Symmetry Breaking Constraints for Value Symmetries in Constraint Satisfaction

Y.C. Law (yclaw@cse.cuhk.edu.hk) and J.H.M. Lee
(jlee@cse.cuhk.edu.hk)
Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, New Territories, Hong Kong

Abstract.
Constraint satisfaction problems (CSPs) sometimes contain both *variable symmetries* and *value symmetries*, causing adverse effects on CSP solvers based on tree search. As a remedy, symmetry breaking constraints are commonly used. While variable symmetry breaking constraints can be expressed easily and propagated efficiently using lexicographic ordering, value symmetry breaking constraints are often difficult to formulate. In this paper, we propose two methods of using symmetry breaking constraints to tackle value symmetries. First, we show theoretically *when* value symmetries in one CSP correspond to variable symmetries in another CSP of the same problem. We also show *when* variable symmetry breaking constraints in the two CSPs, combined using channeling constraints, are consistent. Such results allow us to tackle value symmetries efficiently using additional CSP variables and channeling constraints. Second, we introduce *value precedence*, a notion which can be used to break a common class of value symmetries, namely symmetries of *indistinguishable values*. While value precedence can be expressed using inefficient if-then constraints in existing CSP solvers, we propose efficient propagation algorithms for implementing global value precedence constraints. We also characterize several theoretical properties of the value precedence constraints. Extensive experiments are conducted to verify the feasibility and efficiency of the two proposals.

1. Introduction

Many real life problems can be modeled as *constraint satisfaction problems* (CSPs), which is defined by Mackworth [38] as follows:

> *We are given a set of variables, a domain of possible values for each variable, and a conjunction of constraints. Each constraint is a relation defined over a subset of the variables, limiting the combination of values that the variables in this subset can take. The goal is to find a consistent assignment of values from the domains to the variables so that all the constraints are satisfied simultaneously.*

A CSP often exhibit some symmetries, which are mappings that preserve satisfiability of the CSP. They are a curse of CSP solving algorithms based on tree search, since symmetrically equivalent states in the search tree can be explored more than once. One main approach of symmetry breaking is to add *symmetry breaking constraints* [43] to a

CSP before search, so that some symmetrical equivalent solutions are removed in the reformulated CSP.

There are two common types of CSP symmetries, namely *variable symmetries* and *value symmetries*. Crawford et al. [18] suggested that we can always break variable symmetries using lexicographic ordering constraints. In addition, efficient propagation algorithms [21, 14, 13] exist for maintaining lexicographic ordering. As a result, variable symmetry breaking constraints can be expressed relatively easily and executed efficiently in existing constraint programming systems. However, there are no general methods to date to formulate symmetry breaking constraints for value symmetries in CSPs. In this paper, we propose two methods to remedy this difficulty.

The first method makes uses of multiple viewpoints and channeling constraints [15] to break value symmetries in matrix models [21], which are CSPs with variables indexed and organized into matrices. Flener et al. [21] suggested that it is possible to transform an $(n-1)$-dimensional matrix with variable and value symmetries into an n-dimensional matrix of 0/1 variables that contains only variable symmetries. Symmetry breaking constraints are then expressed in the n-dimensional matrix to break the symmetries of the problem. We formalize this idea by theoretically showing that value symmetries in a matrix model always correspond to variable symmetries in the 0/1 viewpoint. We also *generalize* the idea to characterize the conditions *when* value symmetries in one matrix model correspond to variable symmetries in another non-0/1 matrix model of the same problem. We then give the conditions *when* variable symmetry breaking constraints in two matrix models of the same problem, when combined using channeling constraints, are consistent. Such results enable us to break value symmetries in one viewpoint using variable symmetry breaking constraints in another.

In the second method, we identify an important class of value symmetries, namely symmetries of *indistinguishable values* [10, 28], an example of which is the colors in graph coloring problems. We introduce a notion called *value precedence* and explain how imposing value precedence on a sequence of CSP variables can break symmetries of indistinguishable values in both integer and set domains. Although the value precedence condition on a sequence of variables is easy to express using if-then constraints in many existing constraint programming systems, such a formulation is inefficient both in terms of number of constraints and propagation efficiency. We propose two efficient propagation algorithms for implementing value precedence global constraints on integer and set variable sequences respectively. We also study several theoretical properties of the proposed value precedence constraints.

This paper, a revised and extended version of the work by Law and Lee [37, 36] and Law [35], is organized as follows. Section 2 provides background to the paper. We formally define the concept of CSPs and two common types of CSP symmetries, namely variable and value symmetries. Section 3 presents how to break value symmetries in matrix models using multiple viewpoints and channeling constraints. Section 4 introduces our other method that breaks symmetries of indistinguishable values using value precedence. Section 5 presents experimental results using the two proposals and Section 6 presents a brief review of the related work in symmetry breaking. Section 7 summarizes our contributions, and gives discussions and possible directions for future research.

2. Background

This section provides background to the paper. We provide the definitions of various CSP related concepts and defines two common types of CSP symmetries, namely variable and value symmetries. We give also common existing methods of breaking such types of symmetries.

2.1. Constraint Satisfaction Problems

A *viewpoint* is a pair (X, D), where X is a set of *variables*, and D is a function that maps each $x \in X$ to its associated *domain*, giving the set of possible values for x. There are two common classes of variables in CSPs. An *integer variable* [1] x has an integer domain, i.e., $D(x)$ is a finite integer set. A *set variable* [32, 1] x has a set domain; each element in the domain is a finite integer 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 the variable. The *required set* $RS(x)$ contains elements that belong to all the possible values of the variable. By definition, $RS(x) \subseteq PS(x)$. The domain of x is then represented as $D(x) = \{a \mid RS(x) \subseteq a \subseteq PS(x)\}$. 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 $RS(x)$, a fail is triggered. 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. For ease of description, we abuse terminology by defining the possible set $PS(x)$ of an integer variable x to be $D(x)$.

An *assignment* $x \mapsto a$ in (X, D) means variable $x \in X$ is mapped to the value $a \in D(x)$. A *compound assignment* is a set of assignments in which no variables can be assigned more than once. We overload the $\mapsto$ operator such that $\langle x_{i_1}, \ldots, x_{i_k} \rangle \mapsto \langle a_1, \ldots, a_k \rangle$ means the compound assignment $\{x_{i_j} \mapsto a_j \mid 1 \leq j \leq k\}$. A *complete assignment* is a compound assignment for all variables in a CSP.

A *constraint* in a viewpoint V places restrictions on a subset of variables in V, limiting the combination of values that these variables can take. A *CSP model* M (or simply *model* or *CSP*) of a problem P is a pair (V, C), where $V = (X, D)$ is a viewpoint of P and C is a set of constraints in V for P. Besides using (V, C), we also use the triple (X, D, C) to denote M, i.e., $M = (V, C) = (X, D, C)$. A *solution* of a CSP M is a complete assignment that satisfies all the constraints in C. The set of all solutions of M is denoted by $sol(M)$.

In order to reason with integer and set variables uniformly, we introduce the notion of *decisions* which are analogous to assignments. A *decision* $x \rhd b$ in (X, D) means variable $x \in X$ is mapped to the value $b \in PS(x)$. It has different meanings depending on the class of variable x. If x is an integer variable, $x \rhd b$ simply means $x \mapsto b$. If x is a set variable, $x \rhd b$ means b is added to the required set $RS(x)$ of x, i.e., $b \in x$. A *compound decision* is a set of decisions. Decisions are different from assignments in that multiple decisions are allowed for a set variable, while multiple assignments are not allowed for any variable. For example, if x is an integer variable, then $\theta = \{x \rhd 1, x \rhd 2\}$ is not a valid compound decision. However, if x is a set variable, then θ is a valid compound decision, meaning $\{1, 2\} \subseteq x$. A compound decision has a *scope* indicating the set of assigned variables. For example, for integer variable x and set variables y and z, the compound decision $\{x \rhd 1, y \rhd 1, y \rhd 2\}$ with scope $\{x, y, z\}$ means the compound assignment $\{x \mapsto 1, y \mapsto \{1, 2\}, z \mapsto \emptyset\}$. With the same scope U, compound assignments and compound decisions have a 1-1 correspondence. Therefore, we use compound decisions as well as complete assignments to represent solutions of a CSP interchangeably whenever necessary. We overload the $\rhd$ operator such that $\langle x_{i_1}, \ldots, x_{i_k} \rangle \rhd \langle a_1, \ldots, a_k \rangle$ means the compound decision $\{x_{i_j} \rhd a_j \mid 1 \leq j \leq k\}$.

An *extension* of an assignment $x \mapsto a$ is a compound assignment that includes $x \mapsto a$. A constraint c is *generalized arc consistent* (GAC) [41] if and only if for each variable x in c and $a \in D(x)$, $x \mapsto a$ can be extended to a solution of c. *Generalized arc consistency* (GAC) is prohibitive to enforce on constraints involving set variables. Instead, *set bounds consistency* (SBC) is typically enforced. A constraint c on set variables is *set bounds consistent* (SBC) [32] if and only if for each set variable x in c, both $x \mapsto PS(x)$ and $x \mapsto RS(x)$ can be extended to solutions of c. That is, for each set variable x in c and $a \in PS(x) \setminus RS(x)$, both $c \wedge a \in x$ and $c \wedge a \notin x$ are satisfiable.

week \ golfer	1	2	3	4	5	6
1	1	1	2	2	3	3
2	1	2	1	3	2	3
3	1	2	2	3	3	1

(a) V_G

group \ week	1	2	3
1	$\{1,2\}$	$\{1,3\}$	$\{1,6\}$
2	$\{3,4\}$	$\{2,5\}$	$\{2,3\}$
3	$\{5,6\}$	$\{4,6\}$	$\{4,5\}$

(b) V_P

golfer \ group	1	2	3
1	$\{1,2,3\}$	$\emptyset$	$\emptyset$
2	$\{1\}$	$\{2,3\}$	$\emptyset$
3	$\{2\}$	$\{1,3\}$	$\emptyset$
4	$\emptyset$	$\{1\}$	$\{2,3\}$
5	$\emptyset$	$\{2\}$	$\{1,3\}$
6	$\{3\}$	$\emptyset$	$\{1,2\}$

(c) V_W

week	1			2			3		
golfer \ group	1	2	3	1	2	3	1	2	3
1	1	0	0	1	0	0	1	0	0
2	1	0	0	0	1	0	0	1	0
3	0	1	0	1	0	0	0	1	0
4	0	1	0	0	0	1	0	0	1
5	0	0	1	0	1	0	0	0	1
6	0	0	1	0	0	1	1	0	0

(d) V_Z

Figure 1. Four Equivalent Solutions of $(3,2,3)$ in V_G, V_P, V_W, and V_Z Respectively

2.2. Symmetries

In this subsection, we define two types of symmetries, namely variable symmetries and value symmetries. We illustrate existing methods for breaking such symmetries using the social golfer problem as a running example.

The social golfer problem (SGP), "prob010" in CSPLib,[1] is to find a $\mathcal{W}$-week schedule of $\mathcal{G}$ groups, each containing $\mathcal{S}$ golfers, such that no two golfers can play together more than once. There are totally $\mathcal{N} = \mathcal{G} \times \mathcal{S}$ golfers. We denote each instance of the problem by $(\mathcal{G}, \mathcal{S}, \mathcal{W})$. The SGP is highly symmetric [20]:

1. players can be permuted among the $\mathcal{N}!$ combinations,
2. weeks of schedule can be exchanged, and
3. groups can be exchanged inside weeks.

One way to model the problem into a CSP uses the viewpoint $V_G = (G, D_G)$ which contains an integer variable $g_{i,k}$ for each golfer i in week k with $1 \leq i \leq \mathcal{N}$ and $1 \leq k \leq \mathcal{W}$. The variable domain $D_G(g_{i,k}) = \{1, \ldots, \mathcal{G}\}$ contain the group numbers that golfer i can play in week k. This model M_G is a matrix model [21], since G forms a 2-dimensional matrix of variables. Figure 1(a) gives a solution of the $(3,2,3)$ instance.

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

2.2.1. *Variable Symmetries*

A *variable symmetry* of a CSP $M = (X, D, C)$ is a solution-preserving bijective mapping from the set of variables X to itself, $\sigma : X \to X$. Given a bijective mapping $\sigma : X \to X$, we overload σ to act on a sequence of variables $\vec{x} = \langle x_1, \ldots, x_n \rangle$ such that $\sigma(\vec{x}) = \langle \sigma(x_1), \ldots, \sigma(x_n) \rangle$, and also on a compound decision θ such that $\sigma(\theta) = \{\sigma(x) \rhd a \,|\, (x \rhd a) \in \theta\}$. A variable symmetry σ requires that

$$\theta \in sol(M) \iff \sigma(\theta) \in sol(M).$$

Symmetry (1) of the SGP is an example of variable symmetries in V_G. Consider the solution in Figure 1(a), we can exchange the variables of golfers 1 and 2 to obtain another solution with $\langle g_{1,1}, g_{1,2}, g_{1,3} \rangle \rhd \langle 1, 2, 2 \rangle$ and $\langle g_{2,1}, g_{2,2}, g_{2,3} \rangle \rhd \langle 1, 1, 1 \rangle$. Hence, we have the bijective mapping σ as the identity mapping except $\sigma(\langle g_{1,k}, g_{2,k} \rangle) = \langle g_{2,k}, g_{1,k} \rangle$ for $1 \leq k \leq 3$.

Symmetry (2) is another example of variable symmetries in V_G. In Figure 1(a), we can exchange the variables of weeks 1 and 2 to obtain another solution with $\langle g_{1,1}, \ldots, g_{6,1} \rangle \rhd \langle 1, 2, 1, 3, 2, 3 \rangle$ and $\langle g_{1,2}, \ldots, g_{6,2} \rangle \rhd \langle 1, 1, 2, 2, 3, 3 \rangle$. Hence, we have another bijective mapping σ' which is the identity mapping except $\sigma'(\langle g_{i,1}, g_{i,2} \rangle) = \langle g_{i,2}, g_{i,1} \rangle$ for $1 \leq i \leq 6$.

Variable symmetries can be broken using *lexicographic ordering* [26]. A sequence $\vec{x} = \langle x_1, \ldots, x_n \rangle$ is *lexicographically smaller than or equal to* another sequence $\vec{y} = \langle y_1, \ldots, y_n \rangle$, written as $\vec{x} \leq_{lex} \vec{y}$ or $\vec{y} \geq_{lex} \vec{x}$, if and only if

$$x_1 \leq y_1 \text{ and } \Big(\bigwedge_{1 \leq i' < i} x_{i'} = y_{i'}\Big) \to x_i \leq y_i \text{ for } 1 < i \leq n.$$

The sequence $\vec{x}$ is *lexicographically smaller than* $\vec{y}$, written as $\vec{x} <_{lex} \vec{y}$ or $\vec{y} >_{lex} \vec{x}$, if and only if $\vec{x} \leq_{lex} \vec{y}$ and $\vec{x} \neq \vec{y}$.

In general, a variable symmetry σ can be broken by the lexicographic ordering constraint [18]

$$\vec{x} \leq_{lex} \sigma(\vec{x}),$$

where $\vec{x}$ is a sequence of the variables in the CSP. Using a constraint for each variable symmetry σ, all symmetrical solutions in each equivalence class except the lexicographically smallest one, with respect to the sequence $\vec{x}$, would be removed. Sometimes, these constraints can be simplified [34] to contain fewer variables. An example is the row ordering and column ordering constraints for row and column symmetries [21]. For example, symmetry (1) of the SGP can be broken by the row ordering constraints $\langle g_{i,1}, \ldots, g_{i,\mathcal{W}} \rangle \leq_{lex} \langle g_{i+1,1}, \ldots, g_{i+1,\mathcal{W}} \rangle$ for $1 \leq i < \mathcal{N}$. Similarly, we can break symmetry (2) in V_G by the column ordering constraints $\langle g_{1,k}, \ldots, g_{\mathcal{N},k} \rangle \leq_{lex} \langle g_{1,k+1}, \ldots, g_{\mathcal{N},k+1} \rangle$ for

$1 \leq k < \mathcal{W}$. Bessiere et al. [11] showed the intractability of breaking row and column symmetries completely. These row ordering and column ordering constraints are only a subset of all the variable symmetry breaking constraints. They do not necessarily break all the compositions of the row and column symmetries [21]. There are methods to introduce extra constraints to break more [27] but they are out of the scope of this paper, which focuses on value symmetries.

2.2.2. *Value Symmetries*

A *value symmetry* under a subset $U \subseteq X$ of the variables of a CSP $M = (X, D, C)$, where $PS(x) = PS(x')$ for all $x, x' \in U$, is a solution-preserving bijective mapping on the possible set of the variables in U, $\tau : PS(x) \to PS(x)$ where $x \in U$. Given a bijective mapping $\tau : \mathbb{Z} \to \mathbb{Z}$, we overload τ to act also on a variable subset and a compound decision θ such that $\tau(U, \theta) = \{x \rhd \tau(a) \,|\, (x \rhd a) \in \theta \wedge x \in U\} \cup \{x \rhd a \,|\, (x \rhd a) \in \theta \wedge x \notin U\}$, which is a compound decision such that for each decision $(x \rhd a) \in \theta$, $x \rhd \tau(a)$ is in $\tau(U, \theta)$ if U contains the variable x, and $x \rhd a$ is in $\tau(U, \theta)$ if U does not contain x. A value symmetry τ under U requires that

$$\theta \in sol(M) \iff \tau(U, \theta) \in sol(M).$$

If U is a set of integer variables, τ is called an *integer value symmetry*. If U is a set of set variables, τ is called a *set value symmetry*.

Value symmetry is similar to but more general than value interchangeability [24]. Interchangeable values can be exchanged for a *single* variable without affecting the satisfiability of constraints, while a value symmetry is under a set of variables and can be applied to a solution to form another solution of the same CSP. For example, a value a for a variable x is *fully interchangeable* [24] with a value b for x if and only if there is an integer value symmetry τ under $\{x\}$ such that τ is the identity mapping except $\tau(a) = b$ and $\tau(b) = a$.

Symmetry (3) in the SGP is an example of integer value symmetries in V_G. Consider the solution in Figure 1(a). We can permute the values assigned to the set of variables $U = \{g_{1,1}, \ldots, g_{n,1}\} \subseteq G$ from 1 to 2, from 2 to 3, and from 3 to 1 to obtain another solution with $\langle g_{1,1}, \ldots, g_{6,1}\rangle \rhd \langle 2, 2, 3, 3, 1, 1\rangle$. Thus, we have a value symmetry τ under U with $\tau(1) = 2$, $\tau(2) = 3$, and $\tau(3) = 1$.

Value symmetry breaking constraints are difficult to express in general, since we do not know beforehand which variable will be assigned which value. Value symmetries are usually handled by pre-assigning the affected variables as far as possible with some values without loss of generality. However, these pre-assignments, which must be extensible to solutions, cannot break all value symmetries in general. For example, in the SGP, without loss of generality, we can always have the pre-assignments

$$\langle g_{1,1}, \ldots, g_{\mathcal{N},1} \rangle \rhd \langle \underbrace{1, \ldots, 1}_{\mathcal{S}}, \ldots, \underbrace{\mathcal{G}, \ldots, \mathcal{G}}_{\mathcal{S}} \rangle \quad \text{and}$$
$$\langle g_{1,k}, \ldots, g_{\mathcal{S},k} \rangle \rhd \langle 1, \ldots, \mathcal{S} \rangle \qquad \text{for } k > 1.$$

The former breaks the value symmetries for week 1. The latter breaks the value symmetries of values 1 to $\mathcal{S}$ from week 2 and so on, but those of values $\mathcal{S}+1$ to $\mathcal{G}$ remains intact. Therefore, the larger $\mathcal{G} - \mathcal{S}$, the fewer value symmetries can be broken by the pre-assignments.

Symmetries of indistinguishable values [10, 28] is a special class of value symmetries. A set of values $\{v_1, \ldots, v_k\}$ is *indistinguishable* under $U = \{x_1, \ldots, x_n\}$ if the values imply $k!$ value symmetries τ under U, where $\langle \tau(v_1), \ldots, \tau(v_k) \rangle$ is a permutation of $\langle v_1, \ldots, v_k \rangle$. In the SGP, our previous example of value symmetry τ under $U = \{g_{1,1}, \ldots, g_{6,1}\}$ has the mapping $\tau(1) = 2$, $\tau(2) = 3$, and $\tau(3) = 1$. Actually, the groups $\{1, 2, 3\}$ are indistinguishable values under U, implying $3! = 6$ value symmetries under U, which are the six permutations of $\langle 1, 2, 3 \rangle$. The value symmetry τ is one of the six permutations.

3. Breaking Value Symmetries with Channeling

In this section, we introduce the method of breaking value symmetries using multiple viewpoints and channeling constraints. Our method is applicable to *multi-aspect assignment problems* (MAPs), which can be naturally formulated into various matrix models [21]. In the following, we first describe MAPs, and a general method to derive $n+1$ viewpoints for modeling a MAP with n aspects as matrix models. We then show theoretically *when* a value symmetry in a CSP (V, C) corresponds to a variable symmetry in another CSP (V', C') modeling the same problem. We also show *when* variable symmetry breaking constraints in two viewpoints V and V', connected with channeling constraints [15], are consistent. Using these results, we can tackle value symmetries in (V, C) by expressing variable symmetry breaking constraints using another viewpoint V' and connecting V and V' using channeling constraints.

3.1. Multi-aspect Assignment Problems

In the SGP, there are three *aspects*, corresponding to the sets of golfers, weeks, and groups respectively. Solving the problem is to find a set of tuples of the form $(aGolfer, aWeek, aGroup)$ that satisfies the problem requirements. The SGP is an instance of *multi-aspect assignment problems* (MAPs). A MAP consists of n aspects, each of which corresponds to a set of objects of the problem. Without loss of generality, we define the set of objects of the i-th aspect as $Obj(i) = \{1, \ldots, k_i\}$, where k_i is the number of objects in aspect i. For example, we can use $Obj(1) = \{1, \ldots, \mathcal{N}\}$, $Obj(2) = \{1, \ldots, \mathcal{W}\}$, and $Obj(3) = \{1, \ldots, \mathcal{G}\}$ to denote the set of all golfers, weeks, and groups respectively in the SGP. Solving a MAP is to find a *solution set* of tuples $S \subseteq Obj(1) \times \ldots \times Obj(n)$, i.e., a relation among the n aspects, that satisfies the problem constraints. For example, the tuple $(1, 2, 3)$ in a solution set of the SGP means that golfer 1 plays in group 3 in week 2. Note that a multi-aspect assignment problem is different from a *multidimensional assignment problem* [42], which is an optimization problem subject to some constraints in *particular forms*. Many real life problems, such as combinatorial, configuration, scheduling, design, and assignment problems, are MAPs. As we shall see, MAPs can readily be formulated into *matrix models* [22, 21], which are CSPs in which the variables can be indexed and organized into one or more matrices.

3.2. Viewpoints for Modeling MAPs

A matrix can be multi-dimensional. We also use the array notation in addition to the subscript notation to denote the matrix variables in the following discussions for easier reading. In the following, we describe two types of viewpoints for modeling MAPs, namely the aspect viewpoints and 0/1 viewpoint.

3.2.1. *Aspect Viewpoints*

Given a MAP with n aspects, we can always choose any $n-1$ aspects as matrix indices to form a matrix of variables and the remaining aspect to form the variable domains. For $1 \leq s \leq n$, let

$$X_s = \{x_s[i_1] \cdots [i_{s-1}][i_{s+1}] \cdots [i_n] \mid \bigwedge_{1 \leq k \leq n, k \neq s} i_k \in Obj(k)\}$$

be the matrix of variables using all but the s-th aspect as indices. The variable domains correspond to the objects in the s-th aspect, i.e.,

$$PS(x_s[i_1] \cdots [i_{s-1}][i_{s+1}] \cdots [i_n]) = Obj(s).$$

For easier reading, we use the notation $x[i_1 \cdots i_n \setminus i_s]$ in subsequent discussions to denote the variable $x_s[i_1] \cdots [i_{s-1}][i_{s+1}] \cdots [i_n]$. If the MAP allows only exactly one decision for each variable in X_s, then X_s is a set of integer variables. Otherwise, X_s is a set of set variables. Hence, we can derive n different *aspect viewpoints* $V_1 = (X_1, D_1), \ldots, V_n = (X_n, D_n)$ for a MAP. The subscript k in $V_k = (X_k, D_k)$ denotes the aspect corresponding to the domains in V_k. The variables between any two aspect viewpoints V_s and V_t ($s \neq t$) can be related by the channeling constraints [15]

$$x[i_1 \cdots i_n \setminus i_s] \rhd i_s \iff x[i_1 \cdots i_n \setminus i_t] \rhd i_t \text{ for } \bigwedge_{1 \le k \le n} i_k \in Obj(k).$$

They collectively induce a *channeling function*

$$f_{s,t}(x[i_1 \cdots i_n \setminus i_s] \rhd i_s) = x[i_1 \cdots i_n \setminus i_t] \rhd i_t$$

from decisions in V_s to those in V_t, for $\bigwedge_{1 \le k \le n} i_k \in Obj(k)$. The reverse channeling function $f_{t,s}$ is simply $f_{s,t}^{-1}$. Note that $f_{s,t}^{-1}$ always exists. This can be seen from the channeling constraints that the sets of all possible decisions in V_s and V_t have a one-one mapping.

In the SGP, $V_G = (G, D_G)$ is an aspect viewpoint using the golfers and weeks to form the variables, and sets of groups to form the domain. The other two aspect viewpoints are $V_P = (P, D_P)$ and $V_W = (W, D_W)$. Viewpoint V_P uses the groups and weeks to form the variables, and sets of golfers to form the domain. Viewpoint V_W uses the golfers and groups to form the variables, and sets of weeks to form the domain. Since a group in a particular week can contain multiple golfers, the variables $p_{j,k} \in P$ are set variables with $PS(p_{j,k}) = \{1, \ldots, \mathcal{N}\}$. Similarly, a golfer can have the same group number for multiple weeks, the variables $w_{i,j} \in W$ are also set variables with $PS(w_{i,j}) = \{1, \ldots, \mathcal{W}\}$. Figures 1(a), 1(b), and 1(c) show the same solution of $(3, 2, 3)$ expressed in V_G, V_P, and V_W respectively. The channeling constraints between V_G and V_P are $g_{i,k} \rhd j \iff p_{j,k} \rhd i$, the ones between V_G and V_W are $g_{i,k} \rhd j \iff w_{i,j} \rhd k$, and the ones between V_P and V_W are $p_{j,k} \rhd i \iff w_{i,j} \rhd k$, for $1 \le i \le \mathcal{N}$, $1 \le j \le \mathcal{G}$, and $1 \le k \le \mathcal{W}$.

3.2.2. *0/1 Viewpoint*

Besides the aspect viewpoints, we can use all n aspects of a MAP to form an n-dimensional matrix of 0/1 variables

$$Z = \{z[i_1] \cdots [i_n] \mid \bigwedge_{1 \le k \le n} i_k \in Obj(i_k)\}.$$

Each variable $z[i_1] \cdots [i_n] \in Z$ denotes whether the tuple $(i_1, \ldots, i_n)$ is in a solution of the MAP. Hence, $D_Z(z[i_1] \cdots [i_n]) = \{0, 1\}$, giving us the *0/1 viewpoint* $V_Z = (Z, D_Z)$. For $1 \le s \le n$, the channeling constraints [15] between aspect viewpoint V_s and 0/1 viewpoint V_Z are

$$x[i_1 \cdots i_n \setminus i_s] \rhd i_s \iff z[i_1] \cdots [i_n] \rhd 1 \text{ for } \bigwedge_{1 \le k \le n} i_k \in Obj(k).$$

They collectively induce a channeling function

$$f_{s,Z}(x[i_1 \cdots i_n \setminus i_s] \rhd i_s) = z[i_1] \cdots [i_n] \rhd 1$$

from decisions in V_s to *only* those of the form "$z[i_1]\cdots[i_n] \rhd 1$" in V_Z, for $\bigwedge_{1\leq k\leq n} i_k \in Obj(k)$ (since the channeling constraints never generate decisions of the form "$z[i_1]\cdots[i_n] \rhd 0$"). Again, $f_{Z,s}$ is $f^{-1}_{s,Z}$, which always exists because $f_{s,Z}$ is a one-one mapping. In the SGP, V_Z contains variables $z_{i,k,j}$ for each golfer i, week k, and group j with $D_Z(z_{i,k,j}) = \{0,1\}$. Figure 1(d) shows the same solution as those in Figures 1(a), 1(b), and 1(c), but expressed in V_Z. The channeling constraints between V_Z and, say, V_G, are $g_{i,k} \rhd j \iff z_{i,k,j} \rhd 1$ for $1 \leq i \leq \mathcal{N}$, $1 \leq k \leq \mathcal{W}$, and $1 \leq j \leq \mathcal{G}$.

3.3. From Value Symmetries to Variable Symmetries

In the rest of the section, we suppose $M_s = (V_s, C_s)$, $M_t = (V_t, C_t)$, and $M_Z = (V_Z, C_Z)$ are CSP models for the same MAP with n aspects, where $V_s = (X_s, D_s)$ and $V_t = (X_t, D_t)$ are aspect viewpoints, and $V_Z = (Z, D_Z)$ is the 0/1 viewpoint.

3.3.1. *From Aspect Viewpoint to 0/1 Viewpoint*

Flener et al. [21] suggested that value symmetries in a matrix model can be broken as variable symmetries in the 0/1 viewpoint. The following theorem formally describes this idea and shows that a value symmetry τ in M_s *always* corresponds to a variable symmetry σ in M_Z.

THEOREM 1. *Given a value symmetry τ under $U_s \subseteq X_s$, we have*

$$\sigma(f_{s,Z}(\theta)) = f_{s,Z}(\tau(U_s, \theta))$$

for all $\theta \in sol(M_s)$, where

$$\sigma(z[i_1]\cdots[i_n]) = \begin{cases} z[i_1]\cdots[i_{s-1}][\tau(i_s)][i_{s+1}]\cdots[i_n] \\ \qquad \textit{if } x_s[i_1]\cdots[i_{s-1}][i_{s+1}]\cdots[i_n] \in U_s \\ z[i_1]\cdots[i_n] \\ \qquad \textit{otherwise.} \end{cases}$$

In addition, σ is a variable symmetry in M_Z corresponding to τ in M_s.

Proof. Let $\theta \in sol(M_s)$.

$$\begin{aligned} \tau(U_s,\theta) &= \{x[i_1\cdots i_n \setminus i_s] \rhd i_s \mid \\ &\qquad (x[i_1\cdots i_n \setminus i_s] \rhd i_s) \in \theta \wedge x[i_1\cdots i_n \setminus i_s] \notin U_s\} \\ &\quad \cup \{x[i_1\cdots i_n \setminus i_s] \rhd \tau(i_s) \mid \\ &\qquad (x[i_1\cdots i_n \setminus i_s] \rhd i_s) \in \theta \wedge x[i_1\cdots i_n \setminus i_s] \in U_s\} \\ &\in sol(M_s) \\ f_{s,Z}(\tau(U_s,\theta)) &= \{z[i_1]\cdots[i_n] \rhd 1 \mid \end{aligned}$$

$$
\begin{aligned}
&\qquad (x[i_1 \cdots i_n \setminus i_s] \rhd i_s) \in \theta \wedge x[i_1 \cdots i_n \setminus i_s] \notin U_s\} \\
&\cup \{z[i_1] \cdots [i_{s-1}][\tau(i_s)][i_{s+1}] \cdots [i_n] \rhd 1 \,| \\
&\qquad (x[i_1 \cdots i_n \setminus i_s] \rhd i_s) \in \theta \wedge x[i_1 \cdots i_n \setminus i_s] \in U_s\}
\end{aligned}
$$

$$
\begin{aligned}
f_{s,Z}(\theta) &= \{z[i_1] \cdots [i_n] \rhd 1 \,|\, (x[i_1 \cdots i_n \setminus i_s] \rhd i_s) \in \theta\} \\
&= \{z[i_1] \cdots [i_n] \rhd 1 \,| \\
&\qquad (x[i_1 \cdots i_n \setminus i_s] \rhd i_s) \in \theta \wedge x[i_1 \cdots i_n \setminus i_s] \notin U_s\} \\
&\quad \cup \{z[i_1] \cdots [i_n] \rhd 1 \,| \\
&\qquad (x[i_1 \cdots i_n \setminus i_s] \rhd i_s) \in \theta \wedge x[i_1 \cdots i_n \setminus i_s] \in U_s\} \\
\sigma(f_{s,Z}(\theta)) &= \{z[i_1] \cdots [i_n] \rhd 1 \,| \\
&\qquad (x[i_1 \cdots i_n \setminus i_s] \rhd i_s) \in \theta \wedge x[i_1 \cdots i_n \setminus i_s] \notin U_s\} \\
&\quad \cup \{z[i_1] \cdots [i_{s-1}][\tau(i_s)][i_{s+1}] \cdots [i_n] \rhd 1 \,| \\
&\qquad (x[i_1 \cdots i_n \setminus i_s] \rhd i_s) \in \theta \wedge x[i_1 \cdots i_n \setminus i_s] \in U_s\} \\
&= f_{s,Z}(\tau(U_s, \theta))
\end{aligned}
$$

Note that $f_{s,Z}(\theta)$ and $f_{s,Z}(\tau(U_s, \theta))$ do not consist of all decisions for solutions of M_Z, since $f_{s,Z}$ only generates decisions of the form "$z[i_1] \cdots [i_n] \rhd 1$." But they can be complemented to include decisions with value 0 for all other variables in Z to make them solutions of M_Z. So, σ is a solution preserving bijective mapping. Hence, a variable symmetry in M_Z corresponds to the value symmetry τ in M_s. □

For each solution θ of M_s, since τ is a value symmetry under U_s, $\tau(U_s, \theta)$ is also a solution of M_s. Theorem 1 states that when both θ and $\tau(U_s, \theta)$ are transformed to V_Z via the channeling function $f_{s,Z}$, obtaining $f_{s,Z}(\theta)$ and $f_{s,Z}(\tau(U_s, \theta))$, we can always find a bijective mapping σ such that $\sigma(f_{s,Z}(\theta)) = f_{s,Z}(\tau(U_s, \theta))$. Since σ transforms solutions to solutions, it is a variable symmetry corresponding to τ.

In the SGP, the value symmetry τ under $U = \{g_{1,1}, \ldots, g_{6,1}\}$ with $\tau(1) = 2$, $\tau(2) = 3$, and $\tau(3) = 1$ corresponds to the variable symmetry σ in V_Z where σ is the identity except $\sigma(z_{i,1,1}) = z_{i,1,2}$, $\sigma(z_{i,1,2}) = z_{i,1,3}$, and $\sigma(z_{i,1,3}) = z_{i,1,1}$ for $1 \leq i \leq 6$.

3.3.2. *From Aspect Viewpoint to Aspect Viewpoint*

The previous theorem states that a value symmetry τ in M_s always corresponds to a variable symmetry in M_Z. However, we find that τ does *not* always correspond to a variable symmetry in M_t of aspect viewpoint V_t. The following theorem states the conditions when this correspondence occurs.

THEOREM 2. *Given a value symmetry τ under $U_s \subseteq X_s$, if*

1. *there exists $Obj'(k) \subseteq Obj(k)$ for $1 \leq k \leq n$ and $k \neq s$ such that $U_s = \{x[i_1 \cdots i_n \setminus i_s] \mid \bigwedge_{1\leq k\leq n, k\neq s} i_k \in Obj'(k)\}$, and*
2. *$Obj'(t) = Obj(t)$,*

then there is a mapping $\sigma : X_t \to X_t$ such that $\sigma(f_{s,t}(\theta)) = f_{s,t}(\tau(U_s, \theta))$ for all $\theta \in sol(M_s)$, where

$$\sigma(x[i_1 \cdots i_n \setminus i_t]) = \begin{cases} x[i'_1 \cdots i'_n \setminus i'_t] & \textit{if } \bigwedge_{1\leq k\leq n, k\neq s, k\neq t} i_k \in Obj'(k) \\ x[i_1 \cdots i_n \setminus i_t] & \textit{otherwise.} \end{cases}$$

In the mapping, $i'_j = i_j$ for $j \in \{1, \ldots, n\} \setminus \{s, t\}$ and $i'_s = \tau(i_s)$. In addition, σ is a variable symmetry in M_t corresponding to τ in M_s.

Proof. Without loss of generality, we assume $s < t$ in the proof. Let $\theta \in sol(M_s)$ and $B = \bigwedge_{1\leq k\leq n, k\neq s, k\neq t} i_k \in Obj'(k)$. If $U_s = \{x[i_1 \cdots i_n \setminus i_s] \mid \bigwedge_{1\leq k\leq n, k\neq s} i_k \in Obj'(k) \subseteq Obj(k)\}$, then

$$\begin{aligned} \tau(U_s, \theta) &= \{x[i_1 \cdots i_n \setminus i_s] \rhd i_s \mid \\ &\qquad (x[i_1 \cdots i_n \setminus i_s] \rhd i_s) \in \theta \wedge \neg(B \wedge i_t \in Obj'(t))\} \\ &\quad \cup \{x[i_1 \cdots i_n \setminus i_s] \rhd \tau(i_s) \mid \\ &\qquad (x[i_1 \cdots i_n \setminus i_s] \rhd i_s) \in \theta \wedge B \wedge i_t \in Obj'(t)\} \\ &\in sol(M_s) \end{aligned}$$

$$\begin{aligned} f_{s,t}(\tau(U_s, \theta)) &= f_{s,t}(\{x[i_1 \cdots i_n \setminus i_s] \rhd i_s \mid \\ &\qquad (x[i_1 \cdots i_n \setminus i_s] \rhd i_s) \in \theta \wedge \neg(B \wedge i_t \in Obj'(t))\}) \\ &\quad \cup f_{s,t}(\{x[i_1 \cdots i_n \setminus i_s] \rhd \tau(i_s) \mid \\ &\qquad (x[i_1 \cdots i_n \setminus i_s] \rhd i_s) \in \theta \wedge B \wedge i_t \in Obj'(t)\}) \\ &= \{x[i_1 \cdots i_n \setminus i_t] \rhd i_t \mid \\ &\qquad (x[i_1 \cdots i_n \setminus i_s] \rhd i_s) \in \theta \wedge \neg(B \wedge i_t \in Obj'(t))\} \\ &\quad \cup \{x_t[i_1] \cdots [i_{s-1}][\tau(i_s)][i_{s+1}] \cdots [i_{t-1}][i_{t+1}] \cdots [t_n] \rhd i_t \mid \\ &\qquad (x[i_1 \cdots i_n \setminus i_s] \rhd i_s) \in \theta \wedge B \wedge i_t \in Obj'(t)\} \\ &\in sol(M_t) \end{aligned}$$

If $Obj'(t) = Obj(t)$, then $i_t \in Obj'(t) \iff i_t \in Obj(t)$, which is always true. Hence,

$$\begin{aligned} f_{s,t}(\tau(U_s, \theta)) &= \{x[i_1 \cdots i_n \setminus i_t] \rhd i_t \mid (x[i_1 \cdots i_n \setminus i_s] \rhd i_s) \in \theta \wedge \neg B\} \\ &\quad \cup \{x_t[i_1] \cdots [i_{s-1}][\tau(i_s)][i_{s+1}] \cdots [i_{t-1}][i_{t+1}] \cdots [t_n] \rhd i_t \mid \\ &\qquad (x[i_1 \cdots i_n \setminus i_s] \rhd i_s) \in \theta \wedge B\} \end{aligned}$$

On the other hand,

$$\begin{aligned}
f_{s,t}(\theta) &= \{x[i_1 \cdots i_n \setminus i_t] \rhd i_t \,|\, (x[i_1 \cdots i_n \setminus i_s] \rhd i_s) \in \theta\} \\
&= \{x[i_1 \cdots i_n \setminus i_t] \rhd i_t \,|\, (x[i_1 \cdots i_n \setminus i_s] \rhd i_s) \in \theta \wedge \neg B\} \\
&\quad \cup \{x[i_1 \cdots i_n \setminus i_t] \rhd i_t \,|\, (x[i_1 \cdots i_n \setminus i_s] \rhd i_s) \in \theta \wedge B\} \\
&\in \mathit{sol}(M_s) \\
\sigma(f_{s,t}(\theta)) &= \sigma(\{x[i_1 \cdots i_n \setminus i_t] \rhd i_t \,|\, (x[i_1 \cdots i_n \setminus i_s] \rhd i_s) \in \theta \wedge \neg B\}) \\
&\quad \cup \sigma(\{x[i_1 \cdots i_n \setminus i_t] \rhd i_t \,|\, (x[i_1 \cdots i_n \setminus i_s] \rhd i_s) \in \theta \wedge B\}) \\
&= \{x[i_1 \cdots i_n \setminus i_t] \rhd i_t \,|\, (x[i_1 \cdots i_n \setminus i_s] \rhd i_s) \in \theta \wedge \neg B\} \\
&\quad \cup \{x_t[i_1] \cdots [i_{s-1}][\tau(i_s)][i_{s+1}] \cdots [i_{t-1}][i_{t+1}] \cdots [i_n] \rhd i_t \,| \\
&\qquad (x[i_1 \cdots i_n \setminus i_s] \rhd i_s) \in \theta \wedge B\} \\
&= f_{s,t}(\tau(U_s, \theta))
\end{aligned}$$

The bijective mapping σ preserves solutions in V_t. Hence, it is a variable symmetry in V_t and corresponds to the value symmetry τ in V_s. □

Theorem 2 shows that given a value symmetry τ under U_s in V_s, we can find a solution-preserving bijective mapping σ for variables in M_t (i.e., a variable symmetry in V_t) under two sufficient conditions. First, the variable subset U_s cannot be arbitrarily chosen. We need to ensure that $\mathit{Obj}'(k) \subseteq \mathit{Obj}(k)$ for $1 \leq k \leq n$ and $k \neq s$, i.e., the set of variable indices in U_s has to be the Cartesian product of a subset $\mathit{Obj}'(k)$ of the objects $\mathit{Obj}(k)$ in each aspect k except s. Second, $\mathit{Obj}'(t) = \mathit{Obj}(t)$, i.e., $\mathit{Obj}'(t)$ must contain all the objects in aspect t, which corresponds to the domains in V_t.

We illustrate the two conditions in Theorem 2 using the $(3, 2, 3)$ instance of the SGP. Let the golfers, weeks, and groups be the first, second, and third aspect respectively, giving $\mathit{Obj}(1) = \{1, \ldots, 6\}$ and $\mathit{Obj}(2) = \mathit{Obj}(3) = \{1, 2, 3\}$. In V_G, any value symmetry is under all the golfers in one week. For example, the value symmetry $\tau(1) = 2$, $\tau(2) = 3$, and $\tau(3) = 1$ is under $U = \{g_{1,1}, \ldots, g_{6,1}\} = \{g_{i,k} \,|\, (i, k) \in \mathit{Obj}'(1) \times \mathit{Obj}'(2) \wedge \mathit{Obj}'(1) = \{1, \ldots, 6\} \wedge \mathit{Obj}'(2) = \{1\}\}$, i.e., the set of all golfers in week 1. This satisfies condition (1) in Theorem 2. Consider V_P as the secondary viewpoint, which uses aspect 1 (golfers) to form the domains. Condition (2) is also satisfied because $\mathit{Obj}'(1) = \mathit{Obj}(1) = \{1, \ldots, 6\}$, i.e., $\mathit{Obj}'(1)$ contains all the golfers. Therefore, τ corresponds to a variable symmetry σ in V_P, with $\sigma(\langle p_{1,1}, p_{2,1}, p_{3,1}\rangle) = \langle p_{2,1}, p_{3,1}, p_{1,1}\rangle$. On the other hand, consider V_W as the secondary viewpoint, which uses aspect 2 (weeks) to form the domains, $\mathit{Obj}'(2) = \{1\} \neq \mathit{Obj}(2) = \{1, 2, 3\}$. Hence, condition (2) is *not* satisfied. In this case, τ does not correspond to any variable symmetry in V_W. Figure 2 shows the solution in V_W after applying τ to the solution in Figure

golfer \ group	1	2	3
1	$\{2,3\}$	$\{1\}$	$\emptyset$
2	$\emptyset$	$\{1,2,3\}$	$\emptyset$
3	$\{2\}$	$\{3\}$	$\{1\}$
4	$\emptyset$	$\emptyset$	$\{1,2,3\}$
5	$\{1\}$	$\{2\}$	$\{3\}$
6	$\{1,3\}$	$\emptyset$	$\{2\}$

Figure 2. Another Solution of $(3,2,3)$ Expressed in V_W

1(a). No variable symmetries can transform the solution in Figure 1(c) to the one in Figure 2.

3.4. Symmetry Breaking Constraints in Two Viewpoints

Recall that variable symmetry breaking constraints are easier to express than value symmetry breaking constraints. By Theorems 1 and 2, value symmetries in a matrix model (V, C) can correspond to variable symmetries in another matrix model (V', C') of the same MAP. We can thus break the value symmetries in (V, C) by combining (V, C) and $(V', C' \cup C_s)$ using channeling constraints [15], where C_s is the set of variable symmetry breaking constraints in V' for breaking the value symmetries in V. Since (V, C) and (V', C') are models for the same MAP, C' is logically redundant with respect to C and the channeling constraints. Hence, we can drop any of the constraints in C' when we connect V and V'. However, combining mutually redundant models with channeling constraints increases constraint propagation [15]. Therefore, a possible way is to drop only constraints in C' which are propagation redundant [16, 17] so that there would not be less propagation, but this is outside the scope of the paper. Note that if we drop *all* the constraints in C', then only (V, C) and (V', C_s) are combined, and V' is solely used for expressing the variable symmetry breaking constraints for the value symmetries in V. Variable symmetries in (V, C), if they exist, can be tackled by variable symmetry breaking constraints in V as well. Now that both variable and value symmetries can be tackled by symmetry breaking constraints and channeling constraints, we enjoy the best of both worlds.

An important issue of such symmetry breaking technique is the consistency of the symmetry breaking constraints in the two viewpoints V and V'. Two sets of symmetry breaking constraints are *consistent* [21] if and only if at least one element in each symmetry class of assignments, defined by the compositions of the symmetries under consideration, satisfies both sets of constraints. In row and column symmetries, Flener

$$
\begin{array}{cccccc}
\theta_1 & \theta_2 & \theta_3 & \theta_4 & \theta_5 & \theta_6 \\
\begin{pmatrix} 1 & 3 & 2 \\ 2 & 1 & 3 \\ 3 & 2 & 1 \end{pmatrix} &
\begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \\ 2 & 3 & 1 \end{pmatrix} &
\begin{pmatrix} 2 & 3 & 1 \\ 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix} &
\begin{pmatrix} 2 & 1 & 3 \\ 3 & 2 & 1 \\ 1 & 3 & 2 \end{pmatrix} &
\begin{pmatrix} 3 & 2 & 1 \\ 1 & 3 & 2 \\ 2 & 1 & 3 \end{pmatrix} &
\begin{pmatrix} 3 & 1 & 2 \\ 2 & 3 & 1 \\ 1 & 2 & 3 \end{pmatrix}
\end{array}
$$

(a) V_N

$$
\begin{array}{cccccc}
\theta'_1 & \theta'_2 & \theta'_3 & \theta'_4 & \theta'_5 & \theta'_6 \\
\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{pmatrix} &
\begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \\ 2 & 3 & 1 \end{pmatrix} &
\begin{pmatrix} 2 & 3 & 1 \\ 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix} &
\begin{pmatrix} 3 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix} &
\begin{pmatrix} 2 & 3 & 1 \\ 3 & 1 & 2 \\ 1 & 2 & 3 \end{pmatrix} &
\begin{pmatrix} 3 & 1 & 2 \\ 2 & 3 & 1 \\ 1 & 2 & 3 \end{pmatrix}
\end{array}
$$

(b) V_R

Figure 3. All Solutions of Order 3 QEP*, Expressed in V_N and V_R Respectively

et al. [21] showed that the row ordering constraints and the column ordering constraints are consistent symmetry breaking constraints. In our multiple viewpoint method, we would also want to show that symmetry breaking constraints in two viewpoints can be made consistent. In the following, we give first an example of inconsistent symmetry breaking constraints in two viewpoints, and then theoretical results on how to avoid such inconsistency problem.

3.4.1. *Inconsistent Symmetry Breaking Constraints in Two Viewpoints*

The quasigroup existence problem (QEP), "prob003" in CSPLib, is to find an $\mathcal{N} \times \mathcal{N}$ matrix consisting of numbers 1 to $\mathcal{N}$ with no rows and no columns containing the same number more than once. We consider the variant of the problem (QEP*) which further restricts the main ("southeast") diagonal of the matrix to contain the same number. Figure 3(a) shows all the six solutions of order 3 QEP* (i.e., $\mathcal{N} = 3$). The QEP* is a MAP with three aspects, namely the rows, columns, and numbers. Aspect viewpoint $V_N = (N, D_N)$ uses the rows and columns as indices to form the variables $n_{i,j} \in N$ and the numbers to form the domains $D_N(n_{i,j}) = \{1, \ldots, \mathcal{N}\}$.

In the QEP*, consider the symmetries of (1) the 180° rotation, and (2) the permutation of the numbers in the matrix. Symmetry (1) implies a variable symmetry σ in V_N, with $\sigma(n_{i,j}) = n_{n+1-i,n+1-j}$ for $1 \leq i, j \leq \mathcal{N}$. For order 3 QEP*, we have:

$n_{i,j}$	$n_{1,1}$	$n_{1,2}$	$n_{1,3}$	$n_{2,1}$	$n_{2,2}$	$n_{2,3}$	$n_{3,1}$	$n_{3,2}$	$n_{3,3}$
$\sigma(n_{i,j})$	$n_{3,3}$	$n_{3,2}$	$n_{3,1}$	$n_{2,3}$	$n_{2,2}$	$n_{2,1}$	$n_{1,3}$	$n_{1,2}$	$n_{1,1}$

Symmetry (2) implies that $\{1, \ldots, \mathcal{N}\}$ are indistinguishable values under N in V_N.

Consider a sequence $\vec{h} = \langle h_1, \ldots, h_{|N|} \rangle$ of variables in N. In other words, $h_i \in N$ for $1 \leq i \leq |N| = \mathcal{N}^2$. Symmetry (1) can be broken by symmetry breaking constraint $\vec{h} \leq_{lex} \sigma(\vec{h})$. Although we can form $\mathcal{N}!$ possible variable sequences from N, two common ways of flattening a matrix into sequences are the row-by-row and column-by-column traversals, giving

$$\vec{h}_r = \langle n_{1,1}, n_{1,2}, n_{1,3}, n_{2,1}, n_{2,2}, n_{2,3}, n_{3,1}, n_{3,2}, n_{3,3} \rangle \text{ and}$$
$$\vec{h}_c = \langle n_{1,1}, n_{2,1}, n_{3,1}, n_{1,2}, n_{2,2}, n_{3,2}, n_{1,3}, n_{2,3}, n_{3,3} \rangle$$

respectively for order 3 QEP*. The corresponding symmetry breaking constraints for τ are

$$\vec{h}_r \leq_{lex} \langle n_{3,3}, n_{3,2}, n_{3,1}, \ldots, n_{1,3}, n_{1,2}, n_{1,1} \rangle \text{ and}$$
$$\vec{h}_c \leq_{lex} \langle n_{3,3}, n_{2,3}, n_{1,3}, \ldots, n_{3,1}, n_{2,1}, n_{1,1} \rangle$$

respectively. Note that in order 3 QEP*, $n_{1,1} = n_{3,3}$. Also, $n_{1,2} \neq n_{3,2}$ and $n_{2,1} \neq n_{2,3}$. Therefore, the two constraints can be simplified to $n_{1,2} < n_{3,2}$ and $n_{2,1} < n_{2,3}$ respectively, which accept different solutions. Solutions θ_2, θ_4, and θ_6 in Figure 3(a) satisfy the former constraint, while θ_1, θ_3, and θ_5 satisfy the latter.

By Theorem 2, the value symmetries in V_N become variable symmetries in $V_R = (R, D_R)$, the aspect viewpoint using the numbers and columns to form the variables $r_{k,j} \in R$ and rows to form the domains $D_R(r_{k,j}) = \{1, \ldots, \mathcal{N}\}$. Both the row-by-row and column-by-column traversals of the matrix of variables in R generate, after simplifications [34], the same symmetry breaking constraints $\langle r_{k,1}, \ldots, r_{k,\mathcal{N}} \rangle \leq_{lex} \langle r_{k+1,1}, \ldots, r_{k+1,\mathcal{N}} \rangle$, or equivalently $r_{k,1} < r_{k+1,1}$, for $1 \leq k < \mathcal{N}$. Figure 3(b) shows the same six solutions as in Figure 3(a), but expressed in V_R. In the figure, solution θ'_i corresponds to solution θ_i in Figure 3(a) and the rows of the matrices θ'_i correspond to the number aspect. Only θ'_1 satisfies $r_{k,1} < r_{k+1,1}$, but θ_1 violates the variable symmetry breaking constraint $n_{1,2} < n_{3,2}$. Therefore there are no solutions satisfying $r_{k,1} < r_{k+1,1}$ and $n_{1,2} < n_{3,2}$ simultaneously, and hence they are inconsistent symmetry breaking constraints. On the other hand, θ_1 satisfies both $r_{k,1} < r_{k+1,1}$ and $n_{2,1} < n_{2,3}$ simultaneously. As we shall see, the latter pair of symmetry breaking constraints are consistent.

3.4.2. *Aspect Priorities, Scanning Sequences, and Selections*

We first define several notions which are useful to address the consistency issue for variable symmetry breaking constraints in two viewpoints. In a symmetry breaking constraint $\vec{h} \leq_{lex} \sigma(\vec{h})$ for a variable symmetry σ in an aspect viewpoint V_s, $\vec{h}$ is a sequence of variables in X_s, i.e., $\vec{h}$ is an arbitrary linearization of the matrix to a single dimensional sequence. Given $|X_s|$ variables, there are $|X_s|!$ possible combinations of variable sequences for X_s, and different sequences may generate different variable symmetry breaking constraints in V_s. The QEP* is an example. In the following, we restrict our attention to only the variable sequences generated by aspect priorities. An *aspect priority* in an aspect viewpoint V_s is a sequence of aspects which is a permutation of $\{1, \ldots, n\} \setminus \{s\}$. It is a permutation of all the aspects corresponding to the variable indices in V_s. Similarly, an *aspect priority* in the 0/1 viewpoint V_Z is a sequence of aspects which is a permutation of $\{1, \ldots, n\}$. For example, in the SGP, $\langle golfer, week\rangle$ and $\langle week, golfer\rangle$ are aspect priorities in V_G and $\langle golfer, week, group\rangle$ is an aspect priority in V_Z.

An aspect priority defines a scanning sequence of the variables in a viewpoint. A *scanning sequence* of an aspect priority $\langle k_1, \ldots, k_{n-1}\rangle$ of V_s, denoted by $sseq(\langle k_1, \ldots, k_{n-1}\rangle)$, is a sequence $\langle h_1, \ldots, h_{|X_s|}\rangle$ of X_s such that $h_a \equiv x[i_1 \cdots i_n \setminus i_s]$, where

$$a = 1 + \sum_{1 \leq l < n} ((i_{k_l} - 1) \times \prod_{l < m < n} |Obj(k_m)|).$$

A scanning sequence in a viewpoint is an aspect-by-aspect traversal of the matrix of variables in the viewpoint. There are $n-1$ aspects in an aspect priority in V_s, so there are $(n-1)!$ possible aspect priorities in V_s, and hence the same number of possible scanning sequences for the variables in V_s.

For example, there are three aspects in the QEP* (i.e., $n = 3$), giving $a = (i_{k_1} - 1) \times |Obj(k_2)| + i_{k_2}$ for aspect viewpoint V_N. Let aspects 1, 2, and 3 be the rows, columns, and numbers respectively. In order 3 QEP*, $|Obj(1)| = |Obj(2)| = |Obj(3)| = 3$. On the one hand, the aspect priority $\langle 1, 2\rangle$ ($\langle row, column\rangle$) thus gives $h_{(i-1)\times 3+j} \equiv n_{i,j}$, giving the scanning sequence

$$\vec{h}_r = \langle n_{1,1}, n_{1,2}, n_{1,3}, n_{2,1}, n_{2,2}, n_{2,3}, n_{3,1}, n_{3,2}, n_{3,3}\rangle.$$

On the other hand, the aspect priority $\langle 2, 1\rangle$ ($\langle column, row\rangle$) gives $h_{(j-1)\times 3+i} \equiv n_{i,j}$, giving the scanning sequence

$$\vec{h}_c = \langle n_{1,1}, n_{2,1}, n_{3,1}, n_{1,2}, n_{2,2}, n_{3,2}, n_{1,3}, n_{2,3}, n_{3,3}\rangle.$$

Note that the sequences $\vec{h}_r$ and $\vec{h}_c$ correspond to the row-by-row and column-by-column traversals of the matrix in V_N respectively.

The previous definition of scanning sequence is applicable to aspect viewpoints. We can define scanning sequences of aspect priorities in the 0/1 viewpoint similarly. A scanning sequence $sseq(\langle k_1, \ldots, k_n\rangle)$ of an aspect priority $\langle k_1, \ldots, k_n \rangle$ of V_Z is a sequence $\langle h_1, \ldots, h_{|Z|} \rangle$ of Z such that $h_a \equiv z[i_1] \cdots [i_n]$, where

$$a = 1 + \sum_{1 \leq l \leq n} ((i_{k_l} - 1) \times \prod_{l < m \leq n} |Obj(k_m)|).$$

There are $n!$ possible aspect priorities and scanning sequences in V_Z.

Selection of a sequence $\vec{h}$ under a variable set U, $select(\vec{h}, U)$, is a subsequence of $\vec{h}$ retaining only the variables in U. For example, in order 3 QEP*, selection of $\vec{h}_r$ under $U = \{n_{1,1}, n_{2,1}, n_{3,1}\}$ is $select(\vec{h}_r, U) = \langle n_{1,1}, n_{2,1}, n_{3,1} \rangle$. Similarly, $select(\vec{h}_r, \{n_{1,2}, n_{2,2}, n_{3,2}\}) = \langle n_{1,2}, n_{2,2}, n_{3,2} \rangle$.

3.4.3. *Generating Consistent Symmetry Breaking Constraints*

Before giving theorems to specify the conditions when symmetry breaking constraints in two viewpoints are consistent, we give two lemmas to state the ordering relationship between variables in aspect viewpoints and 0/1 viewpoint.

LEMMA 3. *Given two variables $x[i_1 \cdots i_n \setminus i_s]$ and $x[j_1 \cdots j_n \setminus j_s]$ in V_s and channeling function $f_{s,Z}$, we have*

$$x[i_1 \cdots i_n \setminus i_s] \leq x[j_1 \cdots j_n \setminus j_s] \iff \vec{z}_1 \geq_{lex} \vec{z}_2$$

where

$\vec{z}_1 = \langle z[i_1] \cdots [i_{s-1}][1][i_{s+1}] \cdots [i_n], \ldots, z[i_1] \cdots [i_{s-1}][|Obj(s)|][i_{s+1}] \cdots [i_n] \rangle$ *and*
$\vec{z}_2 = \langle z[j_1] \cdots [j_{s-1}][1][j_{s+1}] \cdots [j_n], \ldots, z[j_1] \cdots [j_{s-1}][|Obj(s)|][j_{s+1}] \cdots [j_n] \rangle$.

Proof. By $f_{s,Z}$, a compound decision of $\vec{z}_1$ is of the form $\vec{z}_1 \rhd \langle 0, \ldots, 0, 1, 0, \ldots, 0 \rangle$ (i.e., only one "1" and the rest are all "0"), and similarly for $\vec{z}_2$. When $x[i_1 \cdots i_n \setminus i_s] \leq x[j_1 \cdots j_n \setminus j_s]$, it means the "1" in $\vec{z}_1$ would never occur to the right of the "1" in $\vec{z}_2$, which means $\vec{z}_1 \geq_{lex} \vec{z}_2$. The opposite is also true. □

The lemma states that two variables $x[i_1 \cdots i_n \setminus i_s]$ and $x[j_1 \cdots j_n \setminus j_s]$ in V_s are in non-decreasing order if and only if their corresponding sequences of variables $\vec{z}_1$ and $\vec{z}_2$ in V_Z are in non-increasing lexicographic order, and vice versa. For example, consider a MAP with three aspects and $|Obj(s)| = 3$. When $\langle x_s[i], x_s[j] \rangle \rhd \langle 1, 2 \rangle$, i.e., $x_s[i] \leq x_s[j]$, we have $\langle z[i][1], z[i][2], z[i][3] \rangle \rhd \langle 1, 0, 0 \rangle$ and $\langle z[j][1], z[j][2], z[j][3] \rangle \rhd \langle 0, 1, 0 \rangle$, i.e., $\langle z[i][1], z[i][2], z[i][3] \rangle \geq_{lex} \langle z[j][1], z[j][2], z[j][3] \rangle$.

Lemma 3 can be generalized to sequences of variables in V_s instead of only single variables.

LEMMA 4. *Given two sequences $\vec{h}$ and $\vec{h}'$ of variables in V_s of equal length, and channeling function $f_{s,Z}$, we have*

$$\vec{h} \leq_{lex} \vec{h}' \iff \vec{z} \geq_{lex} \vec{z}'$$

where $\vec{z}$ and $\vec{z}'$ are respectively formed by replacing each variable $x[i_1 \cdots i_n \setminus i_s]$ in $\vec{h}$ and $\vec{h}'$ with

$$z[i_1] \cdots [i_{s-1}][1][i_{s+1}] \cdots [i_n], \ldots, z[i_1] \cdots [i_{s-1}][|Obj(s)|][i_{s+1}] \cdots [i_n].$$

Proof. Direct consequence of Lemma 3 extended to sequences of variables in V_s instead of two variables $x[i_1 \cdots i_n \setminus i_s]$ and $x[j_1 \cdots j_n \setminus j_s]$. □

We also recall the proposition by Crawford et al. [18] which states the consistency of variable symmetry breaking constraints in 0/1 viewpoint using lexicographic ordering.

PROPOSITION 5. *[18] Let M_Z be a CSP model in 0/1 viewpoint $V_Z = (Z, D_Z)$ and $\vec{h}$ be a sequence of variables in Z. Then the constraints*

$$\vec{h} \leq_{lex} \sigma(\vec{h})$$

for each variable symmetry σ in M_Z are satisfied only by the lexicographically smallest solution in each equivalence class of solutions. Hence, the constraints are consistent for M_Z.

Using Lemmas 3 and 4 and Proposition 5, we can state sufficient conditions for consistent symmetry breaking constraints in two viewpoints. We start with the issue between an aspect viewpoint V_s and the 0/1 viewpoint V_Z. The following theorem applies to any value symmetries.

THEOREM 6. *Let:*

- *σ be a variable symmetry in V_s;*
- *σ' be a variable symmetry in V_Z corresponding to a value symmetry in V_s;*
- *$\vec{k} = \langle k_1, \ldots, k_{n-1} \rangle$ be an aspect priority in V_s; and*
- *$\vec{x}_s = sseq(\vec{k})$ and $\vec{z}$ be sequences of variables in X_s and Z respectively.*

If

$$\vec{z} = sseq(\langle k_1, \ldots, k_{n-1}, s \rangle),$$

then symmetry breaking constraints $\vec{x}_s \leq_{lex} \sigma(\vec{x}_s)$ for σ and $\sigma'(\vec{z}) \leq_{lex} \vec{z}$ for σ' are consistent.

Proof. Let $\sigma(x[i_1 \cdots i_n \setminus i_s]) = x[j_1 \cdots j_n \setminus j_s]$. The variable symmetry σ in V_s corresponds to another variable symmetry σ_z in V_Z where $\sigma_z(z[i_1] \cdots [i_n]) = z[j_1] \cdots [j_{s-1}][i_s][j_{s+1}] \cdots [j_n]$ for $\bigwedge_{1 \le k \le n} i_k \in Obj(k)$. When $\vec{z} = sseq(\langle k_1, \ldots, k_{n-1}, s \rangle)$, $\vec{z}$ can be constructed from $\vec{x}_s$ by substituting each variable $x[i_1 \cdots i_n \setminus i_s]$ in $\vec{x}_s$ with

$$z[i_1] \cdots [i_{s-1}][1][i_{s+1}] \cdots [i_n], \ldots, z[i_1] \cdots [i_{s-1}][|Obj(s)|][i_{s+1}] \cdots [i_n].$$

By Lemma 4, $\vec{x}_s \le_{lex} \sigma(\vec{x}_s) \iff \vec{z} \ge_{lex} \sigma_z(\vec{z})$. By Proposition 5, $\vec{z} \ge_{lex} \sigma_z(\vec{z})$ and $\vec{z} \ge_{lex} \sigma'(\vec{z})$ are consistent, and hence so do $\vec{x}_s \le_{lex} \sigma(\vec{x}_s)$ and $\vec{z} \ge_{lex} \sigma'(\vec{z})$. □

To maintain consistency between the variable symmetry breaking constraints for σ in V_s and σ' in V_Z, Theorem 6 requires that the scanning sequence $sseq(\langle k_1, \ldots, k_{n-1}, s \rangle)$ in V_Z is used. That means the aspect priority in V_Z is the sequence $\langle k_1, \ldots, k_{n-1} \rangle$ followed by s in the last position. Furthermore, the lexicographic order in V_Z is reverse of that in V_s. This is because by Lemma 4, a smaller-than order in V_s corresponds to a greater-than order in V_Z, and vice versa.

In the SGP, Theorem 6 ensures that the variable symmetry breaking constraints $\langle z_{1,k,j}, \ldots, z_{\mathcal{N},k,j} \rangle \ge_{lex} \langle z_{1,k,j+1}, \ldots, z_{\mathcal{N},k,j+1} \rangle$ for $1 \le j < \mathcal{G}$ and $1 \le k \le \mathcal{W}$ in V_Z break the value symmetries in V_G, and are consistent with those variable symmetry breaking constraints in V_G.

The condition when variable symmetries in V_t, corresponding to value symmetries in V_s, can be broken consistently with the variable symmetries in V_s is more difficult to specify. In the 0/1 viewpoint V_Z, there is one more aspect in the variable indices than V_s. We can simply add an aspect to an aspect priority in V_s to form an aspect priority in V_t. Aspect viewpoints V_s and V_t, however, have the same number of aspects as variable indices. We cannot use the same technique to form aspect priorities. Instead, we consider the special class of value symmetries, the symmetries of indistinguishable values, which have a special form of symmetry breaking constraints in V_t to allow us to specify the consistency condition.

THEOREM 7. *Let:*

- σ *be a variable symmetry in* V_s*;*

- τ *be a value symmetry of two indistinguishable values* a *and* b *(where* $a < b$*) under* $U_s = \{x[i_1 \cdots i_n \setminus i_s] \mid \bigwedge_{1 \le k \le n, k \ne s} i_k \in Obj'(k) \subseteq Obj(k)\}$ *in* V_s *with* $Obj'(t) = Obj(t)$*;*

- $\vec{k} = \langle k_1, \ldots, k_{n-2} \rangle$ *be a permutation of* $\{1, \ldots, n\} \setminus \{s, t\}$*; and*

– *$\vec{q}$ be any aspect priority in V_t formed by inserting s into $\vec{k}$ (i.e., $\vec{q}$ is a permutation of $\{1,\ldots,n\}\setminus\{t\}$ and $\vec{k}$ is a subsequence of $\vec{q}$).*

If

$$\vec{h} = sseq(\langle k_1,\ldots,k_{n-2},t\rangle),$$

then symmetry breaking constraints $\vec{h} \leq_{lex} \sigma(\vec{h})$ for σ and $\vec{h}'_a \leq_{lex} \vec{h}'_b$ for σ' are consistent, where σ' is the variable symmetry in V_t corresponding to the value symmetry τ in V_s, $\vec{h}'_j = select(sseq(\vec{q}), U'_j)$ for $j \in \{a,b\}$, and $U'_j = \{x[i_1\cdots i_n \setminus i_t] \mid i_s = j \wedge \bigwedge_{1\leq k\leq n, k\neq s, k\neq t} i_k \in Obj'(k)\}$.

Proof. By Theorem 1, the value symmetry τ in V_s corresponds to a variable symmetry σ'' in V_Z, where

$$\sigma''(z[i_1]\cdots[i_n]) = \begin{cases} z[i_1]\cdots[i_{s-1}][b][i_{s+1}]\cdots[i_n] & \text{if } i_s = a \wedge \bigwedge_{1\leq k\leq n, k\neq s} i_k \in Obj'(k) \\ z[i_1]\cdots[i_{s-1}][a][i_{s+1}]\cdots[i_n] & \text{if } i_s = b \wedge \bigwedge_{1\leq k\leq n, k\neq s} i_k \in Obj'(k) \\ z[i_1]\cdots[i_n] & \text{otherwise.} \end{cases}$$

If $\vec{h} = sseq(\langle k_1,\ldots,k_{n-2},t\rangle)$, then by Theorem 6, $\vec{h} \leq_{lex} \sigma(\vec{h})$ is consistent with $\vec{z} \geq_{lex} \sigma''(\vec{z})$, where $\vec{z} = sseq(\langle k_1,\ldots,k_{n-2},t,s\rangle)$. From the definition of σ'', we can see that $\vec{z} \geq_{lex} \sigma''(\vec{z})$ can be simplified to

$$select(\vec{z}, U''_a) \ \geq_{lex} \ select(\vec{z}, U''_b), \tag{1}$$

where $U''_j = \{z[i_1]\cdots[i_n] \mid i_s = j \wedge \bigwedge_{1\leq k\leq n, k\neq s} i_k \in Obj'(k)\}$ for $j \in \{a,b\}$. Note that the indices i_s of all variables in U''_j are fixed to j (i.e., either a or b). Therefore, for any aspect priority $\vec{w}$ in V_Z formed by inserting s into $\langle k_1,\ldots,k_{n-2},t\rangle$, $select(sseq(\vec{w}), U''_j) \equiv select(\vec{z}, U''_j)$ is always true. Hence, constraint (1) is now equivalent to

$$select(sseq(\vec{w}), U''_a) \ \geq_{lex} \ select(sseq(\vec{w}), U''_b). \tag{2}$$

In particular, consider $\vec{w} = \langle q_1,\ldots,q_{n-1},t\rangle$. Recall that $\vec{q} = \langle q_1,\ldots,q_{n-1}\rangle$ is an aspect priority in V_t formed by inserting s into $\langle k_1,\ldots,k_{n-2}\rangle$. By Lemma 4, constraint (2) is equivalent to

$$select(sseq(\vec{q}), U'_a) \ \leq_{lex} \ select(sseq(\vec{q}), U'_b), \tag{3}$$

where $U'_j = \{x[i_1\cdots i_n \setminus i_t] \mid i_s = j \wedge \bigwedge_{1\leq k\leq n, k\neq s} i_k \in Obj'(k)\}$ for $j \in \{a,b\}$. Constraint $\vec{h} \leq_{lex} \sigma(\vec{h})$ is consistent with $\vec{z} \geq_{lex} \sigma''(\vec{z})$, which is equivalent to constraint (3), therefore $\vec{h} \leq_{lex} \sigma(\vec{h})$ is consistent with constraint (3). □

Suppose a symmetry of two indistinguishable values in V_s corresponds to a variable symmetry in V_t, and we lexicographically order the variables in V_t corresponding to the indistinguishable values (i.e.,

$\vec{h}'_a \leq_{lex} \vec{h}'_b$ in the theorem). Theorem 7 states that when generating the variable symmetry breaking constraints in V_s, aspect t (corresponding to the domain in V_t) must be least prioritized in the aspect priority in V_s. In such case, consistency between the symmetry breaking constraints in V_s and V_t are guaranteed.

Theorem 7 is applicable to symmetries of two indistinguishable values. It can be generalized to handle multiple indistinguishable values.

COROLLARY 8. *Let:*

- *σ be a variable symmetry in V_s;*
- *$\{v_1, \ldots, v_m\}$ be a set of indistinguishable values (where $v_1 < \ldots < v_m$) under $U_s = \{x[i_1 \cdots i_n \setminus i_s] \mid \bigwedge_{1 \leq k \leq n, k \neq s} i_k \in Obj'(k) \subseteq Obj(k)\}$ in V_s with $Obj'(t) = Obj(t)$;*
- *$\vec{k} = \langle k_1, \ldots, k_{n-2} \rangle$ be a permutation of $\{1, \ldots, n\} \setminus \{s, t\}$; and*
- *$\vec{q}$ be any aspect priority in V_t formed by inserting s into $\vec{k}$ (i.e., $\vec{q}$ is a permutation of $\{1, \ldots, n\} \setminus \{t\}$ and $\vec{k}$ is a subsequence of $\vec{q}$).*

If

$$\vec{h} = sseq(\langle k_1, \ldots, k_{n-2}, t \rangle),$$

then symmetry breaking constraints $\vec{h} \leq_{lex} \sigma(\vec{h})$ for σ and $\vec{h}'_1 \leq_{lex} \ldots \leq_{lex} \vec{h}'_m$ for the symmetries of indistinguishable values are consistent, where $\vec{h}'_j = select(sseq(\vec{q}), U'_j)$ for $1 \leq j \leq m$ and $U'_j = \{x[i_1 \cdots i_n \setminus i_t] \mid i_s = j \wedge \bigwedge_{1 \leq k \leq n, k \neq s, k \neq t} i_k \in Obj'(k)\}$.

Proof. Direct consequence of Theorem 7 for all pairs of indistinguishable values. □

For the QEP* example in Section 3.4.1, the symmetry breaking constraint, say, $\langle r_{1,1}, \ldots, r_{1,\mathcal{N}} \rangle \leq_{lex} \langle r_{2,1}, \ldots, r_{2,\mathcal{N}} \rangle$, in V_R corresponds to the constraint $\vec{h}_a \leq_{lex} \vec{h}_b$ in the theorem. There are two possible aspect priorities $\langle 2, 3 \rangle$ and $\langle 3, 2 \rangle$ in V_R, which means $\langle column, number \rangle$ and $\langle number, column \rangle$ respectively. The variable sequence $\langle r_{1,1}, \ldots, r_{1,\mathcal{N}} \rangle$ is the selection of the scanning sequence of both aspect priorities with index value 1 in aspect 3 (numbers), i.e., $\langle r_{1,1}, \ldots, r_{1,\mathcal{N}} \rangle = select(sseq(\langle 2, 3 \rangle), U') = select(sseq(\langle 3, 2 \rangle), U')$ where $U' = \{r_{1,1}, \ldots, r_{1,\mathcal{N}}\}$. Similarly for $\langle r_{2,1}, \ldots, r_{2,\mathcal{N}} \rangle$. Therefore, according to Theorem 7, the variable symmetry breaking constraints in V_N must be generated using the scanning sequence of the aspect priority $\langle 2, 1 \rangle$, i.e., aspect 1 (rows) must be least prioritized, to maintain consistency between the symmetry breaking constraints in V_N and V_R. The variable symmetry breaking constraint $n_{2,1} < n_{2,3}$

is generated using the scanning sequence of the aspect priority $\langle 2, 1\rangle$. Thus, it is consistent with the symmetry breaking constraints in V_R.

Consider again the value symmetries in V_G of the SGP. By Theorem 2, they correspond to variable symmetries in V_P. Theorem 7 and Corollary 8 ensure that the symmetry breaking constraints $\min(p_{j,k}) < \min(p_{j+1,k})$ for $1 \leq j < \mathcal{G}$ and $1 \leq k \leq \mathcal{W}$ in V_P (the degenerated lexicographic ordering constraints for set variables) breaks the value symmetries in V_G. These constraints are consistent with the row and column lexicographic ordering constraints in V_G, which are the simplification results of those generated by both aspect priorities $\langle golfer, week\rangle$ and $\langle week, golfer\rangle$ in V_G. The solution in Figure 1(a) satisfies both types of symmetry breaking constraints.

4. Value Precedence Constraints

The method discussed in the previous section makes use of existing modeling techniques; no new algorithms have to be designed. In this section, we propose another method which tackles an important and common class of value symmetries, namely symmetries of indistinguishable values. This method requires designing new propagation algorithms. In the following, we introduce the notion of value precedence on integer and set sequences and show how the notion can be used to break symmetries of indistinguishable values. Two propagation algorithms for implementing integer and set value precedence global constraints are presented. We also study some theoretical properties attained by various usages of the global constraints.

4.1. Integer and Set Value Precedence

Value precedence of s over t in an integer sequence $\vec{q} = \langle q_0, \ldots, q_{n-1}\rangle$ means that if there exists j such that $q_j = t$, then there must exist $i < j$ such that $q_i = s$. We say that value s is an *antecedent* while value t is a *subsequent*, and that the antecedent s *precedes* the subsequent t in $\vec{q}$, written as $s \prec_{\vec{q}} t$. For example, the sequence $\vec{q} = \langle 0, 2, 2, 1, 0, 1\rangle$ implies $0 \prec_{\vec{q}} 1$, $0 \prec_{\vec{q}} 2$, and $2 \prec_{\vec{q}} 1$. Note that if a value j does not appear in $\vec{q}$, then $i \prec_{\vec{q}} j$ is true for any i. In the previous example, $0 \prec_{\vec{q}} 3$ and $4 \prec_{\vec{q}} 3$ are thus also true. Note also that value precedence is transitive: if $i \prec_{\vec{q}} j$ and $j \prec_{\vec{q}} k$, then $i \prec_{\vec{q}} k$.

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 $\vec{q}$ of sets means that if there exists j such that $s \notin q_j$ and $t \in q_j$, then there must exist $i < j$ such that $s \in q_i$ and $t \notin q_i$. For example, consider the sequence $\vec{q} = \langle\{0,2\},\{0,1\},\emptyset,\{1\}\rangle$. We have $0 \prec_{\vec{q}} 1$ and $2 \prec_{\vec{q}} 1$. We also have $0 \prec_{\vec{q}} 2$, because there is no set in $\vec{q}$ that contains 2 but not 0. Again, if j does not belong to any set in $\vec{q}$, then $i \prec_{\vec{q}} j$ is true for any i. Thus, we also have, say, $0 \prec_{\vec{q}} 4$. Note that set value precedence degenerates to integer value precedence when the cardinality of each set in the sequence is one, because in such case, $t \in q_j$ implies $s \notin q_j$ and $s \in q_i$ implies $t \notin q_i$.

4.1.1. *Value Precedence and Indistinguishable Values*

Value precedence can be used for breaking symmetries of indistinguishable values. Given two indistinguishable values under some variables U in a CSP, we can break the symmetry of the values by maintaining value precedence for them. We have to construct a sequence $\vec{u}$ 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. For example, suppose there are two indistinguishable values $\{0,1\}$ under $\{x_1,x_2,x_3\}$ in a CSP $M = (X,D,C)$, where $X = \{x_0,\ldots,x_4\}$ is a set of set variables. If $\vec{x} \mapsto \langle\{1,2\},\{0,2\},\{0,1\},\{1,2\},\emptyset\rangle$ is a solution of M, where $\vec{x} = \langle x_0,\ldots,x_4\rangle$, then $\vec{x} \mapsto \langle\{1,2\},\{1,2\},\{0,1\},\{0,2\},\emptyset\rangle$ should be another solution of M. We can let $\vec{u} = \langle x_1,x_2,x_3\rangle$ and add the constraint $0 \prec_{\vec{u}} 1$ on variables x_1, x_2, and x_3 to M to break the symmetry. Thus, $\vec{x} \mapsto \langle\{1,2\},\{0,2\},\{0,1\},\{1,2\},\emptyset\rangle$ remains a solution, but its symmetrical counterpart $\vec{x} \mapsto \langle\{1,2\},\{1,2\},\{0,1\},\{0,2\},\emptyset\rangle$ would now be rejected because $0 \prec_{\vec{u}} 1$ is false.

In general, there can be more than two indistinguishable values in a CSP. The following theorem states that we can always use the value precedence $v_0 \prec_{\vec{u}} \ldots \prec_{\vec{u}} v_{k-1}$ to completely break the symmetries of a set of indistinguishable values $V = \{v_0,\ldots,v_{k-1}\}$ under U, where $\vec{u}$ is a sequence of U. For example, if $V = \{0,1,2,3\}$, then we can maintain $0 \prec_{\vec{u}} 1 \prec_{\vec{u}} 2 \prec_{\vec{u}} 3$.

THEOREM 9. *Given a set of indistinguishable values $\{v_0,\ldots,v_{k-1}\}$ under U, in each equivalence class of solutions induced by the symmetries, there is exactly one solution satisfying the value precedence $v_0 \prec_{\vec{u}} \ldots \prec_{\vec{u}} v_{k-1}$, where $\vec{u}$ is a sequence of the variables in U.*

Proof. Given a solution in an equivalence class, for $0 \leq i < k-1$ and $i < j \leq k-1$, if $v_i \prec_{\vec{u}} v_j$ is false, we swap the occurrences of v_i and v_j in the solution to obtain another one with $v_i \prec_{\vec{u}} v_j$ satisfied. After iterating a value for i, we have maintained the value precedence $v_0 \prec_{\vec{u}} \ldots \prec_{\vec{u}} v_i$. Hence, after all, we construct a solution with $v_0 \prec_{\vec{u}} \ldots \prec_{\vec{u}} v_{k-1}$, and swapping the occurrences of any two values in $\{v_0,\ldots,v_{k-1}\}$ would violate this value precedence. □

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 [21]. For example, we have a CSP $M = ((\{x, y\}, D), \{x \neq y\})$, where $D(x) = D(y) = \{1, 2\}$. CSP M has (1) the variable symmetry σ such that $\sigma(\langle x, y\rangle) = \langle y, x\rangle$, and (2) values 1 and 2 are indistinguishable. To break symmetry (1), 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 (2). These two constraints result in no solution, which is undesirable. The following theorem shows when maintaining $s \prec_{\vec{u}} t$ is consistent with variable symmetry breaking constraints.

THEOREM 10. *Let X be the set of variables of a CSP M, and $\vec{x} = \langle x_0, \ldots, x_{n-1}\rangle$ and $\vec{u}$ be sequences of variables in X and $U \subseteq X$ respectively. Suppose σ is a variable symmetry in M and s and t are any two integer indistinguishable values under U. The value precedence constraint $s \prec_{\vec{u}} t$ (*resp. *$t \prec_{\vec{u}} s$) is consistent with the variable symmetry breaking constraint $\vec{x} \leq_{lex} \sigma(\vec{x})$ (*resp. *$\sigma(\vec{x}) \leq_{lex} \vec{x}$) if*

- *$s < t$ (*resp. *$t < s$) and*
- *$\vec{u}$ is a subsequence of $\vec{x}$, i.e., $\vec{u}$ can be formed by deleting some elements from $\vec{x}$.*

Furthermore, if $\vec{x}$ is a sequence of set variables, then the $\leq_{set}$ ordering should be used to compare two sets p and q instead of the $\leq$ ordering to compare two numbers, where $p \leq_{set} q$ if and only if (1) $q = \emptyset$, or (2) $\min(p) < \min(q)$, or (3) $\min(p) = \min(q) \wedge p \setminus \{\min(p)\} \leq_{set} q \setminus \{\min(q)\}$.

Proof. We prove the case of $s \prec_{\vec{u}} t$ and $\vec{x} \leq_{lex} \sigma(\vec{x})$ only since the case of $t \prec_{\vec{u}} s$ and $\sigma(\vec{x}) \leq_{lex} \vec{x}$ is analogous. In each equivalence class of solutions induced by the variable symmetry σ, the symmetry breaking constraint $\vec{x} \leq_{lex} \sigma(\vec{x})$ keeps the lexicographically smaller solution with respect to the sequence $\vec{x}$. If $s < t$, $s \prec_{\vec{u}} t$ also keeps the lexicographically smaller solution with respect to the sequence $\vec{u}$ in each equivalence class of solutions induced by the indistinguishable values. If $\vec{u}$ is a subsequence of $\vec{x}$, then a lexicographic smaller solution with respect to $\vec{u}$ is also a lexicographic smaller one with respect to $\vec{x}$. Hence, $s \prec_{\vec{u}} t$ is consistent with $\vec{x} \leq_{lex} \sigma(\vec{x})$ if $s < t$ and $\vec{u}$ is a subsequence of $\vec{x}$.

The $\leq_{set}$ ordering should be used to compare two sets p and q, since both orderings $\leq_{set}$ for sets and $\leq$ for numbers are equivalent to lexicographic ordering $\geq_{lex}$ on the Boolean (or occurrence) representations of sets and numbers respectively. □

According to Theorem 10, $x \leq y$ and $1 \prec_{\langle x,y \rangle} 2$ are consistent, resulting in a single solution $\langle x,y \rangle \mapsto \langle 1,2 \rangle$. Similarly, $y \leq x$ and $2 \prec_{\langle x,y \rangle} 1$ are consistent, resulting in a single solution $\langle x,y \rangle \mapsto \langle 2,1 \rangle$.

The definition of the $\leq_{set}$ ordering, similar to but different from that of multiset ordering [25], suggests that $\emptyset$ is the largest element in the ordering, and the ordering degenerates to $\leq$ for numbers when the cardinalities of the two sets are 1. For example, $\{1,2\} \leq_{set} \{1,3,4\} \leq_{set} \{1,3\}$.

This $\leq_{set}$ ordering on two sets has the property that it is equivalent to lexicographic ordering $\geq_{lex}$ on the Boolean representations of the two sets. Take the ordering $\{1,2\} \leq_{set} \{1,3,4\} \leq_{set} \{1,3\}$ as an example. Suppse we are recording the occurrences of values 1 to 4. The Boolean representations of the sets $\{1,2\}$, $\{1,3,4\}$, and $\{1,3\}$ are $\langle 1,1,0,0 \rangle$, $\langle 1,0,1,1 \rangle$, and $\langle 1,0,1,0 \rangle$ respectively. We can see that $\langle 1,1,0,0 \rangle \geq_{lex} \langle 1,0,1,1 \rangle \geq_{lex} \langle 1,0,1,0 \rangle$. This property is in parallel to the $\leq$ ordering on numbers stated in Lemma 3.

4.1.2. *Constraints for Maintaining Value Precedence*

Constraints to enforce value precedence $s \prec_{\vec{x}} t$ for a sequence of constrained variables $\vec{x}$ can be constructed straightforwardly from its declarative meaning. In subsequent discussions, we assume that $s \neq t$ and the sequence $\vec{x}$ contains different variables, i.e., the same variable cannot occur more than once in $\vec{x}$. Suppose $\vec{x}$ is a sequence of integer variables. Since s must precede t, x_0, the first variable in $\vec{x}$, must not be assigned t. The constraints are then

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 $\vec{x}$ 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 variables, GAC on the *conjunction* of the n if-then value precedence constraints is equivalent to GAC on each individual if-then value precedence constraint.

THEOREM 11. *Given an integer variable sequence $\vec{x}$, GAC on $s \prec_{\vec{x}} t$ is equivalent to GAC on each individual if-then value precedence constraint for integer variables.*

Proof. Let c_0 and c_j be the constraints $x_0 \neq t$ and $x_j = t \rightarrow \bigvee_{0 \leq i < j} x_i = s$ for $1 \leq j < n$ respectively. GAC on $s \prec_{\vec{x}} t$ is clearly no weaker than GAC on c_j for $0 \leq j < n$ individually. Conversely, suppose each c_j for $0 \leq j < n$ is GAC individually but $s \prec_{\vec{x}} t$ is not GAC. That is, there exists an assignment such that any of its extensions fails to satisfy $s \prec_{\vec{x}} t$. We show by induction that if such an assignment exists, then $s, t \notin D(x_j)$ for $0 \leq j < n$.

As the base case, we have $s, t \notin D(x_0)$ because $x_0 \mapsto s$ alone will satisfy $s \prec_{\vec{x}} t$ and c_0 is GAC. Given $s, t \notin D(x_i)$ for $0 \leq i < j$, $x_j \mapsto s$ alone will satisfy $s \prec_{\vec{x}} t$. Therefore, in order to fail $s \prec_{\vec{x}} t$, we must have $s \notin D(x_j)$. Furthermore, since c_j is GAC, $\bigwedge_{1 \leq i < j} s, t \notin D(x_i)$ implies $t \notin D(x_j)$. Hence, by induction, we have $\bigwedge_{0 \leq j < n} s, t \notin D(x_j)$. However, all possible compound assignments $\vec{x} \mapsto \langle u_0, \ldots, u_{n-1} \rangle$ with $u_i \in D(x_i) \setminus \{s, t\}$ are solutions of $s \prec_{\vec{x}} t$. Thus, $s \prec_{\vec{x}} t$ is also GAC and therefore GAC on $s \prec_{\vec{x}} t$ is equivalent to GAC on c_j for $1 \leq j < n$ individually. $\square$

For set variables, SBC on the *conjunction* of the n if-then value precedence constraints is equivalent to SBC on each individual if-then value precedence constraint.

THEOREM 12. *Given a set variable sequence $\vec{x}$, SBC on $s \prec_{\vec{x}} t$ is equivalent to SBC on each individual if-then value precedence constraint for set variables.*

Proof. Let c_0 and c_j be the 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 $1 \leq j < n$ respectively. SBC on $s \prec_{\vec{x}} t$ is clearly no weaker than SBC on c_j for $0 \leq j < n$ individually. Conversely, suppose each c_j for $0 \leq j < n$ is SBC but $s \prec_{\vec{x}} t$ is not SBC. That is, there exists either an assignment $x_i \mapsto RS(x_i)$ or $x_i \mapsto PS(x_i)$ such that any of its extensions fails to satisfy $s \prec_{\vec{x}} t$. We show by induction that if such an assignment exists, then $(s \notin RS(x_j) \vee t \in PS(x_j)) \wedge (s \in RS(x_j) \vee t \notin PS(x_j))$ for $0 \leq j < n$.

As the base case, since $s \in RS(x_0) \wedge t \notin PS(x_0)$ always satisfies $s \prec_{\vec{x}} t$, in order to fail $s \prec_{\vec{x}} t$, we must have $s \notin RS(x_0) \vee t \in PS(x_0)$. Also, since c_0 is SBC, we have $s \in RS(x_0) \vee t \notin PS(x_0)$.

Given $(s \notin RS(x_i) \vee t \in PS(x_i)) \wedge (s \in RS(x_i) \vee t \notin PS(x_i))$ for $0 \leq i < j$, $s \in RS(x_j) \wedge t \notin PS(x_j)$ always satisfy $s \prec_{\vec{x}} t$. Therefore, we have $s \notin RS(x_j) \vee t \in PS(x_j)$. Furthermore, since c_j is SBC, $\bigwedge_{0 \leq i < j} ((s \notin RS(x_i) \vee t \in PS(x_i)) \wedge (s \in RS(x_i) \vee t \notin PS(x_i)))$ implies $s \in RS(x_j) \vee t \notin PS(x_j)$. Hence, by induction, we have $\bigwedge_{0 \leq j < n} ((s \notin RS(x_j) \vee t \in$

$PS(x_j)) \wedge (s \in RS(x_j) \vee t \notin PS(x_j)))$. However, all possible compound assignments $\vec{x} \mapsto \langle u_0, \ldots, u_{n-1} \rangle$ with $(s \notin RS(x_j) \vee t \in PS(x_j)) \wedge (s \in RS(x_j) \vee t \notin PS(x_j))$ for $0 \leq j < n$ are solutions of $s \prec_{\vec{x}} t$. Thus, $s \prec_{\vec{x}} t$ is also SBC and therefore SBC on $s \prec_{\vec{x}} t$ is equivalent to SBC on c_j for $1 \leq j < n$ individually. □

4.2. Propagation Algorithms for Value Precedence

We develop two propagation algorithms **IntValuePrecede** and **SetValuePrecede** to implement two global constraints for integer and set value precedence respectively. Both global constraints use the same prototype $ValuePrecede(\vec{x}, s, t)$, meaning $s \prec_{\vec{x}} t$, where $\vec{x}$ is a variable sequence and s and t are integer constants. GAC (*resp.* SBC) is enforced on the integer (*resp.* set) value precedence constraint. The integer and set versions of the propagation algorithms are similar. Their time complexity is *linear to the length of* $\vec{x}$. Both of them use three pointers α, β, and γ to point to different indices of $\vec{x}$, but the pointers have different meanings for the two versions. The two algorithms are also similar to that of the lexicographic ordering global constraint [26] in the sense that both maintain pointers running in opposite directions from the two ends of variable sequences. On the other hand, they are different from those of some other global constraints which are developed using automata constructions [14, 13, 9]. In subsequent discussions, we assume the variable sequence $\vec{x} = \langle x_0, \ldots, x_{n-1} \rangle$ to be a sequence of non-repeating variables and s and t to be distinct.

4.2.1. *Integer Version*

In **IntValuePrecede**, pointer α is the smallest index of $\vec{x}$ 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 $\vec{x}$ have value s in their domains, then we define $\alpha = n$. Pointer β is the second smallest index of $\vec{x}$ 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 fewer than two variables in $\vec{x}$ contain value s in their domains, then we define that $\beta = n$. Pointer γ is the smallest index of $\vec{x}$ 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 $\vec{x}$ 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 $\vec{x} = \langle x_0, x_1, x_2, x_3 \rangle$, $s = 1$, and $t = 2$. Suppose we have: Suppose $D(x_0) = \{2, 3\}$, $D(x_1) = \{1, 2, 3\}$, $D(x_2) = \{2\}$, and $D(x_3) = \{1, 3\}$. Then, we have $\alpha = 1$, $\beta = 3$, and $\gamma = 2$.

Recall the integer if-then value precedence constraints $x_0 \neq t$ and $x_j = t \rightarrow \bigvee_{0 \leq i < j} x_i = s$ for $1 \leq j < n$. Pointer α tells that $s \notin D(x_i)$

for $0 \leq i < \alpha$. Thus, we can remove t from $D(x_i)$ for $0 \leq i \leq \alpha$. Our first pruning rule is that:

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

In the above example, we have $\alpha = 1$. Therefore, we can remove value $t = 2$ from the domains of x_0 and x_1 as shown in Figure 4(a).

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

(a)

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

(b)

Figure 4. Illustrating the Use of the Pointers α, β, and γ in **IntValuePrecede**

Pointer γ tells the smallest index of $\vec{x}$ such that x_γ is bound to t. Therefore, according to the if-then value precedence constraints, the constraint $\bigvee_{0 \leq i < \gamma} x_i = s$ must be satisfied. Since $s \notin D(x_i)$ 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 $s \notin D(x_i)$ 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, $s \prec_{\vec{x}} t$ is satisfied. In the above example, we have $\beta > \gamma$ ($3 > 2$). Therefore, we can bound x_α (x_1) to 1, as shown in Figure 4(b), and 1 must precede 2 in $\vec{x}$ afterwards.

The propagation algorithm **IntValuePrecede**, shown in Figure 5, is based on the two pruning rules just described. The variables *alpha*, *beta*, and *gamma* ensure correct positions for the pointers α, β, and γ respectively. Procedure *initialize*() is called when a value precedence constraint is posted. It finds initial values for the pointers *alpha*, *beta*, and *gamma*. In the procedure, procedure *updateAlpha*() is first invoked to search the position for *alpha*, starting from position 0. During the search, the first pruning rule is applied. We then search for a value for *gamma*. Since value t is removed from $D(x_i)$ for $0 \leq i \leq alpha$, *gamma* must be greater than *alpha* and the position search for *gamma* starts from position $alpha + 1$. Note that the second pruning rule cannot be applied at this point because *beta* is not yet initialized. After fixing *gamma*, procedure *updateBeta*() is invoked to initialize *beta*. By definition, $beta > alpha$. Therefore the search starts from position $alpha + 1$. After fixing *beta*, the second pruning rule can be applied.

Procedure *propagate*(i) in Figure 5 is called whenever the domain of x_i is modified. If $gamma < beta$, then value precedence is already

```
procedure initialize()
alpha := 0;
updateAlpha();
beta := alpha;
gamma := alpha;
if alpha < n then
    repeat gamma := gamma + 1
    until gamma = n ∨ D(x_gamme) = {t};
    updateBeta()
endif

procedure updateAlpha()
while alpha < n ∧ s ∉ D(x_alpha)
    D(x_alpha) := D(x_alpha) \ {t};
    alpha := alpha + 1
endwhile
if alpha < n then
    D(x_alpha) := D(x_alpha) \ {t}
endif

procedure updateBeta()
repeat beta := beta + 1
until beta = n ∨ s ∈ D(x_beta);
if beta > gamma then
    D(x_alpha) := D(x_alpha) ∩ {s}
endif
```

```
procedure propagate(i)
if beta ≤ gamma then
    if i = alpha ∧ s ∉ D(x_i) then
        alpha := alpha + 1;
        while alpha < beta
            D(x_alpha) := D(x_alpha) \ {t};
            alpha := alpha + 1
        endwhile
        updateAlpha();
        beta := alpha;
        if alpha < n then
            updateBeta()
        endif
    else if i = beta ∧ s ∉ D(x_i) then
        updateBeta()
    endif
endif

procedure checkGamma(i)
if beta < gamma ∧ i < gamma
        ∧ D(x_i) = {t} then
    gamma := i;
    if beta > i then
        D(x_alpha) := D(x_alpha) ∩ {s}
    endif
endif
```

Figure 5. The **IntValuePrecede** Propagation Algorithm

entailed and no more propagation is needed. Otherwise, if $i = alpha$ and $s \notin D(x_i)$, then we have to update *alpha* and *beta*. The search for new position for *alpha* starts from position *beta*, because x_{beta} is the original second earliest variable that contains s in its domain. Once s is removed from $D(x_{alpha})$, *beta* becomes the first potential value for *alpha*. 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 *beta* is updated after finding a new value for *alpha*. The search for new value for *beta* starts from position $alpha + 1$. The procedure *updateBeta*() is called to update *beta*. In the procedure, once *beta* 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 *beta* has to be updated. Hence, the procedure *updateBeta*() is called to find a new value for *beta* and to apply the second pruning rule.

Procedure *checkGamma*(i) in Figure 5 serves to update *gamma*. It is called whenever x_i is bound. If $i < gamma$ and x_i is bound to t, then *gamma* is updated to i, and the second pruning rule is applied to check whether $beta > gamma$. The **IntValuePrecede** algorithm enforces GAC on $s \prec_{\vec{x}} t$.

THEOREM 13. *Given an integer variable sequence $\vec{x}$ and integers s and t, the* **IntValuePrecede** *algorithm triggers failure if $s \prec_{\vec{x}} t$ is unsatisfiable; otherwise, the algorithm prunes values from domains of variables in $\vec{x}$ such that GAC on $s \prec_{\vec{x}} t$ is enforced and solutions of $s \prec_{\vec{x}} t$ are preserved.*

Proof. The proof makes use of the definitions of the pointers α, β, and γ. The pruning rules 1 and 2 implemented in **IntValuePrecede** ensure that all values removed from the variable domains must not lead to solutions of $s \prec_{\vec{x}} t$. Therefore, **IntValuePrecede** preserves the solutions of $s \prec_{\vec{x}} t$. To show that **IntValuePrecede** enforces GAC on $s \prec_{\vec{x}} t$, consider the two cases of $\gamma < \beta$ and $\beta < \gamma$. Pruning rule 2 implemented in **IntValuePrecede** has already ensured the satisfiability of $s \prec_{\vec{x}} t$ for the former case (by enforcing $x_\alpha = s$). For the latter case, suppose there is an assignment that cannot be extended to a solution of $s \prec_{\vec{x}} t$ but is not removed by **IntValuePrecede**. Since any extension with $x_\alpha \mapsto s$ must be a solution of $s \prec_{\vec{x}} t$, and $x_\alpha \mapsto t$ is removed by pruning rule 1, the inconsistent assignment that cannot be removed by **IntValuePrecede** must be $x_\alpha \mapsto u$, where $u \in D(x_\alpha)\backslash\{s,t\}$. We show, however, that $x_\alpha \mapsto u$ can always be extended to a solution of $s \prec_{\vec{x}} t$. Since $\beta < \gamma$, variables $x_0, \ldots, x_\beta$ are not yet bound to t. Also, by the definitions of α and β, $s \notin D(x_i)$ for $0 \leq i < \beta$ and $i \neq \alpha$. Therefore $D(x_i)\backslash\{s,t\}$ must be non-empty for $0 \leq i < \beta$ and $i \neq \alpha$. Hence, we can extend $x_\alpha \mapsto u$ by assigning any value other than $\{s,t\}$ to $x_0, \ldots, x_{\alpha-1}$ and $x_{\alpha+1}, \ldots, x_{\beta-1}$, s to x_β, and any value to $x_{\beta+1}, \ldots, x_{n-1}$. This extension must be a solution to $s \prec_{\vec{x}} t$. Thus, $x_\alpha \mapsto u$ is consistent and **IntValuePrecede** maintains GAC on $s \prec_{\vec{x}} t$. □

4.2.2. *Set Version*

In **SetValuePrecede**, the meanings of the pointers α, β, and γ are similar to those in the integer version. Pointer α is the smallest index of $\vec{x}$ 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 $\alpha = n$. Pointer β is the second smallest index of $\vec{x}$ 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 $\beta = n$. Pointer γ is the smallest index of $\vec{x}$ 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 $\gamma = n$. As in the integer version, α and β must be updated increasingly, while γ must be updated decreasingly. Let $\vec{x} = \langle x_0, x_1, x_2, x_3 \rangle$, $s = 1$, and $t = 2$. Suppose we have:

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

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

Pointer α tells that $s \notin PS(x_i) \vee t \in RS(x_i)$ for $0 \leq i < \alpha$, which entails $s \notin x_i \vee t \in x_i$. 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, $s \in x_\alpha \vee t \notin x_\alpha$ 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 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 Figure 6(a).

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

(a)

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

(b)

Figure 6. Illustrating the Use of the Pointers α, β, and γ in **SetValuePrecede**

Pointer γ tells that $s \notin PS(x_\gamma) \wedge t \in RS(x_\gamma)$, which entails $s \notin x_\gamma \wedge t \in x_\gamma$. According to the if-then value precedence constraints, $\bigvee_{0 \leq i < \gamma}(s \in x_i \wedge t \notin x_i)$ must be satisfied. By 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 $s \notin x_i \vee t \in x_i$ must be satisfied 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 $s \prec_{\vec{x}} t$ is satisfied once x_α is proved to contain s but not t. In the above example, $3 = \gamma < \beta = 4$. Therefore, 1 can be added to $RS(x_\alpha)$ and 2 can be removed from $PS(x_\alpha)$, as shown in Figure 6(b).

The **SetValuePrecede** algorithm in Figure 7, based on two pruning rules, contains five procedures with the same names as and similar structures to **IntValuePrecede**. Procedure *initialize*(), called when $ValuePrecede(\vec{x}, s, t)$ is posted, initializes *alpha*, *beta*, and *gamma*. Procedure *propagate*(i) is called whenever $D(x_i)$ is modified, i.e., either $PS(x_i)$ or $RS(x_i)$ is modified. If $gamma < beta$, value precedence is already entailed and no more propagation is needed. Otherwise, there are two different cases. First, $i = alpha \wedge (s \notin PS(x_i) \vee t \in RS(x_i))$, *alpha* and *beta* have to be updated. Second, $i = beta \wedge (s \notin PS(x_i) \vee t \in RS(x_i))$, only *beta* has to be updated. Afterward *checkGamma*(i) is called to check if *gamma* needs update. This differs from the integer version, where *checkGamma*(i) is called only when x_i is bound, since, in the set version, *gamma* may need update even when x_i is not bound. The **SetValuePrecede** algorithm enforces SBC on $s \prec_{\vec{x}} t$.

THEOREM 14. *Given a set variable sequence $\vec{x}$ and integers s and t, the* **SetValuePrecede** *algorithm triggers failure if $s \prec_{\vec{x}} t$ is unsatisfiable; otherwise, the algorithm prunes values from domains of variables in $\vec{x}$ such that SBC on $s \prec_{\vec{x}} t$ is enforced and solutions of $s \prec_{\vec{x}} t$ are preserved.*

Proof. The proof makes use of the definitions of the pointers α, β, and γ. Pruning rules 1 and 2 implemented in **SetValuePrecede** ensure that all values removed from the possible sets of variables must not lead to solutions of $s \prec_{\vec{x}} t$, and all values added to the required sets of variables must occur in every solution of $s \prec_{\vec{x}} t$. Therefore, **SetValuePrecede** preserves the solutions of $s \prec_{\vec{x}} t$. To show that **SetValuePrecede** enforces SBC on $s \prec_{\vec{x}} t$, consider the two cases of $\gamma < \beta$ and $\beta < \gamma$. Pruning 2 implemented in **SetValuePrecede** has already ensured the satisfiability of $s \prec_{\vec{x}} t$ for the former case (by enforcing $s \in x_\alpha \wedge t \notin x_\alpha$). For the latter case, suppose there is a variable x_i such that either $x_i \mapsto PS(x_i)$ or $x_i \mapsto RS(x_i)$ cannot be extended to a solution of $s \prec_{\vec{x}} t$. That is, there is a value a in $PS(x_i) \setminus RS(x_i)$ such that either $s \prec_{\vec{x}} t \wedge a \in x_i$ or $s \prec_{\vec{x}} t \wedge a \notin x_i$ is unsatisfiable, but a is neither removed from $PS(x_i)$ nor added to $RS(x_i)$ by **SetValuePrecede**. Since any extension of $x_\alpha \mapsto u_\alpha$ with $s \in u_\alpha \wedge t \notin u_\alpha$ must be a solution of $s \prec_{\vec{x}} t$, in order to fail either $s \prec_{\vec{x}} t \wedge a \in x_i$ or $s \prec_{\vec{x}} t \wedge a \notin x_i$, we must have $s \notin u_\alpha \vee t \in u_\alpha$. We show, however, that it is always possible to construct solutions of $s \prec_{\vec{x}} t$ with $s \notin u_\alpha \vee t \in u_\alpha$. By the definition of γ and the assumption $\beta < \gamma$, $s \in PS(x_i) \vee t \notin RS(x_i)$ for $0 \leq i < \beta$. Also, by the definitions

```
procedure initialize()
alpha := 0;
updateAlpha();
beta := alpha;
gamma := alpha;
if alpha < n then
   repeat gamma := gamma + 1
   until gamma = n
         ∨ (s ∉ PS(x_gamma) ∧ t ∈ RS(x_gamma));
   updateBeta()
endif

procedure updateAlpha()
while alpha < n
         ∧ (s ∉ PS(x_alpha) ∨ t ∈ RS(x_alpha))
   if s ∉ PS(x_alpha) then
      PS(x_alpha) := PS(x_alpha) \ {t}
   else
      RS(x_alpha) := RS(x_alpha) ∪ {s}
   endif
   alpha := alpha + 1
endwhile

procedure updateBeta()
repeat beta := beta + 1
until beta = n
         ∨ (s ∈ PS(x_beta) ∧ t ∉ RS(x_beta));
if beta > gamma then
   PS(x_alpha) := PS(x_alpha) \ {t};
   RS(x_alpha) := RS(x_alpha) ∪ {s}
endif

procedure propagate(i)
if beta ≤ gamma then
   if i = alpha
         ∧ (s ∉ PS(x_i) ∨ t ∈ RS(x_i)) then
      repeat
         if s ∉ PS(x_alpha) then
            PS(x_alpha) := PS(x_alpha) \ {t}
         else
            RS(x_alpha) := RS(x_alpha) ∪ {s}
         endif
         alpha := alpha + 1
      until alpha ≥ beta;
      updateAlpha();
      beta := alpha;
      if alpha < n then
         updateBeta()
      endif
   else if i = beta
         ∧ (s ∉ PS(x_i) ∨ t ∈ RS(x_i)) then
      updateBeta()
   endif
   checkGamma(i)
endif

procedure checkGamma(i)
if beta < gamma ∧ i < gamma
      ∧ s ∉ PS(x_i) ∧ t ∈ RS(x_i) then
   gamma := i;
   if beta > i then
      PS(x_alpha) := PS(x_alpha) \ {t};
      RS(x_alpha) := RS(x_alpha) ∪ {s}
   endif
endif
```

Figure 7. The **SetValuePrecede** Propagation Algorithm

of α and β, $s \notin PS(x_i) \vee t \in RS(x_i)$ for $0 \leq i < \beta$ and $i \neq \alpha$. Hence, it is always possible to have valid assignments $x_i \mapsto u_i$ with $s, t \in u_i \vee s, t \notin u_i$ for $0 \leq i < \beta$ and $i \neq \alpha$. The assignment $x_\beta \mapsto u_\beta$ with $s \in u_\beta \wedge t \notin x_\beta$ is also valid since $s \in PS(x_\beta) \wedge t \notin RS(x_\beta)$. Now that we have the compound assignment $\langle x_0, \ldots, x_\beta \rangle \mapsto \langle u_0, \ldots, u_\beta \rangle$, where $s, t \in u_i \vee s, t \notin u_i$ for $0 \leq i < \beta$ and $i \neq \alpha$, $s \notin u_\alpha \vee t \in u_\alpha$, and $s \in u_\beta \wedge t \notin x_\beta$. However, note that any extensions of this compound assignment are solutions of $s \prec_{\vec{x}} t$. Therefore, it is impossible to fail either $s \prec_{\vec{x}} t \wedge a \in x_i$ or $s \prec_{\vec{x}} t \wedge a \notin x_i$ for any x_i and $a \in PS(x_i) \setminus RS(x_i)$. Thus, **SetValuePrecede** enforces SBC on $s \prec_{\vec{x}} t$. □

4.3. 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: $v_i \prec_{\vec{u}} v_j$ for $0 \le i < j \le k-1$, where $\vec{u}$ 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: $v_i \prec_{\vec{u}} v_{i+1}$ for $0 \le i \le k-2$. Although achieving the same value precedence effect, the former approach can theoretically achieve more propagation than the latter for both integer and set value precedence. Following Debruyne and Bessiere, we define that enforcing a local consistency LC on some constraints C_1 is *strictly stronger* [19] than enforcing LC on some constraints C_2 if and only if (1) any domain reduction performed by the latter can also be done by the former, and (2) there exists an assignment that can be pruned by the former but not the latter.

THEOREM 15. *Given an integer variable sequence $\vec{u}$ and a set of integer indistinguishable values $V = \{v_0, \ldots, v_{k-1}\}$ under U, enforcing GAC on each of $v_i \prec_{\vec{u}} v_j$ for $0 \le i < j \le k-1$ is strictly stronger than enforcing GAC on each of $v_i \prec_{\vec{u}} v_{i+1}$ for $0 \le i \le k-2$.*

Proof. The former is clearly as strong as the latter. To show strictness, suppose $\vec{x} = \langle x_0, \ldots, x_4 \rangle$ is a sequence of integer variables X and $V = \{0, 1, 2, 3\}$ be a set of indistinguishable values under X. Consider $D(x_0) = \{0\}$, $D(x_1) = \{0, 1\}$, $D(x_2) = D(x_3) = \{0, 2\}$, and $D(x_4) = \{3\}$. Each of the constraints $0 \prec_{\vec{x}} 1$, $1 \prec_{\vec{x}} 2$, and $2 \prec_{\vec{x}} 3$ is GAC. However, the constraint $1 \prec_{\vec{x}} 3$ is *not* GAC, since the assignment $x_1 \mapsto 0$ cannot be extended to a solution of $1 \prec_{\vec{x}} 3$. □

THEOREM 16. *Given a set variable sequence $\vec{u}$ and a set of integer indistinguishable values $V = \{v_0, \ldots, v_{k-1}\}$ under U, enforcing SBC on each of $v_i \prec_{\vec{u}} v_j$ for $0 \le i < j \le k-1$ is strictly stronger than enforcing SBC on each of $v_i \prec_{\vec{u}} v_{i+1}$ for $0 \le i \le k-2$.*

Proof. The former is clearly as strong as the latter. To show strictness, consider the set variable sequence $\vec{y} = \langle y_0, \ldots, y_3 \rangle$ 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 $RS(y_3) = \{2\}$. Suppose $V = \{0, 1, 2\}$ is a set of indistinguishable values under $\{y_0, \ldots, y_3\}$. Each of the constraints $0 \prec_{\vec{y}} 1$ and $1 \prec_{\vec{y}} 2\}$ is SBC. However, the constraint $0 \prec_{\vec{y}} 2$ is *not* SBC, since $y_0 \mapsto RS(y_0)$, i.e., $y_0 \mapsto \emptyset$, cannot be extended to a solution of $0 \prec_{\vec{y}} 2$. □

As we shall see in the experimental results, such difference in propagation level, although theoretically possible, might not show up in practice. Furthermore, it is still an open question whether enforcing

GAC on each of $v_i \prec_{\vec{u}} v_j$ for $1 \leq i < j \leq k$ achieves GAC on the multiple value precedence $v_1 \prec_{\vec{u}} \ldots \prec_{\vec{u}} v_k$ as a whole.

5. Experiments

To demonstrate the feasibility and efficiency of the two proposals, we perform experiments on various problems including graph coloring, the concert hall scheduling problem, the SGP, and the Steiner triple system. All experiments are all-solution search using the smallest-domain-first variable ordering heuristic, and are run using ILOG Solver 4.4 [1] on a Sun Blade 1000 workstation with 2GB memory. For models using the multiple viewpoints method, unless otherwise specified, the extra viewpoint is solely used to express variable symmetry breaking constraints that breaks the value symmetries in the primary viewpoint. Only variables in the primary viewpoint are used as branching variables. We report and compare the number of fails and CPU time for each instance of each model. In the tables, the best number of fails and CPU time among the models for each instance are highlighted in bold. A cell labeled with "-" means execution does not terminate in 2 hours of CPU time.

5.1. Graph Coloring

Given a graph and k colors, graph coloring is to color the vertices of the graph with k colors such that the two vertices connected by each edge have different colors. A CSP model of the problem is to use a variable x_i for each vertex with domain $D_X(x_i) = \{1, \ldots, k\}$ representing the colors. Using this aspect viewpoint V_X, the colors $1, \ldots, k$ are indistinguishable values. We build seven different models for this problem to illustrate the effects of the proposals. The no-break model does not break the symmetries of indistinguishable values, and the remaining models break the symmetries in various ways. The if-then model uses if-then value precedence constraints on adjacent pairs of indistinguishable values. The symmetries of indistinguishable values in V_X becomes variable symmetries in the 0/1 viewpoint V_Z. Each color j becomes a sequence of variables $\langle z_{1,j}, \ldots, z_{n,j} \rangle$ in this viewpoint, where n is the number of vertices in a graph. Flener et al. [21] suggested that the 0/1 viewpoint V_Z can always be used to both model a problem and express symmetry breaking constraints. We therefore build the all-bool model using this technique. It solely uses V_Z to express all the problem constraints as well as lexicographic ordering constraints $\langle z_{1,j}, \ldots, z_{n,j} \rangle \leq_{lex} \langle z_{1,j+1}, \ldots, z_{n,j+1} \rangle$ for $1 \leq j < k$ to break the symmetries. The int-bool model contains the problem constraints in V_X, variable symmetry breaking constraints in V_Z which breaks the value symmetries in V_X, and channeling constraints connecting V_X and V_Z. The glb-adj and glb-all models use adjacent-pair and all-pair postings of the value precedence global constraints respectively, which are $i \prec_{\vec{x}} i+1$ for $1 \leq i < k$ and $i \prec_{\vec{x}} j$ for $1 \leq i < j < k$ respectively, where $\vec{x}$ is a sequence of variables in X. The intbool+glbadj model is the int-bool model plus the adjacent-pair postings of value precedence global constraints. This model shows the combined use of both methods together. The vertices of a graph are re-ordered with decreasing degree. As a result, for models using variables in V_X as branching variables, ties of the smallest-domain-first variable ordering heuristic are broken by choosing a more constrained variable [12]. Another consequence is that in all-bool, which does not contain V_X, variables with a smaller vertex number is branched earlier in the search.

Table I shows the experimental results of solving various instances in the Second DIMACS Challenge[2] using the minimal number of colors (k^*). In the results, models that break the value symmetries (except all-bool) are generally more efficient than no-break. The if-then model is sometimes less efficient than no-break, despite the latter's larger search space. The all-bool and no-break models are incomparable. One is sometimes more efficient than the other and vice versa. Models using global constraints are the fastest among all, confirming the efficiency of our integer value precedence propagation algorithm. The glb-all model shows no extra pruning over glb-adj, and is thus slightly less efficient due to the overhead in maintaining additional constraints. The if-then model also has the same propagation as glb-adj and glb-all, but execution is much slower because of the inefficient propagation of if-then constraints. The int-bool model is slower than glb-adj and glb-all, but more robust than no-break and if-then, which cannot solve myciel4.col and le450_5b.col respectively. The intbool+glbadj model has the same propagation as glb-adj. The running time of the former is worse than that of the latter, due to the extra viewpoint used in the model.

Note that ILOG Solver does not enforce GAC on constraints of the form $c_1 \rightarrow c_2$ in general, where c_1 and c_2 are some constraints. Their propagation behavior is that (1) propagation of c_2 is triggered only after c_1 is entailed, and (2) propagation of c_1 is triggered only after $\neg c_2$ is entailed. In our experiments, however, we *empirically* show that GAC *is* enforced on the integer if-then value precedence constraints. as seen from the same number of fails of if-then and glb-adj. This is probably due to the specific form of the if-then value precedence constraints,

[2] Available at `http://mat.gsia.cmu.edu/COLOR/instances.html`.

Table I. Experimental Results for Graph Coloring

instance	k^*	no-break		if-then		all-bool		int-bool		glb-adj		glb-all		intbool+glbadj	
		fails	time	fails	time	fails	time	fails	time	fails	time	fails	time	fails	time
le450_5a.col	5	976680	273.69	**5008**	6644.34	-	-	20683	23.06	**5008**	**2.35**	**5008**	**3.34**	**5008**	8.3
le450_5b.col	5	291840	94.76	-	-	-	-	**18915**	23.73	21172	**8.81**	21172	12.64	21172	29.36
le450_5c.col	5	7440	9.09	223	512.76	-	-	**169**	0.44	223	**0.28**	223	0.36	223	0.72
le450_5d.col	5	5160	8.76	344	661.68	-	-	**289**	0.93	344	**0.37**	344	0.47	344	0.97
queen5_5.col	5	**0**	0.04	**0**	0.01	144	0.06	4	0.01	**0**	**0**	**0**	**0**	**0**	0.01
queen6_6.col	7	17902080	1382.97	**3630**	13.25	411843	133.78	3669	0.85	**3630**	**0.37**	**3630**	0.58	**3630**	0.95
queen7_7.col	7	7242480	850.31	1613	12.64	1647581	1089.32	**1589**	0.58	1613	**0.24**	1613	0.38	1613	0.65
myciel3.col	4	**0**	0.11	**0**	0.03	23	0.01	16	0.01	**0**	**0**	**0**	0.01	**0**	0.01
myciel4.col	5	-	-	**22**	5674.84	4436	453.3	58692	568.74	**22**	**242.84**	**22**	301.54	**22**	629.25
seatplan[3]	5	4200	86.94	**35**	6.37	16361	1.8	39	1.72	**35**	**0.84**	**35**	0.99	**35**	1.91

[3] The instance is *not* from DIMACS. It is a graph coloring variant which has an extra counting constraint to restrict the occurrences of the colors. See `http://home.chello.no/~dudley/#section7`.

i	s_i	e_i	w_i
1	1	2	10
2	2	3	10
3	3	3	10
4	1	3	10

time	1	2	3
room 1	app 1	app 1	app 3
room 2		app 2	app 2

Total income: 30

Figure 8. An Instance of the Concert Hall Scheduling Problem with 2 Rooms and 4 Applications (Left) and an Optimal Solution (Right)

in which the left hand side is a unary assignment constraint, and the right hand side is a disjunction of assignment constraints. These make the left hand side relatively easy be proved true and the right hand side relatively easy be proved false, thus propagation of the if-then constraint can be triggered earlier.

5.2. Concert Hall Scheduling

A hall director receives n applications to use the k identical rooms of a concert hall. Each application i specifies a period $[s_i, e_i]$ and an offered price w_i to use a room for the whole period. The concert hall scheduling problem is to decide which applications to accept in order to maximize the total income. Each accepted application should be assigned the same room during its whole applied period. Figure 8 shows an example with 2 rooms and 4 applications and an optimal schedule. This problem is generalized from one in the Asia Regional Contest of ACM/ICPC 2003, in which $k = 2$,[4] but is a special case of the temporal knapsack problem [8], in which each application can request more than one room.

[4] See `http://www.u-aizu.ac.jp/conference/ACM/problems/all.pdf`.

We use a variable x_i for each application i with domain $D_X(x_i) = \{1, \ldots, k+1\}$ to denote the room assigned to the application. The dummy value $k+1$ is to denote an rejected application. Under this viewpoint, any two identical applications i and j (i.e., $s_i = s_j$, $e_i = e_j$, and $w_i = w_j$) are symmetric and we can use the variable symmetry breaking constraint $x_i \leq x_j$ where $i < j$ to break the symmetry. Moreover, all domain values except the dummy value $k+1$ are indistinguishable values. This is our only benchmark problem in which not all the domain values are indistinguishable. We build the no-break, if-then, all-bool, int-bool, glb-adj, glb-all, and intbool+glbadj models, which have the same meaning as in graph coloring. Several random instances are generated with 40 applications, $1 \leq s_i \leq e_i \leq 100$, and $10 \leq \frac{w_i}{e_i - s_i + 1} \leq 100$, where s_i, e_i, and w_i are uniformly distributed in their ranges, and are solved as an *optimization* problem for the maximum total income with different number of rooms k.

Figure II shows the experimental results. Like graph coloring, models tackling the indistinguishable values (i.e., if-then, int-bool, glb-adj, glb-all, and intbool+glbadj) are generally more efficient than no-break, which is again generally more efficient than all-bool. The glb-adj and glb-all models have the same number of fails; execution time between them is negligible. Unlike graph coloring, int-bool is more efficient than glb-adj and glb-all in most instances, but they are all much more efficient than if-then. Actually, the different propagation behaviors between int-bool and glb-adj (as well as glb-all) make the smallest-domain-first variable ordering heuristic choose different variables to be assigned next during search. It seems the heuristic fits more to int-bool than to glb-adj and glb-all so int-bool has a smaller overall search tree. The intbool+glbadj model has the same propagation as glb-adj but slower execution. We also tried solving an instance obtained from ACM/ICPC 2003 containing 1000 applications within 365 days. The glb-adj, glb-all, and int-bool models can be solved in about 3.2 hours CPU time, while the other models do not terminate after 6 hours execution.

Bartlett et al. [8] suggested a model, which uses one 0/1 variable for each application to denote whether the application is accepted, for the temporal knapsack problem. Their model, which does not involve particular halls, is more efficient than those used in our experiments. The main aim of our experiments, however, is to evaluate symmetry breaking methods applied to CSPs with value symmetries. The quality of models used in this problem is immaterial to us.

Table II. Experimental Results for the Concert Hall Scheduling Problem

instance	k	no-break fail	no-break time	if-then fail	if-then time	all-bool fail	all-bool time	int-bool fail	int-bool time	glb-adj fail	glb-adj time	glb-all fail	glb-all time	intbool+glbadj fail	intbool+glbadj time
inst1	2	7770	2.33	**3928**	2.58	8615	1.49	**3928**	1.32	**3928**	**1.19**	**3928**	**1.19**	**3928**	1.37
inst1	3	187089	59.34	58624	56.63	266788	48.81	**44040**	**16.86**	58624	20.42	58624	20.46	58624	24.63
inst1	4	3448725	1100.68	522716	673.77	4246755	762.93	**374781**	**155.89**	522716	186.89	522716	193.13	522716	235.93
inst1	5	-	-	4239414	6605.37	-	-	**2636415**	**1152.41**	4239414	1545.08	4239414	1600.23	4239414	2063.9
inst2	2	4273	1.12	2176	1.37	34206	4.28	**2175**	0.65	2176	**0.57**	2176	**0.57**	2176	0.69
inst2	3	39827	12.78	26888	26.11	2276645	310.12	**10398**	**4.06**	26888	9.01	26888	9.11	26888	11.22
inst2	4	416916	139.22	143492	192.83	48567736	6787.48	**62033**	**26.3**	143492	49.55	143492	51.53	143492	64.95
inst2	5	3988530	1368.41	1085772	1891.34	-	-	**304979**	**134.9**	1085772	371.37	1085772	391.02	1085772	515.35
inst3	2	5455	2.17	**2752**	1.93	11267	1.91	2753	1.17	**2752**	**1.1**	**2752**	**1.1**	**2752**	1.26
inst3	3	154517	57.78	66633	64.43	447431	80.7	**33164**	**14.92**	66633	26.02	66633	26.36	66633	32.09
inst3	4	3292195	1227.8	786525	998.99	8944941	1588.94	**301427**	**136.67**	786525	317.28	786525	322.09	786525	399.08
inst3	5	-	-	-	-	-	-	**2223074**	**1049.41**	6880554	2833.06	6880554	2950.12	6880554	3685.6
inst4	2	161575	39.34	**83163**	47.84	297649	33.67	83164	23.27	**83163**	**20.38**	**83163**	**20.38**	**83163**	24.78
inst4	3	10851983	2528.77	3027166	2802.17	31956512	3914.99	**2069561**	**605.59**	3027166	738.45	3027166	731.08	3027166	953.92
inst5	2	6652	1.91	**3336**	2.22	11757	1.56	**3336**	1.1	**3336**	**0.99**	**3336**	**0.99**	**3336**	1.12
inst5	3	205487	57	49722	46.91	630171	82.47	**47312**	16.48	49722	**15.45**	49722	15.85	49722	18.46
inst5	4	4828319	1322.54	485948	585.31	17168657	2157.77	**415504**	150.39	485948	**147.91**	485948	153.68	485948	190
inst5	5	-	-	3983836	5771.1	-	-	**2190385**	**841.43**	3983836	1215.36	3983836	1290.53	3983836	1638.05

5.3. Social Golfer Problem

For the SGP, we build integer and set models in V_G and V_P respectively as two bases and tackle the indistinguishable values in V_G (symmetry (3)) and V_P (symmetry (1)) using different methods. We also test both the integer and set versions of the value precedence propagation algorithms in this benchmark. Note that the two sets of experiments should not be compared directly because (1) models in the two sets have different problem constraints, and (2) different search variables are used (integer and set variables).

5.3.1. *Integer Model*

In the integer model, symmetries (1) and (2) are variable symmetries, which can be broken by row and column lexicographic ordering constraints. Note that the row ordering constraints $\langle g_{i,1}, \ldots, g_{i,\mathcal{W}} \rangle \leq_{lex} \langle g_{i+1,1}, \ldots, g_{i+1,\mathcal{W}} \rangle$ can be simplified to $\langle g_{i,1}, g_{i,2} \rangle <_{lex} \langle g_{i+1,1}, g_{i+1,2} \rangle$, since two golfers can meet each other at most once. Similarly, the column ordering constraints $\langle g_{1,k}, \ldots, g_{\mathcal{N},k} \rangle \leq_{lex} \langle g_{1,k+1}, \ldots, g_{\mathcal{N},k+1} \rangle$ can also be simplified to $\langle g_{1,k}, \ldots, g_{\mathcal{G}+1,k} \rangle <_{lex} \langle g_{1,k+1}, \ldots, g_{\mathcal{G}+1,k+1} \rangle$. These problem-specific simplified constraints allows more propagation than the original ones, and therefore are used in our experiments.

Using this basis, we build the int-set and int-bool models which use multiple viewpoints and break the symmetries of indistinguishable values in V_G as variable symmetries in V_P and V_Z respectively. Note that in int-set, we add extra implied constraints $|p_{j,k}| = \mathcal{S}$ for $1 \leq j \leq \mathcal{G}$ and $1 \leq k \leq \mathcal{W}$, since they can increase propagation on the symmetry breaking constraints in V_P. We also build glb-adj and glb-all that

Table III. Experimental Results for the SGP, using Integer Variables

$\mathcal{G},\mathcal{S},\mathcal{W}$	if-then		all-bool		int-bool		int-set		glb-adj		glb-all		intbool+glbadj		intset+glbadj	
	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time
4, 2, 5	**263**	0.23	2998	0.59	307	0.23	**263**	0.29	**263**	**0.15**	**263**	0.17	**263**	0.26	**263**	0.32
4, 2, 6	**279**	0.18	2598	0.51	333	0.18	**279**	0.24	**279**	**0.13**	**279**	**0.13**	**279**	0.2	**279**	0.26
4, 3, 4	**917**	0.36	64027	12.43	1339	0.37	**917**	0.37	**917**	**0.2**	**917**	0.21	**917**	0.28	**917**	0.42
5, 2, 3	**675**	1.39	18599	2.94	1327	1.12	**675**	1.39	**675**	**0.77**	**675**	0.82	**675**	1.22	**675**	1.56
5, 2, 4	**36804**	74.03	527962	122.44	52543	57.9	**36804**	74.3	**36804**	**40.33**	**36804**	43.37	**36804**	63.36	**36804**	82.29
5, 2, 5	**758610**	1400.16	9473196	2505.01	867791	1075.95	**758610**	1458.34	**758610**	**767.03**	**758610**	815.99	**758610**	1176.87	**758610**	1591.2
5, 2, 6	-	-	-	-	6605552	6839.61	-	-	**6083358**	**4835.11**	**6083358**	5245.75	-	-	-	-
5, 2, 8	-	-	-	-	-	-	-	-	**19960565**	**6274.41**	**19960565**	6852.93	-	-	-	-
5, 2, 9	**8325932**	4326.68	14221361	5439.92	9166800	3210.56	**8325932**	4073.58	**8325932**	**2076.84**	**8325932**	2288.03	**8325932**	3351.6	**8325932**	4319.38
5, 3, 3	**207217**	368.98	15624244	3450.06	213328	192.31	**207217**	269.96	**207217**	**149.29**	**207217**	156.01	**207217**	202.64	**207217**	280.75
5, 3, 7	-	-	-	-	**10019241**	3821.9	10954130	6320.45	10954130	**3117.3**	10954130	3329.13	10954130	4460.46	10954130	6543.4
5, 4, 3	**126170**	183.79	-	-	382664	183.63	**126170**	120.58	**126170**	**66.95**	**126170**	69.56	**126170**	85.9	**126170**	123.37
5, 4, 4	-	-	-	-	-	-	-	-	**13867877**	**4262.97**	**13867877**	4437.77	**13867877**	5466.61	-	-
5, 4, 5	-	-	-	-	-	-	-	-	**14849892**	**4996.87**	**14849892**	5228.43	**14849892**	6501.96	-	-
5, 5, 3	**42**	1.94	-	-	21038	13.32	**42**	1.22	**42**	**0.74**	**42**	0.76	**42**	0.9	**42**	1.27
5, 5, 4	**9031**	15.91	-	-	190084	93.7	**9031**	8.6	**9031**	**5.01**	**9031**	5.21	**9031**	6.29	**9031**	8.83
5, 5, 5	**1933**	5.01	-	-	27746	14.26	**1933**	2.58	**1933**	**1.44**	**1933**	1.5	**1933**	1.84	**1933**	2.63
5, 5, 6	**237**	0.88	-	-	1776	1.26	**237**	0.45	**237**	**0.26**	**237**	**0.26**	**237**	0.3	**237**	0.46
6, 2, 3	**39059**	140.87	1329681	257.18	110529	95.63	**39059**	119.85	**39059**	**65.41**	**39059**	71.03	**39059**	93.94	**39059**	124.48
6, 3, 2	**10**	2.83	713811	233.86	202	1.17	**10**	1.69	**10**	**0.89**	**10**	0.96	**10**	1.2	**10**	1.74
6, 4, 2	**377**	3.69	-	-	1981	1.88	**377**	1.8	**377**	**0.98**	**377**	1.04	**377**	1.24	**377**	1.86
6, 5, 2	**1226**	3.99	-	-	2240	1.44	**1226**	1.57	**1226**	**0.8**	**1226**	0.86	**1226**	1.01	**1226**	1.6
6, 6, 3	**20917**	3300.59	-	-	-	-	**20917**	1528.85	**20917**	**953.23**	**20917**	981.05	**20917**	1108.77	**20917**	1558.92
7, 2, 2	**6**	1.07	59576	11	844	0.63	**6**	0.71	**6**	**0.38**	**6**	0.43	**6**	0.55	**6**	0.75
7, 3, 2	**180**	189.85	-	-	7504	63.26	**180**	91.5	**180**	**48.29**	**180**	52.73	**180**	64.62	**180**	94.96
7, 4, 2	**60747**	1234.79	-	-	66985	332.42	**60747**	506.17	**60747**	**269.54**	**60747**	293.39	**60747**	342.27	**60747**	519.28
7, 5, 2	**46007**	365.82	-	-	131666	145.86	**46007**	123.71	**46007**	**69.03**	**46007**	74.05	**46007**	83.49	**46007**	126.07
7, 6, 2	**16447**	128.48	-	-	29485	31.18	**16447**	36.46	**16447**	**18.18**	**16447**	19.15	**16447**	21.22	**16447**	37.09
7, 7, 2	**66**	1.79	-	-	348	0.58	**66**	0.26	**66**	**0.14**	**66**	**0.14**	**66**	**0.14**	**66**	0.26

breaks the symmetries using global constraints. The all-bool model is the same as the one used by Frisch et al. [26] except that we apply the same simplification technique as above to simplify the row and column lexicographic ordering constraints. Since there are two models int-bool and int-set using the multiple viewpoints method, we correspondingly build two models intbool+glbadj and intset+glbadj using the combined method. Table III shows the experimental results.

Again, glb-adj and glb-all are the fastest among all. The performance of int-set and int-bool approaches that of the global constraints models. The glb-adj, glb-all, int-set, and if-then models has the same number of fails. The int-bool model achieves less propagation than them. Nevertheless, its performance is still generally much better than if-then and all-bool. The int-set and int-bool models are incomparable. The former is sometimes slightly slower than the latter, but in certain instances (e.g, $(5, 5, 3)$, $(5, 5, 4)$, $(5, 5, 5)$, and $(6, 6, 3)$), the difference in number of fails between them is so large that int-set shows its robustness and is significantly faster. Both intbool+glbadj and intset+glbadj again have the same propagation as glb-adj, but they are slower in execution.

5.3.2. *Set Model*

In the set model, symmetries (3) and (2) are variable symmetries. Barnier and Brisset [7] suggested the constraints $\min(p_{j,k}) < \min(p_{j+1,k})$ for $1 \leq j < \mathcal{G}$ and $1 \leq k \leq \mathcal{W}$ and $\min(p_{1,k} \setminus \{1\}) < \min(p_{1,k+1} \setminus \{1\})$

for $1 \leq k < \mathcal{W}$ for breaking the symmetries respectively. These constraints are degenerated from the lexicographic ordering constraints with the set ordering we propose and the problem constraints that two variables $p_{j,k}$ and $p_{j',k}$ for some $j \neq j'$ must be disjoint. Using this basis, we again build the glb-adj and glb-all models, using our set value precedence constraints, as well as the set-int and set-bool models that breaks the value symmetries in V_G as variable symmetries in V_P and V_Z respectively. Note that the variable symmetry breaking constraints in V_P of set-int are actually the same as the row ordering constraints in the integer model described in previous subsection. Therefore, the simplified constraints $\langle g_{i,1}, g_{i,2} \rangle <_{lex} \langle g_{i+1,1}, g_{i+1,2} \rangle$ can be used instead. Furthermore, since only variables of weeks 1 and 2 in V_G are used for expressing constraints, the remaining variables in V_G (and the channeling constraints relating them) are removed from the model so that only part of V_G is connected with V_P and less overhead of channeling is incurred. Similarly, in set-bool, only variables of weeks 1 and 2 in V_Z are connected with V_P. Note that when $\mathcal{W} = 2$, there will be no savings in the number of variables. This modeling trick is not applicable when using global constraints, and is actually an advantage of using multiple viewpoints over global constraints to break value symmetries.

The experimental results in Table IV shows that set-int is the most efficient in terms of both the number of fails and CPU time. It always has the smallest number of fails due to the extra propagation of the simplified symmetry breaking constraints. The removal of the unused variables in the second viewpoint also reduces the overhead of channeling. The set-bool model achieves the same amount of propagation as glb-adj. Their runtime are similar in many instances. In larger instances (down in the table), glb-adj is more competitive than set-bool due to the extra variables in the latter model. The models using global constraints still perform much better than if-then and all-bool, confirming the efficiency of our set propagation algorithm. Note that all-bool is slightly different from the one in the previous subsection, because a different aspect priority (and hence scanning sequence) is used to generate the variable symmetry breaking constraints in V_Z. The scanning sequence $\langle golfers, weeks, groups \rangle$ is used in all-bool previously, while $\langle weeks, groups, golfers \rangle$ is used here. The glb-all model achieves more propagation than glb-adj does in some instances. The difference in the number of fails, however, is usually small, so the overhead of extra global constraints cannot be compensated. An exception is the $(4, 4, 4)$ instance where glb-all has significantly fewer fails than glb-adj, and all-bool is the best among all models. Actually, glb-all posts $O(\mathcal{N}^2) = O(\mathcal{G}^2\mathcal{S}^2)$ more constraints than glb-adj. Hence, instances with more golfers incur more overhead than those with fewer golfers. The

Table IV. Experimental Results for the SGP, using Set Variables

$\mathcal{G},\mathcal{S},\mathcal{W}$	if-then		all-bool		set-bool		set-int		glb-adj		glb-all		setbool+glbadj		setint+glbadj	
	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time
4, 2, 4	318	0.86	3935	0.4	318	0.2	**266**	**0.19**	318	0.2	314	0.2	318	0.22	**266**	0.21
4, 2, 5	921	3.08	18391	1.81	921	0.53	**884**	**0.52**	921	0.54	919	0.58	921	0.54	**884**	0.55
4, 2, 6	736	3.5	36012	3.45	736	**0.51**	**721**	**0.51**	736	0.52	736	0.55	736	**0.51**	**721**	0.52
4, 2, 7	100	1.12	41516	4.09	100	0.16	**97**	**0.15**	100	**0.15**	100	**0.15**	100	0.16	**97**	0.16
4, 3, 4	5910	11.87	9453	1.26	5910	1.37	**3222**	**0.81**	5910	1.03	5026	1.15	5910	1.53	**3222**	0.93
4, 4, 4	518890	2187.98	**376**	**0.08**	518890	176.96	15759	8.11	518890	123.1	42993	13.33	518890	213.04	15759	9.45
4, 4, 5	15760	98.1	**410**	**0.12**	15760	7.97	7510	5.63	15760	6.05	14900	7.8	15760	9.22	7510	6.41
5, 2, 3	556	6.35	20017	2.84	556	1.08	**521**	**1.07**	556	1.11	556	1.3	556	1.14	**521**	1.13
5, 2, 4	95109	420.14	1896414	258.64	95109	**78.35**	**95063**	78.46	95109	81.55	95109	92.18	95109	82.59	**95063**	84.03
5, 2, 5	-	-	-	-	2555004	1961.6	**2554588**	**1956.71**	2555004	2003.41	2554992	2240.45	2555004	2070.47	**2554588**	2066.8
5, 2, 9	-	-	-	-	3895074	6853.2	**3895064**	6841.59	3895074	**6770.87**	3895074	7046.1	3895074	7125.63	**3895064**	7130.65
5, 3, 3	491937	1094.32	1505051	339.5	491937	**139.99**	**491452**	140.06	491937	150	491937	223.17	491937	156.72	**491452**	156.68
5, 4, 3	694740	1149.42	1570961	377.43	694740	106.16	**658755**	**98.23**	694740	112.66	694740	225.64	694740	123.68	**658755**	113.93
5, 4, 4	-	-	-	-	31481057	4953.67	**30802587**	**4770.94**	31481057	5307.77	-	-	31481057	5540.36	**30802587**	5338.92
5, 5, 3	8229	25.33	9391	2.95	8229	2.24	**1418**	**0.46**	8229	1.37	8229	3.56	8229	2.62	**1418**	0.53
5, 5, 4	155999	902.96	92589	25.51	155999	45.43	**13009**	**2.1**	155999	26.73	155999	63.39	155999	51.89	**13009**	2.36
6, 2, 3	19920	822.96	1667863	271.14	19920	85.96	**19659**	**85.81**	19920	88.93	19920	113.9	19920	89.24	**19659**	88.41
6, 3, 2	8823	12.62	17630	2.73	8823	2.68	**30**	1.16	8823	**1.11**	8823	2.24	8823	2.91	**30**	1.3
6, 4, 2	22407	27.55	27675	4.19	22407	5.17	**45**	**0.99**	22407	2.01	22407	6.02	22407	5.91	**45**	1.11
6, 5, 2	7090	8.16	5592	0.84	7090	1.72	**60**	**0.1**	7090	0.6	7090	2.73	7090	2.03	**60**	0.13
6, 6, 3	-	-	10260789	3992.7	1719940	490.77	**1521747**	**414.16**	1719940	524.14	1719940	2843.57	1719940	583.25	**1521747**	490.32
7, 2, 2	1276	5.27	9599	1.47	1276	0.97	**21**	0.69	1276	**0.44**	1276	0.65	1276	1.03	**21**	0.75
7, 3, 2	314043	864.12	654575	131.16	314043	119.13	**42**	58.94	314043	**47.97**	314043	102.6	314043	130.19	**42**	65.7
7, 4, 2	-	-	4488705	869.09	3318328	928.12	**63**	**236.73**	3318328	344.75	3318328	1183.69	3318328	1053.34	**63**	261.7
7, 5, 2	1887614	3789.96	1835104	348.14	1887614	497.93	**84**	**42.98**	1887614	174.48	1887614	938.25	1887614	585.95	**84**	47.78
7, 6, 2	117713	220.28	81421	14.57	117713	35.48	**105**	**0.53**	117713	10.84	117713	88.77	117713	42.21	**105**	0.61
7, 7, 2	397	5.43	1493	0.8	397	0.45	**308**	0.7	397	**0.15**	397	1.25	397	0.58	**308**	0.86

setbool+glbadj and setint+glbadj models have the same propagation as set-bool and set-int respectively. Therefore, their execution is slightly longer due to the extra value precedence constraints in the models.

Besides using the smallest-domain-first variable ordering heuristic, we also tried using the default and static lexicographic variable ordering heuristic (i.e., always choose the first unbound variable). We find that the number of fails of glb-adj and glb-all are the same in all instances. It seems that the extra propagation of glb-all only occurs after we add values to the required set of a non-first unbound variable. This shows that the theoretical possibility of extra propagation of glb-all over glb-adj does not guarantee a pruning in search space. Indeed, the search states that lead to extra propagation by glb-all must be reachable during search in order for an actual pruning in practice.

5.4. Steiner Triple System

Let $X = \{1, \ldots, v\}$, where $v \geq 3$. A Steiner triple system $S(v)$ of order v is a set of 3-subset (unordered triples) of X such that every 2-subset of X occurs in exactly one triple of $S(v)$. An example of $S(7)$ is $\{\{1,2,3\},\{1,4,5\},\{1,6,7\},\{2,4,6\},\{2,5,7\},\{3,4,7\},\{3,5,6\}\}$. A Steiner triple system of order v exists if and only if $v \equiv 1, 3 \pmod 6$ [33].

Finding Steiner triple systems of order v is a MAP with two aspects: the triples and the set X. The problem can thus be modeled using an aspect viewpoint V_B with a set of set variables $B = \{b_1, \ldots, b_n\}$ (where

Table V. Experimental Results for Steiner Triple Systems

	if-then		all-bool		set-bool		set-set		glb-adj		glb-all		setset+glbadj		setbool+glbadj	
v	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time
7	82	0.02	29	0.01	75	0.01	**12**	**0**	82	**0**	82	0.01	**12**	**0**	75	0.01
9	17236	6.58	10021	1.37	17092	2.43	500	0.09	17236	1.53	17236	1.77	**153**	**0.03**	17092	2.48
13	-	-	-	-	-	-	34004314	15005.2	-	-	-	-	**3935567**	**1738.24**	-	-

$n = \frac{v(v-1)}{6}$) and $PS(b_i) = \{1, \ldots, v\}$. In V_B, the variable symmetries are that any two variables b_i and b_j can be exchanged. They can be broken by the constraints $\min(b_i) \leq \min(b_{i+1})$ and $\min(b_i) = \min(b_{i+1}) \rightarrow \min(b_i \setminus \{\min(b_i)\}) \leq \min(b_{i+1} \setminus \min(b_{i+1})\})$ for $1 \leq i < n$ (which are the degenerated lexicographic variable symmetry breaking constraints). The values in $PS(b_i) = \{1, \ldots, v\}$ are indistinguishable values. Such value symmetries can be broken by using global constraints (glb-adj and glb-all), channeling with the 0/1 viewpoint (set-bool), and channeling with the other aspect viewpoint V_X with set variables $x_1, \ldots, x_v$ and $PS(x_j) = \{1, \ldots, n\}$ (set-set). The symmetry breaking constraints in V_X are $\min(x_j) \leq \min(x_{j+1})$ and $\min(x_j) = \min(x_{j+1}) \rightarrow \min(x_j \setminus \{\min(x_j)\}) \leq \min(x_{j+1} \setminus \{\min(x_{j+1})\})$ for $1 \leq j < n$, which are similar to those in V_B. Since ILOG Solver does not provide a set minus constraint, in the experiments w e emulate the expression $y \setminus \{\min(y)\}$ using y', where $|y'| = |y| - 1$, $\min(y) \notin y'$, and $y' \subset y$.

Experimental results in Table V show that setset+glbadj achieves the best results. It combines the benefits of set-set and glb-adj and achieves much more propagation than either of them. For $v = 13$, it is faster than set-set, the second most efficient model, by an order of magnitude, while executions of the other models does not terminate after 24 hours. This phenomenon is different from what we observe in previous benchmarks, in which models using combined methods have the same propagation as either methods. This is probably due to the special set minus constraints used in V_X; they do not occur in previous benchmarks. This shows that when symmetry breaking constraints in the second viewpoint involves set minus constraints, the two methods need not be used alone. They can be used simultaneously to obtain even more speedup.

6. Related Work

Symmetry breaking is an important line of research in the constraint community. There are several main types of techniques in breaking symmetries in CSPs. In the first approach, *symmetry breaking constraints*

[43] are added to a CSP so as to traverse fewer number of symmetrical regions during the search for solutions. Crawford et al. [18] suggested a general scheme to add symmetry breaking predicates to satisfiability problems. Aloul et al. [3] improved this scheme by constructing more efficient CNF representations of symmetry-breaking predicates. The scheme by Crawford et al. can be extended to break variable symmetries in CSPs by using lexicographic ordering constraints [26, 14, 13]. For each variable symmetry σ in a CSP, we add a lexicographic ordering constraint $\vec{x} \leq_{lex} \sigma(\vec{x})$ to the CSP, where $\vec{x}$ is a sequence of variables in the CSP. Flener et al. [21] identified row and column symmetries in 2-dimensional matrix models, which are commonly found in many CSPs. Row and column symmetries are the variable symmetries that every two rows and every two columns in a matrix of variables can be exchanged. Given a matrix with n rows and m columns, row and column symmetries collectively imply $n!m!$ variable symmetries. Flener et al. [21] showed that adding constraints to lexicographically order both the rows and the columns are always consistent, although they do not necessarily break all the row and column symmetries. Bessiere et al. [11] showed the intractability of breaking row and column symmetries completely. Multiset ordering constraints [25] and allperm constraints [27] are also available for breaking row and column symmetries in matrix models. Gent [28] designed special constraints to break symmetries of indistinguishable values under some integer variables. The constraints assume that all the domain values of the variables are indistinguishable. It is not clear what consistency is enforced on the constraints. Our value precedence constraints is applicable to both integer and set variables, enforcing GAC and SBC respectively. They break symmetries of two indistinguishable values, and can be posted multiple times to break symmetries of multiple indistinguishable values.

The second approach is to break symmetries dynamically during search [4, 5, 31, 20, 23]. Search algorithms for solving CSPs are modified such that symmetric states are pruned from the search tree as it is developed during execution. A representative of this approach is *Symmetry Breaking During Search* (SBDS) and its variants [31, 29]. Upon a backtrack of the search, SBDS [31] adds a symmetry breaking constraint for each symmetry in a CSP to remove all the states which are symmetric to the one that causes the current backtrack. Backofen and Will introduced *Symmetry Excluding Search* (SES) [4, 5], which is similar to but more general than SBDS. SES allows a search tree to branch over arbitrary constraints instead of simple unary assignment constraints in SBDS. CSP symmetries form symmetry groups; Gent et al. [29] incorporated GAP [2], a computational group theory system, to SBDS such that large symmetry groups can be handled efficiently.

Another representative of the second approach is called *Symmetry Breaking via Dominance Detection* (SBDD) and its variants [20, 44, 46, 6, 7]. In SBDD [20], whenever the search algorithm generates a new search node, we check whether it is dominated by another node previously visited through some symmetries. If so, the current search node can be pruned; otherwise it is processed normally. Unlike SBDS, which uses compound assignments to determine what constraints are to be added upon backtracking, SBDD uses the sets of variable domains at each search node to represent a state in the search tree. A problem specific dominance checker is needed to check whether one state is dominated by another previously visited state. Barnier and Brisset [6, 7] proposed SBDD+, an improvement of SBDD. The key idea of the improvement is a deep pruning technique which allows to prune higher in the search tree whenever possible. Gent et al. [30] again used computational group theory to extend SBDD. They also proposed a generic dominance checker, which avoids the need of implementing a specific checker in SBDD for each problem by a constraint programmer.

It is possible to combine this dynamic approach those using symmetry breaking constraints to tackle symmetries. Smith and Gent [48] showed how the use of symmetry breaking constraints and SBDS can be combined to break row and column symmetries in matrix models [21]. They also empirically compared several different approximations to eliminating the symmetries and an exact method that eliminates the symmetries completely for small matrices. Puget [45] showed how to combine the use of lexicographic ordering constraints and SBDD for row and column symmetries. He also presented a method that adds some lexicographic ordering constraints during the search for solution. These constraints break the symmetries that leave the current partial assignment unchanged.

The third approach to tackle symmetries in CSPs is to use the symmetries to guide the search. Meseguer and Torras [39, 40] propose variable ordering heuristics which select the variable leading to a search subspace with the largest number of distinct states. They also propose a symmetric domain value pruning procedure along the search based on nogood recording.

The fourth approach is to construct specialized search trees that does not contain symmetries [49, 47]. Van Hentenryck et al. [49] studied three classes of CSPs for which symmetry breaking is tractable. These CSP classes, featuring specific forms of indistinguishable values,[5] allow symmetry breaking to be performed in constant time and

[5] Van Hentenryck et al. [49] used the term interchangeable values to denote indistinguishable values.

space during search using dedicated search procedures. Roney-Dougal et al. [47] generalized their idea and introduced GE-tree as a conceptual abstraction in symmetry breaking. A GE-tree with symmetry group G is a search tree such that (1) no search node of the tree is isomorphic (symmetrically equivalent) under G to any other node and (2) given a complete assignment ϕ, there is at least one leaf node of the tree which lies in the orbit of ϕ. Constructing and traversing a GE-tree breaks all symmetries in a CSP, although it is difficult in general to do so for arbitrary symmetries. Roney-Dougal et al. showed the tractability for the case of arbitrary value symmetries by giving a polynomial time algorithm to construct GE-trees for the case.

Constructing GE-trees for symmetries of some indistinguishable values $\{v_1, \ldots, v_k\}$ under U can be equivalent to maintaining the value precedence $v_1 \prec_{\vec{u}} \ldots \prec_{\vec{u}} v_k$, where $\vec{u}$ is a sequence of the variables in U. Figure 9 shows a GE-tree for the symmetries of indistinguishable values $\{1, 2, 3, 4\}$ under $U = \{x_1, x_2, x_3, x_4\}$, where $D(x_i) = \{1, 2, 3, 4\}$ for $1 \leq i \leq 4$. In the tree, each level of nodes (except the leaf level) represents a variable to be labeled. Each edge under a node represents a domain value that is chosen in the labeling process. Therefore, a node can be thought of as a compound assignment constructed by traversing from the root of the tree to that node. The leaf nodes of the GE-tree represent the unique solutions under the symmetries. This GE-tree tree can be constructed using a simple rule [49]: at each node whose level corresponds to variable x_i, suppose the node represents a compound assignment ϕ. We construct edges for the domain values of x_i that have occurred in ϕ, collected in a set V_{old}, and exactly one edge for one new value that is *not* in V_{old}, i.e., one value in $D(x_i) \setminus V_{old}$. For example, in Figure 9, consider the node representing the compound assignment $\phi = \{x_1 \mapsto 1, x_2 \mapsto 2\}$, i.e., the rightmost node in level x_3. Values 1 and 2 occur in ϕ. Therefore, $V_{old} = \{1, 2\}$ and we construct edges labeled 1 and 2. Furthermore, we construct eactly one edge whose value is in $D(x_3) \setminus V_{old}$. In this example, we choose the value 3 among values 3 and 4. Hence, we construct three edges labeled 1, 2, and 3 for this node.

The solutions in the GE-tree are collected in Figure 10. It can be seen that the solutions are exactly the same as those of $1 \prec_{\vec{u}} 2 \prec_{\vec{u}} 3 \prec_{\vec{u}} 4$, where $\vec{u} = \langle x_1, x_2, x_3, x_4 \rangle$.

Although GE-trees can be constructed tractably for arbitrary value symmetries, we break symmetries of indistinguishable values $\{v_0, \ldots, v_{k-1}\}$ using $O(k)$ or $O(k^2)$ value precedence constraints, depending on the adjacent-pair or all-pair of postings. Value precedence constraints can also be used to break such symmetries on set variables, which is not defined for GE-trees.

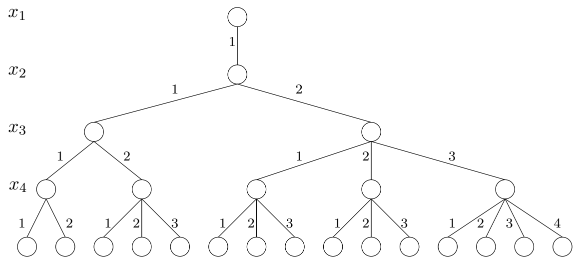

Figure 9. A GE-tree for the Indistinguishable Values $\{1, \ldots, 4\}$ Under $\{x_1, \ldots, x_4\}$

x_1	x_2	x_3	x_4	x_1	x_2	x_3	x_4	x_1	x_2	x_3	x_4	x_1	x_2	x_3	x_4	x_1	x_2	x_3	x_4
1	1	1	1	1	1	2	2	1	2	1	2	1	2	2	2	1	2	3	2
1	1	1	2	1	1	2	3	1	2	1	3	1	2	2	3	1	2	3	3
1	1	2	1	1	2	1	1	1	2	2	1	1	2	3	1	1	2	3	4

Figure 10. Solutions of $1 \prec_{\vec{u}} 2 \prec_{\vec{u}} 3 \prec_{\vec{u}} 4$, where $\vec{u} = \langle x_1, x_2, x_3, x_4 \rangle$

7. Concluding Remarks

We conclude the paper by summarizing our contributions and giving discussions and possible directions for future research.

7.1. Contributions

We have proposed two methods of using symmetry breaking constraints to break value symmetries in CSPs. The contributions of our work can be summarized as follows. First, we have introduced the framework of Multi-aspect Assignment Problems (MAPs), and shown in general how to derive $n+1$ CSP viewpoints for a MAP with n aspects. The viewpoints are called aspect viewpoints and 0/1 viewpoint. Matrix models can then be built using these viewpoints. Many combinatorial problems are instances of MAPs. Hence, our framework allows handy choices of viewpoints for CSP modelers. Second, we have identified the conditions when value symmetries in one aspect viewpoint of a MAP correspond to variable symmetries in another (aspect or 0/1) viewpoint of the same MAP. While value symmetries breaking constraints can be difficult to formulate and express, our work gives possibilities of breaking value symmetries as variable symmetries in another viewpoint with the aid of channeling constraints. Third, we have introduced the notions of aspect priorities and scanning sequences. Using these notions, we have established theorems to identify when symmetry breaking constraints in two viewpoints, connected using channeling constraints, are consistent.

Fourth, we have introduced the notion of value precedence and shown how the notion can be used to design constraints for breaking a common class of value symmetries, namely the symmetries of indistinguishable values. Fifth, we have presented two efficient propagation algorithms for implementing global constraints on integer and set value precedence, enforcing GAC and SBC respectively. The global constraints avoid the use of inefficient if-then constraints. Sixth, we have given theoretical results to characterize several properties of our proposed algorithms in different usage scenarios.

7.2. Discussion

Breaking value symmetries with symmetry breaking constraints is not an easy task. We have proposed two methods to tackle this problem. On one hand, the multiple viewpoints method is purely a modeling technique, involving no invention of specialized propagation algorithms and no alteration to the underlying CSP solver. The method can sometimes tackle arbitrary value symmetries. On the other hand, we design and implement propagation algorithms for developing two global constraints to maintain value precedence. Experimental results show that the two proposed methods are always better than using if-then constraints. Using global constraints is generally more efficient than the multiple viewpoints method, since propagation algorithms are specially designed and no additional variables and channeling constraints are required. The performance of the two methods is reversed only in cases when special modeling tricks can be applied in the additional viewpoint used by the multiple viewpoints method. This, however, does not imply that the multiple viewpoints method is inferior. In fact, the strength of the method lies in exactly the possibility of applying modeling tricks, which is less available to the method of using global constraints in a single viewpoint. Therefore, both methods have their own merits and are valuable to value symmetry breaking. Actually, as we have shown in the Steiner triple system, the two proposed methods do not compete but are *complementary* to each other. They can be used together; the overall benefits are more than those of using either method alone.

7.3. Future Work

Our work proposes a study of using symmetry breaking constraints for value symmetries in CSPs. There is scope for future work. First, Theorem 7, which states the conditions when variable breaking constraints in two aspect viewpoints are consistent, is applicable to only variable symmetries corresponding to symmetries of indistinguishable values in one of the aspect viewpoints. Theorem 6, which states the consistency conditions of variable symmetry breaking constraints between an aspect viewpoint and the 0/1 viewpoint, is applicable to *arbitrary* value symmetries. It would be interesting to generalize Theorem 7 to cover arbitrary value symmetries also.

Second, we have shown in Theorem 15 that GAC on integer value precedence on all pairs of indistinguishable values is strictly stronger than GAC on adjacent pairs of indistinguishable values. However, in our benchmarks using the integer value precedence constraint, models using adjacent-pair and all-pair postings always achieve the same number of fails. It is worthwhile to reason about this phenomenon. Third, the value precedence global constraints can be extended and generalized. For example, a propagation algorithm for maintaining value precedence of multiple values may be designed so that symmetries of multiple indistinguishable values can be broken using only one global constraint. Besides, the antecedent and subsequent in value precedence can be also integer variables instead of simply integer constants in our current implementations, so that the generalized constraints have additional usage besides symmetry breaking. Fourth, our current implementations of the value precedence global constraints enforce GAC and SBC respectively. It would be interesting to design propagation algorithms that enforce other local consistencies, such as bounds consistency for integer value precedence.

Fifth, we have identified and designed constraints to break the symmetries of indistinguishable values. It is possible to do the same for other classes of value symmetries, or even arbitrary symmetries. Since using constraints for breaking symmetries does not involve modifying the underlying CSP solver, such work make symmetry breaking techniques more accessible to CSP modelers. Sixth, in the benchmark of Steiner triple systems, we have demonstrated that the model using both value precedence constraints and the multiple viewpoint approach achieves the best results. A promising future work is to investigate combining different methods of breaking symmetries.

Acknowledgments

We thank the anonymous referees from CP'04, SAC'05, and the *Constraints* journal 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 constraints for our reference. The work described in this paper was substantially supported by grants from the Research Grants Council of the Hong Kong Special Administrative Region (Project no. CUHK4131/05E and CUHK4219/04E).

References

1. *ILOG Solver 4.4 Reference Manual*, 1999.
2. *GAP 4.4.5 Reference Manual*, 2005.
3. F.A. Aloul, K.A. Sakallah, and I.L. Markov. Efficient symmetry breaking for boolean satisfiability. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 271–276, 2003.
4. R. Backofen and S. Will. Excluding symmetries in constraint-based search. In *Proceedings of the 5th International Conference on Principles and Practice of Constraint Programming*, pages 73–87, 1999.
5. R. Backofen and S. Will. Excluding symmetries in constraint-based search. *Constraints*, 7:333–349, 2002.
6. N. Barnier and P. Brisset. Solving the Kirkman's schoolgirl problem in a few seconds. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, pages 477–491, 2002.
7. N. Barnier and P. Brisset. Solving Kirkman's schoolgirl problem in a few seconds. *Constraints*, 10(1):7–21, 2005.
8. M. Bartlett, A.M. Frisch, Y. Hamadi, I. Miguel, S.A. Tarim, and C. Unsworth. The temporal knapsack problem and its solution. In *Proceedings of the 2nd International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 34–48, 2005.
9. N. Beldiceanu, M. Carlsson, and T. Petit. Deriving filtering algorithms from constraint checkers. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming*, pages 107–122, 2004.
10. B. Benhamou. Study of symmetry in constraint satisfaction problems. In *Proceedings of the 2nd Workshop on Principles and Practice of Constraint Programming*, 1994.
11. C. Bessiere, E. Hebrard, B. Hnich, and T. Walsh. The complexity of global constraints. In *Proceedings of the 19th National Conference on Artificial Intelligence*, pages 112–117, 2004.
12. D. Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.
13. M. Carlsson and N. Beldiceanu. Arc-consistency for a chain of lexicographic ordering constraints. Technical Report T2002-18, Swedish Institute of Computer Science, 2002.
14. M. Carlsson and N. Beldiceanu. Revisiting the lexicographic ordering constraint. Technical Report T2002-17, Swedish Institute of Computer Science, 2002.
15. B.M.W. Cheng, K.M.F. Choi, J.H.M. Lee, and J.C.K. Wu. Increasing constraint propagation by redundant modeling: an experience report. *Constraints*, 4(2):167–192, 1999.
16. C.W. Choi, J.H.M. Lee, and P.J. Stuckey. Propagation redundancy in redundant modelling. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming*, pages 229–243, 2003.
17. C.W. Choi, J.H.M. Lee, and P.J. Stuckey. Removing propagation redundant constraints in redundant modeling. *(to appear) ACM Transactions on Computational Logic*, 2006.
18. J. Crawford, M. Ginsberg, E. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In *Proceedings of the 5th International Conference on Principles of Knowledge Representation and Reasoning*, pages 148–159, 1996.

19. R. Debruyne and C. Bessiere. Some practicable filtering techniques for the constraint satisfaction problem. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 412–417, 1997.
20. T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*, pages 93–107, 2001.
21. 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 the 8th International Conference on Principles and Practice of Constraint Programming*, pages 462–476, 2002.
22. P. Flener, A.M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, and T. Walsh. Matrix modelling. In *Proceedings of the Workshop on Modelling and Problem Formulation*, 2001.
23. F. Focacci and M. Milano. Global cut framework for removing symmetries. In *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*, pages 77–92, 2001.
24. E.C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Proceedings of the 9th National Conference on Artificial Intelligence*, pages 227–233, 1991.
25. A. Frisch, I. Miguel, Z. Kiziltan, B. Hnich, and T. Walsh. Multiset ordering constraints. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 221–226, 2003.
26. A.M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, and T. Walsh. Global constraints for lexicographical orderings. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, pages 93–108, 2002.
27. A.M. Frisch, C. Jefferson, and I. Miguel. Constraints for breaking more row and column symmetries. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming*, pages 318–332, 2003.
28. I.P. Gent. A symmetry breaking constraint for indistinguishable values. In *Proceedings of the 1st International Workshop on Symmetry in Constraint Satisfaction Problems*, 2001.
29. I.P. Gent, W. Harvey, and T. Kelsey. Groups and constraints: Symmetry breaking during search. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, pages 415–430, 2002.
30. I.P. Gent, W. Harvey, T. Kelsey, and S. Linton. Generic SBDD using computational group theory. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming*, pages 333–347, 2003.
31. I.P. Gent and B.M. Smith. Symmetry breaking during search in constraint programming. In *Proceedings of the 14th European Conference on Artificial Intelligence*, pages 599–603, 2000.
32. C. Gervet. Interval propagation to reason about sets: Definition and implementation of a practical language. *Constraints*, 1(3):191–244, 1997.
33. T.P. Kirkman. On a problem in combinatorics. *Cambridge and Dublin Mathematical Journal*, 2:191–204, 1847.
34. Z. Kiziltan. *Symmetry Breaking Ordering Constraints*. PhD thesis, Uppsala universitet, 2004.
35. Y.C. Law. *Using Constraints to Break Value Symmetries in Constraint Satisfaction Problems*. PhD thesis, The Chinese University of Hong Kong, 2005.

36. Y.C. Law and J.H.M. Lee. Global constraints for integer and set value precedence. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming*, pages 362–376, 2004.
37. Y.C. Law and J.H.M. Lee. Breaking value symmetries in matrix models using channeling constraints. In *Proceedings of the 20th Annual ACM Symposium on Applied Computing*, pages 375–380, 2005.
38. A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.
39. P. Meseguer and C. Torras. Solving strategies for highly-symmetric CSPs. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 400–405, 1999.
40. P. Meseguer and C. Torras. Exploiting symmetries within constraint satisfaction search. *Artificial Intelligence*, 129(1–2):133–163, 2001.
41. R. Mohr and G. Masini. Good old discrete relaxation. In *Proceedings of the 8th European Conference on Artificial Intelligence*, pages 651–656, 1988.
42. W.P. Pierskalla. The multidimensional assignment problem. *Operations Research*, 16(2):422–431, 1968.
43. J.-F. Puget. On the satisfiability of symmetrical constrained satisfaction problems. In *Proceedings of the 7th International Symposium on Methodologies for Intelligent Systems*, pages 350–361, 1993.
44. J.-F. Puget. Symmetry breaking revisited. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, pages 446–461, 2002.
45. J.-F. Puget. Symmetry breaking using stabilizers. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming*, pages 585–599, 2003.
46. J.-F. Puget. Symmetry breaking revisited. *Constraints*, 10(1):23–46, 2005.
47. C.M. Roney-Dougal, I.P. Gent, T. Kelsey, and S. Linton. Tractable symmetry breaking using restricted search trees. In *Proceedings of the 16th European Conference on Artificial Intelligence*, pages 211–215, 2004.
48. B.M. Smith and I.P. Gent. Reducing symmetry in matrix models: SBDS v. constraints. In *Proceedings of the Workshop on Symmetry in Constraint Satisfaction Problems*, 2001.
49. P. Van Hentenryck, P. Flener, J. Pearson, and M. Agren. Tractable symmetry breaking for CSPs with interchangeable values. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 277–282, 2003.