Automatic Generation of Dominance Breaking Nogoods 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 a useful technique in solving Constraint Optimization Problems (COPs), which can speed up the solution process substantially by prune off many solutions in the search space. However, the technique usually requires human expertise and deep understandings of the problem structures. In this paper, we review recent research progress of dominance breaking, and introduce a new line of work on *automatic dominance breaking*, a framework to automate the identification and generation of effective constraints for dominance breaking in COPs. The main idea is to generate (a) only nogoods as the building blocks to model arbitrary dominance breaking constraints and (b) only the nogoods that are efficiently and conveniently identifiable by constraint satisfaction. At the end, we also discuss possible directions of future work to make automatic dominance breaking more efficient and more useful in practice.

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.ModRef.2022.1

1 Introduction

Constraint Optimization Problems (COPs) are ubiquitous in practice, and Constraint Programming (CP) usually solves COPs with the Branch and Bound (BnB) algorithm. The efficiency of the BnB algorithm depends highly on the structural properties of COPs, among which the existence of *dominance relations* is of practical importance. If one solution is dominated by another, the former is known to be subordinate to the latter with respect to satisfiability and/or the objective value. *Dominance Breaking* is the technique to exclude dominated solutions from the search space, and a wealth of research work [25, 47, 24, 46, 42, 31, 11, 21, 52] show that dominance breaking can dramatically reduce the search space and speed up the BnB algorithm in various problems.

In this paper, we review the research progress of dominance breaking in constraint programming. While dominance breaking is successfully applied to various COPs, past research work often requires sophisticated insights into the problem structures. In particular, constraints for dominance breaking are problem-specific and non-trivial to transfer to another problem domain. We introduce a recent line of work on *automatic dominance breaking*, a framework to *automate the process to make dominance breaking accessible to even non-experts*. The main idea is to restrict the form of dominance breaking constraints to *nogood constraints*, which can be elementary building blocks to model arbitrary constraints in CP. The generation of nogoods are modeled as constraint satisfaction consisting of sufficient conditions to guarantee that such nogoods prune off only subordinate solutions from the search space. The framework is the first of its kind as a truly automated method for dominance breaking for a class of COPs and eliminates the need for manual efforts to identify dominance relations, which will take us one step closer to the *Holy Grail of Computer Science* [17, 18]. In the concluding section, we discuss possible directions for future work to shed light on possibilities for more effective detection and utilization of dominance relations in COPs.

2 Background

A *Constraint Satisfaction Problem (CSP)* P is a tuple (X, D, C) consisting of a finite set of variables $X = \{x_1, \ldots, x_n\}$, a mapping D from a variable $x \in X$ to its finite domain $D(x)$ and a set of constraints C. A *literal* of $P = (X, D, C)$ is of the form $x_i = v_i$ where $x_i \in X$ and $v_i \in D(x_i)$. An *assignment* θ over a set of variables $S \subseteq X$ is a set of literals that has exactly one literal for each variable $x_i \in S$, where $S = var(\theta)$ is the *scope* of θ. We use $\bar{\theta}$ to emphasize that an assignment is a *full assignment* whose scope is X, or it is otherwise a *partial assignment*. Let $\theta[x_i] = v_i$ denote the value assigned by θ to variable $x_i \in var(\theta)$ if the literal $(x_i = v_i) \in \theta$, and we also let $\theta[S'] = \{(x_i = v_i) \in \theta \mid x_i \in S'\}$ to denote the *projection* of an assignment θ over a set of variables $S' \subset var(\theta)$.

A constraint $c \in C$ is a set of assignments over the variables $var(c)$. An assignment θ *satisfies* a constraint c iff $var(\theta) \supseteq var(c)$ and $\theta[var(c)] \in c$. A *solution* of a COP $P = (X, D, C, f)$ is a full assignment that satisfies all constraints in C. If the set of all solutions $sol(P)$ is non-empty, then P is *satisfiable*. A *nogood constraint* derived from θ is a constraint of the form $\neg\theta \equiv \vee_{x \in var(\theta)}(x \neq \theta[x])$, and its *length* is equal to the scope size $|var(\theta)|$. A *decision for a variable* $x \in var(\theta)$ in a nogood constraint $\neg\theta$ is the literal $x = \theta[x]$. A decision for a variable x is *subsumed* iff $D(x) = \{\theta[x]\}$ and is *falisified* iff $\theta[x] \notin D(x)$.

Let $\mathcal{D}^X$ be the set of all full assignments in P. A *Constraint Optimization Problem* (X, D, C, f) extends a CSP with an objective function $f : \mathcal{D}^X \mapsto \mathbb{R}$. Without loss of generality, solving a COP is to find an *optimal solution* $\bar{\theta}_{opt}$ such that $\bar{\theta}_{opt} \in sol(P)$ and $f(\bar{\theta}_{opt}) \leq f(\bar{\theta}')$ for any other solution $\bar{\theta}'$ of P. In other words, the objective function is minimized.

Following Chu and Stuckey [9], we define *dominance relations* to be relations over $\mathcal{D}^X$.

▶ **Definition 1.** *A* dominance relation $\prec$ of P *is a transitive and irreflexive relation such that* $\forall \bar{\theta}, \bar{\theta}' \in \mathcal{D}^X$, *if* $\bar{\theta} \prec \bar{\theta}'$, *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* $f(\bar{\theta}) \leq f(\bar{\theta}')$, *or*
3. *both* $\bar{\theta}$ *and* $\bar{\theta}'$ *are not solutions of* P *and* $f(\bar{\theta}) \leq f(\bar{\theta}')$

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

Dominance relations compare full assignments regarding the satisfaction of constraints and the optimality of the objective, and it is sound to prune all dominated full assignments.

▶ **Theorem 2.** *[9] Let* $\prec$ *be a dominance relation of a COP* $P = (X, D, C, f)$. *We can prune all full assignments* $\bar{\theta}' \in \mathcal{D}^X$ *whenever there exists* $\bar{\theta} \in \mathcal{D}^X$ *such that* $\bar{\theta} \prec \bar{\theta}'$, *without changing the satisfiability or optimal value of* P.

Note that a dominance relation should be both transitive and irreflexive. In other words, if P is satisfiable, then at least one optimal solution $\bar{\theta}_{opt}$ of P is not dominated by any other solutions for a valid dominance relation.

3 Related Work

In this section, we review the main and recent research work of dominance breaking. We first describe existing generic ways to identify dominance relations in Section 3.1, and then discuss methods to exploit dominance relations in Section 3.2.

3.1 Identification of Dominance Relations

Dominance relations are usually identified in a case-by-case manner in the literature [3, 7, 22, 25, 30, 42, 46], and there are a few attempts to automate the identification of dominance relations in COPs. Yu and Wah [54] propose a machine learning method to find candidate dominance relations in combinatorial optimization. While the method can be applied to several optimization problems, the generated candidate dominance relations lack correctness guarantees and require further manual inspection. Fischetti et al. [13, 14] propose a *local dominance procedure* to detect dominance relations in mixed integer linear programs. The idea is to identify a dominating node for a given search node in the BnB search by solving an auxiliary problem. Some design choices, such as the heuristic selection of nodes to solve the auxiliary problem, are required for efficient implementation.

Chu and Stuckey [8, 9] give the first generic method for deriving dominance breaking constraints for COPs with soundness guarantees. The method starts with a set of candidate mappings over the solution set of a given problem, followed by the derivation of dominance breaking constraints based on sufficient conditions for the candidate mappings to map solutions to better solutions. Manual efforts are required to select mappings, identify sufficient conditions, and simplify the dominance breaking constraints. Mears and de la Banda [38] automate the derivation process to a certain extent based on automated symmetry detection, which restricts the candidate mappings to symmetries that map solutions to solutions and map non-solutions to non-solutions. Their method still requires manual selection of symmetries produced by the automated symmetry detection process to be effective. Different from previous approaches, the framework of *automatic dominance breaking* [33, 34, 35] is a fully automatic method to identify dominance relations in COPs with soundness guarantees for a class of constraint optimization problems.

Symmetry relations are special classes of dominance relations, where solutions are equivalent in terms of the satisfaction of constraints and the objective value. Considerable progress has been made in the automatic detection of symmetry relations at different levels of abstraction. A substantial amount of research work [10, 16, 20, 39, 48, 50] focus on the automatic detection of symmetry relations in a CSP instance. While the detected symmetries can be exploited to speed up the solution process of a CSP, the detection may introduce extra overheads. Therefore, there is a line of work [26, 51] to detect symmetries for a class of CSPs so that the detected model-level symmetries can be applied to all CSPs in the class. However, they can only detect relatively "simple" symmetries like interchangeable values and interchangeable variables. Later, Mears et al. [40, 41] proposes a method that lifts the detected instance-level symmetries to the model level by an inductive reasoning process. Symmetries can also arise as a result of modeling decisions when a single problem solution corresponds to multiple assignments to the variables in a constraint model. The automated constraint modeling system CONJURE [1, 2] incorporates an rule-based method to detect modeling symmetries when refining high-level specifications into constraint models.

3.2 Static and Dynamic Dominance Breaking Methods

The methods to exploit dominance relations can be roughly divided into two categories: static and dynamic dominance breaking.

Static dominance breaking is a method to add additional *dominance breaking constraints* before the BnB search, and the constraints make dominated solutions become non-solutions. The method has been applied to speed up the solution process for various problems [25, 47, 24, 46, 42, 31, 11, 21, 52], and we give several example applications of static dominance breaking in recent papers. In the diameter constrained minimum spanning tree problem, de Una et al. [11] adopt a dominance rule from Noronha et al. [44] and add dominance breaking constraints to exclude solutions with higher cost from the search space. Gange and Stuckey [21] study the global constraint for value symmetry breaking and use the concert hall scheduling problem as a benchmark, in which dominance breaking constraints are added to remove suboptimal solutions. Senthooran et al. [52] study a problem of training engineers for service delivery, where they add dominance breaking constraints in the skill allocation subproblem to favour skill addition to an engineer with a subset of skills when compared to another engineer's skills. In general, dominance breaking constraints are of different forms for different problems, and they usually require sophisticated insights of the problem structure.

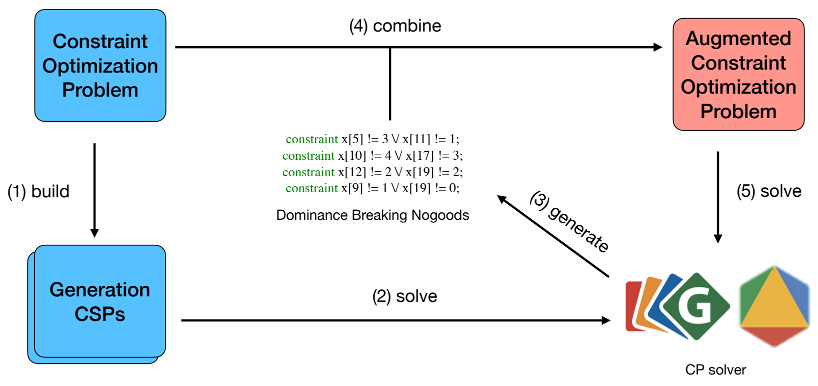
Static dominance breaking may not always be effective in the BnB search, since dominated solutions may improve the objective bound and allow additional pruning of search space. There are empirical evidences [15, 23] showing that dominance breaking constraints can have negative impact on the performance of the BnB search. Theoretical study by Ibaraki [27] also shows that the removal of dominated solutions in the BnB search with depth first search is guaranteed to reduce the number of search nodes only when the dominating solutions have been visited. Therefore, dynamic dominance breaking methods modify the search procedure to exclude only dominated solutions that cannot enable additional search space pruning. Most dynamic dominance breaking methods are problem specific [7, 15, 22, 49, 12]. For example, Focacci and Shaw [15] give a local search method for the travelling salesman problem, which prunes the current branch if it cannot be extended to a solution that is better than the current best solution.

There is few work on generic approaches for dynamic dominance breaking. Chu and Stuckey propose [9] *dominance jumping*, where the propagators of dominance breaking constraints only check the validity but never prune any values from the domain. When the BnB search reach a failure node, the search is restarted, and a temporary strategy is used to guide the search towards another part of the search space which contains better solutions potentially. In this way, the dominating solutions are visited first to obtain stronger objective bounds for pruning. Another generic approach is *automatic caching via constraint projection* [6], in which a caching key is computed and stored for each visited subproblem. Conditions on keys are given to check whether all solutions in the current subproblem are dominated by solutions in a cached subproblem. If the conditions are meet, the current subproblem will not be explored to avoid redundant search.

4 Automatic Generation of Dominance Breaking Nogoods

In this section, we describe an automated method to identify and generate dominance breaking constraints in COPs. The main difficulty for automation lies in the various forms of dominance breaking constraints for different problems, and it is overcome by restricting the form of constraints to be *nogood constraints* [28], which are elementary since any constraints can be represented as a conjunction of several nogood constraints in constraint programming. The workflow of the automated method is as follows (Figure 1):

1. Construct generation CSPs from the objective and constraints of a COP P.
2. Enumerate all solutions of the generation CSPs using a constraint solver.
3. Generate one nogood constraint for each solution of the CSPs.
4. Add all generated nogood constraints to the COP P.
5. Solve the COP with extra constraints by a constraint solver.


Constraint Optimization Problem
(4) combine
Augmented Constraint Optimization Problem
(1) build
Dominance Breaking Nogoods
(3) generate
(5) solve
Generation CSPs
(2) solve
CP solver

```
constraint x[5] != 3 ∨ x[11] != 1;
constraint x[10] != 4 ∨ x[17] != 3;
constraint x[12] != 2 ∨ x[19] != 2;
constraint x[9] != 1 ∨ x[19] != 0;
```

Figure 1 Workflow of Automatic Dominance Breaking for COPs

In this framework, the solution of the generation CSPs are pairs of partial assignments of P over a scope of the same size, where one is *dominating*, and the other is *dominated.* The constraints in generation CSPs are sufficient conditions to ensure that the derived nogood constraint from a dominated partial assignment is a valid dominance breaking constraints for P. The key step is to build generation CSPs *automatically* based on the COP P.

In the remaining of this section, we first provide theoretical explanations on how to construct generation CSPs for a class of COPs with *efficiently checkable* objectives and constraints (Section 4.1), followed by the technique called *common assignment elimination* for more efficient generation of nogoods (Section 4.2). Functional expressions are common in modelling COPs, and we show how to exploit functional constraints to enable automatic dominance breaking for problems with nested function calls (Section 4.3).

4.1 Automatic Dominance Breaking

Chu and Stuckey [8] investigate dominance relations over full assignments, and we generalize it to dominance relations over partial assignments. Let $\mathcal{D}^X_\theta = \{\bar{\theta} \in \mathcal{D}^X \mid \bar{\theta}[var(\theta)] = \theta\}$ be a subset of $\mathcal{D}^X$ in which full assignments are extending from θ. 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 $\prec$ of P. When the context is clear, we also let $\theta \prec \theta'$ denote θ dominates θ' with respect to P.

▶ **Theorem 3.** *If θ dominates θ' with respect to P, then removing all assignments in $\mathcal{D}^X_{\theta'}$ preserves the same satisfiability and optimal value of P.*

Theorem 3 is a direct consequence of Theorem 2 since all full assignments in $\mathcal{D}^X_{\theta'}$ are dominated. Note that removing all dominated full assignments in $\mathcal{D}^X_{\theta'}$ only requires to add a *dominance breaking nogood* $\neg\theta'$ to P. While generating all such nogoods is impractical, we can formulate it as constraint satisfaction to identify and exploit only a subset of such nogoods. Thus, we give a sufficient condition for $\theta \prec \theta'$ with respect to P.

▶ **Theorem 4.** *Let $P = (X, D, C, f)$ be a COP where θ and θ' are two partial assignments over $S \subseteq X$. If there exists a bijective mapping $\sigma : \mathcal{D}^X_{\theta'} \mapsto \mathcal{D}^X_\theta$ such that:*

- *irreflexivity: the transitive closure of σ is irreflexive,*
- *betterment: $\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}$, $f(\sigma(\bar{\theta}')) \leq f(\bar{\theta}')$, and*

- *implied satisfaction:* $\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}$, $\bar{\theta}' \in sol(P)$ *implies that* $\sigma(\bar{\theta}') \in sol(P)$,

then $\theta \prec \theta'$ *with respect to* P.

The idea is to construct a relation $\prec$ by taking the transitive closure of σ, and show that $\prec$ is a dominance relation in P. By Theorem 4, we need to find a bijection mapping to map full assignments in $\mathcal{D}^X_{\theta'}$ to those in $\mathcal{D}^X_{\theta}$ to fulfill irreflexivity, betterment and implied satisfaction. For the ease of analysis, we are interested in a pair (θ, θ') of assignments with the same scope, i.e., $var(\theta) = var(\theta')$, and define the *mutation mapping* for θ and θ' in P.

▶ **Definition 5.** *Let* $\theta, \theta' \in \mathcal{D}^S$ *be two assignments in a COP* $P = (X, D, C, f)$ *over the scope* $S \subseteq X$. *The mutation mapping* $\mu^{\theta' \to \theta} : \mathcal{D}^X_{\theta'} \mapsto \mathcal{D}^X_{\theta}$ *maps* $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$ *to* $\bar{\theta} \in \mathcal{D}^X_{\theta}$ *such that:*
- $\bar{\theta}[x] = \theta[x]$ *and* $\bar{\theta}'[x] = \theta'[x]$ *for* $x \in S$, *and*
- $\bar{\theta}[x] = \bar{\theta}'[x]$ *for* $x \notin S$

In other words, the full assignment $\mu^{\theta' \to \theta}(\bar{\theta}')$ "mutates" the θ' component of $\bar{\theta}'$ to θ. The irreflexivity condition turns out to be simple under the mutation mapping.

▶ **Proposition 6.** *If* $\theta \neq \theta'$, *then the mutation mapping* $\mu^{\theta' \to \theta}$ *is irreflexive, and its transitive closure is also irreflexive.*

What remains is to check the betterment and the implied satisfaction condition in Theorem 4. To generate dominance breaking nogoods efficiently, we add sufficient conditions, which are constraints over θ and θ', to generation CSPs to find pairs of partial assignments.

▶ **Example 7.** Consider a simple COP P to minimize the objective $x_1 + 4x_2 + 2x_3 + 8x_4$ subject to the constraint $3x_1 + 2x_2 + x_3 + x_4 \geq 3$, where $x_i \in \{0, 1\}$ are binary variables for $i = 1, \ldots, 4$. The problem P has a linear objective function and a linear inequality constraint, and a generation CSP consists of:
- Sufficient conditions for betterment: $\sum_{x_i \in S} a_i \theta[x_i] \leq \sum_{x_i \in S} a_i \theta'[x_i]$,
- Sufficient conditions for implied satisfaction: $\sum_{x_i \in S} b_i \theta[x_i] \geq \sum_{x_i \in S} b_i \theta'[x_i]$

where $a = [1, 4, 2, 8]$ and $b = [3, 2, 1, 1]$ are two arrays of integers, and $S = var(\theta) = var(\theta')$ is the scope for θ and θ'. One solution pair is (θ, θ') such that $\theta = \{x_1 = 1, x_3 = 1, x_4 = 0\}$ and $\theta' = \{x_1 = 1, x_3 = 0, x_4 = 1\}$. The derived dominance breaking nogood $\neg(x_1 = 1 \wedge x_3 = 0 \wedge x_4 = 1)$ can remove suboptimal solutions in P by Theorem 3.

Note that sufficient conditions for betterment and implied satisfaction correspond to the objectives and constraints of COPs, and we say that objectives and constraints with such sufficient conditions are *efficiently checkable*. There are other efficiently checkable objectives and constraints in addition to linear objectives and linear constraints in Example 7, and we refer interested readers to the paper [33] for more details.

Recall that our method enumerates all dominance breaking nogoods arising from the solutions of generation CSPs, while Theorems 3 and 4 only consider the soundness of adding one nogood constraint into P. We also need to ensure that all generated nogoods are *compatible* in the sense that not all optimal solutions of P are eliminated.

▶ **Theorem 8.** *If* (θ, θ') *satisfies betterment and implied satisfaction, and* $\theta <_{lex} \theta'$, *i.e.* θ *is lexicographically smaller than* θ', *then the lexicographically smallest optimal solution always satisfies* $\neg\theta'$.

Theorem 8 implies that all generated nogoods $\neg\theta'$ derived from the solutions of the generation CSPs are compatible, and adding all nogoods to the COP preserves the optimal value of P. The idea can be further generalized for more relaxed sufficient conditions that imply compatibility of dominance breaking nogoods [33], and hence the generation CSPs are less restrictive, resulting in more generated nogoods for search space pruning. Note that $\theta <_{lex} \theta'$ also implies $\theta \neq \theta'$, and therefore implies the irreflexivity in Theorem 4.

4.2 Common Assignment Elimination

An important advantage of the method in Section 4.1 is the ability to generate dominance breaking nogoods of different lengths. Example 7 gives an example nogood constraint of length 3, and the same reasoning can be applied to find nogoods of different length. In practice, we usually set the maximum scope size to a fixed integer L and attempt to augment the COP with all nogoods of length $l \leq L$. However, some long nogoods may be useless in pruning in the present of short nogoods, and *common assignment elimination* is a technique to strengthen generation CSPs and avoid generation of such redundant nogoods.

In a constraint programming solver, the domain of variables in a COP $P = (X, D, C, f)$ is usually enforced to be *generalized arc consistent* [37] with respect to all constraints $c \in C$. By definition, the domain is GAC with respect to a nogood constraint iff (a) there exists two decisions that are not subsumed, or (b) there is one decision that is falisified. The propagator maintaining GAC for $\neg\theta$ is triggered when all but one of the decisions are subsumed, and the value $\theta[x]$ is removed from $D(x)$ for the left decision $x = \theta[x]$.

▶ **Proposition 9.** *If $\tilde{\theta} \subseteq \theta$, then (1) $\neg\tilde{\theta} \Rightarrow \neg\theta$, and (2) the domain is GAC with respect to $\neg\theta$ implies that the domain is GAC with respect to $\neg\theta$.*

Proposition 9 implies that $\neg\theta$ is both *logically and propagation redundant* [5] and contributes no extra pruning in a constraint solver. Therefore, if $\neg\tilde{\theta} \in C$, then we can remove $\neg\theta$ and save the runtime cost of the propagator for $\neg\theta$ in constraint propagation.

In order to avoid generating redundant nogoods, we can further enhance the generation CSPs with extra conditions. In particular, we are interested in the case where a variable x is assigned to the same value in both θ and θ'. We say that a literal $(x = v)$ is *commonly eliminable* with respect to a constraint c iff for all pairs (θ, θ') such that $(x = v) \in \theta$ and $(x = v) \in \theta'$, (θ, θ') satisfies c implies that $(\tilde{\theta}, \tilde{\theta}')$ satisfies c where $\tilde{\theta} = \theta \setminus \{x = v\}$, $\tilde{\theta}' = \theta' \setminus \{x = v\}$, and $(x = v)$ is commonly eliminable with respect to a generation CSP if it is commonly eliminable with respect to all constraints in the generation CSP.

▶ **Theorem 10.** *Let c be a constraint over a pair of partial assignments. If a literal $(x = v)$ is commonly eliminable with respect to c, then for all (θ, θ') that satisfies c, there must be $(\tilde{\theta}, \tilde{\theta}')$ satisfying c such that $x \in S \Rightarrow (\theta[x] \neq v \vee \theta'[x] \neq v)$, where $S = var(\theta) = var(\theta')$.*

Theorem 10 implies that if (θ, θ') is a solution pair of a generation CSP such that $(x = v) \in (\theta \cap \theta')$ is commonly eliminable with respect to all constraints in the generation CSP, then there must exist another solution $(\tilde{\theta}, \tilde{\theta}')$ of smaller size, and $\neg\theta'$ is logically and propagation redundant with respect to $\neg\tilde{\theta}'$. Therefore, we can add extra condition $x \in S \Rightarrow (\theta[x] \neq v \vee \theta'[x] \neq v)$ for every commonly eliminable literal $(x = v)$ in order to avoid generating redundant dominance breaking nogoods. Further, if for every $v \in D(x)$, $(x = v)$ that is commonly eliminable with respect to the generation CSP, we can add a more succinct constraint $x \in S \Rightarrow (\theta[x] \neq \theta'[x])$ to generation CSPs.

To find commonly eliminable literals, we need to examine constraints in generation CSPs. Recall in Section 4.1 that a solution pair (θ, θ') of a generation CSP should fulfil: (1) lexicographical ordering: $\theta <_{lex} \theta'$, (2) sufficient conditions for betterment arising from the objective of P, and (3) sufficient conditions for implied satisfaction arising from constraints of P. It is straightforward to show that for an arbitrary literal $(x = v)$ is commonly eliminable with respect to the lexicographical ordering constraint.

▶ **Proposition 11.** *Let θ, θ' be partial assignments over $S = var(\theta) = var(\theta')$. If $x \in S$ and $\theta[x] = \theta'[x]$, then $\theta <_{lex} \theta'$ implies that $\tilde{\theta} <_{lex} \tilde{\theta}'$, where $(\tilde{\theta}, \tilde{\theta}') = (\theta[\tilde{S}], \theta'[\tilde{S}])$, and $\tilde{S} \setminus \{x\}$.*

The commonly eliminable literals are specific with respect to sufficient conditions for betterment and implied satisfaction of COPs.

▶ **Example 12.** Consider the sufficient conditions for betterment and implied satisfaction in Example 7. Since $(x_1 = 1) \in (\theta \cap \theta')$, we must have (1) $\sum_{x_i \in S \setminus \{x_1\}} a_i\theta[x_i] \leq \sum_{x_i \in S \setminus \{x_1\}} a_i\theta'[x_i]$, and (2) $\sum_{x_i \in S \setminus \{x_1\}} b_i\theta[x_i] \leq \sum_{x_i \in S \setminus \{x_1\}} b_i\theta'[x_i]$. Therefore, there must be another solution pair $(\tilde{\theta}, \tilde{\theta}')$ such that $\tilde{\theta} = \{x_3 = 1, x_4 = 0\}$ and $\tilde{\theta}' = \{x_3 = 0, x_4 = 1\}$, and the nogood constraint $\neg\theta' \equiv \neg(x_1 = 1 \wedge x_3 = 0 \wedge x_4 = 1)$ is logically and propagation redundant with respect to $\neg\tilde{\theta}' \equiv \neg(x_3 = 0 \wedge x_4 = 1)$.

Overall, the eliminability is checked by examining the form of objective and constraints in a COP, and thus common assignment elimination can be applied mechanically by analyzing the COP in only one pass. Empirical evaluation confirms the benefits of the technique in nogood generation, and we refer interested readers to the paper [34] for more details.

4.3 Exploiting Functional Constraints

The techniques described in Sections 4.1 and 4.2 is restricted to COPs with only objectives and constraints that are all provably *efficiently checkable.* This prevents the use of automatic dominance breaking for COPs with varying objectives and constraints, especially the ones with nested function calls. In this section, we describe a method to exploit functional constraints and their properties to derive sufficient conditions for betterment and implied satisfaction automatically.

Functional expressions are ubiquitous in problem modelling in a high-level modeling language [19, 43]. In practice, however, COPs are usually specified normalized/flattened into a form with only standard constraints.

▶ **Example 13.** Consider the COP in Example 7 with a new objective to minimize $x_1 - 2\max(x_2, x_3) + 8x_4$, which is not efficiently checkable due to the max function. After normalization, the COP can become:

$$\begin{aligned} &\text{minimize } obj \\ &\text{subject to } obj = x_1 - 2y_1 + 8x_4, y_1 = \max(x_2, x_3), \\ &\qquad\qquad y_2 \geq 3, y_2 = 3x_1 + 2x_2 + x_3 + x_4, \\ &\qquad\qquad x_1, x_2, x_3, x_4 \in \{0, 1\}, y_1, y_2, obj \in \mathbb{Z} \end{aligned} \tag{1}$$

Note that y_1 and y_2 are newly introduced intermediate variables, and obj is the objective variable to be minimized. They are functionally defined by $y_1 = \max(x_2, x_3)$, $y_2 = 3x_1 + 2x_2 + x_3 + x_4$, and $obj = x_1 - 2y_1 + 8x_4$ respectively. We call these functional constraints, while $y_2 \geq 5$ is a non-functional constraint.

In this section, 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 [36] and similar to that shown in Example 13, to a problem model. We have a set $C_Y \subseteq C$ of functional constraints in the form of $y = h(x_{i_1}, \ldots, x_{i_k})$, where $h : \mathcal{D}^{\{x_{i_1}, \ldots, x_{i_k}\}} \mapsto \mathcal{D}^{\{y\}}$ is a function mapping any assignment on variables $\{x_1, \ldots, x_k\}$ to a unique assignment on a defined variable $y \in Y$. For ease of presentation, we treat each non-functional constraint $c \in (C \setminus C_Y)$ as a function returning 0/1 and define a (reified) functional constraint $c_b \equiv (b = c(x_{i_1}, \ldots, x_{i_k}))$, where $b \in \{0, 1\}$ is a *reified variable* such that $\bar{\theta}[b] = 0$ iff $\bar{\theta}$ satisfies c[1]. We let C_B denote the set of

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

(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 COP, where $obj \in Y$. 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. 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 $(y = h(x_{i_1}, \ldots, x_{i_k})) \in C_Y$. 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. In the remaining of this section, we consider only functionally valid full assignments in $\mathcal{D}^X$.

Our aim is to find pairs (θ, θ') of partial assignments 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 $\mathcal{D}^X_{\theta'}$, and we propose to check only if a specific $\bar{\theta}$ dominates $\bar{\theta}'$ by utilizing a *mutation mapping* in Definition 5. Since values of variables in $Y \cup B$ are determined by values of variables in Z, we redefine the mutation mapping to incorporate functional constraints as follows.

▶ **Definition 14.** *The* mutation mapping $\mu^{\theta' \to \theta}$ *for two assignments θ, θ' over the 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* $z \in S$*,*
- $\bar{\theta}[z] = \bar{\theta}'[z]$ *for* $z \in Z \setminus S$*,*
- $\bar{\theta}[y] = h(\bar{\theta}[x_{i_1}], \ldots, \bar{\theta}[x_{i_k}])$ *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, the mutation mapping $\mu^{\theta' \to \theta}$ not only "mutates" the θ' component of $\bar{\theta}'$ to become θ, but also assigns the values of variables in $Y \cup B$ accordingly. The following proposition characterizes some useful properties of the mutation mapping.

▶ **Proposition 15.** *The followings are true for all* $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$ *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 is a direct consequence of Theorems 4 and 8.

▶ **Theorem 16.** *If a pair of assignments* $\theta, \theta' \in \mathcal{D}^S$ *satisfies (1) lexicographical ordering:* $\theta <_{lex} \theta'$*, (2) betterment:* $\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \mu^{\theta' \to \theta}(\bar{\theta}')[obj] \leq \bar{\theta}'[obj]$*, and (3) implied satisfaction:* $\forall b \in B, \forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}$*,* $\mu^{\theta' \to \theta}(\bar{\theta}')[b] \leq \bar{\theta}'[b]$*, then* θ *dominates* θ' *with respect to* P*.*

The lexicographical ordering is a constraint for the pair (θ, θ'), while the betterment and implied satisfaction are predicates in the form of quantified inequalities. Our approach is to recursively find sufficient conditions for these quantified inequalities until the conditions without quantification so that they are constraints involving only θ and θ'.

▶ **Example 17.** Consider the COP in Example 13 and suppose we want to find $\theta, \theta' \in \mathcal{D}^S$ where $S = \{x_1, x_2\}$ such that $\theta \prec \theta'$ with respect to P. Let $\bar{\theta}$ denote $\mu^{\theta' \to \theta}(\bar{\theta}')$ for all full assignment $\bar{\theta}' \in \mathcal{D}^S_{\theta'}$. By Theorem 16, it requires: (1) betterment: $\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \bar{\theta}[obj] \leq \bar{\theta}'[obj]$, and (2) implied satisfaction: $\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}$, $\bar{\theta}[b] \leq \bar{\theta}'[b]$, where b is a reified variable defined by $b = (y_2 \geq 3)$. We find sufficient conditions for betterment as follows:

1. Variable *obj* is defined by $obj = x_1 - 2y_1 + 8x_4$. The betterment condition must hold if
$$\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \bar{\theta}[x_1] - 2\bar{\theta}[y_1] + 8\bar{\theta}[x_4] \leq \bar{\theta}'[x_1] - 2\bar{\theta}'[y_1] + 8\bar{\theta}'[x_4] \tag{2}$$
2. Since the summation function is monotonically increasing, (2) must be true if we have
$$\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \bar{\theta}[x_1] \leq \bar{\theta}'[x_1] \wedge -2\bar{\theta}[y_1] \leq -2\bar{\theta}'[y_1] \wedge 8\bar{\theta}[x_4] \leq 8\bar{\theta}'[x_4] \tag{3}$$

3. By Proposition 15, $\bar{\theta}[x_1] = \theta[x_1]$, $\bar{\theta}'[x_1] = \theta'[x_1]$ and $\bar{\theta}[x_4] = \bar{\theta}'[x_4]$, and (3) is implied by

$$(\theta[x_1] \leq \theta'[x_1]) \wedge (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, -2\bar{\theta}[y_1] \leq -2\bar{\theta}'[y_1]) \tag{4}$$

4. Since $y_1 = \max(x_2, x_3)$, and "max" is monotonically increasing. Therefore, the condition

$$(\theta[x_1] \leq \theta'[x_1]) \wedge (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \bar{\theta}[x_2] \geq \bar{\theta}'[x_2] \wedge \bar{\theta}[x_3] \geq \bar{\theta}'[x_3]) \tag{5}$$

   implies (4). By Proposition 15 again, (5) is implied by $(\theta[x_1] \leq \theta'[x_1]) \wedge (\theta[x_2] \geq \theta'[x_2])$ since $x_2 \in S$ while $x_3 \notin S$.

Similarly, we can find the sufficient condition $3\theta[x_1] + 2\theta[x_2] \geq 3\theta'[x_1] + 2\theta'[x_2]$ for implied satisfaction, which is the same as that in Example 7. Overall, if (θ, θ') satisfies

- the lexicographical ordering: $\theta <_{lex} \theta'$
- sufficient conditions for betterment: $(\theta[x_1] \leq \theta'[x_1]) \wedge (\theta[x_2] \geq \theta'[x_2])$, and
- sufficient conditions for implied satisfaction: $3\theta[x_1] + 2\theta[x_2] \geq 3\theta'[x_1] + 2\theta'[x_2]$,

then $\theta \prec \theta'$ with respect to P, and $\neg\theta'$ is a dominance breaking nogood for P. One example solution pair is (θ, θ') such that $\theta = \{x_1 = 0, x_2 = 1\}$ and $\theta' = \{x_1 = 1, x_2 = 0\}$.

The above derivation procedure utilize the functional constraint and properties such as monotonicity to derive sufficient conditions, and it can be formalized and automated as a derivation algorithm. We refer readers to the paper [35] for more details on the derivation algorithm as well as the soundness and termination results of the algorithm. Empirical study in the paper [35] demonstrates its ability to reveal dominance relations in the literature and discover new dominance relations on problems with ineffective or no known dominance breaking constraints.

5 Experimental Evaluation

In this section, we give experimental results on several benchmarks to show the utility of automatic dominance breaking. For more results of empirical study, we refer interested readers to the papers [33, 34, 35].

We use MiniZinc [43] 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 flattened 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 [45] with version 0.10.4. All experiments are run on Xeon E5-2630 2.60GHz processors. We give results on three benchmark problems, 20 random instances for each problem size:

- *Team-n-m*. The Team Assignment Problem appears in MiniZinc Challenge 2018 [53]. 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. The search strategy is to select the variable with the smallest domain and assign the minimum value to it first.

[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	**2-dom**		**3-dom**		**4-dom**	
Problem	Total	Solving	Total	Solving	Total	Solving	Total
Team-6-5	24.48	10.57	**12.49**	9.70	32.00	8.88	427.73
Team-7-5	276.84	138.88	**146.15**	130.71	225.19	150.83	1745.96
Team-8-5	1983.53	819.58	**829.05**	767.52	1024.43	724.63	5191.70
MaxCover-45	75.91	53.47	53.79	5.07	**9.96**	0.27	83.93
MaxCover-50	615.04	464.81	465.53	26.31	**34.92**	1.12	134.99
MaxCover-55	3576.98	2859.60	2860.27	78.37	**91.53**	2.54	199.11
Sensor-50	156.84	138.65	139.44	94.05	**108.99**	57.34	297.18
Sensor-60	595.46	404.27	405.52	269.61	**296.56**	172.43	709.37
Sensor-70	1615.18	1144.17	1145.83	810.01	**854.61**	651.72	1724.70

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

- *MaxCover-n*. The Budgeted Maximum Coverage Problem [29] 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 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.
- *Sensor-n*. The Sensor Placement Problem [32] is 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.

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**). 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 within the timeout.

In Table 1, we report the geometric mean of the problem solving time (Solving) and the total time (Total) for all benchmarks. We first compare the problem solving time to evaluate the usefulness of the generated nogoods, and we can observe that the generated dominance breaking nogoods significantly reduce the solving time in three benchmarks. As the maximum nogood length L increases and more generated nogoods are added to the problem model, the solving time is usually shorter. To compare the total time (generation time + solving time) of ***L*-dom** against **basic**, we highlight the smallest total time in bold. 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 *Team-n-m*, **2-dom** is the best and reduces up to 58.20% total time than **basic**. In *MaxCover-n* and *Sensor-n*, **3-dom** usually comes on top, and the percentage decrease in runtime is up to 97.44% and 80.48% respectively compared with **basic**. The performance gain of ***L*-dom** in problem solving outweighs the generation time in all three problems when the maximum length L of nogoods is set appropriately.

6 Discussion and Future Work

Automatic generation of dominance breaking nogoods is made possible by focusing on nogoods. While nogood constraints are elementary, the propagation of nogood may not be efficient. One direction of future work is to combine the framework with constraint acquisition [4], in which the generated nogoods can be used as examples to learn and discover more succinct constraints. The method of automatic dominance breaking requires a full constraint instance to synthesize generation CSPs, and the automatic detection of dominance relations from constraint models alone is an interesting line of future work. Also, the static generation of dominance breaking nogoods may sometimes take too much time, and not all nogoods are equally important in solving COPs. We expect dynamic detection of dominance relations and its combination with dynamic dominance breaking methods [6, 9] will save the time for the overall solution process.

References

1 Ozgur Akgun, Alan M Frisch, Ian P Gent, Bilal Syed Hussain, Christopher Jefferson, Lars Kotthoff, Ian Miguel, and Peter Nightingale. Automated symmetry breaking and model selection in Conjure. In *Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming*, pages 107–116. Springer, 2013.

2 Ozgur Akgun, Ian P Gent, Christopher Jefferson, Ian Miguel, and Peter Nightingale. Breaking conditional symmetry in automated constraint modelling with CONJURE. In *Proceedings of the 21st European Conference on Artificial Intelligence*, pages 3–8, 2014.

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

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

5 Chiu Wo Choi, Jimmy Ho Man Lee, and Peter J Stuckey. Propagation redundancy in redundant modelling. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming*, pages 229–243. Springer, 2003.

6 Geoffrey Chu, Maria Garcia de la Banda, and Peter J Stuckey. 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*, pages 71–86. Springer, 2010.

7 Geoffrey Chu and Peter J Stuckey. Minimizing the maximum number of open stacks by customer search. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming*, pages 242–257. Springer, 2009.

8 Geoffrey Chu and Peter J Stuckey. A generic method for identifying and exploiting dominance relations. In *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming*, pages 6–22, 2012.

9 Geoffrey Chu and Peter J Stuckey. Dominance driven search. In *Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming*, pages 217–229. Springer, 2013.

10 James M Crawford, Matthew L Ginsberg, Eugene M Luks, and Amitabha Roy. Symmetry-breaking predicates for search problems. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, pages 148–159, 1996.

11 Diego de Una, Graeme Gange, Peter Schachte, and Peter J Stuckey. Weighted spanning tree constraint with explanations. In *Proceedings of the 13th International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 98–107, 2016.

12 Guillaume Derval, Vincent Branders, Pierre Dupont, and Pierre Schaus. The maximum weighted submatrix coverage problem: A cp approach. In *Proceedings of the 16th International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 258–274, 2019.

13 Matteo Fischetti and Domenico Salvagnin. Pruning moves. *INFORMS Journal on Computing*, 22(1):108–119, 2010. Publisher: INFORMS.

14 Matteo Fischetti and Paolo Toth. A new dominance procedure for combinatorial optimization problems. *Operations Research Letters*, 7(4):181–187, 1988. Publisher: Elsevier.

15 Filippo Focacci and Paul Shaw. Pruning sub-optimal search branches using local search. *Proceedings of CP-AI-OR-02*, pages 181–189, 2002.

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

17 Eugene C Freuder. In pursuit of the holy grail. *Constraints*, 2(1):57–61, 1997. Publisher: Springer.

18 Eugene C Freuder. Progress towards the holy grail. *Constraints*, 23(2):158–171, 2018. Publisher: Springer.

19 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. Publisher: Springer.

20 Alan M Frisch, Ian Miguel, and Toby Walsh. CGRASS: A system for transforming constraint satisfaction problems. In *Proceedings of the 2002 Joint ERCIM/CologNet International Conference on Constraint Solving and Constraint Logic Programming*, pages 15–30. Springer, 2002.

21 Graeme Gange and Peter J Stuckey. Sequential precede chain for value symmetry elimination. In *Proceedings of the 24th International Conference on Principles and Practice of Constraint Programming*, pages 144–159. Springer, 2018.

22 Maria Garcia de la Banda, Peter J Stuckey, and Geoffrey Chu. Solving talent scheduling with dynamic programming. *INFORMS Journal on Computing*, 23(1):120–137, 2011. Publisher: INFORMS.

23 Antoine Gargani and Philippe Refalo. 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*, pages 77–89. Springer, 2007.

24 Ian P Gent, Iain McDonald, Ian Miguel, and Barbara M Smith. Approaches to conditional symmetry breaking. In *Proceedings of the 4th International Workshop on Symmetry and Constraint Satisfaction Problems*, 2004.

25 Lise Getoor, Greger Ottosson, Markus Fromherz, and Björn Carlson. Effective redundant constraints for online scheduling. In *Proceedings of the the Fourteenth AAAI National Conference on Artificial Intelligence*, pages 302–307, 1997.

26 Pascal Van Hentenryck, Pierre Flener, Justin Pearson, and Magnus Ågren. Compositional derivation of symmetries for constraint satisfaction. In *Proceedings of the 6th International Symposium on Abstraction, Reformulation, and Approximation*, pages 234–247. Springer, 2005.

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

28 George Katsirelos and Fahiem Bacchus. Generalized nogoods in CSPs. In *Proceedings of the Twentieth AAAI National Conference on Artificial Intelligence*, pages 390–396, 2005.

29 Samir Khuller, Anna Moss, and Joseph Seffi Naor. The budgeted maximum coverage problem. *Information processing letters*, 70(1):39–45, 1999. Publisher: Elsevier.

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

31 Sebastian Kosch and J Christopher Beck. A new MIP model for parallel-batch scheduling with non-identical job sizes. In *Proceedings of the 11th International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 55–70, 2014.

32 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. Publisher: American Society of Civil Engineers.

33 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.

34 Jimmy H. M. Lee and Allen Z. Zhong. Towards more practical and efficient automatic dominance breaking. In *Proceedings of the Thirty-fifth AAAI Conference on Artificial Intelligence*, pages 3868–3876, 2021.

35 Jimmy H. M. Lee and Allen Z. Zhong. Exploiting functional constraints in automatic dominance breaking for constraint optimization. In *Proceedings of the 22nd International Conference on Principles and Practice of Constraint Programming*, 2022.

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

37 Alan K Mackworth and Eugene C Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial intelligence*, 25(1):65–74, 1985. Publisher: Elsevier.

38 Christopher Mears and Maria Garcia de la Banda. Towards automatic dominance breaking for constraint optimization problems. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, 2015.

39 Christopher Mears, M Garcia De La Banda, and Mark Wallace. On implementing symmetry detection. *Constraints*, 14(4):443–477, 2009. Publisher: Springer.

40 Christopher Mears, Maria Garcia de la Banda, Mark Wallace, and Bart Demoen. A novel approach for detecting symmetries in CSP models. In *Proceedings of the 5th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 158–172. Springer, 2008.

41 Christopher Mears, Maria Garcia De La Banda, Mark Wallace, and Bart Demoen. A method for detecting symmetries in constraint models and its generalisation. *Constraints*, 20(2):235–273, 2015.

42 Jean-Noël Monette, Pierre Schaus, Stéphane Zampelli, Yves Deville, and Pierre Dupont. A CP approach to the balanced academic curriculum problem. In *Proceedings of the 7th International Workshop on Symmetry and Constraint Satisfaction Problems*, pages 56–63, 2007.

43 Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. MiniZinc: Towards a standard CP modelling language. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, pages 529–543. Springer, 2007.

44 Thiago F Noronha, Celso C Ribeiro, and Andréa C Santos. Solving diameter-constrained minimum spanning tree problems by constraint programming. *International Transactions in Operational Research*, 17(5):653–665, 2010.

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

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

47 Les Proll and Barbara Smith. Integer linear programming and constraint programming approaches to a template design problem. *INFORMS Journal on Computing*, 10(3):265–275, 1998. Publisher: INFORMS.

48 Jean-François Puget. Automatic detection of variable and value symmetries. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming*, pages 475–489. Springer, 2005.

49 Hu Qin, Zizhen Zhang, Andrew Lim, and Xiaocong Liang. An enhanced branch-and-bound algorithm for the talent scheduling problem. *European Journal of Operational Research*, 250(2):412–426, 2016. Publisher: Elsevier.

50 Arathi Ramani and Igor L Markov. Automatically exploiting symmetries in constraint programming. In *Proceedings of the 2004 Joint ERCIM/CoLOGNET International Conference on Recent Advances in Constraints*, pages 98–112. Springer, 2004.

51 Pierre Roy and François Pachet. Using symmetry of global constraints to speed up the resolution of constraint satisfaction problems. In *Proceedings of ECAI'98 Workshop on non-binary constraints*, 1998.

52 Ilankaikone Senthooran, Pierre Le Bodic, and Peter J Stuckey. Optimising training for service delivery. In *Proceedings of the 27th International Conference on Principles and Practice of Constraint Programming*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

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

54 C-F Yu and Benjamin W. Wah. Learning dominance relations in combinatorial search problems. *IEEE Transactions on Software Engineering*, 14(8):1155–1175, 1988.