Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)

Static Symmetry Breaking with the Reflex Ordering

Jimmy H. M. Lee and Zichen Zhu
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong
{jlee,zzhu}@cse.cuhk.edu.hk

Abstract

LexLeader, a state of the art static symmetry breaking method, adds a lex ordering constraint for each variable symmetry of the problem to select the lexicographically least solution. In practice, the same method can also be used for partial symmetry breaking by breaking only a given subset of symmetries. We propose a new total ordering, reflex, as basis of a new symmetry breaking constraint that collaborates well among themselves as well as with Precedence constraints, thereby breaking more composition symmetries in partial symmetry breaking. An efficient GAC filtering algorithm is presented for the reflex ordering constraint. We propose the ReflexLeader method, which is a variant of LexLeader using the reflex ordering instead, and give conditions when ReflexLeader is safe to combine with the Precedence and multiset ordering constraints. Extensive experimentations demonstrate empirically our claims and substantial advantages of the reflex ordering over the lex ordering in partial symmetry breaking.

1 Introduction

Symmetries are common in many constraint problems. They can be broken statically [Crawford *et al.*, 1996; Flener *et al.*, 2002] or dynamically [Fahle *et al.*, 2001; Roney-Dougal *et al.*, 2004]. LexLeader [Crawford *et al.*, 1996], as the state of the art static method, adds lex ordering constraints to select the lexicographically least solution to break symmetries of indistinguishable objects. In practice, the same method is often used for partial symmetry breaking by only breaking a subset of symmetries, with DOUBLELEX [Flener *et al.*, 2002] and SNAKELEX [Grayland *et al.*, 2009] being well known special cases. The multiset ordering constraint [Frisch *et al.*, 2009] and Gray code constraint [Narodytska and Walsh, 2013] are also proposed to break variable symmetries based on other orderings.

Given the same subset of symmetries, the pruning power of partial symmetry breaking depends highly on the extra composition symmetries that can be broken by the interactions among symmetry breaking constraints [Lee and Li, 2012]. Recently, Lee and Zhu [2016] propose two heuristics to break more composition symmetries on the basis of Partial Symmetry Breaking During Search [Gent and Smith, 2000]. Their method restricts the search heuristic and thus cannot be combined with other search techniques. The multiset ordering is only a partial order and is thus often used in combination with the lex ordering constraint. The Gray code constraint focuses on Boolean variable vectors. We propose a new total ordering, reflex, and also a reflex ordering constraint for breaking variable symmetries on integer problems. We show empirically that given the same subset of symmetries, reflex ordering constraints (and their combination with the Precedence constraints [Law and Lee, 2004]) can break more composition symmetries than lex ordering constraints (and their combination with the Precedence constraints).

We give theorems to compare the reflex and lex orders. An efficient GAC filtering algorithm is given. Based on this reflex ordering, the ReflexLeader method is thus proposed by adding one reflex ordering constraint for every variable symmetry of the problem or a subset. We prove that ReflexLeader is safe to combine with Precedence [Law and Lee, 2004] which is used to break value interchangeability as long as they use the same fixed variable and value orders. In matrix problems, we prove that multiset ordering [Frisch *et al.*, 2009] the rows (columns) and reflex ordering the columns (rows) are sound. Experimentations demonstrate the reflex ordering constraint can drastically reduce the solution set size, search space and runtime when compared against state-of-the-art methods based on the lex ordering constraint.

2 Background

A *constraint satisfaction problem* (CSP) P is a tuple (X, D, C) where X is a finite set of variables, D is a finite set of domains such that each $x \in X$ has a domain $D(x)$ and C is a set of constraints, each a subset of the Cartesian product $D(x_{i_1}) \times \cdots \times D(x_{i_k})$ of the domains of the involved variables (*scope*). A constraint is *generalized arc consistent (GAC)* iff every value in the domain of a variable in the scope can find compatible values (called *supports*) from the domains of all the other variables in the scope. An *assignment* $x = v$ assigns value v to variable x. A *full assignment* is a set of assignments, one for each variable in X. A *solution* to P is a full assignment that satisfies every constraint in C.

Here we consider symmetry as a property of the set of solutions. A *solution symmetry* [Rossi *et al.*, 2006] is a solution-

preserving permutation on assignments. A *variable symmetry* σ is a bijection on variables that preserves solutions: if $\{x_i = v_i | 1 \leq i \leq n\}$ is a solution, then $\{x_{\sigma(i)} = v_i | 1 \leq i \leq n\}$ is also one. A *value symmetry* θ is a bijection on values that preserves solutions: if $\{x_i = v_i | 1 \leq i \leq n\}$ is a solution, then $\{x_i = \theta(v_i) | 1 \leq i \leq n\}$ is also. A set of values V is *interchangeable* iff any bijection mapping from $V \rightarrow V$ is a value symmetry. A *symmetry class* [Flener *et al.*, 2002] is an equivalence class of full assignments, where two assignments are equivalent if there is some symmetry mapping one into the other. A symmetry breaking method is *sound* (*complete*) iff it leaves at least (most) one solution in each symmetry class.

Given a group of variable symmetries G. LexLeader [Crawford *et al.*, 1996] adds one lex ordering constraint [Frisch *et al.*, 2002], $\leq_{lex}$, per variable symmetry $g \in G$ according to a fixed variable order. DOUBLELEX [Flener *et al.*, 2002] and SNAKELEX [Grayland *et al.*, 2009] are two partial methods based on LexLeader by posting constraints to break only special subsets of matrix symmetries.

3 The Reflex Ordering

We introduce *reflex*, a new total ordering. First, we give some basic definitions. Given an integer vector $\mathbf{a}$. Notation $min(\mathbf{a})$ denotes the min value in $\mathbf{a}$, and $posm(\mathbf{a})$ denotes the first position of $min(\mathbf{a})$ in $\mathbf{a}$. We use $\mathbf{a} - \langle posm(\mathbf{a}) \rangle$ to denote the vector after deleting the first occurance of $min(\mathbf{a})$ from $\mathbf{a}$. Vectors are indexed from 0, and $\mathbf{a}_i$ denotes the ith item of $\mathbf{a}$.

An integer vector $\mathbf{a}$ is *reflexicographically (reflex) smaller than* another integer vector $\mathbf{b}$, written $\mathbf{a} <_{rlex} \mathbf{b}$, iff $\mathbf{a} = NULL$ and $\mathbf{b} \neq NULL$; or $min(\mathbf{a}) < min(\mathbf{b})$; or $min(\mathbf{a}) = min(\mathbf{b})$ and $posm(\mathbf{a}) < posm(\mathbf{b})$; or $min(\mathbf{a}) = min(\mathbf{b})$ and $posm(\mathbf{a}) = posm(\mathbf{b})$ and $\mathbf{a} - \langle posm(\mathbf{a}) \rangle <_{rlex} \mathbf{b} - \langle posm(\mathbf{b}) \rangle$. An integer vector $\mathbf{a}$ is *reflex smaller than or equal to* another integer vector $\mathbf{b}$, written $\mathbf{a} \leq_{rlex} \mathbf{b}$, iff $\mathbf{a} = \mathbf{b}$ or $\mathbf{a} <_{rlex} \mathbf{b}$. In other words, the reflex ordering first compares the least values in the two vectors, then the first occurrence positions of the least values if the values are the same. If both the least values and their first occurrence positions are all equal, the two vectors are compared again after discarding the least values from the vectors respectively.

Consider the following two vectors

$$\mathbf{a} = \langle 3, 1, 4, 1, 4 \rangle, \quad \mathbf{b} = \langle 2, 1, 2, 5, 1 \rangle. \tag{1}$$

$\mathbf{a} <_{rlex} \mathbf{b}$ since the min values in $\mathbf{a}$ and $\mathbf{b}$ are both 1 and both occur first at position 1, but the second occurrence of 1 in $\mathbf{a}$ appears earlier than that in $\mathbf{b}$.

The reflex ordering is like a *mirror reflection* of the lex ordering, which compares first the least positions (always the same) and then the values of the least positions in the vectors. If the least (zeroth) position values are the same, they are discarded from the vectors, which will be compared again.

We can also define reflex in another way. If $pa = (v, p)$ is a pair, then we define $pa.val = v$ and $pa.pos = p$. Given an integer vector $\mathbf{a}$, the *sorted index vector* $\mathbf{a}^s$ of $\mathbf{a}$ is a vector of pairs such that (1) $|\mathbf{a}^s| = |\mathbf{a}| = n$, (2) each item in $\mathbf{a}^s$ is a pair (v, p), (3) $\forall i \in [0, n), \exists j \in [0, n)$ such that $(\mathbf{a}_i, i) = \mathbf{a}^s_j$ and (4) items in $\mathbf{a}^s$ are sorted in increasing order of v and ties are broken by p in increasing order again.

The sorted index vectors $\mathbf{a}^s$ and $\mathbf{b}^s$ of vectors in (1) are

$$\begin{aligned} \mathbf{a}^s &= \langle (1,1), (1,3), (3,0), (4,2), (4,4) \rangle \\ \mathbf{b}^s &= \langle (1,1), (1,4), (2,0), (2,2), (5,3) \rangle. \end{aligned} \tag{2}$$

We overload $<$ for integer pairs and define $(p_1, p_2) < (q_1, q_2)$ iff (1) $p_1 < q_1$ or (2) $p_1 = q_1$ and $p_2 < q_2$. Thus, we can have $\leq_{lex}$ on two vectors of pairs. Suppose $\mathbf{a}$ and $\mathbf{b}$ are two vectors. We state the following theorem without proof.

Theorem 1. *Given two integer vectors* $\mathbf{a}$ *and* $\mathbf{b}$. $\mathbf{a} \leq_{rlex} \mathbf{b}$ *iff* $\mathbf{a}^s \leq_{lex} \mathbf{b}^s$.

We say that $\mathbf{a} <_{lex} \mathbf{b}$ *at position* i iff $\mathbf{a}_i < \mathbf{b}_i$ and $\mathbf{a}_j = \mathbf{b}_j$ for all $j < i$. Consider the two vectors in (1), $\mathbf{a} <_{rlex} \mathbf{b}$ since $\mathbf{a}^s <_{lex} \mathbf{b}^s$ at position 1.

Theorem 2. $<_{rlex}$ *is a total ordering on integer vectors.*

Proof. The result follows directly from Theorem 1 and the fact that $<_{lex}$ is a total ordering. □

The reflex and lex orderings coincide in only a special circumstance.

Theorem 3. *Given two integer vectors* $\mathbf{a}$ *and* $\mathbf{b}$ *where the number of distinct values in them is 2.* $\mathbf{a} \leq_{rlex} \mathbf{b}$ *iff* $\mathbf{a} \leq_{lex} \mathbf{b}$.

Proof. W.L.O.G, we assume there are only 0s and 1s in $\mathbf{a}$ and $\mathbf{b}$. The theorem is trivially true when $\mathbf{a} = \mathbf{b}$. We prove the theorem for $\mathbf{a} \neq \mathbf{b}$. ($\Rightarrow$) $\mathbf{a} <_{rlex} \mathbf{b}$ means $\mathbf{a}^s <_{lex} \mathbf{b}^s$. Suppose $\mathbf{a}^s <_{lex} \mathbf{b}^s$ at position i. Then all values before this position in $\mathbf{a}$ and $\mathbf{b}$ are pairwise equal and $\mathbf{a}_{\mathbf{a}^s_i.pos} = 0$ but $\mathbf{b}_{\mathbf{a}^s_i.pos} = 1$. Thus $\mathbf{a} <_{lex} \mathbf{b}$. ($\Leftarrow$) If $\mathbf{a} <_{lex} \mathbf{b}$ at position j, we must have $\mathbf{a}_j = 0$ and $\mathbf{b}_j = 1$. Thus $\mathbf{a}^s <_{lex} \mathbf{b}^s$ at position k and $\mathbf{a}^s_k.pos = j$. $\mathbf{a} <_{rlex} \mathbf{b}$. □

Proposition 1. *Given two integer vectors* $\mathbf{a}$ *and* $\mathbf{b}$ *where the number of distinct values in* $\mathbf{a}$ *and* $\mathbf{b}$ *is larger than 2. There are cases that (1)* $\mathbf{a} <_{rlex} \mathbf{b}$ *and* $\mathbf{a} <_{lex} \mathbf{b}$ *and (2)* $\mathbf{a} <_{rlex} \mathbf{b}$ *and* $\mathbf{a} >_{lex} \mathbf{b}$.

Proof. (1) $\mathbf{a} = \langle 0, 0, 1, 2, 3, 1 \rangle$ and $\mathbf{b} = \langle 0, 0, 1, 3, 2, 1 \rangle$. (2) $\mathbf{a} = \langle 0, 0, 2, 1, 0, 1 \rangle$ and $\mathbf{b} = \langle 0, 0, 1, 1, 2, 0 \rangle$. □

4 GAC Enforcement

Utilizing the definition of sorted index vector and Theorem 1, the reflex ordering constraint $\mathbf{X} \leq_{rlex} \mathbf{Y}$ can be decomposed into (a) $|\mathbf{X}| + |\mathbf{Y}|$ element constraints and $|\mathbf{X}| + |\mathbf{Y}| - 2$ ordering constraints to sort pairs to get the sorted index variable vectors $\mathbf{X}^s$ and $\mathbf{Y}^s$ and (b) $\mathbf{X}^s \leq_{lex} \mathbf{Y}^s$. However, this decomposition hinders propagation and is also expensive due to the introduction of extra variables and constraints.

In the following, we give a GAC algorithm on the *reflex ordering constraint* $\mathbf{X} \leq_{rlex} \mathbf{Y}$ on two integer variable vectors. We assume $\mathbf{X}$ and $\mathbf{Y}$ share no variables and have the same length. We use $\mathbf{fx}$ (resp. $\mathbf{cy}$) to denote the vector with all variables in $\mathbf{X}$ (resp. $\mathbf{Y}$) assigned the min (resp. max) values for their respective domains. Integer vector $\mathbf{fx}_{x=v}$ is the result of replacing the value assigned to x in $\mathbf{fx}$ to v, and similarly for $\mathbf{cy}_{y=v}$. Functions $min(\mathbf{X}_i)$ and $max(\mathbf{X}_i)$ return the min and max values in $D(\mathbf{X}_i)$ respectively.

The GAC enforcement is based on the following theorems.

Theorem 4. $\mathbf{X} \leq_{rlex} \mathbf{Y}$ *is GAC iff (1)* $\forall x \in \mathbf{X}$, $\mathbf{fx}_{x=max(x)} \leq_{rlex} \mathbf{cy}$ *and (2)* $\forall y \in \mathbf{Y}$, $\mathbf{fx} \leq_{rlex} \mathbf{cy}_{y=min(y)}$.

Proof. ($\Rightarrow$) If $\mathbf{X} \leq_{rlex} \mathbf{Y}$ is GAC, all values in the domains of variables in $\mathbf{X}$ and $\mathbf{Y}$ can find supports. Given any value v in the domain of $x \in \mathbf{X}$, the min values in the domains of all other variables in $\mathbf{X}$ and the max values in the domains of variables in $\mathbf{Y}$ must be supports of v. In particular, (1) is true. Similarly, (2) is true. ($\Leftarrow$) If (1) is true, then for all values v in $D(x)$: $\mathbf{fx}_{x=v} \leq_{rlex} \mathbf{cy}$. Thus all values in the domain of x are supported. Similarly, by (2), all values in the domains of $y \in \mathbf{Y}$ are also supported. □

The proof of the following theorem is similar.

Theorem 5. $\mathbf{X} \leq_{rlex} \mathbf{Y}$ *is unsatisfiable iff* $\mathbf{fx} >_{rlex} \mathbf{cy}$.

Combining Theorems 4 and 5 with 1 respectively, we get the following corollaries.

Corollary 1. $\mathbf{X} \leq_{rlex} \mathbf{Y}$ *is GAC iff (1)* $\forall x \in \mathbf{X}$, $(\mathbf{fx}_{x=max(x)})^s \leq_{lex} \mathbf{cy}^s$ *and (2)* $\forall y \in \mathbf{Y}$, $\mathbf{fx}^s \leq_{lex} (\mathbf{cy}_{y=min(y)})^s$.

Corollary 2. $\mathbf{X} \leq_{rlex} \mathbf{Y}$ *is unsatisfiable iff* $\mathbf{fx}^s >_{lex} \mathbf{cy}^s$.

Enforcing GAC on $\mathbf{X} \leq_{rlex} \mathbf{Y}$ can be achieved by pruning inconsistent values v from the domains of all $x \in \mathbf{X}$ such that $(\mathbf{fx}_{x=v})^s >_{lex} \mathbf{cy}^s$ and from the domains of all $y \in \mathbf{Y}$ such that $\mathbf{fx}^s >_{lex} (\mathbf{cy}_{y=v})^s$. Thus, the upper bounds of all $x \in \mathbf{X}$ and the lower bounds of all $y \in \mathbf{Y}$ are tightened. Note that this does not change $\mathbf{fx}^s$ and $\mathbf{cy}^s$ as long as no domains become empty. To prune values v from the domains of $x \in \mathbf{X}$, we do not generate each $(\mathbf{fx}_{x=v})^s$ to check if $(\mathbf{fx}_{x=v})^s >_{lex} \mathbf{cy}^s$ is true. Similarly, we do not generate each $(\mathbf{cy}_{y=v})^s$ to check if $\mathbf{fx}^s >_{lex} (\mathbf{cy}_{y=v})^s$ is true. We utilize three pointers, $\mathbf{fx}^s$ and $\mathbf{cy}^s$ plus our own rules to prune inconsistent values. Our algorithm resembles more to that of the multiset ordering constraint [Frisch *et al.*, 2009] than that of the lex ordering constraint [Frisch *et al.*, 2002]. The outline of the algorithm follows.

1. Compute $\mathbf{fx}$, $\mathbf{cy}$, $\mathbf{fx}^s$ and $\mathbf{cy}^s$.
2. Set the proper values of the three pointers α, β and γ (to be defined below) to help prune inconsistent values.
3. Tighten the lower bounds of all $y \in \mathbf{Y}$.
4. Tighten the upper bounds of all $x \in \mathbf{X}$.

We give the algorithms to enforce GAC with the help of $\mathbf{fx}^s$ and $\mathbf{cy}^s$, and illustrate with a running example. Consider two disjoint and same sized variable vectors, in which variables are represented by their domains: $\mathbf{X} = \langle\{5\},\{2,3\},\{3\},\{4\},\{1,2,5\},\{3\},\{1,2\}\rangle$ and $\mathbf{Y} = \langle\{2,4\},\{1,2,3\},\{3\},\{2,3\},\{0,1\},\{1,2\},\{0,1\}\rangle$. Then $\mathbf{fx} = \langle 5,2,3,4,1,3,1\rangle$, $\mathbf{cy} = \langle 4,3,3,3,1,2,1\rangle$, $\mathbf{fx}^s = \langle(1,4),(1,6),(2,1),(3,2),(3,5),(4,3),(5,0)\rangle$ and $\mathbf{cy}^s = \langle(1,4),(1,6),(2,5),(3,1),(3,2),(3,3),(4,0)\rangle$.

Variables with singleton domains are automatically assigned. We can check if x is assigned by $x.assigned()$, and $x.gq(v)$ and $x.lq(v)$ enforce $x \geq v$ and $x \leq v$ respectively.

Algorithm 1 is the top level of the GAC algorithm. This algorithm is called whenever the domain bound of a variable in $\mathbf{X}$ or $\mathbf{Y}$ is modified. We first get $\mathbf{fx}$ and $\mathbf{cy}$ using *floor*() and *ceiling*() respectively, and generate the sorted index vectors $\mathbf{fx}^s$ and $\mathbf{cy}^s$ using function *sortinv*() respectively (line 1). To help enforce GAC, we define the following three pointers to $\mathbf{fx}^s$ and $\mathbf{cy}^s$: α, β and γ. They are all initialized to -1 (line 2). Algorithm 2 is called (line 3) to set them so that

Algorithm 1 *Reflex*($\mathbf{X}, \mathbf{Y}$)

1: $\mathbf{fx}^s \leftarrow$*sortinv*(*floor*($\mathbf{X}$)); $\mathbf{cy}^s \leftarrow$*sortinv*(*ceiling*($\mathbf{Y}$));
2: int $\alpha \leftarrow -1$; int $\beta \leftarrow -1$; int $\gamma \leftarrow -1$;
3: *SetPointer*($\mathbf{X}$,$\mathbf{Y}$,$\mathbf{fx}$,$\mathbf{cy}$,α,β,γ);
4: **if** $\alpha = -2$ **then** return **FAILED**;
5: **else if** $\alpha = -1$ **then** return **SUBSUMED**;
6: *Enforce*$\mathbf{Y}$($\mathbf{Y}$,$\mathbf{fx}$,$\mathbf{cy}$,α,β,γ);
7: **if** *Enforce*$\mathbf{X}$($\mathbf{fx}$,$\mathbf{cy}$,$\mathbf{X}_{\mathbf{fx}^s_\alpha.pos}$,$\mathbf{Y}_{\mathbf{fx}^s_\alpha.pos}$,$\alpha$,$\beta$,$\gamma$,$|\mathbf{Y}|$) = **SUBSUMED then** return **SUBSUMED**;
8: return **GACED**;

1. $\mathbf{fx}^s_\alpha < \mathbf{cy}^s_\alpha$ and $\forall i \in [0,\alpha)$, $\mathbf{fx}^s_i = \mathbf{cy}^s_i$;
2. $\mathbf{fx}^s_{\beta+1} > \mathbf{cy}^s_\beta$, $\beta \geq \alpha$ and $\forall i \in (\alpha,\beta]$, $\mathbf{fx}^s_i = \mathbf{cy}^s_{i-1}$;
3. $\mathbf{fx}^s_\gamma > \mathbf{cy}^s_\gamma$, $\gamma > \beta$ and $\forall i \in (\beta,\gamma)$, $\mathbf{fx}^s_i = \mathbf{cy}^s_i$.

At α, $\mathbf{fx}^s <_{lex} \mathbf{cy}^s$. If the vectors are equal, α is still -1. If $\mathbf{fx}^s >_{lex} \mathbf{cy}^s$, α is set to -2. Pointer β is always larger than or equal to α for pruning inconsistent values from the domains of $\mathbf{X}_{\mathbf{fx}^s_\alpha.pos}$ and $\mathbf{Y}_{\mathbf{fx}^s_\alpha.pos}$. For cases where we cannot find such a β, if $\forall i \in (\alpha, |\mathbf{fx}^s|)$, $\mathbf{fx}^s_i = \mathbf{cy}^s_{i-1}$, β is still -1. Otherwise, β is set to -2. Pointer γ points to the index larger than β such that $\langle \mathbf{fx}^s_{\beta+1}, ...\rangle >_{lex} \langle \mathbf{cy}^s_{\beta+1}, ...\rangle$ at position γ. If such case is impossible, γ is still -1. In our example, the values of the three pointers are: $\alpha = 2$, $\beta = 2$ and $\gamma = 3$.

For space reasons, we skip the description of the simple iteration in Algorithm 2, which achieves the definition of α, β and γ described above. One thing to note in Algorithm 2 is that, if $\forall j \leq i, \mathbf{fx}^s_j = \mathbf{cy}^s_j$ (lines 3-4), $x = \mathbf{X}_{\mathbf{fx}^s_i.pos}$ and $y = \mathbf{Y}_{\mathbf{cy}^s_i.pos}$ are set to their min and max values respectively (lines 5-6). The reason is if x is assigned a larger value v than *min*(x) or y is assigned a smaller value v than *max*(y), $(\mathbf{fx}_{x=v})^s >_{lex} \mathbf{cy}^s$ and $\mathbf{fx}^s >_{lex} (\mathbf{cy}_{y=v})^s$. Thus, in our example, $\mathbf{X}_4$, $\mathbf{X}_6$, $\mathbf{Y}_4$ and $\mathbf{Y}_6$ are all enforced to be 1.

In Algorithm 1, if α is set to -2 (line 4), $\mathbf{fx}^s >_{lex} \mathbf{cy}^s$. The constraint is **FAILED** (line 4) according to Corollary 2. If α is still -1 (line 5), $\mathbf{fx}^s = \mathbf{cy}^s$ and all variables in $\mathbf{X}$ and $\mathbf{Y}$ are set to their min and max values respectively. $\mathbf{X} = \mathbf{Y}$ and the constraint is **SUBSUMED** (line 5). Otherwise, Algorithms 3 and 4 are called to prune inconsistent values in the domains of variables in $\mathbf{Y}$ and $\mathbf{X}$ respectively (lines 6-7). If Algorithm 4 returns **SUBSUMED**, $\mathbf{X} <_{rlex} \mathbf{Y}$ for any possible assignment and the constraint is **SUBSUMED** (line 7). After the enforcements, the constraint is **GACED** (line 8).

Algorithm 3 prunes inconsistent values from the domains of variables in $\mathbf{Y}$. Consider all non-assigned variables $y = \mathbf{Y}_i$ where $i < \mathbf{fx}^s_\alpha.pos$ (lines 1-2). If y is assigned a value $v \leq \mathbf{fx}^s_\alpha.val$, (v,i) is moved to the kth position of $(\mathbf{cy}_{y=v})^s$ where $k \leq \alpha$ and $\mathbf{fx}^s >_{lex} (\mathbf{cy}_{y=v})^s$ at position k. In our example, if $\mathbf{Y}_0 = 2$, $(2,0)$ is moved to the second position of $(\mathbf{cy}_{\mathbf{Y}_0=2})^s$ and $\mathbf{fx}^s >_{lex} (\mathbf{cy}_{\mathbf{Y}_0=2})^s$ at this position. We thus enforce $y > \mathbf{fx}^s_\alpha.val$ (line 2). Value 2 is pruned from $D(\mathbf{Y}_0)$. Consider all non-assigned variables $y = \mathbf{Y}_i$ where

Algorithm 2 *SetPointer*($\mathbf{X}$,$\mathbf{Y}$,$\mathbf{fx}$,$\mathbf{cy}$,α,β,γ)

1: int $i \leftarrow 0$; int $j \leftarrow 0$;
2: **while** $i < |\mathbf{X}|$, $j < n$ **do** {
3: **if** $\mathbf{fx}^s_i = \mathbf{cy}^s_j$ **then** {
4: **if** $\alpha = -1$ **then** {
5: $\mathbf{X}_{\mathbf{fx}^s_i.pos} \leftarrow \mathbf{fx}^s_i.val$;
6: $\mathbf{Y}_{\mathbf{cy}^s_j.pos} \leftarrow \mathbf{cy}^s_j.val$;}}
7: **else if** $\mathbf{fx}^s_i < \mathbf{cy}^s_j$ **then**
8: **if** $\alpha = -1$ **then** $\{\alpha \leftarrow i; j\text{--};\}$
9: **else** {
10: **if** $\beta = -1$ **then** $\beta \leftarrow -2$;
11: break;}
12: **else if** $\alpha = -1$ **then** $\{\alpha \leftarrow -2$; break;$\}$
13: **else if** $\beta = -1$ **then** $\{\beta \leftarrow j; i\text{--};\}$
14: **else** $\{\gamma \leftarrow i$; break;$\}$
15: i++; j++;}
16: return;

Algorithm 3 *Enforce*$\mathbf{Y}$($\mathbf{Y}$,$\mathbf{fx}$,$\mathbf{cy}$,α,β,γ)

1: **for** each i in $[0, \mathbf{fx}^s_\alpha.pos)$ **do**
2: **if not** $\mathbf{Y}_i$.*assigned*() **then** $\mathbf{Y}_i.gq(\mathbf{fx}^s_\alpha.val + 1)$;
3: **for** each i in $(\mathbf{fx}^s_\alpha.pos, n)$ **do**
4: **if not** $\mathbf{Y}_i$.*assigned*() **then** $\mathbf{Y}_i.gq(\mathbf{fx}^s_\alpha.val)$;
5: $y \leftarrow \mathbf{Y}_{\mathbf{fx}^s_\alpha.pos}$;
6: **if not** y.*assigned*() **then**
7: **if** $\beta \leq -1$ **then** $y.gq(\mathbf{fx}^s_\alpha.val)$;
8: **else if** $\mathbf{fx}^s_\alpha.pos = \mathbf{cy}^s_\beta.pos$ **then**
9: **if** $\gamma = -1$ **then** $y.gq(\mathbf{fx}^s_\alpha.val)$;
10: **else** $y.gq(\mathbf{fx}^s_\alpha.val + 1)$;
11: **else if** $(max(y), \mathbf{fx}^s_\alpha.pos) > (\mathbf{cy}^s_\beta.val, \mathbf{cy}^s_\beta.pos)$
12: **then** $y.gq(\mathbf{fx}^s_\alpha.val + 1)$;
13: **else** $y.gq(\mathbf{fx}^s_\alpha.val)$;
14: return;

Algorithm 4 *Enforce*$\mathbf{X}$($\mathbf{fx}$,$\mathbf{cy}$,x,y,α,β,γ,n)

1: **if** x.*assigned*() **then**
2: **if** $min(y) > \mathbf{fx}^s_\alpha.val$ **then** return **SUBSUMED**;
3: **else if** $\beta = -1$ **then**
4: **if** $\mathbf{cy}^s_{n-1}.pos < \mathbf{fx}^s_\alpha.pos$ **then**
5: $x.lq(\mathbf{cy}^s_{n-1}.val - 1)$;
6: **else** $x.lq(\mathbf{cy}^s_{n-1}.val)$;
7: **else if** $\beta > -1$ **then**
8: **if** $\mathbf{fx}^s_\alpha.pos = \mathbf{cy}^s_\beta.pos$ **then**
9: **if** $\gamma = -1$ **then** $x.lq(\mathbf{cy}^s_\beta.val)$;
10: **else** $x.lq(\mathbf{cy}^s_\beta.val - 1)$;
11: **else if** $\mathbf{fx}^s_\alpha.pos < \mathbf{cy}^s_\beta.pos$ **then** $x.lq(\mathbf{cy}^s_\beta.val)$;
12: **else** $x.lq(\mathbf{cy}^s_\beta.val - 1)$;
13: return;

$i > \mathbf{fx}^s_\alpha.pos$ (lines 3-4), if y is assigned a value $v < \mathbf{fx}^s_\alpha.val$, (v, i) is moved to the kth position of $(\mathbf{cy}_{y=v})^s$ where $k \leq \alpha$ and $\mathbf{fx}^s >_{lex} (\mathbf{cy}_{y=v})^s$ at this position. In our example, if $\mathbf{Y}_5 = 1$, (1,5) is moved to the first position of $(\mathbf{cy}_{\mathbf{Y}_5=1})^s$ and $\mathbf{fx}^s >_{lex} (\mathbf{cy}_{\mathbf{Y}_5=1})^s$ at this position. We thus enforce $y \geq \mathbf{fx}^s_\alpha.val$ (line 4). In our example, $\mathbf{Y}_i \geq 2$ for all $i > 1$. Value 1 is pruned from $D(\mathbf{Y}_5)$.

For non-assigned variable $y = \mathbf{Y}_{\mathbf{fx}^s_\alpha.pos}$ (line 6), there are five cases. If $\beta \leq -1$ (line 7), $\mathbf{fx}^s \leq_{lex} (\mathbf{cy}_{y=\mathbf{fx}^s_\alpha.val})^s$. We only need to enforce $y \geq \mathbf{fx}^s_\alpha.val$ (line 7). Otherwise, if $\mathbf{fx}^s_\alpha.pos = \mathbf{cy}^s_\beta.pos$ (line 8), items in $\mathbf{fx}^s$ and $(\mathbf{cy}_{y=\mathbf{fx}^s_\alpha.val})^s$ are pairwise equal from positions 0 to β. If $\gamma = -1$ (line 9), $\mathbf{fx}^s \leq_{lex} (\mathbf{cy}_{y=\mathbf{fx}^s_\alpha.val})^s$. Thus we only need to enforce $y \geq \mathbf{fx}^s_\alpha.val$ (line 9). Otherwise (line 10), $\mathbf{fx}^s >_{lex} (\mathbf{cy}_{y=\mathbf{fx}^s_\alpha.val})^s$ at position γ, we enforce $y > \mathbf{fx}^s_\alpha.val$ (line 10). For case $\mathbf{fx}^s_\alpha.pos \neq \mathbf{cy}^s_\beta.pos$ and $(max(y), \mathbf{fx}^s_\alpha.pos)$ occurring after position β at $\mathbf{cy}^s$ (line 11), if y is assigned a value $v < \mathbf{fx}^s_\alpha.val$, $(v, \mathbf{fx}^s_\alpha.pos)$ is moved to the kth position of $(\mathbf{cy}_{y=v})^s$ where $k \leq \alpha$ and $\mathbf{fx}^s >_{lex} (\mathbf{cy}_{y=v})^s$ at position k. If y is assigned $\mathbf{fx}^s_\alpha.val$, $(\mathbf{fx}^s_\alpha.val, \mathbf{fx}^s_\alpha.pos)$ is moved to position α in $(\mathbf{cy}_{y=\mathbf{fx}^s_\alpha.val})^s$. Now $\mathbf{fx}^s_\alpha = (\mathbf{cy}_{y=\mathbf{fx}^s_\alpha.val})^s_\alpha$ and $\mathbf{fx}^s >_{lex} (\mathbf{cy}_{y=\mathbf{fx}^s_\alpha.val})^s$ at position $\beta + 1$. We thus enforce $y > \mathbf{fx}^s_\alpha.val$ (line 12). This case occurs in our example since $\mathbf{fx}^s_\alpha.pos = 1$ and $\mathbf{cy}^s_\beta.pos = 5$, and $(3, 1)$ occurs after position β in $\mathbf{cy}^s$. If $\mathbf{Y}_1$ is assigned value 1, (1,1) is moved to the zeroth position of $(\mathbf{cy}_{\mathbf{Y}_1=1})^s$ and $\mathbf{fx}^s >_{lex} (\mathbf{cy}_{\mathbf{Y}_1=1})^s$ at this position. If $\mathbf{Y}_1$ is assigned value 2, (2,1) is moved to the second position of $(\mathbf{cy}_{\mathbf{Y}_1=2})^s$ and $\mathbf{fx}^s >_{lex} (\mathbf{cy}_{\mathbf{Y}_1=2})^s$ at position 3 since $(3, 2) > (2, 5)$. We thus enforce $\mathbf{Y}_1 > 2$ and values 1 and 2 are pruned from $D(\mathbf{Y}_1)$. Otherwise (line 13), we only need to enforce $y \geq \mathbf{fx}^s_\alpha.val$ (line 13).

Algorithm 4 prunes inconsistent values only from the domain of $x = \mathbf{X}_{\mathbf{fx}^s_\alpha.pos}$. All values in the domains of other variables x' in $\mathbf{X}$ already have supports since $(\mathbf{fx}_{x'=max(x')})^s <_{lex} \mathbf{cy}^s$ at position α. If x is assigned (line 1) and $\mathbf{Y}_{\mathbf{fx}^s_\alpha.pos}$ is always larger than $\mathbf{fx}^s_\alpha.val$ (line 2), there always have $\mathbf{X} <_{rlex} \mathbf{Y}$ at position α. The constraint is **SUBSUMED** (line 2). Otherwise, if $\beta = -1$ (line 3), this means $\mathbf{fx}^s_i = \mathbf{cy}^s_{i-1}$ for all $i > \alpha$. We need to make sure $(x, \mathbf{fx}^s_\alpha.pos) \leq \mathbf{cy}^s_{n-1}$ and x is enforced accordingly (lines 4-6). If $\beta = -2$, all values in the domain of x are supported. If $\beta > -1$ (line 7), there are four cases. If $\mathbf{fx}^s_\alpha.pos = \mathbf{cy}^s_\beta.pos$ (line 8), items in $(\mathbf{fx}_{x=\mathbf{fx}^s_\beta.val})^s$ and $\mathbf{cy}^s$ are pairwise equal from positions 0 to β. If $\gamma = -1$ (line 9), $(\mathbf{fx}_{x=\mathbf{fx}^s_\beta.val})^s \leq_{lex} \mathbf{cy}^s$. Thus we only need to enforce $x \leq \mathbf{fx}^s_\beta.val$ (line 9). Otherwise (line 10), $x < \mathbf{fx}^s_\beta.val$ (line 10). For the case $\mathbf{fx}^s_\alpha.pos < \mathbf{cy}^s_\beta.pos$ (line 11), if x is assigned any value $v > \mathbf{cy}^s_\beta.val$, $(v, \mathbf{fx}^s_\alpha.pos)$ is moved to the kth position of $(\mathbf{fx}_{x=v})^s$ where $k \geq \beta$ and $(\mathbf{fx}_{x=v})^s >_{lex} \mathbf{cy}^s$ at position β. We thus enforce $x \leq \mathbf{fx}^s_\beta.val$ (line 11). This case occurs in our example since $1 < 5$. If $\mathbf{X}_1 = 3$, $(3, 1)$ replaces $(2, 1)$ in $(\mathbf{fx}_{\mathbf{X}_1=3})^s$ and $(\mathbf{fx}_{\mathbf{X}_1=3})^s >_{lex} \mathbf{cy}^s$ at position 2. We thus enforce $\mathbf{X}_1 \leq 2$ and value 3 is pruned from $D(\mathbf{X}_1)$. If $\mathbf{fx}^s_\alpha.pos > \mathbf{cy}^s_\beta.pos$, we enforce $x < \mathbf{fx}^s_\beta.val$ (line 12).

After the enforcements, $\mathbf{X}$ and $\mathbf{Y}$ in our example are updated to: $\mathbf{X} = \langle\{5\}, \{2\}, \{3\}, \{4\}, \{1\}, \{3\}, \{1\}\rangle$ and $\mathbf{Y} = \langle\{4\}, \{3\}, \{3\}, \{2, 3\}, \{1\}, \{2\}, \{1\}\rangle$. $\mathbf{X} \leq_{rlex} \mathbf{Y}$ is GAC.

Theorem 6. *Algorithm 1 enforces GAC of* $\mathbf{X} \leq_{rlex} \mathbf{Y}$.

Proof. Algorithm 1 always terminates. All values pruned in Algorithm 1 are inconsistent values. We now prove values left always have supports. If $\alpha = -2$, the constraint is unsatisfiable according to Corollary 2. If $\alpha = -1$ after running Algorithm 1, all variables in $\mathbf{X}$ and $\mathbf{Y}$ are assigned (lines 5-6 in Algorithm 2), $\mathbf{X} = \mathbf{Y}$. If $\alpha > -1$, based on Corollary 1, we only need to prove for any $x \in \mathbf{X}$ and $y \in \mathbf{Y}$, $(\mathbf{fx}_{x=max(x)})^s \leq_{lex} \mathbf{cy}^s$ and $\mathbf{fx}^s \leq_{lex} (\mathbf{cy}_{y=min(y)})^s$. (1) Consider all variables $x = \mathbf{X}_{\mathbf{fx}_i^s.pos}$ where $i \neq \alpha$, $(\mathbf{fx}_{x=max(x)})^s <_{lex} \mathbf{cy}^s$ at α. (2) Consider variable $x = \mathbf{X}_{\mathbf{fx}_\alpha^s.pos}$. If x is not assigned, there are four cases. If $\beta = -1$, $(\mathbf{fx}_{x=max(x)})^s = \mathbf{cy}^s$ or $(\mathbf{fx}_{x=max(x)})^s <_{lex} \mathbf{cy}^s$ at the position where $(max(x), \mathbf{fx}_\alpha^s.pos)$ appears in $(\mathbf{fx}_{x=max(x)})^s$. If $\beta = -2$, $(\mathbf{fx}_{x=max(x)})^s <_{lex} \mathbf{cy}^s$ at the least position among where $(max(x), \mathbf{fx}_\alpha^s.pos)$ appears in $(\mathbf{fx}_{x=max(x)})^s$ and where β is set to -2 in $\mathbf{cy}^s$ in Algorithm 2. If $\mathbf{fx}_\alpha^s.pos = \mathbf{cy}_\beta^s.pos$ and $\gamma = -1$, $(\mathbf{fx}_{x=max(x)})^s = \mathbf{cy}^s$ or $(\mathbf{fx}_{x=max(x)})^s <_{lex} \mathbf{cy}^s$ at the position where $(max(x), \mathbf{fx}_\alpha^s.pos)$ appears in $(\mathbf{fx}_{x=max(x)})^s$. If $\mathbf{fx}_\alpha^s.pos = \mathbf{cy}_\beta^s.pos$ and $\gamma \neq -1$, or $\mathbf{fx}_\alpha^s.pos \neq \mathbf{cy}_\beta^s.pos$, $(\mathbf{fx}_{x=max(x)})^s <_{lex} \mathbf{cy}^s$ at the position where $(max(x), \mathbf{fx}_\alpha^s.pos)$ appears in $(\mathbf{fx}_{x=max(x)})^s$. (3) Consider all variables $\mathbf{Y}_i$ where $i \neq \mathbf{fx}_\alpha^s.pos$, $\mathbf{fx}^s <_{lex} (\mathbf{cy}_{\mathbf{Y}_i=min(\mathbf{Y}_i)})^s$ at α. (4) Consider variable $y = \mathbf{Y}_{\mathbf{fx}_\alpha^s.pos}$. If y is not assigned, there are three cases. If $\beta \leq -1$, or $\mathbf{fx}_\alpha^s.pos = \mathbf{cy}_\beta^s.pos$ and $\gamma = -1$, $\mathbf{fx}^s = (\mathbf{cy}_{y=min(y)})^s$ or $\mathbf{fx}^s <_{lex} (\mathbf{cy}_{y=min(y)})^s$ at α. If $\mathbf{fx}_\alpha^s.pos = \mathbf{cy}_\beta^s.pos$ and $\gamma \neq -1$, or $\mathbf{fx}_\alpha^s.pos \neq \mathbf{cy}_\beta^s.pos$ and $(max(y), \mathbf{fx}_\alpha^s.pos)$ occurs after β at $\mathbf{cy}^s$, $\mathbf{fx}^s <_{lex} (\mathbf{cy}_{y=min(y)})^s$ at α. If $\mathbf{fx}_\alpha^s.pos \neq \mathbf{cy}_\beta^s.pos$ and $(max(y), \mathbf{fx}_\alpha^s.pos)$ occurs before β at $\mathbf{cy}^s$, $\mathbf{fx}^s <_{lex} (\mathbf{cy}_{y=min(y)})^s$ at α if $min(y) > \mathbf{fx}_\alpha^s.val$ and at $\alpha + 1$ otherwise. □

Theorem 7. *Algorithm 1 runs in $O(nlog(n))$ time for constraint $\mathbf{X} \leq_{rlex} \mathbf{Y}$, where $|\mathbf{X}| = |\mathbf{Y}| = n$.*

Proof. Algorithms *floor*() and *ceiling*() have $O(n)$ time complexity respectively. Algorithm *sortinv*() runs in $O(nlog(n))$. *SetPointer*() runs in $O(n)$. *Enforce*$\mathbf{Y}$() runs in $O(n)$. *Enforce*$\mathbf{X}$() runs in $O(1)$. The total is $O(nlog(n))$. □

Theorem 8. *The space complexity of Algorithm 1 is $O(n)$.*

Proof. Algorithms *floor*(), *ceiling*() and *sortinv*() have space $O(n)$ respectively. Algorithms *SetPointer*(), *Enforce*$\mathbf{Y}$() and *Enforce*$\mathbf{X}$() have space $O(1)$ respectively. □

GAC enforcement of constraint $\mathbf{X} <_{rlex} \mathbf{Y}$ under conditions where $\mathbf{X}$ and $\mathbf{Y}$ do or do not have the same length can be achieved by changing *Enforce*$\mathbf{Y}$() and *Enforce*$\mathbf{X}$() slightly. We do not show them in here due to space limitation.

5 Reflex in Symmetry Breaking

Given a symmetry group G. Symmetry breaking method *ReflexLeader* adds one reflex ordering constraint, $\leq_{rlex}$, per variable symmetry $g \in G$ according to a fixed variable order. Since reflex is a total ordering, we have the following.

Theorem 9. *ReflexLeader is sound and complete.*

In this paper, we use ReflexLeader and LexLeader also to break only a given subset of symmetries.

In matrix problems, lex ordering the rows (columns) breaks all the row (column) symmetries. Similarly, reflex ordering the rows (columns) breaks all the row (column) symmetries since it is a total order. Thus, we can have DOUBLEREFLEX by only adding reflex ordering constraints to break adjacent rows and columns interchangeability.

ReflexLeader also has identical pruning power as LexLeader in the following circumstance.

Theorem 10. *Given a CSP $P = (X, D, C)$ with symmetry group G, $D(x) \subseteq E$ for all $x \in X$ and $|E| = 2$. ReflexLeader and LexLeader have the same solution and node pruning powers when both are enforced GAC and based on the same fixed variable and value orders.*

Proof. We prove $\mathbf{A} \leq_{rlex} \mathbf{B}$ is GAC iff $\mathbf{A} \leq_{lex} \mathbf{B}$ is GAC for any variable vectors $\mathbf{A}$ and $\mathbf{B}$. Any support in $\mathbf{A} \leq_{rlex} \mathbf{B}$ for a value v can be transformed to a support in $\mathbf{A} \leq_{lex} \mathbf{B}$ according to Theorem 3. The reverse is also true. □

Even though ReflexLeader and LexLeader generate the same search tree with Boolean domains, the GAC algorithm of $\mathbf{X} \leq_{lex} \mathbf{Y}$ is $O(n)$ where n is the size of $\mathbf{X}$ and $\mathbf{Y}$. When we use DOUBLEREFLEX and DOUBLELEX to break the matrix symmetries of the 0/1 model of the Balanced Incomplete Block Design problem [Flener *et al.*, 2001], they leave the same search tree but DOUBLELEX will run slightly faster than DOUBLEREFLEX.

Precedence [Law and Lee, 2004] constrains a value s to precede another value t in an integer variable vector $\mathbf{X}$, which means that if there exists j such that $\mathbf{X}_j = t$, then there must exist $i < j$ such that $\mathbf{X}_i = s$. We prove ReflexLeader is safe to combine with Precedence.

Theorem 11. *Given a CSP $P = (X, D, C)$. If P has variable symmetry group G and value interchangeability H. Using ReflexLeader to break G and Precedence to break H is sound as long as they are based on the same fixed variable and value orders.*

Proof. Suppose they are based on the variable and value order Γ. We prove that the reflex least solution according to Γ in each solution symmetry class of symmetry group $G \circ H$ satisfies Precedence. Suppose $\mathbf{S}$ is the reflex least solution in a symmetry class and sequenced according to the variable order in Γ. Suppose further values $i < j$ according to Γ and (a) j occurs earlier than i in $\mathbf{S}$ or (b) j occurs but i does not occur in $\mathbf{S}$. In case (a), we interchange i and j in $\mathbf{S}$ and get a solution $\mathbf{R}$. In case (b), we replace j by i in $\mathbf{S}$ and get a solution $\mathbf{R}$. All the other values in $\mathbf{S}$ and $\mathbf{R}$ occur at the same positions. We simply delete all these values out to get $\mathbf{S}'$ and $\mathbf{R}'$ respectively. Now the first value of $\mathbf{S}'$ is j and the first value of $\mathbf{R}'$ is i, the first i occurs earlier in $\mathbf{R}'$ than that in $\mathbf{S}'$. Since $i < j$, $\mathbf{R}' <_{rlex} \mathbf{S}'$ and $\mathbf{R} <_{rlex} \mathbf{S}$ which contradicts with $\mathbf{S}$ being the reflex least solution. □

The multiset ordering constraint [Frisch *et al.*, 2009] ensures the values taken by two variable vectors, when viewed

Table 1: Unconstrained Matrix Problem

n,m,d	LexLeader+Precedence						ReflexLeader+Precedence					
	Rowwise			MinMin			Rowwise			MinMin		
	$\#s$	$\#f$	t	$\#s$	$\#f$	t	$\#s$	$\#f$	t	$\#s$	$\#f$	t
3 6 4	23,363,134	**0**	35.85	23,363,134	15,592	38.19	**6,853,275**	1,269	11.43	**6,853,275**	**0**	**10.69**
4 4 5	64,568,170	**0**	92.95	64,568,170	11,628	98.61	**24,012,832**	1,357	35.66	**24,012,832**	**0**	**35.61**
4 5 4	436,444,129	**0**	680.27	436,444,129	323,383	723.79	**128,960,546**	21,025	218.50	**128,960,546**	**0**	**204.24**
3 6 5	567,706,858	**0**	855.20	567,706,858	120,776	876.66	**109,194,518**	4,878	172.69	**109,194,518**	**0**	**169.58**

Table 2: Error Correcting Code-Lee Distance

n,c,b	LexLeader						ReflexLeader					
	Rowwise			MinMin			Rowwise			MinMin		
	$\#s$	$\#f$	t	$\#s$	$\#f$	t	$\#s$	$\#f$	t	$\#s$	$\#f$	t
5 6 5	839,073	3,275K	107.32	839,073	9,273K	432.77	**411,138**	1,690K	67.24	**411,138**	**1,688K**	**66.79**
6 4 4	2,789,051	2,622K	107.94	2,789,051	4,556K	190.55	**1,545,486**	1,969K	86.38	**1,545,486**	**1,609K**	**67.72**
8 4 4	20,826,947	30,807K	1,237.29	20,826,947	48,505K	2,122.27	**11,112,142**	30,168K	1,552.85	**11,112,142**	**16,124K**	**671.73**
6 4 5	15,439,833	39,964K	1,618.05	15,439,833	84,670K	4,214.56	**7,771,675**	27,916K	1,322.90	**7,771,675**	**22,745K**	**1,006.53**

as multisets, are ordered. All static symmetry breaking methods on column (row) symmetries are safe to combine with multiset ordering constriants on row (column) symmetries.

Theorem 12. *Suppose M is a static symmetry breaking method. In matrix problems, multiset ordering the rows (columns) and using M to break the columns (rows) interchangeability are sound no matter what variable and value orders are used in each method.*

Proof. Since interchanging columns (rows) does not change the occurrences of values in each row (column), their combination is sound. □

More specifically, the reflex ordering constraint is safe to combine with the multiset ordering constraint.

Corollary 3. *In matrix problems, multiset ordering the rows (columns) and reflex ordering the columns (rows) are sound regardless the variable and value orders used.*

Proposition 1 implies that reflex and lex orderings can conflict with each other. Thus ReflexLeader cannot be combined with LexLeader in general.

6 Experimental Results

This section gives four experiments. All static symmetry breaking constraints are posted based on the rowwise variable order and min value order. We compare ReflexLeader with the efficient and widely used static method LexLeader *given the same subset of symmetries*. All reflex and lex ordering constraints in the experiments can be simplified to constraints on two disjoint and same sized variable vectors.

In searching, we first use the standard rowwise variable order (Rowwise) and min value order. Other orderings, e.g. snake ordering, are also applicable. We skip them for lack of space. We also exploit the smallest min value first (MinMin) variable order and min value order, which align well with ReflexLeader since it always chooses the min value among the values in the domains of all variables to branch.

All experiments are conducted using Gecode Solver 4.2.0 on Intel C2D E8400 3.0Ghz processors with 7GB. In our tables, $\#s$ denotes the number of solutions, $\#f$ denotes the number of failures and t denotes the runtimes. For the last two experiments, an entry with the symbol "-" indicates that the search timed out after the 1-hour limit. The best results are highlighted in **bold**.

Matrix Symmetries. To compare ReflexLeader and LexLeader without effects of other factors, we first solve the unconstrained matrix problem (n, m, d) which contains only $n \times m$ variables but no constraints. Error Correcting Code-Lee Distance (ECCLD) problems with parameters (n, c, b) are also tested using the same model by Lee and Zhu [2014]. Both problems have matrix symmetries. We thus apply ReflexLeader and LexLeader to break the same subset of matrix symmetries used by Lee and Zhu [2015]. Note that [Lee and Zhu, 2015] breaking this subset of symmetries by LexLeader performs better than DOUBLELEX which has similar performance with SNAKELEX [Katsirelos *et al.*, 2010], so that their results are not reported here. The unconstrained matrix problem also has value interchangeability and Precedence can thus be exploited.

In the unconstrained matrix problem (Table 1), the solution set size for ReflexLeader is on average 29% of that of LexLeader. In ECCLD (Table 2), the solution set size for ReflexLeader is reduced by 52% on average. This demonstrates ReflexLeader (and its combination with Precedence) can break more composition symmetries than that of LexLeader (and its combination with Precedence). The runtimes are 3.45 and 3.76 times faster and 1.22 and 4.16 times faster on average under the two orderings for the two problems respectively. ReflexLeader under MinMin runs 1.66 times faster than LexLeader under Rowwise in ECCLD. Note that ReflexLeader and LexLeader have no failures under their respective search orders in the unconstrained matrix problem.

Geometric Symmetries. NNQueen [Kelsey *et al.*, 2004] and Diagonal Latin Square [Harvey, 2005] of size n are tested. Both are modeled with n^2 variables and both have 8 geometric symmetries, value interchangeability and their compositions. We first use the VAR constraints [Puget, 2005a; 2005b], which are simplified from LexLeader, and

Table 3: NNQueens

n	VAR+Precedence						ReflexLeader+Precedence					
	Rowwise			MinMin			Rowwise			MinMin		
	$\#s$	$\#f$	t	$\#s$	$\#f$	t	$\#s$	$\#f$	t	$\#s$	$\#f$	t
7	4	1,070	0.02	-	-	-	**2**	**749**	**0.02**	-	-	-
8	**0**	207,048	2.61	-	-	-	**0**	**130,383**	**1.66**	-	-	-
9	**0**	232,273,997	3,300.56	-	-	-	**0**	**141,371,236**	**2,015.99**	-	-	-

Table 4: Diagonal Latin Square

n	VAR+Precedence						ReflexLeader+Precedence					
	Rowwise			MinMin			Rowwise			MinMin		
	$\#s$	$\#f$	t	$\#s$	$\#f$	t	$\#s$	$\#f$	t	$\#s$	$\#f$	t
6	128	3,000	0.02	128	128,077,130	467.85	**64**	**2,034**	**0.02**	**64**	71,959,952	266.81
7	171,200	1,412,605	11.36	-	-	-	**85,600**	**924,584**	**7.93**	-	-	-

Precedence. Note that *Precedence eliminates all variable symmetries as it specifies that the first row must be* $\langle 1, ..., n \rangle$. And the VAR constraints cannot further delete symmetric solutions. ReflexLeader on symmetry $d1$ (reflection on the diagonal), however, can further prune symmetric solutions. We thus use ReflexLeader to break $d1$ and also Precedence to break value interchangeability.

In Tables 3 and 4, after posting ReflexLeader on $d1$, half of the solutions of VAR+Precedence is further eliminated. The runtimes are 1.64 and 1.43 times faster for the largest instance under Rowwise for the two problems respectively. This shows again the combination of ReflexLeader and Precedence breaks more composition symmetries than that of LexLeader and Precedence. MinMin happens to conflict with the problem constraints. Although MinMin is aligned with ReflexLeader, our method has the flexibility of resorting to other search orders that cooperate well with problem constraints, in this case Rowwise. This shows the advantage of static symmetry breaking methods: to be able to combine with other search heuristics.

Discussions. The experiments show that ReflexLeader can break more composition symmetries in partial symmetry breaking than LexLeader can. The reason is that the canonical solutions chosen by different ReflexLeader constraints vary a lot more than those of LexLeader. Thus the intersection of ReflexLeader's canonical solutions is smaller in size than that of LexLeader. We also show that the combination of ReflexLeader and Precedence can also break more composition symmetries in partial symmetry breaking than that of LexLeader and Precedence. The reason is the canonical solutions chosen by ReflexLeader have less intersections with those chosen by Precedence. Thus ReflexLeader constraints have better interactions amongst themselves as well as with the Precedence constraint.

Multiset ordering constraints alone do not perform well to break symmetries in matrix problems [Frisch *et al.*, 2009] since multiset ordering is only a partial order. People often combine multiset ordering constraints with lex ordering constraints. We show no results of the combinations of multiset ordering constraint with the two total ordering constraints due to space limitation. When combined with multiset ordering on the rows (columns), ReflexLeader and LexLeader can only post symmetries on columns (rows) interchangeability. This means only a linear and small number of symmetries are given to them. Results shows their combinations with multiset ordering constraints result in the same number of solutions for ECCLD. ReflexLeader is only slightly better than LexLeader for the unconstrained matrix problem when combined with multiset ordering constraints and Precedence. When given their aligned heuristics, the partial ReflexLeader alone runs 1.45 and 2.72 times faster than the combination of LexLeader and multiset ordering constraints on average for ECCLD and the unconstrained matrix problem (breaking also value interchangeability) respectively.

We also do not compare the performance of DOUBLEREFLEX and DOUBLELEX for matrix problems for two reasons. First, Lee and Zhu [2015] show that the subset of symmetries we give to ReflexLeader and LexLeader in here is better than adjacent rows and columns interchangeability, with which ReflexLeader and LexLeader are simplified to DOUBLEREFLEX and DOUBLELEX respectively. Second, with more symmetries posted, the difference on the performances of ReflexLeader and LexLeader is more prominent.

ReflexLeader has the advantage to prune more composition symmetries in partial symmetry breaking. The MinMin heuristic matches well with the propagation of reflex ordering constraints. However, MinMin is not a good heuristic to solve many CSP problems. Often, it does not cooperate well with the problem constraints. When the reduction on the search tree size is small and the problem constraints are in conflict with MinMin, using ReflexLeader and MinMin to solve the problem would give no benefits.

7 Conclusion

Our contributions are four fold. First, we introduce the reflex total ordering on integer vectors and compare it with lex. Second, we present the reflex ordering constraint and give an efficient GAC algorithm. Third, we propose a new static symmetry breaking method ReflexLeader and give conditions when ReflexLeader is safe to combine with the Precedence and multiset ordering constraints. Fourth, we give experimentations to show substantial advantages of the reflex ordering over the lex ordering in partial symmetry breaking.

Acknowledgement

We are grateful to the anonymous referees of IJCAI-16 for their comments. This research has been supported by the

grant CUHK413713 from the Research Grants Council of Hong Kong SAR.

References

[Crawford *et al.*, 1996] J. Crawford, M. Ginsberg, E. Luks, and A. Roy. Symmetry breaking predicates for search problems. In *KR'96*, pages 148–159, 1996.

[Fahle *et al.*, 2001] T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In *CP'01*, pages 93–107, 2001.

[Flener *et al.*, 2001] P. Flener, A.M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, and T. Walsh. Matrix modelling. In *ModRef'01*, 2001.

[Flener *et al.*, 2002] P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. In *CP'02*, pages 187–192, 2002.

[Frisch *et al.*, 2002] Alan Frisch, Brahim Hnich, Zeynep Kiziltan, Ian Miguel, and Toby Walsh. Global constraints for lexicographic orderings. In *CP'02*, pages 93–108, 2002.

[Frisch *et al.*, 2009] Alan M Frisch, Brahim Hnich, Zeynep Kiziltan, Ian Miguel, and Toby Walsh. Filtering algorithms for the multiset ordering constraint. *Artificial Intelligence*, 173(2):299–328, 2009.

[Gent and Smith, 2000] I.P. Gent and B.M. Smith. Symmetry breaking in constraint programming. In *ECAI'00*, pages 599–603, 2000.

[Grayland *et al.*, 2009] Andrew Grayland, Ian Miguel, and Colva M Roney-Dougal. Snake Lex: An Alternative to Double Lex. In *CP'09*, pages 391–399, 2009.

[Harvey, 2005] Warwick Harvey. Symmetric relaxation techniques for constraint programming. *Almost-Symmetry in Search*, pages 50–59, 2005.

[Katsirelos *et al.*, 2010] George Katsirelos, Nina Narodytska, and Toby Walsh. On the complexity and completeness of static constraints for breaking row and column symmetry. In *CP'10*, pages 305–320, 2010.

[Kelsey *et al.*, 2004] Tom Kelsey, Steve Linton, and Colva Roney-Dougal. New developments in symmetry breaking in search using computational group theory. In *Artificial Intelligence and Symbolic Computation*, pages 199–210, 2004.

[Law and Lee, 2004] Yat Chiu Law and J.H.M. Lee. Global constraints for integer and set value precedence. In *CP'04*, pages 362–376, 2004.

[Lee and Li, 2012] Jimmy Lee and Jingying Li. Increasing symmetry breaking by preserving target symmetries. In *CP'12*, pages 422–438, 2012.

[Lee and Zhu, 2014] J.H.M. Lee and Z. Zhu. Boosting SBDS for partial symmetry breaking in constraint programming. In *AAAI'14*, pages 2695–2702, 2014.

[Lee and Zhu, 2015] J.H.M. Lee and Z. Zhu. Filtering nogoods lazily in dynamic symmetry breaking during search. In *IJCAI'15*, pages 339–345, 2015.

[Lee and Zhu, 2016] J.H.M. Lee and Z. Zhu. Breaking more composition symmetries using search heuristics. In *AAAI'16*, 2016.

[Narodytska and Walsh, 2013] Nina Narodytska and Toby Walsh. Breaking symmetry with different orderings. In *CP'13*, pages 545–561, 2013.

[Puget, 2005a] J. Puget. Breaking symmetries in all different problems. In *IJCAI'05*, pages 272–277, 2005.

[Puget, 2005b] Jean-Francois Puget. Elimination des symétries dans les problemes injectifs. In *Premières Journées Francophones de Programmation par Contraintes*, 2005.

[Roney-Dougal *et al.*, 2004] Colva M Roney-Dougal, Ian P Gent, Tom Kelsey, and Steve Linton. Tractable symmetry breaking using restricted search trees. In *ECAI'04*, pages 211–215, 2004.

[Rossi *et al.*, 2006] F. Rossi, P. Van Beek, and T. Walsh. *Handbook of constraint programming*. Elsevier, 2006.