Journal of Artificial Intelligence Research 78 (2023) 1-35 Submitted 03/2023; published 09/2023

Exploiting Functional Constraints in Automatic Dominance Breaking for Constraint Optimization

Jimmy H. M. Lee JLEE@CSE.CUHK.EDU.HK
Department of Computer Science and Engineering,
The Chinese University of Hong Kong,
Shatin, N.T., Hong Kong

Allen Z. Zhong ALLEN.ZHONG@MONASH.EDU
Department of Data Science and Artificial Intelligence,
Faculty of Information Technology,
Monash University, Australia

Abstract

Dominance breaking is a powerful technique in improving the solving efficiency of Constraint Optimization Problems (COPs) by removing provably suboptimal solutions with additional constraints. While dominance breaking is effective in a range of practical problems, it is usually problem specific and requires human insights into problem structures to come up with correct dominance breaking constraints. Recently, a framework is proposed to generate nogood constraints automatically for dominance breaking, which formulates nogood generation as solving auxiliary Constraint Satisfaction Problems (CSPs). However, the framework uses a pattern matching approach to synthesize the auxiliary generation CSPs from the specific forms of objectives and constraints in target COPs, and is only applicable to a limited class of COPs.

This paper proposes a novel rewriting system to derive constraints for the auxiliary generation CSPs automatically from COPs with nested function calls, significantly generalizing the original framework. In particular, the rewriting system exploits functional constraints flattened from nested functions in a high-level modeling language. To generate more effective dominance breaking nogoods and derive more relaxed constraints in generation CSPs, we further characterize how to extend the system with rewriting rules exploiting function properties, such as monotonicity, commutativity, and associativity, for specific functional constraints. Experimentation shows significant runtime speedup using the dominance breaking nogoods generated by our proposed method. Studying patterns of generated nogoods also demonstrates that our proposal can reveal dominance relations in the literature and discover new dominance relations on problems with ineffective or no known dominance breaking constraints.

1. Introduction

Constraint Optimization Problems (COPs) are ubiquitous in computer science and artificial intelligence, and have applications in planning (Garrido, Onaindia, & Sapena, 2008; Booth, Tran, Nejat, & Beck, 2016), scheduling (Baptiste, Le Pape, & Nuijten, 2001), packing (Korf, 2004; Fukunaga & Korf, 2007; Korf, Moffitt, & Pollack, 2010), etc. A COP usually consists of an objective function to be minimized or maximized and constraints specifying compatible value assignments to variables. The goal of solving a COP is to find an *optimal solution*, i.e., an assignment of values to variables that satisfies all constraints of the COP and optimizes the objective. Constraint Programming (CP) (Rossi, Beek, & Walsh, 2006) is a classical paradigm for solving COPs. Users can state a COP formally and submit it to a CP solver, which will solve the problem using the Branch and Bound algorithm (Land & Doig, 1960) augmented with constraint propagation (Rossi et al., 2006).

Dominance breaking is a powerful technique to improve the efficiency of CP solvers in solving COPs and has successful applications in a range of problems (Aldowaisan, 2001; Getoor, Ottosson, Fromherz, & Carlson, 1997; Korf, 2004; Prestwich & Beck, 2004). The technique exploits *dominance relations* (Ibaraki, 1977) among assignments in a COP. If some assignments are proved to be *dominated by* others in a dominance relation of a COP, then they are guaranteed to be suboptimal with respect to the satisfiability of constraints and/or objective value. Such dominated assignments can be removed by adding *dominance breaking constraints* into the COP without changing its optimal objective value. However, applying dominance breaking demands sophisticated insights into the problem structures, and therefore it is usually problem-specific and non-trivial to transfer from one problem to another. Chu and Stuckey (2012) give a generic and manual method for deriving dominance breaking constraints for general COPs. Later, Mears and Garcia de la Banda (2015) automate the method to a large extent, but it still requires manual interventions to be effective.

This paper follows the work on automatic dominance breaking for a class of COPs (Lee & Zhong, 2020, 2023), which focuses on generating dominance breaking constraints in the form of *nogoods* (Katsirelos & Bacchus, 2005). In this framework, nogood generation for a COP is formulated as solving constraint satisfaction problems (CSPs) which can be constructed based on the types of objective and constraints in the original COP. For a class of *efficiently checkable* objectives and constraints in COPs, such as linear objectives and linear inequality constraints, the framework matches them to their corresponding constraints in the generation CSPs, which guarantee the generated nogoods to remove suboptimal assignments in the original COPs. As long as a COP consists of efficiently checkable objectives and constraints, the framework can construct CSPs for nogood generation mechanically and generate nogood constraints for dominance breaking automatically by using efficient constraint solvers to solve the CSPs. Yet, the original framework of automatic dominance breaking is restricted to COPs with only objectives and constraints that are all provably efficiently checkable. For example, in order to apply automatic dominance breaking to a COP, the objective function is required to be either additively separable or submodular in the original framework (Lee & Zhong, 2020, 2023). Later, the framework is generalized to allow a COP to contain some non-efficiently checkable constraints, but the technique is effective only when a relatively small number of variables are involved in these constraints (Lee & Zhong, 2021). The restriction on efficiently checkable objectives and constraints prevents the use of the framework for COPs with varying objectives and constraints, especially the ones with nested function calls.

Functional expressions are ubiquitous in problem modeling, while the objective and constraints with functional expressions are usually not efficiently checkable. In practice, however, COPs are usually specified in a high-level modeling language (Frisch, Harvey, Jefferson, Martinez-Hernandez, & Miguel, 2008; Nethercote, Stuckey, Becket, Brand, Duck, & Tack, 2007) and normalized/flattened into a form with only standard constraints.

Example 1. *Consider a simple COP as follows:*

$$\begin{aligned} &\textit{minimize } \max(z_1, z_2) + 4z_3 \\ &\textit{subject to } 2z_1 - 3z_2 * z_3 \leq 5, \\ &\qquad z_1, z_2, z_3 \in \{1, 2, 3\} \end{aligned} \tag{1}$$

The objective with the max *function and the constraint with the multiplying function are not efficiently checkable. After normalization, the COP can become:*

$$\begin{aligned} &\textit{minimize } obj \\ &\textit{subject to } y_2 \leq 5, obj = y_1 + 4z_3, y_1 = \max(z_1, z_2), \\ &\qquad y_2 = 2z_1 - 3y_3, y_3 = z_2 * z_3, \\ &\qquad z_1, z_2, z_3 \in \{1, 2, 3\}, y_1, y_2, y_3, obj \in \mathbb{Z} \end{aligned} \tag{2}$$

Note that y_1, y_2, y_3 and obj are newly introduced variables, and are functionally defined by $y_1 = \max(z_1, z_2)$, $y_2 = 2z_1 - 3y_3$, $y_3 = z_2 * z_3$ and $obj = y_1 + 4z_3$ respectively. We call these functional constraints, while $y_2 \leq 5$ is a non-functional constraint.

In this paper, we propose a method to exploit functional constraints to derive constraints in the generation CSPs, which further enhances the ability of the framework to generate dominance breaking nogoods for COPs with nested function calls. We first generalize the theory of dominance to normalized COPs which contain functionally defined variables and functional constraints. Based on the generalized theory, we formalize the automatic derivation of constraints as a formal rewriting system. When functions in COPs are treated as *uninterpreted functions* (Bryant, Lahiri, & Seshia, 2002), the rewriting system, which comprises the rules of replacement, binding, deletion and general decomposition, can handle COPs with arbitrary nested functions and return their corresponding constraints in nogood generation CSPs. If we further exploit function properties, such as monotonicity, commutativity and associativity, then the system can be extended with decomposition rewriting rules that derive more relaxed sufficient conditions for dominance and enable the generation of more dominance breaking nogoods for the target COPs. The proposed method is implemented on top of MiniZinc (Nethercote et al., 2007), an open-source constraint modeling language. Experimentation on various benchmarks confirms the superior efficiency of the generated nogoods to solve problems with ineffective or no known dominance breaking constraints in the literature. Even when nogoods are costly to generate, we give two case studies on the Steel Mill Slab Design Problem (Frisch, Miguel, & Walsh, 2001a) and the Balanced Academic Curriculum Problem (Castro & Manzano, 2001) to show how we can discover dominance relations and compact dominance (symmetry) breaking constraints by inspecting the nogood patterns of small instances.

The contributions of this work are multi-faceted. First, we further generalize the theory of dominance and formalize a rewriting system for automatic synthesis of generation CSPs. Second, we release a software tool to generate dominance breaking nogoods for MiniZinc, an open-source constraint modeling language. Third, we present extensive experiments to evaluate the effectiveness of the generated nogoods in solving COPs using three state-of-the-art solvers. Fourth, we demonstrate that the insights gained from dominance breaking nogoods can help to discover dominance relations and compact dominance (symmetry) breaking constraints. Compared with the preliminary version of this work that appeared in CP 2022 (Lee & Zhong, 2022), the current paper is improved in the following aspects:

- In Section 4, we formalize the derivation of constraints in generation CSPs as a formal rewriting system instead of giving an ad-hoc algorithm in the conference paper. We also prove the soundness and Church-Rosser properties of the rewriting system.
- We characterize the conditions of rewriting rules that guarantee the soundness and Church-Rosser properties of the rewriting system so that the system can be extended with more rules satisfying the condition and exploiting function properties, which allows the generation of more nogoods.
- We present in Table 1 the standard constraints implemented in our system and their associated function properties.
- In Section 5, we include more experimental results using different solvers and configurations to demonstrate the advantages of generated dominance breaking nogoods in solving COPs.
- In Section 6, we also include more detailed exposition on how to derive compact dominance breaking constraints from generated dominance breaking nogoods.

2. Preliminaries

A *variable* x is an unknown. A *domain* D maps each variable x to the finite set $D(x)$ which contains the possible values for x. An *assignment* θ *on a set of variables* $S = \{x_1, \ldots, x_k\}$ is a tuple $(v_1, \ldots, v_k) \in \mathcal{D}^S = D(x_1) \times \cdots \times D(x_k)$, where $v_j = \theta[x_j]$ is the value assigned to x_j in θ, and $S = var(\theta)$ is the *scope* of θ. We abuse notations to use $\theta[S']$ to denote the tuple formed by projecting $\theta \in \mathcal{D}^S$ onto a subset of variables $S' \subset S$. A *constraint* c is a subset of the Cartesian product $\mathcal{D}^S$ where $S = var(c)$ is the *scope* of c. An assignment θ *satisfies* a constraint c if $\theta[var(c)] \in c$, where $var(\theta) \supseteq var(c)$. We define a *nogood* $\neg\theta$ for an assignment θ to be a constraint of the form $\vee_{x \in var(\theta)}(x \neq \theta[x])$, and its *length* is always equal to the scope size $|var(\theta)|$.

A *Constraint Satisfaction Problem (CSP)* is a tuple (X, D, C) where X is a set of variables, D is a domain for X and C is a set of constraints. A *Constraint Optimization Problem (COP)* (X, D, C, obj) extends a CSP with an objective variable obj which is to be minimized. Let $\bar{\theta} \in \mathcal{D}^X$ denote a *full assignment* whose scope is X. A *solution* of a COP/CSP P is a full assignment $\bar{\theta} \in \mathcal{D}^X$ such that $\bar{\theta}$ satisfies all constraints $c \in C$. We let $sol(P) \subseteq \mathcal{D}^X$ denote the set of all solutions of P. The goal of solving a COP is to find an *optimal solution* $\bar{\theta}^* \in sol(P)$ such that $\bar{\theta}^*[obj] \leq \bar{\theta}[obj]$ for all solutions $\bar{\theta} \in sol(P)$, and $\bar{\theta}^*[obj]$ is the *optimal value* of P.

Following Chu and Stuckey (Chu & Stuckey, 2012), a dominance relation for a COP P is defined as a relation over the set of all full assignments of P.

Definition 1. *(Chu & Stuckey, 2012) A* dominance relation $\prec$ over $\mathcal{D}^X$ *(Chu, Banda, & Stuckey, 2010) is a transitive and irreflexive relation such that* $\forall \bar{\theta}, \bar{\theta}' \in \mathcal{D}^X$*, if* $\bar{\theta} \prec \bar{\theta}'$ *with respect to* P*, then either:*

1. $\bar{\theta}$ is a solution of P and $\bar{\theta}'$ is not a solution of P, or

2. both $\bar{\theta}$ and $\bar{\theta}'$ are solutions of P and $\bar{\theta}[obj] \leq \bar{\theta}'[obj]$, or

3. both $\bar{\theta}$ and $\bar{\theta}'$ are not solutions of P and $\bar{\theta}[obj] \leq \bar{\theta}'[obj]$.

In this case, we say that $\bar{\theta}$ dominates $\bar{\theta}'$ with respect to P.

A full assignment that is dominated by another with respect to P is subordinate to another full assignment concerning satisfiability and/or objective value. Therefore, a dominated full assignment can be removed without changing the optimal objective value of P (Chu & Stuckey, 2012). Dominance relations can be generalized (Lee & Zhong, 2020, 2023) to assignments over $\mathcal{D}^S$ where $S \subseteq X$. Let $\mathcal{D}^X_\theta = \{\bar{\theta} \in \mathcal{D}^X \mid \bar{\theta}[var(\theta)] = \theta\}$ be a subset of $\mathcal{D}^X$. We say that θ *dominates* θ' with respect to P iff $\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \exists \bar{\theta} \in \mathcal{D}^X_\theta$ such that $\bar{\theta} \prec \bar{\theta}'$ for some dominance relation $\prec$ with respect to P. When the context is clear, we abuse notations and let $\theta \prec \theta'$ denote θ dominates θ'.

Theorem 1. *(Lee & Zhong, 2020, 2023) Suppose $\theta, \theta' \in \mathcal{D}^S$ are assignments of $P = (X, D, C, obj)$ where $S \subseteq X$. If $\theta \prec \theta'$ with respect to P, then removing all assignments in $\mathcal{D}^X_{\theta'}$ preserves the same satisfiability and optimal value of P.*

Removing all dominated full assignments in $\mathcal{D}^X_{\theta'}$ only requires adding a nogood $\neg\theta'$ to P. While generating all dominance breaking nogoods is impractical, Lee and Zhong (2020, 2023) formulate it as constraint satisfaction to identify and exploit only a subset of such nogoods of length up to a predetermined threshold L. The high-level algorithm is as follows:

1. Choose a maximum length L for dominance breaking nogoods.
2. For each scope $S \subseteq X$ where $|S| \in \{1, \ldots, L\}$,
   (a) Synthesize *generation CSPs* to search for pairs (θ, θ') of assignments over S, whose constraints imply that $\theta \prec \theta'$ with respect to P.
   (b) Solve all solutions of the generation CSPs.
   (c) Collect the derived nogoods from the solutions (one nogood from each solution).
3. Add all the collected nogoods to the COP before problem-solving.

The size and the complexity of generation CSPs depend on the scope size $|S|$. While the number of dominance breaking nogoods and their collective pruning power increases with larger L, the time of solving generation CSPs also increases. The threshold L is usually relatively small compared to the number of variables $|X|$ so that nogoods can be generated by solving a group of small and easy CSPs.

The key step is to derive constraints in the generation CSPs automatically as sufficient conditions for dominance relations. Lee and Zhong (2020, 2023) give such sufficient conditions or constraints in the generation CSPs directly based on the objective and constraint types. The method requires developers to include certain types of objectives and constraints and define the corresponding sufficient conditions in the system. The method can only be applied to a class of COPs consisting of *efficiently checkable* objectives and constraints. However, it is not easily extensible especially when there are nested function calls as shown in Example 1. To tackle this problem, we generalize the theory of dominance in Section 3 and present a method for automatic sufficient condition derivation in Section 4.

3. Functional Constraints and Dominance

In this paper, we assume that *a COP $P = (X, D, C, obj)$ is the result of applying some sort of flattening procedure*, such as the one used in the MiniZinc compiler (Leo, 2018) and similar to (2) in Example 1, to a problem model. Therefore, constraints with nested function calls are flattened into a set C_Y of functional constraints which are of the form $y = h(x_1, \ldots, x_k)$, where $h : \mathbb{R}^k \mapsto \mathbb{R}$ is a k-ary function. Any assignment on variables $\{x_1, \ldots, x_k\}$ corresponds to a unique assignment on y, so we say that *y is defined by* a functional constraint $y = h(x_1, \ldots, x_k)$. Our proposed method utilizes the functional constraints and the properties of functions to derive sufficient conditions for dominance. Note that a high-level COP can be flattened into different forms depending on the standard constraints provided by the underlying solvers. To be independent of the underlying solvers, dominance breaking nogoods generated by our methods are output as text and appended to the original non-flattened problem model.

Before presenting the generalized theory of dominance, we give the following example to demonstrate how to derive sufficient conditions over a pair of assignments such that one assignment dominates another.

Example 2. *Consider the COP in (2) and $\theta, \theta' \in \mathcal{D}^S$ where $S = \{z_1, z_2\}$. Suppose we restrict $\mathcal{D}^X$ to be the set of full assignments that satisfy all functional constraints in (2). Therefore, values of y_1, y_2, y_3 and obj in a valid full assignment are determined by the functional constraints that define them. Our aim is to find constraints over θ and θ' that implies θ dominates θ' with respect to P.*

For notational convenience, we let $\bar{\theta} \in \mathcal{D}^X_\theta$ denote a corresponding full assignment for each full assignment $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$ such that $\bar{\theta}[z_3] = \bar{\theta}'[z_3]$. In other words, we can obtain $\bar{\theta}$ from $\bar{\theta}'$ by mutating values of z_1, z_2 and z_3 assigned by θ' to those assigned by θ. Following Lee and Zhong (2020, 2023), if we can show that

- *betterment: $\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \bar{\theta}[obj] \leq \bar{\theta}'[obj]$,*
- *implied satisfaction: $\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \bar{\theta}[y_2] \leq \bar{\theta}'[y_2]$, and*
- *not-equal: $\theta \neq \theta'$, i.e., $\theta[z_1] \neq \theta'[z_1] \vee \theta[z_2] \neq \theta'[z_2]$,*

then we can construct a relation $\prec$ over $\mathcal{D}^X$ such that $\bar{\theta} \prec \bar{\theta}'$ for all $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$. If so, one can verify that such a relation will be a dominance relation with respect to (2) by Definition 1, and thus all full assignments in $\mathcal{D}^X_{\theta'}$ can be removed without changing the optimal value. Note that not-equal is a constraint over θ and θ', while betterment and implied satisfaction are quantified inequalities, for which we need to find quantifier-free sufficient conditions as constraints over θ and θ'.

We can find such constraints over θ and θ' for betterment as follows:

- *Variable obj is defined by $obj = y_1 + 4z_3$. If we have*

$$\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \bar{\theta}[y_1] + 4\bar{\theta}[z_3] \leq \bar{\theta}'[y_1] + 4\bar{\theta}'[z_3], \tag{3}$$

  then betterment must hold since $\bar{\theta}$ and $\bar{\theta}'$ both satisfy all functional constraints.

- *Variable y_1 is defined by $y_1 = \max(z_1, z_2)$. It suffices to show that*

$$\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \max(\bar{\theta}[z_1], \bar{\theta}[z_2]) + \bar{\theta}[z_3] \leq \max(\bar{\theta}'[z_1], \bar{\theta}'[z_2]) + \bar{\theta}'[z_3]. \tag{4}$$

- *The summation function is monotonically increasing, (4) must be true if we have*

$$\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \max(\bar{\theta}[z_1], \bar{\theta}[z_2]) \leq \max(\bar{\theta}'[z_1], \bar{\theta}'[z_2]) \wedge \bar{\theta}[z_3] \leq \bar{\theta}'[z_3] \tag{5}$$

- *Inequality ($\bar{\theta}[z_3] \leq \bar{\theta}'[z_3]$) must hold since $\bar{\theta}[z_3] = \bar{\theta}'[z_3]$ for all $\bar{\theta} \in \mathcal{D}^X_{\theta}$ and $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$. Further, since $\bar{\theta}[z_1] = \theta[z_1]$, $\bar{\theta}[z_2] = \theta[z_2]$, $\bar{\theta}'[z_1] = \theta'[z_1]$, and $\bar{\theta}'[z_2] = \theta'[z_2]$, the condition (5) is equivalent to*

$$\max(\theta[z_1], \theta[z_2]) \leq \max(\theta'[z_1], \theta'[z_2]). \tag{6}$$

  Thus, if θ and θ' satisfy (6), the betterment condition must hold.

Similarly, we can find such constraints for implied satisfaction as follows:

- *Variable y_2 is defined by $y_2 = 2z_1 - 3y_3$, the implied satisfaction must be true if*

$$\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, 2\bar{\theta}[z_1] - 3\bar{\theta}[y_3] \leq 2\bar{\theta}'[z_1] - 3\bar{\theta}'[y_3]. \tag{7}$$

- *Since the result of subtraction increases with the increasing of the minuend and the decreasing of the subtrahend, (7) must be true if*

$$\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, 2\bar{\theta}[z_1] \leq 2\bar{\theta}'[z_1] \wedge 3\bar{\theta}[y_3] \geq 3\bar{\theta}'[y_3]. \tag{8}$$

- *Variable y_3 is defined by $y_3 = z_2 * z_3$. Since $z_2, z_3 \geq 0$, $3\bar{\theta}[y_3] \geq 3\bar{\theta}'[y_3]$ must hold if*

$$\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \bar{\theta}[z_2] \geq \bar{\theta}'[z_2] \wedge \bar{\theta}[z_3] \geq \bar{\theta}'[z_3]. \tag{9}$$

  Since $\bar{\theta}[z_3] = \bar{\theta}'[z_3]$, (8) must hold if

$$\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, 2\bar{\theta}[z_1] \leq 2\bar{\theta}'[z_1] \wedge \bar{\theta}[z_2] \geq \bar{\theta}'[z_2]. \tag{10}$$

- *By definitions, we have $\bar{\theta}[z_1] = \theta[z_1]$, $\bar{\theta}[z_2] = \theta[z_2]$, $\bar{\theta}'[z_1] = \theta'[z_1]$, and $\bar{\theta}'[z_2] = \theta'[z_2]$, and therefore (7) and (8) must hold if*

$$\theta[z_1] \leq \theta'[z_1] \wedge \theta[z_2] \geq \theta'[z_2] \tag{11}$$

In other words, if θ and θ' fulfill (6), (11) and $\theta \neq \theta'$, then it suffices to show that θ dominates θ' with respect to the COP in (2). By solving (6), (11) and $\theta \neq \theta'$ as a constraint satisfaction problem, we can obtain one possible solution pair (θ, θ') where $\theta = \{z_1 = 1, z_2 = 2\}$ and $\theta' = \{z_1 = 2, z_2 = 1\}$, and the constraint $\neg\theta' \equiv (z_1 \neq 2 \vee z_2 \neq 1)$ is a dominance breaking nogood in (2). Similar derivation can also be applied to pairs of assignments over other scopes to obtain more dominance breaking nogoods.

Example 2 exemplifies the procedure to derive sufficient conditions for betterment and implied satisfaction considering the functional constraints. We will establish a formal rewriting system for automatic derivation of such sufficient conditions. In the following, we first generalize the theory of dominance to facilitate our presentation.

Recall that a normalized COP consists of functional and non-functional constraints. For simplicity, we associate each non-functional constraint $c \in (C \setminus C_Y)$ with a *reified variable* $b \in \{0, 1\}$, where $\bar{\theta} \in \mathcal{D}^X$ satisfies c if and only if $\bar{\theta}[b] = 1$. In other words, we treat each constraint $c \in (C \setminus C_Y)$ as a function returning 0/1 and define a (reified) functional constraint $c_b \equiv (b = c(x_{i_1}, \ldots, x_{i_k}))$. If $\bar{\theta}[b] \geq \bar{\theta}'[b]$ for two full assignments $\bar{\theta}$ and $\bar{\theta}'$, then $\bar{\theta}'$ satisfies c implies that $\bar{\theta}$ also satisfies c. We let C_B denote the set of (reified) functional constraints and B denote the set of reified variables.

Without loss of generality, let (Z, Y, B) and (C_B, C_Y) be partitions of variables X and constraints C respectively in a normalized COP, where $Z \cup Y \cup B = X$, $C_B \cup C_Y = C$ and $obj \in Y$. Note that Z, Y, B are pairwise disjoint and $C_B \cap C_Y = \emptyset$. In case a variable $y \in Y$ is introduced by the flattening procedure, we set the domain for y to be the largest possible set. Note that when there is no flattening and reification, our definition of a COP degenerates to the classical definition (Rossi et al., 2006).

To exploit functional constraints in normalized COPs, the following definition characterizes a key property of full assignments.

Definition 2. *Let $P = (X, D, C, obj)$ be a normalized COP where (Z, Y, B) and (C_B, C_Y) be a partition of variables X and constraints C respectively. A full assignment $\bar{\theta} \in \mathcal{D}^X$ is* functionally valid *if and only if*

- *$\bar{\theta}[b] = c(\bar{\theta}[x_{i_1}], \ldots, \bar{\theta}[x_{i_k}])$ for a reified constraint $(b = c(x_{i_1}, \ldots, x_{i_k})) \in C_B$, and*
- *$\bar{\theta}[y] = h(\bar{\theta}[x_{i_1}], \ldots, \bar{\theta}[x_{i_k}])$ for a functional constraint $(y = h(x_{i_1}, \ldots, x_{i_k})) \in C_Y$*

Note that when $\bar{\theta}$ in a normalized COP is functionally valid, it corresponds to a full assignment in the original non-flattened problem model. The value for a variable $y \in Y$ (respectively $b \in B$) in a functionally valid full assignment can be computed from $c_y \in C_Y$ (respectively $c_b \in C_B$) as well as values for variables in Z. Since variables in $Y \cup B$ can be auxiliary variables introduced during the flattening process and may not appear in the original problem model, we only focus on generating nogoods involving variables in Z.

In the remainder of the paper, we assume that $P = (X, D, C, obj)$ is a normalized COP and consider only functionally valid full assignments in $\mathcal{D}^X$. Our aim is to find sufficient conditions for a pair of assignments θ and θ' over $S \subseteq Z$ such that $\theta \prec \theta'$ with respect to P. Recall that $\theta \prec \theta'$ requires $\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \exists \bar{\theta} \in \mathcal{D}^X_{\theta}$ such that $\bar{\theta} \prec \bar{\theta}'$ for some dominance relation over $\mathcal{D}^X$. It is expensive to check whether *there exists* $\bar{\theta}$ that dominates $\bar{\theta}'$ for each $\bar{\theta}'$ in $\mathcal{D}^X_{\theta'}$. Instead, we check only whether a specific $\bar{\theta}$ dominates $\bar{\theta}'$ by utilizing a *mutation mapping* for two assignments θ and θ' over the same scope.

Definition 3. *The* mutation mapping *$\mu^{\theta' \to \theta}$ for two assignments $\theta, \theta' \in \mathcal{D}^S$ over a scope $S \subseteq Z$ maps a full assignment $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$ to another full assignment $\bar{\theta} \in \mathcal{D}^X_{\theta}$ such that:*

- *$\bar{\theta}[z] = \theta[z]$ for $z \in var(\theta)$,*
- *$\bar{\theta}[z] = \bar{\theta}'[z]$ for $z \in Z \setminus var(\theta)$,*

- *$\bar{\theta}[y] = h(\bar{\theta}[x_{i_1}], \ldots, \bar{\theta}[x_{i_k}])$ where $y \in Y$ is defined by $y = h(x_{i_1}, \ldots, x_{i_k}) \in C_Y$,*
- *$\bar{\theta}[b] = c(\bar{\theta}[x_{i_1}], \ldots, \bar{\theta}[x_{i_k}])$ where $b \in B$ is defined by $b = c(x_{i_1}, \ldots, x_{i_k}) \in C_B$.*

In other words, $\mu^{\theta' \to \theta}$ "mutates" the θ' component of $\bar{\theta}'$ to become θ and assigns the computed values to variables in $Y \cup B$ accordingly. The following proposition characterizes some useful properties of the mutation mapping in Definition 3.

Proposition 1. *Let $\mu^{\theta' \to \theta}$ be a mutation mapping for two assignments $\theta, \theta' \in \mathcal{D}^S$ over a scope $S \subseteq Z$. The followings are always true for functionally valid full assignments $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$, where $\bar{\theta} = \mu^{\theta' \to \theta}(\bar{\theta}')$:*

- *If $z \in S$, then $\bar{\theta}[z] = \theta[z]$ and $\bar{\theta}'[z] = \theta'[z]$.*
- *If $z \in Z \setminus S$, then $\bar{\theta}[z] = \bar{\theta}'[z]$.*

With the mutation mapping, the following result gives a sufficient condition governing when a partial assignment θ dominates another θ' with respect to P.

Theorem 2. *If a pair of assignments $\theta, \theta' \in \mathcal{D}^S$ satisfies:*

- *empty intersection: $\mathcal{D}^X_{\theta} \cap \mathcal{D}^X_{\theta'} = \emptyset$,*
- *betterment: $\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \mu^{\theta' \to \theta}(\bar{\theta}')[obj] \leq \bar{\theta}'[obj]$, and*
- *implied satisfaction: $\forall b \in B, \forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}$, $\mu^{\theta' \to \theta}(\bar{\theta}')[b] \geq \bar{\theta}'[b]$,*

then θ dominates θ' with respect to P.

Proof. We construct a relation $\prec$ over $\mathcal{D}^X$ such that $\prec = \{(\mu^{\theta' \to \theta}(\bar{\theta}'), \bar{\theta}') \mid \bar{\theta}' \in \mathcal{D}^X_{\theta'}\}$. Since $\mathcal{D}^X_{\theta}$ and $\mathcal{D}^X_{\theta'}$ have empty intersection, the relation $\prec$ is trivially transitive and irreflexive. By betterment and implied satisfaction, one can check $\bar{\theta}$ and $\bar{\theta}'$ must satisfy one of the three conditions of a dominance relation. Therefore, θ dominates θ' with respect to P. □

Theorems 1 and 2 imply that $\neg\theta'$ is a *dominance breaking nogood* that can remove all dominated solutions in $\mathcal{D}^X_{\theta'}$ without changing the optimal value of P. The empty intersection is trivially satisfied if $\theta \neq \theta'$. In order to show that a partial assignment θ' is dominated by another θ using Theorem 2, what remains is to find constraints over θ and θ' that are sufficient conditions for the empty intersection, the betterment and the implied satisfaction conditions. Since empty intersection can be easily modeled by $\vee_{x_i \in S} \theta[x_i] \neq \theta'[x_i]$, we will focus on finding sufficient conditions for the betterment and implied satisfaction in Section 4.

Recall that we enumerate all dominance breaking nogoods $\neg\theta'$ derived from pairs (θ, θ') of partial assignments that are solutions of generation CSPs. It is necessary to ensure that all nogoods are *compatible* in the sense that not all optimal solutions of P are eliminated. To ensure the compatibility of all derived nogoods, a lexicographical ordering constraint between θ and θ' is a sufficient condition to ensure the compatibility of generated nogoods. Given two tuples $\theta = (v_1, \ldots, v_k), \theta' = (v'_1, \ldots, v'_k) \in \mathcal{D}^S$ where $S \subseteq Z$, we say that θ is *lexicographically smaller than* θ', denoted by $\theta <_{lex} \theta'$, if and only if $\exists i \in \{1, \ldots, n\}$ such that $v_i < v'_i$ and $\forall j < i, v_j = v'_j$. If $\theta <_{lex} \theta'$, and the pair (θ, θ') satisfies betterment and implied satisfaction, then we can preserve the *lexicographically smallest optimal solution*, which is an optimal solution of P and is lexicographically smallest among all optimal solutions of P (Lee & Zhong, 2020, 2023).

The above definitions and results degenerate to those in the original framework (Lee & Zhong, 2020, 2023) when Y and C_Y are empty.

4. Automatic Sufficient Condition Derivation

In this section, we present a rewriting system that derives automatically useful sufficient conditions for betterment and implied satisfaction over a pair of assignments $\theta, \theta' \in \mathcal{D}^S$, where $S \subseteq Z$ in a COP. When it is clear from the context, we let $\bar{\theta} = \mu^{\theta' \to \theta}(\bar{\theta}') \in \mathcal{D}^X_{\theta}$ denote the image of $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$ by the mutation mapping in Definition 3.

Note that both betterment and implied satisfaction in Theorem 2 are predicates requiring an inequality to hold for all $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$. The aim of our rewriting system is to derive constraints over θ and θ' without quantifiers as sufficient conditions for the quantified inequalities. We first present a general rewriting system (Definition 4) which only utilizes functional constraints and properties in Proposition 1 to find such sufficient conditions. While this rewriting system is generic, the derived sufficient conditions are sometimes too restricted. Next, we discuss how the generic rewriting system can be extended with rules that derive more relaxed sufficient conditions. In particular, we give examples of rewriting rules that exploit common functional properties such as monotonicity (Definition 5), associativity and commutativity (Definitions 6). With more relaxed sufficient conditions and constraints in generation CSPs, more useful nogoods can be generated for dominance breaking by solving generation CSPs.

4.1 A Rewriting System for Uninterpreted Functions

To formalize the derivation of sufficient conditions, we adopt the inductive definition of *terms* (Baader & Nipkow, 1998) from rewriting systems as follows:

- a variable or a constant is a term, and
- if f is a k-ary function and $t_1, \ldots, t_k$ are terms, then $f(t_1, \ldots, t_k)$ is a term.

By abusing notations, we also define the notation $var(t)$ for a term as follows:

- if t is a constant, then $var(t) = \emptyset$;
- if t is a variable x, then $var(t) = \{x\}$;
- if t is a function term $f(t_1, \ldots, t_k)$, then $var(t) = \cup_i var(t_i)$;

Note that f can either be the constraint c in a reified constraint $b = c(x_{i_1}, \ldots, x_{i_k})$ or the function h in a functional constraint $y = h(x_{i_1}, \ldots, x_{i_k})$. A term t is *fixed in* θ if and only if $x \in var(\theta)$ for all variables x in t; otherwise t is *free*.

A *substitution* is a finite mapping from variables to terms which assigns to each variable x a term t different from x. We write a substitution as $\beta = \{x_{i_1}/t_1, \ldots, x_{i_k}/t_k\}$ where $x_{i_1}, \ldots, x_{i_k}$ are different variables, $t_1, \ldots, t_k$ are terms and $\forall j \in \{1, \ldots, k\}, x_{i_j} \notin var(t_j)$. A substitution β can be *applied to a term* t to obtain $t\beta$ by replacing every occurrence of variable x_{i_j} in $var(t)$ by the term t_j for all $j \in \{1, \dots, k\}$. An assignment θ can be treated as a special substitution that replaces each occurrence of a variable $x \in var(\theta)$ with $\theta[x]$.

Let $\lhd$ be a binary comparison operator in $\{\leq, \geq, =\}$. The betterment and the implied satisfaction conditions in Theorem 2 are predicates in the form $(\forall \bar{\theta}' \in \mathcal{D}^X_{\bar{\theta}'}, t\bar{\theta} \lhd t\bar{\theta}')$, where $t\bar{\theta}$ and $t\bar{\theta}'$ are obtained by substituting each variable in $var(t)$ with its values in $\bar{\theta}$ and $\bar{\theta}'$ respectively. The derivation in Example 2 recursively rewrites the conjunction of quantified inequalities until all variables in the inequalities are fixed in θ and θ'. Therefore, our rewriting system maintains two sets of predicates Q and F where Q is the set of predicates that has to be further rewritten, and F is the set of predicates with all variables in S. We write a rewriting rule as $(Q \cup \{p\}, F) \rightsquigarrow (Q \cup Q', F \cup F')$, in which a predicate p is rewritten into a set Q' of quantified inequalities and a set F' of condition over θ and θ' by a rewriting rule. Note that Q' and F' are sometimes empty.

Our first rewriting system operates without considering the semantics of functions and treats them as uninterpreted functions. These functions are required only to fulfill *functional consistency*, meaning that a function should produce the same output when given the same input arguments. The generic rewriting system comprises four fundamental rules: (1) *Replacement*, which involves substituting a defined variable with its defining function; (2) *Binding*, which removes quantifiers and moving predicates from Q to F whenever feasible; (3) *Deletion*, which disregards predicates on non-interesting variables; and (4) *General Decomposition*, which decomposes a predicate to ones requiring all arguments to have the same values in both assignments. This rewriting system can be applied to predicates with arbitrary functions. The formal definition is as follows.

Definition 4. *Given a normalized COP $P = (Z \cup Y \cup B, D, C_B \cup C_Y, obj)$ and a scope $S \subseteq Z$. The general rewriting system is initialized with the pair $(Q, \{\})$, where $Q = \{(\forall \bar{\theta}' \in \mathcal{D}^X_{\bar{\theta}'}, \bar{\theta}[obj] \leq \bar{\theta}'[obj])\} \cup \{(\forall \bar{\theta}' \in \mathcal{D}^X_{\bar{\theta}'}, \bar{\theta}[b] \geq \bar{\theta}'[b]) \mid b \in B\}$, and applies the following rewriting rules to a predicate $(\forall \bar{\theta}' \in \mathcal{D}^X_{\bar{\theta}'}, t\bar{\theta} \lhd t\bar{\theta}') \in Q$ until Q is empty:*

- *Replacement: if $var(t) \cap (Y \cup B) \neq \emptyset$, then*

$$(Q \cup \{\forall \bar{\theta}' \in \mathcal{D}^X_{\bar{\theta}'}, t\bar{\theta} \lhd t\bar{\theta}'\}, F) \rightsquigarrow (Q \cup \{\forall \bar{\theta}' \in \mathcal{D}^X_{\bar{\theta}'}, t\beta\bar{\theta} \lhd t\beta\bar{\theta}'\}, F),$$

  where $\beta = \{x/f(x_{i_1}, \dots, x_{i_k})\}$ and $x \in var(t) \cap (Y \cup B)$ is defined by $x = f(x_{i_1}, \dots, x_{i_k})$.

- *Binding: if $var(t) \subseteq S \subseteq Z$, then*

$$(Q \cup \{\forall \bar{\theta}' \in \mathcal{D}^X_{\bar{\theta}'}, t\bar{\theta} \lhd t\bar{\theta}'\}, F) \rightsquigarrow (Q, F \cup \{t\theta \lhd t\theta'\}).$$

- *Deletion: if $var(t) \subseteq (Z \setminus S)$, then*

$$(Q \cup \{\forall \bar{\theta}' \in \mathcal{D}^X_{\bar{\theta}'}, t\bar{\theta} \lhd t\bar{\theta}'\}, F) \rightsquigarrow (Q, F).$$

- *General decomposition: if $var(t) \subseteq Z$, $var(t) \cap S \neq \emptyset$ and $var(t) \cap (Z \setminus S) \neq \emptyset$, then*

$$\begin{aligned}
&(Q \cup \{(\forall \bar{\theta}' \in \mathcal{D}^X_{\bar{\theta}'}, t\bar{\theta} \lhd t\bar{\theta}')\}, F) \\
\equiv &(Q \cup \{(\forall \bar{\theta}' \in \mathcal{D}^X_{\bar{\theta}'}, f(t_1\bar{\theta}, \dots, t_k\bar{\theta}) \lhd f(t_1\bar{\theta}', \dots, t_k\bar{\theta}'))\}, F) \\
\rightsquigarrow &(Q \cup \{(\forall \bar{\theta}' \in \mathcal{D}^X_{\bar{\theta}'}, t_i\bar{\theta} = t_i\bar{\theta}') \mid \forall i \in \{1, \dots, k\}\}, F).
\end{aligned}$$

Note that the applicability of rewriting rules in Definition 4 depends on the set $var(t)$ of variables. The conditions are *mutually exclusive* and *exhaustive*, which means that there must be one applicable rewriting rule for each predicate in Q. In particular, when we apply the general decomposition rule, $var(t)$ is a subset of Z and $var(t)$ has a non-empty intersection with both S and $Z \setminus S$, and the term t must contain at least two variables and be a function term of the form $f(t_1, \ldots, t_k)$.

We can check easily the Church-Rosser property of the rewriting system.

Theorem 3. *The rewriting system in Definition 4 has the Church-Rosser property.*

Proof. Since the rewriting rules are mutually exclusive and exhaustive depending on the set $var(t)$ of variables, there is only one applicable rewriting rule to each predicate $p \in Q$. Therefore, the rewriting rules are pairwise commutative. □

The more important property is that the conjunction of all predicates in $Q \cup F$, denoted as $Q \wedge F$, is always a sufficient condition for the conjunction of betterment and implied satisfaction for a COP P.

Theorem 4. *The rewriting rules in Definition 4 preserves the invariant that $Q \wedge F$ is always a sufficient condition for the betterment and the implied satisfaction of P.*

Proof. Since Q is initialized with predicates of the betterment and the implied satisfaction conditions, the statement holds automatically. By induction, it suffices to show that $Q \wedge F$ is still a sufficient condition after applying each rewriting rule:

- Replacement: the predicate $\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, t\bar{\theta} \lhd t\bar{\theta}'$ is equivalent to $\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, (t\beta)\bar{\theta} \lhd (t\beta)\bar{\theta}'$, because all full assignments are all functionally valid by Definition 2.
- Binding: by Proposition 1, since all variables in $var(t)$ also belong to $S \subseteq Z$, we have $\bar{\theta}[x] = \theta[x]$ and $\bar{\theta}'[x] = \theta'[x]$ for all $x \in var(t)$. The predicate $(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, t\bar{\theta} \lhd t\bar{\theta}')$ is equivalent to $(t\theta \lhd t\theta')$.
- Deletion: by Proposition 1 again, when $x \in Z$ and $x \notin S$, we have $\bar{\theta}[x] = \bar{\theta}'[x]$. Therefore, the predicate $t\bar{\theta} = t\bar{\theta}'$ must hold and imply that $t\bar{\theta} \leq t\bar{\theta}'$ and $t\bar{\theta} \geq t\bar{\theta}'$.
- General decomposition: since $f(t_1\bar{\theta}', \ldots, t_k\bar{\theta}')$ is a functional or reified constraint, the conjunction $\wedge_{i=1}^{k}(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, t_i\bar{\theta} = t_i\bar{\theta}')$ implies $(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, f(t_1\bar{\theta}, \ldots, t_k\bar{\theta}) \lhd f(t_1\bar{\theta}', \ldots, t_k\bar{\theta}'))$ by the property of functional consistency.

Therefore, the invariant is preserved by the rewriting system in Definition 4. □

Note that $(Q, F) \rightsquigarrow (Q', F')$ by replacement, binding and deletion are equivalent transformations such that $(Q \wedge F) \Leftrightarrow (Q' \wedge F')$, while $(Q' \wedge F')$ implies $(Q \wedge F)$ after applying general decomposition. By Theorem 4, the execution of the rewriting system can recursively derive the sufficient condition for a predicate in Q until either the predicate becomes a trivial statement or it is transformed into constraints over the pair (θ, θ') due to the mutation mapping. What remains is to show the termination of rewriting.

Theorem 5. *The rewriting system in Definition 4 always terminates, and Q must be empty when the rewriting system terminates.*

Proof. Without loss of generality, we assume that each variable $y \in Y$ appears only in at most one constraint other than the functional constraint $y = h(x_{i_1}, \ldots, x_{i_k})$ that defines y; otherwise, an additional variable y' defined by $y' = h(x_{i_1}, \ldots, x_{i_k})$ can be introduced to replace each extra occurrence of y. By definition of a COP, $Y \cup B$ and $C_Y \cup C_B$ are finite sets. We maintain three natural numbers:

- n_1: the number of variables in $Y \cup B$ that have not been substituted in replacement,
- n_2: the number of occurrences of function symbols in Q, and
- n_3: the sum of $|var(t)|$ for all predicates $(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, t\bar{\theta} \lhd t\bar{\theta}') \in Q$.

We claim that applying each rewriting rule reduces the triple (n_1, n_2, n_3) in a lexicographic sense. Each variable $x \in Y \cup B$ is only substituted when x is in the flattened constraint or the reified constraint, replacement must decrease n_1 by 1. General decomposition decreases n_2 while keeping n_1 unchanged. Further, binding and deletion remove one predicate $(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, t\bar{\theta} \lhd t\bar{\theta}')$ from Q and therefore decrease n_3 by $var(t)$. The termination follows directly from the fact that there is no infinite descending sequence of triples of natural numbers. Rewriting rules in Definition 4 are exhaustive, and therefore Q must be empty when no rules are applicable and the rewriting system terminates. □

We say that the rewriting system is *sound* with respect to a COP P if it always terminates and $Q \wedge F$ is a sufficient condition for the betterment and implied satisfaction of P. The following corollary is a direct consequence of Theorems 4 and 5.

Corollary 1. *The rewriting system in Definition 4 is sound with respect to a COP P.*

By Theorem 3 and Corollary 1, the general rewriting system will always give the same set of predicates after finite steps of rewriting with respect to the scope $S \subseteq Z$.

4.2 Exploiting Function Properties

When applying the general decomposition rule to a predicate of the form $p \equiv (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, f(t_1\bar{\theta}, \ldots, t_k\bar{\theta}) \lhd f(t_1\bar{\theta}', \ldots, t_k\bar{\theta}'))$, the rewriting system in Definition 4 always treat the function f as an uninterpreted function without any properties. This may sometimes result in sufficient conditions that are too restrictive.

Example 3. *Consider Example 2 again. If we apply the general decomposition rule to (4), the resulting set of predicates will become*

$$\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \max(\bar{\theta}[z_1], \bar{\theta}[z_2]) = \max(\bar{\theta}'[z_1], \bar{\theta}'[z_2]) \wedge \bar{\theta}[z_3] = \bar{\theta}'[z_3]. \tag{12}$$

Applying the deletion and binding rules again, we will get the sufficient condition

$$\max(\theta[z_1], \theta[z_2]) = \max(\theta'[z_1], \theta'[z_2]) \tag{13}$$

for betterment. Similarly, we obtain $\theta[z_1] = \theta'[z_1], \theta[z_2] = \theta'[z_2]$ as the sufficient condition for implied satisfaction. Together with the empty intersection condition in Theorem 2, the generation CSP will become unsatisfiable. No solution can be found by solving such a generation CSP, and no nogoods can be generated.

In Example 2, the success in deriving a dominance breaking nogood relies on the fact that the derivation also exploits function properties when deriving sufficient conditions in each step. For instance, when deriving (5) from (4), we also consider the fact that the max function is monotonically increasing. Though both (5) and (12) imply (4), the former is more relaxed than the latter. The more relaxed sufficient conditions in the generation CSPs, the more dominance breaking nogoods are generated by solving the CSPs. In this section, we extend the rewriting system in Definition 4 with additional decomposition rules that exploit common and useful function properties as we will explain in the following subsections. The extended rewriting system only applies the general rewriting rule when no other decomposition rules can be applied for a predicate to be rewritten.

Before presenting concrete decomposition rules, we first characterize the conditions of rules that can guarantee the soundness and the Church-Rosser properties of the rewriting system. Let a decomposition rule be of the form $(Q \cup \{p\}, F) \rightsquigarrow (Q \cup Q', F \cup F')$, where Q' and F' are the sets of resulting predicates from the rewriting of a predicate p. We say that a decomposition rule is

- *diminishing* if the number of function symbols in Q' decreases while there are no new variables introduced in predicates of Q' after rewriting, and
- *invariant-preserving* if $Q' \wedge F'$ is a sufficient condition of p.

We have the following theorem for the soundness of the extended rewriting system.

Theorem 6. *If the rewriting system in Definition 4 is extended with decomposition rules that are both invariant-preserving and diminishing, then the extended system is sound with respect to a COP P.*

Proof. The soundness property of the rewriting system requires the system to terminate and preserve the invariant that $Q \wedge F$ is a sufficient condition for the betterment and the implied satisfaction conditions of a COP P. The proof of preservation of the invariant is essentially the same as that of Theorem 4. Since all decomposition rules are diminishing, we can prove the termination in the same way as the proof of Theorem 5. □

As for the Church-Rosser property, Theorem 3 requires that all rewriting rules in the system are mutually exclusive, while multiple rewriting rules can be applied to the same predicate after introducing additional decomposition rules into the system. Therefore, we define the precedence over additional decomposition rules as follows. We say that a predicate p_1 is *weaker than* another predicate p_2 if and only if p_2 implies p_1, and p_2 is *stronger than* p_1. Suppose that two decomposition rules are both applicable to a predicate p, we say that a decomposition rule *subsumes* another if the conjunction of the resulting predicates of the former is weaker than that of the latter. In the extended rewriting system, we adopt a conservative approach to always apply a rewriting rule that subsumes others to retain the Church-Rosser property.

Proposition 2. *The extended rewriting system has the Church-Rosser property.*

Proof. In the extended rewriting system, all decomposition rules are still mutually exclusive due to the rule precedence. It is also exhaustive since we always apply the general decomposition rule when no special function properties can be exploited. □

In the following, we give several decomposition rules catering for common and useful function properties such as monotonicity, associativity and commutativity for deriving weaker sufficient conditions. For each introduced decomposition rule, we prove its diminishing and invariant-preserving properties. We also analyze the subsumption relation among decomposition rules that can be applied to the same predicate, so that the extended rewriting system always applies the one returning the weakest possible sufficient conditions for the betterment and the implied satisfaction conditions.

4.2.1 Decomposition Rules for Monotonic Functions

The first property of interest is *monotonicity.* A function $f : \mathbb{R}^k \mapsto \mathbb{R}$ is *monotonically increasing* if

$$(\forall i, a_i \leq b_i) \Rightarrow f(a_1, \ldots, a_k) \leq f(b_1, \ldots, b_k)$$

and is *monotonically decreasing* if

$$(\forall i, a_i \geq b_i) \Rightarrow f(a_1, \ldots, a_k) \leq f(b_1, \ldots, b_k)$$

where $a_i, b_i \in \mathbb{R}$. For the ease of presentation, we say that the *reverse operators* of $\leq$, $\geq$ and $=$ are $\geq$, $\leq$ and $=$ respectively. When the function f is monotonically increasing or decreasing, the decomposition rule can be relaxed, resulting in predicates that contain relaxed inequalities which are easier to satisfy.

Definition 5. *Suppose* $p \equiv (\forall \bar{\theta}' \in \mathcal{D}^X_{\bar{\theta}'}, f(t_1\bar{\theta}, \ldots, t_k\bar{\theta}) \lhd f(t_1\bar{\theta}', \ldots, t_k\bar{\theta}')) \in Q$ *is a predicate in the rewriting system, and* $\rhd$ *is the reverse operators of* $\lhd$.

- *Increasing decomposition: if* f *is monotonically increasing, then*

$$(Q \cup \{p\}, F) \rightsquigarrow (Q \cup \{(\forall \bar{\theta}' \in \mathcal{D}^X_{\bar{\theta}'}, t_i\bar{\theta} \lhd t_i\bar{\theta}') \mid \forall i = 1, \ldots, k\}, F)$$

- *Decreasing decomposition: if* f *is monotonically decreasing, then*

$$(Q \cup \{p\}, F) \rightsquigarrow (Q \cup \{(\forall \bar{\theta}' \in \mathcal{D}^X_{\bar{\theta}'}, t_i\bar{\theta} \rhd t_i\bar{\theta}') \mid \forall i = 1, \ldots, k\}, F)$$

We can easily check the following properties.

Theorem 7. *The increasing decomposition and the decreasing decomposition rules in Definition 5 are invariant-preserving and diminishing.*

Proof. The rules are invariant-preserving by the definitions of monotonically increasing and monotonically decreasing functions and the fact that all full assignments are functionally valid. The function symbol f is removed, and therefore the rules are also diminishing. □

Theorem 8. *The increasing and decreasing decomposition rules in Definition 5 both subsume the general decomposition rule in Definition 4.*

Proof. Let $p \equiv (\forall \bar{\theta}' \in \mathcal{D}^X_{\bar{\theta}'}, f(t_1\bar{\theta}, \ldots, t_k\bar{\theta}) \lhd f(t_1\bar{\theta}', \ldots, t_k\bar{\theta}'))$ be a predicate in the rewriting system. Suppose that $(Q \cup \{p\}, F) \rightsquigarrow (Q \cup Q', F)$ by the increasing decomposition (respectively the decreasing decomposition), while $(Q \cup \{p\}, F) \rightsquigarrow (Q \cup Q'', F)$ by the general decomposition. By the fact that each predicate $(\forall \bar{\theta}' \in \mathcal{D}^X_{\bar{\theta}'}, t_i\bar{\theta} = t_i\bar{\theta}')$ must always be a sufficient condition for both $(\forall \bar{\theta}' \in \mathcal{D}^X_{\bar{\theta}'}, t_i\bar{\theta} \lhd t_i\bar{\theta}')$ and $(\forall \bar{\theta}' \in \mathcal{D}^X_{\bar{\theta}'}, t_i\bar{\theta} \rhd t_i\bar{\theta}')$, the conjunction of predicates in Q' must be weaker than that of Q''. □

By Theorems 7 and 8, the increasing decomposition and the decreasing decomposition rules can be included in the rewriting system, and they have higher precedence than the general decomposition when f is monotonically increasing or monotonically decreasing.

4.2.2 Decomposition Rules for Associative and Commutative Functions

We only consider fixed-arity functions so far, but we can generalize them and define a *variadic function* to be a mapping $f : \cup_{k\in\mathbb{N}}\mathbb{R}^k \mapsto \mathbb{R}$ which can take an arbitrary non-zero number of arguments. In this section, we show that associativity and commutativity of variadic functions can make the decomposition rules in Definitions 4 and 5 obtain even weaker sufficient conditions.

To facilitate the presentation, we use a special notation to denote a variadic function. Let $\mathbf{t} = \langle t_1, \dots, t_k\rangle$, $\mathbf{t}_1 = \langle t_1, \dots, t_j\rangle$ and $\mathbf{t}_2 = \langle t_{j+1}, \dots, t_k\rangle$ be vectors of terms, where $1 \leq j \leq k$. Using these notations, the followings denote the same function call: $f(t_1, \dots, t_k)$, $f(\mathbf{t})$ and $f(\mathbf{t}_1, \mathbf{t}_2)$. A variadic function is *associative* if $f(t) = t$ and $f(\mathbf{t}_1, \mathbf{t}_2) = f(f(\mathbf{t}_1), \mathbf{t}_2)$, and is *commutative* $f(t_1, \dots, t_k) = f(t_{\pi(1)}, \dots, t_{\pi(k)})$ where π is a permutation over $\{1, \dots, k\}$, i.e., a bijection from the set to itself. Common variadic functions, such as summation, maximum, and minimum, are usually associative and commutative. By commutativity, a function has the following useful property.

Proposition 3. *Let f be a commutative function and $\theta \in \mathcal{D}^S$ be an assignment where $S \subseteq Z$. If there are $j \geq 1$ fixed terms among $t_1, \dots, t_k$, then we can always find a permutation π over $\{1, \dots, k\}$ such that*

$$\forall\bar{\theta} \in \mathcal{D}^X_\theta, f(t_1\bar{\theta}, \dots, t_k\bar{\theta}) = f(t_{\pi(1)}\bar{\theta}, \dots, t_{\pi(j)}\bar{\theta}, t_{\pi(j+1)}\bar{\theta}, \dots, t_{\pi(k)}\bar{\theta}),$$

where $t_{\pi(1)}, \dots, t_{\pi(j)}$ are all fixed in θ, while $t_{\pi(j+1)}, \dots, t_{\pi(k)}$ are free terms.

The proof directly follows the definition of commutativity. In other words, we can always find a permutation for arguments of a variadic and commutative so that all fixed terms are clustered. If the function f in the general decomposition rule is a commutative and associative variadic function, we can use a more flexible decomposition rule that combines partial information from fixed terms and requires all other unfixed terms to have identical values in both assignments.

Definition 6. *Suppose $p \equiv (\forall\bar{\theta}' \in \mathcal{D}^X_{\theta'}, f(t_1\bar{\theta}, \dots, t_k\bar{\theta}) \lhd f(t_1\bar{\theta}', \dots, t_k\bar{\theta}'))$ is a predicate where there are $j \geq 1$ fixed terms among $t_1, \dots, t_k$. Let π be a permutation over $\{1, \dots, k\}$ such that $\mathbf{t}_1 = \langle t_{\pi(1)}, \dots, t_{\pi(j)}\rangle$ and $\mathbf{t}_2 = \langle t_{\pi(j+1)}, \dots, t_{\pi(k)}\rangle$ consist of all fixed terms and free terms respectively.*

- *General Decomposition with Aggregation: if f is commutative and associative, then*

$$\begin{aligned}&(Q \cup \{p\}, F)\\ \rightsquigarrow&(Q \cup \{(\forall\bar{\theta}' \in \mathcal{D}^X_{\theta'}, t_{\pi(i)}\bar{\theta} = t_{\pi(i)}\bar{\theta}') \mid \forall i = j+1, \dots, k\}, F \cup \{f(\mathbf{t}_1)\theta = f(\mathbf{t}_1)\theta'\}).\end{aligned}$$

The followings show that the new decomposition rule in Definition 6 can replace the general decomposition rule in Definition 4 when the function is commutative and associative.

Theorem 9. *The general decomposition with aggregation rule in Definition 6 is invariant-preserving and diminishing.*

Proof. It is straightforward to see that the rule is diminishing, while the invariant-preserving property is proved as follows:

$$
\begin{aligned}
&(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, f(t_1\bar{\theta}, \ldots, t_k\bar{\theta}) \lhd f(t_1\bar{\theta}', \ldots, t_k\bar{\theta}'))\\
\Leftrightarrow&(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, f(\mathbf{t_1}\bar{\theta}, \mathbf{t_2}\bar{\theta}) \lhd f(\mathbf{t_1}\bar{\theta}', \mathbf{t_2}\bar{\theta}'))\\
\Leftrightarrow&(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, f(f(\mathbf{t_1})\bar{\theta}, \mathbf{t_2}\bar{\theta}) \lhd f(f(\mathbf{t_1})\bar{\theta}', \mathbf{t_2}\bar{\theta}'))\\
\Leftarrow&(\bigwedge_{i=j+1}^{k} (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, t_{\pi(i)}\bar{\theta} = t_{\pi(i)}\bar{\theta})) \wedge (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, f(\mathbf{t_1})\bar{\theta} = f(\mathbf{t_1})\bar{\theta}')\\
\Leftrightarrow&(\bigwedge_{i=j+1}^{k} (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, t_{\pi(i)}\bar{\theta} = t_{\pi(i)}\bar{\theta})) \wedge (f(\mathbf{t_1})\theta = f(\mathbf{t_1})\theta')
\end{aligned}
$$

The second and the third steps are due to the commutativity and the associativity of f respectively, while the fourth step holds because all full assignments are functionally valid. By Proposition 1, the last step holds because $\bar{\theta}[x] = \theta[x]$ and $\bar{\theta}'[x] = \theta'[x]$ for all $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$ and $\bar{\theta} = \mu^{\theta' \to \theta}(\bar{\theta}')$. □

Theorem 10. *The general decomposition with aggregation rule in Definition 6 subsumes the general decomposition rule in Definition 4.*

Proof. Let $p \equiv (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, f(t_1\bar{\theta}, \ldots, t_k\bar{\theta}) \lhd f(t_1\bar{\theta}', \ldots, t_k\bar{\theta}'))$ be a predicate to be rewritten, where f is a commutative and associative varadic function. Suppose that $(Q \cup \{p\}, F) \rightsquigarrow (Q \cup Q', F \cup F')$ by the general decomposition with aggregation rule and $(Q \cup \{p\}, F) \rightsquigarrow (Q \cup Q'', F)$ by the general decomposition rule. After applying the general decomposition rule, the conjunction of predicates in Q'' is

$$
\begin{aligned}
&\bigwedge_{i=1}^{k} (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, t_i\bar{\theta} = t_i\bar{\theta}')\\
\Leftrightarrow&(\bigwedge_{i=1}^{j} (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, t_{\pi(i)}\bar{\theta} = t_{\pi(i)}\bar{\theta})) \wedge (\bigwedge_{i=j+1}^{k} (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, t_{\pi(i)}\bar{\theta} = t_{\pi(i)}\bar{\theta}))\\
\Rightarrow&(\bigwedge_{i=1}^{j} (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, t_{\pi(i)}\bar{\theta} = t_{\pi(i)}\bar{\theta})) \wedge (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, f(\mathbf{t_1})\bar{\theta} = f(\mathbf{t_1})\bar{\theta}')\\
\Leftrightarrow&(\bigwedge_{i=1}^{j} (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, t_{\pi(i)}\bar{\theta} = t_{\pi(i)}\bar{\theta})) \wedge (f(\mathbf{t}_1)\theta = f(\mathbf{t}_1)\theta')
\end{aligned}
$$

where π is as defined in Definition 6. The third step holds because all full assignments are functionally valid, and the last step is by Proposition 1. Therefore, the conjunction of predicates in $Q' \wedge F'$ is weaker than that for Q''. □

We can also have similar relaxed decomposition rules for monotonically increasing or decreasing functions, in which they first aggregate the partial information from fixed terms and then applies increasing or decreasing decomposition from Definition 5.

Definition 7. *Suppose* $p \equiv (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, f(t_1\bar{\theta}, \ldots, t_k\bar{\theta}) \lhd f(t_1\bar{\theta}', \ldots, t_k\bar{\theta}'))$ *is a predicate in the rewriting system such that there are* $j \geq 1$ *fixed terms among* $t_1, \ldots, t_k$*. Let* π *be a permutation over* $\{1, \ldots, k\}$ *such that* $\mathbf{t}_1 = \langle t_{\pi(1)}, \ldots, t_{\pi(j)} \rangle$ *and* $\mathbf{t}_2 = \langle t_{\pi(j+1)}, \ldots, t_{\pi(k)} \rangle$ *consist of all fixed terms and free terms respectively.*

- *Increasing Decomposition with Aggregation: if* f *is monotonically increasing, commutative and associative, then* $(Q \cup \{p\}, F)$ *is rewritten into*

$$(Q \cup \{(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, t_{\pi(i)}\bar{\theta} \lhd t_{\pi(i)}\bar{\theta}') \mid \forall = j+1, \ldots, k\}, F \cup \{f(\mathbf{t}_1)\theta \lhd f(\mathbf{t}_1)\theta'\}).$$

- *Decreasing Decomposition with Aggregation: if* f *is monotonically decreasing, commutative and associative, then* $(Q \cup \{p\}, F)$ *is rewritten into*

$$(Q \cup \{(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, t_{\pi(i)}\bar{\theta} \rhd t_{\pi(i)}\bar{\theta}') \mid \forall i = j+1, \ldots, k\}, F \cup \{f(\mathbf{t}_1)\theta \rhd f(\mathbf{t}_1)\theta'\}).$$

Similar to Theorems 9 and 10, we have the following results for the rules of increasing/decreasing decomposition with aggregation.

Theorem 11. *The rules of increasing decomposition with aggregation and decreasing decomposition with aggregation in Definition 7 are invariant-preserving and diminishing.*

Theorem 12. *The increasing (respectively decreasing) decomposition with aggregation in Definition 7 subsumes the increasing (respectively decreasing) decomposition in Definition 5.*

The following example shows the advantages of decomposition with aggregation.

Example 4. *Suppose we want to find sufficient conditions for* θ *and* θ' *where* $var(\theta) = var(\theta') = \{z_1, z_3\}$*. Let* $(Q \cup \{p\}, F)$ *be the pair of the rewriting system where*

$$p \equiv (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \min(\bar{\theta}[z_1], \bar{\theta}[z_2], \bar{\theta}[z_3]) \leq \min(\bar{\theta}'[z_1], \bar{\theta}'[z_2], \bar{\theta}'[z_3])). \tag{14}$$

If we apply increasing decomposition directly to (14), we get

$$(Q \cup \{(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \bar{\theta}[z_i] \leq \bar{\theta}'[z_i]) \mid i = 1, 2, 3\}, F) \tag{15}$$

Since the min *function is commutative and associative, we can obtain*

$$(Q \cup \{(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \min(\bar{\theta}[z_1], \bar{\theta}[z_3]) \leq \min(\bar{\theta}'[z_1], \bar{\theta}'[z_3])), (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \bar{\theta}[z_2] \leq \bar{\theta}'[z_2])\}, F) \tag{16}$$

by applying the increasing decomposition with aggregation to (14).

After applying binding and deletion in Definition 4 to (15) and (16) respectively, we obtain $\theta[z_1] \leq \theta'[z_1] \wedge \theta[z_3] \leq \theta'[z_3]$ *and* $\min(\theta[z_1], \theta[z_3]) \leq \min(\theta'[z_1], \theta'[z_3])$ *as sufficient conditions for (14). The former sufficient condition is stronger than the latter one.*

Constraint	Defines	Arguments	Mono.	Com. & Asso.
$y = \sum_{i=1}^{n} x_i - d_0$	y	$x_1, \dots, x_n$	Increasing	Yes
$y = \max(x_1, \dots, x_n)$	y	$x_1, \dots, x_n$	Increasing	Yes
$y = \min(x_1, \dots, x_n)$	y	$x_1, \dots, x_n$	Increasing	Yes
$y = \prod_{i=1}^{n} x_i$ where $D(x_i) \subseteq \mathbb{Z}_+$	y	$x_1, \dots, x_n$	Increasing	Yes
$y = element([d_1, \dots, d_n], x)$	y	x	No	No
$y = abs(x)$	y	x	No	No
$y = bool2int(b)$	y	b	Increasing	No
$b_0 \leftrightarrow (x = y)$	b_0	x, y	No	No
$b_0 \leftrightarrow (x \neq y)$	b_0	x, y	No	No
$b_0 \leftrightarrow (x \leq d)$	b_0	x	Decreasing	No
$y = w \cdot x$ where $w \in \mathbb{R}_{\geq 0}$	y	x	Increasing	No
$y = w \cdot x$ where $w \in \mathbb{R}_{<0}$	y	x	Decreasing	No
$b_0 \leftrightarrow and(b_1, \dots, b_n)$	b_0	$b_1, \dots, b_n$	Increasing	Yes
$b_0 \leftrightarrow or(b_1, \dots, b_n)$	b_0	$b_1, \dots, b_n$	Increasing	Yes
$b_0 \leftrightarrow xor(b_1, \dots, b_n)$	b_0	$b_1, \dots, b_n$	No	Yes
$b_0 \leftrightarrow \neg b_1$	b_0	b_1	Decreasing	Yes

Table 1: Common constraints for detecting dominance relations

By Theorems 10 and 12, we can always aggregate partial information from fixed terms to derive weaker sufficient conditions when the variadic function f in a predicate is both commutative and associative.

Table 1 summarizes common constraints in a normalized COP and the properties of functions that can be used for deriving weaker sufficient conditions. For each type of standard constraint, we give the functionally defined variable, arguments, its monotonicity property ("Mono."), and whether the function is commutative and associative ("Com. & Asso."). To facilitate the generation of nogoods, a global constraint can be represented as a conjunction of standard constraints in Table 1 when constructing generation CSPs.

Example 5. *The $alldifferent(z_1, \dots, z_k)$ constraint (Régin, 1994) is a global constraint that enforces a set of variables taking distinct values. To apply the rewriting system, the constraint is first reified into $b_0 \leftrightarrow alldifferent(z_1, \dots, z_k)$. Let $\tilde{D} \subseteq \bigcup_{j=1}^{k} D(z_j)$ be the set of values that appear in the domains of at least two variables in $\{z_1, \dots, z_k\}$ and $d = |\tilde{D}|$. We can compile the constraint into a set of standard constraints:*

- $\bigwedge_{v_i \in \tilde{D}} \bigwedge_{j=1}^{k} (b_{jv_i} \leftrightarrow (z_j = v_i) \wedge y_{jv_i} = bool2int(b_{jv_i}))$,
- $\bigwedge_{v_i \in \tilde{D}} (y_{0v_i} = sum(y_{1v_i}, \dots, y_{kv_i}) \wedge b_{v_i} \leftrightarrow (y_{0v_i} \leq 1))$, *and*
- $b_0 \leftrightarrow and(b_{v_1}, \dots, b_{v_d})$,

where $y_{0v_i}, y_{jv_i}, b_{jv_i}$ and b_v are introduced variables defined by standard functional constraints that enjoy the properties of monotonicity, commutativity and associativity. Therefore, the decomposition rules in Definitions 5 and 6 can be applied to derive sufficient conditions for the implied satisfaction condition of the alldifferent constraint.

Other global constraints, such as global cardinality constraint (Régin, 1996; Oplobedu, Marcovitch, & Tourbier, 1989) and bin packing constraints (Shaw, 2004), can also be supported similarly. Note that global constraints are treated as a conjunction of standard constraints only in synthesizing generation CSPs, and are untouched in problem-solving.

5. Experimental Evaluation

In this section, we report the experimental results on various optimization problems to show the utility of our proposed rewriting system in generating dominance breaking nogoods. We use MiniZinc (Nethercote et al., 2007) as the high-level modeling language and implement our nogood generation method by modifying the publicly available MiniZinc compiler with version 2.6.2[1]. In a compiled model, we treat constraints with the annotation "`defines_var`" as functional constraints, while others are non-functional constraints that should be reified. The generated nogoods for each problem are output as text and then appended to the original MiniZinc models before problem-solving.

The augmented models are submitted to MiniZinc using the Chuffed solver 0.10.4 (Ohrimenko, Stuckey, & Codish, 2009), the CP-SAT solver in OR-Tools v9.6 (Perron & Furnon, 2023), and the Gecode solver 6.3.0 (Schulte, Lagerkvist, & Tack, 2009). The first two are state-of-the-art hybrid solvers with lazy clause generation, and the last one is a well-established pure CP solver. There are totally five different configurations for problem-solving:

- Chuffed(User): using the Chuffed solver with the user-specified search heuristic.
- Chuffed(VSIDS): using the Chuffed solver with the Variable State Independent Decaying Sum search heuristic.
- CP-SAT(User): using the CP-SAT solver with the user-specified search heuristic.
- CP-SAT(Auto): using the CP-SAT solver with the underlying SAT solver's heuristics.
- Gecode(User): using the Gecode solver with the user-specified search heuristic.

Note that our method aims to analyze a user-defined model with nested functions, not necessarily that of the best model. All experiments are run on Quad Xeon Platinum 8268 2.90GHz processors.

There are six benchmark problems including talent scheduling problem (Cheng, Diamond, & Lin, 1993), warehouse location problem (Van Hentenryck, 1999), team assignment problem, budgeted maximum coverage problem (Khuller, Moss, & Naor, 1999), partial cover problem (Kearns, 1990) and sensor placement problem (Krause, Leskovec, Guestrin, VanBriesen, & Faloutsos, 2008). For all benchmarks, we attempt to generate all dominance breaking nogoods of length up to L (**L-dom**), and compare our method to the basic problem model (**basic**) and the model with manual dominance breaking constraints (**manual**) whenever they are available. The timeout for the whole solving process (nogood generation + problem-solving) is set to 7200 seconds, while we reserve at most 3600 seconds for nogood

1. We modify the embedded Geas solver and use the free search option for solving the generation CSPs. Our implementations are available at https://github.com/AllenZzw/auto-dom.

Config.	Group	basic		manual		2-dom		3-dom		4-dom	
		#s	#t	#s	#t	#s	#t	#s	#t	#s	#t
Chuffed (User)	$n = 16$	19	935.07	0	–	20	892.55	**20**	**484.66**	20	2142.00
	$n = 18$	2	6441.28	0	–	2	6497.44	**9**	**4830.68**	5	6897.56
	$n = 20$	0	–	0	–	0	–	1	**6880.22**	0	–
	$n = 22$	0	–	0	–	0	–	0	–	0	–
Chuffed (VSIDS)	$n = 16$	20	213.00	7	6235.85	20	216.32	**20**	**171.36**	20	1785.49
	$n = 18$	18	1628.28	0	–	18	1751.84	**20**	**824.58**	16	5414.10
	$n = 20$	11	5115.84	0	–	11	5086.09	**15**	**2950.20**	6	6757.51
	$n = 22$	0	–	0	–	0	–	0	–	0	–
CP-SAT (User)	$n = 16$	19	2213.05	0	–	18	2191.73	**20**	**365.56**	20	1326.33
	$n = 18$	0	–	0	–	0	–	12	4769.87	**15**	**5886.23**
	$n = 20$	0	–	0	–	0	–	0	–	0	–
	$n = 22$	0	–	0	–	0	–	0	–	0	–
CP-SAT (Auto)	$n = 16$	18	829.80	1	7062.29	18	853.75	**20**	**186.56**	20	1257.97
	$n = 18$	1	6876.40	0	–	1	6828.59	10	3856.81	**15**	**5498.32**
	$n = 20$	0	–	0	–	0	–	**1**	**6751.72**	1	7122.93
	$n = 22$	0	–	0	–	0	–	0	–	0	–
Gecode (User)	$n = 16$	20	372.66	0	–	**20**	**369.46**	2	7074.7	0	–
	$n = 18$	**9**	**6006.68**	0	–	8	5950.27	0	–	0	–
	$n = 20$	0	–	0	–	0	–	0	–	0	–
	$n = 22$	0	–	0	–	0	–	0	–	0	–

Table 2: Comparison of different methods in the talent scheduling problem

generation and use the *remaining* time for problem-solving in L-**dom**. If nogood generation times out in 3600 seconds, we augment the problem model with all nogoods generated before the timeout. Otherwise, the remaining time will be used for problem-solving.

The standard benchmark for optimization problems includes several instance groups of various problem sizes, with each group consisting of 20 random instances. For each method under different configurations, we report the number of solved instances (#s) and the geometric mean of the total time (#t) in seconds. If an instance times out, we use 7200 seconds as its total time to compute the geometric mean of the corresponding instance group, and an entry of ”–” indicates that all instances timed out in an instance group. We compare different methods by first comparing the number of solved instances and then by the geometric mean of the total time. We highlight the best method in bold for each instance group in each configuration. It is worth noting that the original framework of automatic dominance breaking (2020, 2023) is inapplicable to all benchmarks due to nested function calls in either the objective or constraints.

5.1 Talent Scheduling Problem

The Talent Scheduling Problem (Cheng et al., 1993) is prob039 in CSPLib (Gent & Walsh, 1999), which is to place n scenes and minimize the total costs for a set of actors. Each actor appears in several scenes and is paid a fixed cost per day if they are present. All actors need to be present on location from the first scene they are in till the last scene they are in. The problem is modelled as a sequencing problem using one variable for each scene. The objective is a weighted sum of several min/max functions. The manual dominance

Config.	Instance	**basic** #t	**manual** #t	**2-dom** #t	**3-dom** #t	**4-dom** #t
Chuffed (VSIDS)	film118	**561.49**	–	563.35	732.56	4882.80
	film103	1386.03	–	1526.29	**1182.97**	5852.37
	film119	2076.49	–	1127.79	**1314.97**	5048.47
	film116	5298.20	–	2534.79	**1503.89**	6064.68
	film105	4015.22	–	4993.56	**1643.53**	5528.25
	film117	–	–	7150.62	**1803.31**	6158.02
	film114	–	–	–	–	–
	MobStory	–	–	–	–	–

Table 3: Comparison of different methods for instances from Smith (2005)

breaking constraints for **manual** are by Chu and Stuckey (Chu & Stuckey, 2012), which exclude solutions where the total cost decreases by swapping positions of two scenes.

Table 2 shows the results for standard benchmark for $n \in \{16, 18, 20, 22\}$ in different search configurations. We note that the total solving time of **manual** is even larger than that of **basic**. Expressing manual dominance breaking in the MiniZinc model requires additional variables and introduces overheads for propagation. Chu and Stuckey (Chu & Stuckey, 2012) implement the manual dominance breaking constraints in Chuffed, which requires sophisticated and bespoke techniques to reduce the overhead. The generated nogoods by our method only involve variables in the original model, and they can be posted in the MiniZinc model without modifying the backend solver. Overall, the generated nogoods can help to improve the number of solved instances or reduce the mean total time in all configurations except for using the Gecode solver. The **2-dom** configuration has similar number of solved instance and mean total time as the basic configuration for all solvers, since it cannot generate sufficient nogoods for dominance breaking. The **3-dom** configuration usually achieves the smallest total time in all instance groups when using the Chuffed and CP-SAT solvers. The **4-dom** configuration, while having a longer mean time due to the overhead of solving generation CSPs, still achieves the smallest number of solved instances for the CP-SAT solver. The results show the usefulness of generated nogoods in solving the talent scheduling problem, but the best method varies for different configurations.

We also test our method on instances previously used by Smith (2005), and present the results in Table 3, highlighting the smallest total time for each instance. We only show the result for the Chuffed(VSIDS) configuration since it performs the best in the talent scheduling problem (as shown in Table 2). The instances are sorted by the time of the **basic** method and then by the smallest total time of **L-dom**. As shown in Table 3, our method reduces the total time for the majority of instances compared to the **basic** method.

5.2 Warehouse Location Problem

The Warehouse Location Problem (Van Hentenryck, 1999) is prob034 in CSPLib (Gent & Walsh, 1999). In this problem, we are required to choose a subset of possible warehouses in several locations to supply goods for a set of n stores. Each warehouse has a certain capacity defining how many stores it can supply, and there is a cost for supplying goods for different stores. When a warehouse needs to supply more than one store, there is a fixed opening cost. The objective is to minimize the total cost consisting of the total opening cost

Config.	Group	**basic**		**2-dom**		**3-dom**		**4-dom**	
		#s	#t	#s	#t	#s	#t	#s	#t
Chuffed (User)	$n = 35$	0	–	**20**	**36.88**	20	2002.85	20	3629.15
	$n = 40$	0	–	**20**	**109.38**	20	3176.24	20	3742.91
	$n = 45$	0	–	**20**	**154.63**	20	3726.52	20	3722.41
	$n = 50$	0	–	16	917.73	19	3973.95	**19**	**3966.55**
Chuffed (VSIDS)	$n = 35$	0	–	**20**	**44.85**	20	2045.68	20	3634.98
	$n = 40$	2	4310.27	**20**	**114.28**	20	3201.58	20	3765.55
	$n = 45$	4	4382.41	**20**	**145.25**	20	3778.30	20	3742.90
	$n = 50$	3	5923.10	**20**	**463.40**	19	3920.71	19	3928.86
CP-SAT (User)	$n = 35$	2	3447.64	**20**	**47.38**	20	2000.19	20	3615.22
	$n = 40$	8	443.69	**20**	**53.52**	20	2133.72	20	3617.49
	$n = 45$	5	1348.29	**20**	**74.46**	20	3057.62	20	3625.16
	$n = 50$	2	3930.56	**20**	**81.75**	20	3624.23	20	3633.50
CP-SAT (Auto)	$n = 35$	20	43.50	**20**	**37.49**	20	2006.22	20	3619.27
	$n = 40$	19	75.38	**20**	**41.39**	20	2189.52	20	3627.02
	$n = 45$	19	204.93	**20**	**52.72**	20	2906.85	20	3633.75
	$n = 50$	16	809.41	**20**	**70.22**	20	3618.79	20	3628.91
Gecode (User)	$n = 35$	0	–	**20**	**119.43**	20	3629.60	17	5340.69
	$n = 40$	0	–	**19**	**296.92**	12	5231.45	9	6354.57
	$n = 45$	0	–	**20**	**573.97**	3	7087.60	1	7155.10
	$n = 50$	0	–	**13**	**2838.35**	0	–	0	–

Table 4: Comparison of different methods in the warehouse location problem

for all warehouses and the total supplying costs for all stores. We model the problem using one integer variable for each store to indicate its supplying warehouse, and the objective is a weighted sum with nest logical disjunctions to indicate the opening of warehouses.

Table 4 gives the results for instances with $n \in \{30, 35, 40, 45\}$ and 15 warehouses. The results in the table show that the **2-dom** method solves more instances within the time limit and reduces the mean total time compared with the **basic** method. In this problem, it is usually not beneficial to increase the maximum length of nogoods to 3 or 4 since the reduced time in problem-solving cannot compensate for the additional overheads in nogood generation and handling of a large amount of nogoods during problem-solving.

5.3 Team Assignment Problem

The Team Assignment Problem appears in MiniZinc Challenge 2018 and 2022 (Stuckey, Becket, & Fischer, 2010). There are n boards, each consisting of m players. Players from the same board must be assigned to different teams. Each player has a rating and may have preferences regarding whom they want to be in the same team. The total rating of a team is the sum of ratings of all its players. The objective is to maximize the satisfaction of preferences and minimize the range of ratings of all teams simultaneously. We use one integer variable to represent the assigned team for each player. Therefore, the objective is a weighted sum of nested max/min functions to compute the range of team ratings and nested equalities to indicate whether any two players are in the same team.

Table 5 displays the results for $n \in \{9, 10, 11, 12\}$ and $m = 6$ obtained by various solver configurations. The Gecode solver's results are excluded since it failed to solve any instances. The addition of dominance breaking nogoods generally improves the performance of the

Config.	Group	**basic**		**2-dom**		**3-dom**		**4-dom**	
		#s	#t	#s	#t	#s	#t	#s	#t
Chuffed (User)	$n=9$	0	–	1	6576.16	**1**	**6387.49**	1	7008.99
	$n=10$	0	–	**2**	**6161.83**	2	6428.42	2	7013.05
	$n=11$	0	–	0	–	0	–	0	–
	$n=12$	0	–	0	–	0	–	0	–
Chuffed (VSIDS)	$n=9$	12	920.29	**20**	**136.96**	18	669.42	20	3837.97
	$n=10$	11	1841.32	**17**	**333.90**	17	1306.15	15	4491.93
	$n=11$	16	927.98	15	719.01	18	1751.76	**19**	**4125.28**
	$n=12$	11	2447.37	14	1023.45	**19**	**2881.53**	16	4925.37
CP-SAT (User)	$n=9$	9	1053.75	**12**	**939.54**	12	1561.77	12	4408.2
	$n=10$	3	3225.08	**5**	**2826.93**	4	3665.38	5	5549.33
	$n=11$	4	4324.48	3	4641.06	4	4152.11	**5**	**6196.5**
	$n=12$	**2**	**6876.71**	1	5038.65	1	5979.84	1	6942.78
CP-SAT (Auto)	$n=9$	**20**	**1.84**	20	5.39	20	186.04	20	2976.83
	$n=10$	**20**	**4.91**	20	8.82	20	80.4	20	1805.78
	$n=11$	**20**	**7.36**	20	13.81	20	127.28	19	3327.48
	$n=12$	**20**	**16.21**	20	27.1	20	258.17	19	3810.54

Table 5: Comparison of different methods in the team assignment problem

Chuffed(User), Chuffed(VSIDS) and CP-SAT(User) configurations compared to the **basic** method. However, the best method varies for different instance groups and configurations. For the CP-SAT(Auto) configuration, we notice that the addition of dominance breaking nogoods may not enhance the performance of the CP-SAT solver. This is due to the efficiency of the **basic** method across all instance groups. We also observe that the generated nogoods encode a class of symmetry-breaking constraints. In Section 6, we will discuss how to identify dominance and symmetry relations.

5.4 Budgeted Maximum Coverage Problem

The Budgeted Maximum Coverage Problem (Khuller et al., 1999) is a variant of the classical set cover problem. There is a ground set U and a collection T consisting of n subsets of U, where each subset is associated with a cost c_i. The goal is to find a subset of T such that their union covers the maximum number of elements subject to the constraint that the total cost does not exceed a given budget. We use one Boolean variable for each set in T to indicate whether the set is selected or not. The objective is a weighted sum with nested disjunctions to represent whether an element in U is covered or not. The search strategy is to select the unfixed subset in T with the smallest cost first.

Table 6 shows the results for instances with $|U| = n \in 50, 60, 70, 80$. The **basic** method can only solve easy instances in the Chuffed(User), CP-SAT(User), and Gecode(User) configurations, while it can solve most instances in the Chuffed(VSIDS) and CP-SAT(Auto) configurations. However, adding dominance breaking nogoods can solve more instances within the time limit, with **4-dom** achieving the most solved instances among all four methods in all configurations. When considering the geometric mean of total time, the performance of different methods varies for different instance groups and configurations. In general, the effectiveness of dominance breaking nogoods of different lengths depends on the difficulty of instances, which can be estimated by the baseline performance of the **basic**

Config.	Group	**basic**		2-**dom**		3-**dom**		4-**dom**	
		#s	#t	#s	#t	#s	#t	#s	#t
Chuffed (User)	$n = 50$	20	413.68	20	63.95	**20**	**19.87**	20	96.45
	$n = 60$	4	6403.32	12	3688.21	**20**	**758.63**	20	321.28
	$n = 70$	0	–	0	–	1	6552.69	**14**	**2959.77**
	$n = 80$	0	–	0	–	0	–	**2**	**6459.30**
Chuffed (VSIDS)	$n = 50$	20	0.90	**20**	**0.72**	20	4.93	20	95.78
	$n = 60$	20	9.79	**20**	**7.12**	20	14.59	20	224.81
	$n = 70$	20	126.46	20	97.17	**20**	**85.55**	20	470.03
	$n = 80$	20	682.88	20	564.47	**20**	**376.92**	20	1071.38
CP-SAT (User)	$n = 50$	20	513.05	20	38.03	**20**	**5.72**	20	77.68
	$n = 60$	0	–	11	3492.75	19	303.03	**20**	**257.52**
	$n = 70$	0	–	1	4777.46	2	4891.23	**12**	**2810.73**
	$n = 80$	0	–	0	–	0	–	**2**	**6187.46**
CP-SAT (Auto)	$n = 50$	20	0.76	**20**	**0.53**	20	3.93	20	80.70
	$n = 60$	20	6.34	**20**	**4.48**	20	8.63	20	180.72
	$n = 70$	20	122.71	20	69.92	**20**	**45.57**	20	382.62
	$n = 80$	16	994.28	19	703.44	**20**	**337.13**	20	756.08
Gecode (User)	$n = 50$	20	548.31	20	114.80	**20**	**40.1**	20	79.60
	$n = 60$	0	–	10	4685.98	18	1773.96	**20**	**511.63**
	$n = 70$	0	–	0	–	0	–	**4**	**5315.13**
	$n = 80$	0	–	0	–	0	–	0	–

Table 6: Comparison of different methods in the budgeted maximum coverage problem

method. As instances become more challenging, generating longer nogoods becomes more beneficial for this problem.

5.5 Partial Set Cover Problem

The Partial Set Cover Problem (Kearns, 1990) is another variant of the set cover problem. Given a ground set U and a collection T consisting of n subsets of U, the goal is to find a subset of T with the minimum total cost, whose union covers at least K elements in U. Similar to the budgeted maximum coverage problem, we define one Boolean variable for each set. The constraint of partial coverage is modelled as an inequality where we require a sum of nested disjunctions representing whether an element is covered to be larger than K. The search strategy is to select the subset with the smallest cost first.

Table 7 shows the result for standard benchmark with $|U| = n \in \{50, 60, 70, 80\}$. The baseline **basic** method can solve a relatively small number of instances using the user specified search heuristic, and can solve more than half of all instances in CP-SAT(User) and CP-SAT(Auto). The **4-dom** method has the largest number of solved instances in all configurations. Similar to that in the partial set cover problem, the benefit of using dominance breaking nogoods depends on the hardness of an instance in terms of the geometric mean of total time. If the **basic** method can efficiently solve an instance, then **2-dom** or **3-dom** can achieve the best trade-off between overhead of nogood generation and search space pruning. Otherwise, **4-dom** can achieve the smallest total time for hard instances.

Config.	Group	**basic**		**2-dom**		**3-dom**		**4-dom**	
		#s	#t	#s	#t	#s	#t	#s	#t
Chuffed (User)	$n = 50$	3	6009.59	17	575.79	**20**	**43.13**	20	113.93
	$n = 60$	0	–	1	6545.73	15	2214.59	**20**	**462.81**
	$n = 70$	0	–	0	–	0	–	**9**	**4165.95**
	$n = 80$	0	–	0	–	0	–	**2**	**6663.94**
Chuffed (VSIDS)	$n = 50$	20	34.89	**20**	**8.59**	20	8.69	20	113.10
	$n = 60$	20	293.73	20	135.47	**20**	**64.6**	20	291.75
	$n = 70$	16	1469.41	19	1104.00	**20**	**564.86**	20	625.60
	$n = 80$	12	3444.99	14	3126.72	13	2060.37	**19**	**2115.91**
CP-SAT (User)	$n = 50$	3	6509.45	18	135.46	**20**	**5.68**	20	85.16
	$n = 60$	0	–	2	5944.92	18	417.35	**20**	**276.08**
	$n = 70$	0	–	0	–	0	–	**11**	**4508.02**
	$n = 80$	0	–	0	–	0	–	**1**	**6607.88**
CP-SAT (Auto)	$n = 50$	20	17.07	**20**	**1.32**	20	4.34	20	87.04
	$n = 60$	20	98.41	20	66.20	**20**	**11.81**	20	213.81
	$n = 70$	10	2218.60	15	1157.88	**16**	**462.1**	20	608.83
	$n = 80$	6	3188.85	11	1710.74	11	1407.16	**14**	**1820.51**
Gecode (User)	$n = 50$	3	6157.5	17	888.69	**20**	**94.76**	20	97.49
	$n = 60$	0	–	1	6682.94	9	3810.94	**20**	**776.86**
	$n = 70$	0	–	0	–	0	–	**3**	**6877.89**
	$n = 80$	0	–	0	–	0	–	0	–

Table 7: Comparison of different methods in the partial set cover problem

5.6 Sensor Placement Problem

The Sensor Placement Problem (Krause et al., 2008) is a variant of the facility location problem (Cornuéjols, Nemhauser, & Wolsey, 1983), where we need to select a fixed cardinality subset of n locations to place sensors in order to provide service for customers. If we place a sensor at location i, then it provides service to a subset of reachable customers, and the service value for customer j is M_{ij}. Each customer chooses the facility with the highest service value from the opened sensors, and the goal is to maximize the total service value for all customers. We model the problem using one Boolean variable for each sensor to indicate whether it is selected or not. The objective is a sum of nested max/min functions. The search strategy is to select the unfixed location with the highest service value to the set of customers that are reachable by the sensor placed at the location.

Table 8 shows the result for instances with $n \in \{50, 60, 70, 80\}$. All methods can solve most of all instances in this problem under all configurations. Without generated nogoods, the **basic** method can achieve the best performance in Gecode(User). Compared with **basic**, the **2-dom** method can still slightly reduce the geometric mean of total time in CP-SAT and Gecode(User). In other configurations, **3-dom** usually achieves the smallest mean total time in most instance groups. The **4-dom** method uses longer dominance breaking nogoods for problem-solving, but the overhead of nogood generation makes the overall performance non-competitive compared with **basic**.

5.7 Discussion

We conducted extensive experiments on six optimization problems using different solvers to demonstrate the benefits of augmenting the problem model with dominance breaking

Config.	Group	**basic**		**2-dom**		**3-dom**		**4-dom**	
		#s	#t	#s	#t	#s	#t	#s	#t
Chuffed (User)	$n = 50$	20	55.78	**20**	**46.26**	20	45.00	20	195.95
	$n = 60$	20	147.30	20	124.55	**20**	**114.42**	20	454.29
	$n = 70$	20	478.16	20	356.00	**20**	**291.67**	20	974.88
	$n = 80$	20	1356.97	20	1065.48	**20**	**860.06**	20	2077.84
Chuffed (VSIDS)	$n = 50$	20	108.63	20	86.92	**20**	**75.10**	20	215.75
	$n = 60$	20	422.12	20	299.38	**20**	**223.46**	20	567.77
	$n = 70$	20	1021.77	20	757.43	**20**	**609.66**	20	1263.13
	$n = 80$	20	2908.40	20	2051.19	**20**	**1705.65**	20	3153.15
CP-SAT (User)	$n = 50$	20	51.23	**20**	**46.15**	20	49.4	20	169.02
	$n = 60$	20	120.34	**20**	**102.55**	20	107.14	20	382.18
	$n = 70$	20	311.09	20	253.50	**20**	**244.04**	20	753.75
	$n = 80$	20	801.01	20	646.39	**20**	**603.78**	20	1534.20
CP-SAT (Auto)	$n = 50$	20	129.5	20	87.56	**20**	**74.9**	20	195.05
	$n = 60$	20	688.88	20	512.81	**20**	**423.21**	20	577.04
	$n = 70$	17	3808.58	17	1914.38	**19**	**1623.73**	20	2035.21
	$n = 80$	1	6833.14	5	6259.22	6	5842.2	**7**	**5980.07**
Gecode (User)	$n = 50$	20	30.75	**20**	**28.44**	20	33.08	20	156.71
	$n = 60$	20	74.25	**20**	**61.99**	20	71.92	20	356.40
	$n = 70$	20	180.93	**20**	**150.64**	20	158.01	20	733.12
	$n = 80$	20	475.98	**20**	**378.69**	20	388.35	20	1510.43

Table 8: Comparison of different methods in the sensor placement problem

nogoods generated by our proposed method. The experimental results show that the best configuration varies from problem to problem, but our method can improve the performance in terms of the number of solved instances and the total time with nogoods of appropriate length in most cases. However, our method introduces additional overheads in nogood generation and the handling of a large number of nogood constraints. The usefulness of automatic dominance breaking depends on whether the overhead is less than the reduced time for problem-solving.

Determining the optimal length of generated nogoods in advance is non-trivial. Furthermore, we observe in the team assignment problem that additional dominance breaking nogoods may not improve the performance of a solver. Therefore, it is worthwhile to understand the semantics of generated nogoods and apply more advanced methods to exploit the discovered dominance and symmetry relations. In Section 6, we will demonstrate how to manually examine the pattern of nogoods and generalize them to compact dominance breaking constraints for all instances of a problem.

6. Discovering Dominance Relations by Inspection

Our method, which follows the framework of automatic dominance breaking (Lee & Zhong, 2020, 2023, 2021), attempts to generate all dominance breaking nogoods before problem-solving, and sometimes the number of nogoods is so large that generating all nogoods will cost too much time for each problem instance. Note that nogoods are the most basic units of constraints. Every high-level constraint can be decomposed into a set of nogoods, and conversely, it is possible to combine a group of nogoods into a high-level constraint. In this section, we give several case studies on how to discover the high-level dominance breaking constraints by examining the patterns of the generated nogoods by our method.

6.1 Still Mill Slab Design Problem

The first case study is the Still Mill Slab Design Problem (Kalagnanam, Dawande, Trumbo, & Lee, 1998), which is problem 039 in CSPLib (Gent & Walsh, 1999). The problem is to assign n colored orders with different weights to m slabs which has several possible sizes. The total weight of orders assigned to a slab cannot exceed the chosen slab size, and each slab cannot contain orders with more than 2 colors. The loss of each slab is the difference between the chosen slab size and the total weight of orders assigned to the slab. The objective is to minimize the total loss of all slabs. In this problem, we can model the problem using one integer variable x_i for each order i to represent which slab it is assigned to. To minimize the loss, the best size choice for a slab s is the smallest size that is larger than the total weight of all orders that are assigned to the slab. The objective is modelled as follows:

$$\sum_{j=1,\ldots,m} \mathit{element}(\mathit{Loss}, \sum_{i=1,\ldots,n} w_i \cdot \mathit{bool2int}(x_i = j))$$

where w_i is the weight for order i. The array Loss maps different total weights of orders in a slab to its corresponding loss, and $\mathit{element}(\mathit{Loss}, W)$ is the W^{th} element of the array Loss. For example, if a slab size can be either 2, 5 and 7, then $\mathit{Loss} = [0, 1, 0, 2, 1, 0, 1, 0]$ is an array with index starting from 0.

Previous works study different classes of symmetries, one of which is order symmetries (Frisch, Miguel, & Walsh, 2001b). Two orders with identical sizes and colors are equivalent so that their assigned slab can be exchanged. We apply our method to generate nogood of length 2 for the model from MiniZinc Challenge 2017 (Stuckey et al., 2010). The generation always times out in 3600 seconds, and the overhead always outweighs the benefit in problem-solving. Although a single nogood means relatively little, a bunch of them together can derive a meaningful constraint collectively. By generating nogoods of length 2, we observe that when there are nogoods for some pairs of variables x_i and x_j, the set of generated nogoods is always $\{x_i \neq v_i \vee x_j \neq v_j \mid v_i, v_j \in \{1, \ldots, m\} \wedge v_i > v_j\}$. For example, in the instance[2] from CSPLib where there are $n = 111$ orders and $m = n$ slabs, we can generate a set of nogoods as follows:

$$\begin{array}{llll}
x_2 \neq 2 \vee x_{26} \neq 1, & & & \\
x_2 \neq 3 \vee x_{26} \neq 1, & x_2 \neq 3 \vee x_{26} \neq 2 & & \\
x_2 \neq 4 \vee x_{26} \neq 1, & x_2 \neq 4 \vee x_{26} \neq 2, & x_1 \neq 4 \vee x_{26} \neq 3 & \\
x_2 \neq 5 \vee x_{26} \neq 1, & x_2 \neq 5 \vee x_{26} \neq 2, & x_2 \neq 5 \vee x_{26} \neq 3, & x_2 \neq 5 \vee x_{26} \neq 4, \\
\ldots & \ldots & \ldots & \ldots \\
x_2 \neq 111 \vee x_{26} \neq 1, & \ldots & x_2 \neq 111 \vee x_{26} \neq 109, & x_2 \neq 111 \vee x_{26} \neq 110
\end{array}$$

We observe that such a set of nogoods implies that x_1 can never be larger than x_2, and therefore they can be combined into one single inequality constraint $x_1 \leq x_2$. Note that these constraints are designed to ensure that the order i is placed on a slab with an index

2. https://www.csplib.org/Problems/prob038/data/111Orders.txt.html

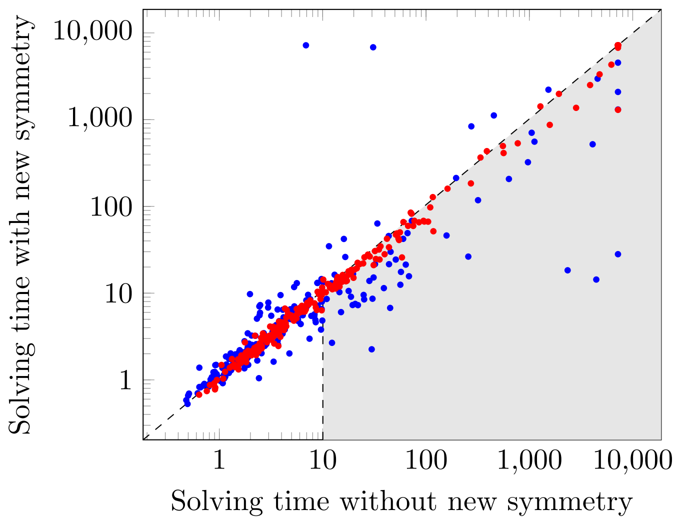

Figure 1: Solving time comparison of Steel Mill Slab Design Problem

less than or equal to the slab index of order j when orders i and j are equivalent. It is surprising to find that two orders are deemed equivalent when

- they have the same size and color, or
- they have the same size, and each order has a colour that is used nowhere else.

While the first case is known in the literature (Frisch et al., 2001b), the second condition has never been revealed and exploited to the best of our knowledge.

We augment the model from MiniZinc Challenge 2017[3] with constraints to break the newly discovered symmetry relationship. Blue dots in Figure 1 represent the solving time with/without symmetry breaking constraints of the new discovered symmetries for all 380 instances from the steel mill slab library[4], and the dots below the diagonal line represent the instance benefiting from the newly discovered constraints. We observe that the solving time is reduced in the majority of cases, especially more so when the solving time of the original model requires more than 10 seconds. The hard instances are represented by dots in the shaded region in Figure 1. Note that both axes are in log scale, and the speed-up of new constraints is up to two orders of magnitude. However, we also observe that several outliers on the top-left part require substantially more solving time after adding the new symmetry breaking constraints. This is due to the conflict between the search heuristic and the static symmetry breaking constraints (Gargani & Refalo, 2007). We also conduct experiments by adopting the SBDS-1UIP method (Chu, Garcia De La Banda, Mears, & Stuckey, 2014) implemented in the Chuffed solver, a dynamic symmetry breaking method combined with lazy clause generation, in order to avoid conflict. Red dots in Figure 1 represents the solving time with/without applying the SBDS-1UIP method using the newly discovered symmetries. Even though the dynamic method can avoid substantially deteriorating the

3. https://github.com/MiniZinc/minizinc-benchmarks/tree/master/steelmillslab
4. http://becool.info.ucl.ac.be/steelmillslab

performance, the overall improvement in solving efficiency is also less than that of the static symmetry breaking constraints.

6.2 Balanced Academic Curriculum Problem

The Balanced Academic Curriculum Problem (Castro & Manzano, 2001) is problem 030 in CSPLib (Gent & Walsh, 1999). There are n courses each associated with several credits representing the effort required to complete the course, and courses need to be assigned to m academic periods subject to the course prerequisite constraints. The workload of each period is the sum of all credits of courses that are assigned to the period. The objective is to minimize the maximum academic load for all periods to balance the loads among academic periods. We use one integer variable x_i for each course to represent its assigned academic period. To balance the workloads, the objective function is a maximum function with nested summation functions as follows:

$$\max_{j=1,\ldots,m}\left(\sum_{i=1}^{n} c_i \cdot \mathit{bool2int}(x_i = j)\right),$$

where c_i is the credits for course i. A prerequisite constraint between course i and j can be modelled as $x_i < x_j$.

By analyzing the nogoods of a small instance, we find that the set of generated nogoods of length 2 involving two variables x_i and x_j is always of the form $\{x_i \neq v_i \vee x_j \neq v_j \mid v_i, v_j \in \{1,\ldots,p\} \wedge v_i > v_j\}$, and can be combined into an inequality constraint $x_i \leq x_j$. When there are such nogoods involving two variables x_i and x_j, courses i and j must satisfy the following conditions:

- their credits are the same, i.e., $c_i = c_j$,
- for any $k \in \{1,\ldots,n\}$, if there is a prerequisite constraint $x_i < x_k$, then there must be a constraint $x_j < x_k$, and
- for any $k \in \{1,\ldots,n\}$, if there is a prerequisite constraint $x_k < x_j$ then there must be a constraint $x_k < x_i$.

These conditions are equivalent to those proposed by Monette, Jean-Noël et al. (Monette, Schaus, Zampelli, Deville, & Dupont, 2007), which shows that our method can also reveal dominance breaking constraints written by experts in the literature. Note that the inequality constraints for the balanced academic curriculum problem are dominance breaking constraints, while those of the steel mill slab design problem is for symmetry breaking.

7. Concluding Remarks

In this paper, we generalize the framework of automatic dominance breaking to constraint optimization problems with nested functions, where the derivation of sufficient conditions in a generation CSP is formulated formally. We identify that common function properties such as monotonicity, commutativity and associativity are useful in deriving weaker sufficient conditions such that more dominance breaking nogoods can be generated. We implement the tool for automatic dominance breaking using the MiniZinc compiler. The experimentation shows that the tool can discover dominance breaking nogoods for COPs with more varying objectives and constraints, and the generated nogoods are effective in pruning the search space and reducing the time for problem-solving.

Our tool is capable of compiling and synthesizing generation CSPs for problems in the MiniZinc benchmarks[5]. However, whether a benchmark can benefit from our method cannot be guaranteed, as solving the generation CSP may sometimes incur significant overhead, or the generated nogoods may not improve problem-solving. In this paper, we only exploit the discovered dominance relations by adding additional constraints to the original optimization problem. It is known that static dominance breaking may not improve solving performance in various problems (Chu & Stuckey, 2015; Monette et al., 2007). In the literature, several advanced methods have been proposed to exploit dominance relations in the Branch and Bound algorithm. For example, Chu and Stuckey (2013) propose *dominance jumping*, which avoids the potential conflicts between static dominance breaking constraints and user-specified search heuristics. Isoart and Régin (2021) proposed utilizing k-opt heuristics, which can be interpreted as exploiting dominance rules, to propagate the mandatory Hamiltonian path constraint in the context of the traveling salesman problem. Exploiting the dominance relations from the generated nogoods to improve search performance in general is an interesting direction for future research.

Our method requires a complete constraint instance to synthesize generation CSPs. Automatic detection of dominance relations from constraint models alone is an interesting area for future research. As demonstrated in the case studies in Section 6, nogoods with relevant semantics can be combined into high-level constraints that can be handled efficiently. One direction for future work is to automate the process of deriving high-level constraints and transferring the constraints from small instances to larger instances of the same problem type. The acquired constraints can help users better understand the target COP and improve the efficiency of existing models.

Our method is also related to lazy clause generation (Ohrimenko et al., 2009), which learns nogoods upon detecting a conflict during constraint propagation and can be thought of as dynamic dominance breaking (Chu & Stuckey, 2015). We anticipate that the nogoods generated by our approach may overlap with those generated by lazy clause generation solvers like Chuffed and CP-SAT, but the extent of this overlap would depend on the specific problems and configurations. A formal analysis on the performance gains of our method in solvers with or without lazy clause generation would be an interesting direction for future research.

Acknowledgements

We are grateful to the anonymous reviewers of CP-2022 and the Journal of Artificial Intelligence Research (JAIR) for their insightful comments and suggestions. We also acknowledge the financial support of a General Research Fund (RGC Ref. No. CUHK 14206321) by the University Grants Committee, Hong Kong.

5. https://github.com/MiniZinc/minizinc-benchmarks

References

Aldowaisan, T. (2001). A new heuristic and dominance relations for no-wait flowshops with setups. *Computers and Operations Research*, *28*(6), 563–584.

Baader, F., & Nipkow, T. (1998). *Term rewriting and all that.* Cambridge University Press.

Baptiste, P., Le Pape, C., & Nuijten, W. (2001). *Constraint-based scheduling: applying constraint programming to scheduling problems*, Vol. 39. Springer Science & Business Media.

Booth, K. E., Tran, T. T., Nejat, G., & Beck, J. C. (2016). Mixed-integer and constraint programming techniques for mobile robot task planning. *IEEE Robotics and Automation Letters*, *1*(1), 500–507.

Bryant, R. E., Lahiri, S. K., & Seshia, S. A. (2002). Modeling and verifying systems using a logic of counter arithmetic with lambda expressions and uninterpreted functions. In *Proceedings of the 14th International Conference on Computer Aided Verification*, pp. 78–92.

Castro, C., & Manzano, S. (2001). Variable and value ordering when solving balanced academic curriculum problem. In *Proceedings of 6th Workshop of the ERCIM WG on Constraints.*

Cheng, T., Diamond, J., & Lin, B. M. (1993). Optimal scheduling in film production to minimize talent hold cost. *Journal of Optimization Theory and Applications*, *79*(3), 479–492.

Chu, G., Banda, M. G. d. l., & Stuckey, P. J. (2010). Automatically exploiting subproblem equivalence in constraint programming. In *Proceedings of the 7th international conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pp. 71–86.

Chu, G., Garcia De La Banda, M., Mears, C., & Stuckey, P. J. (2014). Symmetries, almost symmetries, and lazy clause generation. *Constraints*, *19*(4), 434–462.

Chu, G., & Stuckey, P. J. (2012). A generic method for identifying and exploiting dominance relations. In *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming*, pp. 6–22.

Chu, G., & Stuckey, P. J. (2013). Dominance driven search. In *Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming*, pp. 217–229.

Chu, G., & Stuckey, P. J. (2015). Dominance breaking constraints. *Constraints*, *20*, 155–182.

Cornuéjols, G., Nemhauser, G., & Wolsey, L. (1983). The uncapicitated facility location problem. Tech. rep., Cornell University Operations Research and Industrial Engineering.

Frisch, A. M., Harvey, W., Jefferson, C., Martinez-Hernandez, B., & Miguel, I. (2008). Essence: A constraint language for specifying combinatorial problems. *Constraints*, *13*(3), 268–306.

Frisch, A. M., Miguel, I., & Walsh, T. (2001a). Modelling a steel mill slab design problem. In *Proceedings of the IJCAI-01 Workshop on Modelling and Solving Problems with Constraints*.

Frisch, A. M., Miguel, I., & Walsh, T. (2001b). Symmetry and implied constraints in the steel mill slab design problem. In *Proceedings of the CP'01 Workshop on Modelling and Problem Formulation*, pp. 8–15.

Fukunaga, A. S., & Korf, R. E. (2007). Bin completion algorithms for multicontainer packing, knapsack, and covering problems. *Journal of Artificial Intelligence Research*, *28*, 393–429.

Gargani, A., & Refalo, P. (2007). An efficient model and strategy for the steel mill slab design problem. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, pp. 77–89.

Garrido, A., Onaindia, E., & Sapena, O. (2008). Planning and scheduling in an e-learning environment. A constraint-programming-based approach. *Engineering Applications of Artificial Intelligence*, *21*(5), 733–743.

Gent, I. P., & Walsh, T. (1999). CSPLib: a benchmark library for constraints. In *Proceedings of the 5th International Conference on Principles and Practice of Constraint Programming*, pp. 480–481.

Getoor, L., Ottosson, G., Fromherz, M., & Carlson, B. (1997). Effective redundant constraints for online scheduling. In *Proceedings of the the Fourteenth AAAI National Conference on Artificial Intelligence*, pp. 302–307.

Ibaraki, T. (1977). The power of dominance relations in branch-and-bound algorithms. *Journal of the ACM (JACM)*, *24*(2), 264–279.

Isoart, N., & Régin, J.-C. (2021). A k-Opt Based Constraint for the TSP. In Michel, L. D. (Ed.), *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*, Vol. 210 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 30:1–30:16.

Kalagnanam, J. R., Dawande, M. W., Trumbo, M., & Lee, H. S. (1998). Inventory matching problems in the steel industry. Tech. rep., IBM Research Report, Computer Science/Mathematics.

Katsirelos, G., & Bacchus, F. (2005). Generalized nogoods in CSPs. In *Proceedings of the Twentieth AAAI National Conference on Artificial Intelligence*, pp. 390–396.

Kearns, M. J. (1990). *Computational Complexity of Machine Learning.* MIT Press, Cambridge, MA, USA.

Khuller, S., Moss, A., & Naor, J. S. (1999). The budgeted maximum coverage problem. *Information processing letters*, *70*(1), 39–45.

Korf, R. E. (2004). Optimal rectangle packing: new results.. In *Proceedings of the Fourteenth International Conference on International Conference on Automated Planning and Scheduling*, pp. 142–149.

Korf, R. E., Moffitt, M. D., & Pollack, M. E. (2010). Optimal rectangle packing. *Annals of Operations Research*, *179*(1), 261–295.

Krause, A., Leskovec, J., Guestrin, C., VanBriesen, J., & Faloutsos, C. (2008). Efficient sensor placement optimization for securing large water distribution networks. *Journal of Water Resources Planning and Management*, *134*(6), 516–526.

Land, A., & Doig, A. (1960). An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, *28*(3), 497–520.

Lee, J. H. M., & Zhong, A. Z. (2020). Automatic generation of dominance breaking nogoods for a class of constraint optimization problems. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence*, pp. 1192–1200.

Lee, J. H. M., & Zhong, A. Z. (2021). Towards more practical and efficient automatic dominance breaking. In *Proceedings of the Thirty-fifth AAAI Conference on Artificial Intelligence*, pp. 3868–3876.

Lee, J. H. M., & Zhong, A. Z. (2022). Exploiting functional constraints in automatic dominance breaking for constraint optimization. In Solnon, C. (Ed.), *Proceedings of the 28th International Conference on Principles and Practice of Constraint Programming*, Vol. 235 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 31:1–31:17.

Lee, J. H., & Zhong, A. Z. (2023). Automatic generation of dominance breaking nogoods for a class of constraint optimization problems. *Artificial Intelligence*, *In Press*, 103974.

Leo, K. (2018). *Making the most of structure in constraint models.* PhD Thesis, Monash University.

Mears, C., & de la Banda, M. G. (2015). Towards automatic dominance breaking for constraint optimization problems. In *Proceedings of the 24th International Conference on Artificial Intelligence*.

Monette, J.-N., Schaus, P., Zampelli, S., Deville, Y., & Dupont, P. (2007). A CP approach to the balanced academic curriculum problem. In *Proceedings of the 7th International Workshop on Symmetry and Constraint Satisfaction Problems*, pp. 56–63.

Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., & Tack, G. (2007). MiniZinc: towards a standard CP modelling language. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, pp. 529–543.

Ohrimenko, O., Stuckey, P. J., & Codish, M. (2009). Propagation via lazy clause generation. *Constraints*, *14*(3), 357–391.

Oplobedu, A., Marcovitch, J., & Tourbier, Y. (1989). CHARME: Un langage industriel de programmation par contraintes, illustré par une application chez Renault. In *Ninth International Workshop on ExpertSystems and Their Applications: General Conference*, Vol. 1, pp. 55–70.

Perron, L., & Furnon, V. (2023). Or-tools. https://developers.google.com/optimization/.

Prestwich, S., & Beck, J. C. (2004). Exploiting dominance in three symmetric problems. In *Proceedings of the Fourth International Workshop on Symmetry and Constraint Satisfaction Problems*, pp. 63–70.

Rossi, F., Beek, P. v., & Walsh, T. (2006). *Handbook of constraint programming (Foundations of Artificial Intelligence).* Elsevier Science Inc.

Régin, J.-C. (1994). A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the Twelfth AAAI National Conference on Artificial Intelligence*, pp. 362–367.

Régin, J.-C. (1996). Generalized arc consistency for global cardinality constraint. In *Proceedings of the Thirteenth AAAI National Conference on Artificial intelligence*, pp. 209–215.

Schulte, C., Lagerkvist, M., & Tack, G. (2009). Gecode: Generic constraint development environment. https://www.gecode.org/.

Shaw, P. (2004). A constraint for bin packing. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming*, pp. 648–662.

Smith, B. M. (2005). Caching search states in permutation problems. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming*, pp. 637–651.

Stuckey, P. J., Becket, R., & Fischer, J. (2010). Philosophy of the MiniZinc challenge. *Constraints*, *15*(3), 307–316.

Van Hentenryck, P. (1999). *The OPL optimization programming language.* MIT press.