Exploiting Functional Constraints in Automatic Dominance Breaking for Constraint Optimization

Jimmy H. M. Lee
Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong

Allen Z. Zhong
Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong

Abstract

Dominance breaking is an effective technique to reduce the time for solving constraint optimization problems. Lee and Zhong propose an automatic dominance breaking framework for a class of constraint optimization problems based on specific forms of objectives and constraints. In this paper, we propose to enhance the framework for problems with nested function calls which can be flattened to functional constraints. In particular, we focus on aggregation functions and exploit such properties as monotonicity, commutativity and associativity to give an efficient procedure for generating effective dominance breaking nogoods. Experimentation also shows orders-of-magnitude runtime speedup using the generated dominance breaking nogoods and demonstrates the ability of our proposal to reveal dominance relations in the literature and discover new dominance relations on problems with ineffective or no known dominance breaking constraints.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming

Keywords and phrases Constraint Optimization Problems, Dominance Breaking

Digital Object Identifier 10.4230/LIPIcs.CP.2022.6

Funding We acknowledge the financial support of a General Research Fund (RGC Ref. No. CUHK 14206321) by the University Grants Committee, Hong Kong.

1 Introduction

Dominance relations [7, 18] in Constraint Optimization Problems (COPs) describe relations between two full assignments where one is known to be subordinate compared with another concerning satisfiability and/or objective value. *Dominance breaking*, which adds additional constraints to remove dominated full assignments, is known to be effective in a range of problems [1, 16, 22, 31] but also demands sophisticated insights into the problem structures. Lee and Zhong [25, 24] give the method of automatic dominance breaking for a class of COPs, which can identify dominance relations and generate dominance breaking constraints automatically. Yet, the method 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 is required to be either a separable function or a submodular function. This prevents the use of automatic dominance breaking for COPs with varying objectives and constraints, especially the ones with nested function calls.

Functional expressions are ubiquitous in problem modelling, 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 [10, 28] and normalized/flattened into a form with only standard constraints.

▶ **Example 1.** Consider a simple COP which minimizes the objective $\max(z_1, z_2) + 4z_3$ subject to the constraint $2z_1 + 3z_2 * z_3 \leq 5$, where $z_1, z_2, z_3 \in \{1, 2, 3\}$. The objective with the max function and the constraint with the multiplying function are not efficiently checkable. After normalization, the COP can become:

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

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 to exploit functional constraints to identify useful dominance relations in COPs with nested function calls. We first generalize the theory of dominance to normalized COPs which contain functionally defined variables and functional constraints. We present a method for automatic derivation of sufficient conditions for dominance relations in COPs based on functional constraints and common properties such as monotonicity, commutativity and associativity. The proposed method is implemented on top of the MiniZinc compiler [28]. 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 [11] and the Balanced Academic Curriculum Problem [5] to show how we can discover dominance relations and compact dominance (symmetry) breaking constraints by studying the nogood patterns of small instances.

2 Background

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* $S = \{x_1, \dots, x_k\}$ is a tuple $(v_1, \dots, v_k) \in \mathcal{D}^S = D(x_1) \times \cdots \times D(x_k)$, 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 $S' \subset S$. 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 *functional constraint* is of the form $y = h(x_1, \dots, x_k)$ where $h : \mathcal{D}^{\{x_1, \dots, x_k\}} \mapsto \mathcal{D}^{\{y\}}$ is a function mapping any assignment on variables $\{x_1, \dots, x_k\}$ to a unique assignment on y.

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.

A *dominance relation* $\prec$ *over* $\mathcal{D}^X$ [7] 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) $\bar{\theta} \in sol(P)$ and $\bar{\theta}' \notin sol(P)$, or (2) $\bar{\theta}, \bar{\theta}' \in sol(P)$ and $\bar{\theta}[obj] \leq \bar{\theta}'[obj]$, or (3) $\bar{\theta}, \bar{\theta}' \notin sol(P)$ and $\bar{\theta}[obj] \leq \bar{\theta}'[obj]$. Dominance relations can be generalized [25] 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 2.** *[25] 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.*

Note that removing all dominated full assignments in $\mathcal{D}^X_{\theta'}$ only requires to add a nogood $\neg\theta'$ to P. While generating all dominance breaking nogoods is impractical, Lee and Zhong [25] formulate it as constraint satisfaction to identify and exploit only a subset of such nogoods. The constraints in the *generation CSPs* are sufficient conditions over pairs (θ, θ') of assignments such that $\theta \prec \theta'$ with respect to P. The key step is to derive constraints in the generation CSP automatically as sufficient conditions for dominance relations. Lee and Zhong [25] give such constraints directly based on the objective and constraint types, but 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 [26] and similar to that shown in Example 1, to a problem model. Therefore, we have a set C_Y of functional constraints, each defining a variable $y \in Y$, and a set of non-functional constraints. Our proposed method utilizes the functional constraints and the properties of functions to derive sufficient conditions for dominance as shown in the following example.

▶ **Example 3.** Consider the COP in (1) and $\theta, \theta' \in \mathcal{D}^S$ where $S = \{z_1, z_2\}$. Our aim is to find sufficient conditions over θ and θ' that imply all full assignments in $\mathcal{D}^X_{\theta'}$ can be removed. Suppose we only consider full assignments that satisfy all functional constraints in (1). For each full assignment $\bar{\theta}' \in \mathcal{D}^S_{\theta'}$, we focus on a corresponding $\bar{\theta} \in \mathcal{D}^S$ where $\bar{\theta}[z_3] = \bar{\theta}'[z_3]$. By definition of dominance relations, if we have (a) betterment: $\bar{\theta}[obj] \leq \bar{\theta}'[obj]$, (b) implied satisfaction: $\bar{\theta}[y_2] \leq \bar{\theta}'[y_2]$, and (c) $\theta \neq \theta'$, then $\bar{\theta}'$ is dominated by $\bar{\theta}$ and hence can be removed. We can find sufficient conditions for betterment as follows:

- Variable *obj* is defined by $obj = y_1 + 4z_3$. If we have $\bar{\theta}[y_1] + 4\bar{\theta}[z_3] \leq \bar{\theta}'[y_1] + 4\bar{\theta}'[z_3]$, then $\bar{\theta}[obj] \leq \bar{\theta}'[obj]$ must hold since $\bar{\theta}$ and $\bar{\theta}'$ satisfy all functional constraints.
- Variable y_1 is defined by $y_1 = \max(z_1, z_2)$. It suffices to show that

$$\max(\bar{\theta}[z_1], \bar{\theta}[z_2]) + \bar{\theta}[z_3] \leq \max(\bar{\theta}'[z_1], \bar{\theta}'[z_2]) + \bar{\theta}'[z_3]. \tag{2}$$

- The summation function is monotonically increasing, (2) must be true if we have

$$\max(\bar{\theta}[z_1], \bar{\theta}[z_2]) \leq \max(\bar{\theta}'[z_1], \bar{\theta}'[z_2]) \wedge \bar{\theta}[z_3] \leq \bar{\theta}'[z_3] \tag{3}$$

- Since $\bar{\theta} \in \mathcal{D}^X_{\theta}$ and $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$, we have $\bar{\theta}[z_1] = \theta[z_1]$, $\bar{\theta}[z_2] = \theta[z_2]$, $\bar{\theta}'[z_1] = \theta'[z_1]$, and $\bar{\theta}'[z_2] = \theta'[z_2]$. The condition (3) is equivalent to

$$\max(\theta[z_1], \theta[z_2]) \leq \max(\theta'[z_1], \theta'[z_2]) \tag{4}$$

  Inequality $(\bar{\theta}[z_3] \leq \bar{\theta}'[z_3])$ must hold since $\bar{\theta}[z_3] = \bar{\theta}'[z_3]$. Thus, if θ and θ' satisfy (4), the betterment condition must hold.

Similarly, we can find the sufficient condition for implied satisfaction as follows:

- Variable y_2 is defined by $y_2 = 2z_1 - 3y_3$, $\bar{\theta}[y_2] \leq \bar{\theta}'[y_2]$ must be true if

$$2\bar{\theta}[z_1] \leq 2\bar{\theta}'[z_1] \wedge 3\bar{\theta}[y_3] \geq 3\bar{\theta}'[y_3] \tag{5}$$

- 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

$$\bar{\theta}[z_2] \geq \bar{\theta}'[z_2] \wedge \bar{\theta}[z_3] \geq \bar{\theta}'[z_3] \tag{6}$$

  Since $\bar{\theta}[z_3] = \bar{\theta}'[z_3]$, the latter condition must hold.
- 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 (5) and (6) must hold if

$$\theta[z_1] \leq \theta'[z_1] \wedge \theta[z_2] \geq \theta'[z_2] \tag{7}$$

The generation CSP for θ and θ' contains (4) and (7). To ensure the compatibility of generated nogoods, we can follow Lee and Zhong [25] to add the lexicographic ordering constraint $(\theta[z_1], \theta[z_2]) <_{lex} (\theta'[z_1], \theta'[z_2])$. One possible solution of the generation CSP is the pair (θ, θ') where $\theta = (1, 2)$ and $\theta' = (2, 1)$, and the constraint $\neg\theta' \equiv (z_1 \neq 2 \vee z_2 \neq 1)$ is a dominance breaking nogoods in (1). Similar derivation can also be applied to pairs of assignments over other scopes to obtain more dominance breaking nogoods.

As shown in Example 3, our method identify nogoods by the following process:

1. Choose a cardinality of a scope S
2. Enumerate all possible scope S with the chosen cardinality. For each S:
   a. Derive sufficient conditions and synthesize a generation CSP for S
   b. Solve all solutions of the generation CSP
   c. Collect the derived nogoods from the solutions (one nogood from each solution)
3. Add all the collected nogoods to the COP before solving

The key step is to synthesize a generation CSP considering the functional constraints. In the following, we give a theory of dominance for normalized COPs.

For ease of presentation, we associate each non-functional constraint $c \in (C \setminus C_Y)$ with a *reified variable* $b \in \{0, 1\}$, where a full assignment $\bar{\theta}$ satisfies c iff $\bar{\theta}[b] = 0$[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 $\bar{\theta}[b] \leq \bar{\theta}'[b]$ 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 a partition 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, and therefore, a constraint $c_y \in C_Y$ must be satisfied if the value of $y \in Y$ is computed from the assignments over variables $x_{i_1}, \ldots, x_{i_k}$. Note that when there is no flattening and reification, our definition of a COP degenerates to the classical definition [34].

We say that a full assignment $\bar{\theta}$ is *functionally valid* iff (a) $\bar{\theta}[b] = c(\bar{\theta}[x_{i_1}], \ldots, \bar{\theta}[x_{i_k}])$ for $(b = c(x_{i_1}, \ldots, x_{i_k})) \in C_B$ and (b) $\bar{\theta}[y] = h(\bar{\theta}[x_{i_1}], \ldots, \bar{\theta}[x_{i_k}])$ for $(y = h(x_{i_1}, \ldots, x_{i_k})) \in C_Y$.

[1] Note that this is different from the convention that 0 means false and 1 means true

Note that $\bar{\theta}$ in a normalized COP will correspond to a full assignment in the original non-flattened problem model iff $\bar{\theta}$ is functionally valid. The value for $y \in Y$ (respectively $b \in B$) in a functionally valid full assignment are determined by $c_y \in C_Y$ (respectively $c_b \in C_B$) as well as values for variables in Z. We define the set of *determining variables* $A(x) \subseteq Z$ of a variable $x \in X$ is

- $A(z) = \{z\}$ for a variable $z \in Z$,
- $A(y) = \cup_{x \in var(c_y) \setminus y} A(x)$ for a variable $y \in Y$, and
- $A(b) = \cup_{x \in var(c_b) \setminus b} A(x)$ for a variable $b \in B$.

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* $\bar{\theta}$ for each $\bar{\theta}'$ in $\mathcal{D}^X_{\theta'}$. Instead, we propose to check only if a specific $\bar{\theta}$ dominates $\bar{\theta}'$ by utilizing a *mutation mapping* for two assignments θ and θ' over the same scope.

▶ **Definition 4.** *The* mutation mapping $\mu^{\theta \to \theta'}$ *for two assignments θ, θ' over the scope S 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 values of variables in $Y \cup B$ accordingly. The mutation mapping $\mu^{\theta \to \theta'}$ is a bijection, and thus the inverse mapping $(\mu^{\theta \to \theta'})^{-1} = \mu^{\theta' \to \theta}$ always exists. The following proposition characterizes some useful properties of the mutation mapping.

▶ **Proposition 5.** *The followings are true for all full assignments $\bar{\theta} \in \mathcal{D}^X_{\theta}$ and $\bar{\theta}' = \mu^{\theta \to \theta'}(\bar{\theta})$:*

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

The following result gives a sufficient condition governing when $\theta \prec \theta'$ with respect to P.

▶ **Theorem 6.** *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: $\forall \bar{\theta} \in \mathcal{D}^X_{\theta}, \bar{\theta}[obj] \le \mu^{\theta \to \theta'}(\bar{\theta})[obj]$, and*
- *implied satisfaction: $\forall b \in B, \forall \bar{\theta} \in \mathcal{D}^X_{\theta}, \bar{\theta}[b] \le \mu^{\theta \to \theta'}(\bar{\theta})[b]$,*

then θ dominates θ' with respect to P.

Theorems 2 and 6 imply that $\neg\theta'$ is a *dominance breaking nogood* to remove all dominated solution in $\mathcal{D}^X_{\theta'}$. To further ensure that all generated nogoods are *compatible* in the sense that not all optimal solutions of P are eliminated, a lexicographic ordering constraint $\theta <_{lex} \theta'$ is also added to the generation CSP [25]. What remains is to find constraints over θ and θ' that are sufficient conditions for empty intersection, betterment and implied satisfaction. Empty intersection is trivially satisfied if $\theta \neq \theta'$. In Section 4, we will focus on finding sufficient conditions for betterment and implied satisfaction. Note that the above definitions and results degenerate to those by Lee and Zhong [25] when Y and C_Y are empty.

4 Automatic Sufficient Condition Derivation

In this section, we describe formally how functional constraints and their properties are used for deriving effective sufficient conditions for betterment and implied satisfaction, which are predicates requiring an inequality to hold for all $\bar{\theta} \in \mathcal{D}_{\theta}^{X}$. When it is clear from the context, we let $\bar{\theta}' = \mu^{\theta \to \theta'}(\bar{\theta})$ denote the image by the mutation mapping of θ and θ'. We first present a general algorithm which only utilizes functional constraints, followed by rules that exploits the functional properties of monotonicity, associativity and commutativity.

4.1 General Decomposition

To formalize the derivation of sufficient conditions, we use the inductive definition of *terms* [2]:

- a variable x 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 define $var(t) = \{x\}$ when $t \equiv x$, and $var(t) = \cup_{i=1}^{k} var(t_i)$ when $t \equiv f(t_1, \ldots, t_k)$. 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 $y = h(x_{i_1}, \ldots, x_{i_k})$. We say that a variable x is *fixed* in an assignment θ when the set of determining variables $A(x)$ of x is a subset of $var(\theta)$. A term t is *fixed in* θ iff all variables of t are fixed in θ, i.e., $\wedge_{x \in var(t)}(A(x) \subseteq var(\theta))$; otherwise t is a *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, and $t_1, \ldots, t_k$ are terms such that $\forall j \in \{1, \ldots, k\}, x_{i_j} \not\equiv t_j$. A substitution β can be *applied to a term* t to obtain $t\beta$ by replacing every occurrence of variable x_{i_j} in t by the term t_j for all $j \in \{1, \ldots, k\}$.

Let $\lhd$ be an operator in $\{\leq, \geq, =\}$. The betterment and implied satisfaction condition in Theorem 6 are predicates in the form of quantified inequalities, i.e., $(\forall \bar{\theta} \in \mathcal{D}_{\theta}^{X}, t\bar{\theta} \lhd t\bar{\theta}')$, where $t\bar{\theta}$ and $t\bar{\theta}'$ are obtained by substituting each variable in t with its values in $\bar{\theta}$ and $\bar{\theta}'$ respectively. Algorithm 1 shows an automatic process that derives predicates as sufficient conditions for the betterment and implied satisfaction, and all variables in the sufficient conditions are fixed in θ and θ'. The algorithm maintains two sets of predicates Q and F, where Q consists of predicates that have to be further processed, and F is the set of predicates whose variables are all in S. The while loop continues until Q is empty and all predicates has been processed. In each iteration of the while loop, a predicate is removed from Q and processed by *replacement* (lines 4 to 7), *binding* (lines 8 to 9), *deletion* (lines 10 to 11) and *general decomposition* (lines 12 to 14) rules respectively. Finally, F is returned for synthesizing the generation CSP. Note that t must be a function term of the form $f(t_1, \ldots, t_k)$ in line 13, since $var(t)$ is a subset of Z and $var(t)$ has non-empty intersection with both S and $Z \setminus S$. The following theorem states an important property of Algorithm 1. For simplicity, let $Q \wedge F$ denote the conjunction of all predicates in Q and F.

▶ **Theorem 7.** *Algorithm 1 preserves the invariant that $Q \wedge F$ is always a sufficient condition for betterment and implied satisfaction of P.*

Proof. Since Q is initialized with the betterment and implied satisfaction, the statement holds at the beginning. By induction, it suffices to show that when $Q \wedge F$ is still a sufficient condition after an iteration in the while loop. There are four rules in the iteration:

- Replacement: the predicate $\forall \bar{\theta} \in \mathcal{D}_{\theta}^{X}, t\bar{\theta} \lhd t\bar{\theta}'$ is equivalent to $\forall \bar{\theta} \in \mathcal{D}_{\theta}^{X}, (t\beta)\bar{\theta} \lhd (t\beta)\bar{\theta}'$, because we assume that full assignments are all functionally valid.

Algorithm 1 Deriving sufficient conditions for betterment and implied satisfaction

```
1: $Q \leftarrow \{(\forall\bar{\theta} \in \mathcal{D}_\theta^X, \bar{\theta}[obj] \leq \bar{\theta}'[obj])\} \cup \{(\forall\bar{\theta} \in \mathcal{D}_\theta^X, \bar{\theta}[b] \leq \bar{\theta}'[b]) \mid b \in B\}$, $F \leftarrow \emptyset$
2: while $Q \neq \emptyset$ do
3:     Remove a predicate $p \equiv (\forall\bar{\theta} \in \mathcal{D}_\theta^X, t\bar{\theta} \lhd t\bar{\theta}')$ from $Q$
4:     if $var(t) \cap (Y \cup B) \neq \emptyset$ then
5:         Let $x \in var(t) \cap (Y \cup B)$ be a variable defined by $x = f(x_{i_1}, \ldots, x_{i_k})$
6:         $\beta \leftarrow \{x/f(x_{i_1}, \ldots, x_{i_k})\}$
7:         $Q \leftarrow Q \cup \{(\forall\bar{\theta} \in \mathcal{D}_\theta^X, (t\beta)\bar{\theta} \lhd (t\beta)\bar{\theta}')\}$          // Replacement
8:     else if $var(t) \subseteq S \subseteq Z$ then
9:         $F \leftarrow F \cup \{(t\theta \lhd t\theta')\}$          // Binding
10:    else if $var(t) \subseteq (Z \setminus S)$ then
11:        Continue          // Deletion
12:    else
13:        Let $p$ be $(\forall\bar{\theta} \in \mathcal{D}_\theta^X, f(t_1\bar{\theta}, \ldots, t_k\bar{\theta}) \lhd f(t_1\bar{\theta}', \ldots, t_k\bar{\theta}'))$
14:        $Q \leftarrow Q \cup \{(\forall\bar{\theta} \in \mathcal{D}_\theta^X, t_i\bar{\theta} = t_i\bar{\theta}') \mid \forall i \in \{1, \ldots, k\}\}$          // General Decomposition
15:    end if
16: end while
17: return $F$
```

- Binding: the predicate $(\forall\bar{\theta} \in \mathcal{D}_\theta^X, t\bar{\theta} \lhd t\bar{\theta}')$ is equivalent to $(t\theta \lhd t\theta')$ by Proposition 5, since a variable x in $var(t)$ also belongs to $S \subseteq Z$, and we have $\bar{\theta}[x] = \theta[x]$ and $\bar{\theta}'[x] = \theta'[x]$.
- Deletion: by Proposition 5 again, $\forall x \in var(t)$ where $x \in Z$ 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: the conjunction $\wedge_{i=1}^k(\forall\bar{\theta} \in \mathcal{D}_\theta^X, t_i\bar{\theta} = t_i\bar{\theta}')$ implies $(\forall\bar{\theta} \in \mathcal{D}_\theta^X, f(t_1\bar{\theta}, \ldots, t_k\bar{\theta}) \lhd f(t_1\bar{\theta}', \ldots, t_k\bar{\theta}'))$ by definitions of functional and reified constraints.

Therefore, the invariant is preserved in Algorithm 1. ◀

Note that the replacement, binding and deletion rules are equivalent transformation of predicates, while general decomposition replaces $p \equiv (\forall\bar{\theta} \in \mathcal{D}_\theta^X, f(t_1\bar{\theta}, \ldots, t_k\bar{\theta}) \lhd f(t_1\bar{\theta}', \ldots, t_k\bar{\theta}'))$ into a set of predicates that are sufficient conditions for p.

▶ **Theorem 8.** *Algorithm 1 always 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 $y = h(x_{i_1}, \ldots, x_{i_k})$. By definition of a COP, $Y \cup B$ and $C_Y \cup C_B$ are finite sets. We maintain three natural numbers:

- v_1: the number of variables in $Y \cup B$ that have not been substituted in replacement,
- v_2: the number of occurrences of function symbols in Q, and
- v_3: the sum of $|var(t)|$ for all predicates $(\forall\bar{\theta} \in \mathcal{D}_\theta^X, t\bar{\theta} \lhd t\bar{\theta}') \in Q$.

We claim that applying each rule reduces the triple (v_1, v_2, v_3) in a lexicographic sense. Indeed, each variable $x \in Y \cup B$ is only substituted when x is in the flattened constraint or the reified constraint, replacement must decrease v_1 by 1. General decomposition decreases v_2 while keeping v_1 unchanged. Further, binding and deletion remove one predicate $(\forall\bar{\theta} \in \mathcal{D}_\theta^X, t\bar{\theta} \lhd t\bar{\theta}')$ from Q and therefore decrease v_3 by $var(t)$. The termination follows from the fact that there is no infinite descending sequence of triples of natural numbers. ◀

The set Q is empty upon termination, and the following corollary is a direct consequence of Theorems 7 and 8.

▶ **Corollary 9.** *When Algorithm 1 terminates, the conjunction of predicates in F is a sufficient condition for the betterment and implied satisfaction of P.*

In other words, Algorithm 1 is *sound* in the sense that F consists of predicates that are sufficient conditions for betterment and implied satisfaction in Theorem 6. The general decomposition considers that a function f is general without any properties, but this may result in too restrictive sufficient conditions. For example, if we use Algorithm 1 for the COP in (1), the resulting sufficient conditions for betterment and implied satisfaction will be $\theta[z_1] = \theta'[z_1]$ and $\theta[z_2] = \theta'[z_2]$, which is in conflict with the empty intersection condition in Theorem 6. No solution can be found by solving the generation CSP, and no nogoods can be generated. Therefore, we want more relaxed sufficient conditions as far as possible.

We say that a predicate α is *stronger than* another predicate β iff $\alpha \Rightarrow \beta$, and β is *weaker than* α. The weaker the sufficient conditions in the generation CSP, the more pairs of assignments will satisfy all the conditions and the more nogoods can be found by Theorem 6. The idea is to apply different decomposition rules to derive weaker sufficient conditions based on properties of functions in functional and reified constraints. Aggregation functions [17], such as summation, maximum and minimum, combine multiple values into a single representative value. They are common in modeling COPs and enjoy useful properties such as *monotonicity*, *commutativity* and *associativity.*

4.2 Decomposition 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 $(\forall i, a_i \geq b_i) \Rightarrow f(a_1, \ldots, a_k) \geq f(b_1, \ldots, b_k)$ where $a_i, b_i \in \mathbb{R}$. We also define the *reverse operators* of $\leq$, $\geq$ and $=$ to be $\geq$, $\leq$ and $=$ respectively. When the function f is monotonically increasing or monotonically decreasing, we have the following rules.

▶ **Definition 10.** *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 where $\lhd \in \{\leq, \geq, =\}$ is a comparison operator and $\rhd$ is the reverse operator of $\lhd$.*

- *Increasing decomposition: when f is monotonically increasing, the predicate p is replaced by $\{(\forall \bar{\theta} \in \mathcal{D}^X_\theta, t_i\bar{\theta} \lhd t_i\bar{\theta}') \mid \forall i \in \{1, \ldots, k\}\}$.*
- *Decreasing decomposition: when f is monotonically decreasing, the predicate p is replaced by $\{(\forall \bar{\theta} \in \mathcal{D}^X_\theta, t_i\bar{\theta} \rhd t_i\bar{\theta}') \mid \forall i \in \{1, \ldots, k\}\}$.*

Recall that all full assignments are functionally valid. The following theorem follows directly the definition of monotonically increasing and monotonically decreasing functions.

▶ **Theorem 11.** *The increasing decomposition and decreasing decomposition rules preserve that $Q \wedge F$ is a sufficient condition for betterment and implied satisfaction of P.*

Utilizing the property of monotonicity, increasing decomposition and decreasing decomposition can return weaker sufficient conditions than general decomposition.

▶ **Theorem 12.** *The conjunction of predicates in $\{(\forall \bar{\theta} \in \mathcal{D}^X_\theta, t_i\bar{\theta} \lhd t_i\bar{\theta}') \mid \forall i \in \{1, \ldots, k\}\}$ (respectively $\{(\forall \bar{\theta} \in \mathcal{D}^X_\theta, t_i\bar{\theta} \rhd t_i\bar{\theta}') \mid \forall i \in \{1, \ldots, k\}\}$) is weaker than the conjunction of predicates $\{(\forall \bar{\theta} \in \mathcal{D}^X_\theta, t_i\bar{\theta} = t_i\bar{\theta}') \mid \forall i \in \{1, \ldots, k\}\}$ returned by general decomposition.*

Proof. A predicate $(\forall \bar{\theta} \in \mathcal{D}^X_\theta, t_i\bar{\theta} = t_i\bar{\theta}')$ is always a sufficient condition for both $(\forall \bar{\theta} \in \mathcal{D}^X_\theta, t_i\bar{\theta} \lhd t_i\bar{\theta}')$ and $(\forall \bar{\theta} \in \mathcal{D}^X_\theta, t_i\bar{\theta} \rhd t_i\bar{\theta}')$. ◀

By Theorems 11 and 12, rules in Definition 10 can obtain a weaker sufficient condition for the betterment and implied satisfaction, and the general decomposition in Algorithm 1 should be replaced by them whenever possible.

4.3 Decomposition for Associative and Commutative Functions

In this section, we take advantage of associativity and commutativity so that the general, decreasing and increasing decomposition can obtain even weaker sufficient conditions. An aggregation function [17] f can take an arbitrary non-zero number of arguments, and we use a special notation to denote it. Let $\mathbf{t} = \langle t_1, \ldots, t_k \rangle$, $\mathbf{t}_1 = \langle t_1, \ldots, t_j \rangle$ and $\mathbf{t}_2 = \langle t_{j+1}, \ldots, t_k \rangle$ where $1 \leq j \leq k$, and the followings denote the same function call: $f(t_1, \ldots, t_k)$, $f(\mathbf{t})$ and $f(\mathbf{t}_1, \mathbf{t}_2)$. Two common properties of aggregation functions are:

- *Commutativity*: $f(t_1, \ldots, t_k) = f(t_{i_1}, \ldots, t_{i_k})$ where $\{1, \ldots, k\} = \{i_1, \ldots, i_k\}$.
- *Associativity*: $f(t) = t$ and $f(\mathbf{t}_1, \mathbf{t}_2) = f(f(\mathbf{t}_1), \mathbf{t}_2)$.

By commutativity, we can always permutate the arguments of $f(\mathbf{t})$ so that all fixed terms are clustered together.

▶ **Proposition 13.** *Let f be a commutative function and $\theta \in \mathcal{D}^S$ be an assignment where $S \subseteq Z$. We can always find a permutation $\{i_1, \ldots, i_k\} = \{1, \ldots, k\}$ and partitioning $\mathbf{t}_1 = \langle t_{i_1}, \ldots, t_{i_l} \rangle$ and $\mathbf{t}_2 = \langle t_{i_{l+1}}, \ldots, t_{i_k} \rangle$, where $1 \leq l \leq k$, such that $f(\mathbf{t}) = f(\mathbf{t}_1, \mathbf{t}_2)$ and all fixed arguments in $\mathbf{t}$ are in $\mathbf{t}_1$.*

The proof directly follows the definition of commutativity. We have the following rule for a commutative and associative aggregation function.

▶ **Definition 14.** *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 where $\lhd$ is an operator in $\{\leq, \geq, =\}$.*

- *Aggregation: when f is commutative and associative, p is replaced by the predicate $(\forall \bar{\theta} \in \mathcal{D}^X_\theta, f(f(\mathbf{t}_1), \mathbf{t}_2)\bar{\theta} \lhd f(f(\mathbf{t}_1), \mathbf{t}_2)\bar{\theta}')$, where all fixed terms are in $\mathbf{t}_1$.*

The following theorem follows directly that aggregation is an equivalent transformation.

▶ **Theorem 15.** *The aggregation rule preserves the invariant that $Q \wedge F$ is a sufficient condition for the betterment and the implied satisfaction conditions.*

The aggregation rule allows decomposition to obtain weaker sufficient conditions.

▶ **Theorem 16.** *Let f be a commutative and associative function. Suppose $p \equiv (\forall \bar{\theta} \in \mathcal{D}^X_\theta, f(\mathbf{t}_1, \mathbf{t}_2)\bar{\theta} \lhd f(\mathbf{t}_1, \mathbf{t}_2)\bar{\theta}')$ and $p' \equiv (\forall \bar{\theta} \in \mathcal{D}^X_\theta, f(f(\mathbf{t}_1), \mathbf{t}_2)\bar{\theta} \lhd f(f(\mathbf{t}_1), \mathbf{t}_2)\bar{\theta}')$ such that all fixed terms are in $\mathbf{t}_1$. The conjunction of predicates resulting from applying general decomposition to p is weaker than that to p'.*

Proof. After general decomposition, p is replaced by $\{(\forall \bar{\theta} \in \mathcal{D}^X_\theta, t_{i_j}\bar{\theta} = t_{i_j}\bar{\theta}') \mid \forall j \in \{1, \ldots, k\}\}$, while p' is replaced by $\{(\forall \bar{\theta} \in \mathcal{D}^X_\theta, f(\mathbf{t}_1)\bar{\theta} = f(\mathbf{t}_1)\bar{\theta}')\} \cup \{(\forall \bar{\theta} \in \mathcal{D}^X_\theta, t_{i_j}\bar{\theta} = t_{i_j}\bar{\theta}') \mid \forall j \in \{l+1, \ldots, k\}\}$. The claim directly follows from the fact that the conjunction of $\{(\forall \bar{\theta} \in \mathcal{D}^X_\theta, t_{i_j}\bar{\theta} = t_{i_j}\bar{\theta}') \mid \forall j \in \{1, \ldots, l\}\}$ implies $(\forall \bar{\theta} \in \mathcal{D}^X_\theta, f(\mathbf{t}_1)\bar{\theta} = f(\mathbf{t}_1)\bar{\theta}')$ since all full assignments are functionally valid. ◀

Similar results can also be proved for the increasing decomposition and decreasing decomposition rules in Definition 10. Algorithm 2 gives decomposition rules considering the properties of monotonicity, associativity and commutativity.

▶ **Example 17.** Suppose we want to find sufficient conditions for θ and θ' where $var(\theta) = var(\theta') = \{z_1, z_3\}$. Let $p \equiv (\forall \bar{\theta} \in \mathcal{D}^X_\theta, \min(\bar{\theta}[z_1], \bar{\theta}[z_2], \bar{\theta}[z_3]) \leq \min(\bar{\theta}'[z_1], \bar{\theta}'[z_2], \bar{\theta}'[z_3]))$ be a predicate in Q. If we apply increasing decomposition to p directly, then we get

$$\{(\forall \bar{\theta} \in \mathcal{D}^X_\theta, \bar{\theta}[z_i] \leq \bar{\theta}'[z_i]) \mid i \in \{1, 2, 3\}\} \tag{8}$$

Algorithm 2 New decomposition rules

```
1: Let $\mathbf{t}_1, \mathbf{t}_2$ be a partition of arguments in $\mathbf{t}$ where all fixed terms are in $\mathbf{t}_1$
2: if $f$ is commutative and associative and $|\mathbf{t}_2| \neq 0$ then
3:     $p \leftarrow (\forall \bar{\theta} \in \mathcal{D}_\theta^X, f(f(\mathbf{t}_1), \mathbf{t}_2)\bar{\theta} \lhd f(f(\mathbf{t}_1), \mathbf{t}_2)\bar{\theta}')$        // Aggregation
4: end if
5: Let $p$ be $(\forall \bar{\theta} \in \mathcal{D}_\theta^X, f(t_1\bar{\theta}, \dots, t_{k'}\bar{\theta}) \lhd f(t_1\bar{\theta}', \dots, t_{k'}\bar{\theta}'))$
6: if $f$ is monotonically increasing then
7:     $Q \leftarrow Q \cup \{(\forall \bar{\theta} \in \mathcal{D}_\theta^X, t_i\bar{\theta} \lhd t_i\bar{\theta}') \mid \forall i \in \{1, \dots, k'\}\}$        // Increasing decomposition
8: else if $f$ is monotonically decreasing then
9:     $Q \leftarrow Q \cup \{(\forall \bar{\theta} \in \mathcal{D}_\theta^X, t_i\bar{\theta} \rhd t_i\bar{\theta}') \mid \forall i \in \{1, \dots, k'\}\}$        // Decreasing decomposition
10: else
11:     $Q \leftarrow Q \cup \{(\forall \bar{\theta} \in \mathcal{D}_\theta^X, t_i\bar{\theta} = t_i\bar{\theta}') \mid \forall i \in \{1, \dots, k'\}\}$        // General decomposition
12: end if
```

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

$$\{\forall \bar{\theta} \in \mathcal{D}_\theta^X, \min(\min(\bar{\theta}[z_1], \bar{\theta}[z_3]), \bar{\theta}[z_2]) \leq \min(\min(\bar{\theta}'[z_1], \bar{\theta}'[z_3]), \bar{\theta}'[z_2])\} \tag{9}$$

by the aggregation rule. Then by increasing decomposition, we get

$$\{(\forall \bar{\theta} \in \mathcal{D}_\theta^X, \min(\bar{\theta}[z_1], \bar{\theta}[z_3]) \leq \min(\bar{\theta}'[z_1], \bar{\theta}'[z_3])), (\forall \bar{\theta} \in \mathcal{D}_\theta^X, \bar{\theta}[z_2] \leq \bar{\theta}'[z_2])\} \tag{10}$$

Note that after applying binding and deletion to (8) and (10) 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 p, and the former condition is stronger than the latter one.

The new derivation algorithm replaces line 14 in Algorithm 1 with Algorithm 2, and it has the following properties.

▶ **Theorem 18.** *The new algorithm always terminates.*

Proof. We define a tuple (v_1, v_2, v_3) which is the same as that of Theorem 8 except that v_2 is now the number of *free function terms* in Q. The values v_1 and v_3 decrease similarly, while we argue that v_2 also decreases in the decomposition in Algorithm 2.

- When f is not commutative and associative, decomposition in lines 5 to 11 will decrease the number of function terms, and hence v_2 is also reduced.
- When the function f is commutative and associative, the predicate p is written into $(\forall \bar{\theta} \in \mathcal{D}_\theta^X, f(f(\mathbf{t}_1), \mathbf{t}_2)\bar{\theta} \lhd f(f(\mathbf{t}_1), \mathbf{t}_2)\bar{\theta}')$ by the aggregation rule, where $f(f(\mathbf{t}_1), \mathbf{t}_2)$ is free and $f(\mathbf{t}_1)$ is fixed. The follow-up decomposition in lines 5 to 11 in Algorithm 2 will decrease v_2 while keeping v_1 unchanged.

Hence, the termination follows directly from the fact that there is no infinite decreasing sequence of triples of natural numbers. ◀

▶ **Theorem 19.** *When the new algorithm terminates, the conjunction of predicates in F is a sufficient condition for the betterment and implied satisfaction of P.*

Proof. By Theorem 7, replacement, binding, deletion and general decomposition all preserve the invariant that $Q \wedge F$ is a sufficient condition for the betterment and implied satisfaction. Algorithm 2 also preserve the invariant by Theorems 11 and 15. The statement holds since Q must be empty upon termination. ◀

We note that the *alldifferent* constraint [32], which enforces variables taking distinct values, is commutative but not associative or monotonic. Rules in Definitions 10 and 14 are not applicable at first glance. We can decompose $alldifferent(x_{i_1}, \ldots, x_{i_k})$ into a set of constraints $d_v = \sum_{j=1}^{k} bool2int(x_{i_j} = v)$ and $d_v \leq 1$, one for each value $v \in \cup_{j=1\ldots,k} D(x_{i_j})$. The variable d_v is a newly introduced variable whose value is the number of variables in $\{x_{i_1}, \ldots, x_{i_k}\}$ assigned value v. After decomposition, the set of constraints enjoys the properties of monotonicity, commutativity and associativity, and thus rules in Definitions 10 and 14 are now applicable. The idea can be applied similarly to support other global constraints like the global cardinality constraint [33, 30] and the bin packing constraint [35]. Note that global constraints are decomposed only in synthesizing the generation CSPs, and are untouched in problem solving.

5 Experimental Evaluation

In this section, we give experimental results to show the utility of our proposal. All experiments are run on Xeon E5-2630 2.60GHz processors. We use MiniZinc [28] as the high-level modeling language and implement our nogood generation method by modifying[2] the publicly available MiniZinc compiler with version 2.6.2. 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 MiniZinc model of the corresponding problem.

The augmented models are submitted to MiniZinc for solving using the Chuffed solver [29] with version 0.10.4. Note that our method aims to analyze a user-defined model, not necessarily that of the best model, and we specify the search strategies for all problems to demonstrate the effect of the additional dominance breaking nogoods in search space pruning. We use six benchmark problems, 20 random instances for each problem size. The models for the following three problems are from public benchmark suites:

- *Talent-n.* The Talent Scheduling Problem [6] is problem 039 in CSPLib [15]. Each actor appears in several scenes and is paid a fixed cost per day if they are present. They need to be present on location from the first scene they are in till the last scene they are in. We need to schedule n scenes to minimize the total cost for a set of actors. The dominance breaking constraints for **manual** are by Chu and Stuckey [7].
- *Warehouse-n.* The Warehouse Location Problem [37] is problem 034 in CSPLib [15]. We need to choose a subset of possible warehouses in different locations to supply a set of n existing customers such that the sum of building costs for warehouses and supply costs for customers is minimized.
- *Team-n-m.* The Team Assignment Problem appears in MiniZinc Challenge 2018 [36]. The problem consists of $n \times m$ players, where players have different ratings and need to be assigned to n teams. There are requests regarding which pair of players want to be in the same team. The objective is to satisfy as many requests as possible while balancing the total rating among all teams.

In addition to publicly available models, we also model three more problems in MiniZinc:

- *MaxCover-n.* The Budgeted Maximum Coverage Problem [21] is a variant of the 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

[2] 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.

	basic	**manual**	2-**dom**		3-**dom**		4-**dom**	
Problem	Total	Total	Solving	Total	Solving	Total	Solving	Total
Talent-16	187.79	5929.75	189.95	192.16	130.78	**148.91**	256.46	1988.75
Talent-18	1575.51	7200.00	1565.89	1568.29	672.26	**713.55**	1864.68	5760.68
Talent-20	5013.10	7200.00	4936.18	4960.54	2856.33	**2960.09**	3268.72	7006.10
Warehouse-35	7200.00	N/A	10.29	**52.11**	8.53	2442.71	8.51	3619.87
Warehouse-40	7200.00	N/A	46.08	**111.43**	32.93	3652.15	32.55	3657.33
Warehouse-45	7200.00	N/A	69.41	**140.92**	45.45	3690.84	46.19	3694.63
Team-6-5	24.48	N/A	10.57	**12.49**	9.70	32.00	8.88	427.73
Team-7-5	276.84	N/A	138.88	**146.15**	130.71	225.19	150.83	1745.96
Team-8-5	1983.53	N/A	819.58	**829.05**	767.52	1024.43	724.63	5191.70
MaxCover-45	75.91	N/A	53.47	53.79	5.07	**9.96**	0.27	83.93
MaxCover-50	615.04	N/A	464.81	465.53	26.31	**34.92**	1.12	134.99
MaxCover-55	3576.98	N/A	2859.60	2860.27	78.37	**91.53**	2.54	199.11
PartialCover-45	2383.20	N/A	366.17	368.03	59.44	**70.64**	2.49	90.25
PartialCover-50	3769.26	N/A	780.80	781.73	74.86	**88.45**	6.86	153.90
PartialCover-55	4640.06	N/A	1769.31	1770.42	211.83	**234.41**	15.23	240.68
Sensor-50	156.84	N/A	138.65	139.44	94.05	**108.99**	57.34	297.18
Sensor-60	595.46	N/A	404.27	405.52	269.61	**296.56**	172.43	709.37
Sensor-70	1615.18	N/A	1144.17	1145.83	810.01	**854.61**	651.72	1724.70

Table 1 Comparison of solving time for the **basic**, **manual** and L-**dom** methods.

the union covers the maximum number of elements subject to the constraint that the total cost does not exceed a given budget. The search strategy is to select the unfixed subset in T with the smallest cost first.

- *PartialCover-n.* The Partial Set Cover Problem [20] 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. The search strategy is also to select the subset with the smallest cost first.
- *Sensor-n.* The Sensor Placement Problem [23] is a variant of the facility location problem [8], 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. 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.

Note that the original method by Lee and Zhong [25] can handle none of the benchmarks effectively because of nested function calls in either the objective or constraints. 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 generation and use the *remaining* time for problem solving in L-**dom**. If nogood generation times out, we augment the problem model with all nogoods generated before the timeout.

In Table 1, we report the geometric mean of the problem solving time (Solving) and the total time (Total) for all benchmarks, where “N/A” in the **manual** column indicates that there are no dominance breaking constraints for the problem. We first compare the problem solving time of L-**dom** against **basic** to evaluate the usefulness of the generated nogoods, and we observe that the generated dominance breaking nogoods can significantly reduce the solving time in all benchmarks. As the maximum nogood length L increases and more generated nogoods are added to the problem model, the solving time is usually shorter except for *Talent-n*. We note that the solving time of 4-**dom** is larger than that of 3-**dom** for *Talent-n* due to the overhead caused by a large amount of generated nogoods of length 4.

We also compare the total time (generation time + solving time) of L-**dom** against **basic** and **manual**. For each set of benchmarks, we highlight the fastest time in bold. We observe that the nogood generation time of L-**dom** increases with the maximum nogood length L of the generated nogoods, and there is a trade-off between search space pruning and generation time. The optimal nogood length depends on the problem structure. For *Warehouse-n* and *Team-n-m*, 2-**dom** is the best and reduces up to 99.27% and 58.20% less time than **basic** respectively. In *Talent-n*, *MaxCover-n*, *PartialCover-n* and *Sensor-n*, 3-**dom** usually comes on top, and the percentage decrease in runtime is up to 54.71%, 97.44%, 96.95% and 80.48% respectively compared with **basic**. The performance gain of L-**dom** in problem solving usually outweighs the generation time in a range of problems when the maximum length L of nogoods is set appropriately.

We note that the solving time of **manual** is even larger than that of **basic** in *Talent-n*. Expressing manual dominance breaking for *Talent-n* in the MiniZinc model requires additional variables and introduces overheads for propagation. Chu and Stuckey [7] 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 high-level modeling language without modifying the backend solver.

6 Discovering Dominance Relations

Our method, which is based on that of Lee and Zhong [25, 24] 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. We observe that nogoods are the most basic units of constraints. Every high-level constraint can be decomposed into a group of nogoods, and vice versa. By examining the patterns of the generated nogoods, we could discover the embedded high-level dominance breaking constraints. We give two case studies in this section.

The first case study is the Still Mill Slab Design Problem [19], which is problem 039 in CSPLib [15]. The problem is to assign colored production orders with different sizes to slabs where each slab has a finite number of possible sizes. The total size 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 size of orders assigned to the slab, and the objective is to minimize the total loss of all slabs.

Previous works study different classes of symmetries, one of which is order symmetries [12], that is, two orders with identical size and color are equivalent. We apply our method to generate nogood of length 2 for the model from MiniZinc Challenge 2017 [36], which introduces one variable x_i to specify the slab that orders i is assigned to. The generation always times out within 3600 seconds, and the overhead always outweighs the benefit. Although a single nogood means relatively little, a bunch of them together can derive a meaningful constraint collectively. However, we investigate the semantics of nogoods and discover a new class of symmetries. By generating the nogoods of length 2, we observe that we can group all

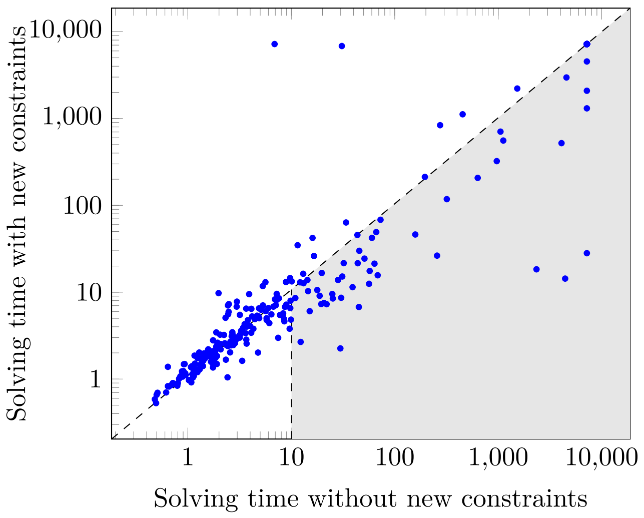

Figure 1 Solving time comparison with/without new symmetry breaking constraints in Steel Mill Slab Design Problem

nogoods involving the same set of variables and find that for some pairs of orders i and j, the nogoods are $x_i \neq v_i \vee x_j \neq v_j$ for all $v_i \in D(x_i), v_j \in D(x_j)$ s.t. $v_i > v_j$, which can be combined into one single inequality constraint $x_i \leq x_j$. These symmetry breaking constraints force the order i to be on a slab whose index is less than or equal to the slab index of order j when orders i and j are equivalent. The surprise is that two orders are identical not only when they have the same size and the same color, but also when *they have the same size and their colors are unique.* To the best of our knowledge, previous studies never reveal and exploit such a symmetry relationship.

We take a constraint model of the steel mill slab design problem from a public benchmark suite[3] and augment it with constraints to break the newly discovered symmetry relationship. Figure 1 shows the solving time 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. Several outliers 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 [13]. We believe the solving time can be reduced further by using dynamic symmetry breaking methods such as SBDS [14] or SBDD [9].

The other case study is the Balanced Academic Curriculum Problem [5], which is problem 030 in CSPLib [15]. There are n courses each associated with several credits representing the effort required to complete the course, and courses need to be assigned to 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.

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

	basic	**manual**	**2-dom**		**3-dom**		**4-dom**	
Problem	Total	Total	Solving	Total	Solving	Total	Solving	Total
Curriculum-60	61.82	23.76	27.80	77.44	21.88	3667.22	20.45	3673.27
Curriculum-65	291.35	62.14	71.26	160.95	62.57	3779.78	66.41	3785.61
Curriculum-70	518.31	133.54	148.84	242.62	126.19	3836.27	126.23	3837.26

Table 2 Comparison of solving time for the Balance Academic Curriculum Problem.

We perform experiments using the same experimental setting as that in Section 5 and report the results for problems with different course numbers in Table 2. The dominance breaking constraints for **manual** are by Monette, Jean-Noël et al. [27]. In general, the problem solving time of our method is smaller than that of **basic** but larger than that of **manual**. The overhead of L-**dom** mainly comes from the generation of dominance breaking nogoods before solving the COP. In addition, the dominance breaking constraints in **manual** are in the form of inequalities, which can be handled more efficiently than nogoods added by L-**dom** in a propagation-based constraint solver. Nevertheless, by analyzing the nogoods of a small instance, we find that the generated nogoods of length 2 can also be combined into inequality constraints similar to the case of the steel mill slab design problem. The inequality constraints we consider are the same as those proposed by Monette, Jean-Noël et al. [27], which shows that our method can also reveal dominance breaking constraints written by experts in the literature.

7 Concluding Remarks

In this work, 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 can compile and synthesize the generation CSPs for problems in the MiniZinc benchmarks[5]. Whether a benchmark can benefit from our method, however, cannot be guaranteed, since solving the generation CSP may sometimes incur a large overhead, or the generated nogoods do not help with problem solving. Our method requires a full constraint instance to synthesize generation CSPs. The automatic detection of dominance relations from constraint models alone is an interesting line of future work. As shown in the case studies in Section 6, nogoods with relevant semantics can be combined into high-level constraints that can be efficiently handled. One direction of future work is to automate the process of deriving high-level constraints by the techniques of automatic discovery of constraint from example solutions [3, 4], where the generated nogoods can be used as examples to learn and discover the desired constraints. The acquired constraints can help users to further understand the target COP and improve the efficiency of the existing models.

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

References

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

2 Franz Baader and Tobias Nipkow. *Term rewriting and all that.* Cambridge University Press, 1998.

3 Nicolas Beldiceanu and Helmut Simonis. ModelSeeker: Extracting global constraint models from positive examples. In *Data Mining and Constraint Programming: Foundations of a Cross-Disciplinary Approach*, pages 77–95. Springer International Publishing, Cham, 2016.

4 Christian Bessiere, Frédéric Koriche, Nadjib Lazaar, and Barry O'Sullivan. Constraint acquisition. *Artificial Intelligence*, 244:315–342, 2017.

5 Carlos Castro and Sebastian Manzano. Variable and value ordering when solving balanced academic curriculum problem. In *Proceedings of the ERCIM Working Group on Constraints*, 2001.

6 TCE Cheng, JE Diamond, and Bertrand MT Lin. Optimal scheduling in film production to minimize talent hold cost. *Journal of Optimization Theory and Applications*, 79(3):479–492, 1993.

7 Geoffrey Chu and Peter J Stuckey. A generic method for identifying and exploiting dominance relations. In *International Conference on Principles and Practice of Constraint Programming*, pages 6–22. Springer, 2012.

8 Gérard Cornuéjols, George Nemhauser, and Laurence Wolsey. The uncapicitated facility location problem. Technical report, Cornell University Operations Research and Industrial Engineering, 1983.

9 Torsten Fahle, Stefan Schamberger, and Meinolf Sellmann. Symmetry breaking. In *International Conference on Principles and Practice of Constraint Programming*, pages 93–107. Springer, 2001.

10 Alan M Frisch, Warwick Harvey, Chris Jefferson, Bernadette Martinez-Hernandez, and Ian Miguel. Essence: A constraint language for specifying combinatorial problems. *Constraints*, 13(3):268–306, 2008.

11 A.M. Frisch, I. Miguel, and T. Walsh. Modelling a steel mill slab design problem. In *Proceedings of the IJCAI-01 Workshop on Modelling and Solving Problems with Constraints*, pages 39–45, 2001.

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

13 Antoine Gargani and Philippe Refalo. An efficient model and strategy for the steel mill slab design problem. In *International Conference on Principles and Practice of Constraint Programming*, pages 77–89. Springer, 2007.

14 Ian P Gent and Barbara M Smith. Symmetry breaking in constraint programming. In *Proceedings of the 14th European Conference on Artificial Intelligence*, pages 599–603, 2000.

15 Ian P Gent and Toby Walsh. CSPLib: a benchmark library for constraints. In *International Conference on Principles and Practice of Constraint Programming*, pages 480–481. Springer, 1999.

16 Lise Getoor, Greger Ottosson, Markus Fromherz, and Björn Carlson. Effective redundant constraints for online scheduling. In *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Conference on Innovative Applications of Artificial Intelligence*, pages 302–307, 1997.

17 Michel Grabisch, Jean-Luc Marichal, Radko Mesiar, and Endre Pap. *Aggregation Functions.* Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2009.

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

19 Jayant R Kalagnanam, Milind W Dawande, Mark Trumbo, and Ho Soo Lee. Inventory matching problems in the steel industry. Technical report, IBM TJ Watson Research Center, 1998.

20 Michael J. Kearns. *Computational Complexity of Machine Learning.* MIT Press, Cambridge, MA, USA, 1990.

21 Samir Khuller, Anna Moss, and Joseph Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999.

22 Richard E Korf. Optimal rectangle packing: New results. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004)*, pages 142–149, 2004.

23 Andreas Krause, Jure Leskovec, Carlos Guestrin, Jeanne VanBriesen, and Christos Faloutsos. Efficient sensor placement optimization for securing large water distribution networks. *Journal of Water Resources Planning and Management*, 134(6):516–526, 2008.

24 Jimmy H. M. Lee and Allen Z. Zhong. Towards more practical and efficient automatic dominance breaking. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI 2021)*, pages 3868–3876, 2021.

25 Jimmy H.M. Lee and Allen Z. Zhong. Automatic generation of dominance breaking nogoods for a class of constraint optimization problems. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence*, pages 1192–1200, 2020.

26 Kevin Leo. *Making the Most of Structure in Constraint Models.* PhD thesis, Monash University, 2018.

27 Jean-Noël Monette, Pierre Schaus, Stéphane Zampelli, Yves Deville, and Pierre Dupont. A CP approach to the balanced academic curriculum problem. In *Symcon'07, The Seventh International Workshop on Symmetry and Constraint Satisfaction Problems*, 2007.

28 Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. MiniZinc: towards a standard CP modelling language. In *International Conference on Principles and Practice of Constraint Programming*, pages 529–543. Springer, 2007.

29 Olga Ohrimenko, Peter J Stuckey, and Michael Codish. Propagation via lazy clause generation. *Constraints*, 14(3):357–391, 2009.

30 A Oplobedu, J Marcovitch, and Y Tourbier. Charme: Un langage industriel de programmation par contraintes, illustré par une application chez renault. In *Ninth International Workshop on Expert Systems and their Applications: General Conference*, volume 1, pages 55–70, 1989.

31 Steven Prestwich and J Christopher Beck. Exploiting dominance in three symmetric problems. In *Fourth International Workshop on Symmetry and Constraint Satisfaction Problems*, pages 63–70, 2004.

32 Jean-Charles Régin. A filtering algorithm for constraints of difference in csps. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 362–367, 1994.

33 Jean-Charles Régin. Generalized arc consistency for global cardinality constraint. In *Proceedings of the 13th National Conference on Artificial intelligence*, volume 1, pages 209–215, 1996.

34 Francesca Rossi, Peter van Beek, and Toby Walsh. Handbook of constraint programming (foundations of artificial intelligence), 2006.

35 Paul Shaw. A constraint for bin packing. In *International Conference on Principles and Practice of Constraint Programming*, pages 648–662. Springer, 2004.

36 Peter J Stuckey, Ralph Becket, and Julien Fischer. Philosophy of the MiniZinc challenge. *Constraints*, 15(3):307–316, 2010.

37 Pascal Van Hentenryck. *The OPL optimization programming language.* MIT press, 1999.