Automatic Generation of Dominance Breaking Nogoods for a Class of Constraint Optimization Problems

Jimmy H.M. Lee, Allen Z. Zhong

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

Abstract

Constraint Optimization Problems (COPs) ask for an assignment of values to variables in order to optimize an objective subject to constraints that restrict the value combinations in the assignment. They are usually solved by the classical Branch and Bound (B&B) search algorithm. *Dominance breaking* is an important technique in B&B to prune assignments that are subordinate to others concerning the objective value and/or the satisfiability of constraints. In practice, the addition of constraints for dominance breaking can drastically speed up the B&B search for solving many COPs. However, identification of suboptimal assignments in COPs and derivation of useful constraints for dominance breaking are usually problem-specific and require sophisticated human insights on the problem structure.

This paper proposes the first theoretical and practical framework for automatic generation of dominance breaking constraints for a class of COPs consisting of *efficiently checkable* objectives and constraints. In particular, the framework focuses on generating nogood constraints representing incompatible value assignments and formulates nogood generation as solving auxiliary constraint satisfaction problems. The proposed method can generate nogoods of varying strengths for dominance breaking by controlling the number of involved variables. Experimentation on various benchmarks demonstrates the effectiveness of the proposal in both efficiency and ease of use. The superior performance is also supported by a theoretical analysis to compare the relative strength of automatically generated nogoods with manually derived dominance breaking constraints in the literature.

Keywords: Dominance breaking constraints, Constraint programming, Constraint optimization, Constraint satisfaction

Email addresses: jlee@cse.cuhk.edu.hk (Jimmy H.M. Lee), zwzhong@cse.cuhk.edu.hk (Allen Z. Zhong)

1. Introduction

Constraint Optimization Problems (COPs) are ubiquitous in practice and have many applications including scheduling [1, 2, 3, 4], planning [5, 6], packing [7], transport routing [8, 9], etc. COPs are composed of a set of (discrete) variables, a set of constraints and an objective function. The aim of solving a COP is to find a solution, which is an assignment of values to variables satisfying all constraints, to maximize or minimize its objective function. Constraint Programming [10] is a classical pillar of artificial intelligence and a typical approach for solving COPs. Users can formulate the COP model mathematically and submit the model to a constraint programming solver, which will in turn search for the assignment with the best objective value. Such a problem-solving method represents the Holy Grail of Computer Science [11, 12]: the user states the problem, the computer solves it.

A constraint programming solver usually solves a COP using the Branch and Bound (B&B) search algorithm [13]. The efficiency of the B&B algorithm depends highly on the pruning of the search tree. There are two common kinds of pruning in B&B search in constraint programming. The first kind is the result of *constraint propagation* [14], where values of variables are removed when they cannot be in any solution. The second kind is effected by a *bounding constraint* which is recursively tightened whenever a new best solution is found during search. In the subsequent search, all feasible solutions with a worse objective value of the new bound are pruned. *Dominance breaking* is different from other pruning techniques in the B&B search and can provide a complementary kind of pruning. It requires the identification of useful properties of optimal solutions in order to add constraints to prune suboptimal solutions. The additional constraints are based on the analysis of the problem structure and are usually problem-specific.

Example 1. *Consider the 0-1 knapsack problem where there is a set of items, each with a price and a weight. The problem asks to select a subset of items with the maximal total profit subject to the constraint that the total weight of chosen items cannot exceed the capacity limit W of a knapsack. The following gives an instance with 4 items and $W = 5$, where $x_i = 1$ means that item i is chosen.*

$$\begin{aligned} &\textit{maximize } 3x_1 + x_2 + 6x_3 + 4x_4 \\ &\textit{subject to } x_1 + 2x_2 + 3x_3 + 4x_4 \leq 5 \\ &\qquad x_i \in \{0, 1\} \textit{ for } i = 1, \ldots, 4 \end{aligned} \tag{1}$$

Observe that the first item has 3 units of profit and 1 unit of weight, while the second item has 1 unit of profit and 2 units of weight. Suppose there is a subset T_1 that includes the second item and excludes the first one. We can always replace the second item with the first one to obtain another subset T_2 with less total weight. If T_1 is feasible, then T_2 is also feasible with an objective value at least as good as that of T_1. Therefore, T_1 can be removed without changing the optimal objective value of (1). By a similar reasoning, we can also replace the

fourth item with the third one in the solution. These insights can be reformulated as constraints to remove suboptimal solutions:

$$\begin{aligned} &\textit{maximize } 3x_1 + x_2 + 6x_3 + 4x_4 \\ &\textit{subject to } x_1 + 2x_2 + 3x_3 + 4x_4 \leq 5 \\ &\qquad x_1 \geq x_2, x_3 \geq x_4 \\ &\qquad x_i \in \{0,1\} \textit{ for } i = 1, \ldots, 4 \end{aligned} \tag{2}$$

Note that $x_1 \geq x_2$ and $x_3 \geq x_4$ are dominance breaking constraints that can make a suboptimal solution in (1) become infeasible in (2). For example, the subset choosing the first and the fourth item is a solution for (1), while it violates $x_1 \geq x_2$ and $x_3 \geq x_4$.

As shown in Example 1, constraints for dominance breaking can prune solutions without giving any bounds on the objective value. Instead, some assignments are proved to be suboptimal, since they are *dominated* by others concerning the objective value and the satisfiability of constraints. The additional constraints characterize some useful properties of solutions with the optimal objective value. Dominated solutions violating these dominance breaking constraints will be pruned in the B&B algorithm. A wealth of research works apply dominance breaking in solving COPs in practice, where empirical evidence shows that dominance breaking can dramatically reduce the search space and speed up the solving process [1, 7, 15, 16, 17, 18].

Dominance breaking is powerful, but it usually takes mathematical wit and tricks to identify opportunities and derive constraints for dominance breaking. It is even more difficult to determine the compatibility of dominance breaking constraints, i.e. whether more than one dominance breaking constraints can be added to the problem simultaneously or not. Consider a variant of the knapsack problem in Example 1, where the first two items has one unit of weight and profit. Following the same reasoning, we can also derive either $x_1 \leq x_2$ or $x_1 \geq x_2$ for dominance breaking. However, adding both constraints into the problem will change the optimal objective value. Additional techniques are required to select a subset of dominance breaking constraints to prune as many suboptimal solutions as possible whereas preserving the optimal objective value.

Achieving success in applying dominance breaking to solve complex problems often requires sophisticated insights into the underlying structures of the problem at hand. The methods are usually problem-specific and non-trivial to transfer from one problem domain to another. Chu and Stuckey [19] developed a generic manual method for deriving dominance breaking constraints in Constraint Optimization Problems (COPs). This method requires considerable manual effort to choose mappings that can potentially improve solutions in terms of the satisfaction of constraints and the optimality of the objective under the candidate mappings. Although Mears and Garcia de la Banda [20] automated this process to a large extent by detecting symmetries that map solutions to equivalent solutions, it still requires manual interventions to be effective.

In this paper, we give the theory and practical implementation of *the first fully automatic method to generate dominance breaking constraints for a class of*

COPs. The key difficulty for automation is that dominance breaking constraints can be in different forms for different problems. We propose to generate constraints only in the form of *nogoods* representing incompatible value assignments in COPs. Nogoods are simple to handle and analyze in constraint programming. They are the most basic form of constraints and can be combined to construct arbitrary constraints. In particular, nogood generation for a COP is formulated mechnically as solving constraint satisfaction problems (CSPs) consisting of constraints which are sufficient conditions to ensure that generated nogoods can remove suboptimal assignments and preserve the optimal objective value. These sufficient conditions correspond to the objective and constraints of a target COP. Once they are specified in a system, CSPs for nogood generation can be constructed and solved automatically to obtain nogoods for dominance breaking. In this way, the task of deriving dominance breaking constraints by non-expert users is transformed into finding sufficient conditions for various kind of objectives and constraints by system developers. To demonstrate the effectiveness of our proposed method, we provide such sufficient conditions for a class of *efficiently checkable* objectives and constraints in this paper, and implement them using the MiniZinc modelling language [21]. Experimentation on various problems, including those with no known dominance breaking constraints in the literature, exhibit runtime improvement of up to three orders of magnitude against the baseline methods in constraint programming. What's more, we provide theoretical comparison between automatically generated nogoods and manually derived dominance breaking constraints in the literature to explain the empirical performance.

Compared with the preliminary version of this work that appeared in IJCAI-PRICAI 2020 [22], the current paper is improved in several aspects:

- Most importantly, we give a theoretical analysis on the strength of the automatically generated nogoods for various problems by comparing them with dominance breaking constraints in the literature in Section 7.
- Secondly, we present more detailed explanation on using lexicographical ordering constraint to ensure that adding automatically generated nogoods preserve at least one optimal solution in Section 4.4.
- Next, we give more efficiently checkable constraints and identify their sufficient conditions for implied satisfaction in Section 4.3.
- Lastly, we include two more benchmarks, the combinatorial auction and the set covering problems, as well as a real-life benchmark with industrial instances in the empirical evaluation of Section 6.

2. Background

In this section, we give preliminaries in COPs, relational mathematics as well as dominance breaking in COPs.

2.1. Constraint Satisfaction and Optimization

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. If $(x_i = v_i) \in \theta$, then $\theta[x_i] = v_i$ is the value assigned by θ to a variable $x_i \in var(\theta)$.

Let $\mathcal{D}^S$ denote the set of all assignments with the scope $S \subseteq X$. A *full assignment* is an assignment whose scope is X, while a *partial assignment* has a scope $S \subset X$. We use $\bar{\theta}$ to emphasize a full assignment. The *projection* $\theta{\downarrow_{S'}}$ *of an assignment* $\theta \in \mathcal{D}^S$ *onto* $S' \subseteq S$ is a partial assignment such that $\theta{\downarrow_{S'}}[x] = \theta[x]$ for all $x \in S'$. Let $\mathcal{D}^X_\theta = \{\bar{\theta} \in \mathcal{D}^X \mid \bar{\theta}{\downarrow_S} = \theta\}$ denote the set of full assignments extending from a partial assignment $\theta \in \mathcal{D}^S$.

A constraint $c \in C$ is a set of assignments over the variables $var(c)$. An assignment θ *satisfies* a constraint c if and only if $\theta{\downarrow_{var(c)}} \in c$, where $var(c) \subseteq var(\theta)$. A *solution* of a CSP $P = (X, D, C)$ 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 $\neg\theta \equiv \vee_{x \in var(\theta)}(x \neq \theta[x])$. The *length* of a nogood connstraint is equal to the scope size $|var(\theta)|$.

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.

2.2. Basic Definitions in Relational Mathematics

Given two sets M and N, a binary relation R over M and N is a set of ordered pairs (m, n) where $m \in M$ and $n \in N$. The set M is the *domain* of R and N is the *codomain* of R. A *mapping* $R : M \mapsto N$ is a binary relation that associates to every element of M exactly one element of N. A mapping R is a *bijection* iff (1) R is *surjective*, i.e., $\forall n \in N, \exists m \in M$ such that $(m, n) \in R$, and (2) R is *injective*, i.e., $\forall (m, n), (m', n') \in R, (m \neq m') \Rightarrow (n \neq n')$. We also write mRn when $(m, n) \in R$.

A relation R is a *homogeneous relation* over a set M if its domain and codomain are the same. Otherwise, it is a *heterogeneous relation*. A homogeneous relation R is *transitive* when $\forall m, m', m'' \in M$, if $(m, m') \in R$ and $(m', m'') \in R$, then $(m, m'') \in R$, and is irreflexive iff $\forall m \in M, (m, m) \notin R$.

2.3. Dominance Relations in COPs

Following Chu and Stuckey [19], we formalize the concept of dominance as a homogeneous relation over the set of all full assignments in a COP.

Definition 1. *[19] A* dominance relation $\prec$ with respect to $P = (X, D, C, f)$ *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.*

Note that the three cases in Definition 1 are only necessary but not sufficient conditions. A dominance relation must also be transitive and irreflexive to ensure that all dominated assignments can be pruned. Consider a simple COP with one variable x and $D(x) = \{0, 1\}$, where the objective function is constant. There are two full assignments $\bar{\theta} = \{x = 0\}$ and $\bar{\theta}' = \{x = 1\}$. If a dominance relation $\prec$ is constructed such that $\bar{\theta} \prec \bar{\theta}'$ and $\bar{\theta}' \prec \bar{\theta}$ simultaneously, then both $\bar{\theta}$ and $\bar{\theta}'$ are pruned, and the problem becomes unsatisfiable. However, no such dominance relation exists by definition. By the transitive property, $\bar{\theta} \prec \bar{\theta}'$ and $\bar{\theta}' \prec \bar{\theta}$ implies that $\bar{\theta} \prec \bar{\theta}$, but it is in conflict with the irreflexive property.

The next theorem follows directly from Definition 1.

Theorem 1. *[19] 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 $\exists \bar{\theta} \in \mathcal{D}^X$ such that $\bar{\theta} \prec \bar{\theta}'$, without changing the satisfiability or the optimal value of P.*

Theorem 1 captures the characteristic of dominance breaking technique. In standard B&B search, the objective value of the current best solution is used as a bound. Feasible solutions with objective value worse than the bound are pruned during search. On the other hand, dominance relations guarantee that dominated solutions can be pruned due to the relative magnitude of their objective compared with some other dominating solutions, even though the actual objective values of dominating solutions are unknown in advance. Such dominance relations are exploited to derived dominance breaking constraints before search as a preprocessing step.

3. Related Works

We review the research work on dominance breaking techniques.

3.1. Identification of Dominance Relations

There have been many works on exploiting dominance relations to boost the B&B search algorithm for solving constraint optimization problems. To the best of our knowledge, dominance relations are first characterized and defined in the context of permutation problems [23]. Ibaraki [24] proves formally that exploiting dominance relations in the B&B algorithm can result in a smaller search tree for the B&B algorithm, which justifies the effectiveness of dominance breaking techniques theoretically.

Research work exploiting dominance relations are mostly problem-specific. With sophisticated insights of problem structures, one could either reformulate COPs with additional *dominance breaking constraints* [1, 17, 18] or augment the B&B search algorithm with *dominance rules* [7, 15, 16, 25, 3, 4] to exclude

dominated assignments. Dominance relations are usually defined, either explicitly or implicitly, in the soundness proofs for dominance breaking techniques in different problems. A wealth of empirical evidence has shown that dominance breaking can dramatically reduce the search space and speed up the B&B solving process, but there is little discussion on how to generalize dominance relations in one problem domain to another.

Early work on the identification of dominance relations attempts to find dominance relations using machine learning [26]. While the method can be applied to a class of problems, the generated candidate dominance relations lack correctness guarantees and requires further manual inspection. Chu and Stuckey [19] give the first generic method for deriving dominance breaking constraints for COPs. Their method starts with candidate mappings over the solution set that possibly map solutions to solutions, followed by the derivation of dominance breaking constraints based on sufficient conditions regarding the satisfaction of constraints and the optimality of the objective under the candidate mappings. Manual efforts are required to select mappings, identify sufficient conditions, and simplify the dominance breaking constraints. Mears and de la Banda [20] later automate the process to a certain extent based on symmetry detection [27]. However, their method still requires manual selection of candidate mappings to be effective. Our work in this paper follows this line of work and gives a method to fully automate the derivation of dominance breaking constraints with soundness guarantees.

3.2. Automatic Methods for Dominance Breaking

There are several automatic methods for dominance breaking when the class of dominance relations are restricted.

Symmetry breaking [28] exploits symmetry relations to avoid searching for symmetrically equivalent solutions and is an important technique in constraint programming. Symmetry relations are bijective mappings on full assignments that have the same objective value and are both solutions/non-solutions, while dominance is a relation on also two full assignments, but one is "better" than the other in terms of satisfiability or objective values. Static symmetry breaking can be considered as a special case of dominance breaking by introducing, for example, the lexicographic ordering on symmetric solutions. Given two symmetric solutions $\bar{\theta}$ and $\bar{\theta}'$ in a COP where $\bar{\theta}$ is lexicographically smaller than $\bar{\theta}'$. We can consider $\bar{\theta}$ to dominate $\bar{\theta}'$ in a dominance relation with respect to the COP. Considerable progress has been made in the automatic detection of symmetry relations in CSPs [29, 30, 31, 32, 33, 34]. Our proposed method can also generate symmetry breaking constraints for a class of COPs.

Lazy Clause Generation [35, 36] and Automatic Caching via Constraint Projection [37, 38] are dynamic methods for dominance breaking. Both methods exploit dominance relations during the B&B search. Lazy clause generation derives a nogood constraint by analyzing the process of constraint filtering that leads to failures, so it mainly focuses on dominance relations when the dominating parts are non-solutions. Automatic Caching via Constraint Projection avoids the exploration of the current subproblem by finding conditions such that

it is dominated by an explored subproblem during the B&B search. In contrast, we propose a static method to derive nogoods dominance breaking before the runtime of the B&B algorithm. None of these methods exploits all possible dominance relations in COPs. They can be used simultaneously to prune more suboptimal solutions in the search space [39].

When the class of problems is restricted to mixed integer linear programs, Fischetti et al. [40, 41] propose a *local dominance procedure* to exploit dominance relations. The idea is to identify a dominating node for a given search node by solving a structured auxiliary optimization problem, either in an exact or heuristic manner, during the B&B search. Some design choices are required for efficient implementation, such as heuristic selection of nodes to solve the auxiliary problem and recording nogoods to prevent redundant solving.

4. Automatic Generation of Dominance Breaking Nogoods

In this section, we give an automated method to identify dominance breaking constraints in the form of nogoods. The idea is to formulate the generation of dominance breaking nogoods as solving auxiliary generation CSPs, which aim to find pairs (θ, θ') of partial assignments of a given COP P such that $\neg\theta'$ is a constraint to remove suboptimal assignments in P. We start with an example generation CSP for the COP in Example 1.

Example 2. *Consider the COP in (1) and a pair of partial assignments* $\theta = \{x_1 = v_1, x_2 = v_2\}$ *and* $\theta' = \{x_1 = v'_1, x_2 = v'_2\}$ *over the same scope, where* $v_1, v_2, v'_1, v'_2 \in \{0, 1\}$ *are unknown integers. Let* σ *be a mapping for* θ *and* θ'*, where a full assignment* $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$ *is mapped to* $\bar{\theta} = \sigma(\bar{\theta}') \in \mathcal{D}^X_{\theta}$ *such that*

$$\bar{\theta}[x_3] = \bar{\theta}'[x_3] \text{ and } \bar{\theta}[x_4] = \bar{\theta}'[x_4]. \tag{3}$$

Based on the objective and constraints of (1), the generation CSP for θ *and* θ' *is formulated as follows.*

$$\begin{aligned} 3v_1 + v_2 &\geq 3v'_1 + v'_2 \\ v_1 + 2v_2 &\leq v'_1 + 2v'_2 \\ v_1 \neq v'_1 &\vee v_2 \neq v'_2 \end{aligned} \tag{4}$$

We claim that if θ *and* θ' *satisfy (4), then all assignments in* $\mathcal{D}^X_{\theta'}$ *can be removed without changing the optimal value of (1). In particular, we show this statement by constructing a relation* $\prec$ *over* $\mathcal{D}^X$ *such that* $\sigma(\bar{\theta}') \prec \bar{\theta}'$ *for all* $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$ *and proving that* $\prec$ *is a dominance relation.*

Since $v_1 \neq v'_1 \vee v_2 \neq v'_2$*, we have* $\mathcal{D}^X_{\theta} \cap \mathcal{D}^X_{\theta'} = \emptyset$*, which implies that* $\prec$ *is transitive and irreflexive by construction. In addition, because* $3v_1 + v_2 \geq 3v'_1 + v'_2$*, for all* $\bar{\theta}' \in \mathcal{D}^X_{\theta}$*, we have*

$$\begin{aligned} & 3v_1 + v_2 \geq 3v'_1 + v'_2 \\ \Leftrightarrow \quad & 3\bar{\theta}[x_1] + \bar{\theta}[x_2] \geq 3\bar{\theta}'[x_1] + \bar{\theta}'[x_2] \\ \Leftrightarrow \quad & 3\bar{\theta}[x_1] + \bar{\theta}[x_2] + 6\bar{\theta}[x_3] + 4\bar{\theta}[x_4] \geq 3\bar{\theta}'[x_1] + \bar{\theta}'[x_2] + 6\bar{\theta}'[x_3] + 4\bar{\theta}'[x_4] \end{aligned}$$

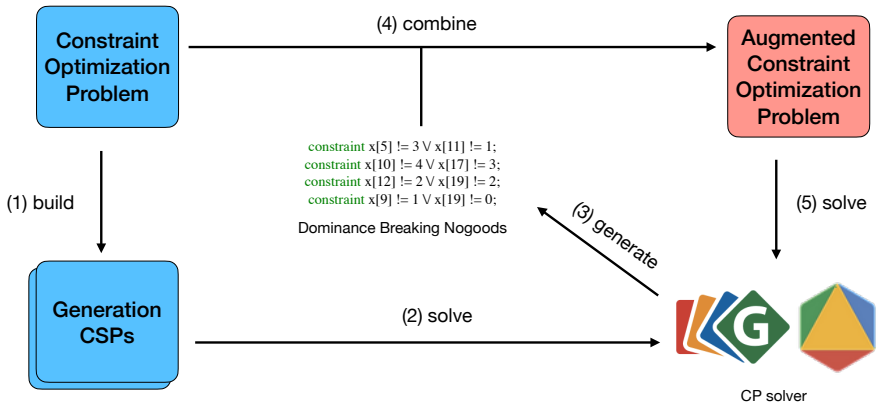


Figure 1: Workflow of Automatic Dominance Breaking for COPs

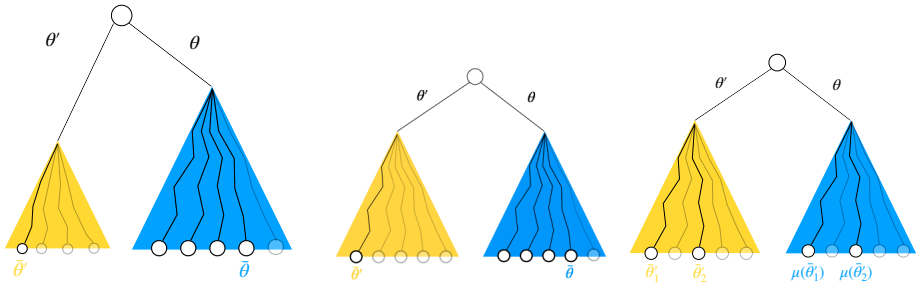
The last step holds due to (3). By similar reasoning, if $v_1 + 2v_2 \leq v'_1 + 2v'_2$, *then* $\bar{\theta}'$ *is a solution implies that* $\bar{\theta}$ *is also a solution. In other words, the full assignment* $\bar{\theta}'$ *and its image* $\sigma(\bar{\theta}')$ *must satisfy either one of the three cases in Definition 1, and thus* $\prec$ *is a dominance relation.*

Because $\prec$ *is a dominance relation, any full assignment* $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$ *can be pruned by Theorem 1 without changing the optimal value of (1). One solution of (4) is* $v_1 = v'_2 = 1$ *and* $v_2 = v'_1 = 0$, *which corresponds to partial assignments* $\theta = \{x_1 = 1, x_2 = 0\}$ *and* $\theta' = \{x_1 = 0, x_2 = 1\}$. *To prune all full assignments in* $\mathcal{D}^X_{\theta'}$, *we can simply add a dominance breaking nogood* $\neg\theta' \equiv (x_1 \neq 0 \vee x_2 \neq 1)$ *to the original COP. We can construct generation CSPs and generate dominance breaking nogoods for other pairs of partial assignments similarly.*

Example 2 demonstrate how to construct a generation CSP for a pair of partial assignments. Our method constructs multiple generation CSPs and generates dominance breaking constraints to remove suboptimal assignments by the following workflow (Figure 1):

1. Given a COP P, analyze the objective and constraints of P and construct auxiliary generation CSPs for identifying dominance breaking nogoods.
2. Enumerate 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 augmented with extra nogoods by a constraint solver.

Note that as long as generation CSPs are constructed automatically, the workflow can be automated and coordinated by a simple program. In the following subsections, we provide detailed theoretical explanations on how to construct generation CSPs for a class of COPs.



(a) Exhaustive enumeration (b) Restriction of same scope (c) Using the mutation mapping

Figure 2: Restriction for checking a subset of pairs of partial assignments

4.1. Dominance Relations on Partial Assignments

Dominance relations over full assignments (Definition 1) can be generalized to partial assignments as follows.

Definition 2. *Let θ, θ' be two partial assignments of a COP $P = (X, D, C, f)$. If $\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}$, $\exists \bar{\theta} \in \mathcal{D}^X_{\theta}$ such that $\bar{\theta} \prec \bar{\theta}'$, then we say $\theta \prec \theta'$.*

The following theorem is a direct consequence of Theorem 1 and Definition 2.

Theorem 2. *Let $P = (X, D, C, f)$ be a COP. If two constraints θ and θ' satisfy that $\theta \prec \theta'$ with respect to P, then P has the same satisfiability or optimal value as $P' = (X, D, C \cup \{\neg\theta'\}, f)$.*

Once we establish the dominance relation $\theta \prec \theta'$, the negation of θ' is the desired *dominance breaking nogood* for P. However, checking all possible pairs by Definition 2 and generating all dominance breaking nogoods will be prohibitively expensive, our approach is to focus on a subset of nogoods that can be generated efficiently. An arbitrary pair of partial assignments in P may involve different number of variables or different sets of variables (Fig 2a). Our first restriction is to focus on pairs of partial assignments over the same scope, which will reduced the number of checking substantially (Fig 2b). Still, checking $\theta \prec \theta'$ needs to find one full assignment in $\mathcal{D}^X_{\theta}$ for each full assignments $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$, which requires nested for-loops in the straightforward implementation. Instead of exhaustive enumeration, we further define the *mutation mapping* $\mu^{\theta' \mapsto \theta}$ for θ and θ' and restrict our attention to check whether a full assignment $\bar{\theta}'$ is dominated by its image $\mu^{\theta' \mapsto \theta}(\bar{\theta}')$ (Fig 2c).

Definition 3. *Let $\theta, \theta' \in \mathcal{D}^S$ be two assignments in a COP P over the same scope S. The mutation mapping $\mu^{\theta' \mapsto \theta} : \mathcal{D}^X_{\theta'} \mapsto \mathcal{D}^X_{\theta}$ maps a full assignment $\bar{\theta}' \in \mathcal{D}^{\theta'}$ to another full assignment $\bar{\theta} \in \mathcal{D}^X_{\theta'}$ 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 $\mu^{\theta' \mapsto \theta}(\bar{\theta}')$ "mutates" the value assigned by θ' in $\bar{\theta}'$ to that assigned by θ. For convenience of presentation, we let $\bar{\theta} = \mu^{\theta' \mapsto \theta}(\bar{\theta}')$ when it is clear from the context.

Example 3. *Consider the two assignments θ and θ' in Example 2. We list all full assignments $\bar{\theta}' \in \mathcal{D}_{\theta}^{X}$ and their images under $\mu^{\theta' \mapsto \theta}$ as follows:*

$\bar{\theta}'$					$\bar{\theta}$			
x_1	x_2	x_3	x_4		x_1	x_2	x_3	x_4
v_1	v_2	0	0	$\mapsto$	v_1'	v_2'	0	0
v_1	v_2	0	1	$\mapsto$	v_1'	v_2'	0	1
v_1	v_2	1	0	$\mapsto$	v_1'	v_2'	1	0
v_1	v_2	1	1	$\mapsto$	v_1'	v_2'	1	1

where the "mutated" parts are highlighted in color.

The following theorem gives a sufficient condition for $\theta \prec \theta'$ based on $\mu^{\theta' \mapsto \theta}$.

Theorem 3. *Let $P = (X, D, C, f)$ be a COP where θ and θ' are two partial assignments in P. If the mutation mapping $\mu^{\theta' \mapsto \theta}$ satisfies:*

- *unequal: $\theta \neq \theta'$,*
- *betterment: $\forall \bar{\theta}' \in \mathcal{D}_{\theta'}^{X}$, $f(\bar{\theta}) \leq f(\bar{\theta}')$, and*
- *implied satisfaction: $\forall \bar{\theta}' \in \mathcal{D}_{\theta'}^{X}$, $\bar{\theta}' \in sol(P)$ implies that $\bar{\theta} \in sol(P)$,*

where $\bar{\theta} = \mu^{\theta' \mapsto \theta}(\bar{\theta}')$, then $\theta \prec \theta'$ with respect to P.

Proof. The proof idea is inspired by Chu and Stuckey [19, 39]. We construct a relation $\prec$ over $\mathcal{D}^X$ such that $\bar{\theta} \prec \bar{\theta}'$ if and only if $\bar{\theta}' \in \mathcal{D}_{\theta'}^{X}$ and $\mu^{\theta' \mapsto \theta}(\bar{\theta}') = \bar{\theta}$. Note that $\prec$ is trivially transitive and irreflexive by the unequal condition $\mathcal{D}_{\theta}^{X} \cap \mathcal{D}_{\theta'}^{X} = \emptyset$. If $\bar{\theta}' \prec \bar{\theta}$, then $\bar{\theta} \not\prec *$ where $*$ is an arbitrary full assignment in $\mathcal{D}_{\theta'}^{X}$.

What remains is to show that for all $\bar{\theta}' \in \mathcal{D}_{\theta'}^{X}$, if $\bar{\theta} \prec \bar{\theta}'$, then $\bar{\theta}$ and $\bar{\theta}'$ will satisfy either one of the three cases in Definition 1:

- Suppose $\bar{\theta}' \notin sol(P)$. If $\bar{\theta} \in sol(P)$, then $\bar{\theta}$ and $\bar{\theta}'$ fulfill the first case in Definition 1. Otherwise, $\bar{\theta} \notin sol(P)$. Since we have the betterment condition, $f(\bar{\theta}) \leq f(\bar{\theta}')$, which fulfills the third case in Definition 1.
- Suppose $\bar{\theta}' \in sol(P)$. By implied satisfaction, $\bar{\theta}$ is also a solution. What's more, $f(\bar{\theta}) \leq f(\bar{\theta}')$ by the betterment condition. Therefore, $\bar{\theta}$ and $\bar{\theta}'$ fulfill the first case in Definition 1.

Therefore, $\prec$ is a dominance relation over $\mathcal{D}^X$ by Definition 1, and θ dominates θ' with respect to P. □

The sufficient condition in Theorem 3 allows us to consider the objective and constraints of a COP P separately to establish $\theta \prec \theta'$ with respect to P. What's more, a full assignment is a solution if it satisfies all constraints of P.

Therefore, we can further consider implied satisfaction for each constraint of P separately.

A generation CSP contains constraints over θ and θ' for unequal, betterment and implied satisfaction. The unequal condition in Theorem 3 simply requires that at least one variable is assigned to different values in θ and θ'. To model betterment and implied satisfaction in a generation CSP, we give their sufficient conditions for several classes of *efficiently checkable* objectives and constraints in Sections 4.2 and 4.3 respectively. As long as such sufficient conditions are included into a system, generation CSPs can be constructed mechanically by the system for a COP consisting of an efficiently checkable objective and efficiently checkable constraints.

Note that our method can also generate nogoods for symmetry breaking. Suppose a pair (θ, θ') of partial assignments satisfying conditions in Theorem 3. A full assignment $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$ is mapped to $\bar{\theta} \in \mathcal{D}^X_{\theta}$ such that $\bar{\theta}$ and $\bar{\theta}'$ have the same objective value and satisfiability of constraints. In other words, $\mathcal{D}^X_{\theta'}$ and $\mathcal{D}^X_{\theta}$ are symmetric to each other. The nogood $\neg\theta'$ is a symmetry breaking constraint. As shown by Flener et al. [42], multiple symmetry breaking constraints can conflict and remove all solutions in CSPs. In optimization problems, we need to preserve the optimal objective value of a COP when adding additional dominance breaking constraints. We will also discuss the compatibility of generated nogoods in Section 4.4.

Unless otherwise stated, all formal results in the remainder of this section are given within the following context: *Let $P = (X, D, C, f)$ be a COP. Suppose $\theta, \theta' \in \mathcal{D}^S$ are two assignments over the same scope, and their mutation mapping is $\mu^{\theta' \mapsto \theta} : \mathcal{D}^X_{\theta'} \mapsto \mathcal{D}^X_{\theta}$.*

4.2. Betterment for Efficiently Checkable Objectives

Now we give sufficient conditions for betterment to hold for $\mu^{\theta' \mapsto \theta}$, namely $\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'},\ f(\bar{\theta}) \leq f(\bar{\theta}')$ where $\bar{\theta} = \mu^{\theta' \mapsto \theta}(\bar{\theta}')$. The key idea is to define the *projection function* $f{\downarrow_S}$ of the objective function f onto S so that the relative magnitude of $f(\bar{\theta})$ and $f(\bar{\theta}')$ can be determined. In the following, we consider two types of efficiently checkable objectives: separable and supermodular/submodular functions.

4.2.1. Separable Objectives

A function f is *separable* if it can be written as a linear combination of functions of individual variables, i.e., $f(\bar{\theta}) = f_1(\bar{\theta}[x_1]) + \cdots + f_n(\bar{\theta}[x_n])$, where each component is $f_i : \mathbb{Z} \mapsto \mathbb{R}$. We define the projection function $f{\downarrow_S}(\theta) = \sum_{x_i \in S} f_i(\theta[x_i])$ for a partial assignment $\theta \in \mathcal{D}^S$.

Theorem 4. *Let f be a separable function. If $f{\downarrow_S}(\theta) \leq f{\downarrow_S}(\theta')$, then $f(\bar{\theta}) \leq f(\bar{\theta}')$ for all full assignments $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$, where $\bar{\theta} = \mu^{\theta' \mapsto \theta}(\bar{\theta}')$.*

Proof. For each full assignment $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$ and $\bar{\theta} = \mu^{\theta' \mapsto \theta}(\bar{\theta}') \in \mathcal{D}^X_{\theta}$, we have

$$f(\bar{\theta}') = f{\downarrow_S}(\theta') + \sum_{x_i \notin S} f_i(\bar{\theta}'[x_i]) \text{ and } f(\bar{\theta}) = f{\downarrow_S}(\theta) + \sum_{x_i \notin S} f_i(\bar{\theta}[x_i]).$$

By Definition 3, $\bar{\theta}[x_i] = \bar{\theta}'[x_i]$ for all $x_i \notin S$. Therefore, $\sum_{x_i \notin S} f_i(\bar{\theta}'[x_i]) = \sum_{x_i \notin S} f_i(\bar{\theta}[x_i])$. Thus, $f{\downarrow_S}(\theta) \leq f{\downarrow_S}(\theta')$ implies that $f(\bar{\theta}) \leq f(\bar{\theta}')$. □

4.2.2. Supermodular and Submodular Objectives

A *supermodular function* is a set function $g : 2^V \mapsto \mathbb{R}$ that assigns each subset $U \subseteq V$ a value $g(U) \in \mathbb{R}$ such that

$$g(U \cup T) - g(U) \leq g(U' \cup T) - g(U')$$

for every $U, U' \subseteq V$ where $U \subseteq U'$ and $T \subseteq V \backslash U'$. In a binary COP $P = (X, D, C, f)$ where $D(x) = \{0, 1\}$ for all variable $x \in X$, an assignment $\theta \in \mathcal{D}^S$ can be associated with a set $U(\theta) = \{i \mid \theta[x_i] = 1\}$. We say that the objective function f of P *is equivalent to a supermodular function* g if $f(\bar{\theta}) = g(U(\bar{\theta}))$. Similarly, we define the projection function $f{\downarrow_S}(\theta) = g(U(\theta))$.

Theorem 5. *Let f be a function which is equivalent to a supermodular function $g : 2^V \mapsto \mathbb{R}$ in a binary COP. If $f{\downarrow_S}(\theta) \leq f{\downarrow_S}(\theta')$ and $U(\theta) \subseteq U(\theta')$, then $f(\bar{\theta}) \leq f(\bar{\theta}')$ for all full assignments $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$, where $\bar{\theta} = \mu^{\theta' \mapsto \theta}(\bar{\theta}')$.*

Proof. By definition of a supermodular function, we have

$$g(U(\theta) \cup T) - g(U(\theta)) \leq g(U(\theta') \cup T) - g(U(\theta')) \tag{5}$$

for $U(\theta) \subseteq U(\theta') \subseteq V$ and any set $T \subseteq V \backslash U(\theta')$. Let $T = \{i \mid \bar{\theta}'[x_i] = 1 \wedge x_i \notin S\}$ for a full assignment $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$. Since f is a function which is equivalent to g, we have $f(\bar{\theta}') = g(U(\bar{\theta}')) = g(U(\theta') \cup T)$. By Definition 3, $\bar{\theta}[x_i] = \bar{\theta}'[x_i]$ for all variables $x_i \notin S$. We also have $f(\bar{\theta}) = g(U(\bar{\theta})) = g(U(\theta) \cup T)$. Therefore, equation (5) becomes

$$\begin{aligned} & f(\bar{\theta}) \leq f(\bar{\theta}') - g(U(\theta')) + g(U(\theta)) \\ \equiv & f(\bar{\theta}) \leq f(\bar{\theta}') - f{\downarrow_S}\,(\theta') + f{\downarrow_S}\,(\theta) \end{aligned}$$

Since $f{\downarrow_S}\,(\theta) \leq f{\downarrow_S}\,(\theta')$, the above inequality implies that $f(\bar{\theta}) \leq f(\bar{\theta}')$. □

A function g is *submodular* if $-g$ is supermodular. Thus minimizing a supermodular function is equivalent to maximizing a submodular function.

Corollary 1. *Let f be a function which is equivalent to a submodular function $g : 2^V \mapsto \mathbb{R}$ in a binary COP. If $f{\downarrow_S}(\theta) \geq f{\downarrow_S}(\theta')$ and $U(\theta) \subseteq U(\theta')$, then $f(\bar{\theta}) \geq f(\bar{\theta}')$ for all full assignments $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$.*

The proof of Corollary 1 is similar to that of Theorem 5. We note that Theorems 3 and 4 can also be easily adapted for maximization.

4.3. Implied Satisfaction for Efficiently Checkable Constraints

Now we consider sufficient conditions for the implied satisfaction in Theorem 3. Note that a full assignment $\bar{\theta}$ is a solution of P iff $\bar{\theta}$ satisfies all constraints $c \in C$. If for all $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$ and $c \in C$, $\bar{\theta}'$ satisfies c implies that $\bar{\theta}$ satisfies c, then the implied satisfaction also hold. This suggests considering each constraint $c \in C$ separately. Furthermore, the following theorem states that we only need to focus on a subset of constraints.

Theorem 6. *If $var(c) \cap S = \emptyset$ for a constraint $c \in C$, a full assignment $\bar{\theta}' \in \mathcal{D}_{\theta'}^X$ always has the same feasibility as $\bar{\theta} \in \mathcal{D}_{\theta}^X$ with respect to c.*

Proof. Since $var(c) \cap S = \emptyset$, we have $x \notin S$ for all $x \in var(c)$. By Definition 3, $\bar{\theta}'[x] = \bar{\theta}[x]$ for all $x \notin S$, and therefore we have $\bar{\theta}'{\downarrow_{var(c)}} = \bar{\theta}{\downarrow_{var(c)}}$. Thus, ${\bar{\theta}'}{\downarrow_{var(c)}} \in c$ if and only if ${\bar{\theta}'}{\downarrow_{var(c)}} \in c$. In other words, $\bar{\theta}'$ has the same feasibility as $\bar{\theta}$ with respect to c. □

By Theorem 6, we consider only constraint $c \in C$ where $var(c) \cap S \neq \emptyset$ from now on. We say a partial assignment θ is *applied to* $c \in C$ by replacing every occurrence of $x \in var(c) \cap S$ by value $\theta[x]$. The resulting constraint $c\theta$ has scope $var(c) \backslash S$. The following proposition states that $c\theta \Rightarrow c\theta'$ is a sufficient condition to prove implied satisfaction.

Proposition 1. *If $c\theta' \Rightarrow c\theta$ holds for a constraint $c \in C$, then $\bar{\theta}'$ satisfies $c \Rightarrow \bar{\theta}$ satisfies c for all $\bar{\theta}' \in \mathcal{D}_{\theta'}^X$ and $\bar{\theta} = \mu^{\theta' \mapsto \theta}(\bar{\theta}')$.*

Proof. Let $S' = var(c) \setminus S$. By Definition 3, $\bar{\theta}'[x] = \bar{\theta}[x]$ for $x \notin S$, which implies that ${\bar{\theta}'}{\downarrow_{S'}} = {\bar{\theta}}{\downarrow_{S'}}$. Since $c\theta' \Rightarrow c\theta$, $c\theta' \subseteq c\theta$, which means that ${\bar{\theta}'}{\downarrow_{S'}} \in c\theta'$ implies that ${\bar{\theta}}{\downarrow_{S'}} \in c\theta$. Thus, we have $\bar{\theta}'$ satisfies $c \Rightarrow \mu^{\theta' \mapsto \theta}(\bar{\theta}')$ satisfies c. □

4.3.1. Unary Constraints

A unary constraint restricts the values that are valid for a single variable $x \in X$, which are usually presolved and cast into the domain constraint of the form $c = (x \in D(x))$. The condition for implied satisfaction is straightforward.

Theorem 7. *Let c be a domain constraint of the form $(x \in D(x))$. If $\theta[x] \in D(x)$, then $c\theta' \Rightarrow c\theta$.*

Proof. Since $\theta[x] \in D(x)$, $c\theta$ is always true, which implies that $c\theta' \Rightarrow c\theta$ always hold. □

Note that if $\theta'[x] \notin D(x)$, then all full assignments $\bar{\theta}'$ in $\mathcal{D}_{\theta'}^X$ are trivially non-solutions of P. An extra nogood constraint $\neg\theta'$ is redundant in a propagation solver. Therefore, we would add $\theta[x] \in D(x)$ and $\theta'[x] \in D(x)$ in the generation CSP for all variables $x \in S$ to avoid generate redundant nogoods.

4.3.2. Linear Inequality Constraints

A linear inequality constraint is $c \equiv (\sum w_i x_i \leq b)$ where $w_i, b \in \mathbb{R}^+$. The sufficient condition for implied satisfaction is stated as follows.

Theorem 8. *Let c be a linear inequality constraint $c \equiv (\sum w_i x_i \leq b)$ and $S' = S \cap var(c)$. If $\sum_{x_i \in S'} w_i \theta[x_i] \leq \sum_{x_i \in S'} w_i \theta'[x_i]$, then $c\theta' \Rightarrow c\theta$.*

Proof. By definition, since $\sum_{x_i \in S'} w_i \theta[x_i] \leq \sum_{x_i \in S'} w_i \theta'[x_i]$, we have

$$
\begin{aligned}
c\theta' &\equiv \Big(\sum_{x_i \in (var(c) \backslash S)} w_i x_i \leq b - \sum_{x_i \in S'} w_i \theta'[x_i] \Big) \\
&\Rightarrow \Big(\sum_{x_i \in (var(c) \backslash S)} w_i x_i \leq b - \sum_{x_i \in S'} w_i \theta[x_i] \Big) \quad \equiv c\theta
\end{aligned}
$$

□

Note that if $\sum_{x_i \in S'} w_i \theta'[x_i] > b$, then any full assignment $\bar{\theta} \in \mathcal{D}_\theta^X$ must not satisfy the linear inequality constraint $c \equiv (\sum w_i x_i \leq b)$ and is trivially a non-solution of P. Therefore, we also add a constraint $\sum_{x_i \in S'} w_i \theta'[x_i] \leq b$ to the generation CSPs.

4.3.3. Boolean Disjunction Constraints

The *Boolean disjunction constraint* $c \equiv (\vee_{x_i \in B} x_i)$ requires at least one Boolean variable in the set B takes the true value.

Theorem 9. *Let c be a Boolean disjunction constraint $(\vee_{x_i \in B} x_i)$ and $S' = S \cap B$. If $\vee_{x_i \in S'} \theta'[x_i] \Rightarrow \vee_{x_i \in S'} \theta[x_i]$, then $c\theta' \Rightarrow c\theta$.*

Proof. Since $e' = \vee_{x_i \in S'} \theta'[x_i] \Rightarrow \vee_{x_i \in S'} \theta[x_i] = e$, either e' and e are both true, or e' is evaluated to false. In the former case, $c\theta'$ and $c\theta$ are always true, which implies that $c\theta' \Rightarrow c\theta$ always holds. As for the latter case, $c\theta' \equiv (\vee_{x_i \in S \setminus S'} x_i)$, and thus $c\theta \equiv e \vee (\vee_{x_i \in S \setminus S'} x_i)$. Therefore, $c\theta' \Rightarrow c\theta$ regardless of the value of e. □

Boolean disjunction constraints can also be extended where Boolean variables are results of binary comparisons $(x \circ b)$ where $x \in X$ is a variable, $b \in \mathbb{Z}$ is an integer and $\circ \in \{=, \neq, \geq, \leq, <, >\}$ is a binary comparison operator.

Theorem 10. *Let c be a Boolean disjunction constraint $\vee_{x_i \in B}(x_i \circ_i b_i)$ where $\circ_i \in \{=, \neq, \geq, \leq, <, >\}$ is a binary comparison and $b_i \in \mathbb{Z}$, and $S' = S \cap B$. If $\vee_{x_i \in S'}(\theta[x_i] \circ_i b_i) \Rightarrow \vee_{x_i \in S'}(\theta'[x_i] \circ_i b_i)$, then $c\theta' \Rightarrow c\theta$.*

The proof of Theorem 10 is similar to that of Theorem 9.

4.3.4. Counting Constraints

Counting constraints restrict the number of occurrences of some values in the set V within a given scope S of variables. The Global Constraint Catalog [43] has the keyword "*Counting Constraints*" under which there are 39 different global constraints, among which the *alldifferent* constraint, the *among* constraint and the *global cardinality* constraint are famous examples.

In this section, we first consider two basic constraints $atleast(T, V, k)$ and $atmost(T, V, k)$ where $T \subseteq X$ is a set of variables, V is a set of values, and $k \in \mathbb{N}$ is an integer. The two constraints require that the number of variables in T that take values in V is at least and at most k respectively.

Theorem 11. *Let c be a constraint of the form $atleast(T, V, k)$ and $S' = T \cap S$. If we have $|\{x \mid \theta[x] \in V \wedge x \in S'\}| \geq |\{x \mid \theta'[x] \in V \wedge x \in S'\}|$, then $c\theta' \Rightarrow c\theta$.*

Proof. The $atleast(T, V, k)$ constraint can be expressed as

$$atleast(T, V, k) \equiv (k \leq |\{x \mid \theta[x] \in V \wedge x \in T\}|)$$

Let $d = |\{x \mid \theta[x] \in V \wedge x \in S'\}|$ and $d' = |\{x \mid \theta'[x] \in V \wedge x \in S'\}|$. Since $T = (T \cap S) \cup (T \setminus S)$ and $d \geq d'$, we have

$$\begin{aligned} c\theta' &\equiv (k \leq d' + |\{x|(x = v) \wedge v \in V \wedge x \in T \setminus S\}|) \\ &\Rightarrow (k \leq d + |\{x|(x = v) \wedge v \in V \wedge x \in T \setminus S\}|) \\ &\equiv c\theta \end{aligned}$$

□

We have a similar result for the constraint $atmost(T, V, k)$.

Theorem 12. *Let c be a constraint of the form $atmost(T, V, k)$ and $S' = T \cap S$. If we have $|\{x \mid \theta[x] \in V \wedge x \in S'\}| \leq |\{x \mid \theta'[x] \in V \wedge x \in S'\}|$, then $c\theta' \Rightarrow c\theta$.*

The proof is similar to that of Theorem 11. We can also avoid generate redundant nogood constraint $\neg\theta'$ for a partial assignment θ' violating $atmost(T, V, k)$. Therefore, we can add a constraint $|\{x \mid \theta'[x] \in V \wedge x \in S'\}| \leq k$ to the generation CSPs.

By Theorems 11 and 12, we can derive the sufficient conditions of implied satisfaction for different counting constraints. The first famous example of counting constraints is the *alldifferent* constraint [44], where $alldifferent(T)$ enforces that all variables in a set T take distinct values. We can treat it as the conjunction $\wedge_{v \in V}\texttt{atmost}(T, \{v\}, 1)$ where $V = \cup_{x \in T} D(x)$ is the union of domains of variables in T. Following Theorem 12, the sufficient condition for implied satisfaction are given as follows.

Theorem 13. *Let c be a constraint of the form $alldifferent(T)$ where $T \subseteq X$. If $\{\theta[x] \mid x \in T \cap S\} \subseteq \{\theta'[x] \mid x \in T \cap S\}$, then $c\theta' \Rightarrow c\theta$.*

Proof. The alldifferent constraint can be expressed as $alldifferent(\{x \mid x \in T \setminus S\} \cup \{x \mid x \in T \cap S\})$. Since $\{\theta[x] \mid x \in T \cap S\} \subseteq \{\theta'[x] \mid x \in T \cap S\}$, we have

$$\begin{aligned} c\theta &= alldifferent(\{x \mid x \in T \setminus S\} \cup \{\theta[x] \mid x \in T \cap S\}) \\ &\Rightarrow alldifferent(\{x \mid x \in T \setminus S\} \cup \{\theta'[x] \mid x \in T \cap S\}) \\ &= c\theta' \end{aligned}$$

Thus, $c\theta' \Rightarrow c\theta$. □

The *alldifferent_except_0* constraint [43] is a generalization of the *alldifferent* constraint where all variables in a set T are required to take distinct values except for those variables that are assigned value 0. The sufficient condition is also related to the set of assigned values of θ and θ' to variables in T.

Theorem 14. *Let c be the constraint $alldifferent\_except\_0(T)$ where $T \subseteq X$. If $\{\theta[x] \mid x \in T \cap S \wedge \theta[x] \neq 0\} \subseteq \{\theta'[x] \mid x \in T \cap S \wedge \theta'[x] \neq 0\}$, then $c\theta' \Rightarrow c\theta$.*

The proof idea is similar to that of Theorem 13.

Another example is the *among* constraint [45, 46], where $among(T, V, k)$ takes a set T of variables, a set V of values and an integer $k \in \mathbb{N}$ as arguments.

The constraint requires that $k = |\{x \mid x \in T \wedge x = v \wedge v \in V\}|$ and can be expressed as the conjunction $atleast(T, V, k) \wedge atmost(T, V, k)$. Thus, we have the following result for $among(T, V, k)$.

Corollary 2. *Let c be the constraint $among(T, V, k)$ where $T \subseteq X$ is a set of variables, V is a set of values and $k \in \mathbb{N}$ is an integer. If $|\{x \mid x \in T \wedge \theta[x] \in V\}| = |\{x \mid x \in T \wedge \theta'[x] \in V\}|$, then $c\theta' \Rightarrow c\theta$.*

The global cardinality constraint [47] is a generalization of both *alldifferent* and *among* constraint. A global cardinality constraint $GCC(T, U)$ takes two arguments where T is a set of variables, U is a set of triples (v_j, l_j, u_j). For each triple in U, value v_j should be taken by at least l_j and at most u_j variables in T. The *alldifferent* constraint is simply a global cardinality constraint where each value can be taken at most once. Note that $GCC(T, U)$ is also a conjunction of *atleast* and *atmost* constraints:

$$GCC(T, U) \equiv \bigwedge_{(v_j, l_j, u_j) \in U} (atleast(T, \{v_j\}, l_j) \wedge atmost(T, \{v_j\}, u_j))$$

Therefore, we have the following result for $GCC(T, U)$.

Corollary 3. *Let c be the constraint $GCC(T, U)$ where $T \subseteq X$ is a set of variables, and U is a set of tuples (v_j, l_j, u_j). If $|\{x \mid x \in S \cap T \wedge \theta[x] = v_j\}| = |\{x \mid x \in S \cap T \wedge \theta[x] = v_j\}|$ for all tuple $(v_j, l_j, u_j) \in U$, then $c\theta' \Rightarrow c\theta$.*

4.3.5. Circuit Constraint

The *circuit* constraint [48, 49] is useful in various graph problems such as the famous Travelling Salesman Problem (TSP). It constrains an array of variables representing successors of each node on a graph, which requires that the resulting edges to form a Hamiltonian cycle. Formally, suppose $G = (V, E)$ is a graph where $|V| = n$. We have one variable x_i for each node $i \in V$ representing the successor node after visiting node i. The constraint $circuit(x_1, \ldots, x_n)$ requires that there is a cyclic permutation $y_1, \ldots, y_n$ of $1 \ldots n$ such that

$$y_{i+1} = x_{y_i}, i = 1, \ldots, n-1$$
$$y_1 = x_{y_n}$$

The sufficient conditions for implied satisfaction require the introduction of additional variables for the path represented by θ and θ'.

Theorem 15. *Let c be a constraint $circuit(x_1, \ldots, x_n)$ and $S' = S \cap \{x_1, \ldots, x_n\}$. If $y_1, \ldots, y_{|S'|+1}$ and $y'_1, \ldots, y'_{|S'|+1}$ are two sets of introduced variables such that:*

(a) *$alldifferent(y_1, \ldots, y_{|S'|+1})$ and $alldifferent(y'_1, \ldots, y'_{|S'|+1})$,*
(b) *$\forall x_i \in S', \exists j \in 1, ..., |S'|, y_j = i \wedge y_{j+1} = \theta[x_i]$,*
(c) *$\forall x_i \in S', \exists j \in 1, ..., |S'|, y'_j = i \wedge y'_{j+1} = \theta'[x_i]$,*
(d) *$y_1 = y'_1$ and $y_{|S|+1} = y'_{|S|+1}$,*

then $c\theta'$ implies $c\theta$.

Proof. Conditions (a) to (d) implies that θ and θ' forms two paths traversing the same set of nodes where they start from the same node $y_1 = y'_1$ and end at the same node $y_{|S'|+1} = y'_{|S'|+1}$. For any partial assignments over $X \setminus S$ that satisfies $c\theta'$, it must form a tour starting from $y_{|S|+1}$ and ending at y_1, which must also be a solution for $c\theta$. Hence, $c\theta' \Leftrightarrow c\theta$. □

4.4. Compatibility of Dominance Breaking Nogoods

Theorems 2 and 3 only consider the soundness of adding one nogood constraint into P. Recall that our method enumerates all pairs (θ, θ') of partial assignments that are solutions of generation CSPs. All drived dominance breaking nogoods $\neg\theta'$ are added to the COP P for search space pruning. It is necessary to ensure that all nogoods are also *compatible* in the sense that not all optimal solutions of P are eliminated.

Example 4. *Consider a variant of the COP in (1) where the first two items has two units of weight and profit. Following the same procedure as Example 2, we can construct a generation CSP for a pair of partial assignments* $\theta = \{x_1 = v_1, x_2 = v_2\}$ *and* $\theta' = \{x_1 = v'_1, x_2 = v'_2\}$ *as follows:*

$$\begin{aligned} 2v_1 + 2v_2 &\geq 2v'_1 + 2v'_2 \\ 2v_1 + 2v_2 &\leq 2v'_1 + 2v'_2 \\ v_1 \neq v'_1 &\vee v_2 \neq v'_2 \end{aligned} \tag{6}$$

where $v_1, v_2, v'_1, v'_2 \in \{0, 1\}$ *are unknown integers. Solving (6) can result in two possible solutions: either (1)* $\theta_1 = \{x_1 = 1, x_2 = 0\}$ *and* $\theta'_1 = \{x_1 = 0, x_2 = 1\}$*, or (2)* $\theta_2 = \{x_1 = 0, x_2 = 1\}$ *and* $\theta'_2 = \{x_1 = 1, x_2 = 0\}$*. Adding both* $\neg\theta'_1$ *and* θ'_2 *into the COP will eliminate all optimal solutions and change the optimal objective value.*

In this section, we propose to add extra constraints in generation CSPs to avoid generating incompatible nogoods. We first show that the lexicographical ordering between θ and θ' is a sufficient condition to ensure the compatibility of generated nogoods. Sometimes, the lexicographical ordering constraint is too restrictive. We further generalize the lexicographical ordering to obtain more relaxed sufficient conditions for compatibility, which will result in generating more dominance breaking nogoods.

4.4.1. Lexicograhpical ordering for Partial Assignments

Given two tuples $(a_1, \ldots, a_n), (b_1, \ldots, b_n) \in \mathbb{R}^n$, we say that $(a_1, \ldots, a_n)$ is *lexicographically smaller than* $(b_1, \ldots, b_n)$, denoted by $(a_1, \ldots, a_n) <_{lex} (b_1, \ldots, b_n)$, if and only if $\exists j \in \{1, \ldots, n\}$ such that $a_j < b_j$ and $\forall j' < j, a_{j'} = b_{j'}$. Recall that an assignment θ is a tuple in $\mathcal{D}^S$. We are interested in preserving the *lexicographically smallest optimal solution* which is an optimal solution of P and is lexicographically smallest among all optimal solutions of P.

Theorem 16. *If* θ *and* θ' *are two partial assignments such that* $\theta <_{lex} \theta'$*, and* (θ, θ') *satisfies betterment and implied satisfaction, then the lexicographically smallest optimal solution always satisfies* $\neg\theta'$*.*

Proof. Suppose $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$ is an optimal solution of P and $\bar{\theta}' = \mu^{\theta' \mapsto \theta}(\bar{\theta}) \in \mathcal{D}^X_{\theta'}$. By Definition 3, $\bar{\theta}[x] = \bar{\theta}'[x]$ for all variables $x \notin S$, and $\theta <_{lex} \theta'$ implies that $\bar{\theta}[x] <_{lex} \bar{\theta}'[x]$. Since θ and θ' satisfy betterment and implied satisfaction, $\bar{\theta}'$ is an optimal solution implies that $\bar{\theta}$ is also an optimal solution. In other words, if there is an optimal solution $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$, there must be another optimal solution $\bar{\theta}$ which is lexicographically smaller. By contraposition, the lexicographically smallest optimal solution does not belong to $\mathcal{D}^X_{\theta'}$ and always satisfy the nogood constraint $\neg\theta'$. □

Theorem 16 implies that if we enforce $\theta <_{lex} \theta'$ in the generation CSPs, the generated nogoods $\neg\theta'$ will not remove the lexicographically smallest optimal solution. This ensures that all generated nogoods $\neg\theta'$ derived from the solutions of the generation CSPs are compatible, which means adding all nogoods to the COP preserves the optimal value of P. We note the idea of using the *lexicograhpical ordering* as the tiebreaker also appears in previous works [19, 40].

4.4.2. Generalized lexicograhpical ordering

The lexicograhpical ordering between θ and θ' is a sufficient condition for that at least one optimal solution is preserved, but sometimes the condition is too strong such that it eliminates useful solutions of the generation CSP.

Example 5. *Consider the COP in (1) again. We want to find a pair of partial assignments θ and θ' which are over the same scope $S = \{x_i, x_j\}$ and satisfies sufficient conditions in Theorems 4, 8 and 16:*

$$\begin{array}{ll} \textit{unequal:} & \theta[x_i] \neq \theta'[x_i] \vee \theta[x_j] \neq \theta'[x_j] \\ \textit{betterment:} & a_i\theta[x_i] + a_j\theta[x_j] \leq a_i\theta'[x_i] + a_j\theta'[x_j] \\ \textit{implied satisfaction:} & b_i\theta[x_i] + b_j\theta[x_j] \geq b_i\theta'[x_i] + b_j\theta'[x_j] \\ \textit{compatibility:} & \theta <_{lex} \theta' \end{array} \tag{7}$$

The only pair that satisfies conditions in (7) is that $\theta_1 = \{x_2 = 0, x_3 = 1\}$ and $\theta'_1 = \{x_2 = 1, x_3 = 0\}$. However, we can observe that $\theta_2 = \{x_1 = 1, x_2 = 0\}$ and $\theta'_2 = \{x_1 = 0, x_2 = 1\}$ is also a pair that satisfies conditions in Theorem 3. The nogood constraint $\neg\theta'_2$ is compatible with $\neg\theta'_1$, and therefore we can add both constraints to P while preserving the optimal solution $\bar{\theta}_{opt} = \{x_1 = 1, x_2 = 0, x_3 = 0\}$.

To discover additional dominance breaking nogoods through the resolution of generation CSPs, we propose a more relaxed sufficient condition for the compatibility of the produced nogoods based on the concept of the *generalized lexicographical ordering*. Unlike strictly enforcing $\theta <_{lex} \theta'$ as before, the generalized lexicographical ordering incorporates *sensitive functions* over $\mathcal{D}^X$. A function f defined over $\mathcal{D}^X$ is *sensitive* if for any $S \subseteq X$, there exists a projection function $f{\downarrow_S}$ defined over $\mathcal{D}^S$ such that $f{\downarrow_S}(\theta) < f{\downarrow_S}(\theta')$. We have the following sufficient conditions for the generalized lexicographical ordering.

Theorem 17. *Suppose θ, θ' are two partial assignments such that*

$$(f_1{\downarrow_S}(\theta), \ldots, f_m{\downarrow_S}(\theta)) <_{lex} (f_1{\downarrow_S}(\theta'), \ldots, f_m{\downarrow_S}(\theta')),$$

where $f_1, \ldots, f_m$ are sensitive functions, and the pair (θ, θ') satisfies betterment and implied satisfaction. There is at least one optimal solution satisfying $\neg\theta'$.

Proof. Without loss of generality, assume that $m = 1$. Let $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$ an optimal solution of P and $\bar{\theta} = \mu^{\theta' \mapsto \theta}(\bar{\theta}') \in \mathcal{D}^X_{\theta}$. Since the pair (θ, θ') satisfies betterment and implied satisfaction, $\bar{\theta}$ must also be an optimal solution. Also, f_1 is a sensitive function, i.e., $f_1{\downarrow_S}(\theta) < f_1{\downarrow_S}(\theta')$ implies that $f_1(\bar{\theta}) < f_1(\bar{\theta}')$. In other words, if there is an optimal solution $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$, there must be another optimal solution $\bar{\theta} \in \mathcal{D}^X_{\theta}$ such that $f_1(\bar{\theta}) < f_1(\bar{\theta}')$. By contraposition, the optimal solution with the smallest value of f_1 among the optimal solutions does not belong to $\mathcal{D}^X_{\theta'}$ and always satisfies the nogood constraint $\neg\theta'$. The proof generalizes trivially to the case when $m > 1$. □

Note that we can use arbitrary sensitive functions in Theorem 17 as long as their projection functions are well-defined. The generalized lexicograhpical ordering degenerates to the lexicograhpical ordering if we use *element functions* $e_1, \ldots, e_n$ where $e_i(\bar{\theta}) = \bar{\theta}[x_i]$. The projection function is defined as

$$e_i{\downarrow_S}(\theta) = \begin{cases} \theta[x_i] & \text{if } x_i \in S \\ 0 & \text{otherwise} \end{cases}$$

By Theorem 17, the sufficient condition for compatibility is that

$$(e_1{\downarrow_S}(\theta), \ldots, e_n{\downarrow_S}(\theta)) <_{lex} (e_1{\downarrow_S}(\theta'), \ldots, e_n{\downarrow_S}(\theta')),$$

which can be simplified into $\theta <_{lex} \theta'$ since $e_i{\downarrow_S}(\theta) = e_i{\downarrow_S}(\theta') = 0$ when $x_i \notin S$.

As shown in Example 5, however, the lexicographical ordering constraint is a too strong sufficient condition for compatibility. In practice, we also use sensitive functions arising from the objective of a COP P. It is easy to verify that separable objectives and supermodular objectives are sensitive with appropriate definitions of their projection functions.

Proposition 2. *Let f be a separable function and the projection function of f be $f{\downarrow_S}(\theta) = \sum_{x_i \in S} f_i(\theta[x_i])$. The separable function f is sensitive.*

Proposition 3. *Let f be a function that is equivalent to a supermodular function g and the projection function of f be $f{\downarrow_S}(\theta) = g(S(\theta))$ where $S(\theta) = \{i \mid \theta[x_i] = 1\}$. The supermodular function f is sensitive.*

Example 6. *Consider Example 5 again. If we replace the compatibility condition in (7) by:*

compatibility: $(a_i\theta[x_i] + a_j\theta[x_j], \theta[x_i], \theta[x_j]) <_{lex} (a_i\theta[x'_i] + a_j\theta[x'_j], \theta[x'_i], \theta[x'_j]),$

then solving the CSP will give us the following solution pairs:

- $\theta_1 = (0, 1)$ *and* $\theta'_1 = (1, 0)$ *where* $var(\theta_1) = var(\theta'_1) = \{x_2, x_3\}$
- $\theta_2 = (1, 0)$ *and* $\theta'_2 = (0, 1)$ *where* $var(\theta_2) = var(\theta'_2) = \{x_1, x_2\}$
- $\theta_3 = (1, 0)$ *and* $\theta'_3 = (0, 1)$ *where* $var(\theta_3) = var(\theta'_3) = \{x_1, x_3\}$

The derived nogood constraints $\neg\theta'_1$, $\neg\theta'_2$ *and* $\neg\theta'_3$ *are all compatible in the COP.*

We can also define sensitive functions arising from linear inequality constraints and counting constraints.

Proposition 4. *Let* c *be a linear inequality constraint* ($\sum w_i x_i \leq b$). *If we define* $f(\bar{\theta}) = \sum w_i \bar{\theta}[x_i]$ *for a full assignment* $\bar{\theta} \in \mathcal{D}^X$ *and the projection function* $f{\downarrow_S}(\theta) = \sum_{x_i \in S} w_i \theta[x_i]$, *then* f *is sensitive.*

Proposition 5. *Let* c *be a constraint* $atmost(T, V, k)$. *If we define* $f(\bar{\theta}) = |\{x \mid \bar{\theta}[x] \in V\}|$ *for a full assignment* $\bar{\theta} \in \mathcal{D}^X$ *and the projection function* $f{\downarrow_S}(\theta) = |\{x \mid \theta[x] \in V \wedge x \in S \cap T\}|$, *then* f *is sensitive.*

Proposition 6. *Let* c *be a constraint* $atleast(T, V, k)$. *If we define* $f(\bar{\theta}) = -|\{x \mid \bar{\theta}[x] \in V\}|$ *for a full assignment* $\bar{\theta} \in \mathcal{D}^X$ *and the projection function* $f{\downarrow_S}(\theta) = -|\{x \mid \theta[x] \in V \wedge x \in S \cap T\}|$, *then* f *is sensitive.*

Recall that our goal is to find more relaxed sufficient conditions for compatibility. In the actual implementation, we usually construct the tuples in Theorem 17 using the sensitive function derived from the objective function, followed by sensitive functions arising from the linear inequality and counting constraints, and finally the element functions for tie-breaking.

5. Modeling Nogood Generation as Constraint Satisfaction

Sections 4 states the sufficient conditions in which $\theta \prec \theta'$ with respect to P, and $\neg\theta'$ can be added to P for dominance breaking. Our proposal is applicable for a COP P as long as the objective and all constraints in P are all efficiently checkable. Generating nogoods with all possible lengths can be computationally intractable. In this section, we propose several implementation techniques to further improve the solving efficiency.

First, we limit the scope size of partial assignments for generating nogood constraints. The following theorem states the complexity result when using the simple generate-and-test method to find all dominance breaking nogoods of certain length.

Proposition 7. *Let* $P = (X, D, C, f)$ *be a COP and* $l \in \mathbb{N}$ *be a positive integer. Suppose* $\max(|D(x_i)|) = d$ *for all variables* $x_i \in X$, *there are* $O(\binom{|X|}{l} \cdot \binom{d^l}{2})$ *pairs of partial assignments* θ *and* θ' *where* $\theta \neq \theta'$ *and* $|S| = l$.

Proof. The candidate pairs can be enumerated by first selecting l variables $x_{i_1}, \ldots, x_{i_l}$ from X and then select two distinct tuples from $D(x_{i_1}) \times \cdots \times D(x_{i_l})$, which has $\binom{\prod_{k=1,\ldots,l} |D(x_{i_k})|}{2} \leq \binom{d^l}{2}$ ways in total. Hence, there are totally $O(\binom{|X|}{l} \cdot \binom{d^l}{2})$ such candidate pairs. □

The total number of pairs grows polynomially with respect to the variable number $|X|$ and the maximum domain size $\max(|D(x_i)|)$, but grows exponentially with respect to the maximum length of dominance breaking nogoods. To ensure the efficiency of our method, we limit the maximum scope size $|S|$ to a fixed integer L and attempt to generate and augment the COP P with nogoods of length $l \leq L$ so that the time for nogood generation is less than the potentially exponential time for solving constraint optimization problems.

Second, we combine multiple generation CSPs for scopes of the same size into a single generation model. As shown in Example 5, sufficient conditions for betterment (Theorems 4 to 5), implied satisfaction (Theorems 7 to 15) and compatibility (Theorems 16 and 17) are nothing but constraints on values assigned by the pair of partial assignment when the scope is fixed. Therefore, enumerating all dominance breaking nogoods of length l requires solving $O(\binom{|X|}{l})$ CSPs in total. We observe that generation CSPs over scopes of the same size have the same type of constraints except that parameters are different. They can be combined by utilizing element constraints [50] in constraint programming.

Figure 3 gives example models for the 0-1 knapsack problem and the generation CSP in the MiniZinc language [21], where Figure 3a is a problem model with n items and a knapsack with the capacity limit W. Lines 1 to 4 include the parameters of the problems, while line 5 declares an array of binary variables, each for one item. Lines 7 and 9 are the objective and the linear inequality constraint respectively. The MiniZinc model for nogood generation is given in Figure 3b. Lines 6 to 9 uses three arrays to model a pair (θ, θ') of partial assignments with the same scope, where we use an array F to represent the indices set of variables in the common scope. The variable arraies $v1$ and $v2$ represent the assigned values in θ and θ' respectively. Thus, if $\exists i \in \{1, \ldots, l\}$ such that $F[i] = k$, then $x_k \in S$, $\theta[x_k] = v1[k]$ and $\theta[x_k] = v2[k]$. Note that there are variable symmetries in the array F, and thus the constraint in line 11 enforces that $F[i] < F[i+1]$ for all $i = 1, \ldots, l-1$. Lines 13 to 19 state two constraints which are the sufficient conditions for implied satisfaction and betterment by Theorem 4 and 8 respectively. The last constraint is derived from Theorem 17 and is to ensure the compatibility between the generated dominance breaking nogoods.

Third, we further add constraints to avoid generate redundant nogoods. Note that a longer nogood constraint can be redundant with respect to a shorter one in a COP.

Proposition 8. *If $\tilde{\theta}'$ and θ' are two partial assignments in a COP P such that $var(\tilde{\theta}') \subset var(\theta')$ and $\tilde{\theta}'[x] = \theta'[x]$ for all variables $x \in var(\tilde{\theta}')$, then $\neg\tilde{\theta}' \Rightarrow \neg\theta'$.*

Proposition 8 is easy to verify and implies that $\neg\theta'$ is logically redundant with respect to $\neg\tilde{\theta}'$ if $\tilde{\theta}' \subset \theta'$. Note that a nogood constraint $c \equiv \neg\theta'$ is usually enforced to be *generalized arc consistent (GAC)* [51] A logically redundant nogood constraint is also *propagation redundant* [52] and contributes no extra pruning in a constraint solver. To avoid generating redundant dominance breaking nogoods, we solve generation CSPs for nogoods with increasing lengths. For

```
int: n; % number of items
int: W; % knapsack capacity
array [1..n] of int: p;  % profits of items
array [1..n] of int: w;  % weights of items
array [1..n] of var 0..1: x;

solve maximize sum(i in 1..n)(p[i] * x[i]);

constraint sum(i in 1..n)(w[i] * x[i]) <= W;
```

(a) The Problem Model

```
int: n; % number of items
int: W; % knapsack capacity
array [1..n] of int: p;  % profits of items
array [1..n] of int: w;  % weights of items

int: l;
array [1..l] of var 1..n: F;
array [1..l] of var 0..1: v1;
array [1..l] of var 0..1: v2;

constraint increasing(F);

var int: p1 = sum(i in 1..l)(p[F[i]] * v1[i]);
var int: p2 = sum(i in 1..l)(p[F[i]] * v2[i]);
constraint p1 >= p2;

var int: w1 = sum(i in 1..l)(w[F[i]] * v1[i]);
var int: w2 = sum(i in 1..l)(w[F[i]] * v2[i]);
constraint w1 <= w2;

constraint lex_less([-p1, w1]++v1,[-p2, w2]++v2);
```

(b) The Model of Generation CSPs

Figure 3: Models for the 0-1 knapsack problem in MiniZinc

each solution $(\tilde{\theta}, \tilde{\theta}')$, we add one constraint

$$\vee_{x_i \in var(\tilde{\theta}')}(x_i \notin var(\theta') \vee \theta'[x_i] \neq \tilde{\theta}'[x_i]) \tag{8}$$

into all generation CSPs with length $l > |var(\tilde{\theta}')|$, so that (θ, θ') will correspond to a nogood $\neg\theta'$ that is not redundant with respect to $\neg\tilde{\theta}'$.

In the actual implementation, we modify the publicly available MiniZinc compiler to analyze the compiled FlatZinc model of a COP. Each constraint/objective in the FlatZinc model corresponds to $O(1)$ constraints in the generation CSP. Such a generation CSP can be constructed mechanically by analyzing the problem in only one pass in negligible time as compared to the time for nogood generation and problem-solving.

6. Empirical Evaluation

This section aims to demonstrate the practical applicability of our automatic dominance breaking method for solving COPs. To model the problems, we use the MiniZinc modelling language[21], and employ Chuffed[35], a widely used constraint programming solver, as the backend solver for both nogood generation and problem-solving. It's worth noting that generating nogoods by solving the generation CSPs and handling additional nogoods while solving a COP may introduce overheads. However, we believe that the efficiency gained by the extra pruning justifies the additional overhead, and we'll show that the benefits of automatic dominance breaking can significantly reduce the overall solving time of constraint solvers. We've made the source code and benchmarks available at `https://github.com/AllenZzw/auto_dom/tree/dominance`.

Our method, called L**-dom**, generates and augments problem models with nogoods of length 2 up to L. To evaluate our method, we compare it against two other methods: the basic problem model (**no-dom**) and the model with manual dominance breaking constraints (**manual**). We use a fixed search heuristic specified in the problem model and also the default free search option, which alternates between the fixed search heuristics and the variable state independent decaying sum search heuristic [53], in the Chuffed solver for each problem. In addition, we include results obtained using the **MIP** method with the CBC 2.10 [54] as the backend solver for reference.

The timeout for the whole solving process (nogood generation + problem-solving) is set to 2 hours, while we reserve 1 hour for nogood generation. If nogood generation times out, we augment the problem model with only the nogoods generated so far. Otherwise, the problem is solved with the remaining time. All experiments are run on Xeon(R) Platinum 8268 2.90GHz using MiniZinc 2.6.0.

6.1. Standard Benchmarks

The standard benchmarks consist of 6 problems, where there are 20 instances for each instance group:

- *Knapsack-n*: the 0-1 knapsack problem is to maximize a *linear objective* subject to a *linear inequality constraint*. The problem ask to select a subset of items where each item i is associated with it profit p_i and weight w_i. We use instances from an online repository[1] where the number of items $n = 100, 150, 200, 250, 300$. We use the fixed search heuristic to select an item with the highest p_i/w_i first. The **manual** method uses dominance breaking constraints in Definition 4.

- *DisjKnapsack-n*: the disjunctively constrained knapsack problem [55] is an extension of the knapsack problem with additional Boolean disjunction constraints. For each instance with n items, we augment the instance by randomly picking $\lfloor \eta n(n-1)/2 \rfloor$ incompatible pairs of items where $\eta = 0.2\%$. The fixed search heuristic is the same as that of *Knapsack*. The **manual** method uses dominance breaking constraints in Definition 5.

- *ConcertSchd-n*: the capacitated concert hall scheduling problem [56] is to maximize a *separable objective* subject to *alldifferent_except_0* constraints. The problem considers a set of concerts each having a start time s_a, an end time e_a, a profit p_a and a required capacity r_a. A concert a can only be placed into a hall with capacity c_h such that $c_h \geq r_a$. We follow Gange and Stuckey [56] to generate random instances with 10 halls and n applications where $n = 20, 25, 30, 35, 40$, with $1 \leq s_a \leq e_a \leq 100$, $200 \leq r_a, c_h \leq 1000$ and $\frac{p_a}{e_a - s_a + 1} \geq 10$. The fixed search heuristic is to include a concert with the highest ratio $\frac{p_a}{e_a - s_a + 1}$ first. The **manual** method uses dominance breaking constraints in Definition 6.

- *MaxCut-n*: the maximum cut problem is to maximize a *submodular* function on a graph whose vertices are partitioned into two complementary set S and T. For $n = 30, 35, 40, 45, 50$, we generate random graphs with n vertices by independently sampling each edge with probability $p = 0.1$ whose weights are integers from 1 to 10. The fixed search heuristic is to include vertices into S in order of their indices. The **manual** method uses dominance breaking constraints in Definition 7.

- *CombAuc-n*: the combinatorial auction problem is to maximize a *linear objective* subject to *linear inequality constraints*. We generate random instances using the scheme of Balas and Ho [57], where there are $m = 100$ items and $n = 100, 150, 200, 250, 300$ bids in the instances. The value of a bidder is selected from $[1, 100]$. The probability of an item being covered by a bid is set to 5%. The fixed search heuristic is to include the next available bid with the highest value first. The **manual** method uses dominance breaking constraints in Definition 8.

- *SetCover-n*: the set covering problem [58] is to minimize a *linear objective* subjective to *linear inequality constraints*. We generate random

[1] `https://people.eng.unimelb.edu.au/pstuckey/dom-jump/`

	no-dom	**manual**	**2-dom**		**3-dom**		**4-dom**	
Problem	Solving	Solving	Solving	Total	Solving	Total	Solving	Solving
Knapsack-100	–	**1.33**	1.12	1.74	0.02	28.09	0.08	1272.53
Knapsack-150	–	141.04	126.55	128.29	0.13	**120.96**	0.54	3600.54
Knapsack-200	–	1854.63	1671.77	1675.13	0.27	**355.41**	0.98	3600.98
Knapsack-250	–	6742.01	6548.54	6550.39	0.77	**811.76**	1.03	3601.03
Knapsack-300	–	–	–	–	1.78	**1597.57**	1.84	3601.84
DisjKnapsack-100	–	**2.91**	7.21	7.69	0.02	23.56	0.04	991.07
DisjKnapsack-150	–	608.95	3363.07	3364.16	0.10	**104.25**	0.28	3600.28
DisjKnapsack-200	–	5971.33	–	–	2.83	**336.32**	0.97	3600.97
DisjKnapsack-250	–	–	–	–	41.38	**850.16**	3.16	3603.16
DisjKnapsack-300	–	–	–	–	481.75	**1841.92**	14.45	3614.45
ConcertSched-25	37.72	19.08	4.05	**4.33**	3.20	5.33	2.87	37.49
ConcertSched-30	1166.07	756.93	228.94	229.00	122.65	**124.61**	136.04	197.55
ConcertSched-35	2645.94	1562.62	659.44	659.57	469.77	**473.39**	389.46	533.44
ConcertSched-40	5090.30	3206.15	1426.05	1426.17	1227.99	**1232.44**	1210.73	1440.56
ConcertSched-45	6248.07	5882.37	4146.53	4146.65	3316.66	**3322.27**	3187.46	3480.05
MaxCut-30	0.60	0.33	0.24	**0.31**	0.10	1.69	0.10	32.85
MaxCut-35	18.04	7.08	5.07	5.17	1.62	**4.69**	1.31	83.75
MaxCut-40	281.78	97.21	69.82	69.93	21.52	**26.15**	16.30	160.53
MaxCut-45	2548.66	1102.00	834.37	834.55	188.30	**196.21**	135.81	385.08
MaxCut-50	7160.27	6765.60	6703.06	6703.11	4671.98	4680.55	3595.26	**3945.98**
CombAuc-100	67.16	**1.12**	1.15	1.36	0.83	3.64	0.97	53.38
CombAuc-150	–	1129.94	1140.36	1140.90	433.17	**443.22**	390.35	600.61
CombAuc-200	–	4086.12	4107.03	4107.52	1847.73	1862.48	1524.59	**1859.62**
CombAuc-250	–	6473.76	6486.24	6486.44	3013.30	**3032.58**	2601.24	3093.52
CombAuc-300	–	6375.00	6381.24	6381.51	2033.18	**2066.10**	1709.82	2535.92
SetCover-100	76.68	1.63	1.39	**1.46**	0.01	4.82	0.01	113.96
SetCover-150	–	3620.55	2668.80	2669.02	0.65	**27.95**	0.02	571.01
SetCover-200	–	6991.12	6677.44	6677.53	39.63	**116.89**	0.06	2027.89
SetCover-250	–	–	7168.51	7168.57	364.78	**523.38**	62.57	3347.68
SetCover-300	–	–	–	–	783.23	**1100.08**	282.64	3882.64

Table 1: Comparison of average solving and total time using fixed search heuristics. The column "Solving" is the average time for problem-solving, and the column "Total" is the average of total time for nogood generation and problem-solving.

instances using the scheme of Umetani [59] with $m = 100$ items, $n = 100, 150, 200, 250, 300$ sets, the covering density to be 5%. The fixed search heuristic is to include sets in order of their indices. The **manual** method uses dominance breaking constraints in Definition 9.

Tables 1 and 2 report the average solving time and total time using fixed search heuristic and the free search option of the Chuffed solver respectively. An entry "–" indicates that the whole solving process time out after the 2-hour time limit. Overall, we observe that the efficiency of all methods is usually better using the free search option for all benchmark problems except *SetCover*. By comparing the problem-solving time of **3-dom** against **no-dom** and **manual**, it is clear that the generated dominance breaking nogoods can drastically reduce the solving time for all benchmarks, especially for large and hard instances. The percentage decreases in solving time of **3-dom** against **no-dom** are at least 34.75% using the fixed search heuristic and at least 45.37% using the free search option. The improvement of solving times can be up to three orders of magnitudes. Note that **no-dom** timeouts in most of the instances of *Knapsack*, *DisjKnapsack*, *CombAuc* and *SetCover*, and therefore the reduction of solving

	no-dom	**manual**	2-**dom**		3-**dom**		4-**dom**		**MIP**
Problem	Solving	Solving	Solving	Total	Solving	Total	Solving	Total	Solving
Knapsack-100	–	**0.39**	0.39	0.83	0.02	26.43	0.12	1493.50	0.01
Knapsack-150	–	**25.01**	25.08	26.40	0.14	114.57	0.16	3600.16	0.01
Knapsack-200	–	994.77	1126.47	1129.69	0.51	**328.63**	0.52	3600.52	0.02
Knapsack-250	–	5648.41	5533.63	5539.61	1.27	**735.65**	1.32	3601.32	0.02
Knapsack-300	–	6957.95	6956.59	6966.22	2.60	**1318.52**	2.37	3602.37	0.03
DisjKnapsack-100	–	**0.81**	2.33	2.70	0.02	24.01	0.08	1295.02	0.01
DisjKnapsack-150	–	180.90	2961.86	2962.78	0.14	**105.08**	0.15	3599.23	0.02
DisjKnapsack-200	–	5631.94	–	–	3.77	**303.85**	3.27	3603.27	0.03
DisjKnapsack-250	–	–	–	–	45.51	**654.55**	44.97	3644.97	0.06
DisjKnapsack-300	–	–	–	–	828.64	**1767.80**	820.77	4060.77	0.07
ConcertSched-25	12.79	8.28	1.30	**1.49**	1.06	2.52	1.31	35.07	1.60
ConcertSched-30	726.28	285.18	22.73	22.79	17.59	**19.89**	18.09	126.38	2.39
ConcertSched-35	1250.13	712.29	61.53	61.69	34.16	**38.90**	28.63	264.88	1.34
ConcertSched-40	1997.80	906.94	232.53	232.70	173.95	**180.75**	134.69	620.15	12.88
ConcertSched-45	3462.22	2666.11	1547.09	1547.39	1386.52	**1397.40**	990.78	1738.05	115.28
MaxCut-30	**0.05**	0.06	0.05	0.09	0.02	1.53	0.02	31.17	0.19
MaxCut-35	0.24	**0.17**	0.12	0.19	0.11	3.13	0.10	88.15	0.74
MaxCut-40	3.24	2.53	2.21	**2.35**	1.18	8.12	0.81	235.75	2.06
MaxCut-45	45.51	43.65	44.01	44.22	20.39	**32.34**	15.35	410.30	5.58
MaxCut-50	2185.55	2290.00	2245.01	2245.28	903.76	**922.78**	483.21	1074.19	11.79
CombAuc-100	1173.01	3.12	3.50	**3.77**	2.56	6.24	2.56	73.40	0.00
CombAuc-150	–	3579.07	3603.27	3603.70	2446.64	**2458.44**	2292.57	2527.58	0.01
CombAuc-200	–	5197.80	5218.82	5219.08	3462.73	3480.50	2795.25	**3181.67**	0.01
CombAuc-250	–	–	7160.79	7160.84	3933.10	**3962.05**	3350.12	4045.66	0.01
CombAuc-300	–	6744.57	6701.04	6701.34	2571.10	**2610.77**	1597.85	2472.81	0.01
SetCover-100	5675.74	67.16	22.80	22.87	0.02	**4.68**	0.01	115.81	0.01
SetCover-150	–	5701.37	5108.01	5108.13	0.82	**27.08**	0.02	554.79	0.01
SetCover-200	–	–	–	–	198.60	**271.79**	0.06	1958.83	0.01
SetCover-250	–	–	–	–	1191.58	**1345.87**	400.83	3568.17	0.01
SetCover-300	–	–	–	–	1944.13	**2215.12**	673.40	4237.09	0.02

Table 2: Comparison of average solving and total using the free search option. The column "Solving" is the average time for problem-solving, and the column "Total" is the average of total time for nogood generation and problem-solving.

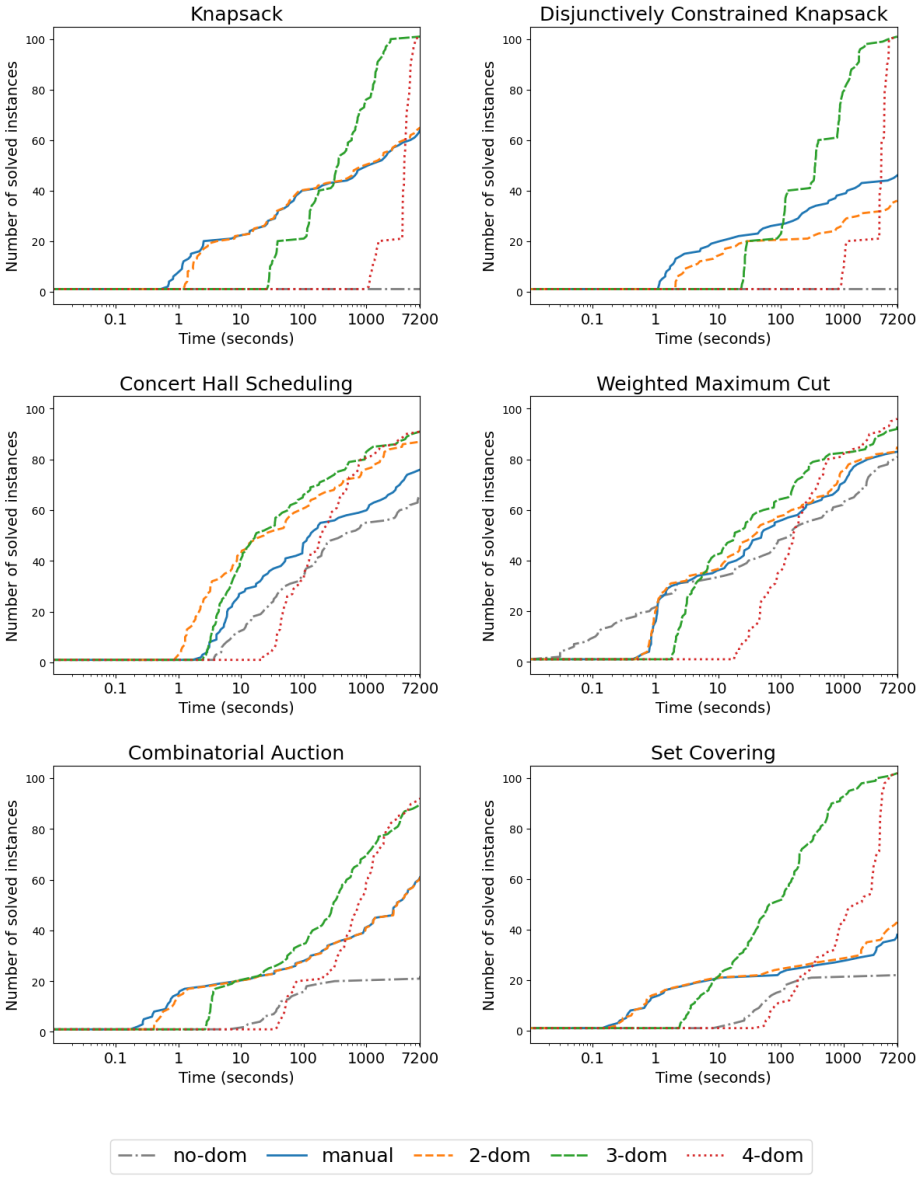


Figure 4: The number of solved instances over time using fixed search heuristics

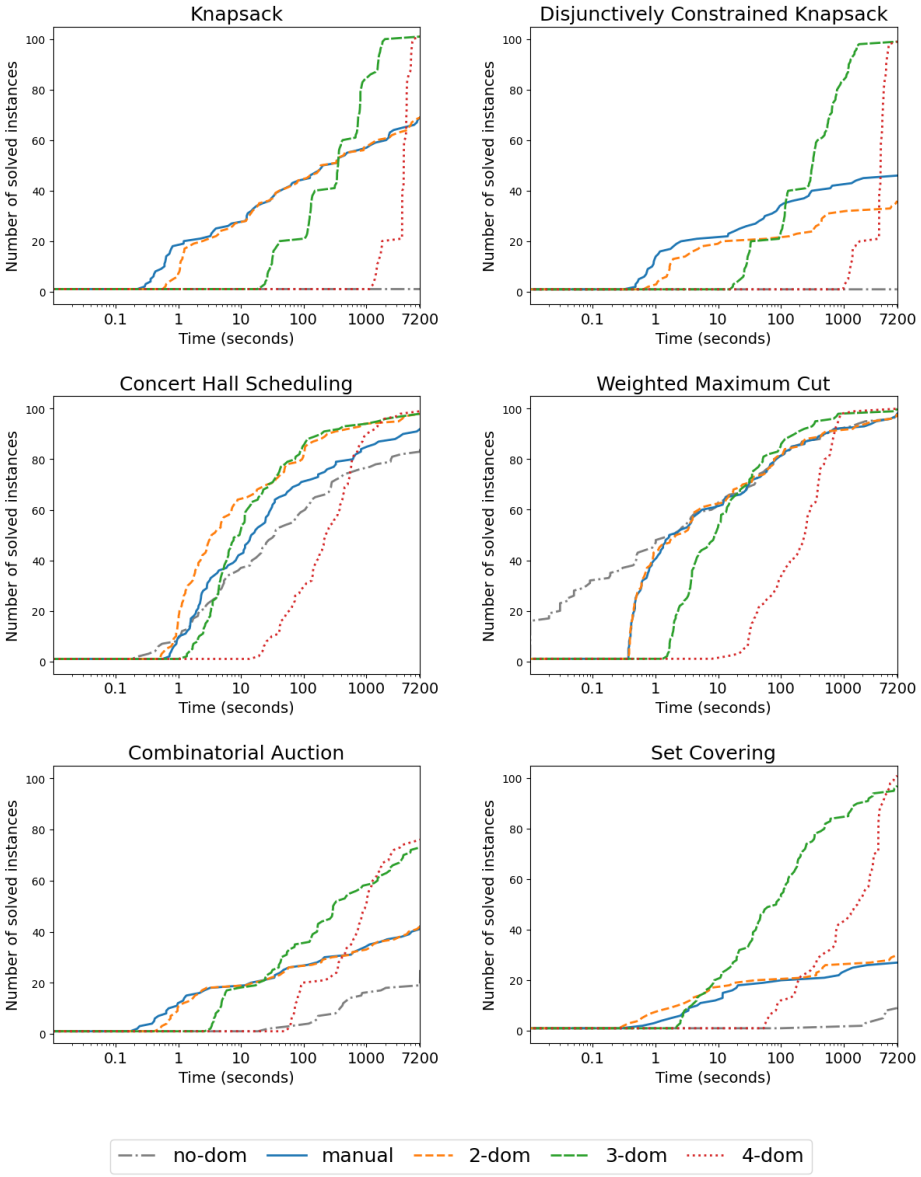


Figure 5: The number of solved instances over time using the free search option

time can be even more significant when there is no timeout limit. Another observation is that more nogood constraints usually help to reduce the solving time as demonstrated in *ConcertSchd*, *MaxCut*, *CombAuc* and *SetCover*. The exception is that the solving time of **4-dom** is larger than that of **3-dom** for some instances of *Knapsack* and *DisjKnapsack*. The reason is that **3-dom** is already efficient in problem-solving, and the overhead of handling more nogoods does not compensate for the searching time for the reduced search space. The results of **MIP** method in Table 2 shows that the instances from the standard benchmarks are relatively easy for integer programming solvers.

We also study the overall performance by comparing the average total time (generation time + solving time) as the evaluation metric. We highlight the smallest total time in bold for each group of instances of different problems. By comparing the difference of the total time and the solving time in each instance group, it is obvious that more time is needed to generate more nogoods. There is a trade-off between stronger pruning and generation time. Still, our method comes out on top with **3-dom** being the best. When using the fixed search heuristic, **3-dom** runs up to 305.60 times and 129.54 times faster than **no-dom** and **manual** respectively. If the free search option is used, then the speed-up of **3-dom** is up to 1212.76 times compared with **no-dom** and 210.54 times compared with **manual**. Again, the actual acceleration of L**-dom** can be even higher in the cases where **no-dom** and **manual** exceed the timeout limit.

Note that **no-dom** and **manual** may sometimes achieve better performance than L**-dom** in some small instances. To study the anytime behaviors [60] of different methods, we also give the number of solved instances versus the running time using fixed and free option in Figures 4 and 5. We observe that **2-dom**, **3-dom** and **4-dom** usually requires more time than **no-dom** and **manual** before they start to solve any instances, because the proposed method needs to solve extra generation CSPs to generate nogood constraints for dominance breaking. The advantage of automatically generated nogood constraints becomes more obvious for harder instances. Within the timeout limit, **3-dom** and **4-dom** solve more instances than **no-dom** and **manual** for all all benchmark problems. Another observation is that **2-dom** and **manual** have similar performance in *Knapsack*, *MaxCut*, *CombAuc*, and *SetCover*, while differs in *DisjKnapsack* and *ConcertHall*. In Section 7, we will give more theoretical analysis to explain the behaviors of **2-dom** and **manual** in different benchmarks.

It is worth noting that instances in the standard benchmarks are randomly generated, which could introduce bias to the results. It is possible that the randomly generated benchmarks adhere to a specific structure that is difficult without dominance breaking. Therefore, we include a real-life benchmark in the next section.

6.2. A Real-life Benchmark

We also compare different methods using a real-life benchmark with industrial instances for bus driver scheduling [61]. The problem can be formulated as a set partitioning problem with a separable objective and linear equality constraints. We are aware of no dominance breaking constraints for this problem in

	no-dom	2-**dom**		3-**dom**		4-**dom**		MIP
Problem	Solving	Solving	Total	Solving	Total	Solving	Total	
r1	361.03	362.15	365.31	160.09	**238.57**	257.80	3772.33	1.46
r1a	–	–	–	4124.49	**4377.42**	2065.75	5398.88	0.59
r2	**30.994**	31.69	37.011	10.98	159.01	22.92	3466.05	0.08
r3	**2010.773**	1975.39	2394.60	1879.29	5479.29	1877.45	5477.45	1.99
t1	**0.003**	0.002	0.020	0.002	0.117	0.005	0.984	0.01

Table 3: Comparison of solving and total time for the bus scheduling problem. The column "Solving" is the average time for problem-solving, and the column "Total" is the average of total time for nogood generation and problem-solving.

the literature. Therefore, we only compare L-**dom** with the baseline **no-dom**. The benchmark suite[2] consists of 12 instances. All methods cannot complete within the timeout limit when using the fixed search heuristic. With the free search option, only 5 instances can be solved by at least one method. We report the result for these 5 instances in Table 3 and highlight the smallest total time for each instance using constraint programming. As shown, **3-dom** can reduce the total time by 33.92% for the *r1* instance compared with **no-dom**. When we compare against the 2 hours timeout limit for **no-dom** for the *r1a* instance, **3-dom** can improve the total time by at least 39.20%.

On the other hand, **no-dom** has the smallest total time for the *r2*, *r3* and *t1* instances. In particular, the *r2* and *t1* instances are relatively easy with a short solving time. Though the solving time of **3-dom** is smaller compared with **no-dom**, the reduced solving time cannot compensate for the overhead of nogood generation. This is in line with our observation from Figures 4 and 5 that L-**dom** has less advantage for easy instances in standard benchmarks.

As for the *r3* instance, **2-dom** cannot speed up the solving time too much, while the nogood generation of **3-dom** and **4-dom** both time out and can generate no additional nogoods over **2-dom** within 1 hour. Therefore, the solving times of **2-dom**, **3-dom** and **4-dom** are similar since they are solving exactly the same problem. More efficient nogood generation can further improve the overall efficiency of our method. As shown in our follow-up work [62], the time for nogood generation can be further reduced by adding constraints in generation CSPs to avoid generating redundant nogoods.

6.3. A Non-linear Benchmark

Tables 2 and 3 show that the performance of **MIP** dominates in the standard and real-life benchmarks, which are linearizable and easy for integer programming. However, to demonstrate the effectiveness of our method in generating nogoods and improving performance for problems with non-linear structures, we introduce a variant of the Business-to-Business (B2B) meeting scheduling problem [63]. We analyze the sufficient conditions for betterment and implied satisfaction of our method for this problem.

[2] `https://github.com/MiniZinc/minizinc-benchmarks/tree/master/bus_scheduling`

The B2B meeting scheduling problem requires assigning a set M of meetings, each of which involves two participants in a set P, to a set T of available time slots, subject to the following constraints:

- Each meeting takes a time slot.
- Each participant can attend at most one meeting in each time slot.
- At each time slot, there can be at most L meetings due to the limit of available locations for holding meetings.

The original problem's objective is to minimize the number of breaks, where a "break" refers to a sequence of idle time slots between two meetings of a participant. We introduce a variant of this problem with a different objective to minimize the total duration of all participants. Here, the duration of a participant is defined as the difference between the time slots of the first and last meetings of the participant. The problem model is as follows:

$$\text{minimize} \sum_{p \in P} (\max(\{x_m \mid m \in M_p\}) - \min(\{x_m \mid m \in M_p\}) + 1) \tag{9a}$$

$$\text{subject to } \mathit{atmost}(\{x_m \mid m \in M\}, \{t\}, L), \forall t \in T \tag{9b}$$

$$\mathit{alldifferent}(\{x_m \mid m \in M\}), \forall p \in P \tag{9c}$$

where x_m is the assigned time slot for the meeting m, and $M_p \subseteq M$ is the set of meetings involving the participant $p \in P$. The sufficient conditions for implied satisfaction of (9b) and (9c) are given in Theorems 12 and 13, while (9a) does not match the types of objectives in Section 4.2. The sufficient condition for betterment of (9a) can be analyzed as follows.

Theorem 18. *Let $f(\bar{\theta})$ be the objetive function $\sum_{p \in P}(\max(\{\bar{\theta}[x_m] \mid m \in M_p\}) - \min(\{\bar{\theta}[x_m] \mid m \in M_p\}) + 1)$. Suppose $\theta, \theta' \in \mathcal{D}^S$ are two assignments over the same scope S, and their mutation mapping is $\mu^{\theta' \mapsto \theta} : \mathcal{D}^X_{\theta'} \mapsto \mathcal{D}^X_{\theta}$. We define the projection functions:*

- $\max{\downarrow_S}(\theta) \equiv \max(\{\theta[x_m] \mid m \in M_p \wedge x_m \in S\})$, *and*
- $\min{\downarrow_S}(\theta) \equiv \min(\{\theta[x_m] \mid m \in M_p \wedge x_m \in S\})$

If we have $\max{\downarrow_S}(\theta) \leq \max{\downarrow_S}(\theta')$, and $\min{\downarrow_S}(\theta) \geq \min{\downarrow_S}(\theta')$ for all participant $p \in P$, then $f(\bar{\theta}) \leq f(\bar{\theta}')$ for all full assignments $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$, where $\bar{\theta} = \mu^{\theta' \mapsto \theta}(\bar{\theta}')$.

Proof. The max function for each participant can be expressed as

$$\begin{aligned}
&\max(\{\bar{\theta}[x_m] \mid m \in M_p\}) \\
=&\max(\{\bar{\theta}[x_m] \mid m \in M_p \wedge x_m \in S\} \cup \{\bar{\theta}[x_m] \mid m \in M_p \wedge x_m \notin S\}) \\
=&\max(\max{\downarrow_S}(\theta), \max(\{\bar{\theta}[x_m] \mid m \in M_p \wedge x_m \notin S\}))
\end{aligned}$$

By Definition 3, $\bar{\theta}[x_m] = \bar{\theta}'[x_m]$ when $x_m \notin S$, we have

$$\max(\{\bar{\theta}[x_m] \mid m \in M_p \wedge x_m \notin S\}) = \max(\{\bar{\theta}'[x_m] \mid m \in M_p \wedge x_m \notin S\})$$

	no-dom	**2-dom**		**3-dom**		**4-dom**		**MIP**
Problem	Solving	Solving	Total	Solving	Total	Solving	Total	Solving
Meeting-20	261.94	156.07	**156.09**	156.09	156.27	146.45	147.84	–
Meeting-25	471.23	233.84	**233.86**	233.80	234.09	250.22	253.17	–
Meeting-30	433.67	97.21	**97.23**	97.98	98.42	103.43	109.23	–
Meeting-35	903.84	197.18	197.22	191.55	192.23	172.32	**181.72**	–
Meeting-40	1078.87	289.59	289.64	287.85	288.86	182.59	**199.17**	–

Table 4: Comparison of average solving and total using the free search option. The column "Solving" is the average time for problem-solving, and the column "Total" is the average of total time for nogood generation and problem-solving.

Therefore, $\max{\downarrow_S}(\theta) \leq \max{\downarrow_S}(\theta')$ implies that $\max(\{\bar{\theta}[x_m] \mid m \in M_p\}) \leq \max(\{\bar{\theta}'[x_m] \mid m \in M_p\})$. The proof for the min function is similar. The objective is a linear combination of the max and the negated min function, and thus $f(\bar{\theta}) \leq f(\bar{\theta}')$. □

Theorem 18 allows us to apply our method and formulate the generation of nogoods as solving auxiliary CSPs. We generated random instances of the B2B Meeting Scheduling Problem using a random generator [64]. The instances had $|M| \in 20, 25, 30, 35, 40$ meetings, $|P| = 10$ participants, and $|T| = 20$ time slots with $L = 10$.

Table 4 presents the average solving and total time for the B2B meeting scheduling problem using the Chuffed solver with the free search option and using the CBC solver 2.10. While the **MIP** method times out and fails to complete the search within the time limit for all instances, constraint programming solves these instances with ease. Compared to the **no-dom** approach, the addition of automatically generated nogoods can reduce the solving time by 43.56% to 81.52%. The **4-dom** approach achieves the best performance in instances with over 35 meetings, while it is also competitive with the **2-dom** and **3-dom** approaches in small instances. These experimental results highlight the potential of our framework to be extended to more complex problems with diverse objectives and constraints.

7. Strength of Dominance Breaking Nogoods

Section 6 demonstrates the effectiveness of the proposed method empirically. In this section, we further give a theoretical study on the strength of generated nogoods to explain their superior performance in benchmark problems compared with dominance breaking constraints derived manually in the literature.

The effect of dominance breaking constraints is to make suboptimal full assignments in a COP become infeasible so that they are pruned in the B&B algorithm. If one dominance breaking constraint c_1 implies another one c_2, then a full assignment violating c_1 must also violate c_2, and c_1 can prune more full assignments. Formally, we say that a constraint c_1 is *logically stronger than* c_2 if c_1 implies c_2, and they are *logically equivalent* if they imply each other. Similarly, a set of constraints C_1 is *logically stronger than* another set of

constraints C_2 if the conjunction of C_1 implies the conjunction of C_2, and they are *logically equivalent* if they imply each other. In the sequel, we compare the set of generated nogoods and dominance breaking constraints in the literature for each benchmark in each subsection.

7.1. 0-1 Knapsack Problem

The 0-1 *knapsack problem* is a problem with a *linear objective* and a *linear inequality constraint*. We use one variable $x_i \in \{0, 1\}$ for each item i to indicate whether the item is selected or not. The problem model is as follows:

$$\text{maximize} \sum_{i=1}^{n} p_i x_i \tag{10a}$$

$$\text{subject to} \sum_{i=1}^{n} w_i x_i \leq W \tag{10b}$$

The parameters p_i and w_i are the profit and the weight of item i, and W is the knapsack capacity. Chu and Stuckey [65] propose the following dominance breaking constraints.

Definition 4. *[65] The set of manual dominance breaking constraints for the 0-1 knapsack problem are $x_i \leq x_j$ for all $i, j \in \{1, \ldots, n\}$ when either*

1. $p_i < p_j \wedge w_i \geq w_j$,
2. $p_i = p_j \wedge w_i > w_j$, *or*
3. $p_i = p_j \wedge w_i = w_j \wedge i < j$,

When the length l of nogoods is set to 2, our method can generate a set of dominance breaking nogoods that is equivalent to the set of manual dominance breaking constraints in Definition 4.

Theorem 19. *When the nogood length is $l = 2$, the set of automatically generated dominance breaking nogoods is logically equivalent to the set of manual dominance breaking constraints in Definition 4.*

Proof. Suppose the solution of the generation CSP is a pair (θ, θ') of partial assignments over the same scope $S = \{x_i, x_j\}$ where $i < j$. By Theorems 4, 8 and 17, (θ, θ') must satisfy:

- betterment: $p_i v_i + p_j v_j \geq p_i v'_i + p_j v'_j$
- implied satisfaction for (10b): $w_i v_i + w_j v_j \leq w_i v'_i + w_j v'_j$
- compatibility: $(-(p_i v_i + p_j v_j), v_i, v_j) <_{lex} (-(p_i v'_i + p_j v'_j), v'_i, v'_j)$

where $v_i = \theta[x_i]$, $v_j = \theta[x_j]$, $v'_i = \theta'[x_i]$ and $v'_j = \theta'[x_j]$. Since $D(x_i) = D(x_j) = \{0, 1\}$, we can exhaust all value combinations and find that there are only two possible valid solutions for the generation CSP: either $(v_i, v_j, v'_i, v'_j) = (1, 0, 0, 1)$ or $(v_i, v_j, v'_i, v'_j) = (0, 1, 1, 0)$. Therefore, we have a set of nogood constraints $\neg\theta'$ of the forms:

- $x_i \neq 0 \vee x_j \neq 1$ when $p_i \geq p_j \wedge w_i \leq w_j \wedge (-p_i, w_i) <_{lex} (-p_j, w_j)\}$,
- $x_i \neq 1 \vee x_j \neq 0$ when $p_i \leq p_j \wedge w_i \geq w_j \wedge (-p_j, w_j) <_{lex} (-p_i, w_i)$, and
- $x_i \neq 1 \vee x_j \neq 0$ when $p_i = p_j \wedge w_i = w_j$.

Since $(x_i \neq 0 \vee x_j \neq 1) \equiv (x_i \geq x_j)$ and $(x_i \neq 1 \vee x_j \neq 0) \equiv (x_i \leq x_j)$, the set of constraints in Definition 4 is equivalent to the set of generated nogoods. □

Theorem 19 is consistent with the empirical result in Section 6, where **2-dom** and **manual** has similar performance in problem-solving. It is easy to see that the set of generated dominance breaking nogoods becomes even stronger when we increment the maximum nogood length L.

Corollary 4. *When the maximum nogood length is $L > 2$, the set of automatically generated dominance breaking nogoods is logically stronger than the set of manual dominance breaking constraints in Definition 4.*

7.2. Disjunctively Constrained Knapsack Problem

The disjunctively constrained knapsack problem [55] is an extension of the knapsack problem with additional constraints that some item pairs cannot be selected simultaneously. The conflicts among items can be represented by an undirected graph $G = (N, E)$ where $N = \{1, \ldots, n\}$ and $(i, j) \in E$ if item i and j are in conflict with each other. Let $\Gamma(i) = \{j \mid (i, j) \in E\}$ be the *neighborhood* of item i in G, and the extra constraints can be modeled by Boolean disjunction constraints.

$$(x_i = 0) \vee (x_j = 0), \forall j \in \Gamma(i), i \in \{1, \ldots, n\} \tag{11}$$

The set of dominance breaking constraints is similar to that for the 0-1 knapsack problem, except that a precondition is required to ensure swapping the value of x_i and x_j does not violate the Boolean disjunction constraints.

Definition 5. *The set of manual dominance breaking constraints for the disjunctively constrained knapsack problem are $(\bigwedge_{k \in \Gamma(j)}(x_k = 0)) \rightarrow (x_i \leq x_j)$ for all $i, j \in \{1, \ldots, n\}$ when either*

1. $p_i < p_j \wedge w_i \geq w_j$,
2. $p_i = p_j \wedge w_i > w_j$, *or*
3. $p_i = p_j \wedge w_i = w_j \wedge i < j$,

The number of variables involved in the manual dominance breaking constraint depends on the size of $\Gamma(j)$. The following theorem states that the set of automatically generated dominance breaking nogoods of length $l = \max(|\Gamma(j)|) + 2$ is logically stronger than manual dominance breaking constraints.

Theorem 20. *When the nogood length is $l = \max(|\Gamma(j)|) + 2$, the set of automatically generated dominance breaking nogoods is logically stronger than the set of manual dominance breaking constraints in Definition 5.*

Proof. It suffices to show that there is a logically equivalent dominance breaking nogood of length $l = |\Gamma(j)| + 2$ for every constraint in Definition 5. When $i \in \Gamma(j)$, the constraint $(\bigwedge_{k\in\Gamma(j)}(x_k = 0)) \rightarrow (x_i \leq x_j)$ becomes a tautology. Therefore, we only consider the case when $i \notin \Gamma(j)$. For the ease of presentation, we assume that $|\Gamma(j)| = 1$, and the proof can be generalized trivially.

When $l = 3$, consider a pair (θ, θ') of partial assignments over the scope $S = \{x_i, x_j, x_k\}$ where $i < j$, $k \in \Gamma(j)$ and (θ, θ') is a solution of the generation CSP with length $l = 3$. The pair (θ, θ') must satisfy sufficient conditions from Theorems 4, 8 and 17, which are similar to those in the proof of Theorem 19. In addition, (θ, θ') must also satisfy

$$((\theta[x_j] = 0) \vee (\theta[x_k] = 0)) \Rightarrow ((\theta'[x_j] = 0) \vee (\theta'[x_k] = 0)) \tag{12}$$

by Theorem 10. One possible solution is to have $\theta[x_k] = \theta'[x_k] = 0$. Following the same reasoning as Theorem 19, we have a set of nogoods of the forms:

- $x_i \neq 0 \vee x_j \neq 1 \vee x_k \neq 0$ when $p_i \geq p_j \wedge w_i \leq w_j \wedge (-p_i, w_i) <_{lex} (-p_j, w_j)$,
- $x_i \neq 1 \vee x_j \neq 0 \vee x_k \neq 0$ when $p_i = p_j \wedge w_i = w_j$, and
- $x_i \neq 1 \vee x_j \neq 0 \vee x_k \neq 0$ when $p_i \leq p_j \wedge w_i \geq w_j \wedge (-p_j, w_j) <_{lex} (-p_i, w_i)$.

Since we have $(x_i \neq 0 \vee x_j \neq 1 \vee x_k \neq 0) \equiv ((x_k = 0) \rightarrow (x_i \geq x_j))$ and $(x_i \neq 1 \vee x_j \neq 0 \vee x_k \neq 0) \equiv ((x_k = 0) \rightarrow (x_i \leq x_j))$, there is a logically equivalent dominance breaking nogood for each dominance breaking constraint in Definition 5. □

Note that the conflict constraints can also be modeled by linear inequality constraints, but the corresponding generation CSP will have fewer solutions. For instance, if we replace (11) with $x_j + x_k \leq 1$, there is a constraint $\theta[x_i] + \theta[x_k] \leq \theta'[x_i] + \theta'[x_k]$ in the generation CSP by Theorem 8, and it is a stronger condition than (12). Following the same reasoning as the proof of Theorem 19, we will find the generation CSP becomes unsatisfiable, and thus there are no dominance breaking nogoods with length $l = 3$.

7.3. Capacitated Concert Hall Scheduling Problem

The *Capacitated Concert Hall Scheduling Problems* [56] is to schedule a set A of applications to a set H of concert halls. Each application $i \in A$ has a period $[s_i, e_i]$, a profit p_i and a requirement r_i, and each concert hall $h \in H$ has a capacity c_h. An application i can be scheduled in a hall h, if the requirement $r_i \leq c_h$, and application i and j cannot be in the same hall if their period are overlapping, i.e. $[s_i, e_i] \cap [s_j, e_j] \neq \emptyset$.

Let $H_i = \{h \in H \mid c_h \geq r_i\}$ be the set of feasible halls for application i. We use one variable $x_i \in H_i \cup \{0\}$ for each application $i \in A$. Application i is scheduled in hall h when $x_i = h$, while it is not scheduled when $x_i = 0$. The non-overlapping requirement can be modeled as *alldifferent_except_0* constraints, where the set of applications that are overlapping at any time point should not be scheduled in the same hall. It is sufficient to consider the start time of all

applications. Let $\Gamma(i) = \{j \mid s_j \leq s_i \leq e_j\}$ be the set of applications whose periods are overlapping with the start time of application $i \in A$. The problem model is as follows:

$$\text{maximize} \sum_{k \in A} p_k \mathbf{1}_{>0}(x_k) \tag{13a}$$

$$\text{subject to } \textit{alldifferent\_except\_0}(\{x_j \mid j \in \Gamma(i)\}), \forall k \in A \tag{13b}$$

$$x_k \in H_k \cup \{0\}, \forall k \in A \tag{13c}$$

where $\mathbf{1}_{>0} : \mathbb{R} \mapsto \{0, 1\}$ is an indicator function returning 1 when $x_i > 0$. The objective is to maximize the total profit of all scheduled concerts. Gange and Stuckey [56] propose constraints to prefer shorter, more profitable concerts.

Definition 6. *[56] The set of manual dominance breaking constraints for capacitated concert hall scheduling problem are $x_i > 0 \rightarrow x_j > 0$ for all $i, j \in A$ where either*

1. $[s_j, e_j] \subseteq [s_i, e_i] \wedge r_j \leq r_i \wedge p_j > p_i$, *or*
2. $[s_j, e_j] \subseteq [s_i, e_i] \wedge r_j \leq r_i \wedge p_i = p_j \wedge i < j$

The following theorem shows the relations between the set of automatically generated dominance breaking nogoods and the set of manual dominance breaking constraints when the length of nogoods is set to $l = 2$.

Theorem 21. *When the nogood length is $l = 2$, the set of automatically generated dominance breaking nogoods is logically stronger than the set of manual dominance breaking constraints in Definition 6.*

Proof. Note that each constraint in Definition 6 is equivalent to the conjunction of several nogood constraints:

$$\begin{aligned} & x_i > 0 \rightarrow x_j > 0 \\ \Leftrightarrow & \neg(x_i > 0) \vee (x_j > 0) \\ \Leftrightarrow & x_i = 0 \vee x_j \neq 0 \\ \Leftrightarrow & \bigwedge_{h \in H_i} (x_i \neq h \vee x_j \neq 0) \end{aligned} \tag{14}$$

It suffices to show that the set of nogoods in (14) is logically equivalent to a subset of generated dominance breaking nogoods when $l = 2$. We will prove that for each nogood $(x_i \neq h \vee x_j \neq 0)$ where $h \in H_i$, there is a pair (θ, θ') of partial assignments over the scope $S = \{x_i, x_j\}$, where $\theta[x_i] = \theta'[x_j] = 0$ and $\theta[x_j] = \theta'[x_i] = h$.

The pair (θ, θ') is a solution of the generation CSP if it satisfies sufficient conditions from Theorems 4, 14 and 17:

- betterment: $p_i \mathbf{1}_{>0}(0) + p_j \mathbf{1}_{>0}(h) = p_j \geq p_i = p_i \mathbf{1}_{>0}(h) + p_j \mathbf{1}_{>0}(0)$
- implied satisfaction for (13b) and (13c)

- $\forall k \in A, \{\theta[x_j] \mid j \in \Gamma(k)\} \subseteq \{\theta'[x_i] \mid i \in \Gamma(k)\}$,
- $\forall k \in A, \theta[x_j] \in H_j \cup \{0\}$

- compatibility: $(-p_j, 0, h) <_{lex} (-p_i, h, 0)$

By definition of $\Gamma(k)$, if $[s_j, e_j] \subseteq [s_i, e_i]$, then $j \in \Gamma(k)$ implies that $i \in \Gamma(k)$, and the implied satisfaction for (13b) must hold. Also, by definition of H_i and H_j, if $r_j \leq r_i$, then $H_i \subseteq H_j$, and the implied satisfaction for (13c) must hold when $\theta[x_j] = \theta'[x_i] = h \in H_i$. Therefore, when $[s_j, e_j] \subseteq [s_i, e_i]$, $p_j \geq p_i$, $h \in H_i$, and $(-p_j, 0, h) <_{lex} (-p_i, h, 0)$, (θ, θ') is a solution of the generation CSP and there is a dominance breaking nogoods $\neg\theta' \equiv (x_i \neq h \vee x_j \neq 0)$. The set of all generated nogoods is the same as that in (14) and is logically equivalent to the set of manual dominance breaking constraints in Definition 6. □

Theorem 21 shows that the set of dominance breaking nogoods is even logically stronger than manual dominance breaking constraints in [56], and the theoretical result is also consistent with the empirical evaluation in Section 6, where **2-dom** has a better performance than **manual** in terms of the problem-solving time and the number of solved instances within the timeout limit.

Again, it is easy to see that the set of generated dominance breaking nogoods becomes even stronger when we increment the maximum nogood length L.

Corollary 5. *When the maximum nogood length is $L > 2$, the set of automatically generated dominance breaking nogoods is logically stronger than the set of manual dominance breaking constraints in Definition 6.*

7.4. Weighted Maximum Cut Problem

Given a weighted undirected graph $G = (V, E)$, the maximum cut problem is to find a partition (V_1, V_2) of V to maximize the total weights of crossing edges. We use one binary variable $x_i \in \{0, 1\}$ for each vertex $i \in V$ to indicate whether i is in V_1. The problem model is as follows:

$$\begin{aligned} \text{maximize} \quad & \sum_{(i,j) \in E} w_{(i,j)} x_i \oplus x_j \\ \text{subject to} \quad & x_i \in \{0, 1\}, \forall i \in V \end{aligned}$$

where $x_i \oplus x_j$ is the *exclusive disjunction* which is 1 only when $x_i \neq x_j$. Note that the objective function is equivalent to the cut function $g(V_1) = g(\{i \mid x_i = 1\})$, which is a submodular set function. Inspired by the local search algorithm [66], we define the following manual dominance breaking constraints.

Definition 7. *The set of manual dominance breaking constraints for the weighted maximum cut problems are $x_i \neq 1 \vee x_j \neq 1$ for all pairs of nodes $\{i, j\}$ when either $g(\{i\}) \geq g(\{i, j\})$ or $g(\{j\}) \geq g(\{i, j\})$.*

When the length l of generated nogoods is 2, the set of generated dominance breaking nogoods is equivalent to the set of constraints in Definition 7.

Theorem 22. *The set of automatically generated dominance breaking nogoods of length $l = 2$ is logically equivalent to the set of manual dominance breaking constraints in Definition 4.*

Proof. By Corollary 1 and Theorem 17, a pair (θ, θ') of partial assignments over the scope $S = \{x_i, x_j\}$ is a solution of the generation CSP for $l = 2$ if:

- betterment: $g(U(\theta)) \geq g(U(\theta')) \wedge U(\theta) \subseteq U(\theta')$
- compatibility: $(-g(U(\theta)), \theta[x_i], \theta[x_j]) <_{lex} (-g(U(\theta'), \theta'[x_i], \theta'[x_j])$

where $U(\theta) = \{k \mid x_k \in S \wedge \theta[x_k] = 1\}$ and $U(\theta') = \{k \mid x_k \in S \wedge \theta'[x_k] = 1\}$. Since $D(x_i) = D(x_j) = \{0, 1\}$, we exhaust all value combinations and find that there are only two possible valid solutions: (1) $\theta[x_i] = \theta'[x_i] = \theta'[x_j] = 1, \theta[x_j] = 0$, and (2) $\theta[x_j] = \theta'[x_i] = \theta'[x_j] = 1, \theta[x_i] = 0$. Therefore, there is a generated nogood constraint $\neg\theta' \equiv (x_i \neq 1 \vee x_j \neq 1)$ when either $g(\{i\}) \geq g(\{i, j\})$ or $g(\{j\}) \geq g(\{i, j\})$, and the set of all generated nogoods of length 2 is equivalent to the set of constraints in Definition 7. □

Similar to those for the 0-1 knapsack problems, the set of generated nogoods becomes even stronger when we increment the maximum nogood length L.

Corollary 6. *When the maximum nogood length is $L > 2$, the set of automatically generated dominance breaking nogoods is logically stronger than the set of manual dominance breaking constraints in Definition 7.*

7.5. Combinatorial Auction Problem

The combinatorial auction problem is to select a subset of n bidders for m items, where each bidder B_i is a subset of $\{1, \ldots, m\}$ and is associated with a profit p_i. The restriction is that each item can appear at most once in the selected bidders, and the aim is to maximize the total values. We use one binary variable $x_i \in \{0, 1\}$ for each bidder to indicate whether the bidder i is selected or not and let $\Gamma(k) = \{i \mid k \in B_i\}$ be the set of bidders that contain item k. The problem model is as follows:

$$\text{maximize} \sum_{i=1}^{n} p_i x_i \tag{15a}$$

$$\text{subject to} \sum_{i \in \Gamma(k)}^{n} x_i \leq 1, \forall k \in \{1, \ldots, m\} \tag{15b}$$

Following the method by Chu and Stuckey [65], we consider dominance breaking constraint derived from mappings that swap the values of variables x_i and x_j.

Definition 8. *The set of manual dominance breaking constraints for the combinatorial auction problems are $x_i \leq x_j$ for all $i, j \in \{1, \ldots, n\}$ when $B_i \supseteq B_j$ and either (1) $p_i < p_j$, or (2) $p_i = p_j \wedge i < j$.*

Theorem 23. *The set of automatically generated dominance breaking nogoods of length $l = 2$ is equivalent to the set of manual dominance breaking constraints in Definition 8.*

Proof. Suppose the solution of the generation CSP is a pair (θ, θ') of partial assignments over the same scope $S = \{x_i, x_j\}$. If $B_i \supseteq B_j$, then $j \in \Gamma(k)$ implies that $i \in \Gamma(k)$. By Theorems 4, 8 and 17, (θ, θ') is a solution of the generation CSP if it satisfies:

- betterment: $p_i v_i + p_j v_j \geq p_i v'_i + p_j v'_j$
- implied satisfaction for (15b):
  - $v_j \leq v'_j$ when $i \in \Gamma(k)$ and $j \notin \Gamma(k)$, or
  - $v_i + v_j \leq v'_i + v'_j$ when $i, j \in \Gamma(k)$
- compatibility: $(-(p_i v_i + p_j v_j), v_i, v_j) <_{lex} (-(p_i v'_i + p_j v'_j), v'_i, v'_j)$

where $v_i = \theta[x_i]$, $v_j = \theta[v_j]$, $v'_i = \theta'[x_i]$ and $v'_j = \theta'[x_j]$. We can exhaust all value combinations and find that there are only two possible valid solutions of the generation CSP: either $(v_i, v_j, v'_i, v'_j) = (1, 0, 0, 1)$ or $(v_i, v_j, v'_i, v'_j) = (0, 1, 1, 0)$. The set of generated nogoods consists of:

- $x_i \neq 0 \vee x_j \neq 1$ when $p_i > p_j$, and
- $x_i \neq 1 \vee x_j \neq 0$ when $p_i \leq p_j$,

which is equivalent to the set of constraints in Definition 8. □

7.6. Set Covering Problems

The set covering problem [58] is to select a collection of subsets whose union equals to the universe $\{1, \ldots, m\}$, and it is a dual problem of the combinatorial auction problem. In this problem, each subset S_i is associated with a cost c_i, and the objective is to minimize the total cost of the selected subsets. We use one binary variable $x_i \in \{0, 1\}$ for each subset, and let $\Gamma(k) = \{i \mid k \in B_i\}$ be the collection of subsets that contain element k. The problem model is as follows:

$$\text{minimize} \sum_{i=1}^{n} c_i x_i \tag{16a}$$

$$\text{subject to} \sum_{i \in \Gamma(k)} x_i \geq 1, \forall k \in \{1, \ldots, m\} \tag{16b}$$

Similar to Definition 8, we can derive the dominance breaking constraints by the method by Chu and Stuckey [65].

Definition 9. *The set of manual dominance breaking constraints for the set covering problems are $x_i \leq x_j$ for all $i, j \in \{1, \ldots, n\}$ when $B_i \subseteq B_j$ and either (1) $c_i > c_j$, or (2) $c_i = c_j \wedge i < j$.*

We have the following result, which is similar to that for the combinatorial auction problem.

Theorem 24. *The set of automatically generated dominance breaking nogoods of length $l = 2$ is equivalent to the set of manual dominance breaking constraints in Definition 9.*

The proof is similar to that of Theorem 23.

8. Concluding Remarks

In this paper, we present the first fully automated method to make dominance breaking accessible to non-experts for a class of constraint optimization problems. Our method formulates the generation of dominance breaking nogoods as a constraint satisfaction problem, providing a systematic and efficient way to generate dominance breaking nogoods. An important advantage of our method is the ability to control the amount and the aggregated strength of the generated nogoods. Empirical results demonstrate that our method can speed up the solving of constraint optimization problems using constraint solvers. Additionally, our theoretical analysis on various benchmark problems shows that our method can discover dominance breaking nogoods that had never been discovered before.

There are several directions for further improvement of our method. The generation of nogoods instance by instance can result in overhead that may not compensate for the reduction in problem-solving time. Our experiment shows that in problems with linear structures, the overall performance is not competitive with integer programming due to the large nogood generation time. Future research can focus on further reducing the generation time, such as through generalizing nogoods from small to large instances of the same problem class. Currently, the examination of nogood patterns still requires human intervention [67].

Our method has its applicability limited to a specific class of problems, resulting from the analysis of sufficient conditions for betterment and implied satisfaction. The Real-world problems, such as instances from the MiniZinc Challenge, consists of more complicated structures and cannot be handled by our current method. One possible solution to this limitation is to analyze the syntactic structures and exploit the functional dependency of variables [67]. Future research can explore alternative ways of addressing this limitation by automating the analysis of sufficient conditions.

In addition, future research can focus on understanding the characteristics and roles of individual nogoods and avoiding redundancies. It would be interesting to study measures to assess the effectiveness of individual nogood constraints in pruning the search space, allowing for the maintenance of only a subset of relevant nogoods. Furthermore, SAT-based techniques for nogood simplification, such as clause vivification [68], can potentially remove redundant literals in generated dominance breaking nogoods in practice.

Acknowledgement

We are grateful to the anonymous reviewers of IJCAI-PRICAI 2020 and the Artificial Intelligence journal for their insightful comments and suggestions. We also acknowledge the financial support of a General Research Fund (RGC Ref. No. CUHK 14206321) by the University Grants Committee, Hong Kong.

References

[1] L. Getoor, G. Ottosson, M. Fromherz, B. Carlson, Effective redundant constraints for online scheduling, in: Proceedings of the the Fourteenth AAAI National Conference on Artificial Intelligence, 1997, pp. 302–307.

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

[3] M. G. de la Banda, P. J. Stuckey, G. Chu, Solving talent scheduling with dynamic programming, INFORMS Journal on Computing 23 (1) (2011) 120–137.

[4] H. Qin, Z. Zhang, A. Lim, X. Liang, An enhanced branch-and-bound algorithm for the talent scheduling problem, European Journal of Operational Research 250 (2) (2016) 412–426.

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

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

[7] R. E. Korf, Optimal rectangle packing: new results., in: Proceedings of the Fourteenth International Conference on International Conference on Automated Planning and Scheduling, 2004, pp. 142–149.

[8] H. Hojabri, M. Gendreau, J.-Y. Potvin, L.-M. Rousseau, Large neighborhood search with constraint programming for a vehicle routing problem with synchronization constraints, Computers & Operations Research 92 (2018) 87–97.

[9] K. E. Booth, J. C. Beck, A constraint programming approach to electric vehicle routing with time windows, in: Proceedings of the 16th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research, Springer, 2019, pp. 129–145.

[10] F. Rossi, P. van Beek, T. Walsh, Handbook of Constraint Programming, Elsevier Science Inc., 2006.

[11] E. C. Freuder, In pursuit of the holy grail, Constraints 2 (1) (1997) 57–61.

[12] E. C. Freuder, Progress towards the holy grail, Constraints 23 (2) (2018) 158–171.

[13] A. Land, A. Doig, An Automatic Method of Solving Discrete Programming Problems, Econometrica: Journal of the Econometric Society (1960) 497–520.

[14] A. K. Mackworth, Consistency in networks of relations, Artificial intelligence 8 (1) (1977) 99–118.

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

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

[17] S. Prestwich, J. C. Beck, Exploiting dominance in three symmetric problems, in: Proceedings of the Fourth International Workshop on Symmetry and Constraint Satisfaction Problems, 2004, pp. 63–70.

[18] J.-N. Monette, P. Schaus, S. Zampelli, Y. Deville, P. Dupont, A CP approach to the balanced academic curriculum problem, in: Proceedings of the 7th International Workshop on Symmetry and Constraint Satisfaction Problems, 2007, pp. 56–63.

[19] G. Chu, P. 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, 2012, pp. 6–22.

[20] C. Mears, M. Garcia de la Banda, M. Wallace, B. Demoen, A method for detecting symmetries in constraint models and its generalisation, Constraints 20 (2) (2015) 235–273.

[21] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, G. Tack, MiniZinc: Towards a standard CP modelling language, in: Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming, Springer, 2007, pp. 529–543.

[22] J. H. M. Lee, A. 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, 2020, pp. 1192–1200.

[23] W. H. Kohler, K. Steiglitz, Characterization and theoretical comparison of branch-and-bound algorithms for permutation problems, Journal of the ACM (JACM) 21 (1) (1974) 140–156.

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

[25] G. Chu, P. 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, Springer, 2009, pp. 242–257.

[26] C.-F. Yu, B. W. Wah, Learning dominance relations in combinatorial search problems, IEEE Transactions on Software Engineering 14 (8) (1988) 1155–1175.

[27] C. D. Mears, Automatic symmetry detection and dynamic symmetry breaking for constraint programming, Ph.D. thesis, Monash University (2009).

[28] I. P. Gent, K. E. Petrie, J.-F. Puget, Symmetry in constraint programming, Foundations of Artificial Intelligence 2 (2006) 329–376.

[29] E. C. Freuder, Eliminating interchangeable values in constraint satisfaction problems, in: Proceedings of the 9th National Conference on Artificial Intelligence, 1991, pp. 227–233.

[30] A. M. Frisch, I. Miguel, T. 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, Springer, 2002, pp. 15–30.

[31] A. Ramani, I. L. Markov, Automatically exploiting symmetries in constraint programming, in: Proceedings of the 2004 Joint ERCIM/-CoLOGNET International Conference on Recent Advances in Constraints, Springer, 2004, pp. 98–112.

[32] J.-F. Puget, Automatic detection of variable and value symmetries, in: Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming, Springer, 2005, pp. 475–489.

[33] P. V. Hentenryck, P. Flener, J. Pearson, M. Ågren, Compositional derivation of symmetries for constraint satisfaction, in: Proceedings of the 6th International Symposium on Abstraction, Reformulation, and Approximation, Springer, 2005, pp. 234–247.

[34] C. Mears, M. Garcia de la Banda, M. Wallace, On implementing symmetry detection, Constraints 14 (4) (2009) 443–477.

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

[36] T. Feydy, P. J. Stuckey, Lazy clause generation reengineered, in: Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming, Springer, 2009, pp. 352–366.

[37] G. Chu, M. Banda, Garcia de la Banda, P. 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, Springer, 2010, pp. 71–86.

[38] G. Chu, M. G. de la Banda, P. J. Stuckey, Exploiting subproblem dominance in constraint programming, Constraints 17 (1) (2012) 1–38.

[39] G. Chu, P. J. Stuckey, Dominance breaking constraints, Constraints 20 (2) (2015) 155–182.

[40] M. Fischetti, D. Salvagnin, Pruning moves, INFORMS Journal on Computing 22 (1) (2010) 108–119.

[41] M. Fischetti, P. Toth, A new dominance procedure for combinatorial optimization problems, Operations Research Letters 7 (4) (1988) 181–187.

[42] P. Flener, A. M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, T. Walsh, Breaking row and column symmetries in matrix models, in: Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming, Springer, 2002, pp. 462–477.

[43] N. Beldiceanu, M. Carlsson, J.-X. Rampon, Global constraint catalog (2010).

[44] J.-C. Régin, A filtering algorithm for constraints of difference in CSPs, in: Proceedings of the Twelfth AAAI National Conference on Artificial Intelligence, 1994, pp. 362–367.

[45] N. Beldiceanu, E. Contejean, Introducing global constraints in CHIP, Mathematical and computer Modelling 20 (12) (1994) 97–123.

[46] C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan, T. Walsh, Among, common and disjoint constraints, in: Proceedings of the 2005 Joint ERCIM/-CoLogNET International Conference on Constraint Solving and Constraint Logic Programming, Springer, 2005, pp. 29–43.

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

[48] L. G. Kaya, J. N. Hooker, A filter for the circuit constraint, in: Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming, Springer, 2006, pp. 706–710.

[49] K. G. Francis, P. J. Stuckey, Explaining circuit propagation, Constraints 19 (1) (2014) 1–29.

[50] P. Van Hentenryck, Constraint satisfaction in logic programming, MIT press, 1989.

[51] A. K. Mackworth, E. C. Freuder, The complexity of constraint satisfaction revisited, Artificial Intelligence 59 (1-2) (1993) 57–62.

[52] C. W. Choi, J. H. M. Lee, P. J. Stuckey, Propagation redundancy in redundant modelling, in: Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming, Springer, 2003, pp. 229–243.

[53] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, S. Malik, Chaff: Engineering an efficient SAT solver, in: Proceedings of the 38th Annual Design Automation Conference, 2001, pp. 530–535.

[54] J. Forrest, T. Ralphs, H. G. Santos, S. Vigerske, J. Forrest, L. Hafer, B. Kristjansson, jpfasano, EdwinStraver, M. Lubin, Jan-Willem, rlougee, jpgoncal1, S. Brito, h-i gassmann, Cristina, M. Saltzman, tosttost, B. Pitrus, F. MATSUSHIMA, to st, coin-or/cbc: Release releases/2.10.10 (Apr. 2023). doi:10.5281/zenodo.7843975.
URL https://doi.org/10.5281/zenodo.7843975

[55] T. Yamada, S. Kataoka, K. Watanabe, Heuristic and exact algorithms for the disjunctively constrained knapsack problem, Information Processing Society of Japan Journal 43 (9).

[56] G. Gange, P. J. Stuckey, Sequential precede chain for value symmetry elimination, in: Proceedings of the 24th International Conference on Principles and Practice of Constraint Programming, Springer, 2018, pp. 144–159.

[57] E. Balas, A. Ho, Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study, in: Combinatorial Optimization, Springer, 1980, pp. 37–60.

[58] S. L. Hakimi, Optimum distribution of switching centers in a communication network and some related graph theoretic problems, Operations research 13 (3) (1965) 462–475.

[59] S. Umetani, Exploiting variable associations to configure efficient local search algorithms in large-scale binary integer programs, European Journal of Operational Research 263 (1) (2017) 72–81.

[60] S. Zilberstein, Using anytime algorithms in intelligent systems, AI magazine 17 (3) (1996) 73–73.

[61] S. D. Curtis, B. M. Smith, A. Wren, Forming bus driver schedules using constraint programming, in: Proceedings of the 1st International Conference on the Practical Applications of Constraint Technologies and Logic Programming, 1999, pp. 239–254.

[62] J. H. Lee, A. Z. Zhong, Towards more practical and efficient automatic dominance breaking, in: Proceedings of the 35th AAAI Conference on Artificial Intelligence, Vol. 35, 2021, pp. 3868–3876.

[63] M. Bofill, J. Espasa, M. Garcia, M. Palahí, J. Suy, M. Villaret, Scheduling b2b meetings, in: Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming, 2014, pp. 781–796.

[64] M. Bofill Arasa, J. Coll Caballero, J. Giráldez-Cru, J. Suy Franch, M. Villaret i Ausellé, The impact of implied constraints on maxsat b2b instances, International Journal of Computational Intelligence Systems, 2022, vol. 15, art. núm. 63.

[65] G. Chu, P. J. Stuckey, Dominance driven search, in: Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming, Springer, 2013, pp. 217–229.

[66] S. Sahni, T. Gonzalez, P-complete approximation problems, Journal of the ACM (JACM) 23 (3) (1976) 555–565.

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

[68] C.-M. Li, F. Xiao, M. Luo, F. Manyà, Z. Lü, Y. Li, Clause vivification by unit propagation in cdcl sat solvers, Artificial Intelligence 279 (2020) 103197.