Using Constraints to Break Value Symmetries in Constraint Satisfaction Problems

Law, Yat Chiu

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Doctor of Philosophy
in
Computer Science and Engineering

September 2005

Abstract

Many real life problems can naturally be modeled as *constraint satisfaction problems* (CSPs), which can sometimes contain both *variable symmetries* and *value symmetries*. Tree search based CSP solving algorithms often suffer from symmetries, which creates symmetrically equivalent states in the search tree. Exploring more than one of the symmetrically equivalent states is a waste of search efforts. Adding symmetry breaking constraints to a CSP can force the search to visit only one of the symmetrical regions and helps reduce search space. While variable symmetry breaking constraints can be expressed relatively easily and executed efficiently by enforcing lexicographic ordering, value symmetry breaking constraints are often difficult to formulate. In this thesis, we propose two methods of using symmetry breaking constraints to tackle value symmetries. In the first method, we show theoretically *when* value symmetries in one CSP model correspond to variable symmetries in another CSP model of the same problem. We also show *when* variable symmetry breaking constraints in the two models, combined using channeling constraints, are consistent. Such results allow tackling value symmetries efficiently using additional CSP variables and channeling constraints. In the second method, we identify a common and important class of value symmetries, namely symmetries of *indistinguishable values*, and introduce *value precedence* to break such symmetries. Although value precedence can be expressed straightforwardly using if-then constraints in existing constraint programming systems, the resulting formulation is inefficient both in terms of size and runtime. We present two propagation algorithms for implementing global constraints on value precedence for integer and set variables respectively. We also characterize the propagation levels attained by various usages of the global constraints and the conditions when the constraints are consistent with variable symmetry breaking constraints. Extensive experiments are conducted to verify the feasibility and efficiency of our two proposals.

摘要

許多現實中的問題都可以被建模為約束滿足問題，並經常包含了變數對稱和數值對稱。以樹形搜索解決約束滿足問題時，對稱性會使得許多互相對稱的搜索狀態被重複檢索，而這些多餘的檢索會直接影響到樹形搜索的解決效率。在約束滿足問題中加入對稱破裂約束，可以強制樹形搜索只會查尋這些對稱搜索狀態的其中一個，從而減少搜索空間。儘管運用高效率的字典序約束可以相對容易地破裂變數對稱，現時卻沒有一般方法可以制定數值對稱的破裂約束。在這篇論文中，我們提出兩個使用對稱破裂約束以應付數值對稱的方案。第一，我們證明了在同一問題下兩個約束滿足問題模型中，其中一個模型的數值對稱和另一模型的變數對稱的對應條件。另外，我們亦顯示出當運用導向約束結合兩個模型時，怎樣維持兩模型的變數對稱破裂約束的相容性。這些結果令我們可以使用額外的約束滿足問題變數和導向約束有效率地破裂數值對稱。第二，我們辨別出一種常見及重要的數值對稱，稱為無識別數值對稱，及提出運用數值優先的概念破裂這種對稱。雖然在現有的約束編程系統中，條件性約束可以顯式地表達數值優先，但是條件性約束卻會大幅增加模型的規模並減慢執行效率。因此，我們設計了兩個傳播演算法，以分別實施整數數值優先和集合數值優先的全局約束。我們亦研究出使用數值優先全局約束時，各種運用方法與傳播程度的關係，以及維持數值優先約束與變數對稱破裂約束的相容性條件。廣泛的實驗證實了這兩個方案的可行性和高效率。

Acknowledgments

I would like to thank Prof. Jimmy Lee who brought me to the research area of constraint programming in 1999. Jimmy has always been a responsible supervisor and is very experienced in doing research. He is always energetic at any time. I would never forget our overnight work before paper submission deadlines. Luckily the results are usually rewarding. I would also like to thank my examiners Professors Alan Frisch, Kwong Sak Leung, and Ho Fung Leung. Their constructive comments help improve the quality of the thesis a lot.

The atmosphere of our research group would not be so lively without fellow groupmates Spencer Fung, Jeff Choi, Clotho Tsang, Gordon Lam, Charles Siu, and May Woo. Our discussions always improved the quality of our research. Thank you Andy Zhang and Fianny Jiang. Although you are not in the constraint group, being in the same office always let me improve my Mandarin and learn a lot of interesting things about different places in China.

Jimmy, Jeff, and I visited Université d'Orléans in France in April 2005. The visit widened my horizon a lot both in terms of research and the French culture. I am very grateful to Prof. Arnaud Lallouet and his family for their warm hospitality towards us. Also many thanks to Gary Leung who spent a lot of time accompanying us in Paris. *Merci beaucoup!*

Finally, I must give my best wishes to my parents and my girl friend Sara Choy. You are always supporting me wholeheartedly thoughout my studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many problems found in artificial intelligence and computer science, such as resource allocation, scheduling, timetabling, configuration, and satisfiability problems, can be modeled as finite domain *constraint satisfaction problems* (CSPs). This thesis reports work on symmetry breaking methods in CSPs, which is an important line of research in the constraint community. In this chapter, we first give a brief introduction to constraint satisfaction problems and an overview of constraint solving techniques. We then introduce the concept of symmetries and discuss the motivations of our research. We also give a summary of the goals achieved in our work.

1.1 Constraint Satisfaction Problems

The definition of *constraint satisfaction problems* (CSPs), in the sense of Mackworth [Mac77], can be stated as follows:

> *We are given a finite set of variables, a finite 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.

Example 1.1 The n-queens problem is to place n queens on a chessboard such that no two queens can attack each other. That means there can be no two queens on each row, each column, and each diagonal. Consider the 4-queens problem. It can be modeled as a CSP using four variables $\{x_1, x_2, x_3, x_4\}$. Each variable x_i denotes the column position of queen i in row i of the chessboard. The domain of possible values of the variables is thus $\{1, 2, 3, 4\}$. The choice of variables enforces that no two queens can be on the same row by default. The constraints in the CSP enforce that no two queens can be on the same:

- column: $x_i \neq x_j$ for $1 \leq i < j \leq 4$; and
- diagonal: $|x_i - x_j| \neq j - i$ for $1 \leq i < j \leq 4$.

One solution of this CSP is to assign values 2, 4, 1, and 3 to the variables x_1, x_2, x_3, and x_4 respectively. ■

Solving CSPs is NP-complete in general. Any solving algorithm is bound to require exponential time in the worst case. A common way to solve CSPs is to use backtracking search [GB65, Gas77, BP81, DP87, Nad89] incorporated with local consistency algorithms. Backtracking search systematically enumerates the search space of a CSP by trying each value from the domain of each variable, and testing whether there are any constraint violations. Local consistencies are properties which are local to individual constraints in a CSP. Examples include node and arc consistencies [Mon74, Mac77], bounds consistency [MS98], generalized arc consistency [MM88], and set bounds consistency [Ger95, Ger97]. Local consistency algorithms enforce these properties to a

CSP. The idea behind these algorithms is to transform a CSP into another CSP with the *same* set of solutions, but the size of the variable domains in the transformed CSP is decreased. During backtracking search, removing a value from a variable domain means pruning a sub-tree of the whole search tree. Therefore, removing unfruitful domain values effectively helps reducing the search space of backtracking search. Some common commercial CSP solvers such as ILOG Solver [ilo99], COSYTEC CHIP [cos03], and the CLPFD library of SICStus Prolog [sic05] are also based on these constraint satisfaction techniques.

1.2 Motivations and Goals

An important line of research in the constraint community is to investigate symmetries in CSPs. Generally speaking, a *symmetry* is a transformation of an entity which preserves *invariance* of the entity. Examples of symmetries include rotation of an equilateral triangle by 60° and reflection of the letter "A" along the vertical axis. In a CSP, the invariance that symmetries preserve is the satisfiability of the CSP. That is, CSP symmetries are mappings that preserve satisfiability of a CSP.

Example 1.2 Consider the CSP modeling the 4-queens problem in Example 1.1. Reflection of the chessboard along the horizontal axis corresponds to the transformations of variables from x_1, x_2, x_3, and x_4 to x_4, x_3, x_2, and x_1 respectively. This transforms the solution $x_1 = 2 \wedge x_2 = 4 \wedge x_3 = 1 \wedge x_4 = 3$ of the model to another solution $x_1 = 3 \wedge x_2 = 1 \wedge x_3 = 4 \wedge x_4 = 2$. This reflection is one of the eight symmetries of the CSP (including the identity mapping), and the two solutions are symmetrically equivalent to each other. The remaining symmetries of the CSP model correspond to the chessboard reflections along the vertical axis, main ("southeast") diagonal, and main skew ("northeast") diagonal, and rotations by 90°, 180°, and 270° clockwise. ■

Symmetries in a CSP are bad for solvers based on backtracking search, since symmetrically equivalent states in the search tree may be explored more than once. Among these states, either all of them lead to a solution, or none of them lead to one. Therefore, exploring any one of them is enough to determine the satisfiability of the CSP. By limiting the search to visit only one of the symmetrical regions, we can *break* the symmetries and reduce the search space of the problem. There are several approaches to symmetry breaking in CSPs. One main approach is to add *symmetry breaking constraints* [Pug93] to a CSP before the search, so that some of the symmetrical equivalent solutions are removed in the reformulated CSP.

Example 1.3 Consider the symmetry of horizontal reflection of the chessboard in the n-queens problem in Example 1.2. We can add the symmetry breaking constraint $x_1 < x_4$ to break the symmetry. As a result, among the two symmetrically equivalent solutions $x_1 = 2 \wedge x_2 = 4 \wedge x_3 = 1 \wedge x_4 = 3$ and $x_1 = 3 \wedge x_2 = 1 \wedge x_3 = 4 \wedge x_4 = 2$, only the former satisfies this constraint and remains as solution of the reformulated CSP. The latter, however, is no longer a solution after adding the constraint. ■

Other approaches of symmetry breaking include remodeling the problem completely [Smi01a, Smi01b], modifying search algorithms [GS00, FSS01], using heuristic methods to avoid symmetries [MT99, MT01], and designing specialized search algorithms [VFPA03, RDGKL04] that contains no symmetries. The work described in this thesis is focused on using symmetry breaking constraints.

There are two common types of CSP symmetries, namely *variable symmetries* and *value symmetries*. Crawford et al. [CGLR96] suggested that we can always break variable symmetries using lexicographic ordering constraints. In addition, efficient propagation algorithms [FFH+02, CB02b, CB02a] also 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. The goal of the research reported in this thesis is to remedy this difficulty, and we propose two methods to tackle this problem.

The first method makes uses of multiple viewpoints and channeling constraints [CCLW99] to break value symmetries in matrix models [FFH+02] of CSPs [LL03, LL05]. Matrix models, in which the CSP variables are indexed and organized into matrices, are commonly and naturally used to model many real life problems such as combinatorial, configuration, scheduling, design, and assignment problems [FFH+01]. Flener et al. [FFH+02] 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 all kinds of symmetries of the problem. We formally describe 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 of *when* value symmetries in one matrix model correspond to variable symmetries in another non-0/1 matrix model of the same problem. We 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* [Ben94, Gen01], and develop symmetry breaking constraints to tackle such symmetries. Examples of indistinguishable values include colors in graph coloring problems and personnel of the same rank in many rostering problems. We introduce the notion of *value precedence* [LL04] on sequences, and explain how imposing value precedence on a sequence of constrained variables can break symmetries of indistinguishable values in both integer and set domains. 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 maintaining value precedence as the basis of efficient implementation of two global constraints on integer and set variable sequences respectively. In addition, we give theoretical results to ensure consistent use of the constraints with other symmetry breaking constraints, and characterize the consistency levels attained by various usages of the global constraints.

Our work gives possibilities of using symmetry breaking constraints to break value symmetries in CSPs. We conduct extensive experiments on various problems with variable and value symmetries to verify the feasibility and efficiency of our two proposals.

1.3 Outline of the Thesis

The rest of the thesis is organized as follows. Chapter 2 provides the background to the thesis. We formally define the concept of CSPs, and briefly describe some CSP solving algorithms including systematic and local search solvers. In particular, we present how consistency techniques can be incorporated into backtracking search to increase CSP solving efficiency. We then give the concept of symmetries in CSPs, and defines two common types of CSP symmetries, namely variable and value symmetries. We discuss also common existing methods for breaking such types of symmetries. Chapter 3 presents one of our proposed methods of breaking value symmetries in matrix models using multiple viewpoints and channeling constraints. We show theoretically under what conditions value symmetries in one viewpoint would correspond to variable symmetries in another viewpoint of the same problem, and how to ensure consistency between variable symmetry breaking constraints in two connected viewpoints. Chapter 4 introduces our other method that breaks symmetries of indistinguishable values using value precedence. We present two propagation algorithms for implementing global constraints for integer and set value precedence respectively and discuss theoretical characteristics of the constraints. Chapter 5 presents experimental results using our two proposals. In Chapter 6, we present a brief review of the related work in symmetry breaking. We conclude the thesis in Chapter 7 by summarizing our contributions and giving discussions and possible directions for future research.

Chapter 2

Background

This chapter provides background to the thesis. We provide the definitions of various concepts relating to CSPs. Furthermore, a description of constraint satisfaction techniques is presented. This includes a brief overview of systematic and local search techniques for solving CSPs. In particular, we present how consistency techniques can be incorporated into backtracking search to increase CSP solving efficiency. We then delineate on the concept of symmetries in CSPs, 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 CSP is a triple (X, D, C), where $X = \{x_0, \ldots, x_{n-1}\}$ is a finite set of variables, D is a function that maps each $x \in X$ to its associated *domain* $D(x)$, giving the finite set of possible values for x, and C is a set of constraints. There are two common classes of variables in CSPs. A *constrained integer variable* (or simply *integer variable*) [ilo99] x has an integer domain, i.e., $D(x)$ is a finite integer set. A *constrained set variable* (or simply *set variable*) [Ger95, Ger97, ilo99] x has a set domain, i.e., 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* (or sometimes called the *lowest upper bound*) $PS(x)$ contains elements that belong to at least one of the possible values of the variable. The *required set* (or sometimes called the *greatest lower bound*) $RS(x)$ 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 $RS(x)$, then a fail is triggered. Similarly, adding a value to $RS(x)$ which is not in $PS(x)$ also triggers a fail. When $PS(x) = RS(x)$, the set variable is bound to its required set. 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* $x \mapsto a$ of a CSP (X, D, C) means that variable $x \in X$ is mapped to the value $a \in D(x)$. A *compound assignment* is a set of assignments $\{x_{i_j} \mapsto a_j \mid 1 \leq j \leq k\}$, where $\{x_{i_1}, \ldots, x_{i_k}\} \subseteq X$ and $a_j \in D(x_{i_j})$ for $1 \leq j \leq k$. Note the requirement that no variables can be assigned more than once in a compound assignment. We overload the $\mapsto$ operator to represent compound assignments such that $\langle x_{i_1}, \ldots, x_{i_k} \rangle \mapsto \langle a_1, \ldots, a_k \rangle$ means $\{x_{i_j} \mapsto a_j \mid 1 \leq j \leq k\}$. A *complete assignment* is a compound assignment for all variables in a CSP.

A constraint $c \in C$ places restrictions on a subset of variables, limiting the combination of values that these variables can take. The subset of variables is called the *scope* of c, denoted by $var(c)$. We abuse terminology by saying that the compound assignment $\phi = \{x_{i_j} \mapsto a_j \mid 1 \leq j \leq k\}$ also has a scope $var(\phi) = \{x_{i_1}, \ldots, x_{a_k}\}$. Given a constraint c and a compound assignment ϕ such that $var(c) \subseteq var(\phi)$, we use $c\phi$ as a predicate to denote whether

ϕ satisfies or violates c. A *solution* of a CSP $M = (X, D, C)$ is a complete assignment ϕ such that $\bigwedge_{c \in C} c\phi$ is true. The set of all solutions of a CSP 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 a CSP (X, D, C) means that 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 that the value b is added to the required set $RS(x)$ of x, i.e., $b \in x$. A *compound decision* is a set of decisions $\{x_{i_j} \rhd a_j \,|\, 1 \leq j \leq k\}$, where $\{x_{i_1}, \ldots, x_{i_k}\} \subseteq X$. Note that 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 also 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.

2.2 Constraint Satisfaction Techniques

Since CSPs are NP-complete [CLRS01] in general, any algorithm for solving CSPs is likely to require exponential time in problem size in the worst case. There are two general classes of algorithms for solving CSPs. The first class of algorithms is systematic search, which enumerates through the possible assignments of the variables. This class of algorithms is guaranteed to find a solution, if there is any, or prove no solutions exist. Therefore, it is sound and complete. The most common algorithm for performing systematic search is backtracking search [GB65, Gas77, BP81, DP87, Nad89]. To improve the efficiency of backtracking search, consistency and propagation techniques are used to remove inconsistent domain values, aiming at detecting failures earlier in the search.

Another class of algorithms is local search [DTWZ94, LLW95, VT96, CLS00], which traverses heuristically and sometimes stochastically the space of complete assignments. Local search algorithms are usually incomplete, i.e., they may fail to return a solution even if one exists. They cannot prove unsatisfiability of CSPs in general. Nevertheless, they have been shown to be efficient in solving some large-scale CSPs which are difficult for solving algorithms based on backtracking search. The work reported in this thesis concerns systematic search algorithms.

In the following subsections, we describe briefly backtracking search, consistency and propagation techniques, incorporation of backtracking search and constraint propagation, and local search for solving CSPs.

2.2.1 Backtracking Search

Backtracking search [GB65, Gas77, BP81, DP87, Nad89] is a common algorithm for performing systematic search to solve a CSP with integer variables. It starts with an empty compound assignment. It incrementally extends the compound assignment by choosing an uninstantiated variable and making an assignment to this variable from the variable's domain. If the new compound assignment assigns values to all the variables in some constraints of the CSP

function $backtracking\_search((X, D, C), \phi)$
if $var(\phi) = X \vee C = \emptyset$ **then**
    return $true$
endif
choose $x \in X \setminus var(\phi)$;
for each $a \in D(x)$ **do**
    $\phi := \phi \cup \{x \mapsto a\}$;
    $C' := \{c | c \in C \wedge var(c) \subseteq var(\phi)\}$;
    if $\bigwedge_{c \in C'} c\phi$ **then**
        return $backtracking\_search((X, D, C \setminus C'), \phi)$
    endif
    $\phi := \phi \setminus \{x \mapsto a\}$
endfor
return $false$

Figure 2.1: Backtracking Search Algorithm for CSPs with Integer Variables

and violates the constraints, the search backtracks and tries another assignment of the variable. If there are no more assignments to try for this variable, the search further backtracks to the previously chosen variable and tries another assignment of that variable. This process is repeated until either a solution is found or there are no more variables to backtrack to. In the latter case, backtracking search proves that no solutions exist for this CSP. Figure 2.1 shows the algorithm for backtracking search [MS98]. The call $backtracking\_search(M, \phi)$ with $\phi = \emptyset$ would start searching a solution for the CSP M with integer variables. If a solution is found, the search returns $true$, and $false$ otherwise. Backtracking search for CSPs with set variables can also be done similarly.

2.2.2 Consistency Techniques and Constraint Propagation

Consistency techniques are algorithms that remove values from the domains of variables without removing any solution to a CSP. The idea behind these techniques is based on the observation that if a value in a variable domain cannot satisfy a constraint in a CSP, then the value must not be in any solution of the CSP. There are several kinds of consistency techniques, which behave differently upon different kinds of constraints. Removal of a variable's domain value by maintaining consistency of one constraint may affect the consistency of another in a CSP, and make other values be removed from other variables. *Constraint propagation* is the spreading of pruning information from one constraint to another. Since consistency techniques enforce properties that are local to individual constraints in a CSP, the properties are usually called *local consistencies*. In the following, we briefly describe some common local consistencies and different algorithms for enforcing them.

Node Consistency

A constraint c is *node consistent* [Mon74, Mac77] with respect to the variable domains D if and only if either $|var(c)| \neq 1$ or for each $j \in D(x)$, where $\{x\} = var(c)$, $c\{x \mapsto j\}$ is true. A CSP (X, D, C) is *node consistent* if each constraint $c \in C$ is node consistent with respect to D.

Example 2.1 Suppose we have $D(x) = \{0, 1, 2, 3, 4\}$. Then the constraint $x < 3$ is *not* node consistent, or *node inconsistent*, because the assignments $x \mapsto 3$ and $x \mapsto 4$ do not satisfy the constraint. If we remove values 3 and 4 from $D(x)$, i.e., $D(x) = \{0, 1, 2\}$, then the constraint becomes node consistent. ■

An algorithm that enforces node consistency to a CSP is straightforward. For each unary constraint (i.e., a constraint with scope size 1) with scope $\{x\}$, if a value in the domain of x violates the constraint, we remove the value from the variable domain. Figure 2.2 shows an algorithm that enforces node

function *node_consistent*$((X, D, C))$
for each $c \in C$ with $|var(c)| = 1$ **do**
  let $var(c) = \{x\}$
  $D(x) := \{a \mid a \in D(x) \wedge c\{x \mapsto a\}\}$
endfor
return D

Figure 2.2: Node Consistency Algorithm

consistency to a CSP [Mac77]. The algorithm returns the new domains of the variables.

Node consistency can be achieved as a preprocessing step before starting the search. Once it is achieved, all unary constraints are always satisfied because the inconsistent values are already removed. Therefore, these unary constraints can be discarded/ignored.

Arc Consistency

Arc consistency is applicable to *binary* constraints, i.e., constraints with scopes of size 2. A constraint c is *arc consistent* [Mon74, Mac77] with respect to the variable domains D if and only if either $|var(c)| \neq 2$ or for each $j \in D(x)$, where $var(c) = \{x, y\}$, there exists $k \in D(y)$ such that $c\{x \mapsto j, y \mapsto k\}$ is true, and for each $k \in D(y)$ there exists $j \in D(x)$ such that $c\{x \mapsto j, y \mapsto k\}$ is true. A CSP (X, D, C) is *arc consistent* if each constraint $c \in C$ is arc consistent with respect to D.

Example 2.2 Suppose we have $D(x) = D(y) = \{0, 1, 2, 3, 4\}$. The constraint $x < y$ is *not* arc consistent, or *arc inconsistent*, because for $4 \in D(x)$ there does not exist a value in $D(y)$ such that the constraint is satisfied. Similarly, for $0 \in D(y)$ there does not exist a value in $D(x)$ such that the constraint is satisfied. If we remove 4 and 0 from $D(x)$ and $D(y)$ respectively, i.e., $D(x) = \{0, 1, 2, 3\}$ and $D(y) = \{1, 2, 3, 4\}$, then the constraint becomes arc consistent.

function $ac\_revise(c, D)$
if $|var(c)| = 2$ **then**
  let $var(c) = \{x, y\}$
  $D(x) := \{a \mid a \in D(x) \wedge \exists b \in D(y) \cdot [c\{x \mapsto a, y \mapsto b\}]\};$
  $D(y) := \{b \mid b \in D(y) \wedge \exists a \in D(x) \cdot [c\{x \mapsto a, y \mapsto b\}]\}$
endif
return D

Figure 2.3: Algorithm for Enforcing Arc Consistency on a Constraint

function $ac\_1((X, D, C))$
repeat
  $W := D;$
  for each $c \in C$ **do**
    $D := ac\_revise(c, D)$
  endfor
until $W = D;$
return D

Figure 2.4: Arc Consistency Algorithm AC-1

■

To make a constraint c arc consistent, we can simply delete all values from the domain of the variables in the constraint that cannot satisfy c. Figure 2.3 shows an algorithm that removes values from the variable domains to make a constraint arc consistent [Mac77].

To make a CSP arc consistent, it is not enough to revise each constraint in the CSP only once. When the revise algorithm $ac\_revise(c, D)$ removes a value from the domain of a variable, other previously revised constraints may become arc inconsistent again. The spreading of pruning information from one constraint to another is called *constraint propagation*. The AC-1 algorithm [Mac77], shown in Figure 2.4, performs constraint propagation and makes a CSP arc consistent by revising all the constraints repeatedly until there are no changes to the variable domains.

function $ac\_3((V, C))$
$Q := C;$
while $Q \neq \emptyset$ **do**
    delete a constraint $c \in Q$;
    $W := D$;
    $D := ac\_revise(c, D)$;
    if $W \neq D$ **then**
        $Q := Q \cup \{c' | c' \in C \wedge var(c) \cap var(c') \neq \emptyset\}$
    endif
endwhile
return D

Figure 2.5: Arc Consistency Algorithm AC-3

The AC-1 algorithm is quite inefficient because domain reduction of one variable makes all constraints in the CSP be revised again. It is obvious that this is not necessary since when a variable has its domain reduced, only constraints that contain this variable are affected. The AC-3 algorithm [Mac77] in Figure 2.5 is an improved version of AC-1. It only re-revises those constraints that can possibly be affected rather than re-revising all constraints without discretion.

There are more sophisticated and efficient arc consistency algorithms such as AC-4 [MH86], AC-5 [Per92], AC-6 [Bes94], AC-7 [BFR99], and AC2001 [BR01, BRYZ05], but their basic ideas are the same: to remove values from domains of variables that cannot contribute solutions to the CSP.

Generalized Arc Consistency

Node and arc consistencies handle only unary and binary constraints respectively in a CSP. When there are constraints in a CSP with more than two variables, node and arc consistency algorithms cannot reduce the variable domains. *Generalized arc consistency* (GAC) [MM88] (or sometimes called *hyper arc consistency*) is the *generalization* of arc consistency to non-binary constraints. It is defined using the notion of extensions of assignments. An *extension* of an assignment $x \mapsto a$ is a compound assignment ϕ that contains $x \mapsto a$, i.e., $(x \mapsto a) \in \phi$. A constraint c is *generalized arc consistent* (GAC) [MM88] with respect to the variable domains D if and only if for each $x \in var(c)$ and $j \in D(x)$, $x \mapsto j$ can be extended to a compound assignment ϕ with $var(\phi) = var(c)$ such that $c\phi$ is true. A CSP (X, D, C) is *generalized arc consistent* (GAC) if each constraint $c \in C$ is GAC with respect to D.

Example 2.3 Suppose we have $D(x) = D(y) = D(z) = \{1, 2, 3\}$. The constraint $x + y + z \leq 4$ is *not* generalized arc consistent, because the three assignments $x \mapsto 3$, $y \mapsto 3$, and $z \mapsto 3$ cannot be extended to satisfy the constraint. If we remove 3 from the domains of x, y, and z, i.e., $D(x) = D(y) = D(z) = \{1, 2\}$, then the constraint becomes generalized arc consistent. ■

Enforcing GAC on a constraint is expensive in general. However, it is sometimes possible to enforce GAC efficiently for some specific constraints, such as the all-different constraints [Rég94], cardinality constraints [Rég96], and lexicographic ordering constraints [FHK+02, CB02b, CB02a].

Set Bounds Consistency

Although node, arc, and generalized arc consistencies are applicable also to set variables, they are prohibitive to enforce. The major reason is that the domain size of a set variable is exponential to the size of its possible set. Also, the use of possible and required sets to represent the domain of a set variable does not allow the representation of all possible domains. For example, given a set variable x, the use of possible and required sets cannot represent the domain $D(x) = \{\emptyset, \{1\}, \{2\}\}$; the alternative $D(x) = \{a \,|\, \emptyset \subseteq a \subseteq \{1, 2\}\}$ is used instead. *Set bounds consistency* (SBC) [Ger95, Ger97] is a local consistency specifically defined for constraints involving set variables.

A constraint c on set variables is *set bounds consistent* (SBC) [Ger95, Ger97] with respect to the variable domains D if and only if for each set variable $x \in var(c)$, both $x \mapsto PS(x)$ and $x \mapsto RS(x)$ can be extended to a compound assignment ϕ with $var(\phi) = var(c)$ such that $c\phi$ is true. In other words, for each set variable $x \in var(c)$ and $a \in PS(x) \setminus RS(x)$, both $c \wedge a \in x$ and $c \wedge a \notin x$ are satisfiable. A CSP (X, D, C) is *set bounds consistent* (SBC) if each constraint $c \in C$ is SBC with respect to D.

Example 2.4 Suppose we have two set variables x and y with $PS(x) = PS(y) = \{0, 1, 2\}$, $RS(x) = \{0\}$, and $RS(y) = \emptyset$. The constraint $x \subseteq y$ is *not* set bounds consistent because $y \mapsto RS(y)$ (i.e., $y \mapsto \emptyset$) cannot be extended to satisfy the constraint. If we add 0 to $RS(y)$, i.e., $RS = \{0\}$, then the constraint becomes set bounds consistent. ■

2.2.3 Incorporating Consistency Techniques into Backtracking Search

Although backtracking search is more efficient than pure "generate and test" for solving CSPs, its performance is still not good enough to tackle CSPs. To improve efficiency, consistency techniques are incorporated into backtracking search [HE80, SF94, GS96, BFR99]. The idea is that in backtracking search, before choosing a variable and making assignment to a value for this variable from the variable's domain, constraint propagation is performed to reduce the domains of the unassigned variables. If the domain of a variables becomes empty after propagation, the current CSP cannot lead to a solution and backtracking occurs. The removal of domain values is undone upon backtracking, and the search makes another assignment for the variable. This process is continued until either a solution is found, or there are no more variables to backtrack to.

We give an example of performing forward checking [HE80] during search using the n-queens problem, which is to place n queens on a chessboard such that no two queens can attack each other. *Forward checking* enforces arc consistency on the constraints between the currently assigned variable and the unassigned variables in the search node. Recall the CSP model $M = (X, D, C)$ of the n-queens problem, which has four variables $X = \{x_1, x_2, x_3, x_4\}$ with domains $D(x_i) = \{1, 2, 3, 4\}$ for $1 \leq i \leq 4$. The constraints in C are $x_i \neq x_j$ and $|x_i - x_j| \neq j - i$ for $1 \leq i < j \leq 4$.

Figure 2.6 shows the search tree of the 4-queens problem performing forward checking. When the search starts, the CSP is already arc consistent. Therefore, no values are removed. Suppose we are trying the variables from x_1 to x_4 and the values 1 to 4 in order. By trying the assignment $x_1 \mapsto 1$, constraint propagation removes values that are on the same row and diagonal of the corresponding queen. Next, the search makes the assignment $x_2 \mapsto 3$. Constraint propagation removes all the values from the domain of x_3; hence backtracking occurs and another value in $D(x_2)$ is tried. Then the search makes the assignment $x_2 \mapsto 4$. After propagation, only one value in $D(x_3)$ is left. Trying the assignment $x_3 \mapsto 2$ makes the domain of x_4 empty. Therefore, the search backtracks again. This time, there are no values in the domains of x_3 and x_2 to be tried anymore, we have to try another value of x_1. After trying $x_1 \mapsto 2$ and propagation, there is only one value left in $D(x_2)$. By assigning x_2 to 4, x_3 must be assigned 1 and then x_4 be assigned 3. In this state, the search obtains a solution $\{x_1 \mapsto 2, x_2 \mapsto 4, x_3 \mapsto 1, x_4 \mapsto 3\}$ of the 4-queens problem with two backtracks. If we want to find another solution of the problem, the search would also backtrack to start finding another solution.

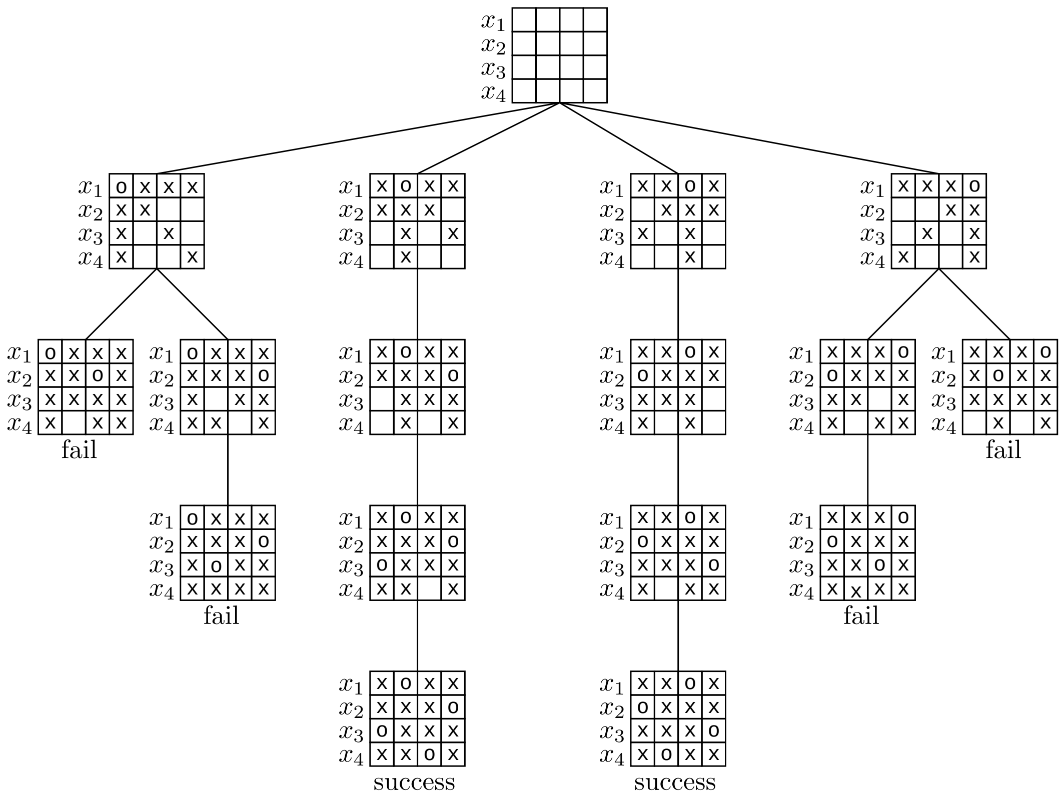

Figure 2.6: Forward Checking on the 4-queens Problem

2.2.4 Local Search

Local search algorithms typically start with a complete assignment of a CSP, and then progressively alter the complete assignment until a solution is found. The altering procedure aims at modifying the complete assignment to one that is "closer" to the solutions of the CSP. An example is the min-conflicts hill-climbing method [MJPL92], which is to choose an arbitrary variable and modify the current assignment of the variable to one that minimizes the number of constraint violations. Local search algorithms can be trapped in local optima, in which no altering of assignments can be made to move closer to the solutions. Therefore, another important component in local search algorithms is the mechanism to escape from such situations. A random restart or modification of the landscape of the search surface can be useful. GENET [DTWZ94] is a classical local search CSP solver that is based on min-conflicts hill-climbing for assignment altering and landscape modification for escaping local optima. It has been extended to many other local search solvers, such as E-GENET [LLW95], GLS [VT96], and $\mathcal{LSDL}$ [CLS00].

2.3 Symmetries

Symmetries can be inherent in a problem, or they can be introduced to CSP models of the problem as a result of distinguishing a set of indistinguishable objects of the problem with different identities. In this thesis, we are interested in symmetries in CSP models. A CSP *symmetry* is a solution-preserving bijective mapping from the set of all possible decisions to itself. Given a CSP M with the set of variables X. Let $\Theta = \{x \rhd a \,|\, x \in X \wedge a \in PS(x)\}$ be the set of all possible decisions of M. A CSP symmetry is a mapping $\lambda : \Theta \to \Theta$

such that

$$\theta \in sol(M) \iff \{\lambda(x \rhd a) \mid (x \rhd a) \in \theta\} \in sol(M).$$

Example 2.5 Consider the CSP modeling the 4-queens problem in Example 1.1. There is a symmetry λ in the model, where $\lambda(x_i \rhd j) = (x_j \rhd (5-i))$ for $1 \leq i, j \leq 4$. This symmetry corresponds to the 90° clockwise rotation of the chessboard described in Example 1.2. ■

The set of all symmetries in a CSP together with the composition operation forms a group [Hun74], which has four major properties. First, the set of symmetries is closed under composition. Second, composition of symmetries is associative. Third, the identity mapping is included as a symmetry. Fourth, for each symmetry λ in a CSP, the mapping λ^{-1}, i.e., the inverse of λ, is also a symmetry in the CSP. A symmetry group partitions the set of possible complete assignments into equivalence classes. Two complete assignments are in the same equivalence class if and only if there is a symmetry transforming one complete assignment to another. Therefore, in each equivalence class, either all elements are solutions of the CSP or all are not solutions. When solving a CSP using backtracking search, the symmetries in the CSP create symmetrically equivalent states in the search tree. Among these symmetrically equivalent states, either all of them lead to solution(s) of the CSP or none of them do. *Symmetry breaking* means avoid revisiting symmetrically equivalent states when solving the CSP. It can help solving CSPs with symmetries more efficiently.

In the following, we define two types of CSP 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, "prob010" in CSPLib [GW99], is to find a $\mathcal{W}$-week schedule of $\mathcal{G}$ groups, each containing $\mathcal{S}$ golfers, such that no two golfers

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

Figure 2.7: A Solution of the $(3, 2, 3)$ Instance of the Social Golfer Problem

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 $(\mathcal{G}, \mathcal{S}, \mathcal{W})$. If we respectively identify the golfers, the weeks, and the groups with unique numbers, then the problem contains the following forms of symmetries [FSS01]:

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 $M_G = (G, D_G, C_G)$ uses a variable $g_{i,k} \in G$ 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 CSP model is a matrix model [FFH+02], since G forms a 2-dimensional matrix of variables. Figure 2.7 gives a solution of the $(3, 2, 3)$ instance.

2.3.1 Variable Symmetries

A *variable symmetry* of a CSP $M = (X, D, C)$ is a symmetry σ such that $\sigma(x \rhd a) = (y \rhd a)$ for all possible decisions $x \rhd a$ of M, i.e., σ maps a decision to another such that the two decisions have the same decided value. Since the value a of a decision $x \rhd a$ remains unchanged upon a variable symmetry, for easier reading, we represent a variable symmetry σ with a 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

$$(\vec{x} \mapsto \vec{a}) \in sol(M) \iff (\sigma(\vec{x}) \mapsto \vec{a}) \in sol(M),$$

where $\vec{x}$ is a sequence of X, or equivalently

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

Example 2.6 Symmetry (1) of the social golfer problem is an example of variable symmetries in M_G. Consider the solution in Figure 2.7, we can exchange the variables of golfers 1 and 2 to obtain another solution with $\langle g_{1,1}, g_{1,2}, g_{1,3} \rangle \mapsto \langle 1, 2, 2 \rangle$ and $\langle g_{2,1}, g_{2,2}, g_{2,3} \rangle \mapsto \langle 1, 1, 1 \rangle$. 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 $1 \le k \le 3$. ■

Example 2.7 Symmetry (2) is another example of variable symmetries in M_G. In Figure 2.7, we can exchange the variables of weeks 1 and 2 to obtain another solution with $\langle g_{1,1}, \ldots, g_{6,1} \rangle \mapsto \langle 1, 2, 1, 3, 2, 3 \rangle$ and $\langle g_{1,2}, \ldots, g_{9,2} \rangle \mapsto \langle 1, 1, 2, 2, 3, 3 \rangle$. 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 $1 \le i \le 6$. ■

Variable symmetries can be broken using *lexicographic ordering* [FHK+02, CB02b, CB02a]. 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 $\vec{x} \le_{lex} \vec{y}$ or $\vec{y} \ge_{lex} \vec{x}$, if and only if

$$x_1 \le y_1 \text{ and } (\bigwedge_{1 \le i' < i} x_{i'} = y_{i'}) \to x_i \le y_i$$

for $2 \leq i \leq n$. 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 $\bigvee_{1 \leq i \leq n} x_i \neq y_i$.

In general, a variable symmetry σ can be broken by the lexicographic ordering constraint [CGLR96]

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

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 symmetry class except the lexicographically smallest one, with respect to the sequence $\vec{x}$, would be removed. Sometimes, these constraints can be simplified [Kiz04] to contain fewer variables. An example is the row ordering and column ordering constraints for row and column symmetries [FFH$^+$02]. For example, symmetry (1) of the social golfer problem 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 M_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 $1 \leq k < \mathcal{W}$. Note that these constraints are just a subset of all the required variable symmetry breaking constraints. They do not completely break the compositions of the row and column symmetries [FFH$^+$02]. There are methods to introduce extra constraints to break more [FJM03] but they are out of the scope of this thesis, which focuses on value symmetries.

2.3.2 Value Symmetries

A *value symmetry* [FFH$^+$02] under a subset $U \subseteq X$ of the variables of a CSP $M = (X, D, C)$ is a symmetry τ such that (1) $\tau(x \rhd a) = (x \rhd b)$ for all possible decisions $x \rhd a$ of variables $x \in U$ and (2) $\tau(x' \rhd a) = (x' \rhd a)$ for all possible decisions $x' \rhd a$ of variables $x' \notin U$. That is, for each variable $x \in U$, τ maps a decision of x to another decision of the same variable x. Furthermore, for each variable $x' \notin U$, τ maps a decision of x' to itself. Since the variable x of a decision $x \rhd a$ remains unchanged upon τ when $x \in U$, for easier reading, we represent a value symmetry τ with a 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 U and a compound decision θ such that $\tau(U, \theta) = \{x \rhd \tau(a) \,|\, (x \rhd a) \in \theta \wedge x \in U\} \cup \{x \rhd a \,|\, (x \rhd a) \in \theta \wedge x \notin U\}$. 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*.[1]

Value symmetry is similar to but more general than value interchangeability [Fre91]. 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* [Fre91] 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 $\tau(b) = a$.

Example 2.8 Symmetry (3) in the social golfer problem is an example of integer value symmetries in M_G. Consider the solution in Figure 2.7. We can permute the values assigned to the set of variables $U = \{g_{1,1}, \ldots, g_{6,1}\} \subseteq G$ from 1 to 2, from 2 to 3, and from 3 to 1 to obtain another solution with

[1] This illustrates the advantage of the notion of decisions over assignments, since the former allows descriptions of both integer and set value symmetries uniformly.

$\langle g_{1,1}, \ldots, g_{6,1} \rangle \rhd \langle 2, 2, 3, 3, 1, 1 \rangle$. 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, do not break all value symmetries in general. For example, in the social golfer problem, without loss of generality, we can always have the pre-assignments

$$\langle g_{1,1}, \ldots, g_{\mathcal{N},1} \rangle \rhd \langle \underbrace{1, \ldots, 1}_{\mathcal{S}}, \ldots, \underbrace{\mathcal{G}, \ldots, \mathcal{G}}_{\mathcal{S}} \rangle$$

as well as

$$\langle g_{1,k}, \ldots, g_{\mathcal{S},k} \rangle \rhd \langle 1, \ldots, \mathcal{S} \rangle$$

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 $\mathcal{G} - \mathcal{S}$ is, the fewer value symmetries can be broken by the pre-assignments.

Symmetries of indistinguishable values [Ben94, Gen01] is a common and special class of value symmetries. A set of *indistinguishable values* $\{v_1, \ldots, v_k\}$ under $U = \{x_1, \ldots, x_n\}$ implies $k!$ value symmetries τ under U, where the sequence $\langle \tau(v_1), \ldots, \tau(v_k) \rangle$ is a permutation of $\langle v_1, \ldots, v_k \rangle$.

Example 2.9 Example 2.8 shows a value symmetry τ under $U = \{g_{1,1}, \ldots g_{6,1}\}$ in M_G of the social golfer problem where $\tau(1) = 2$, $\tau(2) = 3$, 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. ■

Chapter 3

Breaking Value Symmetries Using Channeling Constraints

In this chapter, we introduce our 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 [FFH+02]. In the following, we first describe viewpoints, MAPs, and a general method to derive $n + 1$ viewpoints for modeling a MAP with n aspects as matrix models. We then give theoretical results showing *when* a value symmetry in a CSP (X, D_X, C_X) of viewpoint V corresponds to a variable symmetry in another CSP (Y, D_Y, C_Y) of viewpoint V' modeling the same problem. We also give conditions showing *when* variable symmetry breaking constraints in two viewpoints V and V', when connected with channeling constraints [CCLW99], are consistent. Using these results, we can tackle value symmetries in (X, D_X, C_X) by expressing variable symmetry breaking constraints using another viewpoint V' with variables Y and then connecting the two viewpoints using channeling constraints.

3.1 Viewpoints

When formulating a problem P into a CSP, a major process is to determine the variables and their associated domains, i.e., the sets of possible values of the variables, for the CSP. Different choices of variables and domains are results of viewing the problem P from different angles/perspectives. We define a *viewpoint*[1] [LL02] to be 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 $D(x)$. Given a CSP (X, D, C) modeling the problem P, we say that the pair $V = (X, D)$ is a viewpoint of P. For example, $V_G = (G, D_G)$ is a viewpoint of the social golfer problem, where $g_{i,k} \in G$ for $1 \leq i \leq \mathcal{N}$ and $1 \leq k \leq \mathcal{W}$, and $D_G(g_{i,k}) = \{1, \ldots, \mathcal{G}\}$. The solution in Figure 2.7 is expressed in the viewpoint V_G. In the following, given a CSP $M = (X, D, C)$, we also use the pair (V, C), where $V = (X, D)$, to represent M, i.e., $M = (X, D, C) = (V, C)$.

3.2 Multi-aspect Assignment Problems

In the social golfer problem, 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 social golfer problem 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 $Obj(1)$, $Obj(2)$, and $Obj(3)$ to denote the set of all golfers, weeks, and groups respectively in the social golfer problem, where $Obj(1) = \{1, \ldots, \mathcal{N}\}$, $Obj(2) = \{1, \ldots, \mathcal{W}\}$,

[1]Geelen [Gee92] used the notion of viewpoint loosely and informally without actually defining it.

and $Obj(3) = \{1, \ldots, \mathcal{G}\}$. 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 social golfer problem 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* [Pie68], 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* [FFH+01, FFH+02], which are CSPs in which the variables can be indexed and organized into one or more matrices.

3.3 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.

3.3.1 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 [FFH+02] 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 $x_s[i_1\cdots i_n \setminus i_s]$ to denote the variable $x_s[i_1]\cdots[i_{s-1}][i_{s+1}]\cdots[i_n]$. Hence, we have

$$\begin{aligned} X_s = \{x_s[i_1\cdots i_n \setminus i_s] \mid \textstyle\bigwedge_{1\le k\le n, k\ne s} i_k \in Obj(k)\} \quad \text{and} \\ PS(x_s[i_1\cdots i_n \setminus i_s]) = Obj(s). \end{aligned}$$

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* $V_1 = (X_1, D_{X_1}), \ldots, V_n = (X_n, D_{X_n})$ for a MAP. The subscript k in $V_k = (X_k, D_{X_k})$ denotes the aspect corresponding to the domains in V_k. The variables between any two aspect viewpoints V_s and V_t $(s \ne t)$ can be related by the channeling constraints [CCLW99]

$$x_s[i_1\cdots i_n \setminus i_s] \rhd i_s \iff x_t[i_1\cdots i_n \setminus i_t] \rhd i_t$$

for $\bigwedge_{1\le k\le n} i_k \in Obj(k)$. They collectively induce a *channeling function*

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

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.

Example 3.1 In the social golfer problem, $V_G = (G, D_G)$ 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 of the social golfer problem are $V_P = (P, D_P)$ and $V_W = (W, D_W)$. Viewpoint V_P uses the groups and

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

Figure 3.1: A Solution of $(3,2,3)$ Expressed in 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	$\{3\}$	$\emptyset$	$\{1,2\}$

Figure 3.2: A Solution of $(3,2,3)$ Expressed in V_W

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 $PS(p_{j,k}) = \{1, \ldots, \mathcal{N}\}$ and $PS(w_{i,j}) = \{1, \ldots, \mathcal{W}\}$. Figures 2.7, 3.1, and 3.2 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 $1 \leq i \leq \mathcal{N}$, $1 \leq j \leq \mathcal{G}$, and $1 \leq k \leq \mathcal{W}$. ■

3.3.2 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

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

The variables in Z denote 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 [CCLW99] between aspect viewpoint V_s and 0/1 viewpoint V_Z are

$$x_s[i_1\cdots i_n \setminus i_s] \rhd i_s \iff z[i_1]\cdots[i_n] \rhd 1$$

for $\bigwedge_{1\le k\le n} i_k \in Obj(k)$. They collectively induce a channeling function

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

from decisions in V_s to *only* those of the form "$z[i_1]\cdots[i_n] \rhd 1$" in V_Z, for $\bigwedge_{1\le k\le n} i_k \in Obj(k)$ (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.

Example 3.2 In the social golfer problem, 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 3.3 shows the same solution as those in Figures 2.7, 3.1, and 3.2, 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 $1 \le i \le \mathcal{N}$, $1 \le k \le \mathcal{W}$, and $1 \le j \le \mathcal{G}$. ■

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

Figure 3.3: A Solution of $(3, 2, 3)$ Expressed in V_Z

3.4 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 $V_s = (X_s, D_{X_s})$ and $V_t = (X_t, D_{X_t})$ are aspect viewpoints, and $V_Z = (Z, D_Z)$ is the 0/1 viewpoint.

3.4.1 From Aspect Viewpoint to Aspect Viewpoint

Value symmetries in M_s of aspect viewpoint V_s does *not* always correspond to variable symmetries in M_t of aspect viewpoint V_t. The following theorem states the conditions when this correspondence occurs.

Theorem 3.1 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 $U_s = \{x_s[i_1 \cdots i_n \setminus i_s] \mid \bigwedge_{1 \leq k \leq n, k \neq s} i_k \in Obj'(k)\}$, and

2. $Obj'(t) = Obj(t)$,

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

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

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 $B = \bigwedge_{\substack{1 \le k \le n, \\ k \ne s, k \ne t}} i_k \in Obj'(k)$. If $U_s = \{x_s[i_1 \cdots i_n \setminus i_s] \mid \bigwedge_{\substack{1 \le k \le n, \\ k \ne s}} i_k \in Obj'(k) \subseteq Obj(k)\}$, then

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

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

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

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

On the other hand,

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

The bijective mapping σ is solution preserving in V_t. Hence, it is a variable symmetry in V_t and corresponds to the value symmetry τ in V_s. $\square$

Theorem 3.1 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. The set of variable indices in U_s has to be the Cartesian product of a subset of the objects in each aspect. Second,

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 3.4: Another Solution of $(3,2,3)$ Expressed in V_W

$Obj'(t)$ must contain all the objects in aspect t, which corresponds to the domains in V_t.

We illustrate the two conditions in Theorem 3.1 using the $(3,2,3)$ instance of the social golfer problem. Let the golfers, weeks, and groups be the first, second, and third aspect respectively. Hence, $Obj(1) = \{1,\ldots,6\}$ and $Obj(2) = Obj(3) = \{1,2,3\}$. 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 $U = \{g_{1,1},\ldots,g_{6,1}\} = \{g_{i,k} \mid (i,k) \in Obj'(1) \times Obj'(2) \wedge Obj'(1) = \{1,\ldots,6\} \wedge Obj'(2) = \{1\}\}$, i.e., the set of all golfers in week 1. This satisfies condition (1) in Theorem 3.1. Condition (2) is also satisfied because $Obj'(1) = Obj(1) = \{1,\ldots,6\}$. Therefore, τ corresponds to a variable symmetry σ in V_P, which uses aspect 1 (golfers) to form the domains, 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, $Obj'(2) = \{1\} \neq Obj(2) = \{1,2,3\}$. Hence, condition (2) is *not* satisfied. In this case, τ does not correspond to any variable symmetry in V_W, which uses aspect 2 (weeks) to form the domains. Figure 3.4 shows the solution in V_W after applying τ to the solution in Figure 2.7. No variable symmetries can transform the solution in Figure 3.2 to the one in Figure 3.4.

3.4.2 From Aspect Viewpoint to 0/1 Viewpoint

The previous theorem specifies the conditions when a value symmetry in M_s corresponds to a variable symmetry in M_t. The following theorem shows that a value symmetry τ in M_s *always* correspond to a variable symmetry σ in M_Z.

Theorem 3.2 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

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

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

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

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

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

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, because $f_{s,Z}$ 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 corresponding to the value symmetry τ in M_s, and vice versa. $\square$

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 $1 \leq i \leq 6$.

3.5 Symmetry Breaking Constraints in Two Viewpoints

Recall that variable symmetry breaking constraints are easier to express than value symmetry breaking constraints. By Theorems 3.1 and 3.2, 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 [CCLW99], 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 [CCLW99]. Therefore, a possible way is to drop only constraints in C' which are propagation redundant [CLS03b, CLS03a] so that there would not be less propagation, but this is outside the scope of the thesis. 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* [FFH+02] if and only if at least one element in each symmetry class of assignments, defined by

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

Figure 3.5: All Solutions of Order 3 QEP* Expressed in V_N

the compositions of the symmetries under consideration, satisfies both sets of constraints. 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.

3.5.1 Inconsistent Symmetry Breaking Constraints in Two Viewpoints

The quasigroup existence problem (QEP), "prob003" in CSPLib [GW99], 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 3.5 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 $D_N(n_{i,j}) = \{1, \ldots, \mathcal{N}\}$.

QEP* contains four forms of symmetries. They are

1. the 180° rotation,

2. reflection along the main diagonal,

3. reflection along the main skew ("northeast") diagonal, and

4. 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 example, 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 (4) implies $\mathcal{N}!$ value symmetries τ under N in V_N, where the sequence $\langle \tau(1), \ldots, \tau(\mathcal{N}) \rangle$ is a permutation of $\langle 1, \ldots, \mathcal{N} \rangle$.

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

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

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

$$\langle n_{1,1}, n_{1,2}, n_{1,3}, \ldots, n_{3,1}, n_{3,2}, n_{3,3} \rangle \leq_{lex} \langle n_{3,3}, n_{3,2}, n_{3,1}, \ldots, n_{1,3}, n_{1,2}, n_{1,1} \rangle \quad \text{and}$$
$$\langle n_{1,1}, n_{2,1}, n_{3,1}, \ldots, n_{1,3}, n_{2,3}, n_{3,3} \rangle \leq_{lex} \langle n_{3,3}, n_{2,3}, n_{1,3}, \ldots, n_{3,1}, n_{2,1}, n_{1,1} \rangle$$

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 3.5 satisfy the former constraint, while θ_1, θ_3, and θ_5 satisfy the latter.

By Theorem 3.1, 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

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

Figure 3.6: All Solutions of Order 3 QEP* Expressed in V_R

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 [Kiz04], 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 3.6 shows the same six solutions as in Figure 3.5, but expressed in V_R. In the figure, solution θ'_i corresponds to solution θ_i in Figure 3.5 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.

3.5.2 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 an arbitrary linearization of the matrix to a single dimensional sequence. There are $|X_s|!$ possible 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 social golfer problem, $\langle golfer, week\rangle$ 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 $h_a \equiv x_s[i_1 \cdots i_n \setminus i_s]$, where

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

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

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

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

Example 3.3 The three aspects in the QEP* give $a = (i_{k_1} - 1) \times |Obj(k_2)| + i_{k_2}$ 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 $\langle 1, 2\rangle$ 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 $\langle 2, 1\rangle$ 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. ■

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, given the scanning sequence $\vec{h}_r$ in Example 3.3, we have

$$select(\vec{h}_r, \{n_{1,1}, n_{2,1}, n_{3,1}\}) = \langle n_{1,1}, n_{2,1}, n_{3,1}\rangle \quad \text{and}$$
$$select(\vec{h}_r, \{n_{1,2}, n_{2,2}, n_{3,2}\}) = \langle n_{1,2}, n_{2,2}, n_{3,2}\rangle.$$

3.5.3 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 3.3 Given two variables $x_s[i_1 \cdots i_n \setminus i_s]$ and $x_s[j_1 \cdots j_n \setminus j_s]$ in V_s and channeling function $f_{s,Z}$, we have

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

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 $\vec{z}_1 \triangleright \langle 0,\ldots,0,1,0,\ldots,0\rangle$ (i.e., only one "1" and the rest are all "0"), and similarly for $\vec{z}_2$. When $x_s[i_1\cdots i_n \setminus i_s] \leq x_s[j_1\cdots j_n \setminus j_s]$, 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. $\square$

The lemma states that two variables $x_s[i_1\cdots i_n \setminus i_s]$ and $x_s[j_1\cdots j_n \setminus j_s]$ in V_s are in non-decreasing order if and only if their corresponding sequences of variables $\vec{z}$ and $\vec{z}'$ 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 $\langle x_s[i], x_s[j]\rangle \triangleright \langle 1,2\rangle$, i.e., $x_s[i] \leq x_s[j]$, we have $\langle z[i][1], z[i][2], z[i][3]\rangle \triangleright \langle 1,0,0\rangle$ and $\langle z[j][1], z[j][2], z[j][3]\rangle \triangleright \langle 0,1,0\rangle$, 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 3.3 can be generalized to sequences of variables in V_s instead of only single variables.

Lemma 3.4 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 $x_s[i_1\cdots i_n \setminus i_s]$ 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 3.3 extended to sequences of variables in V_s instead of two variables $x_s[i_1 \cdots i_n \setminus i_s]$ and $x_s[j_1 \cdots j_n \setminus j_s]$. $\square$

We also recall the proposition by Crawford et al. [CGLR96] which states the consistency of variable symmetry breaking constraints in 0/1 viewpoint using lexicographic ordering.

Proposition 3.5 [CGLR96] 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 3.3 and 3.4 and Proposition 3.5, 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 3.6 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 $\sigma(x_s[i_1 \cdots i_n \setminus i_s]) = x_s[j_1 \cdots j_n \setminus j_s]$. 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 $x_s[i_1 \cdots i_n \setminus i_s]$ 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 3.4, $\vec{x}_s \le_{lex} \sigma(\vec{x}_s) \iff \vec{z} \ge_{lex} \sigma_z(\vec{z})$. By Proposition 3.5, $\vec{z} \ge_{lex} \sigma_z(\vec{z})$ and $\vec{z} \ge_{lex} \sigma'(\vec{z})$ are consistent, and hence so do $\vec{x}_s \le_{lex} \sigma(\vec{x}_s)$ and $\vec{z} \ge_{lex} \sigma'(\vec{z})$. $\square$

To maintain consistency between the variable symmetry breaking constraints for σ in V_s and σ' in V_Z, Theorem 3.6 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 3.4, a smaller-than order in V_s corresponds to a greater-than order in V_Z, and vice versa.

In the social golfer problem, Theorem 3.6 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.

We give a further example with a value symmetry which is *not* due to indistinguishable values. Consider the 4-queens problem and the CSP M modeling it introduced in Example 1.1. The problem can be considered as a MAP with two aspects: the rows and columns of the chessboard. The CSP M with variables $X = \{x_1, x_2, x_3, x_4\}$ is in an aspect viewpoint which uses a variable x_i for each row i with domain $D_X(x_i) = \{1, 2, 3, 4\}$ being the set of columns. The CSP M has (1) a variable symmetry σ that corresponds to the chessboard reflection along the horizontal axis, i.e., $\sigma(x_i) = 5 - i$ for $1 \leq i \leq 4$, and (2) a value symmetry τ under X that corresponds to the chessboard reflection along the vertical axis, i.e., $\tau(j) = 5 - j$ for $1 \leq j \leq 4$. Note that τ is *not* due to any indistinguishable values in M. We can break variable symmetry σ using the lexicographic ordering constraint $\langle x_1, x_2, x_3, x_4\rangle \leq_{lex} \langle x_4, x_3, x_2, x_1\rangle$, or equivalently $\langle x_1, x_2\rangle \leq_{lex} \langle x_4, x_3\rangle$. The value symmetry τ can be broken as a variable symmetry in the 0/1 viewpoint V_Z with variables $z_{i,j}$ and domains $D_Z(z_{i,j}) = \{0, 1\}$ for $1 \leq i, j \leq 4$, using the variable symmetry breaking constraint

$$\begin{array}{rl} & \langle z_{1,1}, z_{1,2}, z_{1,3}, z_{1,4}, z_{2,1}, z_{2,2}, z_{2,3}, z_{2,4}, z_{3,1}, z_{3,2}, z_{3,3}, z_{3,4}, z_{4,1}, z_{4,2}, z_{4,3}, z_{4,4}\rangle \\ \geq_{lex} & \langle z_{1,4}, z_{1,3}, z_{1,2}, z_{1,1}, z_{2,4}, z_{2,3}, z_{2,2}, z_{2,1}, z_{3,4}, z_{3,3}, z_{3,2}, z_{3,1}, z_{4,4}, z_{4,3}, z_{4,2}, z_{4,1}\rangle, \end{array}$$

or equivalently

$$\langle z_{1,1}, z_{1,2}, z_{2,1}, z_{2,2}, z_{3,1}, z_{3,2}, z_{4,1}, z_{4,2}\rangle \geq_{lex} \langle z_{1,4}, z_{1,3}, z_{2,4}, z_{2,3}, z_{3,4}, z_{3,3}, z_{4,4}, z_{4,3}\rangle.$$

Note that the aspect priority $\langle row, column\rangle$ is used to generate this constraint, i.e., the column aspect is in the last position. According to Theorem 3.6, this constraint is consistent with $\langle x_1, x_2\rangle \leq_{lex} \langle x_4, x_3\rangle$.

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 3.7 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 $U_s = \{x_s[i_1 \cdots i_n \setminus i_s] \mid \bigwedge_{\substack{1\le k\le n,\\ k\ne s}} i_k \in Obj'(k) \subseteq Obj(k)\}$ in V_s with $Obj'(t) = Obj(t)$;
- σ' be the variable symmetry in V_t corresponding to the value symmetry τ in V_s;
- $\vec{k} = \langle k_1, \ldots, k_{n-2}\rangle$ 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 $\vec{h} \le_{lex} \sigma(\vec{h})$ for σ and $\vec{h}'_a \le_{lex} \vec{h}'_b$ for σ' are consistent, where $\vec{h}'_j = select(sseq(\vec{q}), U'_j)$ for $j \in \{a, b\}$ and $U'_j = \{x_t[i_1 \cdots i_n \setminus i_t] \mid i_s = j \wedge \bigwedge_{\substack{1\le k\le n,\\ k\ne s, k\ne t}} i_k \in Obj'(k)\}$.

Proof By Theorem 3.2, the value symmetry τ in V_s corresponds to a variable symmetry σ'' in V_Z, where

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

If $\vec{h} = sseq(\langle k_1, \ldots, k_{n-2}, t\rangle)$, then by Theorem 3.6, $\vec{h} \le_{lex} \sigma(\vec{h})$ is consistent with

$$\vec{z} \;\; \ge_{lex} \;\; \sigma''(\vec{z}), \tag{3.1}$$

where $\vec{z} = sseq(\langle k_1, \ldots, k_{n-2}, t, s\rangle)$. From the definition of σ'', we can see that constraint (3.1) can be simplified to

$$select(\vec{z}, U''_a) \quad \geq_{lex} \quad select(\vec{z}, U''_b), \tag{3.2}$$

where $U''_j = \{z[i_1]\cdots[i_n] \,|\, i_s = j \wedge \bigwedge_{\substack{1\leq k\leq n,\\ k\neq s}} i_k \in Obj'(k)\}$ 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 (3.2) is now equivalent to

$$select(sseq(\vec{w}), U''_a) \quad \geq_{lex} \quad select(sseq(\vec{w}), U''_b). \tag{3.3}$$

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 3.4, constraint (3.3) is equivalent to

$$select(sseq(\vec{q}), U'_a) \quad \leq_{lex} \quad select(sseq(\vec{q}), U'_b), \tag{3.4}$$

where $U'_j = \{x_t[i_1 \cdots i_n \setminus i_t] \,|\, i_s = j \wedge \bigwedge_{\substack{1\leq k\leq n,\\ k\neq s}} i_k \in Obj'(k)\}$ 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 (3.4), therefore $\vec{h} \leq_{lex} \sigma(\vec{h})$ is consistent with constraint (3.4). $\square$

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 3.7 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 3.7 is applicable to symmetries of two indistinguishable values. The following corollary generalizes the theorem to handle multiple indistinguishable values.

Corollary 3.8 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 $U_s = \{x_s[i_1 \cdots i_n \setminus i_s] \mid \bigwedge_{\substack{1 \leq k \leq n, \\ k \neq s}} i_k \in Obj'(k) \subseteq Obj(k)\}$ 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 $U'_j = \{x_t[i_1 \cdots i_n \setminus i_t] \mid i_s = j \wedge \bigwedge_{\substack{1 \leq k \leq n, \\ k \neq s, k \neq t}} i_k \in Obj'(k)\}$.

Proof Direct consequence of Theorem 3.7 for all pairs of indistinguishable values. $\square$

For the QEP* example in Section 3.5.1, 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 (the 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 3.7, 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 (the 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 $\langle 2, 1 \rangle$. Therefore it is consistent with the symmetry breaking constraints in V_R.

Consider again the value symmetries in V_G of the social golfer problem. By Theorem 3.1, they correspond to variable symmetries in V_P. Theorem 3.7 and Corollary 3.8 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 2.7 satisfies both types of symmetry breaking constraints.

Chapter 4

Value Precedence Constraints

The previous chapter discusses the method of using multiple viewpoints and channeling constraints to express symmetry breaking constraints for value symmetries. In this method, existing modeling techniques are used and no new algorithms have to be designed. In this chapter, 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 that 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

In this section, we define *value precedence* on integer and set sequences. We give its use for breaking an important and common class of symmetries, namely symmetries of indistinguishable values [Ben94, Gen01]. We also describe new sets of constraints for maintaining value precedence are also described.

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

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

4.1.1 Value Precedence and Indistinguishable Values

Value precedence can be used for breaking symmetries of indistinguishable values. Given two indistinguishable values under 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.

Example 4.1 Given the symmetry of 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. Suppose $\vec{x} \mapsto \langle\{1, 2\}, \{0, 2\}, \{0, 1\}, \{1, 2\}, \emptyset\rangle$ is a solution of M, where $\vec{x} = \langle x_0, \ldots, x_4\rangle$. Under the symmetry, $\vec{x} \mapsto \langle\{1, 2\}, \{1, 2\}, \{0, 1\}, \{0, 2\}, \emptyset\rangle$ 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 4.1 Given the symmetries of indistinguishable values $\{v_0, \ldots, v_{k-1}\}$ under $U = \{x_0, \ldots, x_{n-1}\}$. 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 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. □

Besides symmetries of indistinguishable values, a CSP can have other types of symmetries, for example, variable symmetries. 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 [FFH+02]. 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 4.2 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 C_{var} is the set of symmetry breaking constraints $\vec{x} \leq_{lex} \sigma(\vec{x})$ (*resp.* $\sigma(\vec{x}) \leq_{lex} \vec{x}$) for some variable symmetries σ in M, and s and t are any two integer indistinguishable values under U. Maintaining $s \prec_{\vec{u}} t$ (*resp.* $t \prec_{\vec{u}} s$) is consistent with C_{var} 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 $\vec{x} \leq_{lex} \sigma(\vec{x})$ and $s \prec_{\vec{u}} t$ since the case of $\sigma(\vec{x}) \leq_{lex} \vec{x}$ and $t \prec_{\vec{u}} s$ is analogous. In each equivalence class of solutions of the variable symmetries, C_{var} keeps the lexicographically smallest 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 of 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 C_{var} 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. $\square$

According to Theorem 4.2, $x \leq y$ 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 [FMK+03], 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, $\{1, 2\} \leq_{set} \{1, 3, 4\} \leq_{set} \{1, 3\}$. Using this ordering, lexicographic ordering of set sequences becomes possible.

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 3.3.

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 4.3 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. $\Box$

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 4.4 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 $1 \leq i < n$ 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. $\Box$

4.2 Propagation Algorithms for Value Precedence

We develop two *value precedence global constraints* for maintaining value precedence, one for integer variables and the other for set variables, in ILOG Solver 4.4 [ilo99]. Both constraints use the same prototype $ValuePrecede(\vec{x}, s, t)$, meaning $s \prec_{\vec{x}} t$, where $\vec{x}$ is a variable sequence. Note that s and t are integer constants. In particular, GAC (*resp.* SBC) is enforced on the integer (*resp.* set) value precedence constraint. The integer and set versions of the propagation algorithms, namely **IntValuePrecede** and **SetValuePrecede** respectively, are similar. Their complexity is *linear to the length of the variable sequence*. Both of them make use of three pointers, namely α, β, and γ, which point to different indices of the sequence $\vec{x}$, but the pointers have different meanings for the integer and set versions. The two algorithms are also similar to that of the lexicographic ordering global constraint [FHK+02] in the sense that both maintain pointers running in opposite directions from the two ends of variable sequences. In subsequent discussions, we assume the variable sequence $\vec{x} = \langle x_0, \ldots, x_{n-1} \rangle$.

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 that $\alpha = n$. 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 no or only one variable 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:

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

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

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

1. *value t can be removed from the domains of the variables 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 4.1(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 4.1: 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, $\bigvee_{0 \le i < \gamma} x_i = s$ must be satisfied. Since $x_i \neq s$ for $0 \le i < \alpha$, $\bigvee_{0 \le i < \gamma} x_i = s$ can be refined to $\bigvee_{\alpha \le i < \gamma} x_i = s$. Furthermore, pointer β tells that $x_i \neq s$ for $\alpha < i < \beta$. Therefore, if $\gamma < \beta$, then $\bigvee_{\alpha \le i < \gamma} x_i = s$ 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 4.1(b), and 1 must precede 2 in $\vec{x}$ afterwards.

The propagation algorithm **IntValuePrecede**, shown in Figure 4.2, is based on the two pruning rules just described. Procedure *initialize*() is called when a value precedence constraint is posted. It finds initial values for the pointers α, β, and γ. In the procedure, we first search the position for α, starting from position 0. During the search, the first pruning rule is applied. After that, we search for a value for γ. Since value t is removed from $D(x_i)$ for $0 \leq i \leq \alpha$, γ must be greater than α and the position search for γ can start from position $\alpha + 1$. Note that the second pruning rule cannot be applied at this point because pointer β is not yet initialized. After fixing γ, procedure *updateBeta*() is invoked to find a value for β. By definition, β must be greater than α. Therefore the search starts from position $\alpha + 1$. After fixing β, the second pruning rule can be applied.

Procedure *propagate*(i) in Figure 4.2 is called whenever the domain of x_i is modified. If $\gamma < \beta$, then value precedence is already maintained and we can skip the propagation process. Otherwise, if $i = \alpha$ and $s \notin D(x_i)$, then pointers α and β have to be updated. The search for new position for α starts from position β, because x_β is the original second earliest variable that contains s in its domain. Once value s is removed from $D(x_\alpha)$, β becomes the first potential value for α. However, before the search, value t has to be removed from $D(x_i)$ for $\alpha < i < \beta$. During the search, the first pruning rule is applied. Pointer β is updated after finding a new value for α. The search for new value for β starts from position $\alpha + 1$. The procedure *updateBeta*() is called to update β. In the procedure, once β is updated, the second pruning rule is applied to check whether $\beta > \gamma$.

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

We need a procedure to update γ also. Procedure *checkGamma*(i) in Figure 4.2 is called whenever x_i is bound to a value. If $i < \gamma$ and x_i is bound

procedure *initialize*()
$\alpha := 0$;
while $\alpha < n \land s \notin D(x_\alpha)$ **do**
  $D(x_\alpha) := D(x_\alpha) \setminus \{t\}$;
  $\alpha := \alpha + 1$
endwhile
$\beta := \alpha$;
$\gamma := \alpha$;
if $\alpha < n$ **then**
  $D(x_\alpha) := D(x_\alpha) \setminus \{t\}$;
  repeat
    $\gamma := \gamma + 1$
  until $\gamma = n \lor D(x_\gamma) = \{t\}$;
  updateBeta()
endif

procedure *updateBeta*()
repeat
  $\beta := \beta + 1$
until $\beta = n \lor s \in D(x_\beta)$;
if $\beta > \gamma$ **then**
  $D(x_\alpha) := D(x_\alpha) \cap \{s\}$
endif

procedure *propagate*(i)
if $\beta \leq \gamma$ **then**
  if $i = \alpha \land s \notin D(x_i)$ **then**
    $\alpha := \alpha + 1$;
    while $\alpha < \beta$ **do**
      $D(x_\alpha) := D(x_\alpha) \setminus \{t\}$;
      $\alpha := \alpha + 1$
    endwhile
    while $\alpha < n \land s \notin D(x_\alpha)$ **do**
      $D(x_\alpha) := D(x_\alpha) \setminus \{t\}$;
      $\alpha := \alpha + 1$
    endwhile
    if $\alpha < n$ **then**
      $D(x_\alpha) := D(x_\alpha) \setminus \{t\}$
    endif
    $\beta := \alpha$;
    if $\alpha < n$ **then**
      updateBeta()
    endif
  else if $i = \beta \land s \notin D(x_i)$ **then**
    updateBeta()
  endif
endif

procedure *checkGamma*(i)
if $\beta < \gamma \land i < \gamma \land D(x_i) = \{t\}$ **then**
  $\gamma := i$;
  if $\beta > i$ **then**
    $D(x_\alpha) := D(x_\alpha) \cap \{s\}$
  endif
endif

Figure 4.2: The **IntValuePrecede** Propagation Algorithm

to t, then γ is updated to i, and the second pruning rule is applied to check whether $\beta > \gamma$. The **IntValuePrecede** algorithm enforces GAC on $s \prec_{\vec{x}} t$.

Theorem 4.5 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 $u \in D(x_\alpha) \setminus \{s, t\}$. 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 $D(x_i) \setminus \{s, t\}$ 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 $s \prec_{\vec{x}} t$. $\square$

4.2.2 Set Version

In **SetValuePrecede**, the meanings of the pointers α, β, and γ are different from those in the integer version. Pointer α is the smallest index of $\vec{x}$ 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 that $\alpha = n$. 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 that $\beta = n$. 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 that $\gamma = n$. 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
$PS(x_i)$	$\{2,3\}$	$\{1,2\}$	$\{1,2,3\}$	$\{2,3\}$	$\{1\}$
$RS(x_i)$	$\emptyset$	$\{2\}$	$\{3\}$	$\{2,3\}$	$\emptyset$

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

Pointer α tells that $s \notin x_i \vee t \in x_i$ for $0 \leq i < \alpha$. Hence, according to the set if-then value precedence constraints $s \in x_0 \vee t \notin x_0$ and $(s \notin x_j \wedge t \in x_j) \rightarrow \bigvee_{0 \leq i < j} (s \in x_i \wedge t \notin x_i)$ for $1 \leq j < n$, the constraints $s \in x_i \vee t \notin x_i$ for $0 \leq i \leq \alpha$ must be satisfied. Since $s \in PS(x_\alpha) \wedge t \notin RS(x_\alpha)$ must be true, $s \in x_\alpha \vee t \notin x_\alpha$ 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 $PS(x_i)$;*

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

(a)

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

(b)

Figure 4.3: Illustrating the Use of the Pointers α, β, and γ in **SetValuePrecede**

otherwise, s can be added to $RS(x_i)$.

In the above example, value 1 is not in $PS(x_0)$ so that we can remove 2 from $PS(x_0)$. Value 2 is in $PS(x_1)$; thus 1 is added to $RS(x_1)$. The resulting domains are shown in Figure 4.3(a).

Pointer γ tells that $s \notin x_\gamma \wedge t \in x_\gamma$. According to the if-then value precedence constraints, the constraint $\bigvee_{0 \le i < \gamma}(s \in x_i \wedge t \notin x_i)$ must be satisfied. By the meaning of α, this constraint can be refined to $\bigvee_{\alpha \le i < \gamma}(s \in x_i \wedge t \notin x_i)$. Furthermore, pointer β tells that $s \notin x_i \vee t \in x_i$ for $\alpha < i < \beta$. Therefore, if $\gamma < \beta$, then $\bigvee_{\alpha \le i < \gamma}(s \in x_i \wedge t \notin x_i)$ 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, we have $3 = \gamma < \beta = 4$. Therefore, value 1 can be added to $RS(x_\alpha)$ and 2 can be removed from $PS(x_\alpha)$, as shown in Figure 4.3(b).

Like **IntValuePrecede**, the **SetValuePrecede** algorithm in Figures 4.4 and 4.5 is based on two pruning rules. It contains four procedures with the same name as the integer version, and with similar structures. Procedure *initialize*() is called when $ValuePrecede(\vec{x}, s, t)$ is posted. It initializes the pointers α, β, and γ. The two pruning rules are applied during initialization. Procedure *propagate*(i) is called whenever the domain of variable x_i is modified, i.e., either $PS(x_i)$ or $RS(x_i)$ is modified. If $\gamma < \beta$, value precedence is already maintained and no propagation is needed. Otherwise, there are two different cases. In the first case, $i = \alpha \wedge (s \notin PS(x_i) \vee t \in RS(x_i))$, pointers α and β have to be updated. In the second case, $i = \beta \wedge (s \notin PS(x_i) \vee t \in RS(x_i))$, only pointer β has to be updated. After these two cases, procedure *checkGamma*(i) is called to check whether pointer γ has to be updated. This is different from the integer version, where *checkGamma*(i) is called only when x_i is bound to a value. This is because, in the set version, pointer γ may need update even when x_i is not bound.

The **SetValuePrecede** algorithm enforces SBC on $s \prec_{\vec{x}} t$.

Theorem 4.6 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

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

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

Figure 4.4: The **SetValuePrecede** Propagation Algorithm

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

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

Figure 4.5: The **SetValuePrecede** Propagation Algorithm (Continued)

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 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 $s \prec_{\vec{x}} t$. $\square$

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 $0 \leq i < j \leq k-1$, 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 $0 \leq i \leq k-2$. Although achieving the same value precedence effect, the former approach can theoretically achieve stronger propagation than the latter for both integer and set value precedence.

Theorem 4.7 Given an integer variable sequence $\vec{u}$, and a set of integer indistinguishable values $V = \{v_0, \ldots, v_{k-1}\}$ under U. GAC on $v_i \prec_{\vec{u}} v_j$ for $0 \leq i < j \leq k-1$ is strictly stronger than GAC on $v_i \prec_{\vec{u}} v_{i+1}$ for $0 \leq i \leq k-2$.

Proof The former is clearly as strong as the latter. To show strictness, consider the integer variable sequence $\vec{x} = \langle x_0, \ldots, x_3 \rangle$ with $D(x_0) = \{0, 3\}$, $D(x_1) = \{1, 3\}$, $D(x_2) = \{1, 2, 3\}$, and $D(x_3) = \{2\}$. Suppose $V = \{0, 1, 2\}$ is a set of indistinguishable values under $\{x_0, \ldots, x_3\}$. The constraints $\{0 \prec_{\vec{x}} 1, 1 \prec_{\vec{x}} 2\}$ are GAC with respect to the current variable domains, but the constraints $\{0 \prec_{\vec{x}} 1, 1 \prec_{\vec{x}} 2, 0 \prec_{\vec{x}} 2\}$ are not, since $x_0 \mapsto 3$ cannot be extended to a solution. $\Box$

Theorem 4.8 Given a set variable sequence $\vec{u}$, and a set of integer indistinguishable values $V = \{v_0, \ldots, v_{k-1}\}$ under U. SBC on $v_i \prec_{\vec{u}} v_j$ for $0 \leq i < j \leq k-1$ is strictly stronger than SBC on $v_i \prec_{\vec{u}} v_{i+1}$ for $0 \leq i \leq k-2$.

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 $\{y_0, \ldots, y_3\}$. The constraints $\{0 \prec_{\vec{y}} 1, 1 \prec_{\vec{y}} 2\}$ are SBC with respect to the variable domains, but $\{0 \prec_{\vec{y}} 1, 1 \prec_{\vec{y}} 2, 0 \prec_{\vec{y}} 2\}$ are not, since $y_0 \mapsto RS(y_0)$, i.e., $y_0 \mapsto \emptyset$, cannot be extended to a solution. $\Box$

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

It is still an open question whether GAC on each of $v_i \prec_{\vec{u}} v_j$ for $1 \leq i < j \leq k$ achieves GAC on the value precedence $v_1 \prec_{\vec{u}} \ldots \prec_{\vec{u}} v_k$ as a whole. Our conjecture is that this is true.

Chapter 5

Experiments

To demonstrate the feasibility and efficiency of our two proposals, we perform experiments on various problems including graph coloring, concert hall scheduling problem, social golfer problem, and 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 [ilo99] 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. Also, 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. The best number of fails and CPU time among the models for each instance are highlighted in bold.

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 must have different colors. This problem is a MAP with two aspects: the vertices and the colors. To model the problem into a CSP, we can 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, since given a solution, applying any permutations of the colors to the solution always obtains another solution. We build seven different models for this problem to illustrate the effects of our 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. [FFH$^+$02] 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 which makes use of 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 and connects V_X and V_Z with channeling constraints so that the symmetries of indistinguishable values can be tackled as variable symmetries in V_Z, using the lexicographic ordering constraints like those in all-bool but in reversed order. The global-adj and global-all models use adjacent-pair and all-pair postings of the value precedence global constraints. The intbool+glbadj model is the int-bool model in addition to 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 [Bré79].

Another consequence is that in the all-bool model, which does not contain V_X, variables with a smaller vertex number is branched earlier in the search.

Table 5.1 shows the experimental results of solving various instances in the Second DIMACS Challenge using the minimal number of colors (k^*). A cell labeled with a hyphen "-" means execution does not terminate in 2 hours of CPU time. In the results, models that break the symmetries of indistinguishable values (except all-bool) are generally more efficient than the no-break model. The if-then model is sometimes less efficient than no-break, despite the larger search space of the latter. 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 global-all model shows no advantage in pruning over the global-adj model, and is thus slightly less efficient due to the overhead in maintaining additional constraints. The if-then model also has the same propagation as global-adj and global-all, but execution is much slower because of the inefficient propagation of if-then constraints. The int-bool model is slower than the models using global constraints. However, it is more robust than the no-break and if-then models, which cannot solve myciel4.col and le450_5b.col respectively. The intbool+glbadj model has the same propagation as global-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 $exp_1 \rightarrow exp_2$ in general, where exp_1 and exp_2 are some constraints. According to the reference manual of Solver [ilo99], the propagation behavior of such constraints is that (1) propagation of exp_2 is triggered only after exp_1 is proved true, and (2) propagation of exp_1 is triggered only after exp_2 is proved false. In our experiments, however, we *empirically* show that GAC *is* enforced on the

Table 5.1: Experimental Results for the Graph Coloring Problem

instance	k^*	no-break fails	no-break time	if-then fails	if-then time	all-bool fails	all-bool time	int-bool fails	int-bool time	global-adj fails	global-adj time	global-all fails	global-all time	intbool+glbadj fails	intbool+glbadj 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[1]	5	4200	86.94	**35**	6.37	16361	1.8	39	1.72	**35**	**0.84**	**35**	0.99	**35**	1.91

[1]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`.

integer if-then value precedence constraints. This can be seen from the results that the if-then and global-adj models have the same amount of propagation, and our propagation algorithm is enforcing GAC on integer value precedence. This fact is probably due to the specific form of if-then constraints used in value precedence, where the left-hand side of a if-then constraint is a unary assignment constraint, and the right-hand-side is a conjunction 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

There are k identical fine rooms for a concert hall, and the hall director receives n applications to use the rooms. 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 5.1 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 $k = 2$,[2] but is a special case of the temporal knapsack problem [BFH$^+$05], in which more than one room can be requested in each application.

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 value $k+1$ is a dummy value to denote the application is rejected. 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 $i < j$

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

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

<table>
<tr><th>time</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td>room 1</td><td colspan="2">app 1</td><td>app 3</td></tr>
<tr><td>room 2</td><td></td><td colspan="2">app 2</td></tr>
</table>

Total income: 30

Figure 5.1: An Instance of the Concert Hall Scheduling Problem with 2 Rooms and 4 Applications, and One Optimal Solution

to break the symmetry. Moreover, all domain values except the dummy value $k+1$ are indistinguishable values. This is the only benchmark problem in which not all the domain values are indistinguishable. We build the no-break, if-then, all-bool, int-bool, global-adj, global-all, and intbool+glbadj models, which have the same meaning as in the graph coloring problem. 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 5.2 shows the experimental results. Like the graph coloring problem, models breaking the symmetries of indistinguishable values (i.e., if-then, int-bool, global-adj, global-all, and intbool+glbadj) are generally more efficient than no-break, which is again generally more efficient than all-bool. The global-adj and global-all models have the same number of fails, and execution time between them is negligible. Unlike the graph coloring problem, the int-bool model is more efficient than global-adj and global-all in most instances, but they are all much more efficient than the if-then model. Actually, the different propagation behaviors between int-bool and global-adj (as well as global-all) make the smallest-domain-first variable ordering heuristic choose different variables to be assigned next during search. It seems that the heuristic fits more to int-bool than to global-adj and global-all so that int-bool has a smaller overall search tree. The intbool+glbadj model has the same propagation as global-adj, but its execution is slower. We also tried solving an instance obtained from the ACM/ICPC 2003 web site containing 1000 applications within 365 days. The global-adj, global-all, and int-bool can be solved in about 3.2 hours CPU time, while the other models do not terminate after 6 hours execution.

Bartlett et al. [BFH+05] suggested a model, which uses one 0/1 variable for each application to denote whether the application is accepted, for the temporal knapsack problem. The model does not keep track of which hall an application is allocated to. We are aware that this 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.

5.3 Social Golfer Problem

Our next set of experiments makes use of the social golfer problem. For this problem, we build integer and set models in V_G and V_P respectively as two bases and tackle the symmetries of 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).

instance	k	no-break		if-then		all-bool		int-bool		global-adj		global-all		intbool+glbadj	
		fail	time	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time
instance 1	2	7770	2.33	**3928**	2.58	8615	1.49	**3928**	1.32	**3928**	**1.19**	**3928**	**1.19**	**3928**	1.37
instance 1	3	187089	59.34	58624	56.63	266788	48.81	**44040**	**16.86**	58624	20.42	58624	20.46	58624	24.63
instance 1	4	3448725	1100.68	522716	673.77	4246755	762.93	**374781**	**155.89**	522716	186.89	522716	193.13	522716	235.93
instance 1	5	-	-	4239414	6605.37	-	-	**2636415**	**1152.41**	4239414	1545.08	4239414	1600.23	4239414	2063.9
instance 2	2	4273	1.12	2176	1.37	34206	4.28	**2175**	0.65	2176	**0.57**	2176	**0.57**	2176	0.69
instance 2	3	39827	12.78	26888	26.11	2276645	310.12	**10398**	**4.06**	26888	9.01	26888	9.11	26888	11.22
instance 2	4	416916	139.22	143492	192.83	48567736	6787.48	**62033**	**26.3**	143492	49.55	143492	51.53	143492	64.95
instance 2	5	3988530	1368.41	1085772	1891.34	-	-	**304979**	**134.9**	1085772	371.37	1085772	391.02	1085772	515.35
instance 3	2	5455	2.17	**2752**	1.93	11267	1.91	2753	1.17	**2752**	**1.1**	**2752**	**1.1**	**2752**	1.26
instance 3	3	154517	57.78	66633	64.43	447431	80.7	**33164**	**14.92**	66633	26.02	66633	26.36	66633	32.09
instance 3	4	3292195	1227.8	786525	998.99	8944941	1588.94	**301427**	**136.67**	786525	317.28	786525	322.09	786525	399.08
instance 3	5	-	-	-	-	-	-	**2223074**	**1049.41**	6880554	2833.06	6880554	2950.12	6880554	3685.6
instance 4	2	161575	39.34	**83163**	47.84	297649	33.67	83164	23.27	**83163**	**20.38**	**83163**	**20.38**	**83163**	24.78
instance 4	3	10851983	2528.77	3027166	2802.17	31956512	3914.99	**2069561**	**605.59**	3027166	738.45	3027166	731.08	3027166	953.92
instance 5	2	6652	1.91	**3336**	2.22	11757	1.56	**3336**	1.1	**3336**	**0.99**	**3336**	**0.99**	**3336**	1.12
instance 5	3	205487	57	49722	46.91	630171	82.47	**47312**	16.48	49722	**15.45**	49722	15.85	49722	18.46
instance 5	4	4828319	1322.54	485948	585.31	17168657	2157.77	**415504**	150.39	485948	**147.91**	485948	153.68	485948	190
instance 5	5	-	-	3983836	5771.1	-	-	**2190385**	**841.43**	3983836	1215.36	3983836	1290.53	3983836	1638.05

Table 5.2: Experimental Results for the Concert Hall Scheduling Problem

5.3.1 Integer Model

In the integer model, symmetries (1) and (2) are variable symmetries, and they 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 $\langle g_{i,1}, g_{i,2} \rangle <_{lex} \langle g_{i+1,1}, g_{i+1,2} \rangle$, because two golfers can meet each other at most once throughout the weeks. 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 constraint propagation on the symmetry breaking constraints in V_P. We also build the global-adj and global-all models that breaks the symmetries using global constraints. The all-bool model is the same as the one used by Frisch et al. [FHK$^+$02] except that we apply the same simplification technique as above to simplify the row and column lexicographic ordering constraints. That is, the lexicographic ordering constraints on adjacent pairs of golfers involve variables with index k being 1 or 2 only, and those on adjacent pairs of weeks involve variables with index i being from 1 to $\mathcal{G} + 1$ only. Since there are two models int-bool and int-set using the multiple viewpoints method, we corresponding build two models intbool+glbadj and intset+glbadj using the combined method. Table 5.3 shows the experimental results.

Again, the global-adj and global-all models are the fastest among all. The performance of int-set and int-bool approaches that of the global constraints

$\mathcal{G},\mathcal{S},\mathcal{W}$	if-then		all-bool		int-bool		int-set		global-adj		global-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

Table 5.3: Experimental Results for the Social Golfer Problem, using Integer Variables

models. The global-adj, global-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 much better than the if-then and all-bool models in most instances. 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 the int-set model shows its robustness and is significantly faster. Both the intbool+glbadj and intset+glbadj models again have the same propagation as global-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 [BB02, BB05] 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 global-adj and global-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) can be 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 5.4 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 global-adj. Their runtime are similar in many instances. In larger instances (down in the table), global-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 the if-then and all-bool models, confirming the efficiency of our set propagation algorithm. Note that the all-bool model 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 global-all model achieves more propagation then global-adj does in some instances. The difference in the number of fails, however, is usually small, so the overhead of extra postings of the global constraints cannot be compensated. An exception is the $(4, 4, 4)$ instance where global-all has significantly fewer fails than global-adj, and all-bool is the best among all models. Actually, global-all posts $O(\mathcal{N}^2) = O(\mathcal{G}^2\mathcal{S}^2)$ more constraints than global-adj. Therefore, instances with more golfers incur more overhead than those with fewer golfers. The 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 global 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 global-adj and global-all are the same in all instances. It seems that the extra propagation of global-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 global-all does not guarantee a pruning in search space. Indeed, the search states that lead to extra propagation by global-all must be reachable during search in order for an actual pruning in practice.

5.4 Steiner Triple System

Let $X = \{1, \ldots, v\}$ be a set of $v \geq 3$ elements. 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 the set

$$\{\{1,2,3\},\{1,4,5\},\{1,6,7\},\{2,4,6\},\{2,5,7\},\{3,4,7\},\{3,5,6\}\}.$$

A Steiner triple system of order v exists if and only if $v \equiv 1, 3 \pmod 6$ [Kir47].

The problem of finding Steiner triple systems of order v is a MAP with two aspects: the triples and the set X. Therefore, the problem can be modeled as a CSP using an aspect viewpoint V_B with a set of set variables $B = \{b_1, \ldots, b_n\}$ (where $n = \frac{v(v-1)}{6}$) and possible sets $PS(b_i) = \{1, \ldots, v\}$. In this viewpoint, the variable symmetries are that any two variables b_i and b_j can be exchanged. They can be broken by the constraints $\min(b_i) \leq \min(b_{i+1})$

$\mathcal{G}, \mathcal{S}, \mathcal{W}$	if-then		all-bool		set-bool		set-int		global-adj		global-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

Table 5.4: Experimental Results for the Social Golfer Problem, using Set Variables

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). Furthermore, the values in the possible set $\{1, \ldots, v\}$ are indistinguishable values. Such value symmetries can be broken by using global constraints (global-adj and global-all), channeling with the 0/1 viewpoint (set-bool), and channeling with another aspect viewpoint V_X with set variables $x_1, \ldots, x_v$ and possible sets $PS(x_j) = \{1, \ldots, n\}$ (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 5.5 show that the setset+glbadj model achieves the best results. It combines the benefits of set-set and global-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, our two methods need not be used alone. They can be used simultaneously to obtain even more speedup.

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

Table 5.5: Experimental Results for Steiner Triple Systems

Chapter 6

Related Work

In this chapter, we give an overview of related work on symmetry breaking in CSPs. We briefly describe some common approaches of symmetry breaking techniques, and discuss the merits and shortcomings of these approaches.

6.1 Symmetry Breaking Techniques

Symmetry breaking techniques can be classified into five categories: problem remodeling, using symmetry breaking constraints, modifying search algorithms, using heuristic methods, and building specialized search algorithms. In the following, we describe each of the categories.

6.1.1 Problem Remodeling

An approach to remove symmetries in CSPs is to remodel the problem with fewer symmetries. A convention is to use set variables whenever possible. Suppose we are modeling a group of indistinguishable objects in a problem. Using a number of integer variables introduces symmetries to a CSP, since the order of the values assigned to the integer variables is important. For example, $\{x_1 \mapsto 1, x_2 \mapsto 2, x_3 \mapsto 3\}$ is different from $\{x_1 \mapsto 3, x_2 \mapsto 2, x_3 \mapsto 1\}$.

Using set variables [Ger95, Ger97] instead of integer variables avoid introducing symmetries to the CSP. In the previous example, one set variable x can be used instead. Since the order of the elements in a set now becomes unimportant, i.e., $\{1, 2, 3\} = \{3, 2, 1\}$, we can use the assignment $x \mapsto \{1, 2, 3\}$ instead and do not introduce extra symmetries to the CSP. Besides this convention, since this approach is highly problem specific, there are no other general principles on how to do so. Smith [Smi01a, Smi01b] performed a case study on the social golfer problem and discusses several CSP models for the problem with different amount of symmetries. Due to its problem dependency, this approach cannot be widely used in real life applications.

6.1.2 Symmetry Breaking Constraints

In the second approach, *symmetry breaking constraints* [Pug93] are added to a CSP to make fewer number of symmetrical regions being traversed during the search for solutions. Crawford et al. [CGLR96] suggest a general scheme to add symmetry breaking predicates to satisfiability problems. Aloul et al. [ASM03] improve 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 [FHK$^+$02, CB02b, CB02a]. That is, 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. Such constraints help break symmetries of indistinguishable objects, each of which is represented by more than one variable in the CSP. Flener et al. [FFH$^+$02] identified row and column symmetries in 2-dimensional matrix models, which are commonly found in many CSPs. These symmetries generalized to index symmetries in higher dimensional matrix models Row and column symmetries are the variable symmetries that every two rows and every two columns in a matrix of variables can be exchanged. Suppose that there are n rows and m columns in the matrix. They collectively imply $n!m!$ variable symmetries. Flener et al. [FFH+02] showed that adding constraints to lexicographically order both the rows and the columns are always consistent, although they do not break all the row and column symmetries. For example, let $X = \{x_{i,j} \mid 1 \leq i \leq n \wedge 1 \leq j \leq m\}$ be a set of variables organized in a two-dimensional matrix. We can consistently add the symmetry breaking constraints

$$\langle x_{1,1}, \ldots, x_{1,m} \rangle \leq_{lex} \ldots \leq_{lex} \langle x_{n,1}, \ldots, x_{n,m} \rangle$$

to order the rows and

$$\langle x_{1,1}, \ldots, x_{n,1} \rangle \leq_{lex} \ldots \leq_{lex} \langle x_{1,m}, \ldots, x_{n,m} \rangle$$

to order the columns. Multiset ordering constraints [FMK+03] and allperm constraints [FJM03] are also available for breaking row and column symmetries in matrix models. Gent [Gen01] designs special constraints which assume that all the domain values of a set of variables are indistinguishable and breaks the value symmetries among them. Our value precedence constraints also break symmetries of two indistinguishable values, and they can be posted multiple times to break symmetries of multiple indistinguishable values.

6.1.3 Modifications of Search Algorithms

The third approach is to break symmetries dynamically during search [BFP88, BFP96, BW99, BW02, GS00, FSS01, FM01]. 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 [GS00, GHK02]. Upon backtracking of the search, SBDS [GS00] adds a symmetry breaking constraint for each symmetry in a CSP to remove all the states which are symmetric to the one that causes backtracking. Suppose we have the current compound assignment $\{x_{i_1} \mapsto a_1, \ldots, x_{i_k} \mapsto a_k\}$ during backtracking search, and the new assignment $x_{k+1} \mapsto a_{k+1}$ triggers a backtrack. Upon the backtrack, for each symmetry λ in the CSP, SBDS adds the constraint

$$(\bigwedge_{1 \leq j \leq k} \lambda(x_{i_j} \rhd a_j)) \wedge \neg(x_{i_{k+1}} \rhd a_{k+1}) \rightarrow \neg\lambda(x_{i_{k+1}} \rhd a_{k+1})$$

to the CSP.[1] Backofen and Will introduced *Symmetry Excluding Search* (SES) [BW99, BW02], 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. Since CSP symmetries form symmetry groups, Gent et al. [GHK02] incorporate GAP [gap05], a computational group theory system, to SBDS such that large symmetry groups can be handled efficiently.

Another representative of the dynamic approach is called *Symmetry Breaking via Dominance Detection* (SBDD) and its variants [FSS01, Pug02, Pug05, BB02, BB05]. In SBDD [FSS01], 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

[1] SBDS is defined on integer variables only. As we have defined a CSP symmetry as a decision-to-decision mapping, we need to use decisions in the above constraint.

[BB02, BB05] propose 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. [GHKL03] again use computational group theory to extend SBDD. They also propose a generic dominance checker, which avoids the necessity of implementing a specific dominance 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 [SG01] show how the use of symmetry breaking constraints and SBDS can be combined to break row and column symmetries in matrix models [FFH+02]. They also empirically compare several different approximations to eliminating the symmetries and an exact method that eliminates the symmetries completely for small matrices. Puget [Pug03] show how to combine the use of lexicographic ordering constraints and SBDD for row and column symmetries. He also presents 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.

6.1.4 Heuristic Methods

The fourth approach to tackle symmetries in CSPs is to use the symmetries to guide the search. Meseguer and Torras [MT99, MT01] 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.

6.1.5 Specialized Search Algorithms

The fifth approach is to construct specialized search trees that does not contain symmetries [VFPA03, RDGKL04]. Van Hentenryck et al. [VFPA03] study three classes of CSPs for which symmetry breaking is tractable. These CSP classes, featuring specific forms of symmetries of indistinguishable values,[2] allow symmetry breaking to be performed in constant time and space during search using dedicated search procedures. Roney-Dougal et al. [RDGKL04] generalize the idea by Van Hentenryck et al. [VFPA03] and introduce 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. show 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 a set of 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 6.1 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

[2] Van Hentenryck et al. [VFPA03] used the term interchangeable values to denote indistinguishable values.

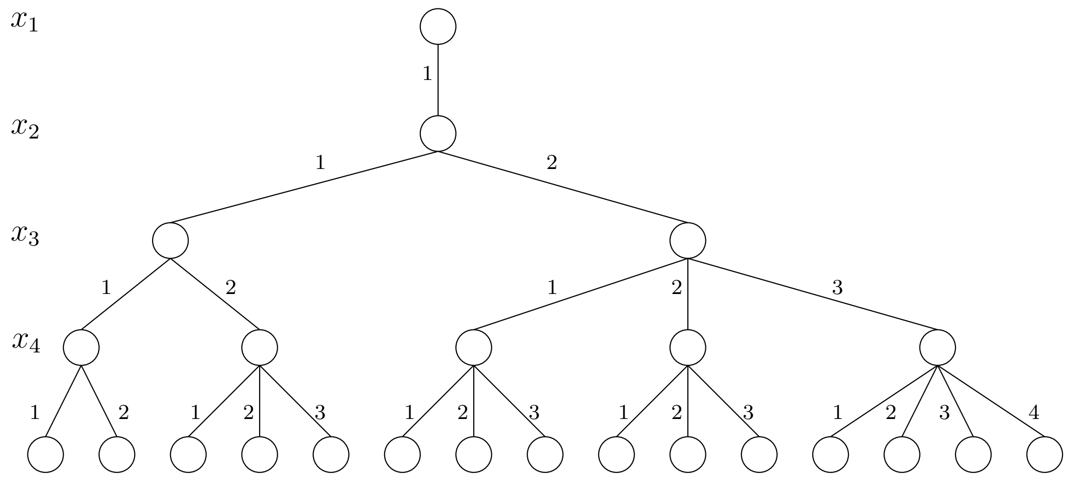

Figure 6.1: A GE-tree for Symmetries of Indistinguishable Values $\{1, 2, 3, 4\}$ Under $\{x_1, x_2, x_3, x_4\}$

nodes of the GE-tree represent the unique solutions under the symmetries. This GE-tree tree can be constructed using a simple rule by Van Hentenryck et al. [VFPA03] 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 as a set called 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 6.1, 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 6.2. 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 $\vec{u} = \langle x_1, x_2, x_3, x_4 \rangle$.

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

Figure 6.2: Solutions of $1 \prec_{\vec{u}} 2 \prec_{\vec{u}} 3 \prec_{\vec{u}} 4$, where $\vec{u} = \langle x_1, x_2, x_3, x_4 \rangle$

Although GE-trees can be constructed tractably for arbitrary value symmetries, we break symmetries of indistinguishable values using a number of value precedence constraints which are only linear or quadratic in size to the number of indistinguishable values, 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. Furthermore, the constraints can be made consistent with variable symmetry breaking constraints fairly easily, as shown in Theorem 4.2.

6.2 Comparison of Techniques

Each of the approaches has its benefits and drawbacks. Problem remodeling is a clean approach. The remodeled CSP is simply an ordinary CSP but with fewer symmetries. Therefore, it is easy to combine with other symmetry breaking techniques. However, there are no general methods or theorems to exploit this approach. It usually requires the knowledge of experienced human modelers. Moreover, the remodeled CSP can be a complicated and unnatural model, producing side effects such as difficult formulation of constraints and loss of solving efficiency.

Using symmetry breaking constraints can have extremely low overheads. For example, it is sometimes possible to express only a few symmetry breaking constraints to break a large number of symmetries. The approach can also deal with some common classes of symmetries effectively. An example is to use a polynomial number of lexicographic ordering constraints to order the symmetric rows and columns in matrix models of CSPs to tackle row and column symmetries [FFH+02], which are super-exponential in size. A drawback is that this approach sometimes cannot eliminate all symmetries. Luks and Roy [LR02, LR04, Roy01] study the computational complexity of generating the "lex-leader" symmetry breaking constraints [CGLR96]. They show that the number of essential constraints among the lex-leader constraints can be exponential. Therefore, the constraints proposed by Flener et al. [FFH+02] cannot break all the row and column symmetries. Another drawback is that symmetry breaking constraints may not respect the variable/value ordering heuristics in backtracking search algorithms. Among a class of symmetrically equivalent solutions of a CSP, the one which occurs earliest in the search tree according to the heuristics may be removed after adding symmetry breaking constraints. One which occurs later in the search tree may be retained instead. This drawback can possibly increase the search efforts required in first solution search.

The approach of modifying search algorithms never conflicts with variable/value ordering heuristics, since it always uses information of the first visited search state for pruning later symmetric part of the search tree. By using computational group theory systems, this approach can break all the symmetries using the symmetry group generators. However, the overhead of the approach can be high. For example, SBDS adds a constraint for each symmetry upon backtracking. As the number of symmetries is usually large or even super-exponential, the number of constraints to be added becomes a burden to the system. Besides, the time complexity of the dominance test function required by SBDD can be high, affecting the overall efficiency of SBDD.

In the heuristic method approach by Meseguer and Torras [MT99, MT01], an advantage is that the proposed heuristics are easy to combine with the popular smallest-domain-first variable ordering heuristic. Meseguer and Torras show that the combined heuristic is more efficient than using standalone heuristics. A drawback of the approach, however, is that when performing all solution search or a CSP has no solutions, backtracking search still has to revisit symmetrically equivalent states, and the heuristic methods cannot help reduce search space. Value pruning procedures have been developed to minimize the effects of this shortcoming.

The approach of building specialized search trees is a conceptual abstraction. For example, in the GE-tree approach [RDGKL04], any search tree contains a GE-tree. A minimal GE-tree is a search tree which breaks all the symmetries. Roney-Dougal et al. [RDGKL04] give a polynomial time algorithm for the construction of a GE-tree for arbitrary value symmetries, and hence show that value symmetries can be broken tractably. However, it is not clear whether the approach can be applied to arbitrary symmetries similarly.

There are other research area regarding symmetries in CSPs. McDonald and Smith [MS02] conduct a systematic study of partial symmetry breaking in CSPs. They show that there is a breakeven point between the amount of saved search efforts and broken symmetries. Therefore, breaking a “suitable” subset of all the symmetries in a CSP can result in better solving efficiency than breaking all the symmetries in the CSP. They discuss different methods of selecting symmetries to break and present a general algorithm for selecting subsets of symmetries so that breaking the selected subsets are close to the breakeven point.

Chapter 7

Concluding Remarks

We conclude the thesis in this chapter 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, and 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 modeling 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 using lexicographic ordering constraints and 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 generalized arc consistency and set bounds consistency respectively. The global constraints avoid the use of inefficient if-then constraints.

Sixth, we have given theoretical results to characterize the exact behavior of our proposed algorithms in different usage scenarios. For example, we have shown the consistent use of value precedence and variable symmetry breaking constraints using lexicographic ordering, and consistency levels achieved upon different use of value precedence on multiple indistinguishable values.

We have performed extensive experiments on various problems and confirm that our two proposed methods are both feasible and efficient. The experimental results support that the two methods are effective to general CSPs which contain value symmetries.

7.2 Discussions

Breaking value symmetries with symmetry breaking constraints is not an easy task. We propose two methods to tackle this problem. The first method makes use of multiple viewpoints and channeling constraints, and the second method uses global constraint implementations of the value precedence condition to break symmetries of indistinguishable values. One the one hand, the first 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 extra 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, so that the overall benefits are more than those of using either method alone.

Despite the efficiency of the symmetry breaking methods proposed, identifying symmetries in a CSP and formulating symmetry breaking constraints for them are still performed mainly by human modelers to date. It should be noted that it takes time for human to perform such CSP reformulations before enjoying the efficiency provided by the methods. However, once we obtain the symmetry breaking constraints for a CSP of one problem instance, those symmetry breaking constraints can usually be generalized and applied to CSPs of other similar problem instances at negligible cost. Therefore, the reusability of reformulation results makes the time invested in the reformulation process still worthwhile.

7.3 Future Work

Our work proposes a study of using constraints to break value symmetries. There is scope for future work.

First, the multiple viewpoints method inherits the benefits of redundant modeling. It would be interesting to study how the extra choice of search variables in multiple viewpoints affects the search efficiency.

Second, Theorem 3.7, 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 3.6, 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 3.7 to apply to arbitrary value symmetries also.

Third, we show in Theorem 4.7 that generalized arc consistency on integer value precedence on all pairs of indistinguishable values is strictly stronger than generalized arc consistency 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. Since the proof of the theorem relies on the use of a domain value which is *not* indistinguishable with other domain values, we conjecture that the two types of postings are equivalent in propagation when *all* the domain values in all the affected variables are indistinguishable.

Fourth, the global value precedence 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 constrained integer variables instead of simply integer constraints in our current implementations, so that the generalized constraints have additional usage besides breaking symmetries of indistinguishable values.

Fifth, our current implementations of the value precedence global constraints enforce generalized arc consistency and set bounds consistency respectively. It would be interesting to design propagation algorithms that enforce other local consistencies, such as bounds consistency for integer value precedence.

Sixth, we identify and design 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 can make symmetry breaking techniques more usable and popular to CSP modelers.

Seventh, in the benchmark of Steiner triple systems, we demonstrate 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.

Bibliography

[ASM03] 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. [96]

[BB02] 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. [89, 98, 99]

[BB05] N. Barnier and P. Brisset. Solving Kirkman's schoolgirl problem in a few seconds. *Constraints*, 10(1):7–21, 2005. [89, 98, 99]

[Ben94] B. Benhamou. Study of symmetry in constraint satisfaction problems. In *Proceedings of the 2nd Workshop on Principles and Practice of Constraint Programming*, 1994. [6, 28, 57]

[Bes94] C. Bessiere. Arc-consistency and arc-consistency again. *Artificial Intelligence*, 65(1):179–190, 1994. [17]

[BFH+05] 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. [83, 85]

[BFP88] C.A. Brown, L. Finkelstein, and P.W. Purdom Jr. Backtrack searching in the presence of symmetry. In *Proceedings of the 6th International Conference on Applied Algebra, Algebraic Algorithms, and Error Correcting Codes*, pages 99–110, 1988. [97]

[BFP96] C.A. Brown, L. Finkelstein, and P.W. Purdom Jr. Backtrack searching in the presence of symmetry. *Nordic Journal of Computing*, 3(3):203–219, 1996. [97]

[BFR99] C. Bessiere, E.C. Freuder, and J.-C. Régin. Using constraint metaknowledge to reduce arc consistency computation. *Artificial Intelligence*, 107(1):125–148, 1999. [17, 19]

[BP81] C.A. Brown and P.W. Purdom Jr. How to search efficiently. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, pages 588–594, 1981. [2, 12]

[BR01] C. Bessiere and J.-C. Régin. Refining the basic constraint propagation algorithm. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 309–315, 2001. [17]

[Bré79] D. Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979. [80]

[BRYZ05] C. Bessiere, J.-C. Régin, R.H.C. Yap, and Y. Zhang. An optimal coarse-grained arc consistency algorithm. *Artificial Intelligence*, 165(2):165–185, 2005. [17]

[BW99] 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. [97, 98]

[BW02] R. Backofen and S. Will. Excluding symmetries in constraint-based search. *Constraints*, 7:333–349, 2002. [97, 98]

[CB02a] M. Carlsson and N. Beldiceanu. Arc-consistency for a chain of lexicographic ordering constraints. Technical Report T2002-18, Swedish Institute of Computer Science, 2002. [5, 18, 25, 96]

[CB02b] M. Carlsson and N. Beldiceanu. Revisiting the lexicographic ordering constraint. Technical Report T2002-17, Swedish Institute of Computer Science, 2002. [5, 18, 25, 96]

[CCLW99] 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. [5, 30, 33, 35, 42]

[CGLR96] 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. [4, 26, 49, 96, 103]

[CLRS01] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, second edition, 2001. [11]

[CLS00] K.M.F. Choi, J.H.M. Lee, and P.J. Stuckey. A Lagrangian reconstruction of GENET. *Artificial Intelligence*, 123:1–39, 2000. [12, 22]

[CLS03a] C.W. Choi, J.H.M. Lee, and P.J. Stuckey. Propagation redundancy for permutation channels. In *Proceedings of the 18th Internation Joint Conference on Artificial Intelligence*, pages 1370–1371, 2003. [42]

[CLS03b] 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. [42]

[cos03] *CHIP V5 Reference Manual*, 2003. [3]

[DP87] D. Dechter and J. Pearl. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34:1–38, 1987. [2, 12]

[DTWZ94] A. Davenport, E. Tsang, C.J. Wang, and K. Zhu. GENET: A connectionist architecture for solving constraint satisfaction problems by iterative improvement. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 325–330, 1994. [12, 22]

[FFH$^+$01] 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. [5, 32]

[FFH+02] 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. [5, 24, 26, 30, 32, 42, 60, 80, 96, 97, 99, 103]

[FHK+02] 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. [18, 25, 65, 87, 96]

[FJM03] 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. [26, 97]

[FM01] 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. [97]

[FMK+03] 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. [61, 97]

[Fre91] E.C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Proceedings of the 9th National Conference on Artificial Intelligence*, pages 227–233, 1991. [27]

[FSS01] 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. [4, 24, 97, 98]

[gap05] *GAP 4.4.5 Reference Manual*, 2005. [98]

[Gas77] J. Gaschnig. A general backtracking algorithm that eliminates most redundant tests. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, page 457, 1977. [2, 12]

[GB65] S.W. Golomb and L.D. Baumert. Backtrack programming. *Journal of the ACM*, 12(4):516–524, 1965. [2, 12]

[Gee92] P.A. Geelen. Dual viewpoint heuristics for binary constraint satisfaction problems. In *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 31–35, 1992. [31]

[Gen01] I.P. Gent. A symmetry breaking constraint for indistinguishable values. In *Proceedings of the 1st International Workshop on Symmetry in Constraint Satisfaction Problems*, 2001. [6, 28, 57, 97]

[Ger95] C. Gervet. *Set Intervals in Constraint Logic Programming: Definition and implementation of a language*. PhD thesis, Université de Franche-Comté, 1995. [2, 9, 18, 19, 96]

[Ger97] C. Gervet. Interval propagation to reason about sets: Definition and implementation of a practical language. *Constraints*, 1(3):191–244, 1997. [2, 9, 18, 19, 96]

[GHK02] 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. [98]

[GHKL03] 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. [99]

[GS96] S.A. Grant and B.M. Smith. The phase transition behavior of maintaining arc consistency. In *Proceedings of the 12th European Conference on Artificial Intelligence*, pages 175–179, 1996. [19]

[GS00] 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. [4, 97, 98]

[GW99] I.P. Gent and T. Walsh. CSPLib: A benchmark library for constraints. In *Proceedings of the 5th International Conference on Principles and Practice of Constraint Programming*, pages 480–481, 1999. Available at `http://www.csplib.org/`. [23, 43]

[HE80] R.M. Haralick and G.L. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980. [19, 20]

[Hun74] T.W. Hungerford. *Algebra*. Springer-Verlag, 1974. [23]

[ilo99] *ILOG Solver 4.4 Reference Manual*, 1999. [3, 9, 64, 79, 81]

[Kir47] T.P. Kirkman. On a problem in combinatorics. *Cambridge and Dublin Mathematical Journal*, 2:191–204, 1847. [91]

[Kiz04] Z. Kiziltan. *Symmetry Breaking Ordering Constraints*. PhD thesis, Uppsala universitet, 2004. [26, 45]

[LL02] Y.C. Law and J.H.M. Lee. Model induction: a new source of CSP model redundancy. In *Proceedings of the 18th National Conference on Artificial Intelligence*, pages 54–60, 2002. [31]

[LL03] Y.C. Law and J.H.M. Lee. Expressing symmetry breaking constraints using multiple viewpoints and channeling constraints. In *Workshop Notes of the Third International Workshop on Symmetry in Constraint Satisfaction Problems*, pages 127–141, 2003. [5]

[LL04] 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. [6]

[LL05] 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. [5]

[LLW95] J.H.M. Lee, H.F. Leung, and H.W. Won. Extending GENET for non-binary CSP's. In *Proceedings of the 7th International Conference on Tools with Artificial Intelligence*, pages 338–342, 1995. [12, 22]

[LR02] E. M. Luks and A. Roy. Symmetry breaking in constraint satisfaction. In *Proceedings of the 7th International Symposium on Artificial Intelligence and Mathematics*, 2002. [103]

[LR04] E. M. Luks and A. Roy. The complexity of symmetry-breaking formulas. *Annals of Mathematics and Artificial Intelligence*, 41(1):19–45, 2004. [103]

[Mac77] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977. [1, 2, 14, 15, 16, 17]

[MH86] R. Mohr and T.C. Henderson. Arc and path consistency revised. *Artificial Intelligence*, 28(2):225–233, 1986. [17]

[MJPL92] S. Minton, M.D. Johnston, A.B. Philips, and P. Laird. Minimizing conflicts: A heuristic method for constraint-satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992. [22]

[MM88] R. Mohr and G. Masini. Good old discrete relaxation. In *Proceedings of the 8th European Conference on Artificial Intelligence*, pages 651–656, 1988. [2, 17, 18]

[Mon74] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Science*, 7(2):95–132, 1974. [2, 14, 15]

[MS98] K. Marriott and P.J. Stuckey. *Programming with Constraints*. The MIT Press, 1998. [2, 13]

[MS02] I. McDonald and B. Smith. Partial symmetry breaking. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, pages 431–445, 2002. [104]

[MT99] 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. [4, 99, 103]

[MT01] P. Meseguer and C. Torras. Exploiting symmetries within constraint satisfaction search. *Artificial Intelligence*, 129(1–2):133–163, 2001. [4, 99, 103]

[Nad89] B.A. Nadel. Constraint satisfaction algorithms. *Computational Intelligence*, 5:188–224, 1989. [2, 12]

[Per92] M. Perlin. Arc consistency for factorable relations. *Artificial Intelligence*, 53(2–3):329–342, 1992. [17]

[Pie68] W.P. Pierskalla. The multidimensional assignment problem. *Operations Research*, 16(2):422–431, 1968. [32]

[Pug93] 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. [4, 96]

[Pug02] J.-F. Puget. Symmetry breaking revisited. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, pages 446–461, 2002. [98]

[Pug03] 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. [99]

[Pug05] J.-F. Puget. Symmetry breaking revisited. *Constraints*, 10(1):23–46, 2005. [98]

[RDGKL04] 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. [4, 100, 104]

[Rég94] J.-C. Régin. A filtering algorithm for constraints of differences in CSPs. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 362–367, 1994. [18]

[Rég96] J.-C. Régin. Generalized arc-consistency for global cardinality constraint. In *Proceedings of the 13th National Conference on Artificial Intelligence*, pages 209–215, 1996. [18]

[Roy01] A. Roy. *Symmetry Breaking and Fault Tolerance in Boolean Satisfiability*. PhD thesis, University of Oregon, 2001. [103]

[SF94] D. Sabin and E.C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proceedings of the 2nd International Workshop on Principles and Practice of Constraint Programming*, volume 874, pages 10–20, 1994. [19]

[SG01] 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. [99]

[sic05] *SICStus Prolog 3.12.2 User's Manual*, 2005. [3]

[Smi01a] B.M. Smith. Reducing symmetry in a combinatorial design problem. In *Proceedings of the 3rd International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 2001. [4, 96]

[Smi01b] B.M. Smith. Reducing symmetry in a combinatorial design problem. In *Proceedings of the Workshop on Modelling and Solving Problems with Constraints*, 2001. [4, 96]

[VFPA03] 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. [4, 100, 101]

[VT96] C. Voudouris and E. Tsang. Partial constraint satisfaction problems and guided local search. In *Proceedings of the 2nd International Conference on Practical Application of Constraint Technology*, pages 337–356, 1996. [12, 22]