Constraints of Difference and Equality: A Complete Taxonomic Characterisation*

Emmanuel Hebrard[1], Dániel Marx[2], Barry O'Sullivan[1], and Igor Razgon[1]

[1] Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland
{e.hebrard|b.osullivan|i.razgon}@4c.ucc.ie
[2] Budapest University of Technology and Economics
Budapest, Hungary
dmarx@cs.bme.hu

Abstract. Many combinatorial problems encountered in practice involve constraints that require that a set of variables take distinct or equal values. The AllDifferent constraint, in particular, ensures that all variables take distinct values. Two soft variants of this constraint were proposed in [4], defined either with respect to a so-called *variable* or *graph*-based cost function. When requiring similarity, as opposed to diversity, one can consider the dual definition either for the cost or for the basic constraint itself, that is, AllEqual in our case. Six cost functions can be defined by exploring every combination of these definitions. It is therefore natural to study the complexity of achieving arc consistency and bounds consistency on them. From our earlier work on this topic an open problem remained, namely achieving bounds consistency on the maximisation of the SoftAllDiff constraint when considering the graph-based cost. In this paper we resolve this problem. Therefore, we give a complete taxonomy of constraints of equality and difference, based on the alternative objective functions used for the soft variants.

1 Introduction

Constraints for reasoning about the diversity or similarity of a set of variables are ubiquitous in constraint programming. For example, in a university timetabling problem we will want to ensure that all courses taken by a particular student are held at different times. Similarly, in meeting scheduling we will want to ensure that the participants of a meeting are scheduled to meet at the same time and in the same place. Sometimes, when the problem is over-constrained, we may wish to maximise the extent to which these constraints are satisfied. Consider again our timetabling example: we might wish to maximise the number of courses that are scheduled at different times when a student's preferences cannot all be met.

* Hebrard, O'Sullivan and Razgon are supported by Science Foundation Ireland (Grant Number 05/IN/I886). Marx is supported by the Magyary Zoltán Felsőoktatási Közalapítvány and the Hungarian National Research Fund (OTKA grant 67651).

In a constraint programming setting, these requirements are normally specified using global constraints. One of the most commonly used global constraints is the AllDifferent [6], which enforces that *all* variables take pair-wise different values. A soft version of the AllDifferent constraint, the SoftAllDiff, has been proposed by the authors of [4]. They proposed two cost metrics for measuring the degree of satisfaction of the constraint, which are to be minimised or maximised: *graph*- and *variable*-based cost. The former counts the number of equalities, whilst the latter counts the number of variables, violating an AllDifferent constraint. When we wish to enforce that a set of variables take equal values, we can use the AllEqual, or its soft variant, the SoftAllEqual constraint, which we recently introduced [3].

When considering these two constraints (AllDifferent and AllEqual), these two costs (graph-based and variable-based) and objectives (minimisation and maximisation) we can define eight algorithmic problems related to constraints of difference and equality. In fact, because the graph-based costs of AllDifferent and AllEqual are dual, only six distinct problems are defined.

When we introduced the SoftAllEqual constraint one open problem remained: namely, the design of an algorithm for achieving bounds consistency on the SoftAllEqual constraint when the objective is to maximise the number of equalities achieved in the decomposition graph of the constraint, i.e. the SoftAllEqual constraint defined by the graph-based cost. In this paper we resolve this open question, and propose an efficient bounds consistency algorithm for this case. This result enables us to fully characterise the complexity of achieving arc consistency and bounds consistency on each of the eight constraints in this class. This paper, therefore, provides a complete taxonomy of constraints of difference and equality.

The remainder of this paper is organised as follows. In Section 2 we introduce the necessary technical background. A complete taxonomy of constraints of equality and difference is presented in Section 3. In Section 4 we present the main technical contribution of the paper, namely the complexity of achieving bounds consistency on the SoftAllEqual when the objective is to optimise the graph-based cost. A filtering algorithm is proposed in Section 5. Concluding remarks are made in Section 6.

2 Background

Constraint Satisfaction. A constraint satisfaction problem (CSP) is a triplet $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ where $\mathcal{X}$ is a set of variables, $\mathcal{D}$ a mapping of variables to sets of values (without loss of generality, we assume $\mathcal{D}(X) \subset \mathbb{Z}$ for all $X \in \mathcal{X}$, and we denote by $min(X)$ and $max(X)$ the minimum and maximum values in $\mathcal{D}(X)$, respectively) and $\mathcal{C}$ a set of constraints that specify allowed combinations of values for subsets of variables. An assignment of a set of variables $\mathcal{X}$ is a set of pairs S such that $|\mathcal{X}| = |S|$ and for each $(X, v) \in S$, we have $X \in \mathcal{X}$ and $v \in \mathcal{D}(X)$. A constraint $C \in \mathcal{C}$ is *arc consistent* (AC) iff, when a variable in the scope of C is assigned any value, there exists an assignment of the other

variables in C such that C is satisfied. This satisfying assignment is called a *domain support* for the value. Similarly, we call a *range support* an assignment satisfying C, but where values, instead of being taken from the domain of each variable ($v \in \mathcal{D}(X)$), can be any integer between the minimum and maximum of this domain following the natural order on $\mathbb{Z}$, that is, $v \in [min(X), \ldots, max(X)]$. A constraint $C \in \mathcal{C}$ is *range consistent* (RC) iff, every value of every variable in the scope of C has a range support. A constraint $C \in \mathcal{C}$ is *bounds consistent* (BC) iff, for every variable X in the scope of C, $min(X)$ and $max(X)$ have a range support. Given a CSP $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$, we shall use the following notation throughout the paper: n shall denote the number of variables, i.e., $n = |\mathcal{X}|$; m shall denote the number of distinct unary assignments, i.e., $m = \sum_{X \in \mathcal{X}} |\mathcal{D}(X)|$; Λ shall denote the total set of values, i.e., $\Lambda = \bigcup_{X \in \mathcal{X}} \mathcal{D}(X)$; finally, λ shall denote the total number of distinct values, i.e., $\lambda = |\Lambda|$;

Soft Global Constraints. Adding a cost variable to a constraint to represent its degree of violation is now common practice in constraint programming. This model was introduced in [7]. It offers the advantage of unifying hard and soft constraints since arc consistency, along with other types of consistencies, can be applied to such constraints with no extra effort. As a consequence, classical constraint solvers can solve over-constrained problems modelled in this way without modification. This approach was applied to a number of other constraints, for instance in [9].

Two natural cost measures have been explored for the AllDifferent and for a number of other constraints. The *variable-based cost* counts how many variables need to change in order to obtain a valid assignment for the hard constraint. The *graph-based cost* counts how many times a component of a decomposition of the constraint is violated. Typically these components correspond to edges of a decomposition graph, e.g. for an AllDifferent constraint, the decomposition graph is a clique and an edge is violated if and only if both variables connected by this edge share the same value. For instance, still for the AllDifferent constraint, the following example shows two solutions involving four variables $X_1, \ldots, X_4$ each with domain $\{a, b\}$:

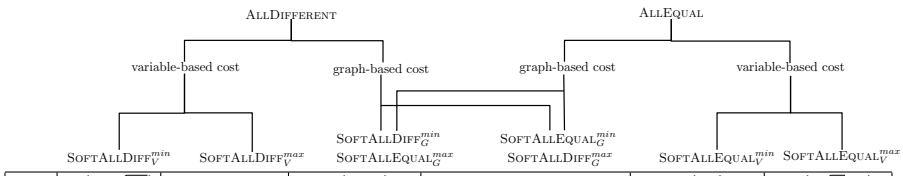
$$S_1 = \{(X_1, a), (X_2, b), (X_3, a), (X_4, b)\}$$

$$S_2 = \{(X_1, a), (X_2, b), (X_3, b), (X_4, b)\}$$

In both solutions, at least two variables must change (e.g., X_3 and X_4) to obtain a valid solution. Therefore, the variable-based cost is 2 for S_1 and S_2. However, in S_1 only two edges are violated, (X_1, X_3) and (X_2, X_4), whilst in S_2, three edges are violated, (X_2, X_3), (X_2, X_4) and (X_3, X_4). Thus, the graph-based cost of S_1 is 2 whereas it is 3 for S_2.

3 Taxonomy

In this section we introduce the taxonomy of the soft constraints related to AllDifferent and AllEqual. We consider the eight algorithmic problems



AC	$\mathcal{O}(n\sqrt{m})$	NP-hard	$\mathcal{O}(nm)$	NP-hard	$\mathcal{O}(m)$	$\mathcal{O}(\sqrt{n}m)$
BC	$\mathcal{O}(n\sqrt{m})$	$\mathcal{O}(nlog(n))$	$\mathcal{O}(nm)$	$\mathcal{O}(min(\lambda^2, n^2)nm)$	$\mathcal{O}(m)$	$\mathcal{O}(nlog(n))$

Fig. 1. Complexity of optimising difference and equality.

related to constraints of difference and equality defined by combining these two constraints, two costs (graph-based and variable-based), and two objectives (minimisation and maximisation). In fact, because the graph-based costs of AllDifferent and AllEqual are dual, only six different problems are thus defined. We close the last remaining cases: the complexity of achieving AC and BC SoftAllEqual$_V^{min}$ in this section, and that of achieving BC on SoftAllDiff$_G^{max}$ in Sections 4 and 5. Based on these results, Figure 1 can now be completed.

The next six paragraphs correspond to the six columns of Figure 1, i.e., to the twelve elements of the taxonomy. For each of them, we briefly outline the current state of the art, using the following assignment as a running example to illustrate the various costs: $S_3 = \{(X_1, a), (X_2, a), (X_3, a), (X_4, a), (X_5, b), (X_6, b), (X_7, c)\}$.

SoftAllDiff*: Variable-based cost, Minimisation.*

Definition 1 (SoftAllDiff$_V^{min}$)**.**

$$\text{SoftAllDiff}_V^{min}(\{X_1, ..X_n\}, N) \Leftrightarrow N \geq n - |\{v \mid X_i = v\}|.$$

Here the cost to minimise is the number of variables that need to be changed in order to obtain a solution satisfying an AllDifferent constraint. For instance, the cost of S_3 is 4 since three of the four variables assigned to a as well as one of the variables assigned to b must change. This objective function was first studied in [4] where the authors give an algorithm for achieving AC in $\mathcal{O}(n\sqrt{m})$. To our knowledge, no algorithm with better time complexity for the special case of bounds consistency has been proposed for this constraint.

SoftAllDiff*: Variable-based cost, Maximisation.*

Definition 2 (SoftAllDiff$_V^{max}$)**.**

$$\text{SoftAllDiff}_V^{max}(\{X_1, ..X_n\}, N) \Leftrightarrow N \leq n - |\{v \mid X_i = v\}|.$$

Here the same cost is to be maximised. In other words, we want to minimise the number of distinct values assigned to the given set of variables, since the complement of this number to n is exactly the number of variables to modify

in order to obtain a solution satisfying an AllDifferent constraint. For instance, the cost of S_3 is 4 and the number of distinct values is $7 - 4 = 3$. This constraint was studied under the name AtMostNValues in [1] where the authors introduce an algorithm in $\mathcal{O}(nlog(n))$ to achieve bc, and in [2] where the authors show that achieving ac is NP-hard since the problem is isomorphic to Min Hitting Set.

SoftAllDiff*: Graph-based cost, Minimisation &* SoftAllEqual*: Graph-based cost, Maximisation.*

Definition 3 (SoftAllDiff$^{min}_{G} \simeq$ SoftAllEqual$^{max}_{G}$)**.**

$$\text{SoftAllDiff}^{min}_{G}(\{X_1, ..X_n\}, N) \Leftrightarrow N \geq |\{\{i, j\} \mid X_i = X_j \ \& \ i \neq j\}|.$$

Here the cost to minimise is the number of violated constraints when decomposing AllDifferent into a clique of binary NotEqual constraints. For instance, the cost of S_3 is 7 since four variables share the value a (six violations) and two share the value b (one violation). Clearly, it is equivalent to maximising the number of violated binary Equal constraints in a decomposition of a global AllEqual. Indeed, these two costs are complementary to $\binom{n}{2}$ of each other (on S_3: $7 + 14 = 21$). An algorithm in $\mathcal{O}(nm)$ for achieving ac on this constraint was introduced in [8]. Again, to our knowledge there is no algorithm improving this complexity for the special case of bc.

SoftAllEqual*: Graph-based cost, Minimisation &* SoftAllDiff*: Graph-based cost Maximisation.*

Definition 4 (SoftAllEqual$^{min}_{G} \simeq$ SoftAllDiff$^{max}_{G}$)**.**

$$\text{SoftAllEqual}^{min}_{G}(\{X_1, ..X_n\}, N) \Leftrightarrow N \geq |\{\{i, j\} \mid X_i \neq X_j \ \& \ i \neq j\}|.$$

Here we consider the same two complementary costs, however we aim at optimising in the opposite way. In [3] the authors show that achieving ac on this constraint is NP-hard, however the complexity of achieving bc is left as an open question. In this paper we show that computing the optimal cost can be done in $\mathcal{O}(min(n\lambda^2, n^3))$ thus demonstrating that bc can be achieved in polynomial time.

SoftAllEqual*: Variable-based cost, Minimisation.*

Definition 5 (SoftAllEqual$^{min}_{V}$)**.**

$$\text{SoftAllEqual}^{min}_{V}(\{X_1, ..X_n\}, N) \Leftrightarrow N \geq n - max_{v \in \Lambda}(|\{i \mid X_i = v\}|)$$

Here the cost to minimise is the number of variables that need to be changed in order to obtain a solution satisfying an AllEqual constraint. For instance, the cost of S_3 is 3 since four variables already share the same value. This is equivalent to maximising the number of variables sharing a given value. Therefore this bound can be computed trivially by counting the occurrences of every

value in the domains, that is, in $\mathcal{O}(m)$. On the other hand, pruning the domains according to this bound without degrading the time complexity is not as trivial, so we show how it can be done.

Theorem 1. AC *on* SoftAllEqual$_V^{min}$ *can be achieved in* $\mathcal{O}(m)$ *steps.*

Proof. We suppose, without loss of generality, that the current upper bound on the cost is k. We first compute the number of occurrences $occ(v)$ for each value $v \in \Lambda$, which can be done in $\mathcal{O}(m)$. There are three cases to consider:

1. First, consider the case where no value appears in $n-k$ domains or more ($\forall v \in \Lambda,\ occ(v) < n-k$). In this case the constraint is violated, hence every value is inconsistent.
2. Second, consider the case where at least one value v appears in the domains of at least $n-k+1$ variables ($\exists v \in \Lambda,\ occ(v) > n-k$). In this case we can build a support for every value $w \in \mathcal{D}(X)$ by assigning all variables in $\mathcal{X} \setminus X$ with v if possible. The resulting assignment has a cost of k, hence every value is consistent.
3. Otherwise, if neither of the two cases above hold, we know that no value appears in more than $n-k$ domains, and that at least one appears $n-k$ times, let W denote the set of such values. In this case, the pair (X, v) is inconsistent iff $v \notin W$ & $W \subset \mathcal{D}(X)$.
   We first suppose that this condition does not hold and show that we can build a support. If $v \in W$ then clearly we can assign every possible variable to v and achieve a cost of k. If $W \not\subset \mathcal{D}(X)$, then we consider w such that $w \in W$ and $w \notin \mathcal{D}(X)$. By assigning every variable with w when possible we achieve a cost of k.
   Now we suppose that this condition holds and show that (X, v) does not have an AC support. Indeed once X is assigned to v the domains are such that no value appear in $n-k$ domains or more, since *every* value in W has now one less occurrence, hence we are back to Case 1.

Computing values satisfying the condition above can be done easily once the number of occurrences have been computed. In Case 3, the domain can be pruned down to the set W of values whose number of occurrences is $n-k$. □

SoftAllEqual*: Variable-based cost, Maximisation.*

Definition 6 (SoftAllEqual$_V^{max}$**).**

$$\text{SoftAllEqual}_V^{max}(\{X_1, ..X_n\}, N) \Leftrightarrow N \leq n - max_{v \in \Lambda}(|\{i \mid X_i = v\}|)$$

Here the same cost has to be maximised. In other words we want to minimise the maximum cardinality of a value. For instance, the cost of S_3 is 3, that is the complement to n of the maximum cardinality of a value ($3 = 7 - 4$). This is exactly equivalent to applying a Global Cardinality constraint (considering only the upper bounds on the cardinalities). In [5] the authors introduce an algorithm in $\mathcal{O}(\sqrt{n}m)$ and in $\mathcal{O}(nlog(n))$ for achieving AC and BC, respectively, on this constraint.

4 The Complexity of Bounds Consistency on SoftAllEqual$^{min}_{G}$

In this section we introduce an efficient algorithm that, assuming the domains are discrete intervals, computes the maximum possible number of pairs of equal values in an assignment. This algorithm allows us to close the last remaining open complexity question in Figure 1: BC on the SoftAllEqual$^{min}_{G}$ constraint. We then improve this algorithm, first by reducing the time complexity thanks to a preprocessing step, before turning it into a filtering method in Section 5.

We start by introducing additional terminology. Given two integers a and b, $a \leq b$, we say that the set of all integers x, $a \leq x \leq b$ is an *interval* and denote it by $[a, b]$. Let $\mathcal{X}$ be the set of variables of the considered CSP and assume that the domains of all the variables of $\mathcal{X}$ are sub-intervals of $[1, \lambda]$. We denote by $\mathbf{ME}(\mathcal{X})$ the set of all assignments P to the variables of $\mathcal{X}$ such that the number of pairs of equal values of P is the maximum possible. The subset of $\mathcal{X}$ containing all the variables whose domains are subsets of $[a, b]$ is denoted by $\mathcal{X}_{a,b}$. The subset of $\mathcal{X}_{a,b}$ including all the variables containing the given value c in their domains is denoted by $\mathcal{X}_{a,b,c}$. Finally the number of pairs of equal values in an element of $\mathbf{ME}(\mathcal{X}_{a,b})$ is denoted by $C_{a,b}(\mathcal{X})$ or just $C_{a,b}$ if the considered set of variables is clear from context. For notational convenience, if $b < a$, then we set $\mathcal{X}_{a,b} = \emptyset$ and $C_{a,b} = 0$. The value $C_{1,\lambda}(\mathcal{X})$ is the number of equal pairs of values in an element of $\mathbf{ME}(\mathcal{X})$.

Theorem 2. *$C_{1,\lambda}(\mathcal{X})$ can be computed in $\mathcal{O}((n + \lambda)\lambda^2)$ steps.*

Proof. The problem is solved by a dynamic programming approach: for every a, b such that $1 \leq a \leq b \leq \lambda$, we compute $C_{a,b}$. The main observation that makes it possible to use dynamic programming is the following: in every $P \in \mathbf{ME}(\mathcal{X}_{a,b})$, there is a value c ($a \leq c \leq b$) such that every variable $X \in \mathcal{X}_{a,b,c}$ is assigned value c. To see this, let value c be a value that is assigned by P to a maximum number of variables. Suppose that there is a variable X with $c \in \mathcal{D}(X)$ that is assigned by P to a different value, say c'. Suppose that c and c' appear on x and y variables, respectively. By changing the value of X from c' to c, we increase the number of equalities by $x - (y - 1) \geq 1$ (since $x \geq y$), contradicting the optimality of P.

Notice that $\mathcal{X}_{a,b} \setminus \mathcal{X}_{a,b,c}$ is the disjoint union of $\mathcal{X}_{a,c-1}$ and $\mathcal{X}_{c+1,b}$ (if $c-1 < a$ or $c+1 > b$, then the corresponding set is empty). These two sets are independent in the sense that there is no value that can appear on variables from both sets. Thus it can be assumed that $P \in \mathbf{ME}(\mathcal{X}_{a,b})$ restricted to $\mathcal{X}_{a,c-1}$ and $\mathcal{X}_{c+1,b}$ are elements of $\mathbf{ME}(\mathcal{X}_{a,c-1})$ and $\mathbf{ME}(\mathcal{X}c + 1, b)$, respectively. Taking into consideration all possible values c, we get

$$C_{a,b} = \max_{c, a \leq c \leq b} \left(\binom{|\mathcal{X}_{a,b,c}|}{2} + C_{a,c-1} + C_{c+1,b} \right). \tag{1}$$

In the first step of the algorithm, we compute $|\mathcal{X}_{a,b,c}|$ for all values of a, b, c. For each triple a, b, c, it is easy to compute $|\mathcal{X}_{a,b,c}|$ in time $\mathcal{O}(n)$, hence all these

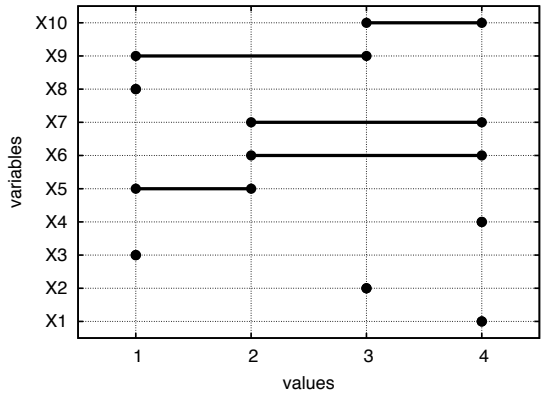
Algorithm 1: Computing $C_{1,\lambda}(\mathcal{X})$

$\forall\ 1 \leq a, b, c \leq \lambda,\ \delta_{a,b,c} \leftarrow |\mathcal{X}_{a,b,c}| \leftarrow C_{a,b} \leftarrow 0$;
foreach $k \in [0, \lambda - 1]$ **do**
  foreach $a \in [1, \lambda]$ **do**
    $b \leftarrow a + k$;
    foreach $X \in \mathcal{X}_{a,b}$ **do**
1     $\delta_{a,b,min(X)} \leftarrow \delta_{a,b,min(X)} + 1$;
2     $\delta_{a,b,max(X)+1} \leftarrow \delta_{a,b,max(X)+1} - 1$;
    foreach $c \in [a, b]$ **do**
3     $|\mathcal{X}_{a,b,c}| \leftarrow |\mathcal{X}_{a,b,c-1}| + \delta_{a,b,c}$;
4     $C_{a,b} \leftarrow max(C_{a,b}, (\binom{|\mathcal{X}_{a,b,c}|}{2} + C_{a,c-1} + C_{c+1,b}))$;
return $C_{1,\lambda}$;

values can be computed in time $\mathcal{O}(n\lambda^3)$. However, the running time can be reduced to $\mathcal{O}((n + \lambda)\lambda^2)$ as follows. For each pair a, b, we determine all the values $|\mathcal{X}_{a,b,c}|$, $a \leq c \leq b$ in time $\mathcal{O}(n + \lambda)$. More precisely, we define $\delta_{a,b,c} = |\mathcal{X}_{a,b,c}| - |\mathcal{X}_{a,b,c-1}|$ and compute $\delta_{a,b,c}$ for every $a < c \leq b$ (Alg. 1, Line 1-2). Observe that if $\mathcal{D}(X) = [a', b']$ for some $X \in \mathcal{X}_{a,b}$, then X contributes only to two of the $\delta_{a,b,c}$ values: it increases $\delta_{a,b,a'}$ by 1 and decreases $\delta_{a,b,b'+1}$ by 1. Thus by going through all variables, we can compute the $\delta_{a,b,c}$ values for a fixed a, b and for all $a \leq c \leq b$ in time $\mathcal{O}(n)$ and we can also compute $|\mathcal{X}_{a,b,a}|$ in the same time bound. Now we can compute the values $|\mathcal{X}_{a,b,c}|$, $a < c \leq b$ in time $\mathcal{O}(\lambda)$ by using the equality $|\mathcal{X}_{a,b,c}| = |\mathcal{X}_{a,b,c-1}| + \delta_{a,b,c}$ iteratively (Alg. 1, Line 3).

In the second step of the algorithm, we compute all the values $C_{a,b}$. We compute these values in increasing order of $b - a$. If $a = b$, then $C_{a,b} = \binom{|\mathcal{X}_{a,a,a}|}{2}$. Otherwise, values $C_{a,c-1}$ and $C_{c+1,b}$ are already available for every $a \leq c \leq b$, hence $C_{a,b}$ can be determined in time $\mathcal{O}(\lambda)$ using Eq. (1) (Alg. 1, Line 4). Thus all the values $C_{a,b}$ can be computed in time $\mathcal{O}(\lambda^3)$, including $C_{1,\lambda}$, which is the value of the optimum solution of the problem. Using standard techniques (storing for each $C_{a,b}$ a value c that minimises (1)), a third step of the algorithm can actually produce a variable assignment that obtains the maximum value. □

Algorithm 1 computes the largest number of equalities one can achieve by assigning a set of variables with interval domains. It can therefore be used to find an optimal solution to either SoftAllDiff$_G^{max}$ or SoftAllEqual$_G^{min}$. Notice that for the latter one needs to take the complement to $\binom{n}{2}$ in order to get the value of the violation cost. Clearly, it follows that achieving range or bounds consistency on these two constraints can be done in polynomial time, since Algorithm 1 can be used as an oracle for testing the existence of a range support. We give an example of the execution of Algorithm 1 in Figure 2. A set of ten variables, from X_1 to X_{10} are represented. Then we give the table $C_{a,b}$ for all pairs $a, b \in [1, \lambda]$.



$C_{a,b}$	$a=1$	$a=2$	$a=3$	$a=4$
$b=1$	1			
$b=2$	$\mathcal{X}_{1,2,1}+C_{2,2}=3$	0		
$b=3$	$\mathcal{X}_{1,3,1}+C_{2,3}=6$	0	0	
$b=4$	$\mathcal{X}_{1,4,1}+C_{2,4}=16$	$\mathcal{X}_{2,4,4}+C_{2,3}=6$	$\mathcal{X}_{3,4,4}+C_{3,3}=3$	1

Fig. 2. A set of intervals, and the corresponding dynamic programming table ($C_{a,b}$).

The complexity can be further reduced if $\lambda >> n$. Let $\mathcal{X}$ be a set of variables with interval domains on $[1,\lambda]$. Consider the function $occ : Q \mapsto [0..n]$, where $Q \subset \mathbb{Q}$ is a set of values of the form $a/2$ for some $a \in \mathbb{Z}$, such that $min(Q) = 1$ and $max(Q) = \lambda$. Intuitively, the value of $occ(a)$ is the number of variables whose domain interval encloses the value a, more formally:

$$\forall a \in Q,\ occ(a) = |\{X \mid X \in \mathcal{X}, min(X) \leq a \leq max(X)\}|.$$

Such an occurrence function, along with the corresponding set of intervals, is depicted in Figure 3. The *crest* of the function *occ* is an interval $[a,b] \in Q$ such that for some $c \in [a,b]$, *occ* is monotonically increasing on $[a,c]$ and monotonically decreasing on $[c,b]$. For instance, on the set intervals represented in Figure 3, $[1,15]$ is a crest since it is monotonically increasing on $[1,12]$ and monotonically decreasing on $[12,15]$.

Let $\mathcal{I}$ be a partition of $[1,\lambda]$ into a set of intervals such that every element of $\mathcal{I}$ is a crest. For instance, $\mathcal{I} = \{[1,15],[16,20],[21,29],[30,42]\}$ is such a partition for the set of intervals shown in Figure 3. We denote by $R_{\mathcal{I}}(\mathcal{X})$ the reduction of $\mathcal{X}$ by the partition $\mathcal{I}$. The reduction has as many variables as $\mathcal{X}$ (2) but the domains are replaced with the set of intervals in $\mathcal{I}$ that overlap with the corresponding variable in $\mathcal{X}$ (3).

$$R_{\mathcal{I}}(\mathcal{X}) = \{X' \mid X \in \mathcal{X}\} \tag{2}$$

$$\forall X \in \mathcal{X},\ \mathcal{D}(X') = \{I \mid I \in \mathcal{I}\ \&\ \mathcal{D}(X) \cap I \neq \emptyset\} \tag{3}$$

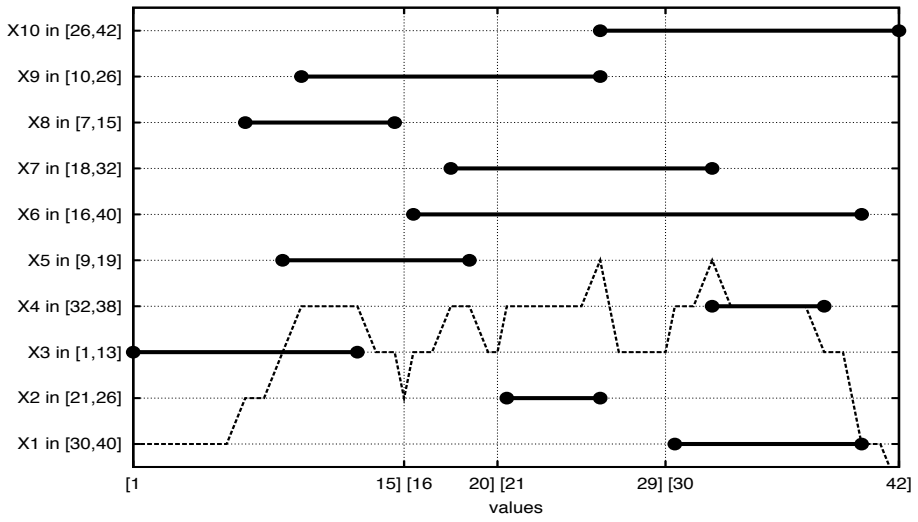


Fig. 3. Some intervals and the corresponding *occ* function.

For instance, the set of intervals depicted in Figure 3 can be reduced to the set shown in Figure 2, where each element in $\mathcal{I}$ is mapped to an integer in $[1, 4]$.

Theorem 3. *If $\mathcal{I}$ is a partition of $[1, \lambda]$ such that every element of $\mathcal{I}$ is a crest of occ, then $\mathbf{ME}(\mathcal{X}) = \mathbf{ME}(R_{\mathcal{I}}(\mathcal{X}))$.*

Proof. First, we show that for any optimal solution $s \in \mathbf{ME}(\mathcal{X})$, we can produce a solution $s' \in \mathbf{ME}(R_{\mathcal{I}}(\mathcal{X}))$ that has the same number of equalities as s. Indeed, for any value a, consider every variable X assigned to this value, that is, such that $s[X] = a$. Let $I \in \mathcal{I}$ be the crest containing a; by definition we have $I \in \mathcal{D}(X')$. Therefore we can assign all these variables to the same value I.

Now we show the opposite, that is, given a solution to the reduced problem, one can build a solution to the original problem with the same cost. The key observation is that, for a given crest $[a, b]$, all intervals overlapping with $[a, b]$ have a common value. Indeed, suppose that this is not the case, that is, there exists $[c_1, d_1]$ and $[c_2, d_2]$ both overlapping with $[a, b]$ such that $d_1 < c_2$. Then $occ(d_1) > occ(d_1 + \frac{1}{2})$ and similarly $occ(c_2 - \frac{1}{2}) < occ(c_2)$. However, since $a \leq d_1 < c_2 \leq b$, $[a, b]$ would not satisfy the conditions for being a crest, hence a contradiction. Therefore, for a given crest I, and for every variable X' such that $s'[X'] = I$, we can assign X to this common value. □

We show that this transformation can be achieved in $\mathcal{O}(nlog(n))$ steps. Observe that $\delta_{1,a,\lambda}$, if it was defined on Q rather than $[1, \lambda]$, would in fact be the

Algorithm 2: Computing a partition into crests $\mathcal{I}$

$\delta \leftarrow \emptyset$;
1 **foreach** $X \in \mathcal{X}$ **do**
  if $\exists (min(X), k) \in \delta$ **then**
    replace $(min(X), k)$ with $(min(X), k+1)$ in δ;
  else
    add $(min(X), 1)$ to δ;
  if $\exists (max(X)+1, k) \in \delta$ **then**
    replace $(max(X)+\frac{1}{2}, k)$ with $(max(X)+\frac{1}{2}, k-1)$ in δ;
  else
    add $(max(X)+\frac{1}{2}, -1)$ to δ;
sort δ by increasing first element;
$\mathcal{I} \leftarrow \emptyset$;
$min \leftarrow max \leftarrow 1$;
2 **while** $\delta \neq \emptyset$ **do**
  $polarity \leftarrow pos$;
  $k = 1$;
  repeat
    pick and remove the first element (a, k) of δ;
    $max \leftarrow round(a) - 1$;
    if $polarity = pos$ & $k < 0$ **then** $polarity \leftarrow neg$;
  until $polarity = pos$ *or* $k < 0$;
  add $[min, max]$ to $\mathcal{I}$;
  $min \leftarrow max + 1$;

derivative of *occ*. Moreover, we can compute it in $\mathcal{O}(nlog(n))$ steps as shown in Algorithm 2. We first compute the non-null values of $\delta_{1,a,\lambda}$ by looping through each variable $X \in \mathcal{X}$ (Line 1). Then we sort them, and finally we create the partition into crests by going through the derivative once and identifying the inflection points. Clearly, the number of elements in δ is bounded by $2n$. Therefore, the complexity of Algorithm 2 is dominated by the sorting of its elements, hence the $\mathcal{O}(nlog(n))$ worst-case time complexity.

Therefore, we can replace every crest by a single value as a preprocessing step and then run Algorithm 1. Moreover, since the number of crests is bounded by n, we obtain the following theorem, where n stands for the number of variables, λ for the number of distinct values, and m for the sum of all domain sizes.

Theorem 4. RC *on* SoftAllEqual$^{min}_{G}$ *can be achieved in* $\mathcal{O}(min(\lambda^2, n^2)nm)$ *steps.*

Proof. If $\lambda \leq n$ then one can achieve range consistency by iteratively calling Algorithm 1 after assigning each of the $\mathcal{O}(m)$ unit assignments $((X, v)\ \forall X \in \mathcal{X}, v \in \mathcal{D}(X))$. The resulting complexity is $\mathcal{O}(n\lambda^2)m$ (the term λ^3 is absorbed by $n\lambda^2$ due to $\lambda \leq n$). Otherwise, we apply the above $\mathcal{O}(nlog(n))$ procedure and similarly achieve range consistency after that. Since after the reformulation $\lambda = \mathcal{O}(n)$, the resulting complexity is $\mathcal{O}(n^3m)$. □

5 A Filtering Method for SoftAllEqual$^{min}_{G}$

In this section, we show that in the particular case where the cost variable N to minimise is such that $max(N) = (\binom{n}{2} - C_{1,\lambda})$, RC can be achieved in the same time complexity as that for computing $C_{1,\lambda}$. In other words, once the current lower bound, computed by Algorithm 1, exactly matches the required value $max(N)$, we can compute and prune range inconsistent values at no extra computational cost. This is an important particular case. Indeed, on the one hand, if $max(N)$ is not as high as this lower bound, the constraint should fail hence no pruning is required. On the other hand, if $max(N)$ is strictly larger than this lower bound, it is less likely that pruning should occur since the constraint is less tight.

We show how one can compute all the values participating in an optimal solution in $\mathcal{O}(min(n\lambda^2, \lambda^3))$ steps. When the cost $(\binom{n}{2} - C_{1,\lambda})$ of an optimal solution matches $max(N)$ all values either participate in an optimal solution or in no solution at all.

In the first step we run Algorithm 1, hence in the rest of the section we assume that all values $C_{a,b}(\mathcal{X})$ and $|X_{a,b,c}|$ are known (i.e. can be computed in a constant time). Moreover, we introduce the following additinal notation:

- Let $V_{a,b}(\mathcal{X})$ be the set of all values c such that $C_{a,b}(\mathcal{X}) = \binom{|\mathcal{X}_{a,b,c}|}{2} + C_{a,c-1}(\mathcal{X}) + C_{c+1,b}(\mathcal{X})$. In other words $c \in V_{a,b}(\mathcal{X})$ if there is an optimal assignment to the set of variables whose domains are subintervals of $[a, b]$, where each variable containing c in its domain is assigned with c. By the above assumption, given an interval $[a, b]$ and $c \in [a, b]$, it is possible to check in $\mathcal{O}(1)$ whether $c \in V_{a,b}(\mathcal{X})$.
- Let $H(\mathcal{X})$ be the *descendance* graph naturally defined by V, that is, whose set of vertices are all sub-intervals $[a, b]$ of $[1, n]$ and there is an arc $([a, b], [c, d])$ if and only if $a = c$ & $(d+1) \in V_{a,b}(\mathcal{X})$ or $d = b$ & $(c-1) \in V_{a,b}(\mathcal{X})$. In other words, given an interval $[a, b]$, any value $c \in V_{a,b}(\mathcal{X})$ such that $a < c < b$ defines two 'children' of $[a, b]$: one is $[a, c-1]$, the other is $[c+1, d]$. If $c = a$ or $c = d$ then there is only one child, namely $[c+1, d]$ and $[a, c-1]$, respectively (of course, if $a = b$ then no other interval is a child of $[a, b]$).
- We say that $[c, d]$ is a *descendant* of $[a, b]$ if $[c, d] = [a, b]$ or $H(\mathcal{X})$ has a path from $[a, b]$ to $[c, d]$. Since computing whether $c \in V_{a,b}(\mathcal{X})$ can be done in $\mathcal{O}(1)$, it is possible to compute in $\mathcal{O}(\lambda)$ the set of arcs leaving $[a, b]$. Therefore, by applying a DFS- or BFS-like method, it is possible to compute in $\mathcal{O}(\lambda^3)$ the set of all descendants of $[1, \lambda]$.
- We say that the interval $[a, b]$ is a *witness* of (X, c) if and only if $a \leq min(X) \leq c \leq max(X) \leq b$, $[a, b]$ is a descendant of $[1, \lambda]$ and $c \in V_{a,b}(\mathcal{X})$.

The intuition behind the following proof is that for an assignment (X, c) to belong to an optimal solution $P \in \mathbf{ME}(\mathcal{X})$, two conditions need to hold. First, the value c must be involved in an optimal solution, that is, it must belong to some set $V_{a,b}(\mathcal{X})$ such that $[a, b]$ is a descendant of $[1, \lambda]$. Second, there must exist at least one variable $X \in \mathcal{X}$ whose domain is included in $[a, b]$ and such

that $c \in \mathcal{D}(X)$. The notion of witness defined above encapsulates these two conditions, we therefore look for a witness $[a, b]$ for (X, c).

We shall proceed by induction: if c belongs to $V_{1,\lambda}(\mathcal{X})$, then $[1, \lambda]$ is a witness. Otherwise, there is some value $d \in V_{1,\lambda}(\mathcal{X})$ such that either $c \in [a, d-1]$ or in $c \in [d+1, b]$. Moreover, (X, c) belongs to the optimal assignment of the variables whose domains are subsets of one of these intervals. We show in the proof that, by proceeding with such an inductive descent, we will eventually encounter a descendant $[a', b']$ of $[1, \lambda]$ such that $c \in V_{a',b'}(\mathcal{X})$ and this interval $[a', b']$ will be the desired witness of (X, c).

The proof of the opposite direction will require a more careful consideration of the descendance relation defined in the list above. In particular, we will notice that if $[a', b']$ is a descendant of $[1, \lambda]$ then any optimal assignment P to the variables whose domains are subsets of $[a', b']$ is a subset of an optimal assignment of $\mathcal{X}$. It will follow that if (X, c) participates in such an assignment P then it participates in a globally optimal assignment as well.

Lemma 1. *(X, c) belongs to an assignment $P \in \mathbf{ME}(\mathcal{X})$ if and only if (X, c) has a witness.*

Proof. Let $[a, b]$ be the domain of X and let $[a', b']$ be a witness of $[a, b]$. Observe first that since $[a', b']$ is a descendant of $[1, \lambda]$, any element $P' \in \mathbf{ME}(\mathcal{X}_{a,b})$ is a subset of an element of $\mathbf{ME}(\mathcal{X})$. This is clear if $[a', b'] = [1, \lambda]$. Assume that $([1, \lambda], [a', b'])$ is an arc of $H(\mathcal{X})$ and assume without loss of generality that $a' = 1$ and $(b'+1) \in V_{1,\lambda}(\mathcal{X})$. That is, $C_{1,\lambda}(\mathcal{X}) = \binom{|\mathcal{X}_{1,\lambda_{b'+1}}|}{2} + C_{a',b'}(\mathcal{X}) + C_{b'+2,\lambda}(\mathcal{X})$. That is, any assignment to $\mathcal{X}_{a',b'}$ that results in $C_{a',b'}(\mathcal{X})$ of equal pairs of values (in other words, any assignment of $\mathbf{ME}(\mathcal{X}_{a',b'})$) can be extended to an assignment of $\mathbf{ME}(\mathcal{X})$. If the distance in $H(\mathcal{X})$ between $[1, \lambda]$ and $[a', b']$ is greater than 1 the observation can be proved by induction applying the argument for distance 1 along the shortest path from $[1, \mathcal{X}]$ to $[a, b]$.

Since $c \in V_{a',b'}(\mathcal{X})$ there is $P' \in \mathbf{ME}(\mathcal{X}_{a',b'})$ where all the variables having c in their domains are assigned with c. Since $[a, b] \subseteq [a', b']$, $(X, c) \in P'$. According to the previous paragraph P' is a subset of assignment $P \in \mathbf{ME}(\mathcal{X})$. Consequently, $(X, c) \in P$ as required.

Conversely, assume that $(X, c) \in P \in \mathbf{ME}(\mathcal{X})$. Observe that in this case either $c \in V_{1,\lambda}(\mathcal{X})$ or there is an interval $[a^*, b^*]$ such that $([1, \lambda], [a^*, b^*])$ is an arc in $H(\mathcal{X})$ and $[a, b] \subseteq [a^*, b^*]$. Indeed assume that $c \notin V_{1,\lambda}(\mathcal{X})$ and let $P \in \mathbf{ME}(\mathcal{X})$ such that $(X, c) \in P$. As has been observed in Theorem 2, there is a value d such that for any variable X' having d in its domain, $(X', d) \in P$. It follows that $d \in V_{1,\lambda}(\mathcal{X})$ and hence $c \neq d$. Then either $[a, b] \subseteq [1, d-1]$ (i.e. $[a^*, b^*] = [1, d-1]$) or $[a, b] \subseteq [d+1, \lambda]$ (i.e. $[a^*, b^* = [d+1, \lambda]$) because otherwise X contains d in its domain and hence $(X, d) \in P$ in contradiction to our assumption that $(X, c) \in P$.

We proceed by induction on the difference between $\lambda - 1$ and $b - a$. If the difference is 0 then, since $[a, b] \subseteq [1, \lambda]$, it follows that $[a, b] = [1, \lambda]$. Thus $[a, b]$ cannot be a proper sub-interval of $[1, \lambda]$ and hence, according to the previous paragraph, $c \in V_{1,\lambda}(\mathcal{X})$. Clearly, $[1, \lambda]$ is a witness of $[a, b]$. Assume now that

$(\lambda - 1) - (b - a) > 0$. If $c \in V_{1,\lambda}(\mathcal{X})$ then $[1, \lambda]$ is a witness of $[a, b]$. Otherwise, let $[a^*, b^*]$ be as in the previous paragraph. By the induction assumption, (X, c) has a witness $[a', b']$ with respect to $\mathcal{X}_{a^*,b^*}$. In other words, $[a, b] \subseteq [a', b']$, $c \in V_{a',b'}(\mathcal{X}_{a^*,b^*})$ and $[a', b']$ is a descendant of $[a^*, b^*]$ in $H(\mathcal{X}_{a^*,b^*})$. Since $\mathcal{X}_{a',b'} \subseteq \mathcal{X}_{a^*,b^*}$,$c \in V_{a',b'}(\mathcal{X})$. It is also not hard to observe that $H(\mathcal{X}_{a^*,b^*})$ is a sub-graph of $H(\mathcal{X})$. It follows that $[a', b']$ is a witness of (X, c) with respect to $\mathcal{X}$. □

Lemma 1 allows us to achieve RC in the following way. For each value (X, c) such that $\mathcal{D}(X) = [a, b]$, check for all super-intervals $[a', b']$ whether $[a', b']$ is a witness of $[a, b]$. Since there are $\mathcal{O}(n\lambda)$ possible values, $\mathcal{O}(\lambda^2)$ super-intervals of the given interval and the witness relation can be checked in $\mathcal{O}(1)$, bounds consistency can be achieved in $\mathcal{O}(n\lambda^3)$. This runtime can be further reduced if before exploring the values we compute an auxiliary Boolean three-dimensional array $MarkedValues$ using the following procedure. Order the sub-intervals $[a, b]$ of $[1, \lambda]$ by decreasing difference $b-a$ and explore them according to this order. For the given interval $[a, b]$, explore all values $c \in [a, b]$ and set $MarkedValues[a][b][c]$ to 1 if and only if the one of the following conditions is true:

1. $[a, b]$ is a descendant of $[1, \lambda]$ in $H(\mathcal{X})$ and $c \in V_{a,b}(\mathcal{X})$;
2. $MarkedValues[a - 1][b][c] = 1$ (only if $a > 1$);
3. $MarkedValues[a][b + 1][c] = 1$ (only if $b < \lambda$).

If none of the above conditions are satisfied then $MarkedValues[a][b][c]$ is set to 0. Clearly, computing the $MarkedValues$ array takes $\mathcal{O}(\lambda^3)$. Having completed the above procedure, the following lemma holds.

Lemma 2. *Let $X \in \mathcal{X}$ be a variable with domain $[a, b]$. Then for any $c \in [a, b]$, (X, c) belongs to an assignment $P \in \mathbf{ME}(\mathcal{X})$ iff $MarkedValues[a][b][c] = 1$.*

Proof. Assume that $MarkedValues[a][b][c] = 1$. If the first condition among the above three is satisfied then $[a, b]$ is a witness of (X, c). Otherwise, let $[a', b']$ be the super-interval of $[a, b]$ that caused $MarkedValues[a][b][c]$ to be set to 1. If the first condition is satisfied with respect to $[a', b']$ then $[a, b]$ is a witness of (X, c). Otherwise there is a super-interval $[a'', b'']$ that caused $MarkedValues[a'][b'][c]$ to be set to 1. Proceeding to argue in this way we explore a sequence of intervals such that every next element in this sequence is a strict super-interval of its predecessor. Clearly this sequence is of finite length and $MarkedValues[a^*][b^*][c]$ for the last element $[a^*, b^*]$ of this sequence is set according to Condition 1. Hence, $[a^*, b^*]$ is a witness of (X, c). Thus if $MarkedValues[a][b][c] = 1$ then (X, c) has a witness and (X, c) belongs to an assignment $P \in \mathbf{ME}(\mathcal{X})$ according to Lemma 1.

Conversely assume that (X, c) belongs to an assignment $P \in \mathbf{ME}(\mathcal{X})$. Let $[a', b']$ be a witness of (X, c) existing according to Lemma 1. Then, by Condition 1, $MarkedValues[a'][b'][c] = 1$. It is not hard to show that $MarkedValues[a][b][c]$ is set to 1 by the inductive application of Conditions 2 and 3. □

Theorem 5. *If the minimum number of allowed equal pairs of values is $C_{a,b}(\mathcal{X})$,* RC *on* $\textsc{SoftAllEqual}_G^{min}$ *can be achieved in $\mathcal{O}((n + \lambda)\lambda^2)$ steps.*

Proof. Achieving BC in this case can be done by filtering all values that do not belong to an assignment $P \in \mathbf{ME}(\mathcal{X})$. By Lemma 2, this can be done by exploring all values (X, c) and for each of them checking in $\mathcal{O}(1)$ whether $MarkedValues[a][b][c] = 1$ where $[a, b]$ is the domain of X. Since the $MarkedValues$ array can be computed in $\mathcal{O}((n+\lambda)\lambda^2)$ (the computation includes all the previously discussed computational steps) and there are $\mathcal{O}(n\lambda)$ values, the theorem follows. □

6 Conclusion

Constraints for reasoning about the number of different assignments to a set of variables are ubiquitous in constraint programming and artificial intelligence. In this paper we considered the global constraints AllDifferent and AllEqual, and their optimisation variants, SoftAllDiff and SoftAllEqual, respectively. A major technical contribution of the paper is an efficient algorithm for optimising the cost of the SoftAllEqual constraint when the objective is to maximise the number of equalities achieved in the decomposition graph of the constraint. Therefore, we give a complete characterisation of these constraints. This paper can be regarded as providing a complete taxonomy of constraints of difference and equality.

References

1. Nicolas Beldiceanu. Pruning for the minimum constraint family and for the number of distinct values constraint family. In *CP*, pages 211–224, 2001.
2. Christian Bessière, Emmanuel Hebrard, Brahim Hnich, Zeynep Kiziltan, and Toby Walsh. Filtering algorithms for the nvalueconstraint. *Constraints*, 11(4):271–293, 2006.
3. Emmanuel Hebrard, Barry O'Sullivan, and Igor Razgon. A soft constraint of equality: Complexity and approximability. In *CP*, pages 358–371, 2008.
4. Thierry Petit, Jean-Charles Régin, and Christian Bessière. Specific filtering algorithms for over-constrained problems. In *CP*, pages 451–463, 2001.
5. Claude-Guy Quimper, Alejandro López-Ortiz, Peter van Beek, and Alexander Golynski. Improved algorithms for the global cardinality constraint. In *CP*, pages 542–556, 2004.
6. Jean-Charles Régin. A filtering algorithm for constraints of difference in csps. In *AAAI*, pages 362–367, 1994.
7. Jean-Charles Régin, Thierry Petit, Christian Bessière, and Jean-Francois Puget. An original constraint based approach for solving over constrained problems. In *CP*, pages 543–548, 2000.
8. Willem Jan van Hoeve. A hyper-arc consistency algorithm for the soft alldifferent constraint. In *CP*, pages 679–689, 2004.
9. Willem Jan van Hoeve, Gilles Pesant, and Louis-Martin Rousseau. On global warming: Flow-based soft global constraints. *J. Heuristics*, 12(4-5):347–373, 2006.