Automatic Identification and Exploitation of Dominance Relations in Constraint Optimization Problems

ZHONG, Zhuowei

A Thesis Submitted in Partial Fulfillment

of the Requirements for the Degree of

Doctor of Philosophy

in

Computer Science and Engineering

The Chinese University of Hong Kong

November 2022

Abstract

Constraint Optimization Problems (COPs) are ubiquitous and challenging in computer science. Constraint Programming (CP), which encompasses the Branch and Bound (BnB) algorithm augmenting with various constraint techniques, is a typical paradigm for solving COPs. Dominance relations describe relations among solutions of COPs, where some solutions are known to be subordinate compared to others concerning satisfiability of constraints and/or optimality of the objective. *Dominance breaking* is a technique to exclude dominated solutions from the search space and can speed up the solving time of the BnB algorithm for many real-life problems. Identification of dominance relations in COPs, however, usually requires a deep understanding of the problems and sometimes even ingenuity. In this thesis, we present several contributions to *automatic dominance breaking* in COPs.

First, we propose a theoretical framework to enable automatic dominance breaking for a class of COPs consisting of *efficiently checkable* objectives and constraints. Constraints for dominance breaking are usually of various forms for different problems. We overcome this difficulty by restricting the form of constraints to be *nogood constraints*, which are elementary constraints and can be handled easily in CP. Our framework formulates the generation of dominance breaking nogoods as constraint satisfaction. We give theorems on the sufficient conditions governing when the generated nogood constraints are valid dominance breaking constraints in COPs. Experimentation on a diversified set of benchmarks confirms the effectiveness of automatically generated nogoods, which compares favorably against manually identified dominance breaking constraints both in efficiency and ease of use.

Second, we further extend the applicability and improve the efficiency of the proposed framework with two theoretical and practical innovations. Our first innovation opens up possibilities to apply automatic dominance breaking on COPs with non-efficiently checkable constraints. We give conditions on when and how non-efficiently checkable constraints can be ignored in nogood generation, and yet our method still produces sufficient useful dominance breaking nogoods for solving COPs. The second innovation identifies redundant and useless dominance breaking nogoods in solving COPs. The constraint satisfaction problems for nogood generation are strengthened using the notion of *Common Assignment Elimination* to avoid generating nogoods that are redundant with respect to others, and thus the nogood generation time is reduced substantially. Our experimental results confirm the enhanced applicability of our theory-backed innovations, which allow us to tackle previously impractical benchmarks.

Third, we enable automatic dominance breaking for COPs containing nested function calls. Such COPs are usually transformed into normalized COPs with introduced variables and functional constraints. We generalize the theory of dominance to normalized COPs and propose a rewriting system for automatic derivation of sufficient conditions in nogood generation based on functional constraints and their properties such as monotonicity, commutativity, and associativity. Experimentation on various benchmarks confirms the efficiency of the generated nogoods to solve problems with ineffective or no known dominance breaking constraints. Even when nogoods are costly to generate, we give case studies to show how we discover new dominance (symmetry) breaking constraints by recognizing the nogood patterns of small instances.

The proposed framework also sharpens our understandings of dominance breaking methods. We give case studies on various COPs comparing the strength of the dominance breaking constraints given in the literature with the generated dominance nogoods, and demonstrate the generality of our proposed framework.

摘要

約束優化問題在計算機科學中常見且充滿挑戰性的問題。約束規劃在分支定界算法上加入各種束技術，是解決約束優化問題的典型方法。其中，約束優化問題中的支配關係描述了問題解之間的關係，其中一些解與其他解相比在約束的可滿足性和/或目標函數的最優性的比較上是次優的。通過添加額外的約束，支配破除這項技術可以利用支配關係來消除次優的賦值。在許多現實問題中，支配破除可以通過大幅減小搜索空間來加快分支定界算法的求解時間。然而，識別約束優化問題的支配破除約束通常需要對問題有深入的了解，有時甚至需要獨創性的想法。在本論文中，我們對約束優化問題中對自動支配破除做出了一些貢獻。

首先，我們針對一類由可被有效檢查的目標函數和約束組成的約束優化問題提出了一個理框架以實現自動支配破除。支配破除通常需要在問題中加入支配破除約束，主要困難在於針對不同的問題有這樣的約束有不同的形式。我們通過將約束的形式限制為衝突約束來克服這個困難。它們在約束規劃中是基本的約束並且可以很容易地處理。我們提出的理論框架是首次在一類約束優化問題中實現自動支配破除。它將支配關係的識別有系統地表達成尋找一對部分賦值的約束滿足問題。我們給出定理來確定一個賦值支配另一個賦值的充分條件。被支配的部分賦值的否定語句就是支配破除衝突。通過對一組多樣化基準測試問題的實驗，我們證實了自動生成的衝突約束的有效性。它們在效率和易用性上都優於人工識別的支配破除約束，並且在求解時間上（包括衝突約束的生成和問題求解的時間）有幾個數量級的加速。

其次，我們通過兩個理論和實踐的創新進一步擴展了所提出框架的適用性並提高了效率。我們的第一個創新讓自動支配破除可以應用到具有非可被有效檢查的約束的優化問題上。我們給出了關於何時以及如何在衝突約束的生成過程中忽略不可被有效檢查的約束，而同時我們的方法仍然會產生足夠有用的衝突約束。第二項創新指出了在解決約束優化問題時多餘和無用的支配破除衝突約束。我們通過共同賦值移除的概念加强了衝突約束生成的可滿

足問題，從而大大減少了衝突約束的生成時間。我們的實驗結果證實了我們的這一有理論支持的方法的增强適用性，這使我們能夠解決原本不可解決的基準測試問題。

第三，我們令自動支配破除的方法可以應用到包含嵌套函數調用的約束優化問題。這樣的約束優化問題通常會轉換為具有引入變量和功能約束的標準化優化問題。我們將我們的理論框架推廣到歸一化的約束優化問題，並提出了一個自動化重寫系統用於基於函數式約束及其單調性、交換性和關聯性等屬性自動推導生成衝突約束的充分條件。對各種基準的實驗證實了生成的衝突約束可以有效求解具有無效或沒有已知的支配破除的問題。即使生成衝突約束的成本較高的時候，我們也提供案例研究來展示我們如何通過識別小問題實例中的衝突約束來發現新的支配（對稱）破除約束。

我們所提出的框架還加深了我們對手動得出的支配性破壞約束的理解。我們給出了關於各種約束優化問題的案例研究，比較了文獻中的支配破除約束與生成的支配破除衝突約束的強度，並證明了我們提出的框架的普適性。

List of Tables

List of Figures

Acknowledgments

First and foremost, I would like to express my heartfelt thanks to my supervisor Jimmy Ho Man Lee for his continuous support and guidance during my postgraduate study. Thanks for his clear introduction to constraint programming, which arouses my interest in conducting research in this field, and his patience in allowing me to explore different topics in the field. I was uncertain about our work on automatic dominance breaking at the beginning of my study, and this work would not have been possible without Jimmy's insights. His demonstration of rigorous academic writing and intriguing presentation sets a high standard that I wish I could also achieve one day. I am grateful for his support for me in attending international conferences, where I can talk to talented researchers from all over the world. Traveling overseas during the pandemic is difficult, and Jimmy and Scarlet's help is indispensable to my return to Hong Kong.

I would like to thank Prof. Fung Yu Young, Prof. Siu On Chan, and Prof. Peter Stuckey for serving as members of my thesis assessment committee. Their interesting questions and constructive comments from different perspectives greatly help me in preparing and polishing this dissertation. Special thanks also go to Dr. Jasper Lee, Prof. Wing Cheong Lau, and Prof. Wei-Shi Zheng for their supervision during different periods of my research journey.

I am indebted to my family for supporting me in all aspects of my life. Thanks to my aunts and my cousins in Hong Kong, from whom I could always receive warm support when in need. The home-cooked meal in their house relieves my homesickness a lot. I wish to thank Eliza for training my cooking skills and sharing many wonderful moments with me. I owe a lot to my grandparents for their understanding, and I wish I could have spent more time with them in the last three years. My loving father and mother are always good friends in my life, and I could not come this far without their company and encouragement.

Contents

Chapter 1

Introduction

Constraint Optimization Problems (COPs) are ubiquitous in practice, and their applications include scheduling [7, 46, 48], planning [15, 22, 48], transport routing [14, 61, 75, 76], etc. Constraint Programming (CP) is a classical pillar of artificial intelligence and a typical approach for solving COPs. Modern constraint solving technology allows users to specify a problem in the form of a *constraint model* consisting of a set of variables, a set of constraints, and possibly an objective function to be optimized, and the model is submitted to a constraint programming solver, which returns solutions by the backtracking search augmented with various constraint techniques. Such a problem-solving method takes us one step closer to the Holy Grail of Computer Science [37, 38]: the user states the problem, the computer solves it.

The efficiency of the search algorithm depends highly on structural properties of COPs, among which the existence of dominance relations is of practical importance. Dominance relations are relations over the set of assignments, i.e. the value combinations assigned to variables of COPs. If one set of assignments dominates another, the latter is known to be subordinate to the former with respect to the satisfaction of constraints and the objective value. When solving COPs, we are only interested in optimal solutions with the best objective value, and *dominance breaking* is a method to exclude dominated assignments from the search space of COPs in order to improve the solving efficiency.

Example 1. *Consider the 0-1 knapsack problem where there is a set of items and a knapsack with a capacity limit W. Each item has a weight and a profit. The problem requires to select a subset of items with the maximal total profit subject to the constraint that the total weight of chosen items cannot exceed W. The following gives an example model for a problem instance, where x_i takes the value* 1 *if and only if item i is chosen.*

$$
\begin{aligned}
&\textit{maximize } 3x_1 + x_2 + 6x_3 + 4x_4 \\
&\textit{subject to } x_1 + 2x_2 + 3x_3 + 4x_4 \leq 5 \\
&\qquad x_i \in \{0, 1\} \textit{ for } i = 1, \ldots, 4
\end{aligned}
\tag{1.1}
$$

We can 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 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 a larger total profit but less total weight. If T_1 is feasible, then T_2 is also feasible with a better objective value. Therefore, T_1 must not be an optimal solution that should have the maximum total profit. By a similar reasoning process, if a solution include the fourth item and exclude the third item, then the solution must not be an optimal solution. We can formulate the statements as dominance breaking constraints and add them to the COP without changing the optimal objective value of the COP as follows.*

$$
\begin{aligned}
&\textit{maximize } 3x_1 + x_2 + 4x_3 + 6x_4 \\
&\textit{subject to } x_1 + 2x_2 + 4x_3 + 3x_4 \leq 5 \\
&\qquad x_1 \geq x_2, x_4 \geq x_3 \\
&\qquad x_i \in \{0, 1\} \textit{ for } i = 1, \ldots, 4
\end{aligned}
\tag{1.2}
$$

Note that the additional constraints $x_1 \geq x_2$ and $x_4 \geq x_3$ can make some suboptimal solutions of (1.1) become infeasible in (1.2). For example, the subset choosing the first and the fourth item is a solution for (1.1), while it violates $x_1 \geq x_2$ and $x_4 \geq x_3$. Therefore, dominance breaking constraints reduce the search space of a COP.

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, and dominated solutions violating these constraints will be pruned in the BnB algorithm. A wealth of research works apply dominance breaking in solving COPs in practice, and empirical evidence shows that dominance breaking can dramatically reduce the search space and speed up the solving process [55, 70, 71, 3, 106, 97].

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 [24]. Additional techniques are required to select a subset of dominance breaking constraints to prune suboptimal solutions as many as possible, while the optimal objective value of a COP is preserved.

The success of past research often requires sophisticated insights into problem structures, and the methods for dominance breaking are problem-specific and non-trivial to transfer from one problem domain to another. There are few works [23, 92] on generic methods for identifying and exploiting dominance relations in COPs, but they still rely on manual and sophisticated derivation, and require insights into the problems.

1.1 Contributions

This thesis presents a formal framework to automate the process of dominance breaking and make it accessible to even non-experts of constraint programming. Our contributions are composed of three parts. First, we present the theory to formulate the derivation of dominance breaking *nogood constraints* mechanically as constraint satisfaction for a class of COPs consisting of *efficiently checkable (EC)* objectives and constraints. We also give practical implementations and empirically demonstrate the effectiveness of our proposed method in reducing the search space for solving various COPs. Second, we provide optimization techniques in order to extend the applicability of the framework for more problems. In particular, we relax the restriction that requires all constraints to be efficiently checkable and improve the efficiency of nogood generation by adding additional constraints to avoid generating useless nogood constraints. Third, we further generalize the framework to COPs consisting of various objective and constraints with nested function calls, which are usually flattened/normalized into functional constraints. We present an automated method that exploits the properties of functional constraints to derive constraints in the problem for nogood generation. Our work in this thesis can provide deeper understanding on dominance relations. We also compare the strength of generated dominance breaking and enable dominance breaking to be applied in more COPs.

1.2 Thesis Outline

The rest of this thesis is organized as follows.

Chapter 2 provides basic backgrounds in constraint programming and dominance relations. We give formal definitions of constraint satisfaction and optimization problems and explain the core concepts, including branch and bound search, constraint propagation, and nogood constraints in constraint programming. Next, we review basic definitions in relational mathematics, based on which we define dominance relations in COPs. We also describe the method of dominance breaking with the help of examples.

Chapter 3 gives the main and recent works on the identification of dominance relations and nogood generation, and also relevant works on static and dynamic dominance breaking methods in constraint programming.

The remaining chapters introduce the main parts of our contributions.

Chapter 4 gives a formal framework for generating dominance breaking constraints for a class of COPs. The key difficulty in generating dominance breaking constraints is that they are in different forms for different problems. To overcome this difficulty, we propose to generate constraints only in the form of *nogood constraints* derived from the negation of dominated assignments in COPs. We give theorems to prove the soundness of our method by giving sufficient conditions of when an assignment dominates another and when multiple generated nogood constraints are compatible based on the objective and constraints.

Chapter 5 extends the applicability of the framework of automatic dominance breaking with two theoretical and practical innovations. First, we relax the restriction that all constraints in a COP are required to be efficiently checkable and formally prove when and how to ignore non-EC constraints in nogood generation. Second, we identify that some generated nogoods make no contributions to search space reduction in a constraint programming solver, and propose the notation of *common assignment elimination* to ban the generation of such fruitless nogoods, thus speeding up the generation process substantially.

Chapter 6 proposes to exploit functional constraints to identify useful dominance relations in COPs with nested function calls. We generalize the theory of automatic dominance breaking to normalized COPs that contain functionally defined variables and functional constraints, and present a rewriting system to derive sufficient conditions automatically for dominance relations in COPs based on functional constraints and common functional properties such as monotonicity, commutativity and associativity. We also show how to use automatically identified nogood constraints to construct more compact constraints for dominance breaking.

At the end of Chapters 4 to 6, experimentation on various COPs demonstrates the effectiveness of our proposals and the efficiency gained by the addition of automatically generated dominance breaking constraints. To formally investigate the generality of our framework, we give case studies in Chapter 7 to compare the logical strength of the generated nogood constraints with existing dominance breaking constraints derived manually in the literature. In Chapter 8, we conclude the thesis and discuss possible directions for future works.

Chapter 2

Background

In this chapter, we give preliminaries in constraint satisfaction and optimization, constraint programming, and dominance relations in constraint optimization problems.

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. We use $\bar{\theta}$ to emphasize that an assignment is a *full assignment* whose scope is X, or it is otherwise a *partial assignment*. Let $\theta[x_i] = v_i$ denote the value assigned by θ to variable $x_i \in var(\theta)$ if the literal $(x_i = v_i) \in \theta$, and we also let $\theta[S'] = \{(x_i = v_i) \in \theta \mid x_i \in S'\}$ to denote the *projection* of an assignment θ over a set of variables $S' \subset var(\theta)$. We also define $\mathcal{D}^X_\theta = \{\bar{\theta} \in \mathcal{D}^X \mid \bar{\theta}{\downarrow_S} = \theta\}$ as the set of full assignments extending from a partial assignment $\theta \in \mathcal{D}^S$ where $S \subseteq X$.

A constraint $c \in C$ is a set of assignments over variables in a set $var(c)$. An assignment θ *satisfies* a constraint c if and only if $var(\theta) \supseteq var(c)$ and $\theta[var(c)] \in c$. 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 $\neg\theta \equiv \vee_{x \in var(\theta)}(x \neq \theta[x])$, and its *length* is equal to the scope size $|var(\theta)|$. A *decision for a variable* $x \in var(\theta)$ in a nogood constraint $\neg\theta$ is the literal $x = \theta[x]$. A decision for a variable x is *subsumed* if and only if $D(x) = \{\theta[x]\}$ and is *falisified* if and only if $\theta[x] \notin D(x)$.

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

Example 2. *Consider a COP $P = (X, D, C, f)$ where $X = \{x_i \mid i = 1, \ldots 4\}$, $D(x_i) = \{0, 1, 2\}$ for $i \in \{1, 2, 3, 4\}$, and $C = \{3x_1 + 2x_2 + 3x_3 + x_4 \geq 6, x_1 \neq x_3, x_2 < x_4\}$. The objective function f is $x_1 - 4x_2 + 3x_3 - 5x_4$. The constraint $c \equiv (x_1 \neq x_3)$ is over the scope $\{x_1, x_3\}$ and can be represented explicitly as a set of assignments consisting of $\{x_1 = 0, x_3 = 1\}$, $\{x_1 = 0, x_3 = 2\}$, $\{x_1 = 1, x_3 = 0\}$, $\{x_1 = 1, x_3 = 2\}$, $\{x_1 = 2, x_3 = 0\}$, and $\{x_1 = 2, x_3 = 1\}$. The full assignment $\bar{\theta} = \{x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 0\}$ is not a solution since it violates the constraint c, i.e. $\bar{\theta}[\{x_1, x_3\}] = \{x_1 = 0, x_3 = 0\} \notin c$. Another full assignment $\bar{\theta}' = \{x_1 = 1, x_2 = 2, x_3 = 0, x_4 = 0\}$ is a solution of the problem and has the objective value $f(\bar{\theta}') = 1 - 4 \times 2 + 2 \times 0 - 5 \times 0 = -7$. The optimal solution for P is $\bar{\theta}_{opt} = \{x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 2\}$ which has the smallest objective value -13.*

2.2 Constraint Programming

Constraint programming (CP) is a paradigm for solving constraint satisfaction and optimization problems. In this section, we introduce the Branch and Bound (BnB) algorithm and constraint propagation in constraint programming. We also discuss nogood constraints which are closely related to the framework of automatic dominance breaking.

2.2.1 Branch and Bound

CSPs and COPs are usually NP-hard. To find a solution of a CSP P, the problem is usually branched into several subproblems $\{P_1, \ldots, P_s\}$ by splitting the domain in such a way that $sol(P) = sol(P_1) \cup \cdots \cup sol(P_s)$. In this thesis, we consider *binary branching* in which a problem is divided into two subproblems $P_l = (X, D_l, C)$ and $P_r = (X, D_r, C)$. The domain D_l and D_r are the same as D except for one variable x, where $D_l(x) = \{v\}$ and $D_r(x) = D(x) \setminus \{v\}$. We call x the *branching variable* and $v \in D(x)$ is the *branching value*.

Example 3. *Consider the CSP $P_0 = (X, D_0, C)$ derived from the COP in Example 2 without the objective function where D_0 is the initial domain without splitting. Figure 2.1 shows the branching of P_0 into subproblems P_l and P_r by selecting x_1 as the branching variable and 0 as the branching value. The domains D_l and D_r are the same as D_0 except that $D_l(x_1) = \{0\}$ and $D_r(x_1) = \{1, 2\}$. The subproblems P_l and P_r can be further branched into subproblems by selecting a branching variable or a branching value from the domain of the next branching variable.*

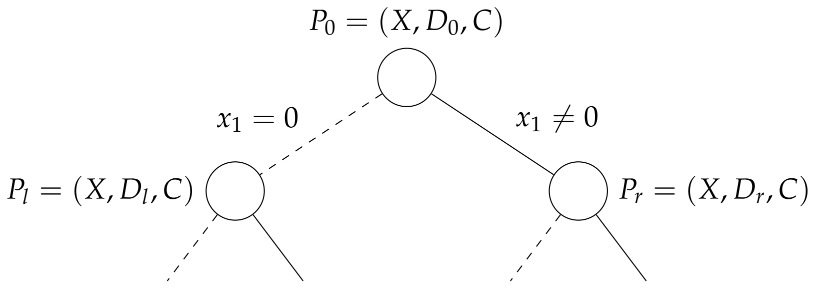


Figure 2.1: *The branching of a constraint satisfaction problem*

The branching process can be structured as constructing a *search tree* $\mathcal{T} = (\mathcal{P}, \mathcal{E})$, where $\mathcal{P}$ is a set of CSPs as the nodes in the tree, and the original CSP P_0 is the root node. When there is an arc $(P, P') \in \mathcal{E}$, P' is generated from P by a branching, and we say that P' is a *child* of P. Each problem $P = (X, D, C) \in \mathcal{P}$ is *associated with* a partial assignment θ if and only if for all variables $x_i \in var(\theta)$, $D(x_i) = \{\theta[x_i]\}$. A node P is a *failure node* if either D is a domain with $D(x) = \emptyset$ for some variable $x \in X$ or a constraint is violated by the associated partial assignment of P. A node P is a *solved node* if it is not a failure node and D is a *singleton* domain such that $|D(x)| = 1$ for all variables $x \in X$. A *leaf node* is either a failure node or a solved node.

In constraint programming, the branch and bound algorithm solves a COP as a series of CSPs with decreasing bars for the objective value. An objective variable, say obj, is defined and linked to the objective function. Whenever a solution is found, a *bounding constraint* $obj < v$ is added where v is the objective value of the newly found solution. This effectively construct a new CSP which only search for a solution with a better objective value. The process is repeated until no solutions can be found for the last CSP, which proves the optimality of the latest solution. The search tree is usually traversed by the depth first search, in which the search algorithm performs the following operations at each node:

1. If the current node is a failure node, then return.
2. If the current node is a solved node, then add a new bounding constraint and return.
3. Otherwise, the current node is a non-leaf node, and then:
   (a) Generate the left child node and recursively traverse the left subtree
   (b) Generate the right child node and recursively traverse the right subtree

Example 4. *Figure 2.2 shows a partial search tree for the COP $P = (X, D_0, C, f)$ in Example 2 using the BnB search algorithm, where failure nodes and solved nodes are highlighted in red and blue respectively. Initially, a CSP $P_0 = (X \cup \{obj\}, D_0, C \cup \{c_{obj}\})$ is constructed from the COP as the root node of the search tree, where $c_{obj} \equiv (obj = x_1 - 4x_2 + 3x_3 - 5x_4)$ is the constraint to link the newly defined variable to the objective function f. Each non-leaf node has exactly two child nodes, and each edge is labelled by the branching decision. The indices of the subproblems follow the order in which the nodes are visited. We give the description of the first several steps of the BnB search process as follows:*

1. *The search starts from the CSP P_0. It is neither a failure node nor a solved node, and the left child node $P_1 = (X \cup \{obj\}, D_1, C \cup \{c_{obj}\})$ is generated such that $D_1(x_i) = \{0, 1, 2\}$ for*

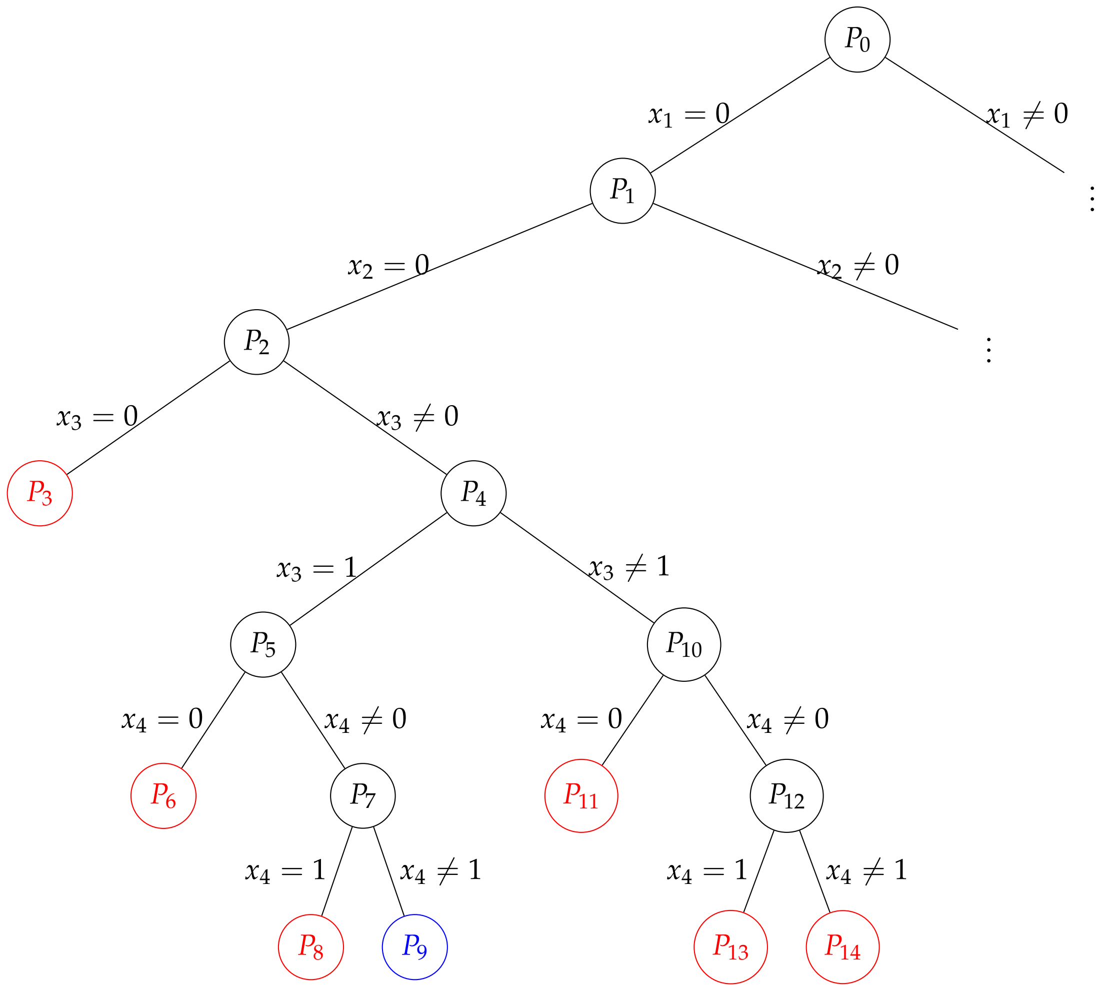


Figure 2.2: *A partial search tree for the example COP*

$i = 2, 3, 4$ *and* $D_1(x_1) = \{0\}$.

2. *The branching process repeats and the subproblems* P_2 *and* P_3 *are generated. When visiting* P_3, *it associates with a partial assignment* $\theta_3 = \{x_1 = 0, x_2 = 0, x_3 = 0\}$. *Since the constraint* $x_1 \neq x_3$ *is violated by* θ_3, P_3 *is a failure node.*

3. *The search is backtracked to* P_2, *and the right child node* $P_4 = (X, D_4, C)$ *is generated such that* $D_4(x_1) = D_4(x_2) = \{0\}$, $D_4(x_3) = \{1, 2\}$ *and* $D_4(x_4) = \{0, 1, 2\}$. *We select* x_3 *as the next branching variable and* 1 *as the branching value to generate the subproblem* P_5.

4. *The search continues to generate and visit subproblems* P_5, P_6, P_7 *and* P_8. *The assignment* $\bar{\theta}_6 = \{x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 0\}$ *associated with* P_6 *violates the constraint* $x_2 < x_4$, *and the assignment* $\bar{\theta}_8 = \{x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 1\}$ *violates the constraint* $2x_1 + 2x_2 + 3x_3 + x_4 \geq 5$. *Therefore, both* P_6 *and* P_8 *are failure nodes.*

5. *The next subproblem* $P_9 = (X \cup \{obj\}, D_9, C \cup \{c_{obj}\})$ *is a solved node since the domain of all variables are singleton, and no constraints are violated. We have found a solution* $\bar{\theta}_9 = \{x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 2\}$ *for the original problem, and the objective value is* $f(\bar{\theta}_9) = 0 - 4 \times 0 + 3 \times 1 - 5 \times 2 = -7$. *A bounding constraint* $obj < -7$ *is added to enforce future solutions to have smaller objective value.*

6. *The search is backtracked to* P_4 *and its right subtree is visited. There are three failure nodes in the subtree under* P_{10}. *The subproblem* P_{11} *is a failure node since its associated assignment violates* $x_2 < x_4$ *similarly as* P_6. *The assignments associated with* P_{13} *and* P_{14} *are* $\{x_1 = 0, x_2 = 0, x_3 = 2, x_4 = 1\}$ *and* $\{x_1 = 0, x_2 = 0, x_3 = 2, x_4 = 2\}$ *respectively. They do not violate any constraint in the original constraint set* C, *but their objective values are larger than* -7, *which violates the bounding constraint added in* P_9. *Therefore,* P_{13} *and* P_{14} *are also failure nodes, and the search is backtracked without adding new bounding constraints.*

2.2.2 Constraint Propagation

An important constituent of constraint programming is *constraint propagation*, which tightens the domain by removing inconsistent values that can no longer be part of any solution. In this way, some subtrees containing no solutions can be removed directly without exploration.

In a constraint programming solver, each constraint c has a *propagator* to remove values from $D(x)$ for all variables $x \in var(c)$. Formally, a domain D_1 is *stronger* than a domain D_2, written $D_1 \sqsubseteq D_2$, if $D_1(x) \subseteq D_2(x)$ for all variables x. A propagator *prop* is a monotonically decreasing function mapping domains to domains; that is, $prop(D) \sqsubseteq D$ for all domains D, and $D_1 \sqsubseteq D_2$ implies that $prop(D_1) \sqsubseteq prop(D_2)$. A propagator $prop_1$ is *stronger* than another propagator $prop_2$ if $prop_1(D) \sqsubseteq prop_2(D)$ for all domains D.

A propagator usually maintains a *consistency notation* which is a condition on domains with respect to constraints. There are many kinds of consistency notations, among which the most common one is *arc consistency (AC)* for a binary constraint over a scope $var(c) = \{x_{i_1}, x_{i_2}\}$ consisting of two variables. Arc consistency requires that for each value $v_1 \in D(x_{i_1})$, there is a value $v_2 \in D(x_{i_2})$ such that the assignment $\{x_{i_1} = v_{i_1}, x_{i_2} = v_{i_2}\} \in c$. Arc consistency can be extended to *generalized arc consistency (GAC)* [90] for constraints with more than two variables. The domain of a COP $P = (X, D, C, f)$ is in GAC for a constraint $c \in C$ if for every $x_{i_j} \in var(c)$ and $v \in D(x_{i_j})$, there exists a full assignment $\bar{\theta} \in \mathcal{D}^X$ such that $\bar{\theta}[x_{i_j}] = v$ and $\bar{\theta}$ satisfies c.

Example 5. *Consider the problem in Example 2 and the constraint $x_2 < x_4$. To enforce GAC on the domain, the propagator for $x_2 < x_4$ removes 2 from $D_0(x_2)$ since no values in $D_0(x_4)$ is strictly larger than 2. Similarly, the value 0 can be removed from $D_0(x_4)$, and the domain of the problem becomes $D_0(x_1) = D_0(x_3) = \{0, 1, 2\}$, $D_0(x_2) = \{0, 1\}$, and $D_0(x_4) = \{1, 2\}$.*

In BnB search, whenever a node P is generated, we enforce consistency and remove values from the domain of P by a *constraint propagation algorithm*, which repeatedly triggers propagators of all constraints in P until a *fix point* in which no propagators can further remove values from the domain. Algorithm 1 shows a classical constraint propagation algorithm called *AC-3* [89] to enforce AC for a binary constraint problem, and it can be extended to enforce GAC for arbitrary constraints [118]. The algorithm starts with initializing the active constraint set Q to be the set C of all constraints. In each iteration, a constraint c is removed from Q and its associated propagator is executed to detect and remove inconsistent

Algorithm 1 AC-3/GAC-3

```
Input: a CSP $P = (X, D, C)$
1:  $Q \leftarrow C$
2:  while $Q$ is not empty do
3:      Choose and remove a constraint $c$ from $Q$
4:      $D' \leftarrow Propagate(c, D)$
5:      if $\exists x \in X, D'(x) \neq D(x)$ then
6:          $X' \leftarrow \{x_i \mid x_i \in var(c)$ where $D'(x_i) \neq D(x_i)\}$
7:          $Q \leftarrow Q \cup \{c' \in C \setminus \{c\} \mid X' \cap var(c) \neq \emptyset\}$
8:          $D \leftarrow D'$
9:      end if
10: end while
```

values, which returns a potentially updated domain D' (line 4). If $D'(x) \neq D(x)$ for some variable x, some values must have been removed, and the consistency for a constraint c with $x \in var(c)$ may no longer hold. Therefore, all constraints involving x are added to Q and the domain D is updated to D' before the next iteration (line 6-8). The iteration continues until Q is empty. The naive implementation of the AC-3 algorithm has the worst-case time complexity $O(|C|d^3)$, where $d = \max_{x_i \in X} D(x_i)$. The GAC-3 algorithm is known to run in $O(|C|r^3 d^{r+1})$ time and $O(|C|r)$ space to enforce GAC on the domain, where $r = \max_{c \in C} |var(c)|$ is the largest arity, i.e. the number of variables in a constraint. The incremental version of AC-3, which is called AC-3.1 [132], can improve the time complexity to $O(|C|d^2)$ by a more fine-grained implementation, and the time complexity of GAC-3.1 is $O(|C|r^2 d^r)$ [13]. After the execution of the constraint propagation algorithm, the updated domain must be in AC/GAC for every constraint $c \in C$.

Example 6. *Consider the partial search tree in Figure 2.2. The values are removed by constraint propagation, and therefore the search will not explore some subtrees without solutions. Initially, the domain D_0 is reduced by the propagator of $x_2 < x_4$ as shown in Example 5. No propagators of other constraints further remove values in the constraint propagation at P_0. At the node P_1, the domain $D_1(x_1) = \{0\}$, and 0 is removed due to the constraint $x_1 \neq x_3$. Hence, the node P_3 will not be generated and visited. Similarly, $D_2(x_2) = \{0\}$ at node P_2, and 0 is removed from $D_2(x_4)$ by the propagator of $x_2 < x_4$. The search will not generate and visit the subproblem P_6 and P_{11} where the*

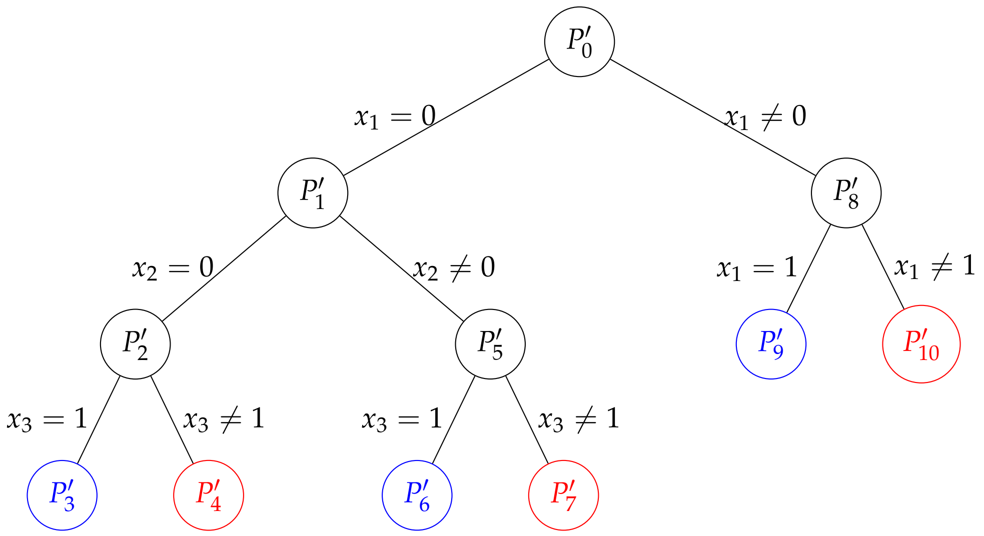


Figure 2.3: *The complete search tree for the example COP with constraint propagation*

domain of x_4 is set to $\{0\}$.

Figure 2.3 shows the complete search tree with GAC-3 executed at each search node, and the number of nodes in the search tree is reduced significantly with constraint propagation.

2.2.3 Nogood Constraints

A *nogood* is a partial assignment that is not part of any solution of a COP P. An assignment constraint for an assignment θ is the conjunction of literals in θ in the form $\wedge_{x \in var(\theta)}(x = \theta[x])$, while a *nogood constraint* $\neg\theta \equiv \vee_{x \in var(\theta)}(x \neq \theta[x])$ is the negation of the assignment constraint for θ. We say that the *length* of a nogood $\neg\theta$ is always equal to the scope size $|var(\theta)|$. A nogood constraint represents a partial assignment that is not part of any solution.

Nogood constraints are the most general forms of constraints. By definition, an arbitrary constraint c over its scope $var(c) = \{x_{i_1}, \ldots, x_{i_k}\}$ is a subset of the Cartesian product $D(x_{i_1}) \times \cdots \times D(x_{i_k})$ which is a set of partial assignments $\{\theta_i \mid var(\theta_i) = var(c) \wedge \theta_i \in c\}$. Note that a partial assignment θ over $var(c)$ either satisfies c or violates it, and we can construct a set $vio(c) = \{\theta_i \mid var(\theta_i) = var(c) \wedge \theta_i \notin c\}$ of assignments. A full assignment $\bar{\theta}$ satisfying c must have the property that $\bar{\theta}[var(c)] \notin vio(c)$. Thus, c is equivalent to the conjunction $\wedge_{\theta \in vio(c)} \neg\theta$ of a set of nogood constraints.

Example 7. *As shown in Example 2, the constraint $c \equiv (x_1 \neq x_3)$ can be represented explicitly as a set of assignments over $\{x_1, x_3\}$. Alternatively, we can construct the set $vio(c)$ consisting of $\{x_1 = 0, x_3 = 0\}$, $\{x_1 = 1, x_3 = 1\}$ and $\{x_1 = 2, x_3 = 2\}$. The constraint c is equivalent to the conjunction*

$$\wedge_{\theta \in vio(c)} \neg\theta \equiv (x_1 \neq 0 \vee x_3 \neq 0) \wedge (x_1 \neq 1 \vee x_3 \neq 1) \wedge (x_1 \neq 2 \vee x_3 \neq 2)$$

The domain of a COP is usually enforced to be GAC with respect to a nogood constraint. A *decision for a variable $x \in var(\theta)$* in a nogood constraint is an equality constraint $x = \theta[x]$. A decision for a variable x is *subsumed* if and only if $D(x) = \{\theta[x]\}$ and is *falisified* if and only if $\theta[x] \notin D(x)$. By definition, the domain of a COP $P = (X, D, C, f)$ is GAC with respect to a nogood constraint $\neg\theta$ if and only if (a) there exists two decisions that are not subsumed, or (b) there is one decision that is falisified. The propagator maintaining GAC for a nogood constraint is triggered when all but one of the decisions are subsumed, and the value $\theta[x]$ is removed from $D(x)$ for the left decision $x = \theta[x]$.

Proposition 1. *Let θ and $\tilde{\theta}$ be two partial assignments in a COP P such that $\tilde{\theta}$ is a subset of θ. If $prop_1$ and $prop_2$ are propagators maintaining GAC for $\neg\tilde{\theta}$ and $\neg\theta$ respectively, then $prop_1$ is stronger than $prop_2$.*

Proposition 1 implies that the propagator of $\neg\theta$ is *propagation redundant* [20] with respect to $\neg\tilde{\theta}$ and contributes no extra pruning in a constraint solver. Therefore, constraint propagation results in the same domain after removing $\neg\theta$, and the runtime cost of the propagator for $\neg\theta$ is reduced.

2.3 Dominance Relations and Dominance Breaking

In this section, we introduce some core concepts of dominance in COPs and the methods to exploit dominance relations in constraint programming.

2.3.1 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* if and only if (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 if and only if $\forall m \in M, (m, m) \notin R$. The *transitive closure* R^+ of a homogeneous relation R over a set M is the smallest relation over M that contains R and is transitive. In this work, we slightly generalize the transitive closure to the heterogeneous relation $R : M \mapsto N$ by a two-step construction: (1) construct a relation $\hat{R}$ over the union $\hat{M} = M \cup N$ such that $\hat{R} = R$ as a set of ordered pairs, and (2) take the transitive closure $\hat{R}^+$ of $\hat{R}$ over the set $\hat{M}$.

2.3.2 Dominance Relations

We define a dominance relation to be a homogeneous relation over the set of all full assignments in a COP.

Definition 1. *[23] 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.

A full assignment that is dominated by another in a dominance relation can be removed from the solution space since it cannot be an optimal solution of P.

Theorem 1. *[23] 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 optimal value of P.*

Theorem 1 is the basis of the theoretical framework proposed in this thesis, in which we prove the soundness of automatically derived dominance breaking constraints in a COP by showing that all full assignments satisfying the constraints are dominated by other full assignments in a constructed dominance relation for the COP.

Note that a dominance relation should be both transitive and irreflexive. In other words, if P is satisfiable, then there is at least one optimal solution $\bar{\theta}_{opt}$ of P that is not dominated by any other solutions. For example, consider a simple COP P with one variable x and $D(x) = \{0, 1\}$, where the objective function is constant. There are two full assignments $\bar{\theta} = \{x = 1\}$ and $\bar{\theta}' = \{x = 0\}$ in P. Both $\bar{\theta}$ and $\bar{\theta}'$ are optimal solutions of P. Given a dominance relation $\prec$ with respect to P, either $\bar{\theta} \prec \bar{\theta}'$ or $\bar{\theta}' \prec \bar{\theta}$, but not both. Otherwise, we will have $\bar{\theta} \prec \bar{\theta}$ or $\bar{\theta}' \prec \bar{\theta}'$ by the transitive property, but it violates the irreflexive property of a dominance relation.

The definition of dominance relations can be generalized to search nodes.

Definition 2. *[23] Let $\mathcal{D}_1, \mathcal{D}_2 \subseteq \mathcal{D}^X$ be the sets of full assignments of two search nodes for a COP $P = (X, D, C, f)$. If $\forall \bar{\theta}' \in \mathcal{D}_2$, $\exists \bar{\theta} \in \mathcal{D}_2$ such that $\bar{\theta} \prec \bar{\theta}'$, then we say $\mathcal{D}_1 \prec \mathcal{D}_2$.*

By Theorem 1, if $\mathcal{D}_1 \prec \mathcal{D}_2$, then we can safely prune the search node with $\mathcal{D}_2$ without changing the optimal objective value.

Example 8. *Consider the COP in Example 2. If we exhaust all full assignments in $\mathcal{D}^X$, we can find that the optimal solution $\bar{\theta}_{opt} = \{x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 2\}$. A simple dominance relation over $\mathcal{D}^X$ can be constructed such that $\bar{\theta}_{opt} \prec_1 \bar{\theta}$ for all $\bar{\theta} \in \mathcal{D}^X \setminus \{\bar{\theta}_{opt}\}$. Another possible dominance relation is that $\bar{\theta} \prec_2 \bar{\theta}'$ for all pairs $(\bar{\theta}, \bar{\theta}')$ such that $\bar{\theta}[x_1] = \bar{\theta}'[x_3] = a$, $\bar{\theta}[x_3] = \bar{\theta}'[x_1] = b$, $\bar{\theta}[x_2] = \bar{\theta}'[x_2]$ and $\bar{\theta}[x_4] = \bar{\theta}'[x_4]$, where $(a, b) \in \{(1, 0), (2, 0), (2, 1)\}$. For*

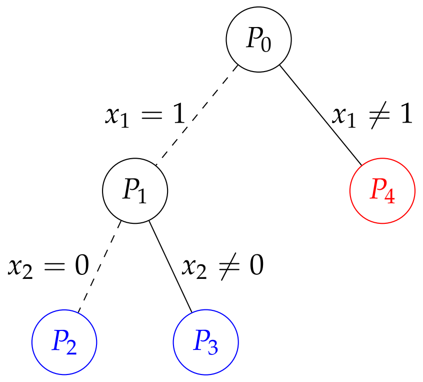


Figure 2.4: *The complete search tree for the example COP with dominance breaking constraints*

instance, $\{x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0\} \prec_2 \{x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 1\}$*, where the former is a solution while the latter is not a solution of* P*. This satisfies the first condition of Definition 1. One can also verify that other pairs of full assignments satisfy one of the three conditions in Definition 1.*

2.3.3 Dominance Breaking

Dominance Breaking is to remove solutions that are proved to be suboptimal with respect to satisfiability and/or objective value. A dominance breaking method is *sound* if it leaves at least one optimal solution in $sol(P)$ if the COP P is satisfiable. The static method to exclude dominated solutions is to add *dominance breaking constraints* in a COP that such that some dominated solutions in $sol(P)$ become non-solutions. Note that a constraint programming solver will not only check the validity for a dominance breaking constraint, but also perform constraint propagation. Values removed by dominance breaking constraints may further trigger the propagator of original constraints in C for domain reduction.

Example 9. *Consider the dominance relation* $\prec_2$ *in Example 8. A full assignment* $\bar{\theta}' = \{x_1 = a, x_2 = 0, x_3 = b, x_4 = 1\}$ *is dominated if* $(a, b) \in \{(1, 0), (2, 0), (2, 1)\}$*. To exclude such dominated full assignments, we can simply add* $x_1 \geq x_3$ *for dominance breaking. After adding this constraint, Algorithm 1 will further reduce the domain of the root node to be* $D_0(x_3) = D_0(x_2) = \{0, 1\}$ *and* $D_0(x_1) = D_0(x_4) = \{1, 2\}$ *after constraint propagation with the additional constraint. Figure 2.4 shows the new search tree with dominance breaking constraints, and it has fewer number of nodes* *compared with the tree in Figure 2.3.*

Dominance relations can also be exploited dynamically to prune suboptimal solutions during search, which requires more sophisticated dominance checking [63, 34] or even modifications of the branch and bound search algorithm [21, 24]. We will discuss methods to exploit dominance relations in more details in Chapter 3. In this thesis, we consider the method of static dominance breaking and automate the process of deriving dominance breaking constraints.

Chapter 3

Related Work

In this chapter, we give the main and recent works on the identification of dominance relations and nogood generation. We also give relevant works on static and dynamic dominance breaking methods in constraint programming.

3.1 Identification of Dominance Relations

Dominance relations are relations over the solution set of COPs regarding the satisfaction of constraints and the optimality of the objective. Dominated solutions are guaranteed to be suboptimal and hence can be removed, since the aim of solving COPs are to find an optimal solution with the best objective value. In the literature, dominance relations are usually identified in a case-by-case manner [3, 22, 46, 55, 70, 97, 106], and there are a few attempts to automate the identification of dominance relations in COPs. Yu and Wah [131] propose a machine learning method to find candidate dominance relations in several combinatorial problems. While the method can be applied to a class of problems, the generated candidate dominance relations lack correctness guarantees and requires further manual inspection. Fischetti et al. [31, 32] propose a *local dominance procedure* to detect dominance relations in mixed integer linear programs. The idea is to identify a dominating node for a given search node by solving an auxiliary optimization problem, either in an exact or heuristic manner, during the BnB search. Some design choices are required for efficient implementation, such as the heuristic selection of nodes to solve the auxiliary problem and the recording of nogoods to prevent redundant solving.

Chu and Stuckey [23, 24] give the first generic method for deriving dominance breaking constraints for COPs. The method starts with a set of candidate mappings over the solution set, followed by the derivation of dominance breaking constraints based on sufficient conditions for the candidate mappings to map solutions to better solutions. Manual efforts are required to select mappings, identify sufficient conditions, and simplify the dominance breaking constraints. Mears and de la Banda [92] automate the derivation process to a certain extent based on automated symmetry detection, which restricts the candidate mappings to symmetries that map solutions to solutions and map non-solutions to non-solutions. Their method still require manual selection of symmetries produced by the automated symmetry detection process to be effective.

Dominance relations are generalizations of symmetry relations in constraint programming [52]. Symmetry relations are bijective mappings on full assignments that have the same objective value and are both solutions/non-solutions. Dominance is a relation also on two full assignments, but one is "better" than the other in terms of satisfiability or objective values. 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, and $\bar{\theta}$ 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. A substantial amount of research works [27, 36, 43, 94, 109, 112] focus on the automatic detection of symmetry relations in a CSP. While the detected symmetries can be exploited to speed up the solution process of a CSP, the detection may introduce extra overheads. Other works [60, 114] propose methods to detect symmetries for a class of CSPs so that the detected model-level symmetries can be applied to all CSPs in the class. However, they can only detect relatively "simple" symmetries like interchangeable values and interchangeable variables. Later, Mears et al. [95, 96] proposes a method that lifts the detected symmetries at the instance level to the model level by an inductive reasoning process. Symmetries can also arise as a result of modeling decisions when a single problem solution corresponds to multiple assignments to the variables in a constraint model. The automated constraint modeling system CONJURE [1, 2] incorporates an automated and rule-based method to detect modeling symmetries when refining high-level specifications into constraint models.

3.2 Nogood Generation

To improve the efficiency of the search algorithm, nogood constraints are usually used to avoid search redundant subtrees or useless subtrees without desired solutions. Nogood constraints can be generated from *nogood learning* and *restarts* in constraint programming.

Nogood learning is a standard technique for improving backtracking search [29], and is a main reason for the success of propositional satisfiability (SAT) solvers. Nogood constraints, which correspond to clauses in SAT solvers, are generated from to failures in the search algorithm, and the *first unique implication point* method [98] is one of the most efficient learning schemes for analyzing the *implication graph* that encodes the information in constraint propagation. Nogood learning is also incorporated in constraint programming solvers. Katsirelos and Bacchus [65] propose *generalized nogoods* to include *non-assignments* which corresponds to values pruned from the domain of variables. They show that generalized nogoods are more compact than standard nogoods and can yield significant improvements in empirical performance of constraint solvers. Later, Ohrimenko et al. [102, 103] propose a hybrid method called *lazy clause generation*, which encodes domain changing behaviors from constraint propagators as clauses inside the SAT solver. The combination of powerful nogood learning in SAT solving and efficient propagation algorithm in constraint programming results in the state-of-the-art performance in various resource constrained scheduling problems [120, 119, 121, 74].

Restart [29, 6, 44] is a technique to make backtracking search more robust by invoking a new run of the search algorithm with different variable and value ordering heuristic.

Nogood constraints for restart [6, 79] encode the current explored search space to avoid redundant search in the next run of the search algorithm. Lee et al. [80] propose a global constraint to maintain and propagate nogoods from restarts more efficiently increasing nogood, and Glorian et al. [56] improve the filtering algorithm by mechanisms to combine nogoods dynamically.

3.3 Static Dominance Breaking

Static dominance breaking is a direct method to removes dominated solutions by adding additional *dominance breaking constraints*, which make dominated solutions become non-solutions, to a target COP before the BnB search. The method has been applied to speed up the solution process for various problems [55, 107, 51, 106, 97, 72, 28, 45, 122], and we give several example applications of static dominance breaking in recent papers. In the diameter constrained minimum spanning tree problem, de Una et al. [28] adopt a dominance rule from Noronha et al [101] and add dominance breaking constraints in the form of disequalities to exclude solutions with higher cost from the search space. Gange and Stuckey [45] use the concert hall scheduling problem as a benchmark, in which implication constraints are added such that shorter and more profitable concerts are preferred in solutions. Senthooran et. al [122] study a problem of training engineers for service delivery, where they add inequality constraints in the skill allocation subproblem to favor skill addition to an engineer with a subset of skills when compared to another engineer's skills. In general, dominance breaking constraints are usually of different forms for different problems, and they usually require sophisticated insights of the problem structure.

Static symmetry breaking methods have been studied extensively in the literature. *Variable symmetries* and *value symmetries* are two main categories, in which there exists a permutation of variables and values respectively such that the satisfaction of constraints are preserved. Crawford et al. [27] propose the LexLeader method to break variable symmetries, which is a general scheme that adds *symmetry breaking constraints* to preserve the lexicographically least solution of a CSP. Later, the global lexicographical ordering constraints are proposed with efficient consistency algorithms [16, 17, 39]. Variable symmetries often arise in matrix models, and DoubleLex [33] is a method to break symmetries in matrix models with a linear number of lexicographical ordering constraints, and it is observed with good performance in practice [66] with the theoretical guarantee on upper bound of remaining solutions in each symmetry class [62]. The classical LexLeader and DoubleLex methods are based on the lexicographical ordering schemes, and other method with different ordering schemes, such as the multiset ordering [41], the SnakeLex ordering [58], the Gray code ordering [99], and the reflex ordering [83], are also proposed for breaking variable symmetries. To break value symmetries in the graceful graph problems, Petrie and Smith [105] adapt the LexLeader method and add appropriate lexicographical ordering constraints. Later, global constraints [110, 128, 129, 78, 77, 45] are also proposed with efficient propagation algorithms to handle value symmetries.

3.4 Dynamic Dominance Breaking

Static dominance breaking may not always be effective in the BnB search, since dominated solutions may improve the objective bound and allow additional pruning of search space in BnB. In the literature [34, 47], there are empirical evidences showing that dominance and symmetry breaking constraints can have negative impact on the performance of the BnB search. Theoretical study by Ibaraki [63] also shows that the removal of dominated solutions in the BnB algorithm with depth first search is guaranteed to reduce the number of search nodes only when the dominating solutions have been visited. Therefore, dynamic dominance breaking methods modify the search procedure to exclude only dominated solutions that cannot enable additional search space pruning. Most dynamic dominance breaking methods are problem specific [22, 34, 46, 111]. For example, Focacci and Shaw [34] give a local search method for the travelling salesman problem, which prunes the current branch if it cannot be extended to a solution that is better than the current best solution.

There are few works on generic approaches for dynamic dominance breaking. Chu and Stuckey propose [24] *dominance jumping*, where the propagators of dominance breaking constraints only check the validity but never prune any values from the domain. When the BnB search reach a failure node, the search is restarted, and a temporary strategy is used to guide the search towards another part of the search space which contains better solutions potentially. In this way, the dominating solutions are visited first to obtain stronger objective bounds for pruning. Another generic approach is *automatic caching via constraint projection* [21], in which a caching description or a key is computed and stored for each visited search node. Conditions on keys are given to check whether all solutions in the current search node are dominated by solutions in a cached one. If the conditions are meet, the current search node will not be explored to avoid redundant search.

With regard to symmetry relations, there are various methods for breaking symmetries dynamically. The representative method is *symmetry breaking during search* [53, 49]. After exploring a search node, a symmetry breaking constraint is constructed and added for each symmetry to avoid exploring the search space that is symmetric to the visited search node. To improve the efficiency, Mears et al. [93] propose lightweight dynamic symmetry breaking that handles only symmetries that are common and can be compactly represented. Lee and Zhu [81, 84] identify the missing pruning opportunities in partial symmetry breaking during search and propose recursive symmetry breaking during search. Later, they also propose a global constraint to reduce the propagation overhead for a sequence of increasing nogoods [82]. Another typical dynamic symmetry breaking method is *symmetry breaking via dominance detection* [30, 108]. The method implements a specific dominance checker for each problem to check if the current search node is dominated by a visited node, and avoids exploring a dominated node. Gent et al. [50] later utilize computational group theory to implement generic dominance checker that can be used for any symmetry arising in a CSP.

Chapter 4

Automatic Generation of Dominance Breaking Nogoods

In this chapter, we give an automated method to identify and exploit dominance relations in a class of constraint optimization problems. 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 a COP.

Example 10. *Consider the COP in (1.1) and a pair of partial assignments $\theta = \{x_1 = v_1, x_2 = v_2\}$ and $\theta' = \{x_1 = v'_1, x_2 = v'_2\}$ 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*

$$\bar{\theta}[x_3] = \bar{\theta}'[x_3] \text{ and } \bar{\theta}[x_4] = \bar{\theta}'[x_4]. \tag{4.1}$$

We construct a CSP as follows:

$$\begin{aligned} 3v_1 + v_2 &\geq 3v'_1 + v'_2 \\ v_1 + 2v_2 &\leq v'_1 + 2v'_2 \\ v_1 \neq v'_1 &\vee v_2 \neq v'_2 \end{aligned} \tag{4.2}$$

We claim that if θ and θ' satisfy (4.2), then all assignments in $\mathcal{D}^X_{\theta'}$ can be removed without changing the optimal value of (1.1).

We show the claim 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, for all $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$, we have

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

The second step holds due to (4.1). In other words, if $3v_1 + v_2 \leq 3v'_1 + v'_2$, then any assignment $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$ will be mapped to $\bar{\theta}$ such that $f(\bar{\theta}) \leq f(\bar{\theta}')$. By similar reasoning, if $v_1 + 2v_2 \leq v'_1 + 2v'_2$, then

$$\bar{\theta}'[x_1] + 2\bar{\theta}'[x_2] + 3\bar{\theta}'[x_3] + 4\bar{\theta}'[x_4] \leq \bar{\theta}[x_1] + 2\bar{\theta}[x_2] + 3\bar{\theta}[x_3] + 4\bar{\theta}[x_4],$$

which means that $\bar{\theta}$ must have less weight than $\bar{\theta}'$, and $\bar{\theta}'$ is a solution implies that $\bar{\theta}$ is also a solution. In other words, the full assignment $\bar{\theta}'$ and its image $\bar{\theta}$ must satisfy either one of the three cases in Definition 1, and $\prec$ is a dominance relation.

Because $\prec$ is a dominance relation, a full assignment $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$ is dominated by some assignments $\sigma(\bar{\theta}') \in \mathcal{D}^X_\theta$ and can be pruned by Theorem 1. One such solution is $\theta = \{x_1 = 0, x_2 = 1\}$ and $\theta' = \{x_1 = 1, x_2 = 0\}$. To prune all full assignments in $\mathcal{D}^X_{\theta'}$, we can simply add a dominance breaking nogood $\neg\theta' \equiv (x_1 \neq 1 \vee x_2 \neq 0)$ to the original COP.

Example 10 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 4.1):

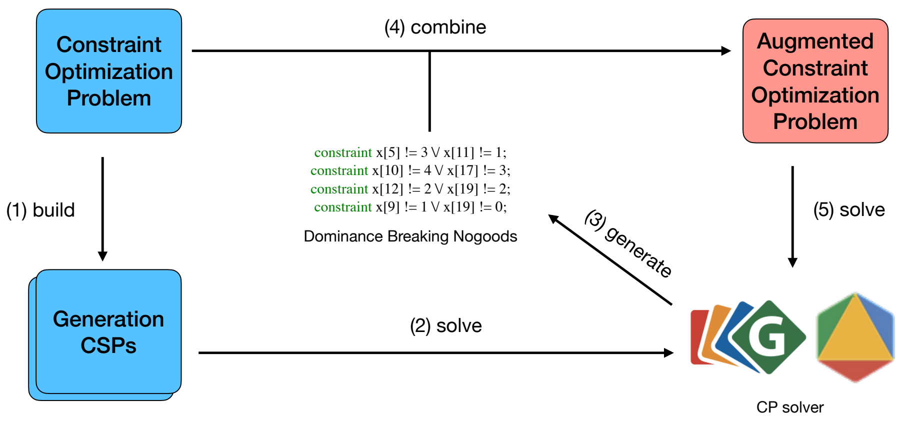
1. Analyze a target COP P and construct auxiliary generation CSPs.

2. Enumerate all solutions of the generation CSPs using a constraint solver.

3. Generate one nogood constraint for each solution of the CSPs.

4. Add all generated nogood constraints to the COP P.

5. Solve the COP with extra nogoods by a constraint solver.


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

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

Figure 4.1: *The workflow of automatic dominance breaking for COPs*

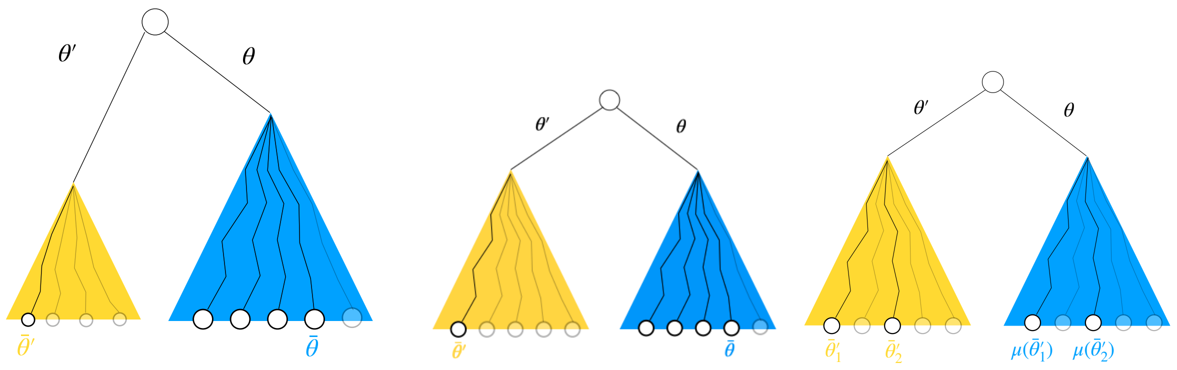
As shown in Example 10, constraints in generation CSPs are sufficient conditions to ensure that the derived nogoods $\neg\theta'$ is a valid dominance breaking constraints for P. As long as generation CSPs are constructed automatically, the workflow can be automated and coordinated by a simple program. In the following subsections, we formalize the reasoning process for a class of COPs and provide detailed theoretical explanations on how to obtain concrete constraints in generation CSPs.

4.1 Dominance Relations over Partial Assignments

Recall that dominance relations over full assignments can be generalized to search nodes in Definition 2. We can adapt the definition to define dominance relations over partial assignments in a straightforward manner.

Definition 3. *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 $\bar{\theta} \prec \bar{\theta}'$ for a dominance relation $\prec$ over $\mathcal{D}^X$, then we say θ dominates θ' ($\theta \prec \theta'$) with respect to P.*

There are several reasons to restrict to sets of full assignments extending from partial



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

Figure 4.2: *Restriction for checking a subset of pairs of partial assignments*

assignments. First, each search node in a search tree is associated with a partial assignment, and the set of full assignments of a search node in Definition 2 is equivalent to the set of full assignments extending from the associated partial assignment. Second, a partial assignment with a fixed scope can be easily represented using an array of integer variables, each corresponding to the value assigned to a variable in the partial assignment as shown in Example 10. Third, removing all full assignments in $\mathcal{D}^X_{\theta'}$ simply requires adding a nogood constraint $\neg\theta'$ to the target COP, and it is easy to handle in a constraint programming solver. What is more, nogood constraints are elementary in constraint programming, and they can be combined to form arbitrary kinds of constraints. The following theorem is a direct consequence of Theorem 1.

Theorem 2. *Let $P = (X, D, C, f)$ be a COP. If two partial assignments 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 3 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 4.2a). Our first restriction is to focus on pairs of partial assignments over the same scope, which will reduced the number of checking substantially (Fig 4.2b). Still, checking $\theta \prec \theta'$ needs to find one full assignment in $\mathcal{D}^X_\theta$ for each full assignments $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$, and the straightforward implementation requires nested for-loops. Instead of exhaustive enumeration, we further define the *mutation mapping* $\mu^{\theta' \to \theta}$ for θ and θ' and restrict our attention to check whether a full assignment $\bar{\theta}'$ is dominated by its image $\mu^{\theta' \to \theta}(\bar{\theta}')$ (Fig 4.2c).

Definition 4. *Let $\theta, \theta' \in \mathcal{D}^S$ be two assignments in a COP P over the same scope S. The mutation mapping $\mu^{\theta' \to \theta} : \mathcal{D}^X_{\theta'} \mapsto \mathcal{D}^X_\theta$ 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 $\mu^{\theta' \to \theta}(\bar{\theta}')$ "mutates" the value assigned by θ' in $\bar{\theta}'$ to that assigned by θ. For convenience of presentation, we let $\bar{\theta} = \mu^{\theta' \to \theta}(\bar{\theta}')$ when it is clear from the context.

Example 11. *Consider the two assignments θ and θ' in Example 10. We list all full assignments $\bar{\theta}' \in \mathcal{D}^X_\theta$ and their images under $\mu^{\theta' \to \theta}$ 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.

With the mutation mapping, the following theorem gives a sufficient condition for $\theta \prec \theta'$ such that each full assignment in $\mathcal{D}^X_{\theta'}$ is transformed into a better one in $\mathcal{D}^X_\theta$.

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 $\mu^{\theta' \to \theta} : \mathcal{D}^X_{\theta'} \mapsto \mathcal{D}^X_\theta$ satisfies:*

- *irreflexivity: the transitive closure of $\mu^{\theta' \to \theta}$ is irreflexive,*

- *betterment: $\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}$, $f(\mu^{\theta' \to \theta}(\bar{\theta}')) \leq f(\bar{\theta}')$, and*

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

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

Proof. The proof idea is inspired by Chu and Stuckey [23, 25]. For the ease of presentation, let σ denote the mutation mapping $\mu^{\theta' \to \theta}$ for θ and θ'. We construct a relation $\prec$ by taking the transitive closure of σ and show that $\prec$ is a dominance relation in P. Note that $\prec$ is both transitive and irreflexive by construction. What remains is to show that for all $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$, if $\sigma(\bar{\theta}') \prec \bar{\theta}'$, then $\sigma(\bar{\theta}')$ and $\bar{\theta}'$ will satisfy either one of the three cases in Definition 1:

- Suppose $\bar{\theta}'$ is a solution of P. If $\bar{\theta} \prec \bar{\theta}'$, then there must be a sequence $\bar{\theta}', \sigma(\bar{\theta}'), \ldots, \sigma^t(\bar{\theta}')$ such that $t \in \mathbb{N}$ and $\sigma^t(\bar{\theta}) = \bar{\theta}$ since $\prec$ is a transitive closure of σ. By the implied satisfaction, $\bar{\theta}'$ being a solution implies that all elements in the sequence are solutions of P. Furthermore, the betterment implies that $f(\bar{\theta}') \geq f(\sigma(\bar{\theta}')) \geq f(\sigma^2(\bar{\theta}')) \geq \cdots \geq f(\sigma^t(\theta)) = f(\bar{\theta})$. Thus, $\bar{\theta}$ and $\bar{\theta}'$ fulfill the second case in Definition 1.

- Suppose $\bar{\theta}'$ is not a solution of P. If $\bar{\theta} \in sol(P)$, then $\bar{\theta}$ and $\bar{\theta}'$ fulfill the first case in Definition 1. Otherwise, $\bar{\theta} \notin sol(P)$. Again, we construct a sequence $\bar{\theta}', \sigma(\bar{\theta}'), \sigma^2(\bar{\theta}'), \ldots, \sigma^t(\bar{\theta}')$ such that $t \in \mathbb{N}$ and $\sigma^t(\bar{\theta}) = \bar{\theta}'$. Since $\bar{\theta}$ is a non-solution, all elements in the sequence are non-solutions by contrapositive of the implied satisfaction. By the betterment again, $f(\bar{\theta}') \geq f(\sigma(\bar{\theta}')) \geq f(\sigma^2(\bar{\theta}')) \geq \cdots \geq f(\sigma^t(\theta)) = f(\bar{\theta})$. Thus, $\bar{\theta}$ and $\bar{\theta}'$ fulfill the third 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 key of Theorem 3 is to define an appropriate mapping σ. The advantage of using a mutation mapping is that we can derive sufficient conditions to fulfill the irreflexivity, betterment and implied satisfaction conditions. As we will show in Sections 4.2 and 4.3, these sufficient conditions are constraints over $\theta, \theta' \in \mathcal{D}^S$ for a class of COPs, and values assigned to variables in $X \setminus S$ are irrelevant. Therefore, searching for pairs (θ, θ') of partial assignments and nogood generation can be modelled as constraint satisfaction.

In the rest of this section, we give sufficient conditions for the irreflexivity, betterment and implied satisfaction in Theorem 3 when the mapping is a mutation mapping $\mu^{\theta' \to \theta}$ for a pair (θ, θ') of partial assignments. The sufficient condition for the irreflexivity condition turns out to be simple.

Theorem 4. *Let $\theta, \theta' \in \mathcal{D}^S$ be two assignments in a COP $P = (X, D, C, f)$ over the same scope $S \subseteq X$ and $\mu^{\theta' \to \theta} : \mathcal{D}^X_{\theta'} \mapsto \mathcal{D}^X_{\theta}$ be the corresponding mutation mapping. If $\theta \neq \theta'$, then the mutation mapping $\mu^{\theta' \to \theta}$ is irreflexive, and its transitive closure is also irreflexive.*

Proof. Since $\theta \neq \theta'$, we have $\mathcal{D}^X_{\theta} \cap \mathcal{D}^X_{\theta'} = \emptyset$. For all full assignments $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$, the image $\bar{\theta} = \mu^{\theta' \to \theta}(\bar{\theta}')$ must not in $\mathcal{D}^X_{\theta'}$. In other words, if $(\bar{\theta}', \bar{\theta}) \in \mu^{\theta' \to \theta}$, then $(\bar{\theta}, *) \notin \mu^{\theta' \to \theta}$ where $*$ is an arbitrary full assignment. Thus, the transitive closure of $\mu^{\theta' \to \theta}$ is also irreflexive. $\square$

The sufficient conditions for betterment and implied satisfaction are based on the objective and constraints of the target COP. We say that objectives and constraints are *efficiently checkable* if the sufficient conditions can be formulated as constraints over θ and θ' of certain forms, and generation CSPs can be constructed mechanically by analyzing the problem and adding constraints in a pattern matching approach. In the following, we give sufficient conditions for different classes of objectives and constraints.

4.2 Betterment for Efficiently Checkable Objectives

Now we give sufficient conditions that imply betterment for $\mu^{\theta' \to \theta}$, namely $\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}$, $f(\bar{\theta} = \mu^{\theta' \to \theta}(\bar{\theta}')) \leq f(\bar{\theta}')$. The key idea is to define a *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 by $f{\downarrow_S}(\theta)$ and $f{\downarrow_S}(\theta')$. In the following, we consider two types of efficiently checkable objective functions: separable functions and supermodular/submodular functions.

4.2.1 Separable Objective Functions

A function f is *separable* if it can be written as a sum 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 5. *Let $\theta, \theta' \in \mathcal{D}^S$ be two assignments in a COP $P = (X, D, C, f)$ over the same scope $S \subseteq X$ and $\mu^{\theta' \to \theta} : \mathcal{D}^X_{\theta'} \mapsto \mathcal{D}^X_{\theta}$ be the corresponding mutation mapping. Suppose f is a separable function. If we have*

$$f{\downarrow_S}(\theta) \leq f{\downarrow_S}(\theta'), \tag{4.3}$$

then $f(\mu^{\theta' \to \theta}(\bar{\theta}')) \leq f(\bar{\theta}')$ for all full assignments $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$.

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

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

By Definition 4, for all $x_i \in X \setminus S$, $\bar{\theta}[x_i] = \bar{\theta}'[x_i]$ and therefore $f_i(\bar{\theta}'[x_i]) = f_i(\bar{\theta}[x_i])$. Thus, $f{\downarrow_S}(\theta) \leq f{\downarrow_S}(\theta')$ implies that $f(\bar{\theta}) \leq f(\bar{\theta}')$. □

Therefore, when the objective of a COP is a separable function, (4.3) is a sufficient condition for betterment and should be added to generation CSPs.

Example 12. *A typical example of separable functions is the linear function $f(\bar{\theta}) = \sum_{x_i \in S} w_i \bar{\theta}[x_i]$, where $w_i \in \mathbb{R}$ is the weight for variable x_i. Another example arises from the classical assignment problem. Given a set A of agents and a set T of tasks, the problem asks for a mapping $m : T \mapsto A$ such that the cost function $\sum_{i \in T} cost(i, m(i))$ is minimized, where $cost(i, m(i))$ is the cost of assigning task i to agent $m(i)$. We can define one variable x_i with domain $D(x_i) = A$ for each task $i \in T$. The cost function can be viewed as a separable function $f(\bar{\theta}) = f_1(\bar{\theta}[x_1]) + \cdots + f_n(\bar{\theta}[x_n])$ where $f_i(\bar{\theta}[x_i]) = cost(i, \bar{\theta}[x_i])$.*

4.2.2 Supermodular and Submodular Objective Functions

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

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

for every $V, V' \subseteq U$ where $V \subseteq V'$ and $T \subseteq U \setminus V'$. 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 $V(\theta) = \{i \mid \theta[x_i] = 1\}$. We say that the objective function f of a binary COP P *is equivalent to a supermodular function* g if $f(\bar{\theta}) = g(V(\bar{\theta}))$ for every $\bar{\theta} \in \mathcal{D}^X$, and similarly we define $f{\downarrow_S}(\theta) = g(V(\theta))$.

Theorem 6. *Let $\theta, \theta' \in \mathcal{D}^S$ be two assignments in a COP $P = (X, D, C, f)$ over the same scope $S \subseteq X$ and $\mu^{\theta' \to \theta} : \mathcal{D}^X_{\theta'} \mapsto \mathcal{D}^X_{\theta}$ be the corresponding mutation mapping. Suppose f is a function which is equivalent to a supermodular function $g : 2^V \mapsto \mathbb{R}$ in a binary COP. If we have*

$$f{\downarrow_S}(\theta) \leq f{\downarrow_S}(\theta') \wedge V(\theta) \subseteq V(\theta'), \tag{4.4}$$

then $f(\mu^{\theta' \to \theta}(\bar{\theta}')) \leq f(\bar{\theta}')$ for all full assignments $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$.

Proof. By definition of a supermodular function, we have

$$g(V(\theta) \cup T) - g(V(\theta)) \leq g(V(\theta') \cup T) - g(V(\theta')) \tag{4.5}$$

for $V(\theta) \subseteq V(\theta') \subseteq V$ and any set $T \subseteq U \setminus V(\theta')$. Let $T = \{i \mid \bar{\theta}'[x_i] = 1 \wedge x_i \in X \setminus S\}$ for a full assignment $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$. Since f is equivalent to g, we have $f(\bar{\theta}') = g(V(\bar{\theta}')) = g(V(\theta') \cup T)$. By Definition 4, $\bar{\theta}[x_i] = \bar{\theta}'[x_i]$ for all variables $x_i \in X \setminus S$, where $\bar{\theta} = \mu^{\theta' \to \theta}(\bar{\theta}')$. We also have $f(\bar{\theta}) = g(V(\bar{\theta})) = g(V(\theta) \cup T)$. Therefore, equation (4.5) becomes

$$\begin{aligned} & f(\bar{\theta}) \leq f(\bar{\theta}') - g(V(\theta')) + g(V(\theta)) \\ \Leftrightarrow & f(\bar{\theta}) \leq f(\bar{\theta}') - f{\downarrow_S}(\theta') + f{\downarrow_S}(\theta) \end{aligned}$$

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.

Theorem 7. *Let $\theta, \theta' \in \mathcal{D}^S$ be two assignments in a COP $P = (X, D, C, f)$ over the same scope $S \subseteq X$ and $\mu^{\theta' \to \theta} : \mathcal{D}^X_{\theta'} \mapsto \mathcal{D}^X_{\theta}$ be the corresponding mutation mapping. Suppose f is a function which is equivalent to a submodular function $g : 2^V \mapsto \mathbb{R}$ in a binary COP. If we have*

$$f{\downarrow_S}(\theta) \geq f{\downarrow_S}(\theta') \wedge V(\theta) \subseteq V(\theta'), \tag{4.6}$$

then $f(\bar{\theta}) \geq f(\bar{\theta}')$ for all full assignments $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$.

The proof of Theorem 7 is similar to that of Theorem 6. When the objective in a binary COP is to minimize a supermodular function or to maximize a submodular function, and (4.4) and (4.6) are sufficient conditions for betterment respectively, and should be added to generation CSPs. We note that Theorems 3 and 5 can also be easily adapted for maximization.

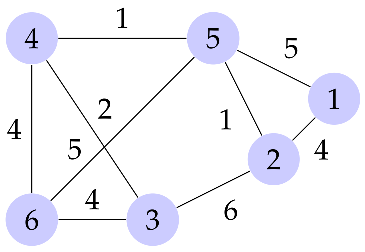


Figure 4.3: *An example weighted undirected graph*

Example 13. *A linear function is an example submodular function. Another typical example is the cut function in a maximum cut problem on a graph $G = (V, E)$. The problem is to find a partition of V to minimize $\sum_{(i,j) \in E}(x_i \otimes x_j)$ where $x_i = 1$ means that a node $i \in V$ is in the first partition, and $(x_i \otimes x_j)$ is 1 when $x_i \neq x_j$. Consider the weighted undirected graph in Figure 4.3. Suppose $\theta = \{x_4 = 0, x_6 = 1\}$ and $\theta' = \{x_4 = 1, x_6 = 1\}$ are assignments over the scope $S = \{x_4, x_6\}$. The associated sets are $V(\theta) = \{6\}$ and $V(\theta') = \{4, 6\}$, and the values of projection function is $f{\downarrow_S}(\theta) = g(V(\theta)) = 4 + 4 + 5 = 13$ and $f{\downarrow_S}(\theta') = g(V(\theta')) = 1 + 2 + 4 + 5 = 12$. The conditions in Theorem 7 are satisfied, and we can verify that $f(\mu^{\theta' \to \theta}(\bar{\theta}')) \geq f(\bar{\theta}')$ for all*

$\bar{\theta}' \in \mathcal{D}^X_{\theta'}$. For example, if $\bar{\theta}' = \{x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 1, x_5 = 0, x_6 = 1\}$, then $\mu^{\theta' \to \theta}(\bar{\theta}') = \{x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0, x_5 = 0, x_6 = 1\}$, and $f(\bar{\theta}) = 4+5+2+1+4 = 16 \geq f(\bar{\theta}') = 1+2+1+4 = 8$.

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 if and only if $\bar{\theta}$ satisfies all constraints $c \in C$. If for all $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$ and all $c \in C$, $\bar{\theta}'$ satisfies c implies that $\bar{\theta}$ satisfies c, then the implied satisfaction also hold. This suggests considering each constraint $c \in C$ separately.

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 $c\theta$ implies that $c\theta'$ is a sufficient condition to prove implied satisfaction.

Proposition 2. *Let $\theta, \theta' \in \mathcal{D}^S$ be two assignments in a COP $P = (X, D, C, f)$ over the same scope $S \subseteq X$ and $\mu^{\theta' \to \theta} : \mathcal{D}^X_{\theta'} \mapsto \mathcal{D}^X_{\theta}$ be the corresponding mutation mapping. 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 $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$, where $\bar{\theta} = \mu^{\theta' \to \theta}(\bar{\theta}')$.*

Proof. Let $S' = var(c) \setminus S$. By Definition 4, $\bar{\theta}'[x] = \bar{\theta}[x]$ for $x \notin S$, and so $\bar{\theta}'\downarrow_{S'} = \bar{\theta}\downarrow_{S'}$. Since $c\theta' \Rightarrow c\theta$, we have $c\theta' \subseteq c\theta$, 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 implies that $\mu^{\theta' \to \theta}(\bar{\theta}')$ satisfies c. □

In the following sections, we consider sufficient conditions for $c\theta' \Rightarrow c\theta$. The sufficient conditions for all constraints should be added to generation CSPs, whose conjunction implies the implied satisfaction in Theorem 3.

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 $x \in D(x)$. The sufficient condition for $c\theta' \Rightarrow c\theta$ is straightforward.

Theorem 8. *Let $\theta, \theta' \in \mathcal{D}^S$ be two assignments in a COP $P = (X, D, C, f)$ over the same scope $S \subseteq X$. Suppose $c \in C$ is 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, and $c\theta' \Rightarrow c\theta$ always hold. □

Note that if $\theta'[x] \notin D(x)$, then all full assignments $\bar{\theta}'$ in $\mathcal{D}^X_{\theta'}$ are trivially non-solutions of P. An extra nogood constraint $\neg\theta'$ is redundant in a propagation solver. Therefore, we would add

$$\theta[x] \in D(x) \wedge \theta'[x] \in D(x) \tag{4.7}$$

for all variables $x \in S$ into generation CSPs.

4.3.2 Linear Inequality Constraints

A linear inequality constraint is of the form $\sum w_i x_i \leq b$ where $w_i, b \in \mathbb{R}$. The sufficient condition for $c\theta' \Rightarrow c\theta$ is stated as follows.

Theorem 9. *Let $\theta, \theta' \in \mathcal{D}^S$ be two assignments in a COP $P = (X, D, C, f)$ over the same scope $S \subseteq X$. Suppose $c \in C$ is a linear inequality constraint of the form $\sum w_i x_i \leq b$, and $S' = S \cap var(c)$. If we have*

$$\sum_{x_i \in S'} w_i \theta[x_i] \leq \sum_{x_i \in S'} w_i \theta'[x_i], \tag{4.8}$$

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

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

□

Note that if $\sum_{x_i \in S'} w_i \theta'[x_i] > b$, then any full assignment $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$ must not satisfy the linear inequality constraint $c \equiv (\sum w_i x_i \leq b)$ and is trivially a nonsolution of P. Therefore, we also add a constraint $\sum_{x_i \in S'} w_i \theta'[x_i] \leq b$ in addition to (4.8) into generation CSPs.

4.3.3 Boolean Disjunction Constraints

The *Boolean disjunction constraint* $\vee_{x_i \in B} x_i$ requires at least one Boolean variable in the set B takes the true value. The following result gives a sufficient condition for $c\theta' \Rightarrow c\theta$ when c is a Boolean disjunction constraint.

Theorem 10. *Let $\theta, \theta' \in \mathcal{D}^S$ be two assignments in a COP $P = (X, D, C, f)$ over the same scope $S \subseteq X$. Suppose $c \in C$ is a Boolean disjunction constraint $(\vee_{x_i \in B} x_i)$ and $S' = S \cap B$. If we have*

$$\vee_{x_i \in S'} \theta'[x_i] \Rightarrow \vee_{x_i \in S'} \theta[x_i], \tag{4.9}$$

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, and it 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 $c\theta \equiv e \vee (\vee_{x_i \in S \setminus S'} x_i)$. 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 11. *Let $\theta, \theta' \in \mathcal{D}^S$ be two assignments in a COP $P = (X, D, C, f)$ over the same scope $S \subseteq X$. Suppose c is a Boolean disjunction constraint $\vee_{x_i \in var(c)} (x_i \circ_i b_i)$ where $b_i \in \mathbb{Z}$ and $\circ_i \in \{=, \neq, \geq, \leq, <, >\}$ is a binary comparison operator. If we have*

$$\vee_{x_i \in S'} (\theta'[x_i] \circ_i b_i) \Rightarrow \vee_{x_i \in S'} (\theta[x_i] \circ_i b_i), \tag{4.10}$$

then $c\theta' \Rightarrow c\theta$, where $S' = S \cap var(c)$.

The proof of Theorem 11 is similar to that of Theorem 10.

4.3.4 Counting Constraints

Counting constraints restrict the number of occurrences of some values in the set V within a given scope T of variables. The Global Constraint Catalog [8] has the keyword "Counting

Constraints" under which there are 39 different global constraints, among which *alldifferent*, *among*, and *global cardinality* constraints 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, and then consider example counting constraints that are composed of several such basic constraints. The *atleast* and *atmost* constraints require that the number of variables in T that take values in V is at least and at most k respectively. The following theorems give the sufficient conditions for $c\theta' \Rightarrow c\theta$.

Theorem 12. *Let $\theta, \theta' \in \mathcal{D}^S$ be two assignments in a COP $P = (X, D, C, f)$ over the same scope $S \subseteq X$. Suppose $c \in C$ is a constraint of the form $atleast(T, V, k)$. If we have*

$$|\{x \mid \theta[x] \in V \land x \in S'\}| \leq |\{x \mid \theta'[x] \in V \land x \in S'\}|, \tag{4.11}$$

then $c\theta' \Rightarrow c\theta$, where $S' = T \cap S$.

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

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

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

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

□

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

Theorem 13. *Let $\theta, \theta' \in \mathcal{D}^S$ be two assignments in a COP $P = (X, D, C, f)$ over the same scope $S \subseteq X$. Suppose $c \in C$ is a constraint of the form $atmost(T, V, k)$ and $S' = T \cap S$. If we have*

$$|\{x \mid \theta[x] \in V \land x \in S'\}| \geq |\{x \mid \theta'[x] \in V \land x \in S'\}|, \tag{4.12}$$

then $c\theta' \Rightarrow c\theta$*, where* $S' = T \cap S$*..*

The proof is also similar to that of Theorem 12. Similar to linear inequality constraints, we can avoid generate redundant nogood constraint $\neg\theta'$ for a partial assignment θ' violating $atmost(T, V, k)$. Therefore, we can add a constraint $|\{x \mid \theta'[x] \in V \wedge x \in S'\}| \leq k$ in addition to (13) into generation CSPs.

By Theorems 12 and 13, we can derive the sufficient conditions for $c\theta' \Rightarrow c\theta$ where c is a counting constraint. The first famous example of counting constraints is the *alldifferent* constraint [115], 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. Further, if $v \in V$ only appears in the domain of one variable, $\texttt{atmost}(T, \{v\}, 1)$ is trivially true. Therefore, the sufficient condition for $c\theta' \Rightarrow c\theta$ is stated as follows.

Corollary 1. *Let* $\theta, \theta' \in \mathcal{D}^S$ *be two assignments in a COP* $P = (X, D, C, f)$ *over the same scope* $S \subseteq X$*. Suppose* $c \in C$ *is a constraint of the form alldifferent*(T) *where* $T \subseteq X$*. If we have*

$$\{\theta[x] \mid x \in T \cap S \wedge \theta[x] \in V'\} \subseteq \{\theta'[x] \mid x \in T \cap S \wedge \theta'[x] \in V'\}, \tag{4.13}$$

where $V' = \cup_{x_1, x_2 \in T, x_1 \neq x_2}(D(x_1) \cap D(x_2))$*, then* $c\theta' \Rightarrow c\theta$*.*

The proof follows directly from the Theorem 13 and the fact that *alldifferent*$(T) \equiv \wedge_{v \in V'}\texttt{atmost}(T, \{v\}, 1)$.

The *alldifferent_except_0* constraint [8] 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.

Corollary 2. *Let* $\theta, \theta' \in \mathcal{D}^S$ *be two assignments in a COP* $P = (X, D, C, f)$ *over the same scope* $S \subseteq X$*. Suppose* $c \in C$ *is the constraint alldifferent_except_0*(T) *where* $T \subseteq X$*. If we have*

$$\{\theta[x] \mid x \in T \cap S \wedge \theta[x] \in V'\} \subseteq \{\theta'[x] \mid x \in T \cap S \wedge \theta'[x] \in V'\}, \tag{4.14}$$

where $V' = \cup_{x_1,x_2 \in T, x_1 \neq x_2}((D(x_1) \cap D(x_2)) \setminus \{0\})$, *then* $c\theta' \Rightarrow c\theta$.

The proof idea is similar to that of Corollary 1.

Another example is the *among* constraint [9, 11], 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 3. *Let* $\theta, \theta' \in \mathcal{D}^S$ *be two assignments in a COP* $P = (X, D, C, f)$ *over the same scope* $S \subseteq X$. *Suppose* $c \in C$ *is 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 we have*

$$|\{x \mid x \in T \wedge \theta[x] \in V\}| = |\{x \mid x \in T \wedge \theta'[x] \in V\}|, \tag{4.15}$$

then $c\theta' \Rightarrow c\theta$.

The global cardinality constraint [104] 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, and 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 4. *Let* $\theta, \theta' \in \mathcal{D}^S$ *be two assignments in a COP* $P = (X, D, C, f)$ *over the same scope* $S \subseteq X$. *Suppose* c *is 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) *such that* $v_j, l_j, u_j \in \mathbb{Z}$. *If we have*

$$|\{x \mid x \in S \cap T \wedge \theta[x] = v_j\}| = |\{x \mid x \in S \cap T \wedge \theta'[x] = v_j\}| \tag{4.16}$$

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 [35, 67] 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 $|V| = n$, and we have one variable x_i for each node $i \in V$ representing the successor node after visiting node i. The constraint $\mathit{circuit}(x_1, \ldots, x_n)$ 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 $c\theta' \Rightarrow c\theta$ require the introduction of additional variables for the path represented by θ and θ'.

Theorem 14. *Let c be a constraint $\mathit{circuit}(x_1, \ldots, x_n)$ 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) $\mathit{alldifferent}(y_1, \ldots, y_{|S'|+1})$ and $\mathit{alldifferent}(y'_1, \ldots, y'_{|S'|+1})$,

(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, $c\theta' \Leftrightarrow c\theta$. $\square$

4.4 Compatibility between 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, and all derived 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 14. *Consider a variant of the COP in (1.1) where the first two items has two units of weight and profit. Following the same procedure as Example 10, 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:*

$$\begin{aligned} 2v_1 + 2v_2 &\geq 2v'_1 + 2v'_2 \\ 2v_1 + 2v_2 &\leq 2v'_1 + 2v'_2 \\ v_1 \neq v'_1 &\vee v_2 \neq v'_2 \end{aligned} \tag{4.17}$$

where $v_1, v_2, v'_1, v'_2 \in \{0, 1\}$ are unknown integers. Solving (4.17) 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, and 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 Lexicographical 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 $\forall j' < j, a_{j'} = b_{j'}$. Note that an assignment $\theta = \{x_{i_j} = v_{i_j} \mid x_{i_j} \in S\}$ over the scope S can be treated as a tuple $(v_{i_1}, v_{i_2}, \ldots, v_{i_k})$ of values, where $|S| = k$ and $i_1 < i_2 < \cdots < i_k$. We are interested in preserving the *lexicographically smallest optimal solution* which is an optimal solution of P and is lexicographically smallest among all solutions with the optimal objective value.

Theorem 15. *Let $\theta, \theta' \in \mathcal{D}^S$ be two assignments in a COP $P = (X, D, C, f)$ over the same scope $S \subseteq X$. If $\theta <_{lex} \theta'$, and (θ, θ') satisfies the betterment and implied satisfaction conditions, 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 $\bar{\theta} = \mu^{\theta' \to \theta}(\bar{\theta}') \in \mathcal{D}^X_{\theta}$. By Definition 4, $\bar{\theta}[x] = \bar{\theta}'[x]$ 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 $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$, then there must be another optimal solution $\bar{\theta} \in \mathcal{D}^X_{\theta}$ which is lexicographically smaller than $\bar{\theta}'$. By contraposition, the lexicographically smallest optimal solution does not belong to $\mathcal{D}^X_{\theta'}$ and always satisfy the nogood constraint $\neg\theta'$. $\square$

Theorem 15 implies that if we enforce $\theta <_{lex} \theta'$ in generation CSPs, generated nogoods will not remove the lexicographically smallest optimal solution. This ensures that all generated nogoods $\neg\theta'$ derived from the solutions of generation CSPs are compatible, and adding all nogoods to the COP preserves the optimal value of P. We note the idea of using the lexicographical ordering as the tiebreaker also appears in previous works [23, 31].

4.4.2 Generalized Lexicographical Ordering

The lexicographical ordering constraint 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 15. *Consider the COP in (1.1) again. If we want to find a pair of partial assignments over a scope $S = \{x_1, x_2\}$, then we can construct a generation CSP by Theorems 4, 5, 9 and 15 as follows:*

$$\begin{aligned}
&\textit{irreflexivity:} && \theta[x_1] \neq \theta'[x_1] \vee \theta[x_2] \neq \theta'[x_2] \\
&\textit{betterment:} && -(3\theta[x_1] + \theta[x_2]) \leq -(3\theta'[x_1] + \theta'[x_2]) \\
&\textit{implied satisfaction:} && \theta[x_1] + 2\theta[x_2] \leq \theta'[x_1] + 2\theta'[x_2] \\
&\textit{compatibility:} && (\theta[x_1], \theta[x_2]) <_{lex} (\theta'[x_1], \theta'[x_2])
\end{aligned} \tag{4.18}$$

The pair $\theta_1 = \{x_2 = 0, x_3 = 1\}$ and $\theta'_1 = \{x_2 = 1, x_3 = 0\}$ satisfy the conditions, and $\neg\theta'_1$ is a valid nogood for the COP P.

Similarly, we can construct a generation CSP for a scope $S = \{x_3, x_4\}$:

$$\begin{aligned}
&\textit{irreflexivity:} && \theta[x_3] \neq \theta'[x_3] \vee \theta[x_4] \neq \theta'[x_4] \\
&\textit{betterment:} && -(6\theta[x_3] + 4\theta[x_4]) \leq -(6\theta'[x_3] + 4\theta'[x_4]) \\
&\textit{implied satisfaction:} && 3\theta[x_3] + 4\theta[x_j] \leq 3\theta'[x_3] + 4\theta'[x_4] \\
&\textit{compatibility:} && (\theta[x_3], \theta[x_4]) <_{lex} (\theta'[x_3], \theta'[x_4])
\end{aligned} \tag{4.19}$$

The CSP (4.19) is unsatisfiable, but we can observe that the pair $\theta_2 = \{x_3 = 1, x_4 = 0\}$ and $\theta'_2 = \{x_3 = 0, x_4 = 1\}$ also satisfy conditions in Theorem 3, and the nogood constraint $\neg\theta'_2$ is compatible with $\neg\theta'_1$. We can add both constraints to P while preserving the optimal solution $\bar{\theta}_{opt} = \{x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0\}$ with the objective value 9.

In order to find more dominance breaking nogoods by solving the generation CSPs, we give a more relaxed sufficient condition for the compatibility of generated nogoods based on the *generalized lexicographical order*. Instead of enforcing $\theta <_{lex} \theta'$, the generalized lexicographical ordering utilizes sensitive functions over $\mathcal{D}^X$. A function h defined over $\mathcal{D}^X$ is *sensitive* if for any $S \subseteq X$, there is a projection function $h{\downarrow_S}$ defined over $\mathcal{D}^S$ such that $h{\downarrow_S}(\theta) < h{\downarrow_S}(\theta')$ implies that $\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, h(\mu^{\theta' \to \theta}(\bar{\theta}')) < h(\bar{\theta}')$.

Theorem 16. *Let $\theta, \theta' \in \mathcal{D}^S$ be two assignments in a COP $P = (X, D, C, f)$ over the same scope $S \subseteq X$. If we have*

$$(h_1{\downarrow_S}(\theta), \ldots, h_m{\downarrow_S}(\theta)) <_{lex} (h_1{\downarrow_S}(\theta'), \ldots, h_m{\downarrow_S}(\theta')),$$

where $h_1, \ldots, h_m$ are sensitive functions, and the pair (θ, θ') satisfies betterment and implied satisfac-

tion, then there is at least one optimal solution satisfying $\neg\theta'$.

Proof. Without loss of generality, assume that $m = 1$. Let $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$ be an optimal solution of P and $\bar{\theta} = \mu^{\theta' \to \theta}(\bar{\theta}') \in \mathcal{D}^X_{\theta}$. Since the pair (θ, θ') satisfies betterment and implied satisfaction, $\bar{\theta}$ must also be an optimal solution. Also, h_1 is a sensitive function, and $h_1{\downarrow_S}(\theta) < h_1{\downarrow_S}(\theta')$ implies that $h_1(\bar{\theta}) < h_1(\bar{\theta}')$. 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 either $h_1(\bar{\theta}) < h_1(\bar{\theta}')$ or $\bar{\theta} <_{lex} \bar{\theta}'$. By contraposition, the lexicographically smallest optimal solution with the smallest value of h_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 $m > 1$. $\square$

Note that we can use arbitrary sensitive functions in Theorem 16 as long as their projection functions are well-defined. The generalized lexicographical ordering degenerates to the lexicographical ordering if we use *element functions* $e_1, \ldots, e_n$ where $e_i(\bar{\theta}) = \bar{\theta}[x_i]$, and the projection function over a scope S 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 16, 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 15, however, the lexicographical ordering 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 3. *Let $\theta, \theta' \in \mathcal{D}^S$ be two assignments in a COP $P = (X, D, C, f)$ over the same scope $S \subseteq X$. Suppose f is a separable function such that $f(\bar{\theta}) = \sum f_i(\bar{\theta}[x_i])$ for all $\bar{\theta} \in \mathcal{D}^X$. The separable function f is sensitive if we define the projection function of f be $f{\downarrow_S}(\theta) = \sum_{x_i \in S} f_i(\theta[x_i])$.*

Proposition 4. *Let $\theta, \theta' \in \mathcal{D}^S$ be two assignments in a COP $P = (X, D, C, f)$ over the same scope $S \subseteq X$. Suppose f is a function that is equivalent to a supermodular function g. The supermodular function f is sensitive if we define the projection function of f be $f{\downarrow_S}(\theta) = g(V(\theta))$ where $V(\theta) = \{i \mid \theta[x_i] = 1\}$.*

Example 16. *Consider the generation CSP in (4.19) again. If we replace the sufficient condition for compatibility by:*

$$(-(6\theta[x_3] + 4\theta[x_4]), \theta[x_3], \theta[x_4]) <_{lex} (-(6\theta'[x_3] + 4\theta'[x_4]), \theta'[x_3], \theta'[x_4]),$$

then solving the generation CSP will result in the desired pair of partial assignments and a nogood constraint $\neg\theta' \equiv (x_3 \neq 0 \vee x_4 \neq 1)$.

We can also define sensitive functions arising from linear inequality constraints and counting constraints. In the following propositions, we assume that $\theta, \theta' \in \mathcal{D}^S$ are two assignments in a COP $P = (X, D, C, f)$ over the same scope $S \subseteq X$.

Proposition 5. *Suppose c is a linear inequality constraint ($\sum w_i x_i \leq b$). If we define $h(\bar{\theta}) = \sum w_i \bar{\theta}[x_i]$ for all full assignment $\bar{\theta} \in \mathcal{D}^X$, and the projection function $h{\downarrow_S}(\theta) = \sum_{x_i \in S} w_i \theta[x_i]$, then h is sensitive.*

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

Proposition 7. *Suppose c is a constraint $atleast(T, V, k)$. If we define $h(\bar{\theta}) = -|\{x \mid \bar{\theta}[x] \in V \wedge x \in T\}|$ for a full assignment $\bar{\theta} \in \mathcal{D}^X$, and the projection function $h{\downarrow_T}(\theta) = -|\{x \mid \theta[x] \in V \wedge x \in T \cap S\}|$, then h 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 16 using the objective function, followed by sensitive functions arising from the linear inequality and counting constraints, and finally the element functions for tie-breaking.

4.5 Modeling Nogood Generation as Constraint Satisfaction

Sections 4.1 to 4.4 state 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, and 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.

Theorem 17. *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 usually limit the maximum scope size $|S|$ to a fixed integer L and attempt to generate and augment the COP P with nogoods of length $l \leq L$.

Second, we further observe that some generated nogoods are redundant. Recall that we construct multiple generation CSPs from the analysis of a target COP, and different CSPs generate dominance breaking nogoods of different lengths. By Proposition 1, an implied nogood constraint is both logically redundant and propogation redundant in the COP, and it contributes no extra pruning in a constraint solver. For each solution $(\tilde{\theta}, \tilde{\theta}')$, we add a nogood constraint

$$\vee_{x_i \in var(\tilde{\theta}')} (x_i \notin var(\theta') \vee \theta'[x_i] \neq \tilde{\theta}'[x_i]) \quad (4.20)$$

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

Third, we combine multiple generation CSPs for scopes of the same size into a single generation model. As shown in Example 15, sufficient conditions for betterment (Theorems 5 to 6), implied satisfaction (Theorems 8 to 14) and compatibility (Theorems 15 and 16) are nothing but constraints on a pair of partial assignments when the scope is fixed. Each generation CSP corresponds to a scope, and therefore enumerating all dominance breaking nogoods requires solving $O(\binom{|X|}{l})$ CSPs in total. We observe that generation CSPs have the same type of constraints except that parameters are different for different scopes, and generation CSPs over scopes of the same size can be combined by utilizing element constraints [126] in constraint programming. In Figure 4.4, we give example models for the 0-1 knapsack problem and the generation CSP in the MiniZinc language [100]. The problem is to maximize the value of items chosen from a set of n items subject to the constraint that the total weights of chosen items cannot exceed the capacity limit W. Figure 4.4a is the problem model. 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 of the generation CSP is given in Figure 4.4b. 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, and $v1$ and $v2$ to 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 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 5 and 9 respectively. The last constraint is derived from Theorem 16 and is to ensure the compatibility between the generated dominance breaking nogoods.

```
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 4.4: *Models for the 0-1 knapsack problem in MiniZinc*

In the actual implementation, we modify the publicly available MiniZinc compiler[1] 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.

4.6 Empirical Evaluation

In this section, we put theory into practice and demonstrate the practicality of automatic dominance breaking in solving COPs. We use the MiniZinc modeling language [100] to model the problem and use Chuffed [103] as the backend solver for both nogood generation and problem solving. Note that generating nogoods by solving the generation CSPs and handling additional nogoods in solving a COP may introduce overheads. There is a trade-off between the overhead and the efficiency gained by the extra pruning. We aim to empirically demonstrate the efficiency of nogood generation and the effectiveness of the generated nogoods in solving COPs.

Our method L**-dom** attempts to generate and augment the problem models with nogoods of length up to L. For problem-solving, we compare our method against the basic problem model (**no-dom**) and the model with manual dominance breaking constraints (**manual**). Our experiments use six benchmarks, 20 instances for each problem configuration:

- *Knapsack*: 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 assocaited with it profit p_i and weight w_i. We use instances from an online repository[2] where the number of items $n = 100, 150, 200, 250, 300$. We use the search heuristic to select an item with the highest p_i/w_i first.

- *DisjKnapsack*: the disjunctively constrained knapsack problem [130] is an extension

[1] https://github.com/MiniZinc/libminizinc

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

of the knapsack problem with additional Boolean disjunction constraints. For each instance of *KP* 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 search heuristic is the same as that of *Knapsack*.

- *ConcertSchd*: the capacitated concert hall scheduling problem [45] 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 [45] 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 $10 \leq \frac{p_a}{e_a - s_a + 1}$.

- *MaxCut*: the maximum cut problem is to maximize a *submodular* function on a graph. For $n \in \{35, 40, 45, 50\}$, 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.

- *CombAuc*: 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 [5], where there are $m = 100$ items and $n = 100, 150, 200, 250, 300$ bids in the instances, and the value of a bidder is selected from $[1, 100]$. The probability of an item being covered by a bid is set to 5%.

- *SetCover*: the set covering problem [59] is to minimize a *linear objective* subjective to *linear inequality constraints*. We generate random instances using the scheme of Umetani [125] with $m = 100$ items, $n = 100, 150, 200, 250, 300$ sets, the covering density to be 5%.

Please see Chapter 7 for more detailed descriptions of the benchmark problems. 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.

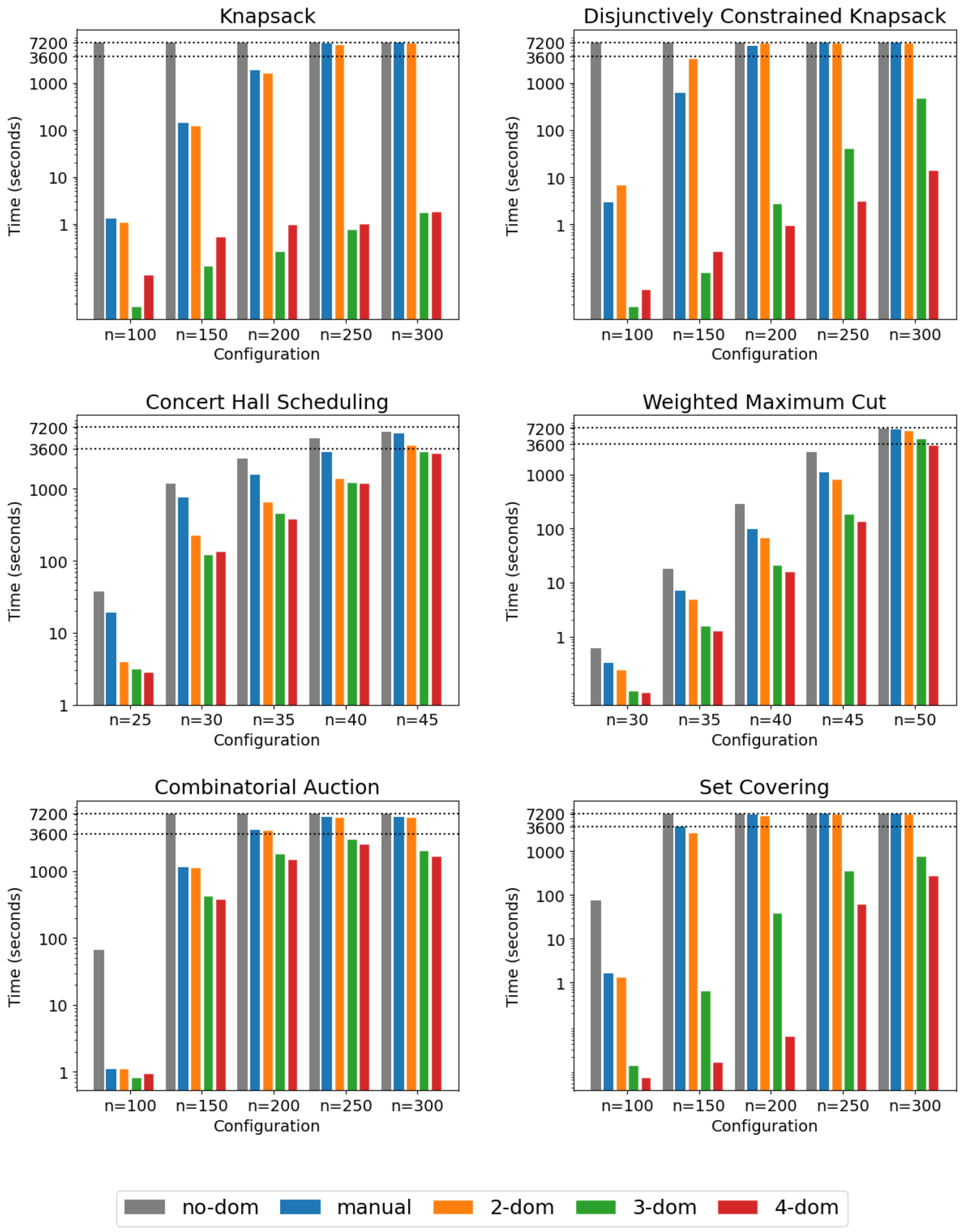


Figure 4.5: *The problem-solving time for different problems*

Figure 4.5 shows the average time for problem-solving in *log scale*. By comparing the problem-solving time (represented by solid bars) 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. More nogood constraints usually help to reduce the search space as demonstrated in *DisjKnapsack*, *ConcertSchd*, *MaxCut*, *CombAuc* and *SetCover*. The exception is that the solving time of **4-dom** is larger than that of **3-dom** for all configurations of *Knapsack* and *DisjKnapsack* with a relatively small number of items. The reason is that **3-dom** is already efficient in problem-solving, and the overhead of adding more nogoods does not compensate for the reduced time.

We also study the overall performance with the average total time (generation time + solving time) as the evaluation metric. Figure 4.6 shows the time for both problem-solving and nogood generation, where the average solving times are represented by solid bars and the nogood generation times are represented by diagonal hatch bars. Note that the ratio of the bars does not reflect the time percentage, and the problem-solving time should not be read directly as the length of the solid bars. By comparing the length of hatched bars for **2-dom**, **3-dom** and **4-dom** in each benchmark, it is obvious to see that more time is needed to generate more nogoods. Therefore, even though more dominance breaking nogoods can help to prune more suboptimal assignments and reduce the time for problem-solving, there is a trade-off between stronger pruning and the overhead of nogood generation. Still, our method comes out on top with the appropriate maximum length of nogoods. In most of the configurations, **3-dom** is usually the best, and the average total time is even smaller than that of **manual**. The only exception is the time for large instances ($n = 50$) of *MaxCut*, where **4-dom** is the best in this configuration. We note that the solving time for many instances exceed the timeout limit, especially for **no-dom** and **manual**, and the actual acceleration of generated nogoods can be even higher.

We give the number of solved instances versus the running time for different methods in Figure 4.7. We can observe that **3-dom** and **4-dom** solves more instances than **no-dom** and **manual** within the timeout limit for all benchmarks. On the other hand, **2-dom**, **3-dom**

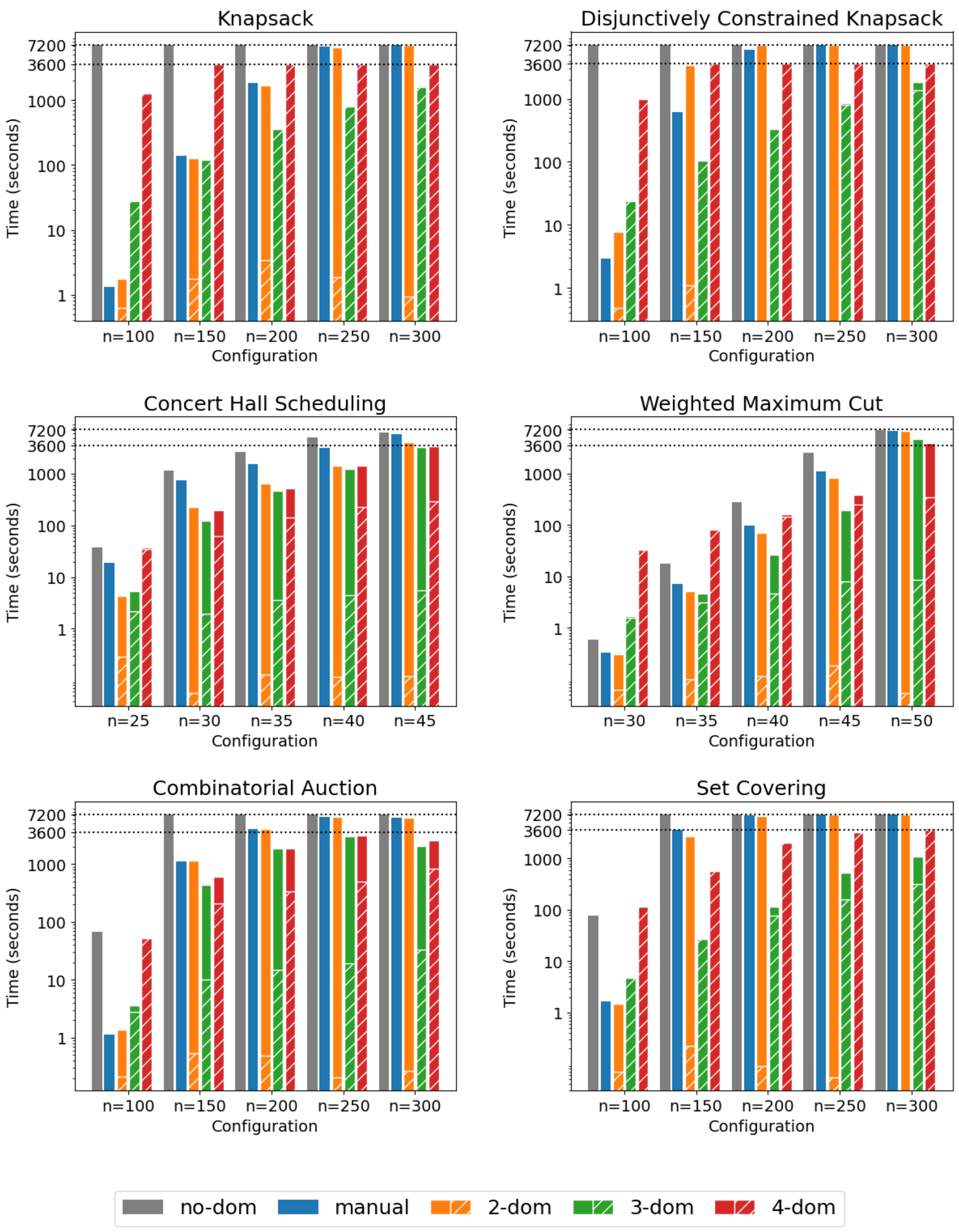


Figure 4.6: *The total time (generation + solving) for different problems*

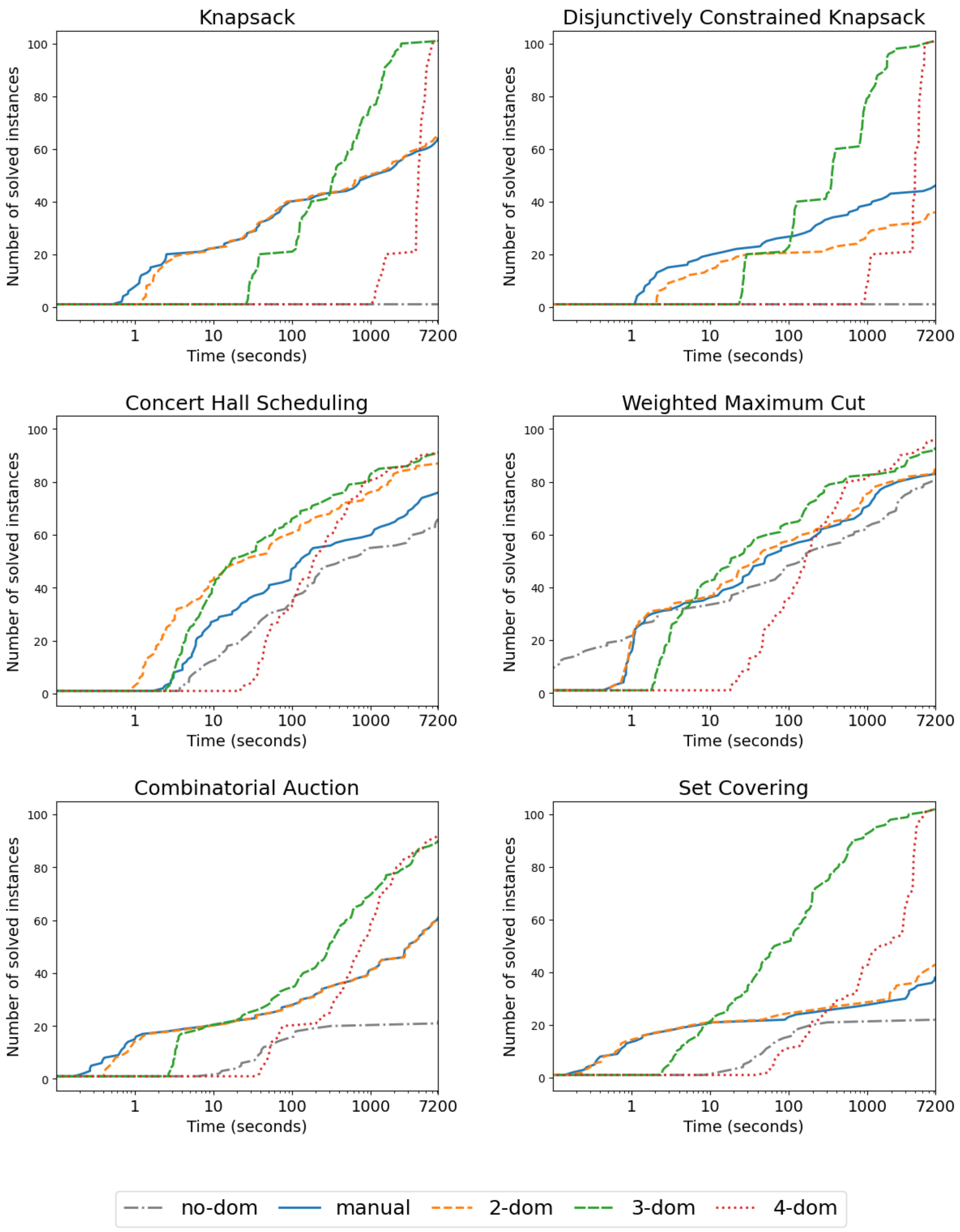


Figure 4.7: *The number of solved instances over time for different problems*

and **4-dom** usually requires more time than **manual** before they start to solve any instances, and this is because the proposed method needs to solve extra generation CSPs to generate nogood constraints for dominance breaking. The overall conclusion is that the advantage of automatically generated nogood constraints becomes more obvious for harder instances. Note that **2-dom** and **manual** have similar performance in *Knapsack*, *MaxCut*, *CombAuc*, and *SetCover*, and we give more theoretical analysis in Chapter 7 to explain the similar behaviors.

4.7 Concluding Remarks

In this chapter, automatic dominance breaking is made possible by focusing on nogoods. Our theorems on sufficient conditions enable us to formulate nogood generation effectively as constraint satisfaction. An important advantage is the ability to control the strength of the generated nogoods. Our method discovers dominance breaking nogoods that had not been discovered before. The method can also be easily integrated into existing constraint modeling systems or solvers. We demonstrate that the method can generate nogoods that speed up the solving of constraint optimization problems in a constraint solver by orders of magnitudes.

Chapter 5

Optimization Techniques for Automatic Dominance Breaking

In this chapter, we extend the applicability of automatic dominance breaking with two theoretical and practical innovations. First, the original method requires all constraints to be efficiently checkable (EC) to guarantee the efficiency of the nogood generation process. We prove formally on when and how non-EC constraints can be ignored in nogood generation by not exploring nogoods that contain variables in the non-EC constraints. This way, sufficient useful nogoods can still be generated when the number of variables in non-EC constraints are relatively small compared to all variables of the problem. Second, we show that some generated nogoods make no contributions in pruning since they are both logically and propagation redundant [20] with respect to other nogoods. We propose *Common Assignment Elimination* to ban generation of such fruitless nogoods, thus speeding up the generation process substantially. Experimentation confirms the enhanced applicability of our theory-backed methods, which allows us to tackle benchmarks that could not be handled by the original method.

5.1 Handling Non-Efficiently Checkable Constraints

To show that a partial assignment θ dominates another θ' in a COP $P = (X, D, C, f)$ by Theorem 3, the implied satisfaction is a necessary condition, which requires that $\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}$, $\bar{\theta}'$ is a solution of P implies that $\mu^{\theta' \to \theta}(\bar{\theta}') \in \mathcal{D}^X_{\theta}$ is also a solution of P. Recall that a full assignment $\bar{\theta}$ is a solution of P if and only if $\bar{\theta}$ satisfies all constraints $c \in C$. The practicality of automatic dominance breaking requires every constraint $c \in C$ to be efficiently checkable, which means there are sufficient conditions on a pair (θ, θ') of partial assignments to show that $\mu^{\theta' \to \theta}(\bar{\theta}')$ satisfies c when $\bar{\theta}'$ satisfies c. This is, however, not always possible. Section 4.3 only give such sufficient conditions for certain classes of constraints. What if P contains constraints with no known sufficient conditions for implied satisfaction (yet)? We propose a way to allow skipping the checking of such non-EC constraints and yet some dominance breaking nogoods can still be found.

We first restate a useful proposition for showing implied satisfaction in Section 4.3. Let $c\theta$ be the constraint obtained by replacing every occurrence of variable x in c by value $\theta[x]$.

Proposition 8. *Let $\theta, \theta' \in \mathcal{D}^S$ be two assignments in a COP $P = (X, D, C, f)$ over the same scope $S \subseteq X$ and $\mu^{\theta' \to \theta} : \mathcal{D}^X_{\theta'} \mapsto \mathcal{D}^X_{\theta}$ be the corresponding mutation mapping. 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 $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$, where $\bar{\theta} = \mu^{\theta' \to \theta}(\bar{\theta}')$.*

For efficiently checkable constraints, there are sufficient conditions to show that $c\theta' \Rightarrow c\theta$. We rely on a useful property to skip checking for non-efficiently checkable constraints.

Lemma 1. *Let $\theta, \theta' \in \mathcal{D}^S$ be two assignments in a COP $P = (X, D, C, f)$ over the same scope $S \subseteq X$ and $\mu^{\theta' \to \theta} : \mathcal{D}^X_{\theta'} \mapsto \mathcal{D}^X_{\theta}$ be the corresponding mutation mapping. Suppose $S \cap var(c) = \emptyset$ for a constraint $c \in C$, then we have $\bar{\theta}' {\downarrow_{var(c)}} = \mu^{\theta' \to \theta}(\bar{\theta}') {\downarrow_{var(c)}}$ for every $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$.*

Proof. Note that $S = var(\theta) = var(\theta')$. The mutation mapping $\mu^{\theta' \to \theta}$ only replaces the θ' component of $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$ with θ, while $\bar{\theta}'[x] = \mu^{\theta' \to \theta}(\bar{\theta}')[x]$ for every $x \in X \setminus S$. If $var(c) \cap var(\theta) = \emptyset$, then $var(c) \subseteq X \setminus S$, and $\bar{\theta} {\downarrow_{var(c)}} = \mu^{\theta' \to \theta}(\bar{\theta}') {\downarrow_{var(c)}}$. □

Recall that a full assignment satisfies a constraint c if $\bar{\theta} {\downarrow_{var(c)}} \in c$. By Lemma 1, when $var(\theta) \cap var(c) = \emptyset$, $\bar{\theta}'$ and $\mu^{\theta' \to \theta}(\bar{\theta}')$ has the same values assigned to variables in $var(c)$ for

```
int: l; % scope size
int: p; % number of variables in non-effectively checkable constraints
array [1..l] of var 1..n: F; % indices of variables in generated nogoods
array [1..p] of var 1..n: N; % indices of skipped variables

constraint forall(i in 1..l, j in 1..p)(F[i] != N[j]);
```

Figure 5.1: *Constraints in a combined model for skipping variables in non-efficiently constraints*

all $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$. Therefore, they must satisfy or violate the constraint c simultaneously. The pair (θ, θ') trivially satisfies implied satisfaction, and do not have to be checked.

Theorem 18. *Let $\theta, \theta' \in \mathcal{D}^S$ be two assignments in a COP $P = (X, D, C_1 \cup C_2, f)$ over the same scope $S \subseteq X$ and $\mu^{\theta' \to \theta} : \mathcal{D}^X_{\theta'} \mapsto \mathcal{D}^X_{\theta}$ be the corresponding mutation mapping. If θ and θ' fulfil that: (1) for all $c \in C_1$, $c\theta' \Rightarrow c\theta$, and (2) for all $c \in C_2$, $S \cap var(c) = \emptyset$, then $\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}$, $\bar{\theta}' \in sol(P)$ implies that $\mu^{\theta' \to \theta}(\bar{\theta}') \in sol(P)$.*

Proof. Let $\bar{\theta}$ be $\mu^{\theta' \to \theta}(\bar{\theta}') \in \mathcal{D}^X_{\theta}$. Suppose that $c \in C_1$. By Proposition 8, since $c\theta' \Rightarrow c\theta$, $\bar{\theta}' \in c \Rightarrow \bar{\theta} \in c$. Otherwise, $c \in C_2$. Implied satisfaction hold automatically by Lemma 1. Thus, $\bar{\theta}'$ satisfies all constraints in P implies that $\bar{\theta}$ satisfies all constraints in P. In other words, $\bar{\theta}' \in sol(P) \Rightarrow \bar{\theta} \in sol(P)$. $\square$

In general, if a constraint c of P is non-efficiently checkable, we can skip solving the generation CSP for a pair (θ, θ') of partial assignments when $var(\theta) \cap var(c) \neq \emptyset$. By Theorems 3 and 18, we can still find some pairs (θ, θ') such that θ dominates θ' with respect to P, but the generated nogoods will not involve variables in $var(c)$. In other words, this method is useful only if $var(c)$ is relatively small with respect to X. Otherwise, very few nogoods can be generated. As shown in Figure 4.4b, generation CSPs over scopes of the same size can be combined into one generation model with element constraints. We can also add an extra constraint into such a model as shown in Figure 5.1, so that θ and θ' containing variables in $var(c)$ would not be explored for identifying dominance breaking nogoods.

5.2 Common Assignment Elimination

The method of automatic dominance breaking constructs multiple generation CSPs for a COP $P = (X, D, C, f)$ to generate dominance breaking nogoods of different lengths, and the constraint set C is augmented with all generated dominance breaking nogoods. In a constraint programming solver, the domain of a COP $P = (X, D, C, f)$ is usually enforced to be *generalized arc consistent* [90] with respect to all constraints $c \in C$. By Proposition 1 in Chapter 2, if $\neg\tilde{\theta}$ and $\neg\theta$ are two nogood constraints such that $\tilde{\theta}$ is a subset of θ, the propagator maintaining GAC for $\neg\tilde{\theta}$ is stronger than that for $\neg\theta$. In other words, the propagator for $\neg\theta$ will not remove any value from the domain, and hence is useless in a constraint programming solver. Section 4.5 describes a method to add a nogood constraint into generation CSPs to avoid generating redundant nogoods. However, a large amount of nogoods in generation CSPs may introduce overheads and hence hinder the efficiency of the nogood generation. In this section, we propose to further enhance generation CSPs with extra conditions to avoid generating redundant nogood constraints derived from pairs of partial assignments with common literals.

By Chapter 4, a pair (θ, θ') of partial assignments over the same scope is a solution of a generation CSP when they satisfy:

- unequal pair: $\theta \neq \theta'$,
- sufficient conditions for betterment of f (Section 4.2),
- sufficient conditions for implied satisfaction of constraints in C (Section 4.3), and
- sufficient conditions for compatibility (Section 4.4).

The conditions can be modelled as constraints over the pair (θ, θ') in generation CSPs. If (θ, θ') satisfies the constraints implies that another pair $(\tilde{\theta}, \tilde{\theta}')$ also satisfies the constraints such that $\tilde{\theta}' \subset \theta'$, then $\neg\theta'$ is a redundant dominance breaking nogood with respect to $\neg\tilde{\theta}'$. In particular, we are interested in the case where a variable x is assigned to the same value in both θ and θ'. We say that a literal $(x = v)$ is *commonly eliminable* with respect to a constraint c in generation CSPs if and only if for all pairs (θ, θ') such that $(x = v) \in \theta \cap \theta'$, (θ, θ') satisfies c implies that $(\tilde{\theta}, \tilde{\theta}')$ satisfies c where $\tilde{\theta} = \theta \setminus \{x = v\}$, $\tilde{\theta}' = \theta' \setminus \{x = v\}$. Immediately, we have the following theorem.

Theorem 19. *Let c be a constraint over a pair of partial assignments. If a literal $(x = v)$ is commonly eliminable with respect to c, then for every pair (θ, θ') of partial assignments satisfying c, there must be another pair $(\tilde{\theta}, \tilde{\theta}')$ also satisfies c such that $(x = v) \notin \tilde{\theta} \cap \tilde{\theta}'$.*

Proof. Suppose that $(x = v)$ is not a common literal of (θ, θ'), i.e. $(x = v) \notin \theta \cap \theta'$. The theorem holds by letting $\tilde{\theta} = \theta$ and $\tilde{\theta}' = \theta'$.

Otherwise, $(x = v) \in \theta \cap \theta'$. Let $\tilde{\theta} = \theta \setminus \{x = v\}$ and $\tilde{\theta}' = \theta' \setminus \{x = v\}$. Immediately, we have $\tilde{\theta}' \subset \theta'$ and $(x = v) \notin \tilde{\theta}' \cap \tilde{\theta}$. Since $(x = v)$ is commonly eliminable with respect to c, (θ, θ') satisfies c implies that $(\tilde{\theta}, \tilde{\theta}')$ satisfies c. $\square$

Theorem 19 implies that if a literal $(x = v)$ is commonly eliminable with respect to all constraints in generation CSPs, then we can enhance a generation CSP for a pair (θ, θ') with an extra condition $\theta[x] \neq v \vee \theta'[x] \neq v$ when $x \in var(\theta)$ to avoid generating redundant dominance breaking nogoods. Further, if for every value $v \in D(x)$, $(x = v)$ is commonly eliminable with respect to the generation CSP, then we can add a more succinct constraint $(\theta[x] \neq \theta'[x])$ to generation CSP when $x \in var(\theta)$. The technique is called *Common Assignment Elimination (CAE)*. The collective strength of the resulting dominance nogoods in pruning search space for P after applying CAE remains unchanged. Thus, the key question is how to show that a literal is commonly eliminable.

By definition, checking eliminability of a literal requires us to go through all constraints in generation CSPs. It is straightforward to show that *any arbitrary* literal is commonly eliminable with respect to the constraint for the unequal pair condition.

Proposition 9. *Let (θ, θ') be a pair of partial assignments of a COP P over the same scope. If $(x = v) \in \theta \cap \theta'$, then $(\theta \neq \theta') \Rightarrow (\tilde{\theta} \neq \tilde{\theta}')$, where $(\tilde{\theta}, \tilde{\theta}') = (\theta \setminus \{x = v\}, \theta' \setminus \{x = v\})$.*

What remains is to analyze what kind of literals are commonly eliminable with respect to constraints representing sufficient conditions for the betterment, implied satisfaction and compatibility conditions. Recall from Sections 4.2 to 4.4 that such sufficient conditions are all arisen from the objective and constraints of a constraint optimization problem. In the remaining of this section, we will give commonly eliminable literals for specific objectives and constraints respectively, and we will summarize how to apply common assignment elimination in Section 5.2.4.

5.2.1 Eliminability for Betterment

This subsection gives commonly eliminable literals with respect to sufficient conditions for betterment in Section 4.2. For the ease of presentation, we restate the sufficient conditions before giving the results of commonly eliminable literals. We consider two types of objectives: separable objectives and supermodular/submodular objectives.

Separable Objectives

A function f is *separable* if it can be written as a sum of functions of individual variables, i.e. $f(\bar{\theta}) = f_1(v_1) + \cdots + f_n(v_n)$ for a full assignment $\bar{\theta} = \{(x_i = v_i) \mid x_i \in X\}$, where each component is $f_i : \mathbb{Z} \to \mathbb{R}$. The projection of a separable function f is defined as $f{\downarrow_S}(\theta) = \sum_{x_i \in S} f_i(\theta[x_i])$ for a partial assignment $\theta \in \mathcal{D}^S$. By Theorem 5, the sufficient condition for betterment is:

$$f{\downarrow_S}(\theta) \leq f{\downarrow_S}(\theta'), \tag{5.1}$$

where S is the scope of θ and θ'. The theorem below states that *an arbitrary assignment* $(x = v)$ is commonly eliminable with respect to (5.1) if f is separable.

Theorem 20. *Let $\theta, \theta' \in \mathcal{D}^S$ be two assignments in a COP $P = (X, D, C, f)$ over the same scope $S \subseteq X$, and the objective f be a separable function. If $(x_t = v_t) \in \theta \cap \theta'$, then $f{\downarrow_S}(\theta) \leq f{\downarrow_S}(\theta')$ implies that $f{\downarrow_{\tilde{S}}}(\tilde{\theta}) \leq f{\downarrow_{\tilde{S}}}(\tilde{\theta}')$ where $S = var(\theta) = var(\theta')$, $\tilde{S} = S \setminus \{x_t\}$, $\tilde{\theta} = \theta \setminus \{x_t = v_t\}$ and $\tilde{\theta}' = \theta' \setminus \{x_t = v_t\}$.*

Proof. If $f{\downarrow_S}(\theta) \leq f{\downarrow_S}(\theta')$, then

$$f{\downarrow_{\tilde{S}}}(\tilde{\theta}) = f{\downarrow_S}(\theta) - f_t(v_t) \leq f{\downarrow_S}(\theta') - f_t(v_t) = f{\downarrow_{\tilde{S}}}(\tilde{\theta}').$$

Therefore, $f{\downarrow_S}(\theta) \le f{\downarrow_S}(\theta')$ implies that $f{\downarrow_{\tilde{S}}}(\tilde{\theta}) \le f{\downarrow_{\tilde{S}}}(\tilde{\theta}')$. □

Supermodular and Submodular Objectives

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

$$g(V \cup T) - g(V) \le g(V' \cup T) - g(V')$$

for every $V, V' \subseteq U$ where $V \subseteq V'$ and $T \subseteq U \setminus V'$. A set function g is *submodular* if $-g$ is supermodular. Given an assignment θ over 0-1 variables, we define a set $V(\theta) = \{i \mid \theta[x_i] = 1\}$. Recall in Section 4.2.2 that a function f on 0-1 variables is *equivalent to a supermodular function* g if $f(\bar{\theta}) = g(V(\bar{\theta}))$ for any full assignment $\bar{\theta}$ of P. By Theorem 6, the sufficient condition for betterment is:

$$f{\downarrow_S}(\theta) \le f{\downarrow_S}(\theta') \wedge V(\theta) \subseteq V(\theta'), \tag{5.2}$$

Removing *a common assignment* $(x = 0)$ from θ and θ' does not affect the represented set $V(\theta)$ and $V(\theta')$. Thus, the condition (5.2) holds for $\tilde{\theta} = \theta \setminus (x = v)$ and $\tilde{\theta}' = \theta' \setminus (x = v)$ whenever it holds for θ and θ'.

Theorem 21. *Let $\theta, \theta' \in \mathcal{D}^S$ be two assignments in a COP $P = (X, D, C, f)$ over the same scope $S \subseteq X$, and the objective f be a function on 0-1 variables that is equivalent to a supermodular function g. If $(x_t = 0) \in \theta \cap \theta'$, then*

(1) $g(V(\theta)) \le g(V(\theta')) \Rightarrow g(V(\tilde{\theta})) \le g(V(\tilde{\theta}'))$

(2) $V(\theta) \subseteq V(\theta') \Rightarrow V(\tilde{\theta}) \subseteq V(\tilde{\theta}')$

where $\tilde{\theta} = \theta \setminus \{x_t = 0\}$ and $\tilde{\theta}' = \theta' \setminus \{x_t = 0\}$.

Proof. Since $V(\theta) = \{i \mid (x_i = 1) \in \theta\}$, $V(\tilde{\theta}) = V(\theta \setminus \{x_t = 0\}) = V(\theta)$. Similarly, $V(\tilde{\theta}') = V(\theta')$. If $g(V(\theta)) \le g(V(\theta'))$, then $g(V(\tilde{\theta})) = g(V(\theta)) \le g(V(\theta')) = g(V(\tilde{\theta}'))$. Also, if $V(\theta) \subseteq V(\theta')$, then $V(\tilde{\theta}) = V(\theta) \subseteq V(\theta') = V(\tilde{\theta}')$. □

5.2.2 Eliminability for Implied Satisfaction

This subsection gives commonly eliminable literals with respect to sufficient conditions for implied satisfaction in Section 4.3. Note that by Proposition 2, it is sufficient to show that $c\theta' \Rightarrow c\theta$. Again, we will restate the sufficient condition before giving the common eliminable literals.

Domain Constraints

A domain constraint $x \in D(x)$ is a unary constraint that restricts a variable x to take values from $D(x)$. By Theorem 8, the sufficient condition for implied satisfaction only requires that $\theta[x] \in D(x)$. Any *arbitrary assignment* $(x = v)$ is commonly eliminable with respect to the sufficient condition for domain constraints.

Theorem 22. *Let $\theta, \theta' \in \mathcal{D}^S$ be two assignments in a COP $P = (X, D, C, f)$ over the same scope $S \subseteq X$, and $x_i \in D(x_i)$ be a domain constraint. If $(x_t = v_t) \in \theta \cap \theta'$, then $c\theta' \Rightarrow c\theta$ implies that $c\tilde{\theta}' \Rightarrow c\tilde{\theta}$, where $\tilde{\theta} = \theta \setminus \{x_t = v_t\}$ and $\tilde{\theta}' = \theta' \setminus \{x_t = v_t\}$.*

Proof. There are two cases. If x_i and x_t are the same, then so must v_i and v_t by definition of a partial assignment, and $x_i \notin var(\tilde{\theta})$. We have $c\tilde{\theta}' \Leftrightarrow c \Leftrightarrow c\tilde{\theta}$. Otherwise, x_i and x_t are different, then we have $c\tilde{\theta}' \Leftrightarrow c\theta' \Rightarrow c\theta \Leftrightarrow c\tilde{\theta}$. $\square$

Linear Inequality Constraints

A linear inequality constraint has the form $\sum w_i x_i \leq b$ where $w_i, b \in \mathbb{R}$. By Theorem 9, the sufficient condition for implied satisfaction is:

$$\sum_{x_i \in S'} w_i \theta[x_i] \leq \sum_{x_i \in S'} w_i \theta'[x_i], \tag{5.3}$$

The sufficient condition still holds when any *arbitrary common literal* $(x = v) \in \theta \cap \theta'$ is removed from θ and θ'.

Theorem 23. *Let $\theta, \theta' \in \mathcal{D}^S$ be two assignments in a COP $P = (X, D, C, f)$ over the same scope $S \subseteq X$, and $\sum w_i x_i \leq b$ be a linear inequality constraint. If $(x_t = v_t) \in \theta \cap \theta'$, then*

$\sum_{(x_i=v_i)\in\theta} w_i v_i \leq \sum_{(x_i=v'_i)\in\theta'} w_i v'_i$ *implies that* $\sum_{(x_i=v_i)\in\tilde{\theta}} w_i v_i \leq \sum_{(x_i=v'_i)\in\tilde{\theta}'} w_i v'_i$*, where* $\tilde{\theta} = \theta \setminus \{x_t = v_t\}$ *and* $\tilde{\theta}' = \theta' \setminus \{x_t = v_t\}$.

The proof idea is similar to that of Theorem 20.

Boolean Disjunctions

The *Boolean disjunction* constraint $\vee_{x_i \in B} x_i$ requires that at least one Boolean variable in the set B takes the true value. It can be extended where Boolean variables are results of binary comparisons $x_i \circ_i b_i$ where $x_i \in X$ is a variable, $b_i \in \mathbb{Z}$ is an integer and $\circ_i \in \{=, \neq, \geq, \leq, <, >\}$ is a binary comparison operator. The sufficient condition for implied satisfaction is:

$$\vee_{x_i \in B}(\theta'[x_i] \circ_i b_i) \Rightarrow \vee_{x_i \in B}(\theta[x_i] \circ_i b_i), \tag{5.4}$$

Note that a Boolean disjunction constraint has the property that $e \vee 0 = e$ where e is a Boolean expression that returns a Boolean value. Thus, the common assignment $(x_i = v_i)$ that results in $false$ for a Boolean expression $(\theta[x_i] \circ_i b_i)$, then it does not affect the validity of the sufficient condition for implied satisfaction.

Theorem 24. *Let* $\theta, \theta' \in \mathcal{D}^S$ *be two assignments in a COP* $P = (X, D, C, f)$ *over the same scope* $S \subseteq X$*, and* c *be a Boolean disjunction constraint* $\vee_{x_i \in B}(x_i \circ_i b_i)$*. If* $(x_t = v_t) \in \theta \cap \theta'$ *and* $v_t \circ_t b_t = false$*, then* $\vee_{(x_i=v'_i)\in\theta'}(v'_i \circ_i b_i) \Rightarrow \vee_{(x_i=v_i)\in\theta}(v_i \circ_i b_i)$ *implies that* $\vee_{(x_i=v'_i)\in\tilde{\theta}'}(v'_i \circ_i b_i) \Rightarrow \vee_{(x_i=v_i)\in\tilde{\theta}}(v_i \circ_i b_i)$*, where* $\tilde{\theta} = \theta \setminus \{x_t = v_t\}$ *and* $\tilde{\theta}' = \theta' \setminus \{x_t = v_t\}$.

Proof. Since $(v_t \circ_t b_t)$ is $false$, we have

$$\vee_{(x_i=v'_i)\in\tilde{\theta}'}(v'_i \circ_i b_i) = \vee_{(x_i=v'_i)\in\tilde{\theta}'}(v'_i \circ_i b_i) \vee false = \vee_{(x_i=v'_i)\in\theta'}(v'_i \circ_i b_i)$$

Similarly, we have

$$\vee_{(x_i=v_i)\in\tilde{\theta}}(v_i \circ_i b_i) = \vee_{(x_i=v_i)\in\tilde{\theta}}(v_i \circ_i b_i) \vee false = \vee_{(x_i=v_i)\in\theta}(v_i \circ_i b_i)$$

Thus, we can show that $\vee_{(x_i=v'_i)\in\theta'}(v'_i \circ_i b_i) \Rightarrow \vee_{(x_i=v_i)\in\theta}(v_i \circ_i b_i)$ implies $\vee_{(x_i=v'_i)\in\tilde{\theta}'}(v'_i \circ_i b_i) \Rightarrow \vee_{(x_i=v_i)\in\tilde{\theta}}(v_i \circ_i b_i)$. □

Counting Constraints

Section 4.3.4 introduces two basic counting 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. More sophisticated counting constraints such as an *alldifferent* constraint, an *alldifferent_except_0* constraint, an *among* constraint and a global cardinality constraint can be treated as a conjunction of several *atleast* and *atmost* constraints. By Theorems 12 and 13, the sufficient condition for an *atleast* constraint is

$$|\{x \mid \theta[x] \in V \land x \in S'\}| \geq |\{x \mid \theta'[x] \in V \land x \in S'\}|, \tag{5.5}$$

and the sufficient condition for an *atmost* constraint is

$$|\{x \mid \theta[x] \in V \land x \in S'\}| \leq |\{x \mid \theta'[x] \in V \land x \in S'\}|, \tag{5.6}$$

where $S' = T \cap S$ and $S = var(\theta) = var(\theta')$. Removing an arbitrary common assignment $x = v$ from θ and θ' does not affect the validity of (5.5) and (5.6).

Theorem 25. *Let $\theta, \theta' \in \mathcal{D}^S$ be two assignments in a COP $P = (X, D, C, f)$ over the same scope $S \subseteq X$, and atleast(T, V, k) and atmost(T, V, k) be two constraints where $T \subseteq X$ is a set of variables, V is a set of values, and $k \in \mathbb{N}$ is an integer. If $(x_t = v_t) \in \theta \cap \theta'$, then*

$$\begin{aligned}&(|\{x \mid \theta[x] \in V \land x \in S'\}| \geq |\{x \mid \theta'[x] \in V \land x \in S'\}|)\\ \Rightarrow&(|\{x \mid \tilde{\theta}[x] \in V \land x \in \tilde{S}'\}| \geq |\{x \mid \tilde{\theta}'[x] \in V \land x \in \tilde{S}'\}|),\end{aligned}$$

and

$$\begin{aligned}&(|\{x \mid \theta[x] \in V \land x \in S'\}| \leq |\{x \mid \theta'[x] \in V \land x \in S'\}|)\\ \Rightarrow&(|\{x \mid \tilde{\theta}[x] \in V \land x \in \tilde{S}'\}| \leq |\{x \mid \tilde{\theta}'[x] \in V \land x \in \tilde{S}'\}|),\end{aligned}$$

where $S' = S \cap T$, $\tilde{\theta} = \theta \setminus \{x = v\}$, $\tilde{\theta}' = \theta' \setminus \{x = v\}$, and $\tilde{S}' = var(\tilde{\theta}) \cap T$.

Proof. For the ease of presentation, we let $d = |\{x \mid \theta[x] \in V \land x \in S'\}|$, $d' = |\{x \mid \theta'[x] \in V \land x \in S'\}|$, $\tilde{d} = |\{x \mid \tilde{\theta}[x] \in V \land x \in \tilde{S}'\}|$ and $\tilde{d}' = |\{x \mid \tilde{\theta}'[x] \in V \land x \in \tilde{S}'\}|$. There are two cases. If $x_t \notin T$, then $d = \tilde{d}$, and $d' = \tilde{d}'$, the statement holds automatically. Otherwise, $x_t \in T$, and we have $d = \tilde{d} + 1$, and $d' = \tilde{d}' + 1$, which implies that the inequalities also hold for $\tilde{\theta}$ and $\tilde{\theta}'$. $\square$

It is straightforward to see that an arbitrary assignment is also commonly eliminable with respect to sufficient conditions for other counting constraints since they can be formed as conjunctions of several *atleast* and *atmost* constraints.

5.2.3 Eliminability for Compatibility

Recall from Section 4.4 that the compatibility of generated dominance breaking nogoods can be achieved by enforcing

$$(h_1{\downarrow_S}(\theta), \ldots, h_m{\downarrow_S}(\theta)) <_{lex} (h_1{\downarrow_S}(\theta'), \ldots, h_m{\downarrow_S}(\theta')), \tag{5.7}$$

between θ and θ', where $S = var(\theta) = var(\theta')$, and $h_1, \ldots, h_m$ are sensitive functions arisen from the objective and constraints of the target COP. Suppose $(x_t = v_t) \in \theta \cap \theta'$ is a common literal, and $\tilde{\theta} = \theta \setminus (x_t = v_t)$ and $\tilde{\theta}' \setminus (x_t = v_t)$ are obtained by removing the common literal such that $\tilde{S} = var(\tilde{\theta}) = var(\tilde{\theta}')$. The following gives a sufficient condition for the eliminability of a literal with respect to the generalized lexicographical ordering constraint in (5.7).

Proposition 10. *If $h_1, \ldots, h_m$ are sensitive functions such that for each function h_i, we have $h_i{\downarrow_S}(\theta) < h_i{\downarrow_S}(\theta')$ implies that $h_i{\downarrow_{\tilde{S}}}(\tilde{\theta}) < h_i{\downarrow_{\tilde{S}}}(\tilde{\theta}')$, then*

$$\begin{aligned}&(h_1{\downarrow_S}(\theta), \ldots, h_m{\downarrow_S}(\theta)) <_{lex} (h_1{\downarrow_S}(\theta'), \ldots, h_m{\downarrow_S}(\theta'))\\ \Rightarrow&(h_1{\downarrow_{\tilde{S}}}(\tilde{\theta}), \ldots, h_m{\downarrow_{\tilde{S}}}(\tilde{\theta})) <_{lex} (h_1{\downarrow_{\tilde{S}}}(\tilde{\theta}'), \ldots, h_m{\downarrow_{\tilde{S}}}(\tilde{\theta}')),\end{aligned}$$

where $S = var(\theta) = var(\theta')$ and $\tilde{S} = var(\tilde{\theta}) = var(\tilde{\theta}')$.

Proposition 10 implies that we can consider eliminable literals with respect to the condition $h_i{\downarrow_S}(\theta) < h_i{\downarrow_S}(\theta')$ for each sensitive function h_i individually. It is straightforward to see that the commonly eliminable literals with respect to $h_i{\downarrow_S}(\theta) < h_i{\downarrow_S}(\theta')$ are the same as those with respect to betterment for separable objectives and supermodular objectives, and those with respect to implied satisfaction for linear inequality constraints and counting constraints when the sensitive functions are defined as those in Section 4.4.2.

Proposition 11. *Suppose f is a separable function such that $f(\bar{\theta}) = \sum f_i(\bar{\theta}[x_i])$ for all $\bar{\theta} \in \mathcal{D}^X$, and $v_t \in D(x_t)$ is an arbitrary value. If we define the projection function of f to be $f{\downarrow_S}(\theta) = \sum_{x_i \in S} f_i(\theta[x_i])$ for all $S \subseteq X$, then $f{\downarrow_S}(\theta) < f{\downarrow_S}(\theta')$ implies that $f{\downarrow_{\tilde{S}}}(\tilde{\theta}) < f{\downarrow_{\tilde{S}}}(\tilde{\theta}')$.*

Proposition 12. *Suppose f is a function that is equivalent to a supermodular function g, and $v_t = 0 \in D(x_t)$. If we define the projection function of f to be $f \downarrow_S (\theta) = g(S(\theta))$ where $S(\theta) = \{i \mid \theta[x_i] = 1\}$, then $f{\downarrow_S}(\theta) < h{\downarrow_S}(\theta')$ implies that $f{\downarrow_{\tilde{S}}}(\tilde{\theta}) < f{\downarrow_{\tilde{S}}}(\tilde{\theta}')$.*

Proposition 13. *Suppose c is a linear inequality constraint ($\sum w_i x_i \leq b$), and $v_t \in D(x_t)$ is an arbitrary value. If we define a function $h : \mathcal{D}^X \mapsto \mathbb{R}$ such that $h(\bar{\theta}) = \sum w_i \bar{\theta}[x_i]$ for all full assignment $\bar{\theta} \in \mathcal{D}^X$, and the projection function $h{\downarrow_S}(\theta) = \sum_{x_i \in S} w_i \theta[x_i]$, then $h{\downarrow_S}(\theta) < h{\downarrow_S}(\theta')$ implies that $h{\downarrow_{\tilde{S}}}(\tilde{\theta}) < h{\downarrow_{\tilde{S}}}(\tilde{\theta}')$.*

Proposition 14. *Suppose c is a constraint atmost(S, V, k), and $v_t \in D(x_t)$ is an arbitrary value. If we define a function $h : \mathcal{D}^X \mapsto \mathbb{R}$ such that $h(\bar{\theta}) = |\{x \mid \bar{\theta}[x] \in V\}|$ for a full assignment $\bar{\theta} \in \mathcal{D}^X$, and the projection function $h{\downarrow_S}(\theta) = |\{x \mid \theta[x] \in V \wedge x \in S\}|$, then $h{\downarrow_S}(\theta) < h{\downarrow_S}(\theta')$ implies that $h{\downarrow_{\tilde{S}}}(\tilde{\theta}) < h{\downarrow_{\tilde{S}}}(\tilde{\theta}')$.*

Proposition 15. *Suppose c is a constraint atleast(S, V, k), and $v_t \in D(x_t)$ is an arbitrary value. If we define a function $h : \mathcal{D}^X \mapsto \mathbb{R}$ such that $h(\bar{\theta}) = -|\{x \mid \bar{\theta}[x] \in V\}|$ for all full assignment $\bar{\theta} \in \mathcal{D}^X$, and the projection function $h{\downarrow_S}(\theta) = -|\{x \mid \theta[x] \in V \wedge x \in S\}|$, then $h{\downarrow_S}(\theta) < h{\downarrow_S}(\theta')$ implies that $h{\downarrow_{\tilde{S}}}(\tilde{\theta}) < h{\downarrow_{\tilde{S}}}(\tilde{\theta}')$.*

The proof of Proposition 11 to 15 are similar to those for Theorems 20 to 25. In other words, commonly eliminable literals for each variable can be obtained from the involved objective and constraints in the target COP.

5.2.4 Applying Common Assignment Elimination

In this section, we describe and showcase how to apply the technique of common assignment elimination. We summarize the results of commonly eliminable literals in Table 5.1, and the procedure to collect commonly eliminable literals for each variable is described in Algorithm 2. When constructing the generation CSPs for a constraint optimization problem,

Objective/Constraint types	Commonly Eliminable Literals
Separable objectives	$x_i = v_i, \forall v_i \in D(x_i)$
Supermodular/submodular objectives	$x_i = 0$
Domain constraints	$x_i = v_i, \forall v_i \in D(x_i)$
Linear inequality constraints	$x_i = v_i, \forall v_i \in D(x_i)$
Boolean disjunction constraints	$x_i = v_i$ s.t. $v_i \circ_i b_i = false$
Counting constraints	$x_i = v_i, \forall v_i \in D(x_i)$

Table 5.1: *Commonly eliminable literals for a variable x_i*

Algorithm 2 Common assignment elimination

```
Input: a COP $P = (X, D, C, f)$
1: Initialize $Q \leftarrow \emptyset$
2: for each variable $x_t \in X$ do
3:     $V \leftarrow \{x_t = v \mid v \in D(x_t)\}$
4:     if $x_t$ is an argument of a supermodular/submodular objective then
5:         $V \leftarrow V \cap \{x_t = 0\}$
6:     end if
7:     if $x_t \in var(c)$ where $c \equiv (\vee_{x_i \in B} x_i \circ_i b_i)$ then
8:         $V \leftarrow V \cap \{x_t = v \mid v \in V$ s.t. $v \circ_t b_t = false\}$
9:     end if
10:    if $|V| == |D(x_t)|$ then
11:        $Q \leftarrow Q \cup \{x_t \in S \Rightarrow \theta[x_t] \neq \theta'[x_t]\}$
12:    else
13:        $Q \leftarrow Q \cup \{(x_t = v) \notin \theta \cap \theta' \mid v \in V\}$
14:    end if
15: end for
16: Return $Q$
```

we collect the common eliminable literals for each variable. The set of commonly eliminable literals consist of all possible literals for a variable x_t initially (Line 3), and we shrink the set by checking whether x_t is involved in a supermodular/submodular objective or a Boolean disjunction constraint (Line 4 to 9). Finally, the constraints for common assignment elimination are collected and later added to generation CSPs for boosting the generation of dominance breaking nogoods. Note that when $(x_t = v)$ is commonly eliminable with respect to all sufficient conditions in the generation CSP, a more succinct constraint is added to the set Q (Line 11).

In the following, we give an example to show CAE in action.

Example 17. *Consider the COP in (1.1) and the generation CSP in (4.2). The COP has a linear objective function and a linear inequality constraint. Following Algorithm 2, any arbitrary literal for each variable $x \in X$ is commonly eliminable. Thus, we add two constraints $\theta[x_3] \neq \theta'[x_3]$ and $\theta[x_4] \neq \theta'[x_4]$ to (4.2).*

Suppose we extend P with a Boolean disjunction constraint $x_1 \vee x_4$. By Theorem 24, only $(x_4 = 0)$ is commonly eliminable literals for x_4 with respect to the sufficient condition for $x_1 \vee x_4$, while any arbitrary assignment $x_i = v$ is commonly eliminable with respect to the objective and other constraints. Thus, we only add the constraint $\theta[x_4] \neq 0 \vee \theta'[x_4] \neq 0$ and $\theta[x_3] \neq \theta'[x_3]$ to (4.2) for common assignment elimination.

When we combine multiple generation CSPs over scopes of the same size into one succinct model, we can still apply CAE by adding the constraint $(x \in var(\theta)) \Rightarrow (\theta[x] \neq v \vee \theta'[x] \neq v)$ for a commonly eliminable literal $(x = v)$ and the constraint $(x \in var(\theta)) \Rightarrow (\theta[x] \neq \theta'[x])$ when literals $(x = v)$ for values in the domain $D(x)$ are all commonly eliminable.

5.3 Experimental Evaluation

We perform experiments on Xeon E7-4830 2.20GHz processors using MiniZinc 2.4.3 [100] to model both the benchmark problems and the corresponding dominance breaking nogood generation respectively. The back-end solver is Chuffed [103]. We use eight benchmark problems, and generate 20 instances for each problem configuration in order to give meaningful comparison. There are six benchmarks including *Knapsack*, *DisjKnapsack*, *ConcertSched*, *MaxCut*, *SetCover* and *CombAuc* are the same as that in Section 4.6. We also introduce two additional benchmarks:

- *KnapsackSide-n* are extensions of the 0-1 knapsack problems to showcase the usefulness of Theorem 18. An instance with n items is augmented with $\lfloor 0.02n \rfloor$ table constraints, in which we randomly select three variables $x_{i_1}, x_{i_2}, x_{i_3}$ and sample each tuple from $D(x_{i_1}) \times D(x_{i_2}) \times D(x_{i_3})$ with probability 0.5. Note that side constraints can also be added to other benchmark problems.

- *PCBoard-n-m* are PC board manufacturing problems [91] with n components and m devices, where there are *linear inequality constraints* and a *linear objective*. The model is from a public MiniZinc repository[1]. Due to the problem structure, the length of dominance breaking nogoods is at least 4.

Note that the original method cannot handle *KnapsackSide* because of the side constraints. In addition, nogood generation for *PCBoard* was too time-consuming with the original method, and the benefits in search reduction cannot compensate the overhead. We use the search strategies specified in the public models in solving the COPs, and the default of Chuffed for the generation models. For all benchmarks, we attempt to generate all dominance breaking nogoods of length up to L without CAE (L**-dom**) or with CAE (L**-dom(*)**) as per our proposal.

5.3.1 Comparison of Generation Time

We use a uniform timeout limit of 1 hour for nogood generation. Figures 5.2 and 5.3 show the time of nogood generation in *log scale*. The CAE technique clearly reduces the nogood generation time for all benchmarks. Note that generating long nogoods without CAE could time out for some large instances, and the difference between **4-dom** and **4-dom(*)** (between **6-dom** and **6-dom(*)** for *PCBoard*) can be more pronounced.

The detailed statistics of the generation times is also included in Table A.2. Let $t_g(L)$ and $t_g^*(L)$ denote the average time for generating nogoods of length up to L without and with CAE respectively. We also compute the *percentage decrease of generation time* $\%_g$, where $\%_g(L) = \frac{t_g(L) - t_g^*(L)}{t_g(L)}$. We only compare L**-dom** and L**-dom(*)** when both of them do not time out. The trends in *Knapsack*, *DisjKnapsack*, *ConcertSched*, *MaxCut*, *CombAuc* and *SetCover* are similar. The percentage decrease is up to 80.87% for $L = 2$, up to 91.42% for $L = 3$ and up to 97.59% for $L = 4$. As for *PCBoard*, **4-dom(*)** and **5-dom(*)** reduce up to 86.83%, 97.17%

[1]https://people.eng.unimelb.edu.au/pstuckey/dominance/

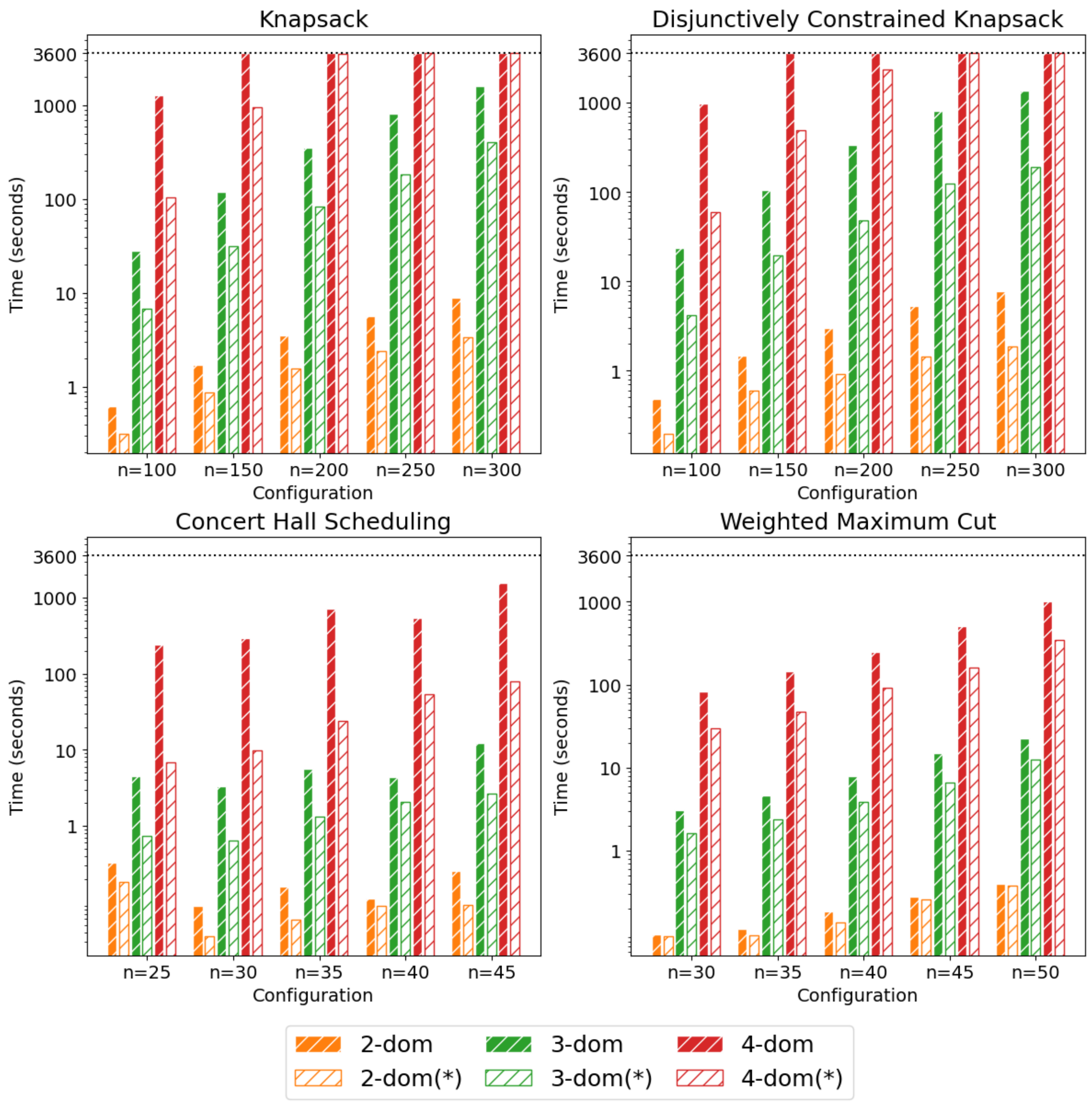


Figure 5.2: *Comparisons of the generation time for* Knapsack, DisjKnapsack, ConcertSched *and* MaxCut

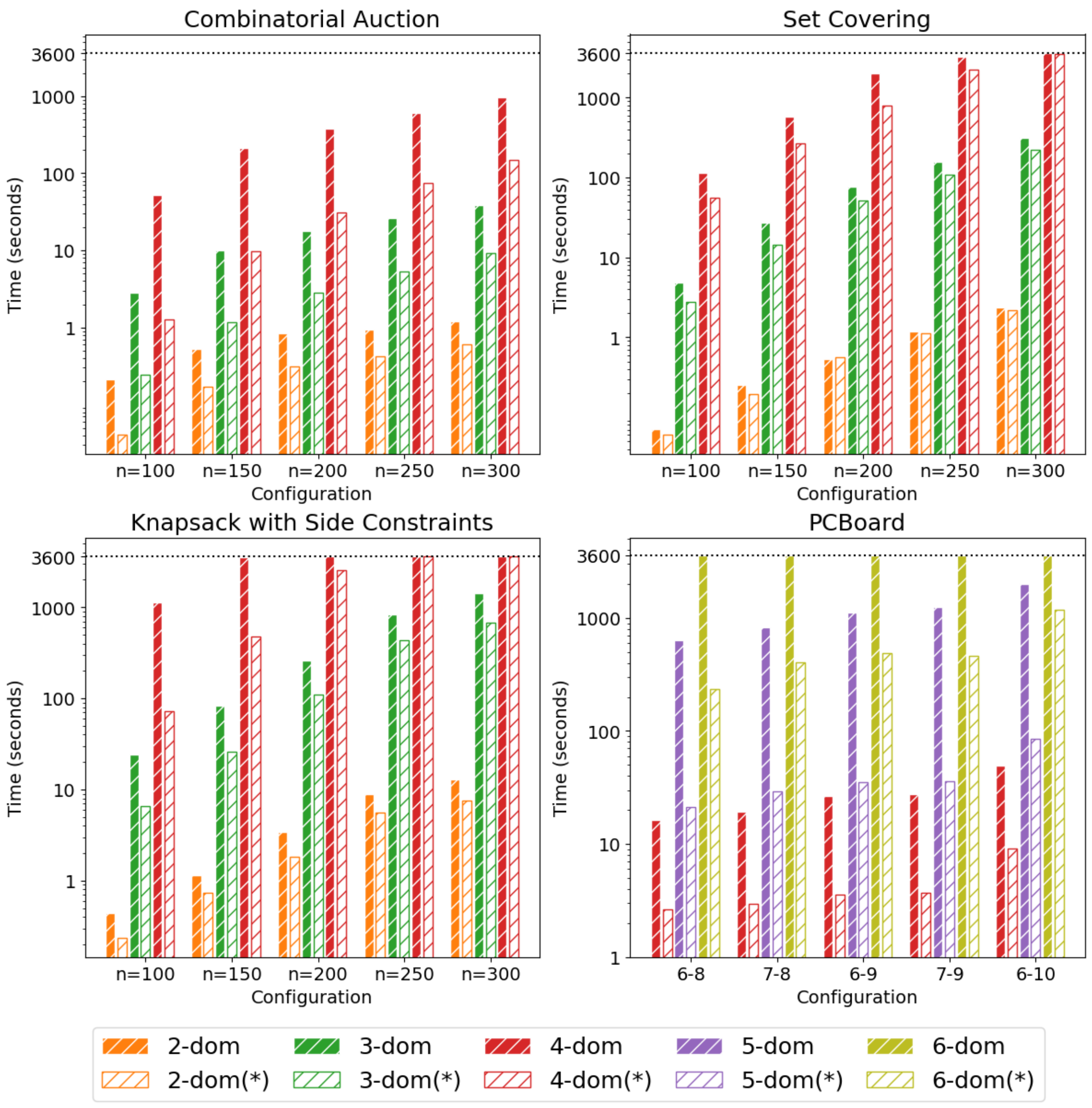


Figure 5.3: *Comparisions of the generation time for* CombAuc, SetCover, KnapsackSide *and* PCBoard

generation time compared with **4-dom** and **5-dom** respectively. The **6-dom** method time out for all problem sizes, while **6-dom(*)** can generate all desired nogoods within 20 minutes. The benefits of CAE become more significant as we attempt to generate longer nogoods.

5.3.2 Comparison of Overall Performance

We study the overall performance with the total time (generation time + solving time) as the evaluation metric. The timeout for the whole solving process (nogood generation + problem-solving) is set to 2 hours, while we keep the timeout for nogood generation as 1 hour. If nogood generation times out, we augment the problem model with only the nogoods generated so far. We also perform experiments using no dominance breaking constraint (**no-dom**) and using manual dominance breaking constraints (**manual**) from the literature [24, 45] , where the timeout is set to 2 hours. Figures 5.4 and 5.5 show the average solving (solid bars) and generation time (diagonal hatch bars) in *log scale*, where the bars for problem-solving time are stacked on the bars for generation time. Automatic dominance breaking outperforms **manual** in almost all benchmarks even before applying CAE. The exception is *PCBoard*, where the method of automatic dominance breaking outperforms **manual** only after CAE is applied.

The detailed statistics of the total times is also included in Table A.3. To compare the total time for L**-dom** and L**-dom(*)**, we also compute the *percentage decrease of total time* $\%_t$ for measurement, i.e. $\%_t(L) = \frac{t(L)-t^*(L)}{t(L)}$, where $t(L)$ and $t^*(L)$ are the average total time of COPs augmented with generated nogood of length up to L. In general, the performance gain of CAE depends on whether nogood generation takes a large part in the whole solving process. The problem structures of *Knapsack*, *DisjKnapsack* and *KnapsackSide* are similar, where the problem augmented with nogoods of length up to 3 are usually the best. The percentage decrease in runtime is at least 42.73% and at most 85.03% after CAE is applied. For *ConcertSched*, **3-dom** and **3-dom(*)** are usually the best, and $\%_t$ is at most 83.78%. As for *MaxCut*, **4-dom(*)** is the best for large instances, which reduce 22.36% time than **4-dom** in *MaxCut*-50. *PCBoard* has the most to gain from CAE, which demands for longer dominance

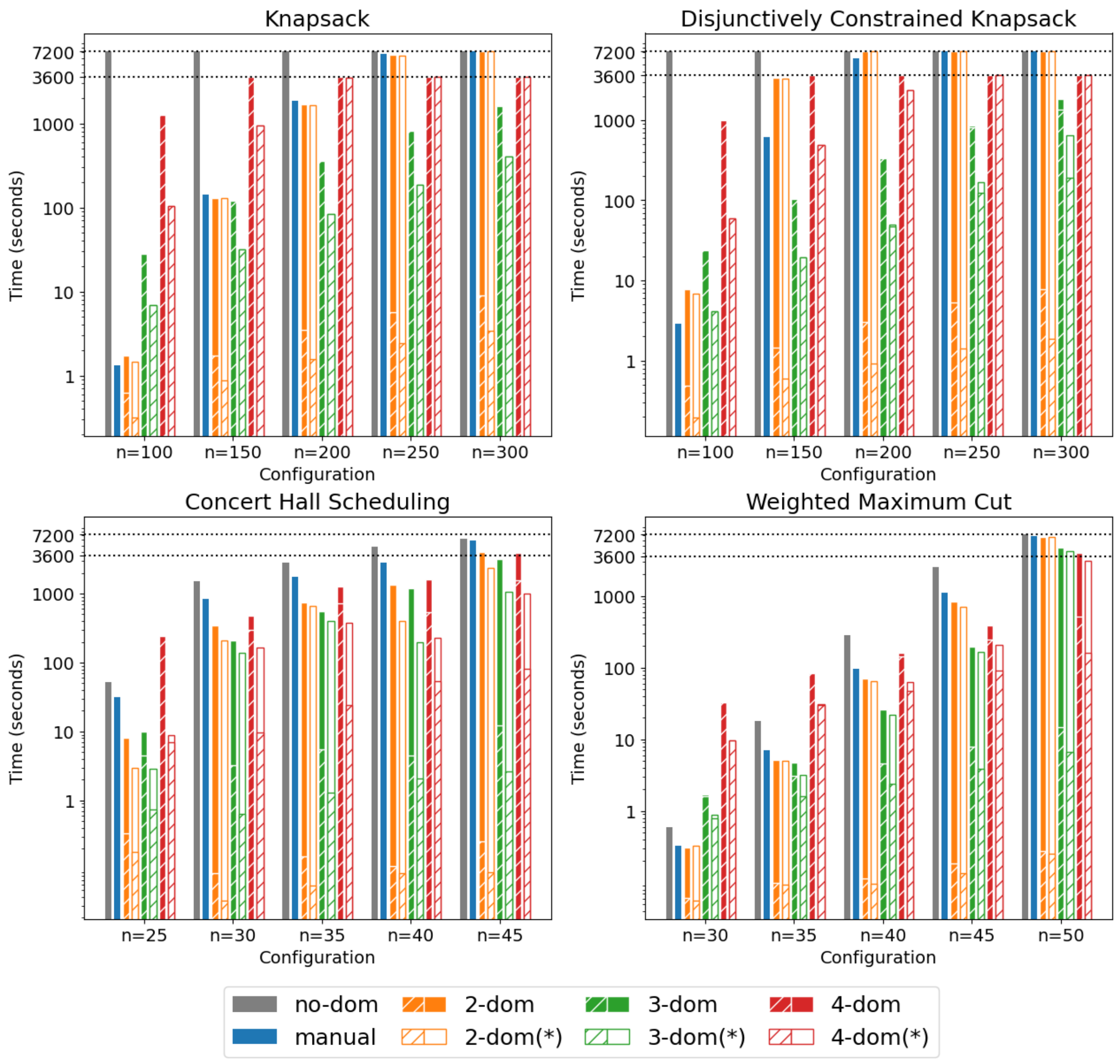


Figure 5.4: *Comparisons of the total time for* Knapsack, DisjKnapsack, ConcertSched *and* MaxCut

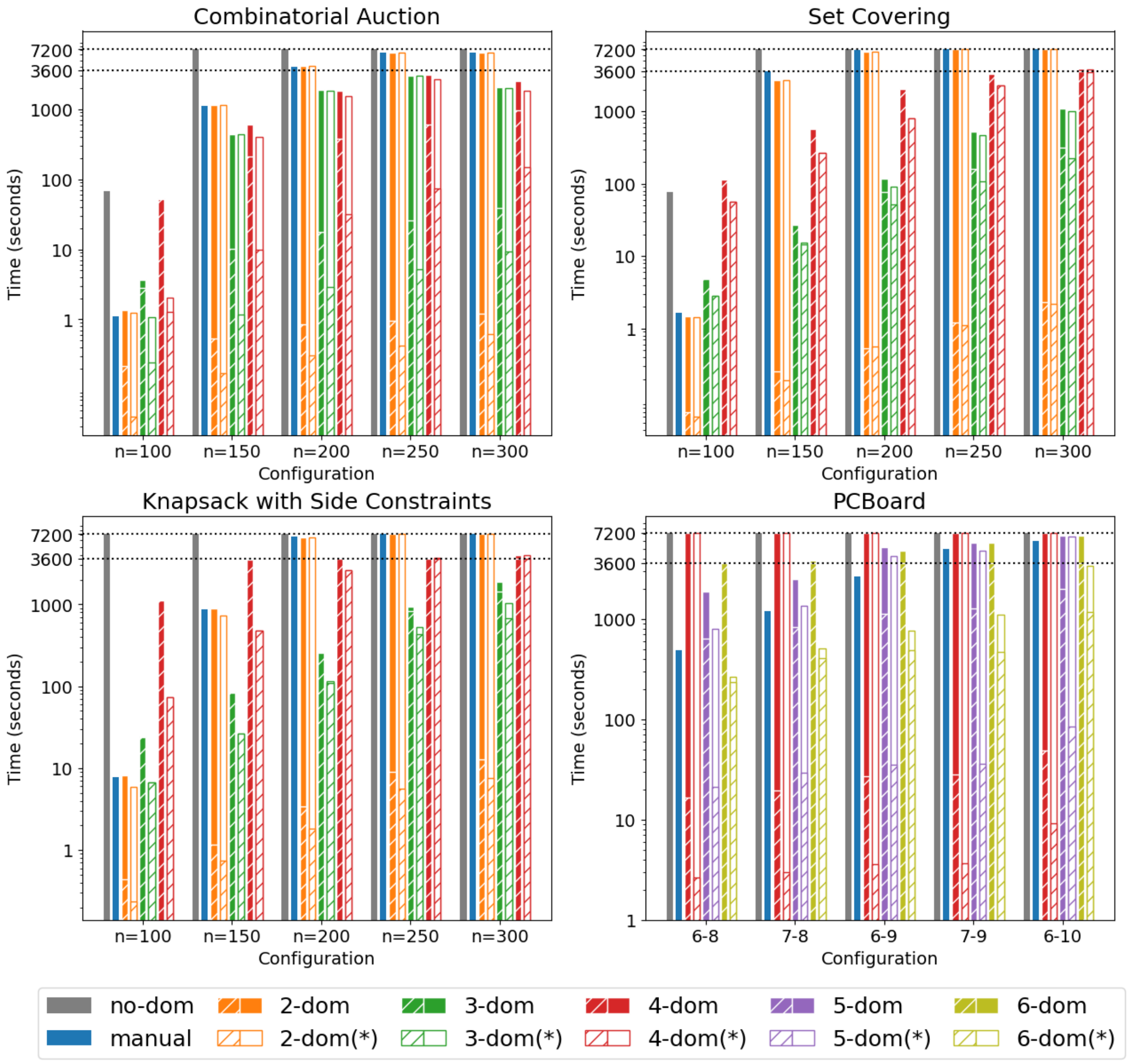


Figure 5.5: *Comparisons of the total time for* CombAuc, SetCover, KnapsackSide *and* PCBoard

breaking nogoods. The method **5-dom** is usually the best when CAE is not applied, but its overall runtimes in all configurations are still than the solving tiem of **manual**. After applying the technique, **6-dom(*)** comes out on top, which reduces from **5-dom** by 3213.74 seconds and **6-dom** by 3777.22 seconds on average. The percentage decrease of **6-dom(*)** can be as much as 92.85% compared with **6-dom**.

Note that the automatically generated dominance breaking nogoods are always stronger in search space reduction when L is large. If we compare L**-dom** and L**-dom(*)**, the problem-solving times are usually the same, which is consistent with Proposition 1. The only exceptions are large instances of *Knapsack*, *DisjKnapsack* and *KnapsackSide*. For these problems, both **4-dom** and **4-dom(*)** cannot finish the process of nogood generation completely within one hour, but **4-dom(*)** generates more nogoods than **4-dom**. In *Knapsack* and *KnapsackSide*, the problem-solving time slightly increases due to the overhead introduced by extra nogoods, while the benefits from the reduction of search space wins over the overhead in *DisjKnapsack*.

5.4 Concluding Remarks

In this chapter, we enlarge the class of problems that can benefit from the method of automatic dominance breaking. Our specific contributions are threefold. First, we remove a restriction of the original method for automatic dominance breaking that all constraints must be efficiently checkable, and demonstrate that the method generates enough nogoods to speed up the solving of problems with small-scope non-EC side constraints. Second, we present the technique called common assignment elimination to avoid the generation of unnecessary nogoods and yet maintain the pruning strength of the generated nogoods. Third, our theory is backed by extensive empirical evaluation, and the results show that the two proposed innovations not only make automatic dominance breaking applicable to more constraint optimization problems, but also substantially improve the overall efficiency.

Chapter 6

Exploiting Functional Constraints in Nogood Generation

Chapters 4 presents the method of automatic dominance breaking for a class of COPs, which can identify dominance breaking constraints that are stronger than the manual ones. Yet, the method of automatic dominance breaking is restricted to COPs with only objectives and constraints that are all known to be *efficiently checkable*. For example, in order to apply automatic dominance breaking to a COP, the objective is required to be either a separable function or a submodular function. This prevents the use of automatic dominance breaking for COPs with varying objectives and constraints, especially the ones with nested function calls.

Functional expressions are ubiquitous in problem modeling, while the objective and constraints with functional expressions are usually not efficiently checkable. In practice, however, COPs are usually specified in a high-level modeling language [40, 100] and normalized/flattened into a form with only standard constraints.

Example 18. *Consider a simple COP which minimizes the objective* $\max(z_1, z_2) + 4z_3$ *subject to the constraint* $2z_1 + 3z_2 * z_3 \leq 5$*, where* $z_1, z_2, z_3 \in \{1, 2, 3\}$*. The objective with the* max *function and the constraint with the multiplying function are not efficiently checkable. After normalization,* *the COP can become:*

$$\begin{aligned} &\textit{minimize } obj \\ &\textit{subject to } obj = y_1 + 4z_3, y_1 = \max(z_1, z_2), \\ &\qquad y_2 \leq 5, y_2 = 2z_1 - 3y_3, y_3 = z_2 * z_3, \\ &\qquad z_1, z_2, z_3 \in \{1, 2, 3\}, y_1, y_2, y_3, obj \in \mathbb{Z} \end{aligned} \tag{6.1}$$

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

In this chapter, we propose to exploit functional constraints to identify useful dominance relations in COPs with nested function calls. We first generalize the theory of dominance to normalized COPs which contain functionally defined variables and functional constraints. We present a rewriting system for automatic derivation of sufficient conditions for dominance relations in COPs based on functional constraints and their special properties such as monotonicity, commutativity and associativity, and submodularity. The proposed method is implemented on top of the MiniZinc compiler [100]. Experimentation on various benchmarks confirms the superior efficiency of the generated nogoods to solve problems with ineffective or no known dominance breaking constraints in the literature.

6.1 Functional Constraints and Dominance

In this chapter, we assume that *a COP $P = (X, D, C, obj)$ is the result of applying some sort of flattening procedure*, such as the one used in the MiniZinc compiler [88] and similar to that shown in Example 18, to a problem model. Therefore, we have a set C_Y of functional constraints, each defining a variable $y \in Y$, and a set of non-functional constraints. Our proposed method utilizes the functional constraints and the properties of functions to derive sufficient conditions for betterment and implied satisfaction as shown in the following example.

Example 19. *Consider the COP in (6.1) and $\theta, \theta' \in \mathcal{D}^S$ where $S = \{z_1, z_2\}$. Our aim is to find sufficient conditions over θ and θ' that imply all full assignments in $\mathcal{D}^X_{\theta'}$ can be removed. Suppose we only consider full assignments that satisfy all functional constraints in (6.1). For each full assignment $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$, we let $\bar{\theta} \in \mathcal{D}^X_{\theta}$ denote a corresponding full assignment such that $\bar{\theta}[z_3] = \bar{\theta}'[z_3]$. By Theorem 3, if we have (a) betterment: $\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \bar{\theta}[obj] \leq \bar{\theta}'[obj]$, (b) implied satisfaction: $\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \bar{\theta}[y_2] \leq \bar{\theta}'[y_2]$, and (c) $\theta \neq \theta'$, then $\bar{\theta}'$ is dominated by $\bar{\theta}$ and hence can be removed. We can find sufficient conditions for betterment as follows:*

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

$$\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \bar{\theta}[y_1] + 4\bar{\theta}[z_3] \leq \bar{\theta}'[y_1] + 4\bar{\theta}'[z_3], \tag{6.2}$$

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

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

$$\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \max(\bar{\theta}[z_1], \bar{\theta}[z_2]) + \bar{\theta}[z_3] \leq \max(\bar{\theta}'[z_1], \bar{\theta}'[z_2]) + \bar{\theta}'[z_3]. \tag{6.3}$$

- *The summation function is monotonically increasing, (6.3) must be true if we have*

$$(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \max(\bar{\theta}[z_1], \bar{\theta}[z_2]) \leq \max(\bar{\theta}'[z_1], \bar{\theta}'[z_2])) \land (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \bar{\theta}[z_3] \leq \bar{\theta}'[z_3]) \tag{6.4}$$

- *Since $\bar{\theta}[z_3] \leq \bar{\theta}'[z_3]$ for all $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$, the second inequality $\bar{\theta}[z_3] \leq \bar{\theta}'[z_3]$ in (6.4) must hold. We also have $\bar{\theta}[z_1] = \theta[z_1]$, $\bar{\theta}[z_2] = \theta[z_2]$, $\bar{\theta}'[z_1] = \theta'[z_1]$, and $\bar{\theta}'[z_2] = \theta'[z_2]$, and therefore the condition (6.4) is equivalent to*

$$\max(\theta[z_1], \theta[z_2]) \leq \max(\theta'[z_1], \theta'[z_2]). \tag{6.5}$$

  Thus, if θ and θ' satisfy (6.5), the betterment condition must hold.

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

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

$$(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, 2\bar{\theta}[z_1] \leq 2\bar{\theta}'[z_1]) \land (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, 3\bar{\theta}[y_3] \geq 3\bar{\theta}'[y_3]) \tag{6.6}$$

- *Variable y_3 is defined by $y_3 = z_2 * z_3$. The value for z_2, z_3 must be greater than 0. Therefore, $3\bar{\theta}[y_3] \geq 3\bar{\theta}'[y_3]$ must hold if*

$$(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \bar{\theta}[z_2] \geq \bar{\theta}'[z_2]) \wedge (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \bar{\theta}[z_3] \geq \bar{\theta}'[z_3]). \tag{6.7}$$

  Since $\bar{\theta}[z_3] = \bar{\theta}'[z_3]$ for all $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$, the latter condition must hold.

- *By definitions, we have $\bar{\theta}[z_1] = \theta[z_1]$, $\bar{\theta}[z_2] = \theta[z_2]$, $\bar{\theta}'[z_1] = \theta'[z_1]$, and $\bar{\theta}'[z_2] = \theta'[z_2]$, and therefore (6.6) and (6.7) must hold if*

$$\theta[z_1] \leq \theta'[z_1] \wedge \theta[z_2] \geq \theta'[z_2] \tag{6.8}$$

In other words, if θ and θ' fulfill (6.5) and (6.8), then θ and θ' satisfy the betterment and implied satisfaction conditions. By Theorem 15, we can add the lexicographic ordering constraint $(\theta[z_1], \theta[z_2]) <_{lex} (\theta'[z_1], \theta'[z_2])$ to ensure the compatibility of generated nogoods. One possible solution of the generation CSP is the pair (θ, θ') where $\theta = \{z_1 = 1, z_2 = 2\}$ and $\theta' = \{z_1 = 2, z_2 = 1\}$, and the constraint $\neg\theta' \equiv (z_1 \neq 2 \vee z_2 \neq 1)$ is a dominance breaking nogood for the COP in (6.1).

Example 19 describe the derivation process for assignments over one scope, and similar derivation can also be applied to assignments over other scopes to obtain more dominance breaking nogoods. In the following of this chapter, we automate the derivation process in Example 19 to derive sufficient conditions over θ and θ' so that $\neg\theta'$ is a dominance breaking nogood in a COP. The high-level algorithm for nogood generation is as follows:

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

The key step is to synthesize a generation CSP for each scope considering the functional constraints. Note that the high-level algorithm for nogood generation is different from that in Chapter 4, where one generation CSP is constructed for all scopes of the same size.

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

Without loss of generality, let (Z, Y, B) and (C_B, C_Y) be a partition of variables X and constraints C respectively in a normalized COP, where $Z \cup Y \cup B = X$, $C_B \cup C_Y = C$ and $obj \in Y$. Note that Z, Y, B are pairwise disjoint and $C_B \cap C_Y = \emptyset$. In case a variable $y \in Y$ is introduced by the flattening procedure, we set the domain for y to be the largest possible set, and therefore, a constraint $c_y \in C_Y$ must be satisfied if the value of $y \in Y$ is computed from the assignments over variables $x_{i_1}, \ldots, x_{i_k}$. Note that when there is no flattening and reification, our definition of a COP degenerates to the classical definition [113].

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

Definition 5. *Let $P = (X, D, C, obj)$ be a normalized COP where (Z, Y, B) and (C_B, C_Y) be a partition of variables X and constraints C respectively. A full assignment $\bar{\theta} \in \mathcal{D}^X$ is* functionally valid *if and only if*

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

When $\bar{\theta}$ in a normalized COP is functionally valid, it corresponds to a full assignment in the original non-flattened COP. The value for a variable $y \in Y$ (respectively $b \in B$) in a functionally valid full assignment can be computed from $c_y \in C_Y$ (respectively $c_b \in C_B$)

as well as values for variables in Z. Therefore, we can focus on finding pairs of partial assignments θ and θ' over $S \subseteq Z$ such that $\theta \prec \theta'$ with respect to P.

In the remainder of this chapter, we assume that $P = (X, D, C, obj)$ is a normalized COP and consider only functionally valid full assignments in $\mathcal{D}^X$. Recall that $\theta \prec \theta'$ requires $\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \exists \bar{\theta} \in \mathcal{D}^X_{\theta}$ such that $\bar{\theta} \prec \bar{\theta}'$ for some dominance relation over $\mathcal{D}^X$. It is expensive to check whether *there exists* $\bar{\theta}$ that dominates $\bar{\theta}'$ for each $\bar{\theta}'$ in $\mathcal{D}^X_{\theta'}$. Instead, we check only whether a specific $\bar{\theta}$ dominates $\bar{\theta}'$ by utilizing a *mutation mapping* for two assignments θ and θ' over the same scope. The mutation mapping in Definition 4 is generalized to functionally valid full assignments in a normalized COP as follows.

Definition 6. *The* mutation mapping $\mu^{\theta' \to \theta}$ *for two assignments $\theta, \theta' \in \mathcal{D}^S$ over a scope $S \subseteq Z$ maps a full assignment $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$ to another full assignment $\bar{\theta} \in \mathcal{D}^X_{\theta}$ such that:*

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

In other words, $\mu^{\theta' \to \theta}$ "mutates" the θ' component of $\bar{\theta}'$ to become θ and assigns the computed values to variables in $Y \cup B$ accordingly. When it is clear from the context, we let $\bar{\theta} = \mu^{\theta' \to \theta}(\bar{\theta}')$ denote the image of $\bar{\theta}'$ by the mutation mapping. The following proposition characterizes some useful properties of the mutation mapping in Definition 6.

Proposition 16. *Let $\mu^{\theta' \to \theta}$ be a mutation mapping for two assignments $\theta, \theta' \in \mathcal{D}^S$ over a scope $S \subseteq Z$. The followings are always true for all $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$:*

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

The following result gives a sufficient condition governing when a partial assignment θ dominates another θ' with respect to P.

Theorem 26. *If a pair of assignments $\theta, \theta' \in \mathcal{D}^S$ satisfies:*

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

then θ dominates θ' with respect to P.

The proof is essentially the same as that of Theorem 3, which is to construct a dominance relation over $\mathcal{D}^X$ such that all full assignments in $\mathcal{D}^X_{\theta'}$ are dominated. Again, Theorems 2 and 26 imply that $\neg\theta'$ is a *dominance breaking nogood* to remove all dominated solutions in $\mathcal{D}^X_{\theta'}$. To further ensure that all generated nogoods are *compatible* in the sense that not all optimal solutions of P are eliminated, a lexicographic ordering constraint $\theta <_{lex} \theta'$ can be added as stated in Theorem 15.

Empty intersection in Theorem 26 is trivially satisfied if $\theta \neq \theta'$. In order to show that a partial assignment θ' is dominated by another θ using Theorem 26, we also need to find constraints over θ and θ' that are sufficient conditions for the betterment and the implied satisfaction conditions. In the next section, we will show how to derive such sufficient conditions in an automatic manner. Note that the above definitions and results degenerate to those in Chapter 4 when Y and C_Y are empty.

6.2 Automatic Sufficient Condition Derivation

Observe that the betterment and implied satisfaction conditions in Theorem 26 are both predicates requiring an inequality to hold for all $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$, and we formulate the derivation process as a rewriting system. We first present a general rewriting system (Definition 7) which only utilizes functional constraints and the property in Definition 5 to find sufficient conditions. While this rewriting system is generic, the derived sufficient conditions are sometimes too restricted. We discuss more rewriting rules that exploit common functional properties such as monotonicity (Definition 8), associativity and commutativity (Definition 9 and 10) as well as submodularity (Definition 11) to derive more relaxed sufficient conditions, and hence more useful nogoods can be generated for dominance breaking.

6.2.1 A General Rewriting System

To formalize the derivation of sufficient conditions, we adopt the inductive definition of *terms* [4] from rewriting systems as follows:

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

By abusing notations, we define $var(t) = \{x\}$ when $t \equiv x$, and $var(t) = \cup_{i=1}^{k} var(t_i)$ when $t \equiv f(t_1, \ldots, t_k)$. Note that f can either be the constraint c in a reified constraint $b = c(x_{i_1}, \ldots, x_{i_k})$ or the function h in a functional constraint $y = h(x_{i_1}, \ldots, x_{i_k})$. A term t is *fixed in* θ if and only if $var(t) \subseteq var(\theta)$; otherwise t is *free*.

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

Let $\lhd$ be a binary comparison operator in $\{\leq, \geq, =\}$, and we say that the *reverse operators* of $\leq$, $\geq$ and $=$ are $\geq$, $\leq$ and $=$ respectively. The betterment and the implied satisfaction conditions in Theorem 26 are all predicates in the form of quantified inequalities, i.e., $(\forall \bar{\theta}' \in \mathcal{D}_{\theta'}^{X}, t\bar{\theta} \lhd t\bar{\theta}')$, where $t\bar{\theta}$ and $t\bar{\theta}'$ are obtained by substituting each variable in $var(t)$ with its values in $\bar{\theta}$ and $\bar{\theta}'$ respectively. The derivation in Example 19 recursively rewrites a quantified inequality until the term t is fixed in θ and θ'. Formally, the rewriting system maintains two sets of predicates (Q, F) where Q is the set of predicates that have to be further rewritten, and F is the set of predicates involved only variables in S. We write $(Q \cup \{p\}, F) \rightsquigarrow (Q \cup Q', F \cup F')$ when a predicate p is transformed into a set Q' of quantified inequalities and a set F' of conditions over θ and θ' by a rewriting rule. Note that Q' and F'

are sometimes empty.

Definition 7. *Given a normalized COP $P = (Z \cup Y \cup B, D, C_B \cup C_Y, obj)$ and a scope $S \subseteq Z$. The general rewriting system is initialized with the pair $(Q, \{\})$, where $Q = \{(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \bar{\theta}[obj] \leq \bar{\theta}'[obj])\} \cup \{(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \bar{\theta}[b] \geq \bar{\theta}'[b]) \mid b \in B\}$. When there is a predicate $(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, t\bar{\theta} \lhd t\bar{\theta}')$ in the system, the system applies the following rewriting rules:*

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

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

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

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

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

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

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

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

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

Note that rewriting rules in Definition 7 are conditional, which depends on the set $var(t)$ of variables. The conditions are *mutually exclusive* and *exhaustive*, which means that given a predicate in Q, there must be one applicable rewriting rule. In particular, when we apply the general decomposition rule, $var(t)$ is a subset of Z and $var(t)$ has non-empty intersection with both S and $Z \setminus S$, and the term t must be a function term of the form $f(t_1, \ldots, t_k)$ which has at least two variables in $var(t)$. For simplicity, let $Q \wedge F$ denote the conjunction of all predicates in Q and F. The following theorem states an important property of the rewriting system in Definition 7.

Theorem 27. *The rewriting system in Definition 7 preserves the invariant that $Q \wedge F$ is always a sufficient condition for the betterment and the implied satisfaction of P.*

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

- Replacement: the predicate $\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, t\bar{\theta} \lhd t\bar{\theta}'$ is equivalent to $\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, (t\beta)\bar{\theta} \lhd (t\beta)\bar{\theta}'$, because all full assignments are all functionally valid by Definition 5.
- Binding: by Proposition 16, since all variables in $var(t)$ also belongs to $S \subseteq Z$, we have $\bar{\theta}[x] = \theta[x]$ and $\bar{\theta}'[x] = \theta'[x]$ for all $x \in var(t)$. The predicate $(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, t\bar{\theta} \lhd t\bar{\theta}')$ is equivalent to $(t\theta \lhd t\theta')$.
- Deletion: by Proposition 16 again, when $x \in Z$ and $x \notin S$, we have $\bar{\theta}[x] = \bar{\theta}'[x]$. Therefore, the predicate $t\bar{\theta} = t\bar{\theta}'$ must hold and imply that $t\bar{\theta} \leq t\bar{\theta}'$ and $t\bar{\theta} \geq t\bar{\theta}'$.
- General decomposition: the conjunction $\wedge_{i=1}^{k}(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, t_i\bar{\theta} = t_i\bar{\theta}')$ implies $(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, f(t_1\bar{\theta}, \dots, t_k\bar{\theta}) \lhd f(t_1\bar{\theta}', \dots, t_k\bar{\theta}'))$ by Definition 5, since $f(t_1\bar{\theta}', \dots, t_k\bar{\theta}')$ is a functional or reified constraint.

Therefore, the invariant is preserved by the rewriting system in Definition 7. $\square$

Note that $(Q, F) \rightsquigarrow (Q', F')$ by replacement, binding and deletion are equivalent transformations such that $(Q \wedge F) \equiv (Q' \wedge F')$, while $(Q' \wedge F')$ implies $(Q \wedge F)$ after applying general decomposition. By Theorem 27, the execution of the rewriting system can recursively derive the sufficient condition for a predicate in Q until either the predicate becomes a trivial statement or it is transformed into constraints over the pair (θ, θ') due to the mutation mapping. What remains is to show the termination of rewriting.

Theorem 28. *The rewriting system in Definition 7 always terminates, and Q must be empty when the rewriting system terminates.*

Proof. Without loss of generality, we assume that each variable $y \in Y$ appears only in at most one constraint other than the functional constraint $y = h(x_{i_1}, \ldots, x_{i_k})$. By definition of a COP, $Y \cup B$ and $C_Y \cup C_B$ are finite sets. We maintain three natural numbers:

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

We claim that applying each rewriting rule reduces the triple (n_1, n_2, n_3) in a lexicographic sense. Each variable $x \in Y \cup B$ is only substituted when x is in the flattened constraint or the reified constraint, replacement must decrease n_1 by 1. General decomposition decreases n_2 while keeping n_1 unchanged. Further, binding and deletion remove one predicate $(\forall\bar{\theta}' \in \mathcal{D}^X_{\theta'}, t\bar{\theta} \lhd t\bar{\theta}')$ from Q and therefore decrease n_3 by $var(t)$. The termination follows directly from the fact that there is no infinite descending sequence of triples of natural numbers. Rewriting rules in Definition 7 are exhaustive, and therefore Q must be empty when no rules are applicable and the rewriting system terminates. $\square$

The following corollary is a direct consequence of Theorems 27 and 28.

Corollary 5. *The rewriting system in Definition 7 always terminates, and the conjunction of predicates in F is a sufficient condition for the betterment and the implied satisfaction conditions of P upon termination of the system.*

In addition to the soundness property stated in Corollary 5, we can also show that the rewriting system is confluent, i.e. the ordering in which the rewriting rules are applied does not affect the eventual result.

Theorem 29. *The rewriting system in Definition 7 is confluent.*

Proof. It suffices to show that the rewriting system has the diamond property [4], i.e., if $(Q, F) \rightsquigarrow (Q', F')$ and $(Q, F) \rightsquigarrow (Q'', F'')$ by rewriting rules in Definition 7, then there exists (Q''', F''') such that $(Q', F') \rightsquigarrow (Q''', F''')$ and $(Q'', F'') \rightsquigarrow (Q''', F''')$. Note that conditions of the four rules in the rewriting system are mutually exclusive and exhaustive. Therefore, if $(Q', F') \neq (Q'', F'')$, they must be the results of rewriting different predicates p and p' in Q. Therefore, (Q''', F''') can be obtained by rewriting $p' \in Q'$ and rewriting $p \in Q''$. $\square$

The rewriting system in Definition 7 considers that the function f is general without any properties when applying general decomposition, but this may result in too restrictive sufficient conditions. For example, if we apply the rewriting system to the COP in (6.1), the resulting sufficient conditions for the betterment and the implied satisfaction will be $\theta[z_1] = \theta'[z_1]$ and $\theta[z_2] = \theta'[z_2]$, which is in conflict with the empty intersection condition in Theorem 26. No solution can be found by solving such a generation CSP and no nogoods can be generated. Therefore, we want more relaxed sufficient conditions as far as possible.

6.2.2 Decomposition Rules Using Function Properties

In this section, we give a rewriting system that returns more relaxed sufficient conditions for the betterment and implied satisfaction conditions. Recall that all rewriting rules in Definition 7 are equivalent transformations, except for the general decomposition rule. The idea is to apply different decomposition rules based on properties of functions in functional and reified constraints. We say that a predicate p_1 is *weaker than* another predicate p_2 if and only if $p_2 \Rightarrow p_1$, and p_2 is *stronger than* p_1. The weaker the sufficient conditions in generation CSPs, the more pairs of assignments will satisfy all the conditions and the more nogoods can be found by Theorem 26.

Algorithm 3 gives the rewriting system that exploits functions with special properties, which takes a normalized COP P and a scope S as inputs and returns a set F of predicates for the construction of generation CSPs. Similar to the general rewriting system in Definition 7, Algorithm 3 applies replacement (Line 6), binding (Line 8), deletion (Line 10) and general decomposition (Line 16). The difference is that a different decomposition rule is applied to a function f when it is in a pre-defined library L of functions with special properties. It is straightforward to see that the new rewriting system is still confluent.

Theorem 30. *The rewriting system in Algorithm 3 is confluent.*

Algorithm 3 A rewriting system for deriving sufficient conditions

```
    Input: a normalized COP $P = (Z \cup Y \cup B, D, C_B \cup C_Y, obj)$, a scope $S \subseteq Z$
    Output: a set $F$ of predicates over $\theta, \theta' \in \mathcal{D}^S$
 1: $Q \leftarrow \{(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \bar{\theta}[obj] \leq \bar{\theta}'[obj])\} \cup \{(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \bar{\theta}[b] \geq \bar{\theta}'[b]) \mid b \in B\}$, $F \leftarrow \emptyset$
 2: while $Q \neq \emptyset$ do
 3:     Remove a predicate $p \equiv (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, t\bar{\theta} \lhd t\bar{\theta}')$ from $Q$
 4:     if $var(t) \cap (Y \cup B) \neq \emptyset$ then
 5:         Let $x \in var(t) \cap (Y \cup B)$ be a variable defined by $x = f(x_{i_1}, \ldots, x_{i_k})$
 6:         $Q \leftarrow Q \cup \{(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, (t\beta)\bar{\theta} \lhd (t\beta)\bar{\theta}')\}$ where $\beta \equiv \{x / f(x_{i_1}, \ldots, x_{i_k})\}$
 7:     else if $var(t) \subseteq S \subseteq Z$ then
 8:         $F \leftarrow F \cup \{(t\theta \lhd t\theta')\}$
 9:     else if $var(t) \subseteq (Z \setminus S)$ then
10:         Continue
11:     else
12:         Let $p$ be $(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, f(t_1\bar{\theta}, \ldots, t_k\bar{\theta}) \lhd f(t_1\bar{\theta}', \ldots, t_k\bar{\theta}'))$
13:         if $f$ is in a pre-defined library $L$ of functions then
14:             $(Q, F) \leftarrow (Q \cup Q', F \cup F')$ where $Q'$ and $F'$ are from the decomposition of $p$
15:         else
16:             $Q \leftarrow Q \cup \{(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, t_i\bar{\theta} = t_i\bar{\theta}') \mid \forall i = 1, \ldots, k\}$
17:         end if
18:     end if
19: end while
```

Proof. The new system still possesses the diamond property as the stated in Theorem 29. □

The most important property of Algorithm 3 is the soundness as that in Corollary 5. If there are no restriction on the decomposition rules for functions in the library L, Algorithm 3 may not terminate or return a set of predicates that are sufficient conditions for the betterment and the implied satisfaction condition. Therefore, we need to characterize the validity of desired decomposition rules.

A decomposition rule rewrites a predicate p into a set Q' of quantified inequalities and a set F' of conditions over θ and θ', and we say that it is *sound* if $Q' \wedge F'$ is a sufficient condition of p. We also say that it is *diminishing* if the number of function symbols in Q' must be decrease while there are no new variables introduced in predicates of Q' after rewriting.

Theorem 31. *If all decomposition rules for functions in the library L are sound and diminishing, then the rewriting system in Algorithm 3 always terminates, and the conjunction of predicates in F*

is a sufficient condition for the betterment and implied satisfaction conditions of P upon termination.

Proof. Since all decomposition rules for functions in L are sound and diminishing, the proof for Theorems 27 and 28 are still valid, which implies the soundness of Algorithm 3. $\square$

Theorem 31 implies that the rewriting system in Algorithm 3 is extensible with sound and diminishing decomposition rules. In the following, we give several decomposition rules catering for special function properties. In particular, we highlight the aggregation functions which usually possess useful properties such as monotonicity, associativity, and commutativity for deriving weaker sufficient conditions. The cut function in a graph is also given as an example to demonstrate that the rewriting system can cooperate with an ad-hoc decomposition rule for a specific functional constraint. Note that when several decomposition rules are applicable to a predicate, the principle is to apply the one that returns the weakest possible sufficient conditions.

Decomposition for Aggregation Functions

Aggregation functions [57], such as summation, maximum, and minimum, combine multiple values into a single representative value, and they are common in modeling COPs. We describe several rewriting rules to derive weaker sufficient conditions by exploiting the properties of aggregation functions.

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

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

and is *monotonically decreasing* if

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

where $a_i, b_i \in \mathbb{R}$. When the function f is monotonically increasing or monotonically decreasing, we have the following rewriting rules.

Definition 8. *Suppose* $p \equiv (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, f(t_1\bar{\theta}, \ldots, t_k\bar{\theta}) \lhd f(t_1\bar{\theta}', \ldots, t_k\bar{\theta}')) \in Q$ *is a predicate in the rewriting system.*

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

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

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

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

We can show that both the increasing decomposition and the decreasing decomposition rules have the desired properties in Theorem 31.

Theorem 32. *The increasing decomposition and the decreasing decomposition rules in Definition 8 are sound and diminishing.*

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

What is more, applying the increasing decomposition and the decreasing decomposition rules can obtain weaker sufficient conditions.

Theorem 33. *Suppose* $p \equiv (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, f(t_1\bar{\theta}, \ldots, t_k\bar{\theta}) \lhd f(t_1\bar{\theta}', \ldots, t_k\bar{\theta}'))$ *is a predicate in the rewriting system. If* $(Q \cup \{p\}, F) \rightsquigarrow (Q \cup Q', F)$ *by the increasing decomposition (respectively the decreasing decomposition) and* $(Q \cup \{p\}, F) \rightsquigarrow (Q \cup Q'', F)$ *by the general decomposition, then the conjunction of predicates in* Q' *is weaker than that in* Q''*.*

Proof. The result follows the fact that all full assignments are functionally valid, and each predicate $(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, t_i\bar{\theta} = t_i\bar{\theta}')$ is always a sufficient condition for both $(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, t_i\bar{\theta} \lhd t_i\bar{\theta}')$ and $(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, t_i\bar{\theta} \rhd t_i\bar{\theta}')$. □

By Theorem 33, the general decomposition should always be replaced by increasing decomposition or decreasing decomposition whenever f is monotonically increasing or monotonically decreasing.

Other common and useful properties of aggregation functions are associativity and commutativity. We can show that decomposition rules in Definitions 7 and 8 can obtain even weaker sufficient conditions by using these two properties. Note that an aggregation function f can take an arbitrary non-zero number of arguments. To facilitate the presentation, we use a special notation to denote it. Let $\mathbf{t} = \langle t_1, \ldots, t_k \rangle$, $\mathbf{t}_1 = \langle t_1, \ldots, t_j \rangle$ and $\mathbf{t}_2 = \langle t_{j+1}, \ldots, t_k \rangle$ be vectors of terms, where $1 \le j \le k$. Using these notations, the followings denote the same function call: $f(t_1, \ldots, t_k)$, $f(\mathbf{t})$ and $f(\mathbf{t}_1, \mathbf{t}_2)$. Aggregation functions usually possess the following two properties:

- *Commutativity*: $f(t_1, \ldots, t_k) = f(t_{\pi(1)}, \ldots, t_{\pi(k)})$ where π is a permutation over $\{1, \ldots, k\}$.
- *Associativity*: $f(t) = t$ and $f(\mathbf{t}_1, \mathbf{t}_2) = f(f(\mathbf{t}_1), \mathbf{t}_2)$.

A permutation over a set is a bijection from the set to itself. By commutativity, we can always find a permutation for arguments of $f(t_1, \ldots, t_k)$ so that all fixed terms are clustered.

Proposition 17. *Let f be a commutative function and $\theta \in \mathcal{D}^S$ be an assignment where $S \subseteq Z$. If there are $j \ge 1$ fixed terms among $t_1, \ldots, t_k$, then we can always find a permutation π over $\{1, \ldots, k\}$ such that*

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

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

The proof directly follows the definition of commutativity. The following gives a rewriting rule in replacement of the general decomposition rule in Definition 7 for a commutative and associative function.

Definition 9. *Suppose $p \equiv (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, f(t_1\bar{\theta}, \ldots, t_k\bar{\theta}) \lhd f(t_1\bar{\theta}', \ldots, t_k\bar{\theta}'))$ is a predicate in the rewriting system such that there are $j \ge 1$ fixed terms among $t_1, \ldots, t_k$. Let π be a permutation over $\{1, \ldots, k\}$ such that $\mathbf{t}_1 = \langle t_{\pi(1)}, \ldots, t_{\pi(j)} \rangle$ and $\mathbf{t}_2 = \langle t_{\pi(j+1)}, \ldots, t_{\pi(k)} \rangle$ consists of all fixed terms and free terms respectively.*

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

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

We can show that the new rule can be added to the rewriting system in Algorithm 3.

Theorem 34. *The general decomposition with aggregation rule is sound and diminishing.*

Proof. It is straightforward to see that the rule is diminishing, while the soundness is proved as follows:

$$
\begin{aligned}
&(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, f(t_1\bar{\theta}, \ldots, t_k\bar{\theta}) \lhd f(t_1\bar{\theta}', \ldots, t_k\bar{\theta}')) \\
\Leftrightarrow &(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, f(\mathbf{t}_1\bar{\theta}, \mathbf{t}_2\bar{\theta}) \lhd f(\mathbf{t}_1\bar{\theta}', \mathbf{t}_2\bar{\theta}')) \\
\Leftrightarrow &(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, f(f(\mathbf{t_1})\bar{\theta}, \mathbf{t}_2\bar{\theta}) \lhd f(f(\mathbf{t_1})\bar{\theta}', \mathbf{t}_2\bar{\theta}')) \\
\Leftarrow &(\bigwedge_{i=j+1}^{k} (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, t_{\pi(i)}\bar{\theta} = t_{\pi(i)}\bar{\theta})) \wedge (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, f(\mathbf{t_1})\bar{\theta} = f(\mathbf{t_1})\bar{\theta}') \\
\Leftrightarrow &(\bigwedge_{i=j+1}^{k} (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, t_{\pi(i)}\bar{\theta} = t_{\pi(i)}\bar{\theta})) \wedge (f(\mathbf{t_1})\theta = f(\mathbf{t_1})\theta')
\end{aligned}
$$

The second and the third steps are due to the commutativity and the associativity of f respectively, while the fourth step holds because all full assignments are functionally valid. By Proposition 16, the last step holds because $\bar{\theta}[x] = \theta[x]$ and $\bar{\theta}'[x] = \theta'[x]$ for all $\bar{\theta}' \in \mathcal{D}^X_{\theta'}$ and $\bar{\theta} = \mu^{\theta' \to \theta}(\bar{\theta}')$. □

The advantage of the rule in Definition 9 is that the derived sufficient conditions are weaker compared with those from the general decomposition rule in Definition 7.

Theorem 35. *Suppose $p \equiv (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, f(t_1\bar{\theta}, \ldots, t_k\bar{\theta}) \lhd f(t_1\bar{\theta}', \ldots, t_k\bar{\theta}'))$ is a predicate in the rewriting system. If $(Q \cup \{p\}, F) \rightsquigarrow (Q \cup Q', F \cup F')$ by the general decomposition with aggregation and $(Q \cup \{p\}, F) \rightsquigarrow (Q \cup Q'', F)$ by the general decomposition, then the conjunction of predicates in $Q' \wedge F'$ is weaker than that for Q''.*

Proof. After applying the general decomposition rule, the conjunction of predicates in Q'' is

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

where π is as defined in Definition 9. The third step holds because all full assignments are functionally valid, and the last step is by Proposition 16. □

We can also define similar decomposition rules for a monotonic, commutative and associative function.

Definition 10. *Suppose $p \equiv (\forall\bar{\theta}' \in \mathcal{D}^X_{\theta'}, f(t_1\bar{\theta}, \ldots, t_k\bar{\theta}) \lhd f(t_1\bar{\theta}', \ldots, t_k\bar{\theta}'))$ is a predicate in the rewriting system such that there are $j \geq 1$ fixed terms among $t_1, \ldots, t_k$. Let π be a permutation over $\{1, \ldots, k\}$ such that $\mathbf{t}_1 = \langle t_{\pi(1)}, \ldots, t_{\pi(j)}\rangle$ and $\mathbf{t}_2 = \langle t_{\pi(j+1)}, \ldots, t_{\pi(k)}\rangle$ consists of all fixed terms and free terms respectively.*

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

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

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

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

The rules are diminishing and sound, and can derive weaker sufficient conditions than those in Definition 8. The proofs are similar to those for Theorems 34 and 35.

We illustrate the advantages of aggregation using the following example.

Example 20. *Suppose we want to find sufficient conditions for θ and θ' where $var(\theta) = var(\theta') = \{z_1, z_3\}$. Let $(Q \cup \{p\}, F)$ be the pair of the rewriting system where*

$$p \equiv (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \min(\bar{\theta}[z_1], \bar{\theta}[z_2], \bar{\theta}[z_3]) \leq \min(\bar{\theta}'[z_1], \bar{\theta}'[z_2], \bar{\theta}'[z_3])). \tag{6.9}$$

If we apply increasing decomposition directly to (6.9), we get

$$(Q \cup \{(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \bar{\theta}[z_i] \leq \bar{\theta}'[z_i]) \mid i = 1, 2, 3\}, F) \tag{6.10}$$

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

$$(Q \cup \{\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \min(\min(\bar{\theta}[z_1], \bar{\theta}[z_3]), \bar{\theta}[z_2]) \leq \min(\min(\bar{\theta}'[z_1], \bar{\theta}'[z_3]), \bar{\theta}'[z_2])\}, F) \tag{6.11}$$

by applying aggregation to (6.9). Using the increasing decomposition rule, we get

$$(Q \cup \{(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \min(\bar{\theta}[z_1], \bar{\theta}[z_3]) \leq \min(\bar{\theta}'[z_1], \bar{\theta}'[z_3])), (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, \bar{\theta}[z_2] \leq \bar{\theta}'[z_2])\}, F) \tag{6.12}$$

Note that after applying binding and deletion in Definition 7 to (6.10) and (6.12) respectively, we obtain $\theta[z_1] \leq \theta'[z_1] \wedge \theta[z_3] \leq \theta'[z_3]$ and $\min(\theta[z_1], \theta[z_3]) \leq \min(\theta'[z_1], \theta'[z_3])$ as sufficient conditions for (6.9), and the former condition is stronger than the latter one.

Table 6.1 summarizes common constraints in a normalized COP and their properties that can be used for deriving weaker sufficient conditions. For each type of standard constraint, we give the functionally defined variable, arguments, its monotonicity properties (“Mono."), and whether the function commutative and associative (“Com. & Asso."). To facilitate the generation of nogoods, we can compile a global constraint into a conjunction of standard constraints in Table 6.1 when constructing generation CSPs.

Example 21. *The alldifferent$(z_1, \ldots, z_k)$ constraint [115] is a global constraint that enforces a set of variables taking distinct values. To apply the rewriting system, the constraint is first reified into $b_0 \leftrightarrow$ alldifferent$(z_1, \ldots, z_k)$. Let $\tilde{D} \subseteq \bigcup_{j=1}^{k} D(z_j)$ be the set of values that appear in the domains of at least two variables in $\{z_1, \ldots, z_k\}$ and $d = |\tilde{D}|$. We can compile the constraint into a set of canonical constraints:*

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

Table 6.1: *Common constraints for detecting dominance relations*

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

where $y_{0v_i}, y_{jv_i}, b_{jv_i}$ *and* b_v *are introduced variables defined by canonical functional constraints that enjoy the properties of monotonicity, commutativity and associativity. Therefore, the decomposition rules in Definitions 8 and 9 can be applied to derive sufficient conditions for the implied satisfaction condition of the alldifferent constraint.*

The idea can be applied similarly to support other global constraints like the global cardinality constraint [116, 104] and the bin packing constraint [123]. Note that global constraints are treated as a conjunction of canonical constraints only in synthesizing generation CSPs, and are untouched in problem solving.

Ad-hoc Decomposition Rules

While aggregation functions are useful in deriving sufficient conditions, decomposition rules in the last section by no means exhaust all possible properties that are helpful in the derivation. In this section, we give an example of the cut function to demonstrate how the rewriting system can be extended with ad-hoc decomposition rules.

Let $G = (U, E)$ be a graph and $(V, U \setminus V)$ be a partition of V. The *cut* of such a partition in graph G is $\sum_{u \in V, v \in (U \setminus V)} w_{uv}$ where $w_{uv} \in \mathbb{R}$ is the weight of an edge $(u, v) \in E$. Following Section 4.2.2, a cut can be modelled in a binary COP, where there is one variable x_u with $D(x_u) = \{0, 1\}$ for each vertex $u \in U$. A partial/full assignment θ can represent a partition $(V, U \setminus V)$ where $V = \{v \mid (x_v = 1) \in \theta\}$, and the cut value $f(\theta)$ of the partition is equal to $\sum_{u \in V, v \in (U \setminus V)} w_{uv}$. We have the following decomposition rule for a cut function.

Definition 11. *Suppose $p \equiv (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, f(t_1\bar{\theta}, \ldots, t_k\bar{\theta}) \geq f(t_1\bar{\theta}', \ldots, t_k\bar{\theta}'))$ is a predicate in the rewriting system such that there are $j \geq 1$ fixed terms among $t_1, \ldots, t_k$. Let π be a permutation over $\{1, k\}$ such that $\mathbf{t}_1 = \langle t_{\pi(1)}, \ldots, t_{\pi(j)} \rangle$ and $\mathbf{t}_2 = \langle t_{\pi(j+1)}, \ldots, t_{\pi(k)} \rangle$ consists of all fixed terms and free terms respectively.*

- *Cut Decomposition: if f is a cut function, then $(Q \cup \{p\}, F)$ is rewritten into*

$$(Q \cup \{(\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, t_i\bar{\theta} = t_i\bar{\theta}) \mid t_i \in \mathbf{t}_2\}, F \cup \{t_i\theta \leq t_i\theta' \mid t_i \in \mathbf{t}_1\} \cup \{f(\theta) \geq f(\theta')\}),$$

  where $\mathbf{t}_1 = \langle t_{\pi(1)}, \ldots, t_{\pi(j)} \rangle$ and $\mathbf{t}_2 = \langle t_{\pi(j+1)}, \ldots, t_{\pi(k)} \rangle$ is a partition of $\{t_1, \ldots, t_k\}$ such that $\mathbf{t}_1$ consists of all fixed terms, π is a permutation of $\{1, \ldots, k\}$, and $1 \leq j \leq k$.

Similar to Theorem 7, it can be show that the conjunction of the derived predicates by the decomposition rule is a sufficient condition of the rewritten predicate due to the submodular property of the cut function.

Proposition 18. *The cut decomposition rule is sound and diminishing.*

Also, it is straightforward to see that the derived predicates by the cut decomposition are weaker than the general decomposition in Definition 7.

Proposition 19. *Let $p \equiv (\forall \bar{\theta}' \in \mathcal{D}^X_{\theta'}, f(t_1\bar{\theta}, \dots, t_k\bar{\theta}) \lhd f(t_1\bar{\theta}', \dots, t_k\bar{\theta}')) \in Q$ be a predicate where $\lhd$ is an operator in $\{\leq, \geq, =\}$. If $(Q \cup \{p\}, F) \rightsquigarrow (Q \cup Q', F)$ by the general decomposition and $(Q \cup \{p\}, F) \rightsquigarrow (Q \cup Q', F \cup F')$ by the cut decomposition, then $Q'' \wedge F''$ is weaker than that of Q'.*

The proof of Proposition 19 is similar to that of Theorem 33.

Note that a submodular function can also be monotone. For example, a linear constraint $y = \sum_{x_i \in S} \bar{\theta}[x_i]$ is both submodular and monotonically increasing, and we can decompose the functional constraint by either the increasing decomposition with aggregation in Definition 10 or the cut decomposition in Definition 11. Recall that our principle is to obtain the weakest possible sufficient conditions, and therefore we would apply the former rule for a linear constraint.

6.3 Experimental Evaluation

In this section, we give experimental results to show the utility of our proposal. All experiments are run on Xeon E5-2630 2.60GHz processors. We use MiniZinc [100] as the high-level modeling language and implement our nogood generation method by modifying[1] the publicly available MiniZinc compiler with version 2.6.2. In a compiled model, we treat constraints with the annotation "`defines_var`" as functional constraints, while others are non-functional constraints that should be reified. The generated nogoods for each problem are output as text and then appended to the MiniZinc model of the corresponding problem. The augmented models are submitted to MiniZinc for solving using the Chuffed solver [103] with version 0.10.4. Note that our method aims to analyze a user-defined model, not necessarily that of the best model, and we specify the search strategies for all problems to demonstrate the effect of the additional dominance breaking nogoods in search space pruning. We use six benchmark problems, 20 random instances for each problem size. The models for the following three problems are from public benchmark suites:

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

- *Talent-n*. The Talent Scheduling Problem [19] is problem 039 in CSPLib [54]. Each actor appears in several scenes and is paid a fixed cost per day if they are present. They need to be present on location from the first scene they are in till the last scene they are in. We need to schedule n scenes to minimize the total cost for a set of actors. The dominance breaking constraints for **manual** are by Chu and Stuckey [23].

- *Warehouse-n*. The Warehouse Location Problem [127] is problem 034 in CSPLib [54]. We need to choose a subset of possible warehouses in different locations to supply a set of n existing customers such that the sum of building costs for warehouses and supply costs for customers is minimized.

- *Team-n-m*. The Team Assignment Problem appears in MiniZinc Challenge 2018 [124]. The problem consists of $n \times m$ players, where players have different ratings and need to be assigned to n teams. There are requests regarding which pair of players want to be in the same team. The objective is to satisfy as many requests as possible while balancing the total rating among all teams.

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

- *MaxCover-n*. The Budgeted Maximum Coverage Problem [69] is a variant of the set cover problem. There is a ground set U and a collection T consisting of n subsets of U, where each subset is associated with a cost c_i. The goal is to find a subset of T such that the union covers the maximum number of elements subject to the constraint that the total cost does not exceed a given budget. The search strategy is to select the unfixed subset in T with the smallest cost first.

- *PartialCover-n*. The Partial Set Cover Problem [68] is another variant of the set cover problem. Given a ground set U and a collection T consisting of n subsets of U, the goal is to find a subset of T with the minimum total cost, whose union covers at least K elements in U. The search strategy is also to select the subset with the smallest cost first.

- *Sensor-n*. The Sensor Placement Problem [73] is a variant of the facility location problem [26], where we need to select a fixed cardinality subset of n locations to place sensors in order to provide service for customers. If we place a sensor at location i, then it provides service to a subset of reachable customers, and each customer will choose the facility with the highest service value from the opened sensors, and the goal is to maximize the total service value. The search strategy is to select the unfixed location with the highest service value to the set of customers that are reachable by the sensor placed at the location.

Note that the original method in Chapter 4 can handle none of the benchmarks effectively because of nested function calls in either the objective or constraints. For all benchmarks, we attempt to generate all dominance breaking nogoods of length up to L (L-**dom**), and compare our method to the basic problem model (**basic**) and the model with manual dominance breaking constraints (**manual**) whenever they are available. The timeout for the whole solving process (nogood generation + problem solving) is set to 7200 seconds, while we reserve at most 3600 seconds for nogood generation and use the *remaining* time for problem solving in L-**dom**. If nogood generation times out, we augment the problem model with all nogoods generated before the timeout.

In Table 6.2, we report the geometric mean of the problem solving time (Solving) and the total time (Total) for all benchmarks, where "N/A" in the **manual** column indicates that there are no known dominance breaking constraints for the problem in the literature. We first compare the problem solving time of L-**dom** against **basic** to evaluate the usefulness of the generated nogoods, and we observe that the generated dominance breaking nogoods can significantly reduce the solving time in all benchmarks. As the maximum nogood length L increases and more generated nogoods are added to the problem model, the solving time is usually shorter except for *Talent-n*. We note that the solving time of 4-**dom** is larger than that of 3-**dom** for *Talent-n* due to the overhead caused by a large amount of generated nogoods of length 4.

We also compare the total time (generation time + solving time) of L-**dom** against **basic**

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

Table 6.2: *Comparison of the solving time and the total time for various benchmarks*

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

We note that the solving time of **manual** is even larger than that of **basic** in *Talent-n*. Expressing manual dominance breaking for *Talent-n* in the MiniZinc model requires additional variables and introduces overheads for propagation. Chu and Stuckey [23] implement the manual dominance breaking constraints in Chuffed, which requires sophisticated and bespoke techniques to reduce the overhead. The generated nogoods by our method only involve variables in the original model, and they can be posted in the high-level modeling language without modifying the backend solver.

Appendix A.3 gives more experimental results on evaluating the effects of dominance breaking nogoods using different search configurations.

6.4 Discovering Dominance Relations

Our method, which is based on that of Chapters 4, attempts to generate all dominance breaking nogoods before problem solving, and sometimes the number of nogoods is so large that generating all nogoods will cost too much time for each problem instance. We observe that nogoods are the most basic units of constraints. Every high-level constraint can be decomposed into a group of nogoods, and vice versa. By examining the patterns of the generated nogoods, we could discover the embedded high-level dominance breaking constraints. We give two case studies in this section.

The first case study is the Still Mill Slab Design Problem [64], which is problem 039 in CSPLib [54]. The problem is to assign colored production orders with different sizes to slabs where each slab has a finite number of possible sizes. The total size of orders assigned to a slab cannot exceed the chosen slab size, and each slab cannot contain orders with more than 2 colors. The loss of each slab is the difference between the chosen slab size and the total size of orders assigned to the slab, and the objective is to minimize the total loss of all slabs.

Previous works study different classes of symmetries, one of which is order symmetries [42], that is, two orders with identical size and color are equivalent. We apply our method to generate nogood of length 2 for the model from MiniZinc Challenge 2017 [124], which introduces one variable x_i to specify the slab that orders i is assigned to. The generation always times out within 3600 seconds, and the overhead always outweighs the benefit. Although a single nogood means relatively little, a bunch of them together can derive a meaningful constraint collectively. We investigate the semantics of nogoods and discover a new class of symmetries. By generating the nogoods of length 2, we observe that we can

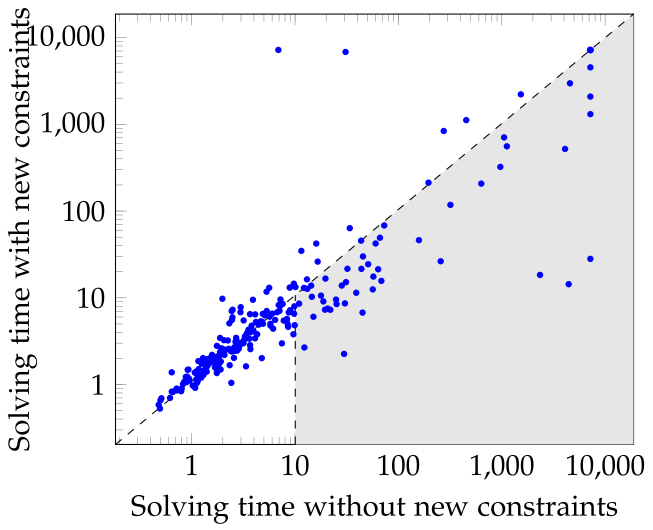


Figure 6.1: *Comparison of the solving time with/without new symmetry breaking constraints in steel mill slab design problem*

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

We take a constraint model of the steel mill slab design problem from a public benchmark suite[2] and augment it with constraints to break the newly discovered symmetry relationship. Figure 6.1 shows the solving time for all 380 instances from the steel mill slab library[3], and the dots below the diagonal line represent the instances benefiting from the newly discovered constraints. We observe that the solving time is reduced in the majority of cases, especially more so when the solving time of the original model requires more than

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

[3]http://becool.info.ucl.ac.be/steelmillslab

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

Table 6.3: *Comparison of the solving time for the balance academic curriculum problem*

10 seconds. The hard instances are represented by dots in the shaded region in Figure 6.1. Note that both axes are in log scale, and the speed-up of new constraints is up to two orders of magnitude. Several outliers require substantially more solving time after adding the new symmetry breaking constraints. This is due to the conflict between the search heuristic and the static symmetry breaking constraints [47]. We believe the conflicts can be mitigated by using dynamic symmetry breaking methods such as SBDS [53] or SBDD [30].

The other case study is the Balanced Academic Curriculum Problem [18], which is problem 030 in CSPLib [54]. There are n courses each associated with several credits representing the effort required to complete the course, and courses need to be assigned to academic periods subject to the course prerequisite constraints. The workload of each period is the sum of all credits of courses that are assigned to the period. The objective is to minimize the maximum academic load for all periods to balance the loads among academic periods.

We perform experiments using the same experimental setting as that in Section 6.3 and report the results for problems with different course numbers in Table 6.3. The dominance breaking constraints for **manual** are by Monette, Jean-Noël et al. [97]. In general, the problem-solving time of our method is smaller than that of **basic** but larger than that of **manual**. The overhead of L**-dom** mainly comes from the generation of dominance breaking nogoods before solving the COP. In addition, the dominance breaking constraints in **manual** are in the form of inequalities, which can be handled more efficiently than nogoods added by L**-dom** in a propagation-based constraint solver. Nevertheless, by analyzing the nogoods of a small instance, we find that the generated nogoods of length 2 can also be combined into inequality constraints similar to the case of the steel mill slab design problem. The inequality constraints we consider are the same as those proposed by Monette, Jean-Noël et al. [97], which shows that our method can also reveal dominance breaking constraints written by experts in the literature.

6.5 Concluding Remarks

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

Chapter 7

Strength of Dominance Breaking Nogoods

In this chapter, we study the logical strength of the generated dominance breaking nogoods. In general, we can always generate dominance breaking nogoods that are stronger than manually derived dominance breaking constraints if the length of nogoods is sufficiently large. If we set the scope of θ and θ' to X, then the generated nogood constraints eliminate all suboptimal full assignments in a COP.

We are interested in the relative strength of generated nogoods with a limited length compared with manual dominance breaking constraints given in the literature. 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 the conjunctions are *logically equivalent* if they imply each other. We show in various problems that the set of generated nogoods of a certain length are logically equivalent or even stronger than manual dominance breaking constraints given in the literature.

7.1 0-1 Knapsack Problems

Recall from the last section that 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:

$$\text{maximize } \sum_{i=1}^{n} p_i x_i \tag{7.1a}$$

$$\text{subject to } \sum_{i=1}^{n} w_i x_i \leq W \tag{7.1b}$$

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 [24] propose the following dominance breaking constraints.

Definition 12. *[24] 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 12.

Theorem 36. *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 12.*

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 5, 9 and 16, (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 (7.1b): $w_i v_i + w_j v_j \leq w_i v_i' + w_j v_j'$

- compatibility: $(-(p_iv_i+p_jv_j), w_iv_i+w_jv_j, v_i, v_j) <_{lex} (-(p_iv'_i+p_jv'_j), w_iv'_i+w_jv'_j, v'_i, v'_j)$

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 12 is equivalent to the set of generated nogoods. □

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 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 12.*

7.2 Disjunctively Constrained Knapsack Problems

The disjunctively constrained knapsack problem [130] 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.

$$(x_i = 0) \vee (x_j = 0), \forall j \in \Gamma(i), i \in \{1, \ldots, n\} \tag{7.2}$$

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 13. *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 $l = \max_{j\in N}(|\Gamma(j)|) + 2$ is logically stronger than manual dominance breaking constraints.

Theorem 37. *When the nogood length is $l = \max_{j\in N}(|\Gamma(j)|) + 2$, the set of automatically generated dominance breaking nogoods is logically stronger than the set of manual dominance breaking constraints in Definition 13.*

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 13. 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 5, 9 and 16, which are similar to those in the proof of Theorem 36. In addition, (θ, θ') must also satisfy

$$((\theta[x_j] = 0) \vee (\theta[x_k] = 0)) \Rightarrow ((\theta'[x_j] = 0) \vee (\theta'[x_k] = 0)) \tag{7.3}$$

by Theorem 11. One possible solution is to have $\theta[x_k] = \theta'[x_k] = 0$. Following the same reasoning as Theorem 36, 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 13. $\square$

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 (7.2) 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 9, and it is a stronger condition than (7.3). Following the same reasoning as the proof of Theorem 36, we will find the generation CSP becomes insatisfiable, and thus there are no dominance breaking nogoods with length $l = 3$.

7.3 Capacitated Concert Hall Scheduling Problems

The *Capacitated Concert Hall Scheduling Problems* [45] 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 can be modelled as follows:

$$\text{maximize} \sum_{i \in A} p_i \mathbf{1}_{>0}(x_i) \tag{7.4a}$$

$$\text{subject to } \textit{alldifferent\_except\_0}(\{x_j \mid j \in \Gamma(i)\}), \forall i \in A \tag{7.4b}$$

$$x_i \in H_i \cup \{0\}, \forall i \in A \tag{7.4c}$$

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 [45] propose constraints to prefer shorter, more profitable concerts.

Definition 14. *[45] 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 38. *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 14.*

Proof. Note that each constraint in Definition 14 is equivalent to the conjunction of several nogood constraints:

$$\begin{aligned} & x_i > 0 \rightarrow x_j > 0 \\ \Leftrightarrow & \neg(x_i > 0) \vee (x_j > 0) \\ \Leftrightarrow & x_i = 0 \vee x_j \neq 0 \\ \Leftrightarrow & \bigwedge_{h \in H_i} (x_i \neq h \vee x_j \neq 0) \end{aligned} \tag{7.5}$$

It suffices to show that the set of nogoods in (7.5) is logically equivalent to a subset of generated dominance breaking nogoods when $l = 2$.

Consider a pair (θ, θ') of partial assignments over a scope $S = \{x_i, x_j\}$, where $i < j$, $\theta[x_i] = \theta'[x_j] = 0$ and $\theta[x_j] = \theta'[x_i] = h \in H_j$. The pair (θ, θ') is a solution of the generation CSP if it satisfies sufficient conditions from Theorems 5 and 16, and Corollary 2:

- 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 (7.4b) and (7.4c)
  - $\forall k \in A, \{\theta[x_j] \mid j \in \Gamma(k)\} \subseteq \{\theta'[x_i] \mid i \in \Gamma(k)\}$,
  - $\forall k \in A, \theta[x_j] \in H_j \cup \{0\}$
- 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 (7.4b) 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 (7.4c) 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 (7.5) and is logically equivalent to the set of manual dominance breaking constraints in Definition 14. □

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 7. *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 14.*

7.4 Weighted Maximum Cut Problems

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 [117], we define the following manual dominance breaking constraints.

Definition 15. *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 15.

Theorem 39. *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 12.*

Proof. By Corollary 7 and Theorem 16, 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: $g(V(\theta)) \geq g(V(\theta')) \wedge V(\theta) \subseteq V(\theta')$
- compatibility: $(-g(V(\theta)), \theta[x_i], \theta[x_j]) <_{lex} (-g(V(\theta'), \theta'[x_i], \theta'[x_j])$

where $V(\theta) = \{k \mid x_k \in S \wedge \theta[x_k] = 1\}$ and $V(\theta') = \{k \mid x_k \in S \wedge \theta'[x_k] = 1\}$. Since $D(x_i) = D(x_j) = \{0, 1\}$, we exhaust all value combinations and find that there are only two possible valid solutions:

- $\theta = \{x_i = 1, x_j = 0\}$ and $\theta' = \{x_i = 1, x_j = 1\}$, or
- $\theta = \{x_i = 0, x_j = 1\}$ and $\theta' = \{x_i = 1, x_j = 1\}$.

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 15. □

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 8. *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 15.*

In general, we can observe that in a weighted undirected graph $G = (V, E)$, if moving a set of vertices out of a set $V(\theta)$ for a partial assignments will increase the cut value, then θ cannot be extended to an optimal solution of the maximum cut problem on G.

7.5 Combinatorial Auction Problems

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:

$$\text{maximize} \sum_{i=1}^{n} p_i x_i \tag{7.6a}$$

$$\text{subject to} \sum_{i \in \Gamma(k)}^{n} x_i \leq 1, \forall k \in \{1, \ldots, m\} \tag{7.6b}$$

Following the method by Chu and Stuckey [24], we consider dominance breaking constraint derived from mappings that swap the values of variables x_i and x_j.

Definition 16. *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 40. *The set of automatically generated dominance breaking nogoods of length $l = 2$ is logically stronger to the set of manual dominance breaking constraints in Definition 16.*

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 $i < j$. If $B_i \supseteq B_j$, then $j \in \Gamma(k)$ implies that $i \in \Gamma(k)$. By Theorems 5, 9 and 16, (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 (7.6b):
  - $\forall k \in \{1, \dots, m\}, v_j \leq v'_j$ when $i \in \Gamma(k)$ and $j \notin \Gamma(k)$, or
  - $\forall k \in \{1, \dots, m\}, v_i + v_j \leq v'_i + v'_j$ 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 16. □

7.6 Set Covering Problems

The set covering problem [59] is to select a collection of subsets whose union equals to the universe $\{1, \dots, m\}$, 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:

$$\text{minimize } \sum_{i=1}^{n} c_i x_i \tag{7.7a}$$

$$\text{subject to } \sum_{i \in \Gamma(k)}^{n} x_i \geq 1, \forall k \in \{1, \dots, m\} \tag{7.7b}$$

Similar to Definition 16, we can derive the dominance breaking constraints by the method by Chu and Stuckey [24].

Definition 17. *The set of manual dominance breaking constraints for the set covering problems are $x_i \leq x_j$ for all $i, j \in \{1, \dots, n\}$ when $B_i \subseteq B_j$ and either (1) $c_i > c_j$, or (2) $c_i = c_j \wedge i < j$.*

We can show that the generated nogoods constraints of length 2 is logically stronger to the manually defined dominance breaking constraints.

Theorem 41. *The set of automatically generated dominance breaking nogoods of length $l = 2$ is logically stronger to the set of manual dominance breaking constraints in Definition 17.*

The proof is similar to that of Theorem 40.

7.7 PC Board Problems

The PC Board Problems [91] is to assign components of various types to $|M|$ machines with n slots on each machine. There are n_t components of type $t \in T$, and $\sum_{t \in T} n_t = |M|n$. At most one component of type t can be assigned to the same machine. If a component of type t is assigned to a machine $m \in M$, then it obtains the profit p_{mt}. The incompatibility constraint requires that two components of conflicting types cannot be assigned to the same machine. Similar to the disjunctively constrained knapsack problem, the conflicts can be represented by a graph $G = (T, E)$ where $(t, t') \in E$ if components of type t and type t' are in conflict with each other.

To model the problem, we use one binary variable $x_{mt} \in \{0, 1\}$ for each machine $m \in M$ and each component type $t \in T$, indicating whether there is a component of type t on machine m. The model is as follows:

$$\text{maximize} \quad \sum_{m \in M, t \in T} x_{mt} p_{mt} \tag{7.8a}$$

$$\text{subject to} \quad (x_{mt} = 0) \vee (x_{mt'} = 0), \forall m \in M, t \in T, t' \in \Gamma(t) \tag{7.8b}$$

$$\sum_{m \in M} x_{mt} = n_t, \forall t \in T \tag{7.8c}$$

$$\sum_{t \in T} x_{mt} = n, \forall m \in M \tag{7.8d}$$

where $\Gamma(t) = \{t' \mid (t, t') \in E\}$ is the *neighborhood* of t in the conflict graph G.

Chu and Stuckey [24] consider swapping two components of different types on two different machines such that either the utility increases or remains the same after the swapping. If these two component can be swapped without violating the incompatibility constraints on both machines, then we can post dominance breaking constraints to enforce the swapping.

Definition 18. *[24] The set of manual dominance breaking constraints for the PC board problems are*

$$\begin{aligned}(\bigwedge_{\bar{t} \in \Gamma(t) \setminus \{t'\}} (x_{m\bar{t}} = 0) \wedge \bigwedge_{\bar{t} \in \Gamma(t') \setminus \{t\}} (x_{m'\bar{t}} = 0)) \\ \rightarrow (x_{mt} \neq 0 \vee x_{mt'} \neq 1 \vee x_{m't} \neq 1 \vee x_{m't'} \neq 0)\end{aligned} \tag{7.9}$$

for all machines $m, m' \in M$ and types $t, t' \in T$ where either

1. *$p_{mt} + p_{m't'} > p_{m't} + p_{mt'}$, or*

2. *$p_{mt} + p_{m't'} = p_{m't} + p_{mt'}$ and $(m, t) <_{lex} (m', t')$*

The precondition in the constraint is to ensure that no components in conflict with t and t' are assigned to machine m and m' respectively after swapping.

We first consider a simplified case where no pairs of component types are conflicting and show that the set of generated nogoods is stronger when $l = 4$.

Lemma 2. *Suppose $\forall t \in T, \Gamma(t) = \emptyset$. The set of automatically generated dominance breaking nogoods is logically equivalent to the set of manual dominance breaking constraints in Definition 18* *when the length of nogood is $l = 4$.*

Proof. Without loss of generality, we assume that $m < m'$. Since $\forall t \in T, \Gamma(t) = \emptyset$, constraint (7.9) becomes $(x_{mt} \neq 0 \vee x_{mt'} \neq 1 \vee x_{m't} \neq 1 \vee x_{m't'} \neq 0)$, which is in the form of a nogood constraint. We consider a pair (θ, θ') of partial assignments over the scope $S = \{x_{mt}, x_{mt'}, x_{m't}, x_{m't'}\}$. The variables in θ and θ' only involve in four constraints: (1) $\sum_{m \in M} x_{mt} = n_t$ for component type t and t', and (2) $\sum_{t \in T} x_{mt} = n$ for machine m and m'. By Theorems 5, 9 and 16, the generation CSP requires:

- betterment: $\sum_{x_{ij} \in S} \theta[x_{ij}] p_{ij} \geq \sum_{x_{ij} \in S} \theta'[x_{ij}] p_{ij}$
- implied satisfaction for (7.8c):
  - $\theta[x_{mt}] + \theta[x_{m't}] = \theta'[x_{mt}] + \theta'[x_{m't}]$, and
  - $\theta[x_{mt'}] + \theta[x_{m't'}] = \theta'[x_{mt'}] + \theta'[x_{m't'}]$,
- implied satisfaction for (7.8d):
  - $\theta[x_{mt}] + \theta[x_{mt'}] = \theta'[x_{m't}] + \theta'[x_{m't'}]$, and
  - $\theta[x_{mt}] + \theta[x_{mt'}] = \theta'[x_{m't}] + \theta'[x_{m't'}]$
- compatibility: $(-\sum_{x_{ij} \in S} \theta[x_{ij}] p_{ij}, \theta) <_{lex} (-\sum_{x_{ij} \in S} \theta'[x_{ij}] p_{ij}, \theta')$

When we have

$$\theta[x_{mt}] = 1, \theta[x_{mt'}] = 0, \theta[x_{m't}] = 0, \theta[x_{m't'}] = 1,$$

$$\theta'[x_{mt}] = 0, \theta'[x_{mt'}] = 1, \theta'[x_{m't}] = 1, \theta'[x_{m't'}] = 0,$$

the pair (θ, θ') satisfies all constraints in the generation CSP under the conditions in Definition 18. Therefore, the set of generated dominance breaking nogoods is logically equivalent to the set of constraints in Definition 18. $\square$

When there are conflicting components, manual dominance breaking constraints will have the precondition that $\bigwedge_{\bar{t} \in \Gamma(t)} (x_{m\bar{t}} = 0)$ and $\bigwedge_{\bar{t} \in \Gamma(t')} (x_{m'\bar{t}} = 0)$, and we show that the manual dominance breaking constraint is equivalent to a generated dominance breaking nogoods of length at most $l = |\Gamma(t)| + |\Gamma(t')| + 4$.

Theorem 42. *Each manual dominance breaking constraint in Definition 18 is equivalent to a generated dominance breaking nogood of length $l = |\Gamma(t)| + |\Gamma(t')| + 4$.*

Proof. Without loss of generality, we assume that $\Gamma(t) = \{k\}$ and $\Gamma(t') = \{k'\}$. Constraint (7.9) is equivalent to a nogood constraint

$$(x_{mk} \neq 0 \vee x_{m'k'} \neq 0 \vee x_{mt} \neq 0 \vee x_{mt'} \neq 1 \vee x_{m't} \neq 1 \vee x_{m't'} \neq 0) \tag{7.10}$$

Similar to Lemma 2, we consider a pair (θ, θ') of partial assignments such that

$$\theta[x_{mk}] = 0, \theta[x_{m'k'}] = 0, \theta[x_{mt}] = 1, \theta[x_{mt'}] = 0, \theta[x_{m't}] = 0, \theta[x_{m't'}] = 1,$$
$$\theta'[x_{mk}] = 0, \theta'[x_{m'k'}] = 0, \theta'[x_{mt}] = 0, \theta'[x_{mt'}] = 1, \theta'[x_{m't}] = 1, \theta'[x_{m't'}] = 0,$$

Since θ and θ' have the same values for x_{mk} and $x_{m'k'}$, they fulfill sufficient conditions from Theorems 5, 9 and 16 by similar reasoning as that in Lemma 2. The additional constraints are (7.8b) for type k and k'. By Theorem 11, it is trivial to verify that the sufficient conditions for implied satisfaction are satisfied since $\theta[x_{mk}] = \theta[x_{m'k'}] = \theta'[x_{mk}] = \theta'[x_{m'k'}] = 0$. Therefore, the pair is a solution the generation CSP, and $\neg\theta'$ is equivalent to constraint (7.10). □

Since there is one generated dominance breaking nogoods of length $l = |\Gamma(t)| + |\Gamma(t')| + 4$ for each constraint in Definition 18, the set of all nogoods must be stronger than the set of manual dominance breaking constraints.

Corollary 9. *When the maximum nogood length is $L = |\Gamma(t)| + |\Gamma(t')| + 4$, the set of automatically generated dominance breaking nogoods is logically stronger than the set of manual dominance breaking constraints in Definition 18.*

7.8 Concluding Remarks

In this chapter, we provide several case studies to compare the strength of manually identified dominance breaking constraints and automatically generated nogoods by the method of automatic dominance breaking. The results show that generated nogoods are logically equivalent to or even stronger than manual constraints, which indicates that the method of automatic dominance breaking can reveal dominance breaking constraints in the literature and even identify more opportunities to exploit dominance relations. The theoretical insights also support the performance gains demonstrated by the experimental results in Chapters 4 to 6.

Chapter 8

Conclusion and Future Works

This thesis studies the automation of identification and exploitation of dominance relations in constraint optimization problems. The contributions are threefold based on our three published papers [85, 86, 87]. First, we propose the framework of automatic dominance breaking which focuses on generating nogood constraints for dominance breaking. The generation of nogoods is formulated mechanically as constraint satisfaction, and therefore dominance breaking is automated for a class of optimization problems that consist of efficiently checkable objectives and constraints. Second, we propose two theoretical and practical techniques to relax the restriction of efficiently checkable constraints and improve the efficiency of nogood generation. The proposals enable the method to be applied to more benchmark problems that cannot be handled by the original method. Finally, we further enlarge the class of problems with nested function calls, which are common in a high-level modeling language in constraint programming. We identify a set of elementary functional constraints and their useful properties such as monotonicity, associativity, commutativity and submodularity, and the construction of nogood generation problems is formulated as an automated term rewriting system. Both theoretical case studies and experimental results on benchmark problems demonstrate the ability of our proposed methods in discovering opportunities to exploit dominance relations and reducing the search space substantially for solving constraint optimization problems.

There are several future research directions. While nogood constraints are elementary in constraint programming, the propagation of nogood may not be efficient. As shown in Section 6.4, nogoods with relevant semantics can be combined into high-level constraints that can be handled more efficiently. One direction of future work is to automate the process of deriving high-level constraints by the techniques of automatic discovery of constraints from example solutions [10, 12], where the generated nogoods can be used as examples to learn and discover the desired constraints. The acquired constraints can help users to further understand the target COP and improve the efficiency of the existing models. Second, solving the generation CSP may sometimes incur a large overhead, and whether a benchmark can benefit from our method cannot be guaranteed. Our method requires a full constraint instance to synthesize generation CSPs, and the automatic detection of dominance relations from constraint models alone is an interesting line of future work. Finally, the static generation of dominance breaking nogoods may sometimes take too much time, and not all nogoods are equally important in solving COPs. We expect dynamic detection of dominance relations and its combination with dynamic dominance breaking methods [21, 24] will save time for the overall solution process.

Appendix A

Supplementary Experiment Results

A.1 Supplementary Results for Section 4.6

Table A.1 shows the average solving times and the average total times. An entry with the symbol "–" indicates that the whole solving process time out after the 2-hour limit. We highlight the fastest total time in bold for each configuration of different benchmark problems.

	basic	**manual**	**2-dom**		**3-dom**		**4-dom**	
Problem	Total	Total	Solving	Total	Solving	Total	Solving	Total
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 A.1: *Comparison of the solving time and the total time for* ***basic****,* ***manual*** *and* ***L-dom***

A.2 Supplementary Results for Section 5.3

Table A.2 shows the time in seconds for generating dominance breaking nogoods of length up to L. The columns t_g and t_g^* are the average time for generating nogoods without and with common assignment elimination respectively. An entry with the symbol "–" indicates that the generation timed out after the 1-hour limit. The *percentage decrease of generation time ($\%_g$)* for each length L is computed as follows:

$$\%_g = \frac{t_g - t_g^*}{t_g}$$

The entry "N/A" means that the decrease of generation time is not available since t_g or t_g^* exceeds the time limit of 1 hour.

Table A.3 shows the total time (generation time + solving time) in seconds for nogood generation and problem solving of models augmented with generated nogoods of length up to L. The time t_s and t_s^* are the average problem solving time without and with common assignment elimination respectively. An entry with the symbol "–" indicates that the whole solving process time out after the 2-hour limit. The *percentage decrease of total time* $\%_t$ for each length L is computed as follows:

$$\%_t = \frac{(t_s + t_g) - (t_s^* + t_g^*)}{t_s + t_g}$$

The entry "N/A" means that the decrease of total time is not available since $t_s + t_g$ or $t_s^* + t_g^*$ exceeds the time limit of 2 hours.

Problem	$L = 2$			$L = 3$			$L = 4$		
	t_g	t^*_g	$\%_g$	t_g	t^*_g	$\%_g$	t_g	t^*_g	$\%_g$
Knapsack-100	0.63	0.32	49.14%	28.08	6.83	75.67%	1272.45	103.59	91.86%
Knapsack-150	1.74	0.87	50.05%	120.83	31.39	74.02%	–	949.11	N/A
Knapsack-200	3.53	1.58	55.38%	355.14	83.02	76.62%	–	3494.99	N/A
Knapsack-250	5.64	2.41	57.27%	810.98	184.71	77.22%	–	–	N/A
Knapsack-300	8.94	3.42	61.79%	1595.79	405.14	74.61%	–	–	N/A
DisjKnapsack-100	0.48	0.20	59.25%	23.54	4.15	82.37%	991.03	59.20	94.03%
DisjKnapsack-150	1.47	0.59	59.69%	104.15	19.56	81.22%	–	488.63	N/A
DisjKnapsack-200	3.02	0.92	69.54%	333.49	47.66	85.71%	–	2377.76	N/A
DisjKnapsack-250	5.32	1.42	73.27%	808.78	125.44	84.49%	–	–	N/A
DisjKnapsack-300	7.80	1.87	76.05%	1360.17	190.35	86.01%	–	–	N/A
ConcertSched-25	0.33	0.18	45.16%	4.53	0.74	83.56%	239.73	6.92	97.11%
ConcertSched-30	0.09	0.04	59.71%	3.29	0.64	80.38%	296.20	9.77	96.70%
ConcertSched-35	0.16	0.06	62.50%	5.60	1.31	76.53%	724.18	24.06	96.68%
ConcertSched-40	0.11	0.09	19.95%	4.43	2.07	53.35%	540.29	53.98	90.01%
ConcertSched-45	0.25	0.09	63.56%	12.29	2.67	78.29%	1549.71	80.10	94.83%
MaxCut-30	0.10	0.09	5.02%	3.08	1.63	46.99%	82.44	29.82	63.83%
MaxCut-35	0.11	0.10	15.17%	4.63	2.38	48.62%	144.24	47.79	66.87%
MaxCut-40	0.18	0.14	26.50%	7.91	3.92	50.42%	249.27	90.77	63.59%
MaxCut-45	0.28	0.26	7.87%	14.83	6.64	55.20%	512.00	161.54	68.45%
MaxCut-50	0.40	0.38	4.82%	22.80	12.51	45.12%	1013.92	348.94	65.59%
CombAuc-100	0.21	0.04	80.87%	2.81	0.24	91.42%	52.41	1.26	97.59%
CombAuc-150	0.53	0.17	68.28%	10.04	1.17	88.33%	210.26	9.88	95.30%
CombAuc-200	0.84	0.31	62.98%	17.70	2.85	83.88%	377.02	31.26	91.71%
CombAuc-250	0.94	0.42	55.69%	25.91	5.26	79.71%	599.54	74.25	87.62%
CombAuc-300	1.22	0.61	49.69%	38.96	9.35	75.99%	978.75	149.02	84.77%
SetCover-100	0.07	0.06	13.36%	4.81	2.81	41.52%	113.95	55.49	51.30%
SetCover-150	0.26	0.19	24.74%	27.29	14.56	46.65%	570.99	267.33	53.18%
SetCover-200	0.54	0.56	-4.07%	77.25	52.03	32.65%	2027.83	805.53	60.28%
SetCover-250	1.19	1.11	6.77%	158.60	107.11	32.46%	3285.10	2258.66	31.25%
SetCover-300	2.34	2.17	7.18%	316.85	222.67	29.73%	–	3486.21	N/A
KnapsackSide-100	0.44	0.24	46.74%	24.10	6.63	72.49%	1118.47	72.69	93.50%
KnapsackSide-150	1.15	0.74	35.88%	83.79	26.16	68.78%	3548.08	473.34	86.66%
KnapsackSide-200	3.44	1.83	46.65%	257.54	109.29	57.57%	–	2575.88	N/A
KnapsackSide-250	8.97	5.59	37.68%	834.80	431.76	48.28%	–	–	N/A
KnapsackSide-300	12.85	7.61	40.77%	1428.66	675.02	52.75%	–	–	N/A

Problem	$L = 4$			$L = 5$			$L = 6$		
	t_g	t^*_g	$\%_g$	t_g	t^*_g	$\%_g$	t_g	t^*_g	$\%_g$
PCBoard-8-6	16.35	2.62	83.98%	633.37	21.02	96.68%	–	232.79	N/A
PCBoard-8-7	19.33	2.96	84.69%	833.90	28.92	96.53%	–	406.03	N/A
PCBoard-9-6	26.66	3.55	86.70%	1116.76	35.18	96.85%	–	487.97	N/A
PCBoard-9-7	27.83	3.66	86.83%	1258.38	35.60	97.17%	–	465.96	N/A
PCBoard-10-6	49.09	9.14	81.39%	2008.01	84.88	95.77%	–	1174.32	N/A

Table A.2: *Comparison of the time for generating nogoods of different lengths*

Problem	$L=2$			$L=3$			$L=4$		
	t_g+t_s	$t_g^*+t_s^*$	$\%_t$	t_g+t_s	$t_g^*+t_s^*$	$\%_t$	t_g+t_s	$t_g^*+t_s^*$	$\%_t$
Knapsack-100	1.74	1.47	15.37%	28.09	6.85	75.62%	1272.53	103.66	91.85%
Knapsack-150	128.29	129.88	-1.24%	120.96	31.53	73.93%	3600.54	949.71	73.62%
Knapsack-200	1675.13	1636.61	2.30%	355.41	83.31	76.56%	3600.98	3496.71	2.90%
Knapsack-250	6550.39	6466.25	1.28%	811.76	185.39	77.16%	3601.03	3601.52	-0.01%
Knapsack-300	–	–	N/A	1597.57	406.94	74.53%	3601.84	3602.41	-0.02%
DisjKnapsack-100	7.69	6.77	11.89%	23.56	4.17	82.32%	991.07	59.25	94.02%
DisjKnapsack-150	3364.16	3252.09	3.33%	104.25	19.66	81.14%	3600.28	488.90	86.42%
DisjKnapsack-200	–	–	N/A	336.32	50.33	85.03%	3600.97	2378.98	33.93%
DisjKnapsack-250	–	–	N/A	850.16	169.53	80.06%	3603.16	3602.70	0.01%
DisjKnapsack-300	–	–	N/A	1841.92	647.65	64.84%	3614.45	3608.43	0.17%
ConcertSched-25	8.09	2.93	63.74%	10.12	2.87	71.66%	244.68	8.84	96.39%
ConcertSched-30	343.25	207.24	39.62%	210.50	138.65	34.13%	481.84	163.14	66.14%
ConcertSched-35	753.36	671.00	10.93%	547.82	395.61	27.78%	1273.04	372.36	70.75%
ConcertSched-40	1339.94	398.77	70.24%	1201.01	194.79	83.78%	1622.20	226.65	86.03%
ConcertSched-45	4023.27	2357.65	41.40%	3127.77	1051.99	66.37%	3847.61	1006.10	73.85%
MaxCut-30	0.31	0.32	-5.80%	1.69	0.90	46.95%	32.85	9.64	70.65%
MaxCut-35	5.17	5.03	2.81%	4.69	3.18	32.32%	83.75	31.18	62.77%
MaxCut-40	69.93	64.07	8.38%	26.15	21.70	16.99%	160.53	62.72	60.93%
MaxCut-45	834.55	699.00	16.24%	196.21	165.86	15.47%	385.08	204.86	46.80%
MaxCut-50	6703.11	6646.30	0.85%	4680.55	4209.72	10.06%	3945.98	3063.76	22.36%
CombAuc-100	1.36	1.24	9.04%	3.64	1.06	70.79%	53.38	2.06	96.15%
CombAuc-150	1140.90	1139.02	0.16%	443.22	435.33	1.78%	600.61	402.09	33.05%
CombAuc-200	4107.52	4109.78	-0.05%	1862.48	1853.41	0.49%	1859.62	1540.64	17.15%
CombAuc-250	6486.44	6473.60	0.20%	3032.58	3044.67	-0.40%	3093.52	2695.97	12.85%
CombAuc-300	6381.51	6379.09	0.04%	2066.10	2028.17	1.84%	2535.92	1831.95	27.76%
SetCover-100	1.46	1.45	0.81%	4.82	2.82	41.43%	113.96	55.50	51.30%
SetCover-150	2669.02	2682.66	-0.51%	27.95	15.23	45.51%	571.01	267.34	53.18%
SetCover-200	6677.53	6682.50	-0.07%	116.89	91.17	22.00%	2027.89	805.59	60.27%
SetCover-250	7168.57	7168.98	-0.01%	523.38	470.22	10.16%	3347.68	2258.87	32.52%
SetCover-300	–	–	N/A	1100.08	1004.03	8.73%	3882.64	3784.07	2.54%
KnapsackSide-100	8.10	5.94	26.62%	24.17	6.68	72.35%	1118.69	72.84	93.49%
KnapsackSide-150	885.57	733.12	17.21%	84.44	26.67	68.42%	3549.26	476.08	86.59%
KnapsackSide-200	6590.88	6515.05	1.15%	263.73	115.21	56.32%	3606.12	2607.48	27.69%
KnapsackSide-250	–	–	N/A	924.84	529.69	42.73%	3686.44	3688.16	-0.05%
KnapsackSide-300	–	–	N/A	1892.80	1031.61	45.50%	3953.78	3968.15	-0.36%

Problem	$L=4$			$L=5$			$L=6$		
	t_g	t_g^*	$\%_g$	t_g	t_g^*	$\%_g$	t_g	t_g^*	$\%_g$
PCBoard-8-6	–	–	N/A	1876.87	796.20	57.58%	3693.64	264.16	92.85%
PCBoard-8-7	–	–	N/A	2489.31	1340.27	46.16%	3849.23	508.01	86.80%
PCBoard-9-6	–	–	N/A	5215.44	4224.66	19.00%	4818.17	762.27	84.18%
PCBoard-9-7	–	–	N/A	5741.28	4790.46	16.56%	5711.61	1091.41	80.89%
PCBoard-10-6	–	–	N/A	6745.53	6605.94	2.07%	6813.18	3373.89	50.48%

Table A.3: *Comparison of the total time using nogoods of different lengths*

A.3 Supplementary Results for Section 6.3

In this section, we give more experimental results for benchmarks in Section 6.3 using different search configurations of the Chuffed solver [103]:

- user: using the user-specified search heuristic with lazy clause generation.
- no-lazy: using the user-specified search heuristic without lazy clause generation.
- vsids: using the variable state independent decaying sum heuristic with lazy clause generation.
- togglee-vsids: alternating between the user-specified search heuristic and the variable state independent decaying sum heuristic with lazy clause generation.
- free: using the variable state independent decaying sum search heuristic with lazy clause generation and setting the restart base to be 100.

We use the same experiment protocol as that in Section 6.3, and all experiment are run on Quad Xeon Platinum 8268 2.90GHz processors. Table A.4 to A.9 shows the geometric mean of the problem solving time (Solving) and the total time (Total) in seconds for the six benchmarks in Section 6.3, where "N/A" in the **manual** column indicates that there no known dominance breaking constraints for the problem in the literature. For each configuration, we highlight the smallest total time in bold.

		basic	**manual**	2-**dom**		3-**dom**		4-**dom**	
Configuration	Instance	Solving	Solving	Solving	Total	Solving	Total	Solving	Total
user	$n = 16$	935.07	7200.00	891.83	892.55	458.48	**484.66**	564.79	2142.00
	$n = 18$	6441.28	7200.00	6497.04	6497.44	4790.89	**4830.68**	3252.11	6897.56
	$n = 20$	7199.83	7200.00	7199.38	7199.86	6825.22	**6880.22**	3578.53	7200.00
no-lazy	$n = 16$	649.63	7200.00	669.30	670.10	357.56	**382.74**	1671.54	3516.40
	$n = 18$	6371.08	7200.00	6445.00	6445.44	5202.48	**5235.85**	3593.63	7200.00
	$n = 20$	7199.82	7200.00	7199.37	7199.86	6818.12	**6873.73**	3578.56	7200.00
vsids	$n = 16$	213.00	6235.85	215.69	216.32	149.58	**171.36**	279.31	1785.49
	$n = 18$	1628.28	7200.00	1751.22	1751.84	781.93	**824.58**	1562.94	5414.10
	$n = 20$	5115.84	7200.00	5085.53	5086.09	2882.40	**2950.20**	3080.24	6757.51
togglee-vsids	$n = 16$	828.19	7200.00	883.59	884.27	451.94	**479.89**	565.45	2178.71
	$n = 18$	6559.02	7200.00	6549.21	6549.59	4739.58	**4781.30**	3364.35	6974.51
	$n = 20$	7199.83	7200.00	7199.41	7199.86	6865.76	**6921.27**	3589.38	7200.00
free	$n = 16$	218.93	5576.67	233.96	234.65	170.19	**192.70**	333.05	1877.21
	$n = 18$	1710.41	7189.26	1769.89	1770.68	987.76	**1029.23**	1841.96	5708.09
	$n = 20$	5181.36	7200.00	5139.87	5140.44	3880.31	**3953.56**	3215.67	6866.49

Table A.4: *Comparison of the solving time and the total time for talent scheduling problem using different search configurations*

		basic	**manual**	2-**dom**		3-**dom**		4-**dom**	
Configuration	Instance	Solving	Solving	Solving	Total	Solving	Total	Solving	Total
user	$n = 30$	7199.81	N/A	2.63	**19.11**	3.99	1090.20	4.13	3608.11
	$n = 35$	7199.86	N/A	11.07	**36.88**	17.36	2002.85	19.02	3629.15
	$n = 40$	7199.78	N/A	50.54	**109.38**	63.75	3176.24	88.41	3742.91
	$n = 45$	7199.84	N/A	100.46	**154.63**	94.43	3726.52	93.13	3722.41
	$n = 50$	7199.82	N/A	798.79	**917.73**	285.22	3973.95	285.76	3966.55
no-lazy	$n = 30$	7199.80	N/A	37.75	**61.03**	62.68	1160.10	71.23	3703.43
	$n = 35$	7199.81	N/A	233.86	**266.78**	376.43	2506.28	372.55	4130.84
	$n = 40$	7199.79	N/A	1431.55	**1491.62**	1546.52	4896.36	1663.94	5530.35
	$n = 45$	7199.78	N/A	3886.61	**3934.79**	2605.00	6264.49	2577.10	6284.13
	$n = 50$	7199.76	N/A	6893.32	**6937.53**	3599.12	7200.00	3599.15	7200.00
vsids	$n = 30$	5334.24	N/A	4.41	**21.81**	7.21	1155.07	6.83	3610.98
	$n = 35$	7199.85	N/A	19.52	**44.85**	28.75	2045.68	26.29	3634.98
	$n = 40$	4310.27	N/A	69.46	**114.28**	119.52	3201.58	106.10	3765.55
	$n = 45$	4382.41	N/A	101.26	**145.25**	138.96	3778.30	117.12	3742.90
	$n = 50$	5923.10	N/A	391.77	**465.01**	167.17	3920.71	181.86	3928.86
togglee-vsids	$n = 30$	7199.77	N/A	2.67	**18.67**	4.38	1119.19	4.49	3608.16
	$n = 35$	7199.86	N/A	10.90	**37.70**	19.92	2035.04	17.19	3628.30
	$n = 40$	7199.79	N/A	52.67	**108.06**	77.77	3147.71	76.17	3723.55
	$n = 45$	7199.74	N/A	94.43	**154.10**	99.39	3739.59	87.40	3714.64
	$n = 50$	7199.86	N/A	840.22	**944.31**	321.98	4094.81	321.57	4088.68
free	$n = 30$	**9.82**	N/A	3.60	20.74	5.28	1053.72	5.33	3609.78
	$n = 35$	**32.93**	N/A	14.18	39.68	19.81	1950.47	21.01	3629.98
	$n = 40$	103.66	N/A	40.76	**81.56**	75.92	3245.02	78.29	3736.82
	$n = 45$	435.33	N/A	75.51	**118.10**	102.17	3682.35	103.91	3731.84
	$n = 50$	1993.63	N/A	321.88	**400.81**	165.21	3921.75	160.36	3911.45

Table A.5: *Comparison of the solving time and the total time for warehouse location problem using different search configurations*

		basic	**manual**	**2-dom**		**3-dom**		**4-dom**	
Configuration	Instance	Solving	Solving	Solving	Total	Solving	Total	Solving	Total
user	$n = 8, m = 6$	5036.37	N/A	2062.95	**2073.58**	1896.55	2440.60	1391.44	5854.01
	$n = 9, m = 6$	7199.84	N/A	6453.90	6459.42	5635.64	**6098.72**	3204.56	6922.67
	$n = 10, m = 6$	7157.01	N/A	6481.20	**6490.04**	6036.78	6675.89	3428.13	7074.74
	$n = 11, m = 6$	7199.81	N/A	7188.13	7199.80	6099.53	7199.74	3599.32	**7199.72**
	$n = 12, m = 6$	7199.78	N/A	7183.05	7199.79	5301.85	7199.68	3599.10	**7199.61**
no-lazy	$n = 8, m = 6$	7199.82	N/A	7197.30	7199.86	7045.64	7199.85	3602.53	**7199.82**
	$n = 9, m = 6$	7199.85	N/A	7195.80	7199.85	6915.31	7199.82	3599.44	**7199.80**
	$n = 10, m = 6$	7199.84	N/A	7193.31	7199.84	6644.24	7199.79	3599.45	**7199.77**
	$n = 11, m = 6$	7199.82	N/A	7031.36	7199.81	6029.56	7199.75	3599.41	**7199.75**
	$n = 12, m = 6$	7199.78	N/A	7184.50	7199.78	5378.66	7199.64	3599.22	**7199.62**
vsids	$n = 8, m = 6$	135.56	N/A	87.22	**96.08**	56.08	291.62	38.72	3755.85
	$n = 9, m = 6$	761.65	N/A	120.94	**130.38**	157.06	585.86	122.91	3870.59
	$n = 10, m = 6$	1857.40	N/A	379.31	**405.39**	404.68	1273.24	315.03	4300.88
	$n = 11, m = 6$	1149.44	N/A	625.02	**654.45**	391.59	1767.92	271.81	4068.67
	$n = 12, m = 6$	2447.37	N/A	986.78	**1023.45**	627.52	2881.53	886.01	4925.37
togglee-vsids	$n = 8, m = 6$	4844.41	N/A	2115.51	**2127.64**	1943.15	2534.05	1368.57	5794.80
	$n = 9, m = 6$	7199.84	N/A	6372.30	6377.48	5689.76	**6229.64**	3242.20	6936.29
	$n = 10, m = 6$	7134.29	N/A	6422.07	**6430.94**	6075.85	6679.76	3390.52	7041.33
	$n = 11, m = 6$	7199.80	N/A	7188.54	7199.81	6094.21	7199.74	3599.33	**7199.74**
	$n = 12, m = 6$	7199.73	N/A	7183.24	7199.79	5325.01	7199.65	3599.04	**7199.57**
free	$n = 8, m = 6$	27.62	N/A	19.44	**23.80**	17.27	181.55	18.02	3688.54
	$n = 9, m = 6$	57.83	N/A	45.36	**51.21**	35.53	334.77	41.93	3646.76
	$n = 10, m = 6$	133.26	N/A	105.43	**114.55**	83.70	668.82	98.32	3725.49
	$n = 11, m = 6$	178.81	N/A	158.74	**173.99**	137.01	1267.50	133.93	3757.69
	$n = 12, m = 6$	409.56	N/A	325.52	**346.33**	240.87	2144.72	352.19	4127.71

Table A.6: *Comparison of the solving time and the total time for team assignment problem using different search configurations*

		basic	**manual**	2-**dom**		3-**dom**		4-**dom**	
Configuration	Instance	Solving	Solving	Solving	Total	Solving	Total	Solving	Total
user	$n = 50$	372.59	N/A	66.03	66.30	15.09	**20.81**	0.34	98.40
	$n = 60$	6675.37	N/A	3597.35	3597.96	881.18	900.23	68.17	**330.18**
	$n = 70$	7199.94	N/A	7199.30	7199.93	6767.93	6788.30	1807.53	**3197.45**
	$n = 80$	7199.91	N/A	7199.14	7199.92	7170.73	7199.94	5606.78	**6523.35**
	$n = 90$	7199.94	N/A	7199.22	7199.96	7162.04	7199.94	5747.59	**7199.85**
no-lazy	$n = 50$	90.87	N/A	20.99	21.26	6.47	**11.92**	0.39	100.73
	$n = 60$	3932.57	N/A	1236.64	1237.29	339.89	355.44	44.56	**289.92**
	$n = 70$	7199.93	N/A	7150.91	7151.53	6148.51	6168.32	1224.82	**2259.78**
	$n = 80$	7199.92	N/A	7199.13	7199.92	7171.27	7199.93	5441.46	**6377.91**
	$n = 90$	7199.94	N/A	7199.18	7199.95	7162.04	7199.94	5761.14	**7199.84**
vsids	$n = 50$	0.96	N/A	0.54	**0.79**	0.25	5.07	0.03	95.53
	$n = 60$	11.56	N/A	8.43	**8.95**	4.75	15.79	1.10	225.15
	$n = 70$	133.46	N/A	104.05	104.93	65.77	**90.07**	15.59	479.36
	$n = 80$	674.62	N/A	574.35	575.90	331.01	**382.69**	154.03	1075.20
	$n = 90$	4717.68	N/A	4176.67	4177.77	2883.90	**2953.84**	1498.60	3317.09
toggle-vsids	$n = 50$	370.29	N/A	66.97	67.24	15.53	**21.28**	0.34	98.99
	$n = 60$	6747.80	N/A	3500.11	3500.74	875.52	894.40	71.17	**330.83**
	$n = 70$	7199.94	N/A	7199.33	7199.92	6696.77	6718.24	1799.34	**3267.77**
	$n = 80$	7199.92	N/A	7199.11	7199.92	7169.04	7199.92	5540.80	**6448.72**
	$n = 90$	7199.94	N/A	7199.23	7199.97	7162.17	7199.96	5770.51	**7199.84**
free	$n = 50$	0.92	N/A	0.54	**0.82**	0.27	5.07	0.05	100.87
	$n = 60$	15.13	N/A	10.25	**10.99**	6.19	18.85	1.50	228.35
	$n = 70$	342.49	N/A	219.80	220.98	112.85	**143.11**	23.27	486.84
	$n = 80$	2253.30	N/A	2050.27	2113.73	1199.13	**1403.51**	324.92	1511.10
	$n = 90$	6731.40	N/A	6758.17	6759.04	6284.80	6337.35	3575.08	**5549.91**

Table A.7: *Comparison of the solving time and the total time for budgeted maximum coverage problem using different search configurations*

		basic	**manual**	2-**dom**		3-**dom**		4-**dom**	
Configuration	Instance	Solving	Solving	Solving	Total	Solving	Total	Solving	Total
	$n = 50$	6382.73	N/A	762.66	763.59	42.33	**53.94**	0.29	117.76
	$n = 60$	7199.95	N/A	6756.30	6756.92	2952.19	2984.79	140.55	**512.66**
user	$n = 70$	7199.95	N/A	7034.95	7199.95	7020.08	7199.94	3904.17	**4619.03**
	$n = 80$	7199.93	N/A	7199.36	7199.96	7176.20	7199.95	5927.26	**6775.10**
	$n = 90$	7199.94	N/A	7199.26	7199.96	7162.93	7199.96	5855.01	**7199.87**
	$n = 50$	3915.22	N/A	280.63	281.58	20.56	**31.47**	0.57	120.73
	$n = 60$	6734.35	N/A	5927.93	5928.50	2052.56	2089.82	123.92	**492.57**
no-lazy	$n = 70$	7167.62	N/A	7035.01	7199.94	7020.43	7199.94	3252.33	**4028.99**
	$n = 80$	7199.91	N/A	7199.35	7199.95	7176.74	7199.94	5709.80	**6587.53**
	$n = 90$	7199.93	N/A	7199.23	7199.95	6895.58	7199.94	5706.54	**7199.85**
	$n = 50$	46.15	N/A	10.78	11.70	1.79	**9.15**	0.07	117.23
	$n = 60$	505.50	N/A	218.47	219.72	73.72	**102.90**	5.16	295.94
vsids	$n = 70$	1944.02	N/A	1370.20	1371.41	605.02	**643.09**	103.20	690.10
	$n = 80$	3650.11	N/A	3297.45	3299.57	2263.63	2349.43	862.70	**2192.06**
	$n = 90$	6886.50	N/A	6496.76	6752.52	5256.49	**5302.60**	3686.00	5385.49
	$n = 50$	6312.22	N/A	757.51	758.33	41.06	**52.78**	0.28	117.01
	$n = 60$	7199.92	N/A	6792.52	6793.15	3070.35	3107.39	142.81	**498.08**
toggle-vsids	$n = 70$	7199.95	N/A	7199.45	7199.95	7183.78	7199.94	3903.96	**4546.32**
	$n = 80$	7199.93	N/A	7199.37	7199.97	7177.11	7199.96	5963.59	**6772.70**
	$n = 90$	7199.94	N/A	7199.24	7199.96	7164.30	7199.98	5818.91	**7199.86**
	$n = 50$	41.06	N/A	8.46	9.37	1.93	**9.57**	0.09	116.80
	$n = 60$	579.55	N/A	214.66	216.93	70.85	**108.05**	6.01	303.22
free	$n = 70$	2803.96	N/A	2011.04	2013.11	931.87	998.03	141.45	**770.46**
	$n = 80$	3526.94	N/A	3237.37	3244.23	2090.34	2248.97	1017.98	**2907.76**
	$n = 90$	6377.07	N/A	6374.45	6375.25	6120.74	6174.15	4080.58	**6171.80**

Table A.8: *Comparison of the solving time and the total time for partial set cover problem using different search configurations*

		basic	**manual**	**2-dom**		**3-dom**		**4-dom**	
Configuration	Instance	Solving	Solving	Solving	Total	Solving	Total	Solving	Total
user	$n = 50$	46.60	N/A	38.02	38.58	28.51	**37.82**	26.04	173.76
	$n = 60$	155.75	N/A	128.10	128.88	95.17	**113.50**	94.04	456.69
	$n = 70$	483.11	N/A	362.14	363.09	269.30	**298.84**	270.85	982.32
	$n = 80$	1279.87	N/A	952.47	953.75	738.70	**783.84**	716.57	2036.10
	$n = 90$	3130.86	N/A	2246.19	2247.73	1617.90	**1681.35**	1580.59	3666.19
no-lazy	$n = 50$	5.79	N/A	4.69	**5.23**	3.93	12.09	3.64	147.99
	$n = 60$	14.62	N/A	11.49	**12.23**	9.26	25.97	8.80	362.46
	$n = 70$	37.54	N/A	28.08	**29.02**	21.57	48.46	20.30	725.67
	$n = 80$	82.81	N/A	64.44	**65.59**	47.82	87.67	46.00	1261.13
	$n = 90$	200.61	N/A	140.48	**142.08**	99.42	159.30	99.82	2126.43
vsids	$n = 50$	91.19	N/A	70.58	71.15	51.92	**60.80**	47.30	214.05
	$n = 60$	404.03	N/A	288.14	288.92	198.41	**217.28**	191.70	563.36
	$n = 70$	1062.26	N/A	761.33	762.25	603.28	**631.94**	544.35	1275.11
	$n = 80$	2853.53	N/A	1961.21	1962.46	1589.95	**1632.57**	1662.12	2992.26
	$n = 90$	6001.75	N/A	4414.18	4415.75	3831.28	**3892.05**	3545.42	5594.86
toggle-vsids	$n = 50$	46.00	N/A	38.19	38.77	29.34	**38.13**	28.89	193.64
	$n = 60$	165.90	N/A	123.12	123.94	103.51	**121.88**	87.27	459.98
	$n = 70$	513.96	N/A	350.68	351.63	277.43	**307.00**	268.64	962.74
	$n = 80$	1378.79	N/A	981.29	982.57	751.04	**795.22**	766.97	2079.57
	$n = 90$	3266.54	N/A	2090.35	2091.95	1652.25	**1717.19**	1650.13	3732.15
free	$n = 50$	97.38	N/A	75.37	75.93	54.98	**64.23**	47.88	197.89
	$n = 60$	525.51	N/A	339.23	340.13	248.25	**267.73**	212.12	609.02
	$n = 70$	2551.71	N/A	1266.53	1267.56	871.64	**903.53**	739.25	1482.12
	$n = 80$	6809.42	N/A	5342.12	5343.51	3827.45	**3880.80**	3259.58	4653.61
	$n = 90$	7199.71	N/A	7197.83	7199.77	6752.91	**6813.06**	4935.26	6997.26

Table A.9: *Comparison of the solving time and the total time for sensor placement problem using different search configurations*

Appendix B

Publications

The results in this thesis have been included in the following publications:

- Conference Publications
  - Jimmy H.M. Lee and Allen Z. Zhong. **Automatic Dominance Breaking for a Class of Constraint Optimization Problems**, Proceedings of the 29th International Joint Conference on Artificial Intelligence and the 17th Pacific Rim International Conference on Artificial Intelligence (IJCAI-PRICAI 2020), pages 1192–1200, Yokohama, Japan, July, 2020.
    The paper contributes to Chapter 4.
  - Jimmy H.M. Lee and Allen Z. Zhong. **Towards More Practical and Efficient Automatic Dominance Breaking**, Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI 2021), pages 3868–3876, a virtual conference, February, 2021.
    The paper contributes to Chapter 5.
  - Jimmy H. M. Lee, Allen Z. Zhong. **Exploiting Functional Constraints in Automatic Dominance Breaking for Constraint Optimization**. Proceedings of the 28th International Conference on Principles and Practice of Constraint Programming (CP 2022), pages 31:1–31:17, July, 2022.
    The paper contributes to Chapter 6.

- Under Review
  - Jimmy H. M. Lee, Allen Z. Zhong. **Towards Automatic Generation of Dominance Breaking Nogoods for Constraint Optimization Problems**. Submitted to Artificial Intelligence.

    The paper is related to Chapters 4, 5 and 7

Bibilography

[1] Akgun, O., Frisch, A. M., Gent, I. P., Hussain, B. S., Jefferson, C., Kotthoff, L., Miguel, I., and Nightingale, P. Automated symmetry breaking and model selection in Conjure. In *Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming* (2013), Springer, pp. 107–116.

[2] Akgun, O., Gent, I. P., Jefferson, C., Miguel, I., and Nightingale, P. Breaking conditional symmetry in automated constraint modelling with CONJURE. In *Proceedings of the 21st European Conference on Artificial Intelligence* (2014), pp. 3–8.

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

[4] Baader, F., and Nipkow, T. *Term rewriting and all that*. Cambridge University Press, 1998.

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

[6] Baptista, L., and Marques-Silva, J. Using randomization and learning to solve hard real-world instances of satisfiability. In *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming* (2000), Springer, pp. 489–494.

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

[8] Beldiceanu, N., Carlsson, M., and Rampon, J.-X. Global constraint catalog, 2010.

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

[10] Beldiceanu, N., and Simonis, H. A model seeker: Extracting global constraint models from positive examples. In *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming* (2012), Springer, pp. 141–157.

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

[12] Bessiere, C., Koriche, F., Lazaar, N., and O'Sullivan, B. Constraint acquisition. *Artificial Intelligence 244* (2017), 315–342.

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

[14] Booth, K. E., and Beck, J. C. 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* (2019), Springer, pp. 129–145.

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

[16] Carlsson, M., and Beldiceanu, N. Arc-consistency for a chain of lexicographic ordering constraints. Tech. rep., Swedish Institute of Computer Science, 2002.

[17] Carlsson, M., and Beldiceanu, N. Revisiting the lexicographic ordering constraint. Tech. rep., Swedish Institute of Computer Science, 2002.

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

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

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

[21] Chu, G., Banda, M. G. d. l., and Stuckey, P. J. 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* (2010), Springer, pp. 71–86.

[22] Chu, G., and Stuckey, P. J. Minimizing the maximum number of open stacks by customer search. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming* (2009), Springer, pp. 242–257.

[23] Chu, G., and Stuckey, P. J. 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.

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

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

[26] Cornuéjols, G., Nemhauser, G., and Wolsey, L. The uncapicitated facility location problem. Tech. rep., Cornell University Operations Research and Industrial Engineering, 1983.

[27] Crawford, J. M., Ginsberg, M. L., Luks, E. M., and Roy, A. Symmetry-breaking predicates for search problems. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning* (1996), pp. 148–159.

[28] de Una, D., Gange, G., Schachte, P., and Stuckey, P. J. Weighted spanning tree constraint with explanations. In *Proceedings of the 13th International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems* (2016), pp. 98–107.

[29] Dechter, R. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artificial Intelligence 41*, 3 (1990), 273–312.

[30] Fahle, T., Schamberger, S., and Sellmann, M. Symmetry breaking. In *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming* (2001), Springer, pp. 93–107.

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

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

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

[34] Focacci, F., and Shaw, P. Pruning sub-optimal search branches using local search. In *Proceedings of Fourth International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimisation Problems* (2002), pp. 181–189.

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

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

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

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

[39] Frisch, A., Hnich, B., Kiziltan, Z., Miguel, I., and Walsh, T. Global constraints for lexicographic orderings. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming* (2002), pp. 93–108.

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

[41] Frisch, A. M., Hnich, B., Kiziltan, Z., Miguel, I., and Walsh, T. Filtering algorithms for the multiset ordering constraint. *Artificial Intelligence 173*, 2 (2009), 299–328.

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

[43] Frisch, A. M., Miguel, I., and Walsh, T. 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* (2002), Springer, pp. 15–30.

[44] Frost, D., and Dechter, R. Dead-end driven learning. In *Proceedings of the Twelfth AAAI National Conference on Artificial Intelligence* (1994), pp. 294–300.

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

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

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

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

[49] Gent, I. P., Harvey, W., and Kelsey, T. Groups and constraints: Symmetry breaking during search. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming* (2002), Springer, pp. 415–430.

[50] Gent, I. P., Harvey, W., Kelsey, T., and Linton, S. Generic SBDD using computational group theory. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming* (2003), Springer, pp. 333–347.

[51] Gent, I. P., McDonald, I., Miguel, I., and Smith, B. M. Approaches to conditional symmetry breaking. In *Proceedings of the 4th International Workshop on Symmetry and Constraint Satisfaction Problems* (2004).

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

[53] Gent, I. P., and Smith, B. M. Symmetry breaking in constraint programming. In *Proceedings of the 14th European Conference on Artificial Intelligence* (2000), pp. 599–603.

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

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

[56] Glorian, G., Boussemart, F., Lagniez, J.-M., Lecoutre, C., and Mazure, B. Combining nogoods in restart-based search. In *Proceedings of the 23nd International Conference on Principles and Practice of Constraint Programming* (2017), Springer, pp. 129–138.

[57] Grabisch, M., Marichal, J.-L., Mesiar, R., and Pap, E. *Aggregation Functions*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2009.

[58] Grayland, A., Miguel, I., and Roney-Dougal, C. M. Snake lex: An alternative to double lex. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming* (2009), Springer, pp. 391–399.

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

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

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

[62] Huang, X., and Lee, J. H. Doublelex revisited and beyond. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence* (2019), pp. 1101–1107.

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

[64] Kalagnanam, J. R., Dawande, M. W., Trumbo, M., and Lee, H. S. Inventory matching problems in the steel industry. *IBM Research Report, Computer Science/Mathematics* (1998).

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

[66] Katsirelos, G., Narodytska, N., and Walsh, T. On the complexity and completeness of static constraints for breaking row and column symmetry. In *Proceedings of the 16th International Conference on Principles and Practice of Constraint Programming* (2010), Springer, pp. 305–320.

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

[68] Kearns, M. J. *Computational Complexity of Machine Learning*. MIT Press, Cambridge, MA, USA, 1990.

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

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

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

[72] Kosch, S., and Beck, J. C. A new MIP model for parallel-batch scheduling with non-identical job sizes. In *Proceedings of the 11th International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research* (2014), pp. 55–70.

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

[74] Kreter, S., Schutt, A., Stuckey, P. J., and Zimmermann, J. Mixed-integer linear programming and constraint programming formulations for solving resource availability cost problems. *European Journal of Operational Research 266*, 2 (2018), 472–486.

[75] Lam, E., and Hentenryck, P. V. A branch-and-price-and-check model for the vehicle routing problem with location congestion. *Constraints 21*, 3 (2016), 394–412.

[76] Lam, E., and Van Hentenryck, P. Branch-and-check with explanations for the vehicle routing problem with time windows. In *Proceedings of the 23nd International Conference on Principles and Practice of Constraint Programming* (2017), Springer, pp. 579–595.

[77] Law, Y., and Lee, J. H. Symmetry breaking constraints for value symmetries in constraint satisfaction. *Constraints 11*, 2 (2006), 221–267.

[78] Law, Y. C., and Lee, J. H. Global constraints for integer and set value precedence. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming* (2004), Springer, pp. 362–376.

[79] Lecoutre, C., Sais, L., Tabary, S., and Vidal, V. Recording and minimizing nogoods from restarts. *Journal on Satisfiability, Boolean Modeling and Computation 1*, 3-4 (2006), 147–167.

[80] Lee, J. H., Schulte, C., and Zhu, Z. Increasing nogoods in restart-based search. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence* (2016), pp. 3426–3433.

[81] Lee, J. H., and Zhu, Z. Boosting SBDS for partial symmetry breaking in constraint programming. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence* (2014), pp. 2695–2702.

[82] Lee, J. H., and Zhu, Z. An increasing-nogoods global constraint for symmetry breaking during search. In *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming* (2014), Springer, pp. 465–480.

[83] Lee, J. H., and Zhu, Z. Static symmetry breaking with the reflex ordering. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence* (2016), pp. 758–765.

[84] Lee, J. H., and Zhu, Z. Towards breaking more composition symmetries in partial symmetry breaking. *Artificial Intelligence 252* (2017), 51–82.

[85] Lee, J. H. M., and Zhong, A. Z. 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.

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

[87] Lee, J. H. M., and Zhong, A. Z. Exploiting functional constraints in automatic dominance breaking for constraint optimization. In *28th International Conference on Principles and Practice of Constraint Programming (CP 2022)* (Dagstuhl, Germany, 2022), C. Solnon, Ed., vol. 235 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, pp. 31:1–31:17.

[88] Leo, K. *Making the Most of Structure in Constraint Models*. PhD Thesis, Monash University, 2018.

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

[90] Mackworth, A. K., and Freuder, E. C. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial intelligence 25*, 1 (1985), 65–74.

[91] Martin, R. The challenge of exploiting weak symmetries. In *Proceedings of the 2005 Joint ERCIM/CoLogNET International Workshop on Constraint Solving and Constraint Logic Programming* (2005), Springer, pp. 149–163.

[92] Mears, C., and de la Banda, M. G. Towards automatic dominance breaking for constraint optimization problems. In *Proceedings of the 24th International Conference on Artificial Intelligence* (2015), pp. 360–366.

[93] Mears, C., Garcia De La Banda, M., Demoen, B., and Wallace, M. Lightweight dynamic symmetry breaking. *Constraints 19*, 3 (2014), 195–242.

[94] Mears, C., Garcia De La Banda, M., and Wallace, M. On implementing symmetry detection. *Constraints 14*, 4 (2009), 443–477.

[95] Mears, C., Garcia de la Banda, M., Wallace, M., and Demoen, B. A novel approach for detecting symmetries in CSP models. In *Proceedings of the 5th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems* (2008), Springer, pp. 158–172.

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

[97] Monette, J.-N., Schaus, P., Zampelli, S., Deville, Y., and Dupont, P. 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.

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

[99] Narodytska, N., and Walsh, T. Breaking symmetry with different orderings. In *Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming* (2013), Springer, pp. 545–561.

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

[101] Noronha, T. F., Ribeiro, C. C., and Santos, A. C. Solving diameter-constrained minimum spanning tree problems by constraint programming. *International Transactions in Operational Research 17*, 5 (2010), 653–665.

[102] Ohrimenko, O., Stuckey, P. J., and Codish, M. Propagation= lazy clause generation. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming* (2007), Springer, pp. 544–558.

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

[104] Oplobedu, A., Marcovitch, J., and Tourbier, Y. 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* (1989), vol. 1, pp. 55–70.

[105] Petrie, K. E., and Smith, B. M. Symmetry breaking in graceful graphs. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming* (2003), Springer, pp. 930–934.

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

[107] Proll, L., and Smith, B. Integer linear programming and constraint programming approaches to a template design problem. *INFORMS Journal on Computing 10*, 3 (1998), 265–275.

[108] Puget, J.-F. Symmetry breaking revisited. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming* (2002), Springer, pp. 446–461.

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

[110] Puget, J.-F. An efficient way of breaking value symmetries. In *Proceedings of the Twenty-first AAAI National Conference on Artificial intelligence* (2006), pp. 117–122.

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

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

[113] Rossi, F., Beek, P. v., and Walsh, T. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., 2006.

[114] Roy, P., and Pachet, F. Using symmetry of global constraints to speed up the resolution of constraint satisfaction problems. In *Proceedings of ECAI'98 Workshop on non-binary constraints* (1998).

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

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

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

[118] Schrijvers, T., Demoen, B., Duck, G., Stuckey, P., and Frühwirth, T. Automatic implication checking for CHR constraints. *Electronic notes in theoretical computer science 147*, 1 (2006), 93–111.

[119] Schutt, A., Feydy, T., and Stuckey, P. J. Explaining time-table-edge-finding propagation for the cumulative resource constraint. In *Proceedings of the 10th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems* (2013), Springer, pp. 234–250.

[120] Schutt, A., Feydy, T., Stuckey, P. J., and Wallace, M. G. Explaining the cumulative propagator. *Constraints 16*, 3 (2011), 250–282.

[121] Schutt, A., and Stuckey, P. J. Explaining producer/consumer constraints. In *Proceedings of 22nd International Conference on Principles and Practice of Constraint Programming* (2016), Springer, pp. 438–454.

[122] Senthooran, I., Le Bodic, P., and Stuckey, P. J. Optimising training for service delivery. In *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)* (Dagstuhl, Germany, 2021), L. D. Michel, Ed., vol. 210 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 48:1–48:15.

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

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

[125] Umetani, S. 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.

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

[127] Van Hentenryck, P. *The OPL optimization programming language*. MIT press, 1999.

[128] Walsh, T. General symmetry breaking constraints. In *Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming* (2006), Springer, pp. 650–664.

[129] Walsh, T. Symmetry breaking using value precedence. In *Proceedings of the 17th European Conference on Artificial Intelligence* (2006), pp. 168–172.

[130] Yamada, T., Kataoka, S., and Watanabe, K. Heuristic and exact algorithms for the disjunctively constrained knapsack problem. *Information Processing Society of Japan Journal 43*, 9 (2002).

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

[132] Zhang, Y., and Yap, R. H. Making AC-3 an optimal algorithm. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence* (2001), pp. 316–321.