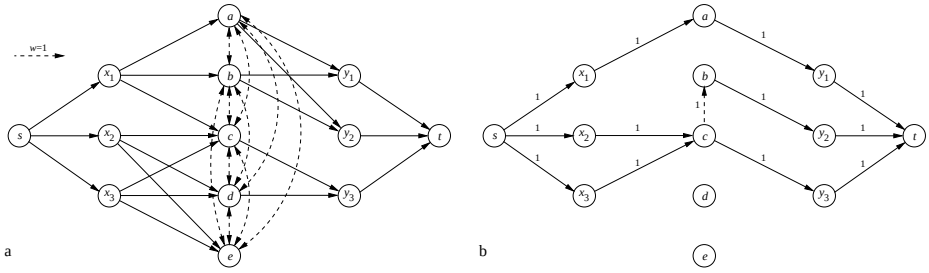


Fig. 12. a) Graph representation for the variable-based `soft_same` constraint. For all arcs the capacity is 1. Dashed arcs indicate the inserted weighted arcs with weight 1. b) Example: arcs and associated flows used in solution $x_1 = a$, $x_2 = c$, $x_3 = c$, $y_1 = a$, $y_2 = b$, $y_3 = c$ of weight 1.

from Theorem 2. □

The constraint $\texttt{soft\_same}(X, Y, z, \mu_{\text{var}})$ can be made domain consistent by applying Algorithm 1 and Theorem 1. Consistency can again be checked by computing an initial flow in $O(n(m+n\log n))$ time and, similar to the `soft_gcc`, domain consistency can be achieved in $O(\Delta(m + n\log n))$ time, where $\Delta = \min(n, |D_X|)$.

9 Conclusion

Many real-life problems are over-constrained; constraint programming deals with the issue by softening constraints. In this paper we have proposed a generic constraint propagation algorithm for soft versions of an important class of global constraints; those that can be represented by a flow in a graph. To allow solutions that were originally infeasible, we have added violation arcs to the graph, with an associated cost. Hence, a flow that represents an originally infeasible solution induces a cost. This cost corresponds to a violation measure of the soft global constraint. We have applied our method to soften the `alldifferent`, the `gcc`, the `regular`, and the `same` constraints, and described efficient domain consistency algorithms for each of them. The results are summarized in Table 1.

We have used existing violation measures, but also introduced new measures for the `soft_gcc` and the `soft_regular` constraint. For many of those violation measures we applied in this paper, the cost function assigned unit penalties. The framework makes it possible to use more varied costs, though. Namely, we may even assign any *convex* cost function to the arcs without changing the complexity of the algorithms; see [1, p. 543–565]. In fact, the cost function applied to the decomposition-based `soft_alldifferent` constraint can be regarded as a convex cost function on the aggregation of the parallel arcs. Such cost functions could also be advantageous in other cases, for example to distinguish between insertions

constraint	violation measure	consistency check	domain consistency
general case		$O(K)$	$O(nd \cdot \mathrm{SP})$
`soft_alldifferent`	variable-based	$O(m\sqrt{n})$	$O(m)$ [21]
`soft_alldifferent`	decomposition-based	$O(mn)$	$O(m)$ [15]
`soft_gcc`	variable-based	$O(n(m + n \log n))$	$O(\Delta(m + n \log n))$
`soft_gcc`	value-based	$O((n + k)(m + n \log n))$	$O(\Delta(m + n \log n))$
`soft_regular`	variable-based	$O(m)$	$O(m)$
`soft_regular`	edit-based	$O(m)$	$O(m)$
`soft_same`	variable-based	$O(n(m + n \log n))$	$O(\Delta(m + n \log n))$

Table 1. Time complexity for soft global constraints on n variables with maximum domain size d, and various violation measures. Here "consistency check" denotes the time complexity to check that the constraint is consistent, while "domain consistency" denotes the additional time complexity to make the constraint domain consistent, given at least one solution. Each algorithm is based on a graph G with m arcs. K denotes the time complexity to compute a minimum-weight flow in G, while SP denotes the time to compute a minimum-weight path in G. Finally, $k = |D_X|$ and $\Delta = \min(n, k)$.

and deletions in the `soft_regular` constraint. This could also be particularly useful in application to personnel rostering problems, where one could penalize differently (using `soft_gcc`) the over-usage of values associated to night shifts or day shifts for instance.

The approach to soft constraints which we favored in this paper allows us to restrict the amount of violation of individual (global) constraints and to use that restriction to perform domain filtering in the spirit of cost-based filtering for optimization constraints. Our results hopefully provide helpful ingredients for modeling and solving real-life over-constrained problems efficiently with constraint programming.

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. P. Baptiste, C. Le Pape, and L. Péridy. Global Constraints for Partial CSPs: A Case-Study of Resource and Due Date Constraints. In M.J. Maher and J.-F. Puget, editors, *Proceedings of the Fourth International Conference on Principles and Practice of Constraint Programming (CP98)*, volume 1520 of *LNCS*, pages 87–101. Springer, 1998.
4. N. Beldiceanu. Global Constraints as Graph Properties on Structured Network of Elementary Constraints of the Same Type. Technical Report T2000/01, SICS, 2000.
5. 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.
6. N. Beldiceanu, I. Katriel, and S. Thiel. Filtering Algorithms for the Same Constraint. In J.-C. Régin and M. Rueher, editors, *Proceedings of the First International Conference on the Integration of AI and OR Techniques in Constraint*

Programming for Combinatorial Optimization Problems (CPAIOR 2004), volume 3011 of *LNCS*, pages 65–79. Springer, 2004.
7. N. Beldiceanu and T. Petit. Cost Evaluation of Soft Global Constraints. In J.-C. Régin and M. Rueher, editors, *Proceedings of the First International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2004)*, volume 3011 of *LNCS*, pages 80–95. Springer, 2004.
8. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Satisfaction and Optimization. *Journal of the ACM*, 44(2):201–236, 1997.
9. R. Dechter. On the expressiveness of networks with hidden variables. In *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI)*, pages 555–562. AAAI Press/The MIT Press, 1990.
10. R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.
11. 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.
12. 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.
13. F. Focacci, A. Lodi, and M. Milano. Optimization-Oriented Global Constraints. *Constraints*, 7(3–4):351–365, 2002.
14. E.C. Freuder and R.J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58(1-3):21–70, 1992.
15. W.J. van Hoeve. A Hyper-Arc Consistency Algorithm for the Soft Alldifferent Constraint. In M. Wallace, editor, *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP 2004)*, volume 3258 of *LNCS*, pages 679–689. Springer, 2004.
16. J.E. Hopcroft and R.M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.
17. J. Larrosa. Node and Arc Consistency in Weighted CSP. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pages 48–53. AAAI Press, 2002.
18. 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.
19. R. Mohr and G. Masini. Good Old Discrete Relaxation. In *European Conference on Artificial Intelligence (ECAI)*, pages 651–656, 1988.
20. 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.
21. T. Petit, J.-C. Régin, and C. Bessière. Specific Filtering Algorithms for Over-Constrained Problems. In T. Walsh, editor, *Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming (CP 2001)*, volume 2239 of *LNCS*, pages 451–463. Springer, 2001.
22. J.-C. Régin. A Filtering Algorithm for Constraints of Difference in CSPs. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI)*, volume 1, pages 362–367. AAAI Press, 1994.
23. 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.

24. J.-C. Régin. Arc Consistency for Global Cardinality Constraints with Costs. In J. Jaffar, editor, *Proceedings of the Fifth International Conference on Principles and Practice of Constraint Programming (CP'99)*, volume 1713 of *LNCS*, pages 390–404. Springer, 1999.
25. J.-C. Régin. Cost-Based Arc Consistency for Global Cardinality Constraints. *Constraints*, 7:387–405, 2002.
26. J.-C. Régin, T. Petit, C. Bessière, and J.-F. Puget. An Original Constraint Based Approach for Solving over Constrained Problems. In R. Dechter, editor, *Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming (CP 2000)*, volume 1894 of *LNCS*, pages 543–548. Springer, 2000.
27. F. Rossi, C. Petrie, and V. Dhar. On the equivalence of constraint satisfaction problems. In *Proceedings of the 9th European Conference on Artificial Intelligence (ECAI)*, pages 550–556, 1990.
28. Z. Ruttkay. Fuzzy constraint satisfaction. In *Proceedings of the First IEEE Conference on Evolutionary Computing*, pages 542–547, 1994.
29. 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.
30. 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.
31. A. Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency.* Springer, 2003.
32. M. Sellmann. Cost-Based Filtering for Shorter Path Constraints. In F. Rossi, editor, *Proceedings of the Ninth International Conference on Principles and Practice of Constraint Programming (CP 2003)*, volume 2833 of *LNCS*, pages 694–708. Springer, 2003.
33. R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1:146–160, 1972.