Breaking Symmetry of Interchangeable Variables and Values*

Y.C. Law[1], J.H.M. Lee[1], Toby Walsh[2], and J.Y.K. Yip[1]

[1] Deparment of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
{yclaw,jlee,ykyip}@cse.cuhk.edu.hk
[2] National ICT Australia and School of CSE, University of New South Wales, Sydney, Australia
tw@cse.unsw.edu.au

Abstract. A common type of symmetry is when both variables and values partition into interchangeable sets. Polynomial methods have been introduced to eliminate all symmetric solutions introduced by such interchangeability. Unfortunately, whilst eliminating all symmetric solutions is tractable in this case, pruning all symmetric values is NP-hard. We introduce a new global constraint called SIGLEX and its GAC propagator for pruning some (but not necessarily all) symmetric values. We also investigate how different postings of the SIGLEX constraints affect the pruning performance during constraint solving. Finally, we test these static symmetry breaking constraints experimentally for the first time.

1 Introduction

When solving complex real-life problems like staff rostering, symmetry may dramatically increase the size of the search space. A simple and effective mechanism to deal with symmetry is to add static symmetry breaking constraints to eliminate symmetric solutions [1–4]. Alternatively, we can modify the search procedure so that symmetric branches are not explored [5–7]. Unfortunately, eliminating all symmetric solutions is NP-hard in general. In addition, even when all symmetric solutions can be eliminated in polynomial time, pruning all symmetric values may be NP-hard [8]. One way around this problem is to develop polynomial methods for special classes of symmetries.

One common type of symmetry is when variables and/or values are interchangeable. For instance, in a graph colouring problem, if we assign colours (values) to nodes (variables), then the colours (values) are fully interchangeable. That is, we can permute the colours throughout a solution and still have a proper colouring. Similarly, variables may be interchangeable. For example, if two nodes (variables) have the same set of neighbours, we can permute them and keep a proper colouring. We call this *variable and value interchangeability*. It has also been called *piecewise symmetry* [9] and *structural symmetry* [10]. Recent results show that we can eliminate all symmetric solutions due

* We thank the anonymous referees for their constructive comments. The work described in this paper was substantially supported by grants (Project no. CUHK4131/05 and CUHK4219/04E) from the Research Grants Council of the Hong Kong SAR. NICTA is funded by DCITA and ARC through Backing Australia's Ability and the ICT Centre of Excellence program.

to variable and value interchangeability in polynomial time. Sellmann and Van Hentenryck gave a polynomial time dominance detection algorithm for dynamically breaking such symmetry [10]. Subsequently, Flener, Pearson, Sellmann and Van Hentenryck identified a set of static symmetry breaking constraints to eliminate all symmetric solutions [9]. In this paper, we propose using a linear number of the new SIGLEX constraint for breaking such symmetry. A SIGLEX constraint orders the interchangeable variables as well as the interchangeable values. Its propagator is based on a decomposition using REGULAR constraints [11].

2 Background

A *constraint satisfaction problem* (CSP) consists of a set of n *variables*, each with a finite *domain* of possible values, and a set of *constraints* specifying allowed combinations of values for given subsets of variables. A constraint restricts values taken by some subset of variables to a subset of the Cartesian product of the variable domains. Without loss of generality, we assume that variables initially share the same domain of m possible values, d_1 to d_m. Each finite domain variable takes one value from this domain. We also assume an ordering on values in which $d_i < d_j$ iff $i < j$. A *solution* is an assignment of values to variables satisfying the constraints.

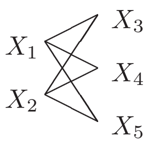
A global constraint has a parameterised number of variables. We will use four common global constraints. The first, $\text{AMONG}([X_1, .., X_n], v, M)$, holds iff $|\{i \mid X_i \in v\}| = M$. That is, M of the variables from X_1 to X_n take values among the set v. Combining together multiple AMONG constraints gives the global cardinality constraint [12]. $\text{GCC}([X_1, .., X_n], [d_1, .., d_m], [O_1, .., O_m])$ holds iff $|\{i \mid X_i = d_j\}| = O_j$ for $1 \leq j \leq m$. That is, O_j of the variables from X_1 to X_n take the value d_j. If $O_j \leq 1$ for all j then no value occurs more than once and we have an all different constraint. $\text{ALLDIFF}([X_1, \ldots, X_n])$ holds iff $X_i \neq X_j$ for $1 \leq i < j \leq n$. Finally, a global constraint that we will use to encode other global constraints is the REGULAR constraint [11]. Let $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ denote a *deterministic finite automaton* (DFA) where Q is a finite set of states, Σ an alphabet, $\delta : Q \times \Sigma \rightarrow Q$ a partial transition function, q_0 the initial state and $F \subseteq Q$ the set of final states. $\text{REGULAR}([X_1, \ldots, X_n], \mathcal{M})$ holds iff the string $[X_1, \ldots, X_n]$ belongs to the regular language recognised by $\mathcal{M}$. Quimper and Walsh encode a linear time GAC propagator for the REGULAR constraint using ternary constraints [13]. They introduce variables for the state of the DFA after each character, and post ternary constraints ensuring that the state changes according to the transition relation. One advantage of this encoding is that we have easy access to the states of the DFA. In fact, we will need here to link the final state to a finite domain variable.

Systematic search constraint solvers typically explore partial assignments using backtracking search, enforcing a local consistency at each search node to prune values for variables which cannot be in any solution. We consider a well known local consistency called generalized arc consistency. Given a constraint C on finite domain variables, a *support* is an assignment to each variable of a value in its domain which satisfies C. A constraint C on finite domain variables is *generalized arc consistent* (GAC) iff for each variable, every value in its domain belongs to a support.

3 Variable and value interchangeability

We suppose that there is a partition of the n finite domain variables of our CSP into a disjoint sets, and the variables within each set are interchangeable. That is, if we have a solution $\{X_i = d_{sol(i)} \mid 1 \leq i \leq n\}$ and any bijection σ on the variable indices which permutes indices within each partition, then $\{X_{\sigma(i)} = d_{sol(i)} \mid 1 \leq i \leq n\}$ is also a solution. We also suppose that there is a partition of the m domain values into b disjoint sets, and the values within each set are interchangeable. That is, if we have a solution $\{X_i = d_{sol(i)} \mid 1 \leq i \leq n\}$ and any bijection σ on the value indices which permutes indices within each partition, then $\{X_i = d_{\sigma(sol(i))} \mid 1 \leq i \leq n\}$ is also a solution. If $n = a$ we have just interchangeable values, whilst if $m = b$ we have just interchangeable variables. We will order variable indices so that $X_{p(i)}$ to $X_{p(i+1)-1}$ is the ith variable partition, and value indices so that $d_{q(j)}$ to $d_{q(j+1)-1}$ is the jth value partition where $1 \leq i \leq a$, $1 \leq j \leq b$. In other words, $p(i)$ and $q(j)$ give the starting indices of the ith variable partition and the jth value partition respectively.

Example 1 *Consider a CSP problem representing 3-colouring the following graph:*



Nodes are labelled with the variables X_1 to X_5. Values correspond to colours. X_1 and X_2 are interchangeable as the corresponding nodes have the same set of neighbours. If we have a proper colouring, we can permute the values assigned to X_1 and X_2 and still have a proper colouring. Similarly, X_3, X_4 and X_5 are interchangeable. The variables thus partition into two disjoint sets: $\{X_1, X_2\}$ and $\{X_3, X_4, X_5\}$. In addition, we can uniformly permute the colours throughout a solution and still have a proper colouring. Thus, the values partition into a single set: $\{d_1, d_2, d_3\}$. In graph colouring, variable partitions can be identified by checking whether two nodes have the same set of neighbours, while in general problems, the underlying symmetry can be discovered automatically [14].

Flener *et al.* [9] show that we can eliminate all solutions which are symmetric due to variable and value interchangeability by posting the following constraints:

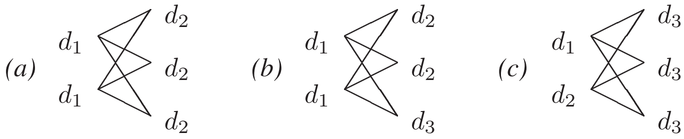
$$X_{p(i)} \leq .. \leq X_{p(i+1)-1} \qquad \forall\, i \in [1, a] \qquad (1)$$

$$\textsc{Gcc}([X_{p(i)}, .., X_{p(i+1)-1}], [d_1, .., d_m], [O^i_1, .., O^i_m]) \qquad \forall\, i \in [1, a] \qquad (2)$$

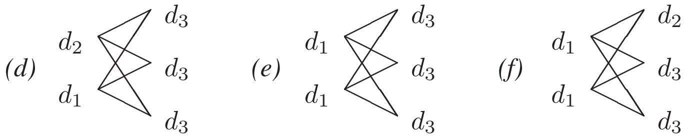
$$(O^1_{q(j)}, .., O^a_{q(j)}) \geq_{\text{lex}} .. \geq_{\text{lex}} (O^1_{q(j+1)-1}, .., O^a_{q(j+1)-1}) \quad \forall\, j \in [1, b] \qquad (3)$$

$(O^1_k, .., O^a_k)$ is called the *signature* of the value d_k, which gives the number of occurrences of the value d_k in each variable partition. Note that the signature is invariant to the permutation of variables within each equivalence class. By ordering variables within each equivalence class using (1), we rule out permuting interchangeable variables. Similarly, by lexicographically ordering the signatures of values within each equivalence class using (3), we rule out permuting interchangeable values.

Example 2 *Consider again the 3-colouring problem in Example 1. There are 30 proper colourings of this graph. When we post the above symmetry breaking constraints, the number of proper colourings reduces from 30 to just 3:*



Each colouring is representative of a different equivalence class. In fact, it is the lexicographically least member of its equivalence class. On the other hand, the following colourings are eliminated by the above symmetry breaking constraints:



For instance, the proper colouring given in (e) is symmetric to that in (a) since if we permute d_2 with d_3 in (e), we get (a). The proper colouring given in (e) is eliminated by the symmetry breaking constraint $(O_2^1, O_2^2) \geq_{\text{lex}} (O_3^1, O_3^2)$ since $O_2^1 = O_3^1 = 0$ (neither d_2 nor d_3 occur in the first equivalence class of variables) but $O_2^2 = 0$ and $O_3^2 = 3$ (d_2 does not occur in the second equivalence class of variables but d_3 occurs three times).

Suppose BREAKINTERCHANGEABILITY$(p, q, [X_1, .., X_n])$ is a global constraint that eliminates all symmetric solutions introduced by interchangeable variables and values. That is, BREAKINTERCHANGEABILITY orders the variables within each equivalence class, as well as lexicographically orders the signatures of values within each equivalence class. It can be seen as the conjunction of the ordering, GCC and lexicographic ordering constraints given in Equations (1), (2) and (3). Enforcing GAC on such a global constraint will prune all symmetric values due to variable and value interchangeability. Not surprisingly, decomposing this global constraint into separate ordering, GCC and lexicographic ordering constraints may hinder propagation.

Example 3 *Consider again the 3-colouring problem in Example 1. Suppose X_1 to X_5 have domains $\{d_1, d_2, d_3\}$, the signature variables O_1^1, O_2^1, O_3^1 have domains $\{0, 1, 2\}$, whilst O_1^2, O_2^2, O_3^2 have domains $\{0, 1, 2, 3\}$. Flener et al.'s decomposition and the binary not-equals constraints between variables representing neighbouring nodes are GAC. However, by considering (a), (b) and (c), we see that GAC on* BREAKINTERCHANGEABILITY *and the binary not-equals constraints ensures $X_1 = d_1$, $X_2 \neq d_3$, $X_3 \neq d_1$, $X_4 \neq d_1$ and $X_5 \neq d_1$.*

As decomposing BREAKINTERCHANGEABILITY hinders propagation, we might consider a specialised algorithm for achieving GAC that prunes all possible symmetric values. Unfortunately enforcing GAC on such a global constraint is NP-hard [8].

4 A new decomposition

We propose an alternative decomposition of BREAKINTERCHANGEABILITY. This decomposition does not need global cardinality constraints which are expensive to propagate. In fact, Flener *et al.*'s decomposition requires a propagator for GCC which prunes the bounds on the number of occurrence of values. The decomposition proposed here uses just REGULAR constraints which are available in many solvers or can be easily added using simple ternary transition constraints [13]. This new decomposition can be efficiently and incrementally propagated.

The results in Table 5 of [15] suggest that propagation is rarely hindered by decomposing a chain of lexicographic ordering constraints into individual lexicographic ordering constraints between neighbouring vectors. Results in Table 1 of [16] also suggest that propagation is rarely hindered by decomposing symmetry breaking constraints for interchangeable values into symmetry breaking constraints between neighbouring pairs of values in each equivalence class. We therefore propose a decomposition which only considers the signatures of neighbouring pairs of values in each equivalence class.

This decomposition replaces BREAKINTERCHANGEABILITY by a linear number of symmetry breaking constraints, SIGLEX. These lexicographically order the signatures of *neighbouring* pairs of values in each equivalence class, as well as order variables within each equivalence class. We decompose BREAKINTERCHANGEABILITY into SIGLEX$(k, p, [X_1, .., X_n])$ for $q(j) \leq k < q(j+1) - 1$, $1 \leq j \leq b$. The global constraint SIGLEX$(k, p, [X_1, .., X_n])$ itself holds iff:

$$X_{p(i)} \leq .. \leq X_{p(i+1)-1} \qquad \forall\, i \in [1, a] \tag{4}$$

$$\text{AMONG}([X_{p(i)}, .., X_{p(i+1)-1}], \{d_k\}, O_k^i) \qquad \forall\, i \in [1, a] \tag{5}$$

$$\text{AMONG}([X_{p(i)}, .., X_{p(i+1)-1}], \{d_{k+1}\}, O_{k+1}^i) \quad \forall\, i \in [1, a] \tag{6}$$

$$(O_k^1, .., O_k^a) \geq_{\text{lex}} (O_{k+1}^1, .., O_{k+1}^a) \tag{7}$$

SIGLEX orders the variables within each equivalence class and lexicographically orders the signatures of two interchangeable and neighbouring values. To propagate each SIGLEX constraint, we give a decomposition using REGULAR constraints which does not hinder propagation.

Theorem 1 *GAC can be enforced on* SIGLEX$(k, p, [X_1, .., X_n])$ *in* $O(n^2)$ *time.*

Proof: We first enforce the ordering constraints $X_{p(i)} \leq .. \leq X_{p(i+1)-1}$ on each variable partition. We then channel into a sequence of four valued variables using the constraints: $Y_i^k = (X_i > d_{k+1}) + (X_i \geq d_{k+1}) + (X_i \geq d_k)$. That is, $Y_i^k = 3$ if $X_i > d_{k+1}$, $Y_i^k = 2$ if $X_i = d_{k+1}$, $Y_i^k = 1$ if $X_i = d_k$, and $Y_i^k = 0$ if $X_i < d_k$.

Within the ith variable partition, we enforce GAC on a REGULAR constraint on $Y_{p(i)}^k$ to $Y_{p(i+1)-1}^k$ to compute the difference between O_k^i and O_{k+1}^i and assign this difference to a new integer variable D_k^i. The automaton associated with this REGULAR constraint has state variables $Q_{p(i)}^k$ to $Q_{p(i+1)-1}^k$ whose values are tuples containing the difference between the two counts seen so far as well as the last value seen (so that we can ensure that values for Y_i^k are increasing). From (δ, y), the transition function on seeing Y_i^k moves to the new state $(\delta + (Y_i^k = 2) - (Y_i^k = 1), \max(y, Y_i^k))$ if and only if $Y_i^k \geq y$. The initial state is $(0, 0)$. We set the difference between the two counts in the final state variable equal to the new integer variable D_k^i (which is thus constrained to equal $O_{k+1}^i - O_k^i$) Finally, we ensure that the vectors, $(O_k^1, .., O_k^a)$ and $(O_{k+1}^1, .., O_{k+1}^a)$ are ordered using a final REGULAR constraint on the difference variables, D_k^1 to D_k^a. The associated automaton has 0/1 states, a transition function which moves from state b to $b \vee (D_k^i < 0)$ provided $D_k^i \leq 0$ or $b = 1$, an initial state 0 and 0 or 1 as final states.

The constraint graph of all the REGULAR constraints is Berge-acyclic. Hence enforcing GAC on these REGULAR constraints achieves GAC on the variables Y_i^k [17]. Consider a support for the Y_i^k variables. We can extend this to a support for the X_i variables simply by picking the smallest value left in their domains after we have enforced GAC on the channelling constraints between the X_i and Y_i^k variables. Support for values left in the domains of the X_i variables can be constructed in a similar way. Enforcing GAC on this decomposition therefore achieves GAC on SIGLEX$(k, p, [X_1, .., X_n])$.

Assuming bounds can be accessed and updated in constant time and a constraint is awoken only if the domain of a variable in its scope has been modified, enforcing GAC on the ordering constraints takes $O(n)$ time, on the channelling constraints between X_i and Y_i^k takes $O(n)$ time, on the first set of REGULAR constraints which compute D_k^i takes $O(n^2)$ time, and on the final REGULAR constraint takes $O(na)$ time. As $a \leq n$, enforcing GAC on SIGLEX takes $O(n^2)$ time. $\diamond$

We compare this with the GCC decomposition in [9]. This requires a GCC propagator which prunes the bounds of the occurrence variables. This will take $O(mn^2 + n^{2.66})$ time [12]. To break the same symmetry, we need to post up to $O(m)$ SIGLEX constraints, which take $O(mn^2)$ time in total to propagate. In the best case for this new decomposition, m grows slower than $O(n^{0.66})$ and we are faster. In the worst case, m grows as $O(n^{0.66})$ or worse and both propagators take $O(mn^2)$ time. The new decomposition is thus sometimes better but not worse than the old one. We conjecture that the two decompositions are incomparable. The GCC decomposition reasons more globally about occurrences, whilst the SIGLEX decomposition reasons more globally about supports of increasing value. Indeed, we can exhibit a problem on which the SIGLEX decomposition gives exponential savings. We predict that the reverse is also true.

Theorem 2 *On the pigeonhole problem $PHP(n)$ with n interchangeable variables and $n + 1$ interchangeable values, we explore $O(2^n)$ branches when enforcing GAC and breaking symmetry using the* GCC *decomposition irrespective of the variable and value ordering, but we solve in polynomial time when enforcing GAC using* SIGLEX.

Proof: The problem has $n+1$ constraints of the form $\bigvee_{i=1}^n X_i = d_j$ for $1 \leq j \leq n+1$, with $X_i \in \{d_1, .., d_{n+1}\}$ for $1 \leq i \leq n$. The problem is unsatisfiable by a simple pigeonhole argument. Enforcing GAC on SIGLEX$(i, [X_1, \dots, X_n])$ for $i > 0$ prunes d_{i+1} from X_1. Hence, X_1 is set to d_1. Enforcing GAC on SIGLEX$(i, p, [X_1, \dots, X_n])$ for $i > 1$ now prunes d_{i+1} from X_2. The domain of X_2 is thus reduced to $\{d_1, d_2\}$. By a similar argument, the domain of each X_i is reduced to $\{d_1, \dots d_i\}$. The SIGLEX constraints are now GAC. Enforcing GAC on the constraint $\bigvee_{i=1}^n X_i = d_{n+1}$ then proves unsatisfiability. Thus, we prove that the problem is unsatisfiable in polynomial time. On the other hand, using the GCC decomposition, irrespective of the variable and value ordering, we will only terminate each branch when $n-1$ variables have been assigned (and the last variable is forced). A simple calculation shows that the size of the search tree as least doubles as we increase n by 1. Hence we will visit $O(2^n)$ branches before declaring the problem unsatisfiable. $\diamond$

5 Some special cases

Variables are not interchangeable

Suppose we have interchangeable values but no variable symmetries (i.e. $a = n$ and $b < m$). To eliminate all symmetric solutions in such a situation, Law and Lee introduced value precedence [4]. This breaks symmetry by constraining when a value is first used. More precisely, PRECEDENCE$(k, [X_1, .., X_n])$ holds iff $\min\{i \mid X_i = d_k \vee i = n+1\} < \min\{i \mid X_i = d_{k+1} \vee i = n+2\}$. That is, the first time we use d_k is before the first time we use d_{k+1}. This prevents the two values being interchanged. It is not hard to show that the SIGLEX constraint is equivalent to value precedence in this situation.

Theorem 3 PRECEDENCE$(k, [X_1, .., X_n])$ *is equivalent to* SIGLEX$(k, p, [X_1, .., X_n])$ *when* $n = a$ *(i.e.,* $p(i) = i$ *for* $i \in [1, n]$*).*

Proof: If $n = a$ then the vectors computed within SIGLEX, namely $(O^1_k, .., O^a_k)$ and $(O^1_{k+1}, .., O^a_{k+1})$, are n-ary 0/1 vectors representing the indices at which d_k and d_{k+1} appear. Lexicographically ordering these vectors ensures that either d_k is used before d_{k+1} or neither are used. This is equivalent to value precedence. $\diamond$

In this case, the propagator for SIGLEX mirrors the work done by the propagator for PRECEDENCE given in [16]. Although both propagators have the same asymptotic cost, we might prefer the one for PRECEDENCE as it introduces fewer intermediate variables.

All variables and values are interchangeable

Another special case is when all variables and values are fully interchangeable (i.e. $a = b = 1$). To eliminate all symmetric solutions in such a situation, Walsh introduced a global constraint which ensures that the sequence of values is increasing but the number of their occurrences is decreasing [16]. More precisely, DECSEQ$([X_1, .., X_n])$ holds iff $X_1 = d_1$, $X_i = X_{i+1}$ or ($X_i = d_j$ and $X_{i+1} = d_{j+1}$) for $1 \leq i < n$ and $|\{i \mid X_i = d_k\}| \geq |\{i \mid X_i = d_{k+1}\}|$ for $1 \leq k < m$. Not surprisingly, the SIGLEX constraint ensures such an ordering of values.

Theorem 4 DECSEQ$([X_1, .., X_n])$ *is equivalent to* SIGLEX$(k, p, [X_1, .., X_n])$ *for* $1 \leq k < m$ *when* $a = b = 1$ *(i.e.,* $p : \{1\} \rightarrow \{1\}$*).*

Proof: Suppose SIGLEX$(k, p, [X_1, .., X_n])$ holds for $1 \leq k < m$. Then $O^1_k \geq O^1_{k+1}$ for $1 \leq k < m$. Now $O^1_k = |\{i \mid X_i = d_k\}|$. Hence $|\{i \mid X_i = d_k\}| \geq |\{i \mid X_i = d_{k+1}\}|$ for $1 \leq k < m$. Suppose $O^1_1 = 0$. Then $O^1_k = 0$ for $1 \leq k \leq m$ and no values can be used. This is impossible. Hence $O^1_1 > 0$ and d_1 is used. As $X_1 \leq .. \leq X_n$, $X_1 = d_1$. Suppose that d_k is the first value not used. Then $O^1_k = 0$. Hence $O^1_j = 0$ for all $j > k$. That is, all values up to d_k are used and all values including and after d_k are not used. Since $X_i \leq X_{i+1}$, it follows that $X_i = X_{i+1}$ or ($X_i = d_j$ and $X_{i+1} = d_{j+1}$) for $1 \leq i < n$. Thus, DECSEQ$([X_1, .., X_n])$ holds. The proof reverses easily. $\diamond$

6 Variable partition ordering

Suppose there are two variable partitions $\{X_1, X_2\}$ and $\{X_3, X_4, X_5\}$, and all domain values $d_1, \ldots, d_5$ are interchangeable. Section 4 suggests that we can break the symmetry using $\text{SigLex}(k, p, \boldsymbol{X})$ for $1 \leq k < 5$, where $\boldsymbol{X} = [X_1, \ldots, X_5]$, $p(1) = 1$ and $p(2) = 3$. In fact, the symmetry can be also broken by posting the SigLex constraints in another way: $\text{SigLex}(k, p', \boldsymbol{X}')$ for $1 \leq k < 5$, where $\boldsymbol{X}' = [X'_1, X'_2, X'_3, X'_4, X'_5] = [X_3, X_4, X_5, X_1, X_2]$, $p'(1) = 1$ and $p'(2) = 4$. The former posting places the partition $\{X_1, X_2\}$ in front of $\{X_3, X_4, X_5\}$ in SigLex, and vice versa for the latter. The two postings eliminate different symmetric solutions, i.e., the solutions of the two postings are different. We observe that in the presence of AllDiff constraints, the order of the variable partitions placed in the SigLex constraints affects propagation. In the following, we study the issue of variable partition ordering in details.

In the above example, suppose that we also have $\text{AllDiff}([X_1, X_2])$ and $\text{AllDiff}([X_3, X_4, X_5])$, and we enforce GAC on these constraints. GAC on the former set of SigLex constraints alone removes $d_2, \ldots, d_5$ from the domain of X_1 (due to the lexicographic ordering on the signatures), making X_1 grounded. This triggers propagation on $\text{AllDiff}([X_1, X_2])$ that removes d_1 from the domain of X_2. Further propagation on the constraints results in $X_1 \in \{d_1\}$, $X_2 \in \{d_2\}$, $X_3 \in \{d_1, d_3\}$, $X_4 \in \{d_1, d_2, d_3, d_4\}$ and $X_5 \in \{d_1, d_2, d_3, d_4, d_5\}$. Note that all variables in the partition $\{X_1, X_2\}$ are grounded.

On the other hand, if we use the constraints $\text{SigLex}(k, p', \boldsymbol{X}')$ for $1 \leq k < 5$, then enforcing GAC on these constraints and the two AllDiff constraints results in $X_1 \in \{d_1, d_4\}$, $X_2 \in \{d_1, d_2, d_4, d_5\}$, $X_3 \in \{d_1\}$, $X_4 \in \{d_2\}$ and $X_5 \in \{d_3\}$. This time, all variables in the partition $\{X_3, X_4, X_5\}$ are grounded.

In general, not every variable partition would contain an AllDiff constraint. If, however, all domain values are interchangeable and the *first* variable partition placed in the SigLex constraints contains an AllDiff, GAC on the SigLex and AllDiff constraints will either cause a domain wipe-out or ground *all* variables in the first partition.

Theorem 5 *Enforcing GAC on* $\text{AllDiff}([X_{p(1)}, \ldots, X_{p(2)-1}])$ *and the set of* SigLex *constraints decomposed from* BreakInterchangeability *causes either domain wipe-out or* $X_i = d_i$ *for* $p(1) \leq i \leq \min(p(2) - 1, m)$ *if* $b = 1$.

Proof: Consider two cases $m < p(2) - 1$ or $m \geq p(2) - 1$. The former causes a domain wipe-out, as there are fewer domain values than variables in the AllDiff constraint.

For the latter case, we first prove by induction that $X_i \notin \{d_{i+1}, \ldots, d_m\} \forall p(1) \leq i \leq p(2) - 1$. When $i = p(1) = 1$, suppose conversely $X_1 = d_k$ for any $1 < k \leq m$. As SigLex implies $X_1 \leq \ldots \leq X_{p(2)-1}$, we get $O^1_k \geq 1$ and $O^1_1 = 0$. But $O^1_k > O^1_1$ violates the lexicographic order on the signatures. Hence, $X_1 \neq d_k$ for $1 < k \leq m$, i.e., $X_1 = d_1$. Assume the cases are true $\forall 1 \leq i' < i$. Suppose conversely $X_i = d_k$, for any $i < k \leq m$. Since $\forall i' < i, X_{i'} \notin \{d_{k-1}, d_k\}$, we get $O^1_k \geq 1$ and $O^1_{k-1} = 0$, which contradicts with the lexicographic order. This completes the induction.

We can now prove that for $p(1) \leq i < p(2)$, either $X_i = d_i$ or X_i has empty domain. Since $X_1 \notin \{d_2, \ldots, d_m\}$, obviously either $X_1 = d_1$ or X_1 has empty domain. Suppose $X_1 = d_1$, enforcing GAC on $\text{AllDiff}([X_{p(1)}, \ldots, X_{p(2)-1}])$ will remove

d_1 from the domains of $X_2, \ldots, X_{p(2)-1}$. Now, $X_2 \neq d_1$ and $X_2 \notin \{d_3, \ldots, d_m\}$. Then we have $X_2 = d_2$ or X_2 has empty domain. We can repeat the process of enforcing GAC on ALLDIFF$([X_{p(1)}, \ldots, X_{p(2)-1}])$ to consequently make either $X_i = d_i$ or domain wipe-out for $p(1) \leq i < p(2)$. $\diamond$

Therefore, if more than one variable partition contains an ALLDIFFconstraint, then placing the largest variable partition at the front in the SIGLEX constraints ensures the most variables are grounded, and therefore the most simplification of the problem. This can be seen from the above example, in which the first posting grounds only two variables, while the second posting grounds three. Furthermore, in the ALLDIFF constraints, a grounded variable in a larger partition triggers more prunings than one in a smaller partition, and enforcing GAC on SIGLEX tends to prunes values from variables earlier in the variable sequence, due to the $\geq_{\text{lex}}$ ordering on the signatures. Therefore, it is a good idea to place larger variable partitions at the front in the SIGLEX constraints to increase the chance of more prunings due to a grounded variable. This gives us a heuristic to rearrange the variables in the SIGLEX constraints so that *(1) variable partitions with* ALLDIFF *constraints are ordered before those without* ALLDIFF*, and (2) among those variable partitions with* ALLDIFF *constraints, order them in decreasing partition size.*

Theorem 5 applies to problems where all domain values are interchangeable. When there is more than one value partition ($b > 1$), enforcing GAC on the SIGLEX constraints does not necessarily ground the first variable in the first partition, since the first value $d_{q(j)}$ in every partition j can remain in its domain, making no subsequent groundings of the other variables in the partition by the SIGLEX and ALLDIFF constraints. Nonetheless, propagation by other problem constraints or variable instantiations during search can eventually ground variables and trigger the prunings by the ALLDIFF constraints. Therefore, in the case of $b > 1$, it is still worthwhile to reorder the variables using this heuristic. Note that this variable partition ordering heuristic helps improve the amount of pruning for the SIGLEX decomposition. Although it can be applied also to the GCC decomposition, the heuristic may not help here.

The discussion brings out an interesting question about posting symmetry breaking constraints. Ideally, symmetry breaking constraints remove all but one solution from each equivalence class of solutions. Different postings of the symmetry breaking constraints can leave a different solution. This is true for SIGLEX and also other symmetry breaking constraints. In terms of eliminating symmetric solutions, it does not matter which solution we leave. However, in terms of propagation, apparently the different postings give different behaviour. This is one of the first times it has been shown that breaking symmetry to leave a particular distinguished element of a symmetry class can reduce search. It would be interesting to study this systematically and formally in the future.

7 Implementation notes

The proof of Theorem 1 already gives an overview of the implementation of the SIGLEX global constraint, which involves enforcing the $\leq$ ordering of the X_i variables, the channelling between the X_i and Y_i^k variables, and $a + 1$ REGULAR constraints, where a is the number of variable partitions. The first a REGULAR constraints are used to do the counting, while the last one enforces the $\geq_{\text{lex}}$ ordering using the final state information associated with the automata in the first a REGULAR constraints. The maintenance of the $\leq$ ordering and the channeling is straightforward. The REGULAR constraint, however, has to be slightly modified to fit the requirement of our implementation. In particular, we introduce an extra finite domain variable F_s to the REGULAR constraint so that REGULAR$([X_1, \ldots, X_n], F_s, \mathcal{M})$ means F_s is the final state of the string $[X_1, \ldots, X_n]$ admissible by the DFA $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$, where Q is the set of all states, Σ is the alphabet, δ is the state transition, q_0 is the initial state, and F is the set of final states.

Pesant [11] proposes a GAC propagator for the original REGULAR constraint by maintaining an associated layered directed multigraph $(N^1, \ldots, N^{n+1}, A)$, where n is the number of variables in the constraint. Let $\Sigma = \{v_1, \ldots, v_n\}$. Each layer $N^i = \{q_0^i, \ldots, q_{|Q|-1}^i\}$ contains a node q_l^i for each state $q_l \in Q$ and directed arcs in A appear only between two consecutive layers. The graph is acyclic by construction. There exists an arc from q_k^i to q_l^{i+1} iff there exists some v_j in the domain of X_i such that $\delta(q_k, v_j) = q_l$. The arc is labelled with the value v_j allowing the transition between the two states.

The multigraph is *consistent* if each node has a non-zero in-degree and a non-zero out-degree. Suppose a multigraph is inconsistent, i.e., there exists a node with either in-degree or out-degree being zero. We can make the multigraph consistent again by removing the node together with all its incoming or outgoing arcs from the graph. When the arc from q_k^i to q_l^{i+1} is removed, we check if the out-degree of q_k^i become zeros and if the in-degree of q_l^{i+1} becomes zero to ensure consistency of the multigraph. Pesant [11] gives a theorem stating that REGULAR$([X_1, \ldots, X_n], \mathcal{M})$ is GAC iff the domain of X_i is equal to the set of all labels from the outgoing arcs originating from nodes in layer N^i of a consistent multigraph associated with $\mathcal{M}$.

In the original constraint REGULAR$([X_1, \ldots, X_n], \mathcal{M})$, propagation is triggered when some values are deleted from the domain of some variable X_i. This corresponds to removing arcs from the associated multigraph of $\mathcal{M}$. In the new constraint REGULAR$([X_1, \ldots, X_n], F_s, \mathcal{M})$, we have to allow *also* triggerings caused by value deletions from the domain of F_s. This corresponds to removing a node in layer N^{n+1} of the associated multigraph of $\mathcal{M}$. If such a removal causes inconsistency in the multigraph, Pesant's procedure is still able to restore consistency. We can easily verify that REGULAR is GAC iff the domain of X_i is equal to the set of all labels from the outgoing arcs originating from nodes in layer N^i of a consistent multigraph associated with $\mathcal{M}$, and the domain of F_s is the set of nodes (states) in layer N^{n+1}.

We show in the proof of Theorem 1 that a SIGLEX constraint can be decomposed into several REGULAR constraints without hindering propagation, since the constraint graph of the decomposition is Berge-acyclic. However, we provide a global constraint implementation for SIGLEX as it provides opportunities for efficiencies. In our implementation, we achieved GAC on SIGLEX using a two-pass iteration. In the first pass, we enforce GAC on the decomposed constraints in a forward manner, from the $\leq$ ordering constraints on the X_i variables to the final REGULAR constraint for the $\geq_{\text{lex}}$ ordering. In the second pass, the constraints are propagated again but in the reverse order. This two-pass iteration guarantees that each constraint in the decomposition is propagated at most twice but GAC is still enforced on one SIGLEX constraint as a whole.

8 Experiments

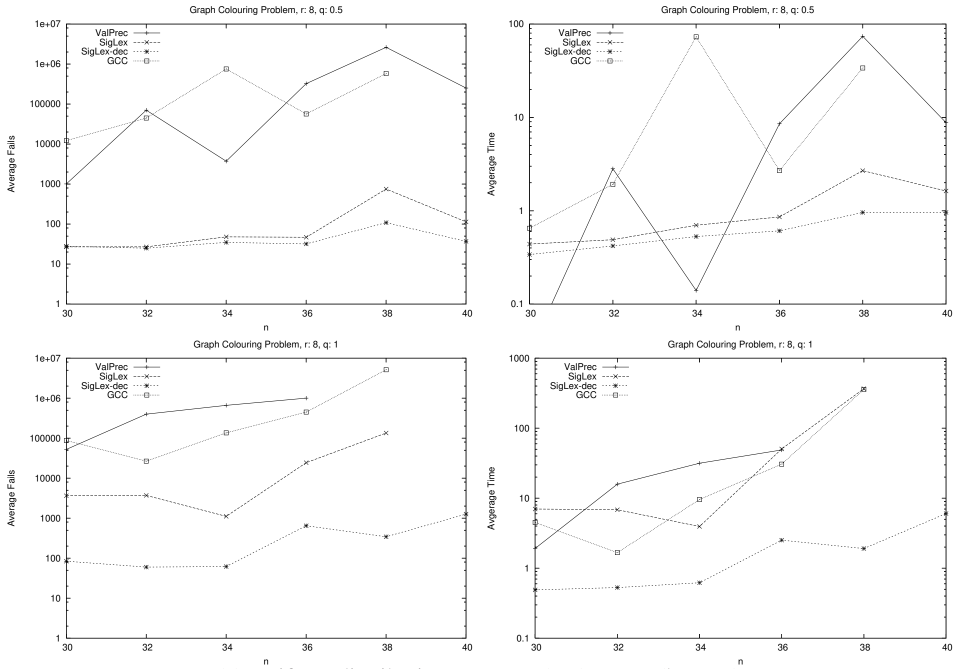
To test the efficiency and effectiveness of the SIGLEX constraints, we perform experiments on the graph colouring and concert hall scheduling problems. We compare the SIGLEX constraints against (1) the GCC decomposition (GCC) and (2) PRECEDENCE constraints with $\leq$ ordering constraints (ValPrec). All three methods break both the variable and value interchangeability. When using SIGLEX constraints, we consider two variable partition orderings: the data file ordering (SigLex) and the decreasing partition size ordering (SigLex-dec) introduced in Section 6. The experiments are run on a Sun Blade 2500 (2 $\times$ 1.6GHz US-IIIi, 2GB RAM) using ILOG Solver 4.4. The time limit is 1 hour. The variable ordering heuristic is to choose first a variable with the smallest domain. Both benchmark problems are optimisation problems, and we report the number of fails and CPU time to find and prove the optimum in the results.

8.1 Graph colouring

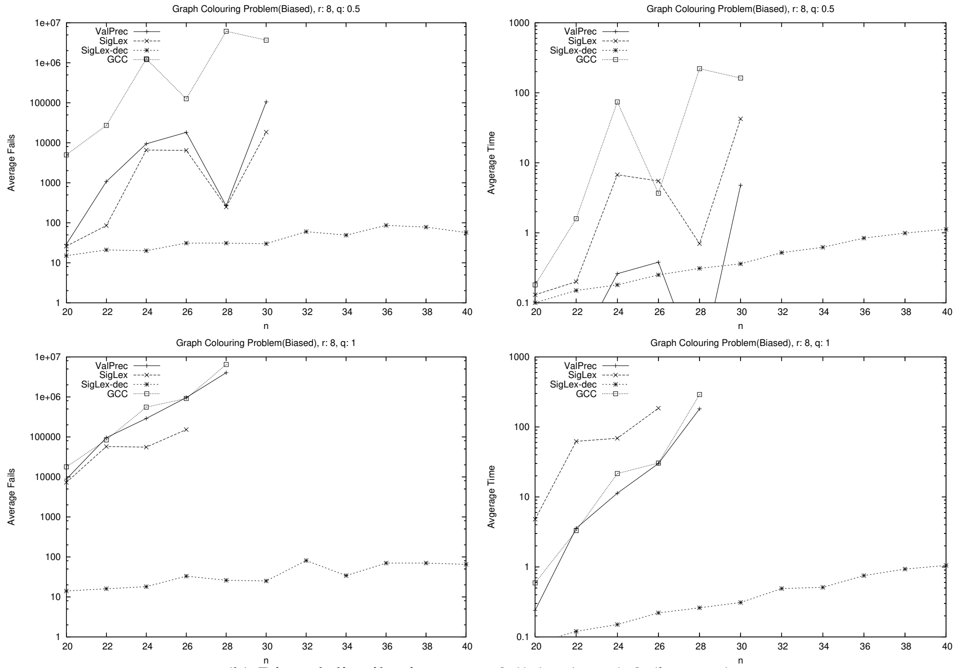
In graph colouring, nodes having the same set of neighbours form a partition and are interchangeable. We generate random graphs using four parameters $\langle n, r, p, q\rangle$, where n is the number of nodes and r is the maximum node partition size. We start from an independent graph (graph with no arcs) with n nodes and ensure node interchangeability while adding arcs to the graph in two steps. First, the subgraph containing nodes of two partitions must be either a complete bipartite or independent graph. In Example 1, the graph between partitions $\{X_1, X_2\}$ and $\{X_3, X_4, X_5\}$ is complete bipartite. The parameter p is the proportion of complete bipartite subgraphs between pairs of partitions. Second, the subgraph in one partition must also be either complete or independent. In Example 1, both subgraphs of the two partitions are independent. The parameter q is the proportion of complete subgraphs among the partitions. A complete subgraph in a partition is modelled using an ALLDIFF constraint on the variable partition.

With the four parameters, we generate two types of random graphs using different distributions on the variable partition size. In the first type, the number of nodes n_i in the ith partition is *uniformly* distributed in $[1, r]$. Since n is initially fixed, if the generated value of a particular n_i makes the total number of nodes exceed n, then the ith partition will be the final partition and its size n_i will be chosen such that the total number of nodes is exactly n. The second type has a *biased* distribution. The size of the first $\lfloor \frac{n}{2} \rfloor$ partitions are preset to 1, i.e., $\lfloor \frac{n}{2} \rfloor$ of the nodes are not interchangeable at all. The remaining nodes are then partitioned using a uniform distribution like in the first type. The latter type of graphs models a common scenario in real life problems where variable interchangeability occurs in only a subset of the variables. We test with various values of n, $r = 8$, $p = 0.5$ and $q \in \{0.5, 1\}$, and 20 instances are generated for each set of parameters. Fig. 1(a) and (b) show the experimental results for the uniform and biased distributions respectively. A data point is plotted only when at least 90% of the instances are solved within the time limit. All graphs are plotted in the log-scale.

At the same parameter setting, the instances of biased distribution are more difficult to solve than those of uniform distribution, since the former instances have fewer symmetries than the latter and thus fewer symmetry breaking constraints can be posted to reduce the search space. Nevertheless, for both distributions, SigLex and SigLex-dec



(a) Uniform distribution: $q = 0.5$ (top) or 1.0 (bottom)



(b) Biased distribution: $q = 0.5$ (top) or 1.0 (bottom)

Fig. 1. Graph colouring: average number of fails (left) and time (right)

took fewer number of fails than GCC and ValPrec in almost all parameter settings. The fewer number of fails, however, does not always lead to better run times, due to the overhead incurred by the introduction of intermediate variables inside the implementation of the SigLex constraint. ValPrec is competitive only for small values of n. The relative performance of SigLex and SigLex-dec over GCC and ValPrec increases as q increases. Among the two variable partition orderings in SigLex, SigLex-dec has much better performance than SigLex in both number of fails and run time, confirming the effectiveness of our proposed heuristic. The performance advantage of SigLex-dec over the other models becomes larger as both n and q increases.

8.2 Concert hall scheduling

A concert hall director receives n applications to use the k identical concert halls. Each application specifies a period and an offered price to use a hall for the whole period. The concert hall scheduling problem [18] is to decide which applications to accept in order to maximise the total income. Each accepted application should be assigned the same hall during its whole applied period. We use a variable to represent each application whose domain is $\{1, \ldots, k+1\}$ in two value partitions. Values $1, \ldots, k$ represent the k interchangeable halls, while the value $k+1$ represents a rejected application. Variables representing identical applications (same period and offered price) are interchangeable and form a partition. We generate problem instances with a maximum of $r = 8$ identical applications, and the size of each partition is generated uniformly. We test with n from 20 to 40 in steps of 2, $r = 8$ and $k \in \{10, 14\}$. Experimental results are shown in Fig. 2.

Regarding the number of fails, SigLex-dec achieves the best result and ValPrec performs the worst. Regarding the run time, SigLex-dec also achieves the best for almost all cases. SigLex has a slower run-time on this problem, despite a better number of fails than ValPrec and GCC. This is again due to the implementation overhead in the SigLex constraint. The decreasing variable partition ordering heuristic helps hugely to improve the pruning performance and hence outweigh the overhead to reduce the run time. We also generated instances using a biased distribution like in graph colouring and obtained similar experimental results. Due to space limitation, we skip the details.

Note that the concert hall scheduling problem does not really have AllDiff constraints in the variable partitions. However, the problem constraints are very similar to AllDiff: two variables X_i and X_j representing two interchangeable applications cannot take the same value (hall) *if* the two applications are not rejected. That is, $X_i \neq X_j$ if $X_i \neq k+1$ and $X_j \neq k+1$. The propagation behaviour of such kind of constraint is still similar to that of AllDiff. Thus, the decreasing size variable partition ordering can still help improve the pruning performance.

9 Related work

Puget proved that symmetries can always be eliminated by the additional of suitable constraints [1]. Crawford *et al.* presented the first general method for constructing such symmetry breaking constraints, which are so-called "lex-leader" constraints [2]. They also argued that it is NP-hard to eliminate all symmetric solutions in general. The full

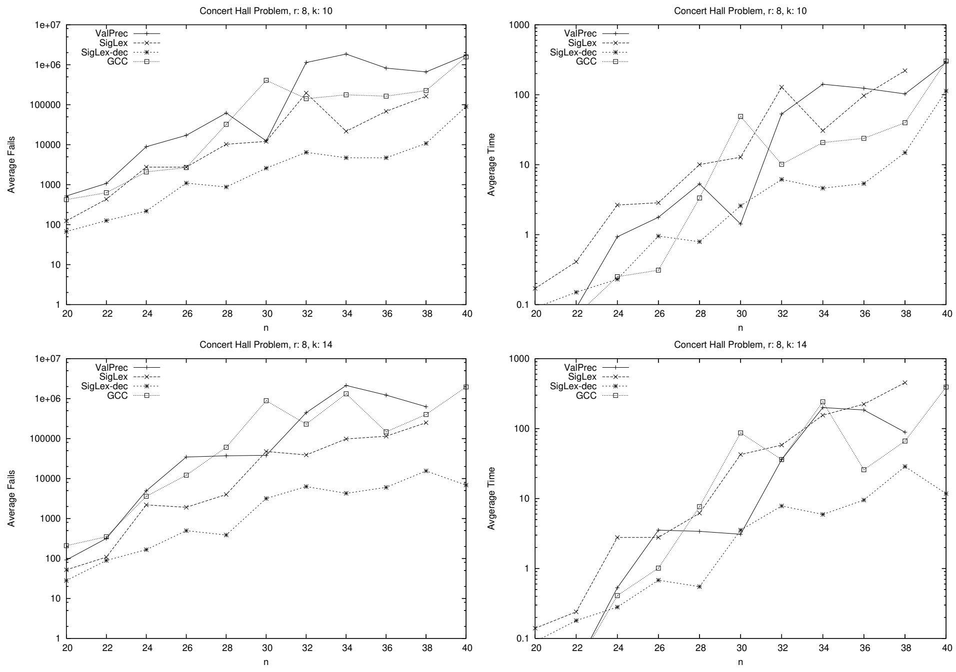


Fig. 2. Concert hall scheduling (uniform distribution): average number of fails (left) and time (right), $k = 10$ (top) or 14 (bottom)

set of lex-leader constraints can often be simplified. For example, when variables are interchangeable and must take all different values, Puget showed that the lex-leader constraints simplify to a linear number of binary inequality constraints [19]. To break value symmetry, Puget introduced one variable per value and a linear number of binary constraints [20]. Law and Lee formally defined value precedence and proposed a specialised propagator for breaking the special type of value symmetry between two interchangeable values [4]. Walsh extended this to a propagator for any number of interchangeable values [16]. Finally, an alternative way to break value symmetry statically is to convert it into a variable symmetry by channelling into a dual viewpoint and using lexicographic ordering constraints on this dual view [3, 18]. Different postings of symmetry breaking constraints can leave a different solution from each equivalence class of solutions and affect search performance. Frisch *et al.* discussed choosing a good posting to give the best propagation and allow new implied constraints [21]. Smith presented experiments with different symmetry breaking constraints using the graceful graph problem, but did not give heuristics for choosing a good posting [22].

10 Conclusions

We have considered breaking the symmetry introduced by interchangeable variables and values. Whilst there exist polynomial methods to eliminate all symmetric solutions, pruning all symmetric values is NP-hard. We have introduced a new propagator called SIGLEX for pruning some (but not necessarily all) symmetric values. The new propagator is based on a decomposition using REGULAR constraints. We have also introduced a heuristic for ordering the variable partitions when posting SIGLEX constraints that improves pruning. Finally, we have tested these symmetry breaking constraints experimentally for the first time and shown that they are effective in practice.

References

1. Puget, J.F.: On the satisfiability of symmetrical constrained satisfaction problems. In: Proc. of ISMIS'93. (1993) 350–361
2. Crawford, J., Luks, G., Ginsberg, M., Roy, A.: Symmetry breaking predicates for search problems. In: Proc. of KR'96. (1996) 148–159
3. Flener, P., Frisch, A., Hnich, B., Kiziltan, Z., Miguel, I., Pearson, J., Walsh, T.: Breaking row and column symmetry in matrix models. In: Proc. of CP'02. (2002) 462–476
4. Law, Y.C., Lee, J.: Global constraints for integer and set value precedence. In: Proc. of CP'04. (2004) 362–376
5. Fahle, T., Schamberger, S., Sellmann, M.: Symmetry breaking. In: Proc. of CP'01. (2001) 93–107
6. Gent, I., Smith, B.: Symmetry breaking in constraint programming. In: Proc. of ECAI'00. (2000) 599–603
7. Roney-Dougal, C., Gent, I., Kelsey, T., Linton, S.: Tractable symmetry breaking using restricted search trees. In: Proc. of ECAI'04. (2004) 211–215
8. Walsh, T.: Breaking value symmetry. In: Proc. of CP'07. (2007)
9. Flener, P., Pearson, J., Sellmann, M., Van Hentenryck, P.: Static and dynamic structural symmetry breaking. In: Proc. of CP'06. (2006) 695–699
10. Sellmann, M., Van Hentenryck, P.: Structural symmetry breaking. In: Proc. of IJCAI'05. (2005) 298–303
11. Pesant, G.: A regular language membership constraint for finite sequences of variables. In: Proc. of CP'04. (2004) 482–295
12. Quimper, C., van Beek, P., Lopez-Ortiz, A., Golynski, A.: Improved algorithms for the global cardinality constraint. In: Proc. of CP'04. (2004) 542–556
13. Quimper, C., Walsh, T.: Global grammar constraints. In: Proc. of CP'06. (2006) 751–755
14. Puget, J.F.: Automatic detection of variable and value symmetries. In: Proc. of CP'05. (2005) 475–489
15. Frisch, A., Hnich, B., Kiziltan, Z., Miguel, I., Walsh, T.: Propagation algorithms for lexicographic ordering constraints. Artificial Intelligence **170** (2006) 803–908
16. Walsh, T.: Symmetry breaking using value precedence. In: Proc. of ECAI'06. (2006) 168–172
17. Beeri, C., Fagin, R., Maier, D., Yannakakis, M.: On the desirability of acyclic database schemes. JACM **30** (1983) 479–513
18. Law, Y.C., Lee, J.: Symmetry breaking constraints for value symmetries in constraint satisfaction. Constraints **11** (2006) 221–267
19. Puget, J.F.: Breaking symmetries in all different problems. In: Proc. of IJCAI'05. (2005) 272–277
20. Puget, J.F.: Breaking all value symmetries in surjection problems. In: Proc. of CP'05. (2005) 490–504
21. Firsch, A., Jefferson, C., Miguel, I.: Symmetry breaking as a prelude to implied constraints: A constraint modelling pattern. In: Proc. of ECAI'04. (2004) 171–175
22. Smith, B.: Sets of symmetry breaking constraints. In: Proc. of SymCon'05. (2005)