Global Constraints for Integer and Set Value Precedence

Y.C. Law and J.H.M. Lee

Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong
{yclaw, jlee}@cse.cuhk.edu.hk

Abstract. The paper introduces *value precedence* on integer and set sequences. A useful application of the notion is in breaking symmetries of indistinguishable values, an important class of symmetries in practice. Although value precedence can be expressed straightforwardly using if-then constraints in existing constraint programming systems, the resulting formulation is inefficient both in terms of size and runtime. We present two propagation algorithms for implementing global constraints on value precedence in the integer and set domains. Besides conducting experiments to verify the feasibility and efficiency of our proposal, we characterize also the propagation level attained by various usages of the global constraints as well as the conditions when the constraints can be used consistently with other types of symmetry breaking constraints.

1 Introduction

Symmetry is a beauty in nature, but a curse to solving algorithms of constraint satisfaction problems (CSPs). This paper concerns an important class of symmetries, namely those induced by indistinguishable values, examples of which include colors in graph coloring problems and personnels of the same rank in many rostering problems. We introduce the notion of *value precedence* on sequences, and explain how imposing value precedence on a sequence of constrained variables can break symmetries of indistinguishable values in both integer and set domains.

The value precedence condition on a sequence of variables is easy to express using if-then constraints, but such a formulation is inefficient both in terms of number of constraints and propagation efficiency. We propose linear time propagation algorithms for maintaining value precedence as the basis of efficient implementation of two global constraints on integer and set variable sequences respectively. Experimental results on three benchmarks confirm the efficiency of our proposal. In addition, we give theoretical results to (a) ensure consistent use of the constraint with other symmetry breaking constraints and (b) characterize the consistency level attained by various usages of the global constraints.

Interchangeable values [5] can be exchanged for a single variable without affecting the satisfaction of constraints, while indistinguishable values in a solution assignment can be swapped to form another solution of the same problem. Gent [9] designs a special constraint which assumes that all the domain values of a set of variables are indistinguishable and breaks the symmetries among them. Our proposed constraint, however, breaks the symmetry of two indistinguishable values. Multiple constraints have to be used to tackle a set of indistinguishable values. This allows breaking symmetries where only part of the domain values of the variables are indistinguishable. Frisch *et al.* [6, 8] present global constraints for lexicographic ordering and multiset ordering on two sequences of variables, which are useful in breaking row and column symmetries in matrix models [4].

2 Background

A CSP is a triple (X, D, C), where $X = \{x_0, \ldots, x_{n-1}\}$ is a set of variables, D is a function that maps each $x \in X$ to its associated domain $D(x)$, giving the set of possible values for x, and C is a set of constraints. There are two common classes of variables in CSPs. A *constrained integer variable* (or simply *integer variable*) [11] x has an integer domain, i.e., $D(x)$ is an integer set. A *constrained set variable* (or simply *set variable*) [11] x has a set domain, i.e., each element in the domain is a set. In most implementations, the domain of a set variable x is represented by two sets. The *possible set* $PS(x)$ contains elements that belong to at least one of the possible values of x. The *required set* $RS(x)$ contains elements that belong to all the possible values of x. By definition, $RS(x) \subseteq PS(x)$. Domain reduction of a set variable x is done by removing values from $PS(x)$ and adding values to $RS(x)$. If a value being removed from $PS(x)$ is in $RS(x)$, then a fail is triggered. Similarly, adding a value to $RS(x)$ which is not in $PS(x)$ also triggers a fail. When $PS(x) = RS(x)$, the set variable is bound to its required set.

An *assignment* $x \mapsto a$ means that variable $x \in X$ of a CSP is mapped to the value $a \in D(x)$. A *compound assignment* $\langle x_{i_1}, \ldots, x_{i_k} \rangle \mapsto \langle a_1, \ldots, a_k \rangle$ denotes the k assignments $x_{i_j} \mapsto a_j$ for $1 \leq j \leq k$. Note the requirement that no variables can be assigned more than once in a compound assignment. A *complete assignment* is a compound assignment for all variables in a CSP. A *solution* of a CSP (X, D, C) is a complete assignment making all constraints in C true.

An *extension* of an assignment $x \mapsto a$ for a variable x in a constraint c is a compound assignment that includes $x \mapsto a$. A constraint c is *generalized arc consistent* (GAC) [13] if and only if every assignment $x \mapsto a$ for each variable x in c, where $a \in D(x)$, can be extended to a solution of c. GAC is prohibitive to enforce on constraints involving set variables. Instead, set bounds consistency [10] is typically enforced on these constraints. A constraint c on set variables is *set bounds consistent* [10] (SBC) if and only if the $PS(x)$ and $RS(x)$ of each set variable x in c can be extended to a solution of c.

3 Integer and Set Value Precedence

In this section, we define *value precedence* for integer and set sequences, and give its use for symmetry breaking and methods for maintaining such a constraint.

Value precedence of s over t in an integer sequence $\boldsymbol{x} = \langle x_0, \ldots, x_{n-1} \rangle$ means if there exists j such that $x_j = t$, then there must exist $i < j$ such that $x_i = s$. We say that value s is an *antecedent* while value t is a *subsequent*, and that the antecedent s *precedes* the subsequent t in $\boldsymbol{x}$, written as $s \prec_{\boldsymbol{x}} t$. For example, the sequence $\boldsymbol{x} = \langle 0, 2, 2, 1, 0, 1 \rangle$ implies $0 \prec_{\boldsymbol{x}} 1$, $0 \prec_{\boldsymbol{x}} 2$, and $2 \prec_{\boldsymbol{x}} 1$. Note that if a value j does not appear in $\boldsymbol{x}$, then $i \prec_{\boldsymbol{x}} j$ is true for any i. In the previous example, $0 \prec_{\boldsymbol{x}} 3$ and $4 \prec_{\boldsymbol{x}} 3$ are thus also true. Note that value precedence is transitive: if $i \prec_{\boldsymbol{x}} j$ and $j \prec_{\boldsymbol{x}} k$, then $i \prec_{\boldsymbol{x}} k$.

The notion of value precedence can be extended to sequences of sets, where antecedents and subsequents are elements of the sets in the sequence. *Value precedence* of s over t in a sequence $\boldsymbol{x}$ of sets means that if there exists j such that $s \notin x_j$ and $t \in x_j$, then there must exist $i < j$ such that $s \in x_i$ and $t \notin x_i$. For example, consider the sequence $\boldsymbol{x} = \langle \{0, 2\}, \{0, 1\}, \emptyset, \{1\} \rangle$. We have $0 \prec_{\boldsymbol{x}} 1$ and $2 \prec_{\boldsymbol{x}} 1$. We also have $0 \prec_{\boldsymbol{x}} 2$, because there is no set in $\boldsymbol{x}$ that contains 2 but not 0. Again, if j does not belong to any set in $\boldsymbol{x}$, then $i \prec_{\boldsymbol{x}} j$ is true for any i. Thus, we also have, say, $0 \prec_{\boldsymbol{x}} 4$.

3.1 Application of Value Precedence

Value precedence can be used for breaking symmetries of indistinguishable values. Two values s and t are *indistinguishable* [2, 9] under a subset of variables $U \subseteq X$ of a CSP P if and only if for each solution of P, swapping occurrences of s and t in the assigned values of the variables in U obtains another solution of P. For example, let $\boldsymbol{x} = \langle x_0, \ldots, x_4 \rangle$ and $\boldsymbol{u} = \langle x_1, x_2, x_3 \rangle$ be sequences of X and U respectively. Suppose $\boldsymbol{x} \mapsto \langle \{1, 2\}, \{0, 2\}, \{0, 1\}, \{1, 2\}, \emptyset \rangle$ is a solution of P, and values 0 and 1 are indistinguishable under U. Then, swapping the occurrences of 0 and 1 in the assignments of U in the solution obtains $\boldsymbol{x} \mapsto \langle \{1, 2\}, \{1, 2\}, \{0, 1\}, \{0, 2\}, \emptyset \rangle$, which should be another solution of P.

Given two indistinguishable values under U in a CSP, we can break the symmetry of the values by maintaining value precedence for them. We have to construct a sequence $\boldsymbol{u}$ of U, and assume one value to be the antecedent and the other to be the subsequent. Without loss of generality, we usually pick the smaller value as antecedent. In the previous example, we have $\boldsymbol{x} = \langle x_0, \ldots, x_4 \rangle$ and $\boldsymbol{u} = \langle x_1, x_2, x_3 \rangle$. The symmetry of the indistinguishable values 0 and 1 under U can be broken by maintaining the constraint $0 \prec_{\boldsymbol{u}} 1$ on variables x_1, x_2, and x_3 in $\boldsymbol{u}$. Thus, $\boldsymbol{x} \mapsto \langle \{1, 2\}, \{0, 2\}, \{0, 1\}, \{1, 2\}, \emptyset \rangle$ remains a solution, but its symmetrical counterpart $\boldsymbol{x} \mapsto \langle \{1, 2\}, \{1, 2\}, \{0, 1\}, \{0, 2\}, \emptyset \rangle$ would now be rejected because $0 \prec_{\boldsymbol{u}} 1$ is false.

Sometimes there can be more than two indistinguishable values in a CSP. Suppose $V = \{v_0, \ldots, v_{k-1}\}$ is a set of mutually indistinguishable values for variables in U. To break the symmetries induced by V, we can impose an arbitrary value precedence $v_{i_0} \prec_{\boldsymbol{u}} \cdots \prec_{\boldsymbol{u}} v_{i_{k-1}}$ on the values in V for $\boldsymbol{u}$, where $\boldsymbol{u}$ is a sequence of U. Suppose $V = \{0, 1, 2, 3\}$. We can maintain $0 \prec_{\boldsymbol{u}} 1 \prec_{\boldsymbol{u}} 2 \prec_{\boldsymbol{u}} 3$.

Besides symmetries of indistinguishable values, a CSP can have other types of symmetries. A *variable symmetry* of a CSP, for example, is a bijective mapping σ from the set of variables $X = \{x_0, \ldots, x_{n-1}\}$ of the CSP to itself,

$\sigma : X \rightarrow X$, such that for each solution $\boldsymbol{x} \mapsto \langle v_0, \ldots, v_{n-1}\rangle$ of the CSP where $\boldsymbol{x} = \langle x_0, \ldots, x_{n-1}\rangle$, there exists another solution $\langle\sigma(x_0), \ldots, \sigma(x_{n-1})\rangle \mapsto \langle v_0, \ldots, v_{n-1}\rangle$ for the CSP. Variable symmetries can be broken by expressing lexicographic ordering constraints $\boldsymbol{x} \leq_{lex} \langle\sigma(x_0), \ldots, \sigma(x_{n-1})\rangle$ for each variable symmetry σ in the CSP [3].[1] When tackling both variable symmetries and symmetries of indistinguishable values simultaneously in a CSP, we have to ensure that the two corresponding sets of symmetry breaking constraints are consistent. For example, we have a CSP $P = (\{x, y\}, D, \{x \neq y\})$, where $D(x) = D(y) = \{1, 2\}$. P has (a) the variable symmetry σ such that $\sigma(x) = y$ and $\sigma(y) = x$, and (b) values 1 and 2 are indistinguishable. To break symmetry (a), we can use the constraint $x \leq y$ (which is a degenerated lexicographic ordering); whereas $2 \prec_{\langle x,y\rangle} 1$ can break symmetry (b). These two constraints result in no solution, which is undesirable.

Two sets of symmetry breaking constraints are *consistent* [4] if and only if at least one element in each symmetry class of assignments, defined by the composition of the symmetries under consideration, satisfies both sets of constraints. The following theorem shows when maintaining $s \prec_{\boldsymbol{u}} t$ is consistent with variable symmetry breaking constraints.

Theorem 1. *Let X be the set of variables of a CSP, and $\boldsymbol{x}$ and $\boldsymbol{u}$ be sequences of X and $U \subseteq X$ respectively. Suppose in the CSP, C_{var} is the set of symmetry breaking constraints $\boldsymbol{x} \leq_{lex} \langle\sigma(x_0), \ldots, \sigma(x_{n-1})\rangle$ (*resp. *$\langle\sigma(x_0), \ldots, \sigma(x_{n-1})\rangle \leq_{lex} \boldsymbol{x}$) for some variable symmetries σ, and s and t are any two integer indistinguishable values under U. Maintaining $s \prec_{\boldsymbol{u}} t$ (*resp. *$t \prec_{\boldsymbol{u}} s$) is consistent with C_{var} if $s < t$ (*resp. *$t < s$) and $\boldsymbol{u}$ is a subsequence of $\boldsymbol{x}$, i.e., $\boldsymbol{u}$ can be formed by deleting some elements from $\boldsymbol{x}$.*

According to Theorem 1, $x \leq y$ and $1 \prec_{\langle x,y\rangle} 2$ (or $y \leq x$ and $2 \prec_{\langle x,y\rangle} 1$) are consistent, resulting in a single solution $\langle x, y\rangle \mapsto \langle 1, 2\rangle$ (or $\langle x, y\rangle \mapsto \langle 2, 1\rangle$).

In order for Theorem 1 to apply also to set variables, we need to define an ordering for sets. One possible definition, which is similar to that of multiset ordering [8], is as follows. A set x is *smaller than or equal to* another set y, written as $x \leq_{set} y$, if and only if (1) $y = \emptyset$, or (2) $\min(x) < \min(y)$, or (3) $\min(x) = \min(y) \rightarrow (x \setminus \{\min(x)\} \leq_{set} y \setminus \{\min(y)\})$. Note that $\emptyset$ is the largest element in the ordering. For example, $\{1, 2\} \leq_{set} \{1, 3, 4\} \leq_{set} \{1, 3\}$. Using this ordering, lexicographic ordering of set sequences becomes possible.

3.2 Constraints for Maintaining Value Precedence

Constraints to enforce value precedence $s \prec_{\boldsymbol{x}} t$ for a sequence of constrained variables $\boldsymbol{x}$ can be constructed straightforwardly from its declarative meaning. Suppose $\boldsymbol{x}$ is a sequence of integer variables. Since s must precede t, x_0, the first variable in $\boldsymbol{x}$, must not be assigned t. The constraints are then

[1] Sometimes the lexicographic ordering constraints can be simplified to contain fewer variables. An example is the row ordering and column ordering constraints for row and column symmetries [4].

1. $x_0 \neq t$ and
2. $x_j = t \rightarrow \bigvee_{0 \leq i < j} x_i = s$ for $1 \leq j < n$.

If $\boldsymbol{x}$ is a sequence of set variables, then t must not be in x_0 without being accompanied by s. Hence, the constraints are

1. $s \in x_0 \vee t \notin x_0$ and
2. $(s \notin x_j \wedge t \in x_j) \rightarrow \bigvee_{0 \leq i < j}(s \in x_i \wedge t \notin x_i)$ for $1 \leq j < n$.

Note that for both integer and set variables, we need n constraints, which we collectively call *if-then value precedence constraints*, to maintain value precedence. Among the n constraints, one is a unary constraint, and the remaining $n - 1$ are if-then constraints. The following theorem shows that for integer (*resp.* set) variables, GAC (*resp.* SBC) on the *conjunction* of the n if-then value precedence constraints is equivalent to GAC (*resp.* SBC) on each individual if-then value precedence constraint.

Theorem 2. *Given an integer (*resp. *set) variable sequence $\boldsymbol{x}$. GAC (*resp. *SBC) on $s \prec_{\boldsymbol{x}} t$ is equivalent to GAC (*resp. *SBC) on each individual if-then value precedence constraint for integer (*resp. *set) variables.*

4 Propagation Algorithms for Value Precedence

We develop two *value precedence global constraints* for maintaining value precedence, one for integer variables and the other for set variables, in ILOG Solver 4.4 [11]. Both constraints use the same prototype $ValuePrecede(\boldsymbol{x}, s, t)$, meaning $s \prec_{\boldsymbol{x}} t$, where $\boldsymbol{x}$ is a variable sequence. Note that s and t are integer constants. In particular, GAC (*resp.* SBC) is enforced on the integer (*resp.* set) value precedence constraint. The integer and set versions of the propagation algorithms, namely **IntValuePrecede** and **SetValuePrecede** respectively, are similar. Their complexity is *linear to the length of the variable sequence.* Both of them make use of three pointers, namely α, β, and γ, which point to different indices of the sequence $\boldsymbol{x}$, but the pointers have different meanings for the integer and set versions. The two algorithms are also similar to that of the lexicographic ordering global constraint [6] in the sense that both maintain pointers running in opposite directions from the two ends of variable sequences. In subsequent discussions, we assume the variable sequence $\boldsymbol{x} = \langle x_0, \ldots, x_{n-1} \rangle$.

4.1 Integer Version

In **IntValuePrecede**, pointer α is the smallest index of $\boldsymbol{x}$ such that s is in the domain of x_α, i.e., $s \in D(x_\alpha)$ and $s \notin D(x_i)$ for $0 \leq i < \alpha$. If no variables in $\boldsymbol{x}$ have value s in their domains, then we define that $\alpha = n$. Pointer β is the second smallest index of $\boldsymbol{x}$ such that s is in the domain of x_β, i.e., $s \in D(x_\beta)$ and $s \notin D(x_i)$ for $\alpha < i < \beta$. If no or only one variable in $\boldsymbol{x}$ contain value s in their domains, then we define that $\beta = n$. Pointer γ is the smallest index of $\boldsymbol{x}$ such that x_γ is bound to t, i.e., $D(x_\gamma) = \{t\}$ and $D(x_i) \neq \{t\}$ for $0 \leq i < \gamma$. If no variables in $\boldsymbol{x}$ are bound to t, then we define that $\gamma = n$. During propagation, α and β must be increasingly updated, while γ must be decreasingly updated. For example, let $\boldsymbol{x} = \langle x_0, x_1, x_2, x_3 \rangle$, $s = 1$, and $t = 2$. Suppose we have:

$\boldsymbol{x}$	x_0	x_1	x_2	x_3
$D(x_i)$	$\{2,3\}$	$\{1,2,3\}$	$\{2\}$	$\{1,3\}$

Then, we have $\alpha = 1$, $\beta = 3$, and $\gamma = 2$.

Recall that the integer if-then value precedence constraints are $x_0 \neq t$ and $x_j = t \rightarrow \bigvee_{0 \leq i < j} x_i = s$ for $1 \leq j < n$. Pointer α tells that $x_i \neq s$ for $0 \leq i < \alpha$. Hence, we must have $x_i \neq t$ for $0 \leq i \leq \alpha$. Our first pruning rule is that:

1. *value t can be removed from the domains of the variables before or at position α in $\boldsymbol{x}$.*

In the above example, we have $\alpha = 1$. Therefore, we can remove value 2 from the domains of x_0 and x_1 as shown in Fig. 1(a).

$\boldsymbol{x}$	x_0	x_1	x_2	x_3
$D(x_i)$	$\{3\}$	$\{1,3\}$	$\{2\}$	$\{1,3\}$
		$\uparrow \alpha$	$\uparrow \gamma$	$\uparrow \beta$

(*a*)

$\boldsymbol{x}$	x_0	x_1	x_2	x_3
$D(x_i)$	$\{3\}$	$\{1\}$	$\{2\}$	$\{1,3\}$
		$\uparrow \alpha$	$\uparrow \gamma$	$\uparrow \beta$

(*b*)

Fig. 1. Illustrating the use of the pointers α, β, and γ in **IntValuePrecede**

Pointer γ tells the smallest index of $\boldsymbol{x}$ such that x_γ is bound to t. Therefore, according to the if-then value precedence constraints, $\bigvee_{0 \leq i < \gamma} x_i = s$ must be satisfied. Since $x_i \neq s$ for $0 \leq i < \alpha$, $\bigvee_{0 \leq i < \gamma} x_i = s$ can be refined to $\bigvee_{\alpha \leq i < \gamma} x_i = s$. Furthermore, pointer β tells that $x_i \neq s$ for $\alpha < i < \beta$. Therefore, if $\gamma < \beta$, then $\bigvee_{\alpha \leq i < \gamma} x_i = s$ becomes $x_\alpha = s$. Our second pruning rule is that:

2. *if $\gamma < \beta$, then x_α can be bound to s.*

Note that once x_α is bound to s, $s \prec_{\boldsymbol{x}} t$ is satisfied. In the above example, we have $3 = \beta > \gamma = 2$. Therefore, we can bound x_1 (x_α) to 1, as shown in Fig. 1(b), and 1 must precede 2 in $\boldsymbol{x}$ afterwards.

The propagation algorithm **IntValuePrecede**, shown in Fig. 2, is based on the two pruning rules just described. Procedure *initialize*() is called when a value precedence constraint is posted. It finds initial values for the pointers α, β, and γ. In the procedure, we first search the position for α, starting from position 0. During the search, the first pruning rule is applied. After that, we search a value for γ. Since value t is removed from $D(x_i)$ for $0 \leq i \leq \alpha$, γ must be greater than α and the position search for γ can start from position $\alpha + 1$. Note that the second pruning rule cannot be applied at this point because pointer β is not yet initialized. After fixing γ, procedure *updateBeta*() is invoked to find a value for β. By definition, β must be greater than α. Therefore the search starts from position $\alpha + 1$. After fixing β, the second pruning rule can be applied.

```
procedure initialize()
α := 0;
while α < n ∧ s ∉ D(x_α) do D(x_α) := D(x_α) \ {t}; α := α + 1 endwhile
β := α; γ := α;
if α < n then
   D(x_α) := D(x_α) \ {t};
   repeat γ := γ + 1 until γ = n ∨ D(x_γ) = {t};
   updateBeta()
endif

procedure propagate(i)
if β ≤ γ then
   if i = α ∧ s ∉ D(x_i) then
      α := α + 1;
      while α < β do D(x_α) := D(x_α) \ {t}; α := α + 1 endwhile
      while α < n ∧ s ∉ D(x_α) do D(x_α) := D(x_α) \ {t}; α := α + 1 endwhile
      if α < n then D(x_α) := D(x_α) \ {t} endif
      β := α;
      if α < n then updateBeta() endif
   else if i = β ∧ s ∉ D(x_i) then
      updateBeta()
   endif
endif

procedure updateBeta()
repeat β := β + 1 until β = n ∨ s ∈ D(x_β);
if β > γ then D(x_α) := D(x_α) ∩ {s} endif

procedure checkGamma(i)
if β < γ ∧ i < γ ∧ D(x_i) = {t} then
   γ := i;
   if β > i then D(x_α) := D(x_α) ∩ {s} endif
endif
```

Fig. 2. The **IntValuePrecede** propagation algorithm

Procedure *propagate*(i) in Fig. 2 is called whenever the domain of x_i is modified. If $\gamma < \beta$, then value precedence is already maintained and we can skip the propagation process. Otherwise, if $i = \alpha$ and $s \notin D(x_i)$, then pointers α and β have to be updated. The search for new position for α starts from position β, because x_β is the original second earliest variable that contains s in its domain. Once value s is removed from $D(x_\alpha)$, β becomes the first potential value for α. However, before the search, value t has to be removed from $D(x_i)$ for $\alpha < i < \beta$. During the search, the first pruning rule is applied. Pointer β is updated after finding a new value for α. The search for new value for β starts from position $\alpha + 1$. The procedure *updateBeta*() is called to update β. In the procedure, once β is updated, the second pruning rule is applied to check whether $\beta > \gamma$.

In procedure *propagate*(i), if $i = \beta$ and $s \notin D(x_i)$, then only pointer β has to be updated. Hence, the procedure *updateBeta*() is called to find a new value for β and to apply the second pruning rule.

We need a procedure to update γ also. Procedure *checkGamma*(i) in Fig. 2 is called whenever x_i is bound to a value. If $i < \gamma$ and x_i is bound to t, then γ is updated to i, and the second pruning rule is applied to check whether $\beta > \gamma$. The **IntValuePrecede** algorithm enforces GAC on $s \prec_{\boldsymbol{x}} t$.

Theorem 3. *Given an integer variable sequence* $\boldsymbol{x}$ *and integers* s *and* t. *The* **IntValuePrecede** *algorithm triggers failure if* $s \prec_{\boldsymbol{x}} t$ *is unsatisfiable; otherwise, the algorithm prunes values from domains of variables in* $\boldsymbol{x}$ *such that GAC on* $s \prec_{\boldsymbol{x}} t$ *is enforced and solutions of* $s \prec_{\boldsymbol{x}} t$ *are preserved.*

4.2 Set Version

In **SetValuePrecede**, the meanings of the pointers α, β, and γ are different from those in the integer version. Pointer α is the smallest index of $\boldsymbol{x}$ such that s is in the possible set of x_α and t is *not* in the required set of x_α, i.e., $s \in PS(x_\alpha) \wedge t \notin RS(x_\alpha)$ and $s \notin PS(x_i) \vee t \in RS(x_i)$ for $0 \leq i < \alpha$. If $s \notin PS(x_i) \vee t \in RS(x_i)$ for $0 \leq i < n$, then we define that $\alpha = n$. Pointer β is the second smallest index of $\boldsymbol{x}$ such that s is in the possible set of x_β and t is *not* in the required set of x_β, i.e., $s \in PS(x_\beta) \wedge t \notin RS(x_\beta)$ and $s \notin PS(x_i) \vee t \in RS(x_i)$ for $\alpha < i < \beta$. If $\alpha = n$ or $s \notin PS(x_i) \vee t \in RS(x_i)$ for $\alpha < i < n$, then we define that $\beta = n$. Pointer γ is the smallest index of $\boldsymbol{x}$ such that s is *not* in the possible set of x_γ and t is in the required set of x_γ, i.e., $s \notin PS(x_\gamma) \wedge t \in RS(x_\gamma)$ and $s \in PS(x_i) \vee t \notin RS(x_i)$ for $0 \leq i < \gamma$. The definition of γ implies $s \notin x_\gamma \wedge t \in x_\gamma$. If $s \in PS(x_i) \vee t \notin RS(x_i)$ for all $0 \leq i < n$, then we define that $\gamma = n$. As in the integer version, α and β must be updated increasingly, while γ must be updated decreasingly. Let $\boldsymbol{x} = \langle x_0, x_1, x_2, x_3 \rangle$, $s = 1$, and $t = 2$. Suppose we have:

$\boldsymbol{x}$	x_0	x_1	x_2	x_3	x_4
$PS(x_i)$	$\{2,3\}$	$\{1,2\}$	$\{1,2,3\}$	$\{2,3\}$	$\{1,2\}$
$RS(x_i)$	$\emptyset$	$\{2\}$	$\{3\}$	$\{2,3\}$	$\emptyset$

Then, we have $\alpha = 2$, $\beta = 4$, and $\gamma = 3$.

Pointer α tells that $s \notin x_i \vee t \in x_i$ for $0 \leq i < \alpha$. Hence, according to the set if-then value precedence constraints $s \in x_0 \vee t \notin x_0$ and $(s \notin x_j \wedge t \in x_j) \rightarrow \bigvee_{0 \leq i < j}(s \in x_i \wedge t \notin x_i)$ for $1 \leq j < n$, the constraints $s \in x_i \vee t \notin x_i$ for $0 \leq i \leq \alpha$ must be satisfied. Since $s \in PS(x_\alpha) \wedge t \notin RS(x_\alpha)$ must be true. Therefore $s \in x_\alpha \vee t \notin x_\alpha$ is already consistent. Consequently, our first pruning rule for **SetValuePrecede** is to maintain consistency on $s \in x_i \vee t \notin x_i$ for $0 \leq i < \alpha$.

1. *For* $0 \leq i < \alpha$, *if* s *is not in* $PS(x_i)$, *then* t *can be removed from* $PS(x_i)$; *otherwise,* s *can be added to* $RS(x_i)$.

In the above example, value 1 is not in $PS(x_0)$ so that we can remove 2 from $PS(x_0)$. Value 2 is in $PS(x_1)$; thus 1 is added to $RS(x_1)$. The resulting domains are shown in Fig. 3(a).

Pointer γ tells that $s \notin x_\gamma \wedge t \in x_\gamma$. According to the if-then value precedence constraints, the constraint $\bigvee_{0 \leq i < \gamma}(s \in x_i \wedge t \notin x_i)$ must be satisfied. By

$\boldsymbol{x}$	x_0	x_1	x_2	x_3	x_4
$PS(x_i)$	$\{3\}$	$\{1,2\}$	$\{1,2,3\}$	$\{2,3\}$	$\{1,2\}$
$RS(x_i)$	$\emptyset$	$\{1,2\}$	$\{3\}$	$\{2,3\}$	$\emptyset$
			$\uparrow \alpha$	$\uparrow \gamma$	$\uparrow \beta$

(a)

$\boldsymbol{x}$	x_0	x_1	x_2	x_3	x_4
$PS(x_i)$	$\{3\}$	$\{1,2\}$	$\{1,3\}$	$\{2,3\}$	$\{1,2\}$
$RS(x_i)$	$\emptyset$	$\{1,2\}$	$\{1,3\}$	$\{2,3\}$	$\emptyset$
			$\uparrow \alpha$	$\uparrow \gamma$	$\uparrow \beta$

(b)

Fig. 3. Illustrating the use of the pointers α, β, and γ in **SetValuePrecede**

the meaning of α, this constraint can be refined to $\bigvee_{\alpha \leq i < \gamma}(s \in x_i \wedge t \notin x_i)$. Furthermore, pointer β tells that $s \notin x_i \vee t \in x_i$ for $\alpha < i < \beta$. Therefore, if $\gamma < \beta$, then $\bigvee_{\alpha \leq i < \gamma}(s \in x_i \wedge t \notin x_i)$ becomes $s \in x_\alpha \wedge t \notin x_\alpha$. Our second pruning rule for **SetValuePrecede** is that:

2. *if $\gamma < \beta$, then s can be added to $RS(x_\alpha)$ and t can be removed from $PS(x_\alpha)$.*

The constraint $s \prec_{\boldsymbol{x}} t$ is satisfied once x_α is proved to contain s but not t. In the above example, we have $3 = \gamma < \beta = 4$. Therefore, value 1 can be added to $RS(x_\alpha)$ and 2 can be removed from $PS(x_\alpha)$, as shown in Fig. 3(b).

Like **IntValuePrecede**, the **SetValuePrecede** algorithm in Fig. 4 is based on two pruning rules. It contains four procedures with the same name as the integer version, and with similar structures also. Procedure *initialize*() is called when $ValuePrecede(\boldsymbol{x}, s, t)$ is posted. It initializes the pointers α, β, and γ. The two pruning rules are applied during initialization. Procedure $propagate(i)$ is called whenever the domain of variable x_i is modified, i.e., either $PS(x_i)$ or $RS(x_i)$ is modified. If $\gamma < \beta$, value precedence is already maintained and no propagation is needed. Otherwise, there are two different cases. In the first case, $i = \alpha \wedge (s \notin PS(x_i) \vee t \in RS(x_i))$, pointers α and β have to be updated. In the second case, $i = \beta \wedge (s \notin PS(x_i) \vee t \in RS(x_i))$, only pointer β has to be updated. After these two cases, procedure $checkGamma(i)$ is called to check whether pointer γ has to be updated. This is different from the integer version, where $checkGamma(i)$ is called only when x_i is bound to a value. This is because, in the set version, pointer γ may need update even when x_i is not bound. The **SetValuePrecede** algorithm enforces SBC on $s \prec_{\boldsymbol{x}} t$.

Theorem 4. *Given a set variable sequence $\boldsymbol{x}$ and integers s and t. The* **SetValuePrecede** *algorithm triggers failure if $s \prec_{\boldsymbol{x}} t$ is unsatisfiable; otherwise, the algorithm prunes values from domains of variables in $\boldsymbol{x}$ such that SBC on $s \prec_{\boldsymbol{x}} t$ is enforced and solutions of $s \prec_{\boldsymbol{x}} t$ are preserved.*

5 Multiple Indistinguishable Values

In many circumstances, there are more than two indistinguishable values in the same problem, but our global constraints can deal with only two such values at a time. To break symmetries on a set of variables U induced by a set of indistinguishable values $V = \{v_0, \ldots, v_{k-1}\}$ for $k > 2$, we can impose the $ValuePrecede()$ constraints using *all* pairs of values in V: $v_i \prec_{\boldsymbol{u}} v_j$ for

procedure *initialize*()
$\alpha := 0$;
while $\alpha < n \wedge (s \notin PS(x_\alpha) \vee t \in RS(x_\alpha))$ **do**
  if $s \notin PS(x_\alpha)$ **then** $PS(x_\alpha) := PS(x_\alpha) \setminus \{t\}$ **else** $RS(x_\alpha) := RS(x_\alpha) \cup \{s\}$ **endif**
  $\alpha := \alpha + 1$
endwhile
$\beta := \alpha$; $\gamma := \alpha$;
if $\alpha < n$ **then**
  repeat $\gamma := \gamma + 1$ **until** $\gamma = n \vee (s \notin PS(x_\gamma) \wedge t \in RS(x_\gamma))$;
  updateBeta()
endif

procedure *propagate*(i)
if $\beta \leq \gamma$ **then**
  if $i = \alpha \wedge (s \notin PS(x_i) \vee t \in RS(x_i))$ **then**
    if $s \notin PS(x_i)$ **then** $PS(x_i) := PS(x_i) \setminus \{t\}$ **else** $RS(x_i) := RS(x_i) \cup \{s\}$ **endif**
    $\alpha := \alpha + 1$;
    while $\alpha < \beta$ **do**
      if $s \notin PS(x_\alpha)$ **then** $PS(x_\alpha) := PS(x_\alpha) \setminus \{t\}$ **else** $RS(x_\alpha) := RS(x_\alpha) \cup \{s\}$ **endif**
      $\alpha := \alpha + 1$
    endwhile
    while $\alpha < n \wedge (s \notin PS(x_\alpha) \vee t \in RS(x_\alpha))$ **do**
      if $s \notin PS(x_\alpha)$ **then** $PS(x_\alpha) := PS(x_\alpha) \setminus \{t\}$ **else** $RS(x_\alpha) := RS(x_\alpha) \cup \{s\}$ **endif**
      $\alpha := \alpha + 1$
    endwhile
    $\beta := \alpha$;
    if $\alpha < n$ **then** *updateBeta*() **endif**
  else if $i = \beta \wedge (s \notin PS(x_i) \vee t \in RS(x_i))$ **then**
    updateBeta()
  endif
  checkGamma(i)
endif

procedure *updateBeta*()
repeat $\beta := \beta + 1$ **until** $\beta = n \vee (s \in PS(x_\beta) \wedge t \notin RS(x_\beta))$;
if $\beta > \gamma$ **then** $PS(x_\alpha) := PS(x_\alpha) \setminus \{t\}$; $RS(x_\alpha) := RS(x_\alpha) \cup \{s\}$ **endif**

procedure *checkGamma*(i)
if $\beta < \gamma \wedge i < \gamma \wedge s \notin PS(x_i) \wedge t \in RS(x_i)$ **then**
  $\gamma := i$;
  if $\beta > i$ **then** $PS(x_\alpha) := PS(x_\alpha) \setminus \{t\}$; $RS(x_\alpha) := RS(x_\alpha) \cup \{s\}$ **endif**
endif

Fig. 4. The **SetValuePrecede** propagation algorithm

$0 \leq i < j \leq k-1$, where $\boldsymbol{u}$ is a sequence of U. By transitivity of value precedence, however, an alternative is to impose constraints using only *adjacent* pairs of values in V: $v_i \prec_{\boldsymbol{u}} v_{i+1}$ for $0 \leq i \leq k-2$. Although achieving the same value precedence effect, the two approaches differ in the level of propagation.

Theorem 5. *Given an integer (*resp. *set) variable sequence $\boldsymbol{u}$, and a set of integer indistinguishable values $V = \{v_0, \ldots, v_{k-1}\}$ under U. GAC (*resp. *SBC)* *on* $v_i \prec_{\boldsymbol{u}} v_j$ *for* $0 \leq i < j \leq k-1$ *is strictly stronger than GAC* (resp. *SBC) on* $v_i \prec_{\boldsymbol{u}} v_{i+1}$ *for* $0 \leq i \leq k-2$.

For example, consider the variable sequence $\boldsymbol{x} = \langle x_0, \ldots, x_3 \rangle$ with $D(x_0) = \{0, 3\}$, $D(x_1) = \{1, 3\}$, $D(x_2) = \{1, 2, 3\}$, and $D(x_3) = \{2\}$. Suppose $V = \{0, 1, 2\}$ is a set of indistinguishable values under $\{x_0, \ldots, x_3\}$. The constraints $\{0 \prec_{\boldsymbol{x}} 1, 1 \prec_{\boldsymbol{x}} 2\}$ are GAC with respect to the current variable domains, but the constraints $\{0 \prec_{\boldsymbol{x}} 1, 1 \prec_{\boldsymbol{x}} 2, 0 \prec_{\boldsymbol{x}} 2\}$ are not, since $x_0 \mapsto 3$ cannot be extended to a solution. Suppose $\boldsymbol{y} = \langle y_0, \ldots, y_3 \rangle$ is a sequence of set variables with $PS(y_0) = \{0\}, PS(y_1) = \{1\}, PS(y_2) = PS(y_3) = \{1, 2\}, RS(y_0) = RS(y_1) = RS(y_2) = \emptyset$, and $RS(y_3) = \{2\}$. The constraints $\{0 \prec_{\boldsymbol{y}} 1, 1 \prec_{\boldsymbol{y}} 2\}$ are SBC with respect to the variable domains, but $\{0 \prec_{\boldsymbol{y}} 1, 1 \prec_{\boldsymbol{y}} 2, 0 \prec_{\boldsymbol{y}} 2\}$ are not, since $y_0 \mapsto RS(y_0)$, i.e., $y_0 \mapsto \emptyset$, cannot be extended to a solution.

As we shall see in the experimental results, such difference in propagation level, although theoretically possible, might not show up often in practice.

6 Experiments

To demonstrate the feasibility of our proposal, we test our implementations on the Schur's lemma and the social golfer problem. The experiments aim to compare (a) the effect of all-pair and adjacent-pair posting of the global constraints and (b) our global constraints against the use of if-then value precedence constraints. We report also the results of another of our recently developed approach to maintain value precedence using multiple viewpoints and channeling [12].

All the experiments are run using ILOG Solver 4.4 [11] on a Sun Blade 1000 workstation with 2GB memory. We report the number of fails and CPU time for each instance of each model. The best number of fails and CPU time among the models for each instance are highlighted in bold.

6.1 Schur's Lemma

Schur's lemma, "prob015" in CSPLib,[2] is the problem of putting n balls labeled $\{1, \ldots, n\}$ into three boxes such that for any triple of balls (x, y, z) with $x + y = z$, not all are in the same box. This problem has a solution if $n < 14$. We experiment with a variant of this problem, where the triple (x, y, z) must consist of distinct values to relax the unsatisfiability condition. To model the problem into CSPs, we use variables $\boldsymbol{x} = \langle x_1, \ldots, x_n \rangle$ all with domain $\{1, 2, 3\}$, where the variables and domain values represent the balls and the boxes respectively. In this representation, the domain values 1, 2, and 3 are indistinguishable, and we can use the value precedence constraint to break the symmetries. In order to increase the difficulty of the problem, we "glue" two copies of the same instance together to form a larger instance. Suppose $P = (X, D_X, C_X)$ is a Schur's lemma problem. We replicate a copy of P and systematically replace all variables in X by variables in Y such that $X \cap Y = \emptyset$, yielding $P' = (Y, D_Y, C_Y)$ which

[2] Available at `http://www.csplib.org`.

Table 1. Experimental Results for the Schur's Lemma

	adj-pair		all-pair		if-then		int-bool		int-set	
n	fails	time	fails	time	fails	time	fails	time	fails	time
7	**130**	**0.11**	**130**	0.11	**130**	0.18	449	0.24	2232	0.57
8	**811**	**0.52**	**811**	0.54	**811**	0.91	1489	1.04	5478	2.4
9	**8506**	**1.87**	**8506**	1.97	**8506**	3.57	9733	3.64	17093	8.4
10	**38373**	**6.13**	**38373**	6.4	**38373**	12.81	40541	11.76	53663	27.68
11	**141150**	**16.33**	**141150**	17.23	**141150**	36.65	144546	30.73	165152	73.4
12	**419979**	**35.42**	**419979**	37.73	**419979**	87.46	424828	65.53	454876	159.41
13	**942128**	**65.93**	**942128**	70.28	**942128**	174.42	948450	119.92	987858	295.08

is semantically equivalent to P. We try to solve $(X \cup Y, D, C_X \cup C_Y)$, where $D(x) = D_X(x)$ for $x \in X$ and $D(y) = D_Y(y)$ for $y \in Y$. This gluing operation doubles the number of variables and constraints, and introduces also a variable symmetry σ such that $\sigma(x_i) = y_i$ and $\sigma(y_i) = x_i$. This variable symmetry can be broken by the constraint $\boldsymbol{x} \leq_{lex} \boldsymbol{y}$ as ensured by Theorem 1, where $\boldsymbol{x}$ and $\boldsymbol{y}$ are sequences of X and Y respectively. We test this problem on five models. The experimental results of searching for all solutions are summarized in Table 1. The first and second models use value precedence constraints on adjacent pairs (adj-pair) and all pairs (all-pair) of the values respectively. The third model (if-then) uses the if-then constraints on adjacent pairs of values. The fourth (int-bool) and fifth (int-set) models use multiple viewpoints and channeling constraints [12].

Models using global constraints are substantially more efficient than the other approaches. The all-pair and adj-pair models achieve the same pruning, which is shared also by the if-then model. Therefore, the all-pair model is slightly slower since it has to process more value precedence constraints. Results of the channeling approach (int-bool and int-set) are provided for reference purposes only, since the approach relies on purely modeling techniques and no invention of new propagation algorithms. Its advantage is simplicity of and readiness for use in existing constraint programming systems. Although the channeling approach achieves less propagation, it is more efficient than the if-then model.

We have also experimented on Flener *et al.*'s version of Schur's lemma [4]. Our global constraints' results are more efficient than those of Gent reported by Flener *et al.* [4].

6.2 Social Golfer Problem

The social golfer problem, "prob010" in CSPLib, is to find a w-week schedule of g groups, each containing s golfers, such that no two golfers can play together more than once. The total number of golfers is $n = g \times s$. We denote an instance of the problem as (g, s, w). The problem is highly symmetric [1]:

1. Players can be permuted among the $n!$ combinations.
2. Weeks of schedule can be exchanged.
3. Groups can be exchanged inside weeks.

In the following, we describe an integer and a set model for the problem, so as to test both the integer and the set versions of the global constraints.

Table 2. Experimental Results for the Social Golfer Problem, using Integer Variables

	adj-pair		all-pair		if-then		int-bool		int-set	
g, s, w	fails	time	fails	time	fails	time	fails	time	fails	time
5,3,5	**26429**	**4.26**	**26429**	4.69	**26429**	10.92	26577	5.6	**26429**	8.66
5,3,7	**8235**	**1.94**	**8235**	2.14	**8235**	5.95	8435	2.68	**8235**	4.22
5,4,3	**51314**	**13.63**	**51314**	14.66	**51314**	44.32	51733	17.33	**51314**	28.28
5,4,4	**1127237**	**351.07**	**1127237**	377.52	**1127237**	1118.24	1132576	444.62	**1127237**	728.64
6,2,11	**54**	**0.02**	**54**	0.03	**54**	0.07	**54**	0.04	**54**	0.06
6,3,5	**1141372**	**321.97**	**1141372**	364.2	**1141372**	919.49	1145472	418.08	**1141372**	634.68
6,4,3	**2226446**	**651.88**	**2226446**	725.96	**2226446**	2592.27	2249286	812.3	**2226446**	1332.83
7,2,13	**1039**	**0.3**	**1039**	0.37	**1039**	1.08	1081	0.48	**1039**	0.61
7,3,4	**351**	**0.09**	**351**	0.11	**351**	0.36	358	0.13	**351**	0.19
7,4,3	**1093376**	**368.07**	**1093376**	423.14	**1093376**	1873.08	1116598	454.44	**1093376**	770.46
7,5,2	**48794**	**22.83**	**48794**	25.64	**48794**	152.57	50257	27.81	**48794**	49.56
8,3,5	**785865**	**249.1**	**785865**	302.19	**785865**	1073.7	791800	321.13	**785865**	510.85
8,4,9	**17**	**0.09**	**17**	0.09	**17**	0.73	18	0.11	**17**	0.17
8,5,2	**71463**	**38.04**	**71463**	43.46	**71463**	321.33	74679	45.22	**71463**	82.24
8,8,9	**19**	**0.3**	**19**	0.35	**19**	3.92	**19**	0.36	**19**	0.61
9,5,2	**9686**	**6.35**	**9686**	7.27	**9686**	71.65	10248	7.51	**9686**	13.74

Integer Model. One way to model the social golfer problem is to use variables $g_{i,k}$ for each golfer i in week k with $0 \leq i < n$ and $0 \leq k < w$. The domain of the variables $D(g_{i,k}) = \{0, \ldots, g-1\}$ contains the group numbers that golfer i can play in week k.

In this integer model, symmetries 1 and 2 are variable symmetries, and they can be broken by row ordering and column ordering constraints [4]. Note that these constraints do not completely break the compositions of the row and column symmetries. There are methods [7, 8] to introduce extra constraints to break more of them but they are out of the scope of this paper. Symmetry 3 is an example of symmetries of indistinguishable values. Therefore we can express value precedence constraints to break the symmetries. Theorem 1 ensures the safe posting of both types of symmetry breaking constraints.

Table 2 shows the experimental results of solving for the first solution of various instances using different models respectively. The results are similar to those for the Schur's lemma. Models using global constraints are the fastest among all, confirming the efficiency of our integer propagation algorithm. Again, the all-pair model shows no advantage in pruning over the adj-pair model, and is thus slightly less efficient due to the overhead in maintaining additional constraints. The if-then model, achieving the same amount of propagation as the global constraint approach, performs the worst in runtime among all models. Note that the performance of int-bool model approaches that of the global constraint models.

Set Model. Another way to model the social golfer problem is to use variables $p_{j,k}$ for each group j in week k with $0 \leq j < g$ and $0 \leq k < w$. Since a group in a week can contain multiple golfers, the variables $p_{j,k}$ are set variables and their domains are represented by the possible set $PS(p_{j,k}) = \{0, \ldots, n-1\}$, which is the set of golfers.

Table 3. Experimental Results for the Social Golfer Problem, using Set Variables

	no-break		adj-pair		all-pair		if-then		set-bool		set-int	
g, s, w	fails	time	fails	time	fails	time	fails	time	fails	time	fails	time
5,3,5	62	0.03	**38**	**0.02**	**38**	0.02	**38**	0.26	**38**	0.04	**38**	0.04
5,3,7	716851	**313.66**	**716827**	329.71	**716827**	446.65	-	-	**716827**	496.82	**716827**	480.04
5,4,3	2602	0.21	**107**	**0.02**	**107**	0.04	**107**	0.22	**107**	0.04	**107**	0.03
5,4,4	2886	0.37	**391**	**0.07**	**391**	0.13	**391**	0.93	**391**	0.13	**391**	0.13
6,2,11	**66**	**0.12**	**66**	0.13	**66**	0.15	**66**	2.39	**66**	0.17	**66**	0.16
6,3,5	**51**	0.04	**51**	**0.03**	**51**	0.05	**51**	0.44	**51**	0.06	**51**	0.05
6,4,3	20652	1.89	**1011**	**0.13**	**1011**	0.32	**1011**	2.1	**1011**	0.31	**1011**	0.28
7,2,13	**672**	0.66	**672**	**0.65**	**672**	0.72	**672**	34.07	**672**	0.79	**672**	0.74
7,3,4	30	**0.02**	**23**	0.03	**23**	0.03	**23**	0.36	**23**	0.04	**23**	0.03
7,4,3	35860	3.85	**2827**	**0.38**	**2827**	1.25	**2827**	6.58	**2827**	1	**2827**	0.91
7,5,2	-	-	**10503**	**0.93**	**10503**	5.11	**10503**	15.48	**10503**	2.91	**10503**	2.38
8,3,5	32216	**5.85**	**32192**	6.34	**32192**	13.58	**32192**	102.18	**32192**	12.03	**32192**	11.64
8,4,9	-	-	-	-	-	-	-	-	-	-	-	-
8,5,2	-	-	**20519**	**2.04**	**20519**	15.81	**20519**	47.09	**20519**	6.63	**20519**	5.67
8,8,9	64	**0.42**	**64**	0.44	**64**	1.94	**64**	35.95	**64**	0.76	**64**	0.68
9,5,2	-	-	**4021**	0.49	**4021**	4.25	**4021**	15.06	**4021**	1.64	**4021**	1.49

In this model, symmetries 2 and 3 are variable symmetries, and they can be broken by constraints $\min(p_{j,k}) < \min(p_{j+1,k})$ for $0 \leq j \leq g-2$ and $0 \leq k < w$ and $\min(p_{0,k} \setminus \{0\}) < \min(p_{0,k+1} \setminus \{0\})$ for $0 \leq k \leq w-2$ respectively [1]. These constraints are the result of simplifying the corresponding lexicographic ordering constraints for breaking the variable symmetries. Symmetry 1 becomes symmetries of indistinguishable values $\{0, \ldots, n-1\}$, which can be tackled by value precedence constraints. Again, Theorem 1 ensures the consistency of the two sets of symmetry breaking constraints.

Table 3 summarizes the experimental results of solving for the first solution of various problem instances using various models. A cell labeled with "-" means that the search does not terminate in one hour of CPU time. In this experiment, we report also the result of a model (no-break) with no indistinguishable value symmetry breaking constraints, since there are instances with few symmetries to break (as indicated by the number of fails) during the search for the first solution. In those cases, the no-break model edges the performance of the adj-pair and all-pair models, but the good news is that the margin is small. This shows that our global constraint implementations incur low overhead. In the cases with substantial pruning of search space by symmetry breaking, the adj-pair and all-pair models perform substantially better in terms of percentage speedup than the other models although the timings are small in general. In this experiment, all models with symmetry breaking achieve the same propagation.

7 Conclusion

The contributions of our work are three-fold. First, the notion of value precedence is introduced. We show how the notion can be used to design constraints for breaking symmetries of indistinguishable values. Second, we present linear time propagation algorithms for implementing global constraints on value precedence. Experiments are conducted to verify the efficiency of our proposal. Results confirm that our implementations incur little overhead and are robust. Third, we give theoretical results to characterize the exact behavior of our proposed algorithms in different usage scenarios.

An interesting line of future research is to generalize the value precedence constraints. First, the antecedent and subsequent can be also constrained integer variables instead of just integer constants. Second, Theorem 5 ensures that more propagation can be achieved if we can maintain value precedence on an arbitrary non-singleton set of values simultaneously.

Acknowledgments

We thank the anonymous referees for their constructive comments which help improve the quality of the paper. We also acknowledge The University of York for providing the source of the lexicographic ordering global constraint for our reference. The work described in this paper was substantially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region (Project no. CUHK4219/04E).

References

1. N. Barnier and P. Brisset. Solving the Kirkman's schoolgirl problem in a few seconds. In *Proceedings of CP-02*, pages 477–491, 2002.
2. B. Benhamou. Study of symmetry in constraint satisfaction problems. In *Proceedings of PPCP-94*, 1994.
3. J. Crawford, M. Ginsberg, E. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In *Proceedings of KR-96*, pages 148–159, 1996.
4. P. Flener, A. M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. In *Proceedings of CP-02*, pages 462–476, 2002.
5. E. C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Proceedings of AAAI-91*, pages 227–233, 1991.
6. A. M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, and T. Walsh. Global constraints for lexicographical orderings. In *Proceedings of CP-02*, pages 93–108, 2002.
7. A. M. Frisch, C. Jefferson, and I. Miguel. Constraints for breaking more row and column symmetries. In *Proceedings of CP-03*, pages 318–332, 2003.
8. A. M. Frisch, I. Miguel, Z. Kiziltan, B. Hnich, and T. Walsh. Multiset ordering constraints. In *Proceedings of IJCAI-03*, pages 221–226, 2003.
9. I.P. Gent. A symmetry breaking constraint for indistinguishable values. In *Proceedings of SymCon-01*, 2001.
10. C. Gervet. Interval propagation to reason about sets: Definition and implementation of a practical language. *Constraints*, 1(3):191–244, 1997.
11. ILOG. *ILOG Solver 4.4 Reference Manual*, 1999.
12. Y. C. Law and J. H. M. Lee. Breaking value symmetries in matrix models using channeling constraints. Technical report, The Chinese Univ. of Hong Kong, 2004.
13. R. Mohr and G. Masini. Good old discrete relaxation. In *Proceedings of ECAI-88*, pages 651–656, 1988.