UNIVERSITY OF COPENHAGEN
FACULTY OF SCIENCE

Master thesis

Leveraging cutting planes reasoning for faster Pseudo-Boolean solving and optimization

Ziyang Men

Supervisor: Jakob Nordström, Stephan Gocht

Submitted: May 30, 2022

This thesis has been submitted to the Department of Computer Science, University of Copenhagen

Abstract

Among the execution of modern Boolean Satisfiability (SAT) and Mixed Integer Programming (MIP) solvers, presolving is a commonly employed procedure between the encoding and solving phases with multiple preprocessing techniques to simplify the instance size or improve its "strength". However, this powerful tool has not yet been widely used in Pseudo-Boolean Solving/Optimization closely related to SAT and MIP. In this report, we study categories of SAT/MIP preprocessing techniques and try to adapt or lift them into Pseudo-Boolean configuration. The experiments show that a considerable number of instances could be solved faster after being processed by the presolving when using a Pseudo-Boolean solver, while this still fails to help solve more instances within a certain time limit.

Keywords: Presolve, Pseudo-Boolean Solving, Pseudo-Boolean Optimization, SAT Preprocessing, MIP Preprocessing

Table of Contents

Chapter 1

Introduction

Boolean Satisfiability (SAT) is the canonical NP-complete problem in computational complexity theory [Coo71; Lev73]. A large body of theoretical work provides circumstantial evidence for and builds on the hypothesis that this problem is impossible to solve efficiently in the worst case, e.g., the *Strong Exponential Time Hypothesis (SETH)* [IP01; CIP09] states that 3-SAT cannot be solved in sub-exponential time in the worst case and if it is true, then P $\neq$ NP.

The *cutting-plane proof system* introduced in [CCT87], incorporating the *Gomory-Chvátal* algorithm [Gom63; Chv73a] for *Integer Linear Programming (ILP)*, translates the disjunctive clauses in a SAT formula to linear inequalities and manipulates them to derive a contradiction, in which the question of Boolean satisfiability is reduced to the geometry of polytopes over the integer numbers, which further formalizes the *Pseudo-Boolean (PB)* solving/optimization [HR69; BH02; MM06]. The cutting-plane proof system operates on arbitrary 0-1 linear integer constraints, which is referred to as Pseudo-Boolean constraints, and a Pseudo-Boolean formula is a set of Pseudo-Boolean constraints which is also known as the 0-1 ILP.

Although SAT is believed to be hard on the theoretical side, applied research over the last two decades has meet a quick development. Combined with various heuristic and dedicated data structures, the *conflict-driven clause learning (CDCL)* [GRA99; Mos+01a] based SAT/PB solvers such as [BS97; MS99; Mos+01b; CK05; SE05; SS06; LP10; EN18] have shown to be highly efficient tools for solving large-scale practical optimization problems, especially in detecting the satisfiability for *Conjunctive Normal Form (CNF)* encoded decision instances. However, existing SAT/PB solvers still struggle to compete with *Mixed Integer Programming (MIP)* solvers, e.g., [Bix07; Ach09; BBL14] when the inputs are general optimization instances, as observed in [EN18; Dev+21].

A significant reason for the extraordinary performance of modern linear programming solvers is the application of the schemes and routines that are commonly referred to as *presolve*. Presolve is a process where the problem input by the user is examined for logical reduction opportunities based on presolving or preprocessing techniques. The goal is to reduce the size of the problem passed to an optimizer. A reduction in problem size typically translates to a reduction in total run time (even including the time spent

in the presolve itself) [Man87]. It might not always be possible to apply presolving on problem instances, meaning that the problem size may remain unchanged or there is no acceleration for the afterward problem-solving even if the instance size has been reduced, as discussed in [Sav94; Mar99; FM05a; BJK21]. However, looking at a broader and general picture, it was reported in [AW13; Ach+20] that the presolving is a powerful tool for MIP and in some cases, it indeed effects whether the problem is solvable or not.

There is already some research conducted on SAT preprocessing techniques. One of the earliest and most important contribution was the *Bounded Variable Elimination* [Bie04b; BJK21] and is followed by the *Hyper Binary Resolution*, [Bac02; HJB13], *Block Clause Elimination* [JBH10a; JBH10b], etc. A comprehensive overview of SAT-oriented presolving techniques is reported in [BJK21]. Similarly, [BMW75] is one of the earliest papers introducing presolving techniques, such as *Identifying Exclusive Row Structure*, *Variable Fixing*, etc., into MIP solving systematically. Following research such as *0-1 inequalities* [JS80; GS81; CJP83; HP91], *Generalized Probing Techniques* [Sav94], *Linear Programming* [AA95], *Dominate Column* [Gam+15], etc., has substantially enriched the contents of MIP presolving. A thorough summary of the current research on MIP presolving techniques appears in [Ach+20].

In contrast, to the best of our knowledge, related publications on presolving techniques specifically for Pseudo-Boolean solving/optimization are surprisingly small: [DG02] firstly uses the *Constraint Strengthening* as a preprocessing technique in Pseudo-Boolean solving, the adapted technique is also known as the *Strengthening Rule* in SAT community later. Two dedicated variable handle techniques from the MIP view are introduced in [BHP08] while the effect is reported to be marginal. The [MLM09] adapts the *Extended Hyper Resolution* in SAT clause inference for some specific types of Pseudo-Boolean constraints and the work in [MML10] uses the Pseudo-Boolean solver itself, integrated with *Constraint Branching* from ILP, as a preprocessor for following unsatisfiability-based algorithm. Later MaxSAT and Pseudo-Boolean solver [Pio20] combined with MaxSAT preprocessor [Kor+17] has shown good performance in MaxSAT Evaluation 2019.

This report aims to benefit from the advanced preprocessing techniques employed in both SAT and MIP communities, try to lift/adapt some of them into Pseudo-Boolean configuration and evaluate the corresponding effectiveness concerning the Pseudo-Boolean instances and the solving phase on a Pseudo-Boolean solver. None of novel presolving techniques are proposed in this report, but as far as we know, some of them are firstly adapted into Pseudo-Boolean configuration.

The remaining contents are organized as follows: in the next chapter Preliminary, the notation system we use and some basic concepts about SAT/MIP/PB are provided. The following two chapters SAT preprocessing and MIP preprocessing tries to lift/adapt SAT and MIP preprocessing techniques respectively. Chapter Experimental evaluation measures to what extent some presolving techniques will affect the following solving phase on practical side, the report then ends with the Conclusions and future work.

Chapter 2

Preliminary

2.1 Notation system

A Pseudo-Boolean *variable* is a binary variable $x^\sigma = \{0,1\}, \sigma \in \{0,1\}$ where $x^1 = x, x^0 = \bar{x}$ and $x^\sigma + x^{1-\sigma} = 1$. Denote the lower bound for a Pseudo-Boolean variable x_i^σ as l_i^σ equals to 0 and the upper bound as u_i^σ equals to 1.

A Pseudo-Boolean *constraint* is a 0-1 linear integer constraint:

$$\sum_{i,\sigma} a_i^\sigma x_i^\sigma \circ A, \quad a_i^\sigma \in \mathbb{Z}, A \in \mathbb{Z} \tag{2.1}$$

where $i \in \{1, \cdots, n\}, \sigma \in \{0,1\}$ and $\circ = \{\leq, \geq, =, <, >\}$. For simplicity, we use a shorthand notation $[n] = \{1, \cdots, n\}$ to represent a set of indices. The constant a_i^σ is called the *constraint coefficient* of x_i^σ. Sometimes, it is convenient to use the constraints written in *normalized form*:

$$\sum_{i\in[n],\sigma\in\{0,1\}} a_i^\sigma x_i^\sigma \geq A, \quad a_i^\sigma \in \mathbb{N}, \min\{a_i^0, a_i^1\} = 0, A \in \mathbb{N}^+ \tag{2.2}$$

where the constant term $A = \deg(\sum_{i\in[n]} a_i^\sigma x_i^\sigma \geq A)$ is referred to as *degree (of falsity)*. The condition $\min\{a_i^0, a_i^1\} = 0$ ensures x_i^0 and x_i^1 cannot appear in a same constraint.

Given a set of variable indices S, denote $x_S^\sigma := \{x_i^\sigma : i \in S\}$ to be the variables indexed by S. Similarly, for a set of constraint indices R, denote $C_R := \{C_j : j \in R\}$, i.e., the constraints whose index is contained in R. Write $x_i^\sigma \in C_j$ if constraint C_j contains variable x_i^σ (with non-zero coefficient), and $x_S^\sigma \in C_j$ if $x_i^\sigma \in C_j$ holds for all $i \in S$. Say a constraint C_j is *k-ary* if it contains exactly k distinguish variables, i.e., $k = \arg\max_{|S|} x_S^\sigma \in C_j$, specifically, if $k = 1$, we call it a *unit constraint*.

The Pseudo-Boolean *formula* F is a conjunction of $M = \{1, \cdots, m\}$ pseudo-Boolean constraints C_j over $N = \{1, \cdots, n\}$ variables $x_N^\sigma, \sigma \in \{0,1\}$ such that:

$$F \doteq \bigwedge_j C_j, \quad j \in M \tag{2.3}$$

Given two subsets $S \subseteq N, R \subseteq M$, denote $F(x_S^\sigma, C_R)$ as the *sub-formula* induced from F by picking out constraints C_R which cover variables $x_S^\sigma, \sigma \in \{0,1\}$. For simplicity, we denote all variables in F as $Vars$ and for a constraint C, denote $Vars(C) = \{x_i^\sigma | x_i^\sigma \in C\}$.

For a subset $S \subseteq N$, a *truth assignment* ρ is a mapping $\{x_i^\sigma \to \{0,1\} | \forall i \in S\}$, i.e., $\rho(x_i^\sigma)$ implies $x_i^\sigma := 1$ and $x_i^{1-\sigma} := 0$, and is called a *partial assignment (trial)* if $S \subset N$ or a *total assignment* if $S = N$. Specifically, write $\rho(\emptyset)$ to indicate no value has been assigned for any variable.

Similarly, define a *substitution* ω as a (partial) function $Vars \to Vars \cup \{0,1\}$, e.g., $\omega = \{x_i^\sigma \mapsto x_j^\sigma\}$ implies x_i^σ is substituted by x_j^σ. Applying a substitution ω to a constraint C shown in 2.2 is shorthand as $C\restriction_\omega$, which is equivalent to substitute all variables in C according to the ω and normalize it again, likewise for a formula, we write $F\restriction_\omega = \bigwedge_j C_j\restriction_\omega$.

Given a formula F and a constraint C, denote $F \vDash C$ as F *implies* C, i.e., all satisfying assignment for F must satisfy C as well, and denote $F \vDash F'$ if $F \vDash C'$ holds for all constraints $C' \in F'$. Specifically, if F contains only one constraint D, we write $D \vDash C$.

Pseudo-Boolean Solving (PBS) is to decide whether a given formula F is *satisfiable/feasible*, i.e., determine the existence of a truth assignment ρ such that F is evaluated to true. The *Pseudo-Boolean Optimization* (PBO) is to find a satisfying assignment to F that minimizes the objective function:

$$\sum_{i \in [n], \sigma \in \{0,1\}} w_i^\sigma x_i^\sigma, \quad w_i^\sigma \in \mathbb{N} \tag{2.4}$$

where the constant term w_i^σ is called the *objective coefficient* of x_i^σ.

2.2 Concept explanation

1. *Constraint transformation.* Given a Pseudo-Boolean constraint, it is always possible to substitute the variables whose coefficient is negative by its negation with positive coefficient, e.g.,

$$-x_i^\sigma \geq 0 \quad \Longrightarrow \quad x_i^{1-\sigma} \geq 1 \tag{2.5}$$

Applying this scheme, we can change every Pseudo-Boolean constraint into normalized form.

Given a Pseudo-Boolean constraint in normalized form 2.2 with

$$\sum_{i \in [n], \sigma \in \{0,1\}} a_i^\sigma = K$$

The *negation* of C is defined to be:

$$\neg C \quad \doteq \sum_{i \in [n], \sigma \in \{0,1\}} a_i^\sigma x_i^{1-\sigma} \geq K - A + 1 \tag{2.6}$$

2. *Constraint classification.* Using the same constraint classification in [MLM09], we divide general Pseudo-Boolean constraints in normalized form into three different categories:

   a) *Clauses*, which is the same as a CNF SAT clause:

   $$\sum_{i\in[n],\sigma\in\{0,1\}} x_i^\sigma \geq 1 \tag{2.7}$$

   b) *Cardinality constraints*, where all coefficients equal to 1, i.e., :

   $$\sum_{i\in[n],\sigma\in\{0,1\}} x_i^\sigma \geq A \tag{2.8}$$

   c) *General constraints*:

   $$\sum_{i\in[n],\sigma\in\{0,1\}} a_i^\sigma x_i^\sigma \geq A \tag{2.9}$$

   Sometimes it is convenient to represent a normalized constraint in equivalent smaller-or-equal relation, hence we define the *reverse general constraints* to be the constraints in form:

   $$\sum_{i\in[n],\sigma\in\{0,1\}} a_i^\sigma x_i^\sigma \leq A \tag{2.10}$$

   and the *equal general constraints* to be:

   $$\sum_{i\in[n],\sigma\in\{0,1\}} a_i^\sigma x_i^\sigma = A \tag{2.11}$$

3. *Slack.* During the solving phase of SAT/PB solver, the *slack* of a constraint in normalized form under a partial assignment (trial) ρ measures how far this constraint is from being falsified by ρ, which is defined as the sum of the coefficients of all literals that are not falsified (i.e., those variables unassigned value or has been assigned true) by ρ minus the degree of the falsity of this constraint:

   $$slack\left(\sum_{i\in[n],\sigma\in\{0,1\}} a_i^\sigma x_i^\sigma \geq A;\rho\right) = \sum_{\rho(x_i^\sigma)\neq 0} a_i^\sigma - A \tag{2.12}$$

   The constraint is *conflicting* under ρ if

   $$slack\left(\sum_{i\in[n],\sigma\in\{0,1\}} a_i^\sigma x_i^\sigma \geq A;\rho\right) < 0 \tag{2.13}$$

   and for x_{i^*} not in the domain of ρ, it propagates $x_{i^*}^{\sigma^*}$ under ρ if:

   $$0 \leq slack\left(\sum_{i\in[n],\sigma\in\{0,1\}} a_i^\sigma x_i^\sigma \geq A;\rho\right) < a_{i^*}^{\sigma^*} \tag{2.14}$$

4. *Constraint strength.* Intuitively, formulating the strengthen of a constraint is tricky, partially because that constraints have interleaving effects during the solving: one constraint which is trivially satisfied at present might play an essential role during the next iteration, as documented in [BJK21] [1]. However, it is still possible to draw some indistinct picture towards the strength of a constraint which could be helpful for further analysis:

   a) From the view of the cutting-plane and feasible solution region, we say a constraint is *stronger* than the other if it could cut off more assignments, which is equivalent to fewer feasible assignments for this constraint. For example:

   $$x_1 + x_2 + x_3 \leq 1$$
   $$x_1 + x_2 + x_3 \leq 2$$

   the first constraint is stronger since it rules out assignment patterns $\{x_1 = x_2 = 1; x_1 = x_3 = 1; x_2 = x_3 = 1\}$ satisfying the second constraint.
   Another example is:

   $$\sum_{i=1}^{k} x_i + \sum_{j=1}^{l} y_j \geq \beta$$
   $$\sum_{j=1}^{l} y_j \geq \beta$$

   The second constraint is stronger since the feasible region of assignment is $\{0,1\}^l$, which is smaller compared with the $\{0,1\}^k \times \{0,1\}^l$ defined by the first one, in other words, every satisfying assignment for the second constraint will satisfy the first one but not vice versa.

   b) In Pseudo-Boolean setting, one possible view for the strength of a normalized constraint C is to evaluate its initial slack, i.e., $slack(C, \rho(\emptyset))$, generally a constraint with smaller initial slack could be stronger. This is because the execution of a CDCL solver lies on the unit propagation on variables, and after every variable decision it is not worse to make as much unit-propagation as possible since this could reduce the size of the search tree afterward. In Pseudo-Boolean solving, the unit propagation checks the constraints' slack regarding the current partial assignment. Note that unless the solver undoes assignment, the slack for a constraint will not increase, which leads to that those constraints with smaller initial slack are more "likely" to incur a unit propagation.

[1] This is quite similar to the categories of presolving techniques, where one technique is unable to make further progress, another method might be applicable, and might even modify the formula in ways that trigger the first technique again.

For example, consider two general constraints C and D below:

$$\begin{aligned} C &\doteq \quad 3x_1 + 2x_2 + x_3 \geq 3 \\ D &\doteq \quad 4y_1 + 2y_2 + 2y_3 \geq 3 \end{aligned}$$

We have $slack(C; \rho(\emptyset)) = 3$ smaller than $slack(D; \rho(\emptyset)) = 5$, and find that falsify arbitrary variable in C will cause a unit propagation to others, while this happens to D only if $y_1 := 0$.

5. *Some general conditions must be hold for presolving techniques.* In the rest of our work, the following criteria will be employed to test whether a presolving approach is valid for Pseudo-Boolean Solving/Optimization. Without additional comments, the aforementioned presolving techniques are supposed to satisfy all the following conditions.

   a) A presolving technique should preserve the **satisfiability** of the problem.
   Whenever a presolving technique adds a non-trivial constraint or fix value for some variables, it cuts off some vertexes over the solution region, leading to the removal of some feasible solutions. This is where the power of the presolving comes from, but another side of the sword is the possibility of a satisfiable problem being unsatisfiable afterward. Symmetrically, discard constraints or removal of variables may lose the boundary of the feasible region which may lead to an unsatisfiable instance becoming satisfiable. In this case, it is necessary to ensure one preprocessing technique will permanently preserve the satisfiability of the problem.

   Specifically, if the problem is PBO, the technique should preserve the optimality as well, e.g., it is allowable for this technique to cut off some optimal solutions on the feasible region but not all.

   b) A presolving technique should not break the **binary bound** of variables.
   The Pseudo-Boolean problem is intrinsic 0-1 ILP and the underlying reasoning logic of a Pseudo-Boolean solver is established on variables being boolean. Whenever a presolving technique adds a new variable or modifies the variable bounds, it should ensure the variable is still binary.

   c) The original solution should be **reconstructable**.
   Since the structure of the problem after a non-trivial presolving has been modified, the following solving phase is actually conducted on a subspace w.r.t the original one defined by the constraints in the non-presolved case; therefore, if the solution is not reconstructable then we actually solve the reduced problem rather than the original one. Besides, since the best commercial solver cannot fully guarantee the produced solution is correct, the original solution becomes a significant element within the validation phase of the solver.

   d) The expected **execution time** for a presolving technique should be negligible w.r.t the solving phase.

When we start to presolve an instance, we actually begin to solve it. In this case, if we talk about the solving time for a problem, it is reasonable to take the execution time for presolving techniques into consideration as well; otherwise it is always possible to presolve the problem to solve, e.g., simply enumerate solutions, which is indeed trivial for the acceleration of the solving phase for following SAT/PB/MIP solver.

Chapter 3

SAT preprocessing

In this section, we follow the routine presented in [BJK21] to lift some SAT preprocessing techniques into Pseudo-Boolean setting.

3.1 Classical CNF level techniques

We start from four classical CNF level techniques: *Unit Propagation*, *Failed Literal Elimination*, *Pure Literal Elimination*, and *Connected Components*. These category techniques are inspired mainly by the underlying scheme of SAT solving, e.g., the conflict analysis and the property of variables or clauses.

3.1.1 Unit propagation

In SAT community, the *Unit Propagation* [DP60] is to check whether a formula includes a unit clause, e.g., a clause that contains exactly one literal. If so, clearly this literal has to be true. All clauses containing such literal are trivially satisfied then and can be discarded. After that, the negation of this literal is removed from the formula.

A simple but efficient algorithm that implements the unit propagation is called *two-watched literal scheme* [Mos+01c]. The idea is that one clause contains more than two unassigned literals will not trigger a unit propagation, the algorithm then *watches* two arbitrary literals in a clause and once either of them is assigned to false, this literal will be replaced with another unassigned literal in the clause; if it is not possible to make a replacement, the other watched literal will be unit propagated to true.

In MIP community, unit propagation is also known as *Boolean Constraint Propagation (BCP)* [CK05]. In Pseudo-Boolean setting, consider a *unit constraint* C such as:

$$C \doteq a_i^\sigma x_i^\sigma \geq A, \quad i \in [n], \sigma \in \{0, 1\} \tag{3.1}$$

If $A > a_i^\sigma$ the constraint is infeasible; otherwise setting $x_i^\sigma := 1$ would satisfy the

constraint. For all other constraints $D \in F \setminus \{C\}$ in form:

$$D \doteq a_i^\sigma x_i^\sigma + \sum_{j,\sigma} a_j^\sigma x_j^\sigma \geq B \tag{3.2}$$

where $j \neq i, \sigma \in \{0,1\}$. If $slack(D; \rho(x_i^\sigma)) < 0$ then a conflict has been reached, otherwise variable x_j^σ will be unit propagated to true if its coefficient satisfies:

$$0 \leq slack(D; \rho(x_i^\sigma)) < a_j^\sigma \tag{3.3}$$

which may further trigger another unit propagation. We can repeatedly apply the unit propagation until either it derives a conflict or no more unit-constraints are left. Finally, all variables that are propagated to true can be weakened.

3.1.2 Failed variable

In SAT solving, a literal ℓ is called a *failed literal* [ZM88; Le 01] if adding unit clause (ℓ) to formula yields a conflict. Similarly, in Pseudo-Boolean setting, we say a variable $x_i^\sigma, i \in N$ is a *failed variable* w.r.t a formula F if adding constraint $x_i^\sigma = 1$ to F would cause a conflict by unit propagation, which means F implies $x_i^{1-\sigma} = 1$. Such technique that adds the negation of the failed variable to the formula is called *Failed-Variable Probing*, which is a variant of the classical MIP preprocessing technique Probing.

In practical SAT preprocessing, it needs quadratic time to simplify and test all literals whether they are failed literals [BJK21]. Combined with the linear time in unit propagation, it needs cubic time to perform the total failed literal detection. One of the optimization methods proposed by [ABS99; SNS02] is to skip those literals that are propagated during the unit propagation since the last time a failed literal has been found (because they will not trigger a propagation) and these literals will become available until next new failed literal is detected.

3.1.3 Pure variable

In SAT solving, one literal ℓ is called a *pure literal* [DP60; Dun+67] if its negation $\bar{\ell}$ is not contained in the formula. Here we say a *pure variable* $x_i^\sigma, i \in N$ is a variable whose negation $x_i^{1-\sigma} \notin F$. For Pseudo-Boolean Solving, assuming the formula is in normalized form, we can set all pure variables to true since this will always decrease the constraint's degree. Besides, for Pseudo-Boolean Optimization with normalized constraints, a pure literal can be set to true in advance if and only if its objective coefficient is non-positive as well.

3.1.4 Connected components

The SAT community has observed that a formula can often be divided into separate components that can be solved independently [BS06]. Such a strategy that seeks independent sub-problems within a formula is also suitable for Pseudo-Boolean Solving and Optimization.

Specifically, given a Pseudo-Boolean formula F with N variables and M constraints, one can construct a bipartite graph $G = (L \cup R, E)$ where all variables $x_i^\sigma \in L, \forall i \in N$ and constraints $C_j \in R, \forall j \in M$. There is an edge between x_i^σ and C_j if and only if C_j contains either x_i^σ or $x_i^{1-\sigma}$ [1]. By finding the disconnected components in G we can separate F into smaller sub-formulas, which can be solved independently.

For example, consider following constraints:

$$\begin{aligned} 3x_1 + 2x_2 + x_3 &\geq 3 \\ 2\bar{x}_1 + x_3 &\geq 1 \\ x_4 + x_5 &\geq 2 \end{aligned}$$

the corresponding bipartite graph G is shown in Figure 3.1.

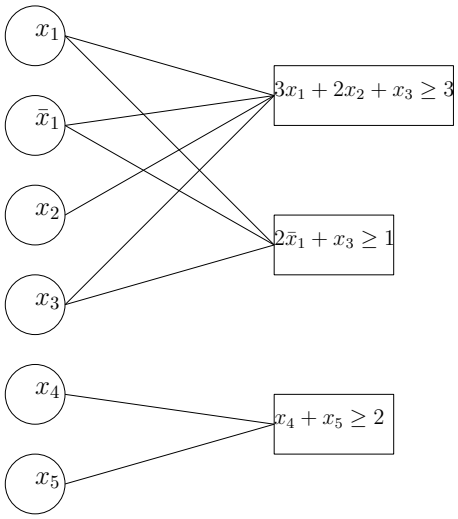


Figure 3.1: Bipartite graph for PB formula with two disconnected components

As mentioned, this method is feasible for both Pseudo-Boolean Solving and Pseudo-Boolean Optimization. Because, for the former, a union of satisfiable assignments of every component is satisfiable for the whole problem as well; for the latter, a minimal global objective is valid if and only if all sub-objectives for disconnected components are minimized.

3.2 Resolution-based preprocessing

Similarly to that cutting-plane underlies the basis of Pseudo-Boolean solver, the *resolution proof system* [Bla37; DR01; DLL62; Rob65] provides the theoretical support

[1] We can also add a edge between x_i^σ and $x_i^{1-\sigma}$ and add edges if and only if x_i^σ is contained in C_j, but this will destroy the bipartite.

for most modern SAT solvers [BN21]. Given two clauses B, C and a variable x, the *resolution rule* allows to derive a new clause as follows:

$$\frac{B \vee x \quad C \vee \bar{x}}{B \vee C} \tag{3.4}$$

Correspondingly, in cutting-plane proof system, we can perform *cancelling addition* between two constraints as:

$$\frac{a_j x_j^\sigma + \sum_{i \neq j,\sigma} a_i^\sigma x_i^\sigma \geq A \quad b_j x_j^{1-\sigma} + \sum_{i \neq j,\sigma} b_i^\sigma x_i^\sigma \geq B}{\sum_{i \neq j,\sigma} \left((b_j a_i^\sigma / d) + (a_j b_i^\sigma / d)\right) x_i^\sigma \geq b_j A/d + a_j B/d - a_j b_j / d}$$

where d is the greatest common divisor of a_j and b_j.

The resolution-based preprocessing technique aims to apply the resolution rule on clauses in some specific manner beforehand to derive valuable inferences that may guide the following *Davis-Putnam-Logemann-Loveland (DPLL)* search in SAT/PB solver, accompanying by the acceptable increase of the clause number. Since the cutting-plane could efficiently simulate resolution [BN21], all these techniques are directly applied to clauses PB constraints and the focus of this section is to lift them for other types of constraints. Four commonly used SAT preprocessing are included in this section, namely, the *Bounded Variable Elimination*, *Implication Graph-based Resolution*, *Hyper Binary Resolution*, and *Extended Hyper Resolution*.

3.2.1 Bounded variable elimination

In SAT community, the *Bounded Variable Elimination (BVE)* [Bie04b; SP04a; SP04b], also known as *Clause Distribution* [DP60], performs all possible resolutions over a certain literal ℓ, adds back the resolvent, and removes clauses containing ℓ. Since simply eliminating a single literal would incur a quadratic increment of clause number [BJK21], the original bounded variable elimination is generally performed only when resolving a literal will not increase the clause number too much, which explains the meaning of "bounded" as well.

In Pseudo-Boolean setting, the idea here is to choose a variable x_i^σ and perform all possible canceling addition upon this variable and add back the resolvent constraints. After that, we can remove x_i^σ from the formula by using the *Fourier-Motzklin Elimination (FME)*[2] [Dan72].

Since performing canceling addition is implicationally sound, meaning that every satisfying assignment for the formula and resolvent constraint satisfies the initial formula as well, combining with that FME does not change the feasible solution over the system, above procedure preserves the satisfiability.

[2]The idea of FME is to rewrite the inequality system based on x, the variable to be eliminated, and formulate the new system by setting all upper bounds of x greater than the lower bounds of x.

The BVE combined with Subsumption and tautology elimination could be more powerful as documented in [BJK21]. For example, consider following constraints:

$$\begin{aligned} 3x_1 + 2x_2 + x_3 + x_4 &\geq 5 \\ 2\bar{x}_2 + \bar{x}_4 &\geq 1 \\ 3x_1 + x_3 &\geq 2 \end{aligned}$$

assume x_2 is the bounded variable, we can perform cancelling addition over x_2 between first two constraints, which yields:

$$3x_1 + x_3 + x_4 + \bar{x}_4 \geq 4$$

eliminate tautology we have:

$$3x_1 + x_3 \geq 3$$

which subsumes the third constraint and thence we can remove it.

On the implementation side, as initiated in [Bie04a; Bal+16] and pointed out in [BJK21], it could be helpful to relax the number of bound clauses added *incrementally*, i.e., the number to be added formed a geometric series $0, 8, 16, 32 \cdots, 8192$. Another trick is to take the order of eliminated variable candidates into consideration. As discussed in [BJK21], a common method is to use a min-binary-heap storing variables based on their appearance frequency and always eliminate the one on the top of heap. The heap is then updated dynamically when the elimination is performed.

3.2.2 Techniques based on implication graphs

Given a Pseudo-Boolean formula, the corresponding *binary implication graph* [APT79] is a directed graph, where the nodes are variables in the formula and there is a directed edge $e = (x_i^\sigma, x_j^\sigma)$ if and only if there is a 2-ary constraint such as:

$$a_i^\sigma x_i^{1-\sigma} + a_j^\sigma x_j^\sigma \geq A, \quad i \neq j, \sigma \in \{0,1\}, a_j^\sigma \geq A \tag{3.5}$$

meaning that $x_i^\sigma := 1$ would cause unit propagation $x_j^\sigma := 1$. The condition $a_j^\sigma \geq A$ ensures that the formula can still maintain feasible when $x_i^\sigma := 1$, otherwise $x_i^\sigma \equiv 0$.

Given a binary implication graph, the nodes in the same *Strong-Connected-Component (SCC)* are *logically equivalent* [VT95; Li00] since setting one of the nodes to true will unit propagate others to true via the translate direction indicated by the directed edges. In this case, variables in same SCC can be replaced by a new variable. If both a variable and its negation are contained in same SCC then the formula is unsatisfiable. Performing canceling addition along the edges in one SCC would yield a trivially satisfied constraint.

For example, consider following constraints:

$$
\begin{aligned}
2\bar{x}_1 + x_2 &\geq 1 \\
3\bar{x}_2 + 2x_3 &\geq 2 \\
2\bar{x}_3 + x_5 &\geq 1 \\
2\bar{x}_4 + x_2 &\geq 1 \\
3\bar{x}_5 + 2x_4 &\geq 2 \\
3\bar{x}_5 + 2x_6 &\geq 2
\end{aligned}
$$

the corresponding implication graph is shown in Figure 3.2: the SCC formed by x_2, x_3, x_4, x_5

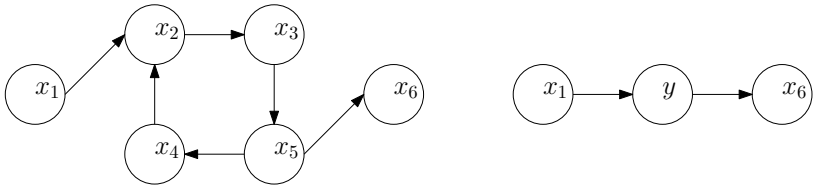


Figure 3.2: Binary implication graph for the formula and the equivalent form

can be replaced by a newly introduced binary variable y and above constraints are equivalent to:

$$
\begin{aligned}
2\bar{x}_1 + y &\geq 1 \\
3\bar{y} + 2x_6 &\geq 2
\end{aligned}
$$

This approach is applied for PBS since all possible satisfying assignments should follow the topology shown in the corresponding binary implication graph. For PBO, assume an index set $S \subseteq N$ where x_S^σ contained in same SCC and w_S^σ are their objective coefficients, simply replace x_S^σ by a new introduced binary variable y and set the objective coefficient $w_y := \sum_{i \in S, \sigma} w_i^\sigma$ will preserve the optimality as well.

3.2.3 Hyper binary resolution

The *binary resolution* is to perform cancelling addition over a certain variable between two 2-ary constraints:

$$
\frac{a_i^\sigma x_i^\sigma + a_j^\sigma x_j^\sigma \geq A \qquad b_i^\sigma x_i^{1-\sigma} + b_j^\sigma y_j^\sigma \geq B}{(a_j^\sigma b_i^\sigma / d) x_j^\sigma + (a_i^\sigma b_j^\sigma / d) y_j^\sigma \geq b_i^\sigma A/d + a_i^\sigma B/d} \tag{3.6}
$$

where $d = gcd(a_i^\sigma, b_i^\sigma)$. The outcome is either:

1. another 2-ary constraint if $x_j^\sigma \neq y_j^\sigma$ and $a_i^\sigma, a_j^\sigma, b_i^\sigma, b_j^\sigma$ are non-zeros;

2. a constant inequality if $x_j^\sigma = y_j^{1-\sigma}$ and $a_j^\sigma b_i^\sigma = a_i^\sigma b_j^\sigma$;

3. a unit constraint otherwise.

for the third case, we can perform unit propagation then.

Furthermore, consider performing canceling addition between one long constraint and a group of 2-ary constraints simultaneously, formally, given a constraint C and a constant k such as:

$$C \doteq \sum_{i,\sigma} a_i^\sigma x_i^\sigma \geq A, \quad i \in \{1, \cdots, k\} \tag{3.7}$$

and set of 2-ary constraints D_i in form:

$$D_i \doteq b_i^\sigma x_i^{1-\sigma} + b_j^\sigma x_j^\sigma \geq A_i, \quad \forall i \in \{1, \cdots, k-1\} \tag{3.8}$$

the *Hyper Binary Resolution (HBR)* [Bac02] allows the derivation of a new 2-ary constraint D:

$$D \doteq \alpha x_k^\sigma + \beta x_j^\sigma \geq \gamma \tag{3.9}$$

where α, β, γ are the coefficients obtained by performing cancelling addition over x_i^σ among C and D_i sequentially, where $i = \{1, \cdots, k-1\}$. Specifically, if $k = j$, D would become a unit constraint and may trigger another unit propagation. The final constraint D can be added back to the formula and perform another binary resolution then. Since cancelling addition is implicationally sound, this will perserve both satisfiability and optimality.

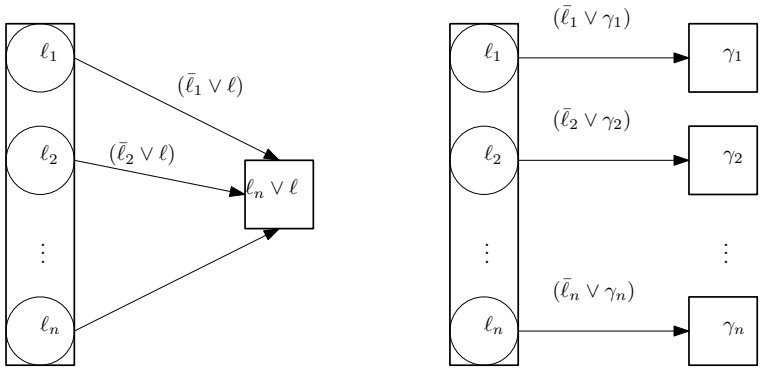
On the implementation side, as introduced in [HJB10; AJS08], detecting the possible HBR can be carried on a binary implication graph using the *transitive closure* and the idea is to check whether all literals in the graph have a common neighbor. As for adding edges to the implication graph, one naive approach is to add edges indicated only from binary constraints, but as point out in [Bac02; HJB13], one can run unit propagation for every variable and add edges from the negation of this variable to those who are propagated. The effectiveness of HBR is stressed in [BJK21], which shows that by repeatedly using HBR and unit propagation, a formula could be simplified significantly.

3.2.4 General implication graph and Extended hyper resolution

As introduced in [AJS08], it is possible to construct the implication graph based on clauses with arbitrary size. The idea is to add an edge $(\bar{\ell}_i, \ell_j), i, j \in [n]$ associated with a *context* $\eta = \{\bar{\ell}_k | k \in [n] \setminus \{i, j\}\}$ for every clause C with n literals $\ell_1, \cdots, \ell_n$ where $n \geq 3$, such that an assignment satisfies η makes C a binary clause.

Unlike HBR which aims to obtain a single binary clause, the *Extend Hyper Resolution (EHR)* [AJS08] tries to derive as many clauses as possible, formally:

$$\frac{(\ell_1 \vee \ell_2 \vee \cdots \vee \ell_n) \quad (\bar{\ell}_1 \vee \gamma_1), \cdots, (\bar{\ell}_n \vee \gamma_n))}{(\bigcup_{1 \leq i \leq n} (\gamma_i))} \tag{3.10}$$



Hyper Binary Resolution Extend Hyper Resolution

Figure 3.3: Comparison of Hyper Binary Resolution and Extend Hyper Resolution. Circles $\ell_1, \cdots, \ell_n$ represent the literals and all the rectangles $\gamma_1, \cdots, \gamma_n$ represent the clauses.

where γ_i are sub-clauses. Figure 3.3 inspired from [AJS08] illustrates the difference between HBR and EHR.

Since EHR can be performed on a general implication graph efficiently [AJS08], therefore how to construct the general implication graph with respect to a Pseudo-Boolean formula becomes the key component when lifting this technique. The attempt which tries to resolve this problem appears in [MLM09]. For clause constraint, their approach is the same as those used in SAT. For cardinality constraints, e.g., $C = x_1^\sigma + \cdots + x_n^\sigma \geq A$, the principle to construct an implication graph $G(V, E)$ where:

- nodes $x_i^\sigma, x_i^{1-\sigma} \in V$ if and only if $x_i^\sigma \in C$;
- edges $e = (x_i^{1-\sigma}, x_j^\sigma, \eta) \in E$ if and only if $x_i^\sigma \in C$ and $n - A - 1 \geq 0$ and $\eta = \bigcup_k \{x_k^{1-\sigma}\}$ with $(i+1) \mod n \leq k \leq (i+n-A-1) \mod n$ for each j such as $(i+n-A) \mod n \leq j \leq (i+A+1) \mod n$.

for example, given constraint

$$x_1 + x_2 + x_3 + x_4 \geq 2$$

above inferences will add edges

$$\begin{aligned} &(\bar{x}_1, x_3, \bar{x}_2), \quad (\bar{x}_1, x_4, \bar{x}_2), \\ &(\bar{x}_2, x_4, \bar{x}_3), \quad (\bar{x}_2, x_1, \bar{x}_3), \\ &(\bar{x}_3, x_1, \bar{x}_4), \quad (\bar{x}_3, x_2, \bar{x}_4), \\ &(\bar{x}_4, x_2, \bar{x}_1), \quad (\bar{x}_4, x_3, \bar{x}_1). \end{aligned}$$

For general constraint e.g., $C = a_i^\sigma x_i^\sigma + \cdots + a_n^\sigma x_n^\sigma \geq A$ where $\sum_{i=1}^n a_i^\sigma = m$, their approach is to add only small portion of edges to the graph, namely:

- nodes $x_i^\sigma, x_i^{1-\sigma} \in V$ if and only if $x_i^\sigma \in C$;
- edges $e = (x_i^{1-\sigma}, x_j^\sigma, \{\}) \in E$ if and only if $x_i^\sigma \in C$ and one of the following two cases occurs:
  1. $m - a_i^\sigma = A$, meaning that $x_i^\sigma := 0$ implies $x_j^\sigma := 1, \forall j \neq i$;
  2. $m - a_i^\sigma - a_j^\sigma < A$, meaning that x_i^σ, x_j^σ cannot be both set to false in order to satisfy C.

Actually, deciding the corresponding edge on an implication graph w.r.t a constraint equivalent to performing probing on certain variables and detecting whether this will fix other variables' value. The principle for general constraints above is analogous to the strong and weak Simple probing on a single constraint and we could add more edges indicated by a general constraint with only small extra effort.

Formally, considering a general constraint C in form:

$$C \doteq \quad \sum_{i,\sigma} a_k^\sigma x_k^\sigma \geq A, \quad k \in N, \sigma \in \{0, 1\} \tag{3.11}$$

Given a (partial) assignment ρ, two variables $x_i^\sigma, x_j^\sigma \in C$, two indices sets $P, Q \subseteq N$ such that $x_P^\sigma, x_Q^\sigma \in C$, we have:

3. edges $e = (x_i^{1-\sigma}, x_j^\sigma, \{\}) \in E$ if and only if $x_i^\sigma \in C$ and $0 < slack(C, \rho(x_i^{1-\sigma})) < a_j^\sigma$;

4. edges $e = (x_i^{1-\sigma}, x_Q^\sigma, x_P^\sigma) \in E$ if and only if $x_i^\sigma \in C$ and all $x_j^\sigma \in x_Q^\sigma$ satisfy $0 < slack(C, \rho(\{x_i^{1-\sigma}\} \cup x_P^{1-\sigma})) < a_j^\sigma$.

The positive slack ensures the constraint is feasible under current assignment. The principle 3 indicates x_j^σ would be unit propagated by x_i^σ and principle 4 finds those x_j^σ that will be unit propagated to true if both x_i^σ and x_P^σ are falsified. However, simply enumerate all possible x_P^σ in principle 4 requires exponential amount of work; one of possible substitutions is to sort the unassigned variables in C based on the ascending order of their coefficients a_k^σ and find the smallest/largest P which satisfies the condition. This can be done in time $O(N + \log N)$, i.e., calculating the cumulative value of sorted a_k^σ and do binary search with $slack(C, \rho(x_i^{1-\sigma}))$, hence adding (partially) edges shown in 3 and 4 could be finished within $O(MN \log N)$.

For example, given following constraint C:

$$C \doteq \quad 2x_1 + 5x_2 + 4x_3 + 3x_4 \geq 2$$

with assignment $\rho(\bar{x}_1)$ and $slack(C, \rho(\bar{x}_1)) = 10$. Figure 3.4 illustrates the sorted variables and cumulative prefix coefficient sum, where we can add the edge $(\bar{x}_1, x_2, \{\bar{x}_3, \bar{x}_4\})$ then.

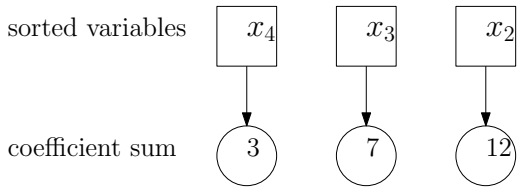


Figure 3.4: Sorted variables and cumulative prefix sum

3.3 CNF preprocessing beyond resolution

Given a constraint C and a formula F, if F and $F \wedge C$ are equal-satisfiable, we say C is *redundant with respect to* F [HKB17; BT19]. It is easy to detect whether a clause/constraint is redundant w.r.t a formula F by simply adding its negation to F and checking whether this leads to conflict such as

$$F \wedge \neg C \models \bot \tag{3.12}$$

i.e., there is no assignment ρ which $\rho(F) \wedge \rho(\neg C)$ evaluates to true. This is sufficient since any assignment where $\rho(F \wedge \neg C) = 1$ will imply $\rho(F) = 1$ and $\rho(C) = 0$, meaning that ρ satisfies F but falsifies C. A Pseudo-Boolean generalization of above redundancy is called *substitution redundancy* which appears on [GN21] as:

Definition 3.3.1 (Substitution redundancy)**.** Constraint C is redundant with respect to F if and only if there exists a *witness* ω, which is a substitution, such that:

$$F \wedge \neg C \models (F \wedge C)\restriction_\omega \tag{3.13}$$

i.e., every satisfying assignment for $F \wedge \neg C$ must satisfy $(F \wedge C)\restriction_\omega$. Based on the definition above, we present two redundancy removal-based presolving techniques for PBS: *Subsumption* and *Block Constraint Elimination*.

3.3.1 Subsumption

In SAT solving, a clause C is *subsumed* by another clause D if the literals in C are a superset of D [BJK21], which implies that every satisfying assignment for D must satisfy C as well. There are three different categories of subsumption:

- *Forward subsumption.* Given a clause C and a formula F, forward subsumption tests whether C is subsumed by F, i.e., subsumed by some clauses $D \in F$;
- *Backward subsumption.* Backward subsumption starts from a given clause D and tests whether there are some clauses $C \in F$ are subsumed by D; if so, C will be removed from F;
- *Self subsumption.* Self subsumption is to find those clauses $C \vee \ell$ and $D \vee \bar{\ell}$ such that resolving ℓ between these two clauses yields C, if so, D subsumes C and the clause $C \vee \ell$ can be replaced by C.

In Pseudo-Boolean setting, consider two distinguished constraints C and D, say C is *subsumed* by D if C can be derived from D by adding *Literal axioms*:

$$\frac{}{x_i^\sigma \geq 0} \tag{3.14}$$

specifically, we have $1 \geq 0$ or equivalently $0 \geq -1$. This is also known as C is *syntactically implied* by D [GN21]. For example, consider following two constraints:

$$\begin{aligned} C \doteq &\quad 3x_1 + 2x_2 + x_3 \geq 2 \\ D \doteq &\quad 3x_1 + x_2 + x_3 \geq 3 \end{aligned}$$

where constraint C is subsumed by D by adding axioms $x_2 \geq 0$ and $0 \geq -1$ to D.

A simple observation is that one constraint which is subsumed by another always has non-larger degree since adding literal axioms will not increase the degree of a constraint, while this inference is not sound, e.g.,

$$\begin{aligned} C \doteq &\quad x_1 + x_2 \geq 1 \\ D \doteq &\quad x_1 + x_3 \geq 1 \end{aligned}$$

Therefore, given index sets $I, U, V \subseteq N$ where $I \cap U \cap V = \emptyset$ and two constraints C and D in form:

$$C \doteq \quad a_I^\sigma x_I^\sigma + a_U^\sigma x_U^\sigma \geq A \tag{3.15}$$

$$D \doteq \quad b_I^\sigma x_I^\sigma + b_V^\sigma x_V^\sigma \geq B \tag{3.16}$$

denote $d = \sum_{i \in I}(b_i^\sigma - a_i^\sigma)$, then one of the possible methods to detect if C is subsumed (syntactically implied) by D is to check whether

$$A = \begin{cases} B - \sum_{v \in V} b_v^\sigma & , d \leq 0 \\ B - \sum_{v \in V} b_v^\sigma - d & , d > 0 \end{cases} \tag{3.17}$$

holds or not. Since every syntactical transformation step from constraint D to whom it consumes C only leads to increase of the variable number or non-increase of the degree (remove a variable equivalent to adding its negation), therefore every satisfying assignment for D must satisfy C and we can discard the latter without influence the satisfiability of the formula.

Theorem 3.3.1. Discard the subsumed constraint preserves the optimality.

Proof. Proof by contradiction. Given two constraints D and C where $D \vDash C$ and formula $F_1 \doteq F \wedge D$ and $F_2 \doteq F \wedge D \wedge C$ with same objective function f, F_1 obtains its optimal O_1 by assignment ρ_1 and F_2 obtains its optimal O_2 by assignment ρ_2. Assume $O_1 < O_2$. By property of subsumption we have ρ_1 satisfies F_2 as well, since they have same objective function, then the objective value $f\restriction_{\rho_1}$ for F_2 would equal to O_1, contradicting to $O_1 < O_2$. Proof finish. □

Another method to detect the subsumption is to check whether C is implied by D using the redundancy detection 3.12, i.e., check whether $D \models C$ holds or not. This is equivalent to examine

$$(D \wedge \neg C) \models \bot \tag{3.18}$$

3.3.2 Blocked clause elimination

Given a constraint C and formula F with all constraints encoded in normalized form, we have the following observations:

Observation 3.3.2. C is redundant w.r.t F if C has form:

$$a_W^\sigma x_W^\sigma + a_U^\sigma x_U^\sigma \geq A \tag{3.19}$$

where $W, U \subseteq N, W \cap U = \emptyset, x_W^\sigma \notin F$ and $\sum_{w \in W} a_W^\sigma \geq A$.

Proof. Let the witness $\omega = \{x_w \mapsto 1 | \forall w \in W\}$. Now $C\restriction_\omega$ trivially satisfied, since any satisfying assignment for $F \wedge \neg C$ satisfies F as well, then the proof follows. □

Here x_W^σ are group of Pure variables with respect to F and it is always not worse to set them to true even if $\sum_{w \in W} a_w^\sigma < A$.

Observation 3.3.3 (Block Clause Elimination)**.** Given a clause constraint C:

$$C \doteq \quad x_i^\sigma + x_W^\sigma + x_U^\sigma \geq 1 \tag{3.20}$$

where $\{i\}, W, U \subseteq N, W \cap U = \emptyset$, and a formula F contains only clause constraints. Then C is redundant w.r.t F if for all constraints $\{D\} \in F$ where $x_i^{1-\sigma} \in D$, we have:

$$D \doteq \quad x_i^{1-\sigma} + x_j^{1-\sigma} + x_V^\sigma \geq 1 \tag{3.21}$$

where $V \subseteq N, j \in W$ and $\bigcup\{j\} = W$, i.e., perform canceling addition between C and D on x_i^σ will generate tautology (x_j^σ and $x_j^{1-\sigma}$).

Proof. Let the witness $\omega = \{x_i^\sigma \mapsto x_i^{1-\sigma}\}$. Any satisfiable assignment ρ to $F \wedge \neg C$ will set all variables in C to false, i.e., $\rho(x_k^\sigma) = 0, \forall x_k^\sigma \in C$, therefore constraints $C\restriction_\omega$ and $(F \setminus \{D\}) \ni x_i^\sigma$ are satisfied by $x_i^{1-\sigma}$. Constraints $\{D\} \in F\restriction_\omega$ are satisfied since $\rho(x_j^{1-\sigma}) = 1$ by construction. By definition 3.3.1 we finish the proof. □

Note that above witness ω is equivalent to

$$\omega' = \{x_i^\sigma \mapsto 1, x_i^{1-\sigma} \mapsto 0, x_j^\sigma \mapsto 0, x_j^{1-\sigma} \mapsto 1 | j \in W\}$$

Based on this observation, we could generalize the *Blocked Clause Elimination (BCE)* into Pseudo-Boolean configuration, which was first proposed in SAT community by [Kul99] and later used as a preprocessing technique in [JBH10b].

Theorem 3.3.4 (Blocked Constraint Elimination)**.** A normalized constraint C is redundant w.r.t a formula F where all constraints are encoded in normalized form if C is in form:

$$C \doteq \quad kx_i^\sigma + a_W^\sigma x_W^\sigma + a_U^\sigma x_U^\sigma \geq A \tag{3.22}$$

and for all constraints $\{D\} \in F$ where $x_i^{1-\sigma} \in D$ (w.l.o.g, we assume the coefficient of $x_i^{1-\sigma}$ is k), they are in form:

$$D \doteq \quad kx_i^{1-\sigma} + b_P^{1-\sigma} x_P^{1-\sigma} + a_V^\sigma x_V^\sigma \geq B \tag{3.23}$$

where $\{i\}, W, P, U, V \subseteq N, P \subseteq W, \bigcup P = W, \{i\} \cap W \cap \{U \cup V\} = \emptyset$, meanwhile $k \geq A, \sum_{p \in P} b_p^{1-\sigma} \geq B$ and $x_W^\sigma \notin F \setminus \{D\}$. Then we say C is a *block constraint* w.r.t F and *blocked* by x_i^σ

Proof. Let the witness $\omega = \{x_i^\sigma \mapsto 1, x_i^{1-\sigma} \mapsto 0, x_j^\sigma \mapsto 0, x_j^{1-\sigma} \mapsto 1 | j \in W\}$. Every satisfying assignment for $F \wedge \neg C$ would always satisfy $(F \wedge C)\restriction_\omega$ as well, since $C\restriction_\omega$ and $D\restriction_\omega$ after substitution are already satisfied by construction (by x_i^σ and $x_P^{1-\sigma}$ respectively), and substitute x_i^σ and $x_j^{1-\sigma}, j \in W$ to true in F never worse. Meanwhile, $x_W^\sigma \notin F \setminus \{D\}$ and substitute them to false has no influence on the satisfiability of constraints $F \setminus \{D\}$. Proof finish. □

Actually, C could remain redundant even if there are some constraints $E \subseteq F \setminus \{D\}$ contain x_W^σ, the idea here is that E needs to contain x_i^σ meanwhile its coefficient should be large enough to "compensate" substitute x_P^σ to false, which leads to the following corollary:

Corollary 3.3.4.1. Given constraints C and D have form defined in Theorem 3.3.4 but without restriction $x_W^\sigma \notin F \setminus \{D\}$, then C remain redundant w.r.t. F if all constraints $E \subseteq F \setminus \{D\}$ that contain variables $x_P^\sigma, P \subseteq W$, have the form:

$$E \doteq \quad a_i^\sigma x_i^\sigma + a_P^\sigma x_P^\sigma + a_Q^\sigma x_Q^\sigma \geq K$$

where $Q \subseteq N, Q \cap \{i\} \cap W = \emptyset$ and $a_i^\sigma \geq \sum_{p \in P} a_p^\sigma + K$.

For example, consider a constraint C as:

$$3x_1 + 2x_2 + x_3 + x_4 \geq 2$$

and a formula F as:

$$\begin{aligned} 3\bar{x}_1 + 2\bar{x}_2 + x_5 &\geq 2 \\ 3\bar{x}_1 + \bar{x}_3 + x_6 + x_7 &\geq 1 \\ 6x_1 + 2x_2 + x_3 &\geq 3 \\ x_4 + x_5 + x_6 + x_7 &\geq 3 \end{aligned}$$

set the witness $\omega = \{x_1 \mapsto 1, x_2 \mapsto 0, x_3 \mapsto 0\}$ and we have $(F \wedge C)|_\omega$ to be:

$$\begin{aligned} x_4 &\geq -1 \\ x_5 &\geq 0 \\ x_6 + x_7 &\geq 0 \\ 0 &\geq -3 \\ x_4 + x_5 + x_6 + x_7 &\geq 3 \end{aligned}$$

clearly, any satisfying assignment for $F \wedge \neg C$ must satisfy $x_4 + x_5 + x_6 + x_7 \geq 3$, which is also the only non-trivial constraint in $(F \wedge C)|_\omega$. In this case, C is a blocked constraint w.r.t F which is blocked by x_1.

Similarly to the Subsumption, the BCE preserve the satisfiability of the formulas since every satisfying assignment for $F \wedge \neg C$ must satisfy $(F \wedge C)|_\omega$. The proof of optimality preservation is analogous to the proof of 3.3.1.

Chapter 4

MIP preprocessing

Compared with the SAT preprocessing that tries to derive "stronger" constraints from the existing ones (partially because of the nature of various proof systems) where the formula size may increase in general, the MIP preprocessing techniques emphasize exploring the bound properties of the current variables and constraints, such as bound lifting, probing, parallel rows/columns detection, etc. In most cases, the presolving does not introduce new constraints but tries to fix certain variables [1] or remove category constraints, which generally results in a reduction in the size of the input formula.

The division of the following techniques rises from [Ach+20]. We will start from preprocessing focus on individual constraints and variables to those applied to multiple ones. The section ended with the presolving techniques based on the structure of the whole problem. Since both PBS and PBO are strictly included in MIP, the adaption of these techniques needs to pay more attention to the preservation of binary variables instead.

4.1 Reduction for individual constraints

4.1.1 Model cleanup and removal of redundant constraints

The MIP version is from [Ach+20] and in Pesudo-Boolean setting, given a general constraint C:

$$\sum_{i\in[n],\sigma\in\{0,1\}} a_i^\sigma \geq A \tag{4.1}$$

if $\sum_{i,\sigma} a_i^\sigma < A$ the problem is infeasible, since the maximum slack of the constraint is

$$slack(C, \rho(\emptyset)) = \sum_{i\in[n]} a_i^\sigma - A < 0 \tag{4.2}$$

and there are no variables that can be propagated.

[1] In Pseudo-Boolean setting, any non-trivial bound strengthening of a variable, e.g., $0 \to \lceil \frac{1}{2} \rceil \to 1$, results in a value fix for that variable.

4.1.2 Bound strengthening

This technique that removes the redundant bound for a decision variable is also known as *Domain Propagation*, see [Sav94; FM05b; Ach07]. Given a general constraint C in form:

$$a_i^\sigma x_i^\sigma + \sum_{j,\sigma} a_j^\sigma x_j^\sigma \geq A, \quad i,j \in [n], j \neq i, \sigma \in \{0,1\} \tag{4.3}$$

we can lift the lower bound of x_i^σ by:

$$x_i^\sigma \geq \lceil \frac{A - \sum_{j,\sigma} a_j^\sigma x_j^\sigma}{a_i^\sigma} \rceil \geq \lceil \frac{A - \sum_j a_j^\sigma}{a_i^\sigma} \rceil \tag{4.4}$$

which is equivalent to propagate x_i^σ to true if the maximal slack of C is smaller than a_i^σ, namely:

$$0 \leq slack(C, \rho(\emptyset)) = a_i^\sigma + \sum_j a_j^\sigma - A < a_i^\sigma \tag{4.5}$$

Similarly, for a reverse general constraint D:

$$a_i^\sigma x_i^\sigma + \sum_{j,\sigma} a_j^\sigma x_j^\sigma \leq A, \quad i,j \in [n], j \neq i, \sigma \in \{0,1\} \tag{4.6}$$

we could derive an upper bound for x_i^σ as:

$$x_i^\sigma \leq \lfloor \frac{A - \sum_{j,\sigma} a_j^\sigma x_j^\sigma}{a_i^\sigma} \rfloor \leq \lfloor \frac{A}{a_i^\sigma} \rfloor \tag{4.7}$$

4.1.3 Coefficient strengthening

In MIP preprocessing, the *Coefficient Strengthening* [Sav94] modifies the coefficients of a constraint such that the LP relaxation of the problem gets tighter inside the box defined by the bounds of the variable meanwhile, the domain of integer solution remains unchanged. This arises the definition of the *Constraint Domination* [Ach+20], which is:

Definition 4.1.1 (Constraint Domination)**.** Given two constraints $ax \leq b$ and $\tilde{a}x \leq \tilde{b}$ where $a, \tilde{a} \in \mathbb{R}^n, b, \tilde{b} \in \mathbb{R}$ and x are either integer variables or continuous variables bounded by $[l, u]$. Say $ax \leq b$ dominates $a'x \leq b'$ if

$$\{x \in \mathbb{R}^n | x \in [l,u], ax \leq b\} \subset \{x \in \mathbb{R}^n | x \in [l,u], \tilde{a}x \leq \tilde{b}\}$$

In Pseudo-Boolean setting, consider the reverse general constraints in form:

$$a_i^\sigma x_i^\sigma + \sum_{j,\sigma} a_j^\sigma x_j^\sigma \leq A, \quad i,j \in [n], j \neq i, \sigma \in \{0,1\} \tag{4.8}$$

we can strengthen the coefficient of x_i^σ if $a_i^\sigma \geq d$ where $d := A - \sum_{j,\sigma} a_j^\sigma > 0$ with

$$(a_i^\sigma - d)x_i^\sigma + \sum_{j,\sigma} a_j^\sigma x_j^\sigma \leq A - d \tag{4.9}$$

and constraint 4.9 dominates initial one 4.8 in subspace $x_i^\sigma \in \{0,1\}$. Because when $x_i^\sigma = 1$ two constraints are identical and when $x_i^\sigma = 0$, constraint 4.8 and 4.9 can be read as $\sum_{j,\sigma} a_j^\sigma x_j^\sigma \leq A$ and $\sum_{j,\sigma} a_j^\sigma x_j^\sigma \leq A - d$, which shows the modified constraint still remains redundant.

4.1.4 Chvatal-Gomory strengthening of inequalities

The idea of *Chvatal-Gomory Strengthening* [Gom63; Chv73b] is to multiply a positive real number $s \in \mathbb{R}$ on both sides of a constraint and check whether this constraint could be tighter. Specifically, consider following general constraint C:

$$\sum_{i,\sigma} a_i^\sigma x_i^\sigma \geq A, \quad i \in [n], \sigma \in \{0,1\} \tag{4.10}$$

if following two conditions both hold:

$$\lceil a_i^\sigma \cdot s \rceil A / \lceil A \cdot s \rceil \leq a_i^\sigma, \quad \forall x_i^\sigma \in C \tag{4.11}$$
$$\lceil a_j^\sigma \cdot s \rceil A / \lceil A \cdot s \rceil < a_j^\sigma, \quad \exists x_j^\sigma \in C \tag{4.12}$$

then we can replace C as:

$$\sum_{i,\sigma} \lceil a_i^\sigma \cdot s \rceil x_i^\sigma \geq \lceil A \cdot s \rceil, \quad i \in [n], \sigma \in \{0,1\} \tag{4.13}$$

This is equivalent to applying the generalized division rule. For example:

$$\begin{aligned} & 3x_1 + 2x_2 \leq 4 \\ \xrightarrow{s=\frac{1}{3}} \quad & x_1 + x_2 \leq 2 \end{aligned}$$

4.1.5 Euclidean reduction

Given a general constraint C:

$$\sum_{i,\sigma} a_i^\sigma x_i^\sigma \geq A, \quad i \in [n], \sigma \in \{0,1\} \tag{4.14}$$

the *Euclidean reduction* [Ach+20] divides the *greatest common divisor (gcd)* of the above coefficients $d = gcd(a_i^\sigma), i \in [n]$ on the constraints, which can be treated as a particular case of application of generalized division rule with divisor $1/d$.

4.1.6 Simple probing on a single constraint

The idea of *simple probing* [Ach+20] is to temporally fix the value of some binary variables and test whether this will imply the value for others.

The *strong* version of simple probing on a single constraint is to only consider the equal general constraint in form:

$$a_i^\sigma x_i^\sigma + \sum_{j,\sigma} a_j^\sigma x_j^\sigma = A, \quad i,j \in [n], i \neq j, \sigma \in \{0,1\} \tag{4.15}$$

if $a_i^\sigma + \sum_j a_j^\sigma = 2A$ and $a_i^\sigma = A$, then fix the value of x_i^σ would cause a propagation on all x_j^σ, namely:

$$x_i^\sigma = 0 \to x_j^\sigma = 1, \quad \forall j \neq i \tag{4.16}$$

$$x_i^\sigma = 1 \to x_j^\sigma = 0, \quad \forall j \neq i \tag{4.17}$$

consequently, we can make the substitution: $x_j^\sigma = 1 - x_i^\sigma, \forall j \neq i$.

The *weak* version of simple probing considers a general constraint C:

$$a_i^\sigma x_i^\sigma + \sum_{j,\sigma} a_j^\sigma x_j^\sigma \geq A, \quad i,j \in [n], i \neq j, \sigma \in \{0,1\} \tag{4.18}$$

if both two conditions below hold:

$$slack(\sum_{j,\sigma} a_j^\sigma x_j^\sigma \geq A, \rho(\emptyset)) \geq A \tag{4.19}$$

$$slack(\sum_{j,\sigma} a_j^\sigma x_j^\sigma \geq A, \rho(x_k^{1-\sigma})) < A, \quad k = \arg\min_j a_j^\sigma \tag{4.20}$$

then setting $x_i^\sigma = 0$ will propagate all $x_j^\sigma = 1$.The second condition makes sure that falsify arbitrary variable will falsify the constraint. For instance, in constraint below:

$$3x_1 + 2x_2 + 2x_3 \geq 4$$

falsify arbitrary variables will propagate others to true.

4.1.7 Doubleton equation substitution

The definition of *doubleton equation* was proposed in [BBG83], e.g., an equal general constraint consists exactly two distinct variables:

$$a_i^\sigma x_i^\sigma + a_j^\sigma x_j^\sigma = A \tag{4.21}$$

where $i \neq j, \sigma \in \{0,1\}$. Given a doubleton equation C, one can always substitute one variable, say $x_i^\sigma \in C$, by the other x_j^σ and discard this equation. The resulting is a

modified bound on x_j^σ:

$$l_j^\sigma \leq x_i^\sigma = \frac{A - a_j^\sigma x_j^\sigma}{a_i^\sigma} \leq u_j^\sigma \tag{4.22}$$

$$\iff \frac{A - a_i^\sigma u_j^\sigma}{a_j^\sigma} \leq x_j^\sigma \leq \frac{A - a_i^\sigma l_j^\sigma}{a_j^\sigma} \tag{4.23}$$

substitute $l_j^\sigma = 0$ and $u_j^\sigma = 1$ we have:

$$\frac{A - a_i^\sigma}{a_j^\sigma} \leq x_j^\sigma \leq \frac{A}{a_j^\sigma} \tag{4.24}$$

which is equivalent to perform Probing over x_i^σ on C and strengthen the bound of x_j^σ accordingly.

4.1.8 Simplify inequalities

This approach removes the "unnecessary" variables in a constraint aggressively and derive other possible reasoning constraints [Bes+21]. Concretely, given a reverse general constraint $C \ni x_S^\sigma$ in form:

$$\sum_{i,\sigma} a_i^\sigma x_i^\sigma \leq A \tag{4.25}$$

for all $i \in S, \sigma \in \{0, 1\}$, where S is an ordered set sorted by the decreasing order of absolute value of the coefficients. Then we find a partition $S = L \cup R$ where $|a_l^\sigma| > |a_r^\sigma|$ hold for all $l \in L, r \in R$ and seek for a minimized $|L|$ such that

$$\sum_{j \in L, \sigma \in \{0,1\}} a_j^\sigma \geq A \tag{4.26}$$

then we derive a new constraint:

$$\sum_{j \in L, \sigma \in \{0,1\}} \left(\frac{a_j^\sigma}{d}\right) x_j^\sigma \leq \lfloor \frac{A}{d} \rfloor \tag{4.27}$$

where d is the greatest common divisor for $a_j^\sigma, j \in L$. For example,

$$\begin{aligned} & 15x_1 + 15x_2 + 7x_3 + 3x_4 + x_5 \leq 26 \\ \iff \quad & 15x_1 + 15x_2 \leq 26 \\ \iff \quad & x_1 + x_2 \leq 1 \end{aligned}$$

If this is not always possible, then we set d to be the greatest common divisor for all coefficient $a_i^\sigma \in S$ and simplify the degree of C to be $\lfloor \frac{A}{d} \rfloor \cdot d$.

4.1.9 Singleton columns substitution

The definition of *singleton column* appears in [AA95] which is a column x_j such that:

$$\exists(j,k) : a_{ij} = 0, \forall i \neq k, a_{kj} \neq 0 \tag{4.28}$$

i.e., have only one non-zero entry in the constraint matrix. In Pseudo-Boolean setting, this is equivalent to a variable x_i^σ which is contained in only one constraint. A *free column singleton* is a column singleton with infinite lower and upper bounds, which can be substituted out of the problem if it appears within an equation:

$$a_{ij}x_j + \sum_{k \neq j} a_{ik}x_k = b_i \tag{4.29}$$

as

$$x_j = \frac{b_i - \sum_{k \neq j} a_{ik}x_k}{a_{ij}} \tag{4.30}$$

since x_j can take an arbitrary value. Furthermore, if x_j is *implied-free*[2] on both sides, it can be replaced using 4.30 above as well; if only one side not implied, say u_j, we can lose the LHS of the constraint as:

$$b_i - u_j a_{ij} \leq \sum_{k \neq j} a_{ik}x_k \leq b_i \tag{4.31}$$

In Pseudo-Boolean setting, since all variables are bounded, above constraint 4.31 degenerates into:

$$A - a_i^\sigma \leq \sum_{j \neq i, \sigma} a_j^\sigma x_j^\sigma \leq A \tag{4.32}$$

which is identical to weakening the singleton variable x_i^σ and deriving the bound for the sum of other variables.

4.2 Reduction for individual variables

4.2.1 Dual fixing and bound strengthening

In the rest of this section we assume all constraints are raw Pseudo-Boolean constraints as defined in 2.1 with equality or less-or-equal operator.

For *dual fixing* [Ach+20], denote a variable x_i^σ which does not appear in any equations and x_i^σ has objective coefficient $w_i^\sigma \geq 0$ and constraint coefficients $a_i^\sigma \geq 0$ holds for all constraints where it is included, then x_i^σ can be fixed to 0 if its negation $x_i^{1-\sigma}$ has objective coefficient $w_i^{1-\sigma} \leq 0$ and all coefficients $a_i^{1-\sigma} \leq 0$. Since after the optimal

[2] The implied bound of x_j by other constraints is within the variable initial bound.

solution for the presolved formula has been found, one can always find a value for x_i^σ that satisfies all constraints.

If we are unable to fix a variable, we can obtain tighter bounds using dual arguments. Specifically, given a variable x_i^σ where $w_i^\sigma \geq 0$ and constraint C_j with less-or-equal operator, denote that:

$$M^+ = \{j \in M : x_i^\sigma \in C_j, a_i^\sigma > 0\} \tag{4.33}$$

$$M^- = \{j \in M : x_i^\sigma \in C_j, a_i^\sigma < 0\} \tag{4.34}$$

then for all $j \in M^+ \cup M^-$, if $x_i^\sigma := 0$ causes all constraints in M^- redundant [3], i.e.,

$$\sum_{k \neq i} a_k^\sigma \leq A, \quad x_k^\sigma \in C_j \tag{4.35}$$

then x_i^σ is redundant for the feasibility of C_j and we can fix it to false.

4.2.2 Substitute implied free variables

Using the Bound strengthening we may derive a lower and upper bound for x_i^σ, denote l' and u' the tightest bound for x_i^σ, say x_i^σ is an *implied free variable* if $[l', u'] \subseteq [0, 1]$ [Ach+20].

Now consider an equality constraint:

$$a_i^\sigma x_i^\sigma + \sum_{j,\sigma} a_j^\sigma x_j^\sigma = A \quad i, j \in [n], i \neq j, \sigma \in \{0, 1\} \tag{4.36}$$

if x_i^σ is an implied free variable, meanwhile $a_j^\sigma / a_i^\sigma \in \mathbb{Z}$ hold for all $j \neq i$, then we can substitute x_i^σ out of the formula by:

$$x_i^\sigma = (A - \sum_{j,\sigma} a_j^\sigma x_j^\sigma)/a_i^\sigma \tag{4.37}$$

and the formula is infeasible if $A/a_i \notin \mathbb{Z}$.

4.3 Reductions that consider multiple constraints at the same time

4.3.1 Parallel and nearly parallel rows

Given two PB constraints C and D:

$$C \doteq \sum_{i,\sigma} a_i^\sigma x_i^\sigma \circ A \tag{4.38}$$

$$D \doteq \sum_{i,\sigma} b_i^\sigma x_i^\sigma \circ B \tag{4.39}$$

[3] for $j \in M^+$, setting $x_i^\sigma := 0$ never worse

where $i \in [n], \sigma \in \{0,1\}, \circ \in \{\geq, =\}$. Say C and D are *parallel* [AA95] if exists a real number $s \in \mathbb{R}, s \neq 0$ such that $a_i^\sigma = sb_i^\sigma$ holds. If two constraints are parallel, then following inference hold:

1. If both C and D are equal constraints:

$$C \doteq \sum_{i,\sigma} a_i^\sigma x_i^\sigma = A \tag{4.40}$$

$$D \doteq \sum_{i,\sigma} b_i^\sigma x_i^\sigma = B \tag{4.41}$$

   then D can be discarded if $A = sB$. The problem is infeasible if $A \neq sB$.

2. If exactly one constraint is an equation:

$$C \doteq \sum_{i,\sigma} a_i^\sigma x_i^\sigma = A \tag{4.42}$$

$$D \doteq \sum_{i,\sigma} b_i^\sigma x_i^\sigma \geq B \tag{4.43}$$

   then D can be discarded if $s > 0, A \geq sB$ or $s < 0, A \leq sB$ holds.

3. If both constraints are inequalities:

$$C \doteq \sum_{i,\sigma} a_i^\sigma x_i^\sigma \geq A \tag{4.44}$$

$$D \doteq \sum_{i,\sigma} b_i^\sigma x_i^\sigma \geq B \tag{4.45}$$

   then D can be discarded if $s > 0, A \geq sB$ and symmetrically if $s > 0, A \leq sB$, then C can be discarded. If $s < 0, sB < A$, the two constraints can be merged into a *ranged constraint* in form:

$$sB \leq \sum_{i,\sigma} a_i^\sigma x_i^\sigma \leq A \tag{4.46}$$

   or if $s < 0, sB = A$, we will have an equal constraint:

$$\sum_{i,\sigma} a_i^\sigma x_i^\sigma = A \tag{4.47}$$

If two constraints are *nearly parallel*, we can still perform such detection. Two constraints C and D are said to be nearly parallel if they contain a different singleton variable each, plus two set of identical variables, such that:

$$C \doteq a_i^\sigma x_i^\sigma + \sum_{k,\sigma} a_k^\sigma x_k^\sigma \circ A \tag{4.48}$$

$$D \doteq a_j^\sigma x_j^\sigma + \sum_{k,\sigma} b_k^\sigma x_k^\sigma \circ B \tag{4.49}$$

where $i, j, k \in [n], i \neq j \neq k, \circ \in \{\geq, =\}$ and $\exists s \in \mathbb{R}, s \neq 0$ such that $a_k^\sigma = sb_k^\sigma$ holds for all k. If C and D are nearly parallel, then following inferences hold:

1. If both constraints are equations:

$$C \doteq a_i^\sigma x_i^\sigma + \sum_{k,\sigma} a_k^\sigma x_k^\sigma = A \tag{4.50}$$

$$D \doteq a_j^\sigma x_j^\sigma + \sum_{k,\sigma} b_k^\sigma x_k^\sigma = B \tag{4.51}$$

then we can substitute $x_j^\sigma := tx_i^\sigma + d$ where $t = a_i^\sigma/(sa_j^\sigma)$ and $d = (sB - A)/(sa_j^\sigma)$. And we can tighten the bounds of x_i^σ with relation $x_i^\sigma = (x_j^\sigma - d)/t$ as:

$$l_i^\sigma := \max\{0, -d/t\} \quad \text{and} \quad u_i^\sigma := \min\{1, (1-d)/t\}, \quad t > 0 \tag{4.52}$$

$$l_i^\sigma := \max\{0, (1-d)/t\} \quad \text{and} \quad u_i^\sigma := \min\{1, -d/t\}, \quad t < 0 \tag{4.53}$$

The formula is infeasible if the updated bound $l_i^\sigma > u_i^\sigma$. Furthermore, the two constraints are parallel after substitution and we can discard D then.

2. If only C is an equation and contains a singleton variable x_i^σ such that:

$$C \doteq a_i^\sigma x_i^\sigma + \sum_{k,\sigma} a_k^\sigma x_k^\sigma = A \tag{4.54}$$

$$D \doteq \sum_{k,\sigma} b_k^\sigma x_k^\sigma \geq B \tag{4.55}$$

then we can tighten the bounds of x_i^σ by:

$$l_i^\sigma := \max\{0, (A - sB)/a_i^\sigma\}, \quad sa_i^\sigma < 0 \tag{4.56}$$

$$u_i^\sigma := \min\{1, (A - sB)/a_i^\sigma\}, \quad sa_i^\sigma > 0 \tag{4.57}$$

D is redundant after strengthening the bound of x_i^σ and can be discarded then.

3. If both constraints are inequalities:

$$C \doteq a_i^\sigma x_i^\sigma + \sum_{k,\sigma} a_k^\sigma x_k^\sigma \geq A \tag{4.58}$$

$$D \doteq a_j^\sigma x_j^\sigma + \sum_{k,\sigma} b_k^\sigma x_k^\sigma \geq B \tag{4.59}$$

where $a_i^\sigma = sa_j^\sigma, A = sB$, then we can substitute $x_j^\sigma := x_i^\sigma$ and discard D.

4.3.2 Non-zero cancellation

In MIP, the *non-zero cancellation*, also known as *sparsify*, decreases the number of non-zero entries in the coefficient matrix by adding an equation to an inequality [CM93]. Specifically, in Pseudo-Boolean setting, consider an equation constraint C and a general constraint D:

$$C \doteq \sum_{i\in I,\sigma} a_i^\sigma x_i^\sigma + \sum_{j\in J,\sigma} a_j^\sigma x_j^\sigma + \sum_{u\in U,\sigma} a_u^\sigma x_u^\sigma = A \tag{4.60}$$

$$D \doteq \sum_{i\in I,\sigma} b_i^\sigma x_i^\sigma + \sum_{j\in J,\sigma} b_j^\sigma x_j^\sigma + \sum_{v\in V,\sigma} b_v^\sigma x_v^\sigma \geq B \tag{4.61}$$

where $I, J, U, V \subseteq N, I\cap J\cap U\cap V = \emptyset, \sigma \in \{0,1\}$ and assume $\exists s \in \mathbb{R} : sa_i^\sigma = b_i^\sigma, sa_j^\sigma \neq b_j^\sigma$ hold for all $i \in I, j \in J$ above. Then we can remove x_I^σ out of the D by subtracting sC [4] to D as:

$$D' \doteq \sum_{j\in J,\sigma} (b_j^\sigma - sa_j^\sigma)x_j^\sigma - \sum_{u\in U,\sigma} sa_u^\sigma x_u^\sigma + \sum_{v\in V,\sigma} b_v^\sigma x_v^\sigma \geq B - sA \tag{4.62}$$

the non-zero coefficients in the matrix are reduced by $|I| - |U|$ and in this case the transformation is applied if $|I| > |U|$.

In the Pseudo-Boolean setting, we can reduce more non-zero entries by perform canceling addition overall x_i^σ. Assume a general constraint E:

$$E \doteq \sum_{i\in I,\sigma} b_i^{1-\sigma} x_i^{1-\sigma} + \sum_{j\in J,\sigma} b_j^\sigma x_j^\sigma + \sum_{v\in V,\sigma} b_v^\sigma x_v^\sigma \geq K \tag{4.63}$$

with $I, J, U, V \subseteq N, I \cap J \cap U \cap V = \emptyset, \sigma \in \{0,1\}$ and $\exists s \in \mathbb{R} : sa_i^\sigma = b_i^{1-\sigma}$ hold for all $i \in I$. One can rule out the $x_i^{1-\sigma}$ by perform cancelling addition on all $x_i^{1-\sigma}$ at once, namely:

$$E' \doteq \sum_{j\in J,\sigma} (sa_j^\sigma + b_j^\sigma)x_j^\sigma + \sum_{u\in U,\sigma} a_u^\sigma x_u^\sigma + \sum_{v\in V,\sigma} b_v^\sigma x_v^\sigma \geq K + sA - \sum_{i\in I,\sigma} b_i^{1-\sigma} \tag{4.64}$$

Still, the non-zero coefficient in the matrix reduced by $|I| - |U|$.

4.3.3 Bound and coefficient strengthening

The basic idea of *Bound and coefficient strengthening* are presented in [Sav94]. Recall that in Bound strengthening we can derive bounds for a variable x_i^σ from general constraints and reverse general constraints C which it is involved, namely:

$$a_i^\sigma x_i^\sigma + \sum_{j,\sigma} a_j^\sigma x_j^\sigma \circ A, \quad \circ \in \{\geq, \leq\} \tag{4.65}$$

$$\implies \quad \lceil \frac{A - \sum_j a_j^\sigma}{a_i^\sigma} \rceil \leq x_i^\sigma \leq \lfloor \frac{A}{a_i^\sigma} \rfloor \tag{4.66}$$

[4] sC is just a shorthand for multiplying s on both sides of C

where $i \neq j, \in [n], \sigma \in \{0, 1\}$. This actually comes from the bound

$$0 \leq \sum_{j,\sigma} a_j^\sigma x_j^\sigma \leq \sum_j a_j^\sigma \tag{4.67}$$

The idea for bound strengthening with multiple row is to find the tightest lower bound of $\sum_{j,\sigma} a_j^\sigma x_j^\sigma$ implied among different rows. Since we assume all coefficient $a_j^\sigma \in \mathbb{N}$, then the analysis of lower bound is trivial, in turn we seek for a maximized upper bound, correspondingly,

$$x_i^\sigma \geq \max . \lceil \frac{A - \sum_j a_j^\sigma}{a_i^\sigma} \rceil \tag{4.68}$$

This could be expensive to calculate such bounds for every constraint; one compromise method uses the maximized upper bound among only a specific group of constraints.

4.3.4 Clique merging

A *conflict graph* [ANS00] is a graph which has a node for each binary variable and its negation, there is an edge for two nodes if and only if setting one true will imply the other to false. One specific type of constraints whose corresponding conflict graph forms a clique is the *set packing/partitioning constraint* [Ach+20]:

$$\sum_{i,\sigma} x_i^\sigma + \sum_{j,\sigma} (1 - x_j^\sigma) \leq 1, \quad i, j \in [n], i \neq j, \sigma \in \{0, 1\} \tag{4.69}$$

Clearly, setting arbitrary $x_i^\sigma := 1$ will force all $x_j^\sigma := 0$, which suggests there are edges from every x_i^σ to all x_j^σ and thereby the corresponding conflict graph forms a clique. The idea of clique merging is to combine several set packing constraints into a single one.

This can be finished by firstly search for a larger clique which subsumes the existing one by add the variables according to the conflict graph [JP82; Sav94], once the new clique has been found we discard other constraints dominated by this clique. For example, consider following constraints:

$$\begin{aligned} &x_1 + x_2 \leq 1 \quad x_2 + x_3 \leq 1 \quad x_3 + x_4 \leq 1 \\ &x_1 + x_3 \leq 1 \quad x_2 + x_4 \leq 1 \\ &x_1 + x_4 \leq 1 \end{aligned}$$

The corresponding conflict graph G is shown in Figure 4.1.

Every constraint is a set packing constraint which corresponds to an edge of the "box" in G; clearly, a larger clique formed by nodes x_1, x_2, x_3, x_4 includes every small clique (a single edge), therefore we can derive the constraint:

$$x_1 + x_2 + x_3 + x_4 \leq 1 \tag{4.70}$$

This can be treated as adding x_3 and x_4 to $x_1 + x_2 \leq 1$ and we can then discard all constraints except the $x_1 + x_2 \leq 1$.

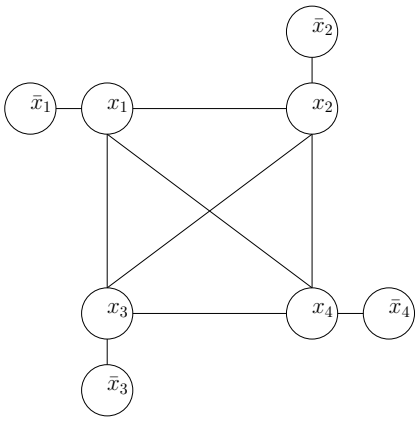


Figure 4.1: Conflict graph for set packing constraints

4.4 Reduction that considers multiple variables at the same time

4.4.1 Fix redundant penalty variables

A *penalty variable* [Ach+20] x_i^σ is a singleton variable such that $w_i^\sigma > 0$ and $a_i^\sigma > 0$ for all general constraints and $a_i^\sigma < 0$ for all reverse general constraints. Intuitively, those penalty variables with large objective coefficients and small constraints coefficients should be discarded before others, which is also known as *Column Stuffing* [Gam+15].

Formally, given a partition of variable indices U and V, a set of constraints indices S where $|S| = M - 1$, consider a PBO formula in normalized form:

$$\min . \quad w_U^\sigma x_U^\sigma + w_V^\sigma x_V^\sigma \tag{4.71}$$

$$\sum_{u,\sigma} a_u^\sigma x_u^\sigma + \sum_{v,\sigma} a_v^\sigma x_v^\sigma \geq A_q, \quad q \in M \setminus S \tag{4.72}$$

$$\sum_{v,\sigma} b_v^\sigma x_v^\sigma \geq A_s, \quad s \in S \tag{4.73}$$

where $u \in U, v \in V, \sigma \in \{0, 1\}$ and all x_u^σ are penalty variables. W.l.o.g, we assume $U = [u]$ and

$$\frac{w_1^\sigma}{|a_1^\sigma|} \leq \cdots \leq \frac{w_u^\sigma}{|a_u^\sigma|} \tag{4.74}$$

If there exists an index $k \in U$ such that:

$$\sum_{u \in [k],\sigma} a_u^\sigma + \sum_{v \in V,\sigma} a_v^\sigma \geq A_q \tag{4.75}$$

then we can set all penalty variables $x_u^\sigma, u \in U \setminus [k]$ to 0 since these more "expensive" penalty variables are never needed for feasibility of the constraint with degree A_q.

4.4.2 Parallel columns

Two variables x_i^σ and x_j^σ are called *parallel* [Ach+20] if they appear in the same set of constraints and there exists a non-zero real number λ such that all of their coefficients a_i^σ and a_j^σ in same constraint satisfy:

$$a_i^\sigma = \lambda a_j^\sigma \tag{4.76}$$

Then if $w_i^\sigma = \lambda w_j^\sigma$, we can merge x_i^σ and x_j^σ into a new variable y by:

$$y := x_i^\sigma + \lambda x_j^\sigma \tag{4.77}$$

with bounds:

$$l_y = \begin{cases} 0, & \text{for } \lambda > 0 \\ \lambda, & \text{for } \lambda < 0 \end{cases} \tag{4.78}$$

$$u_y = \begin{cases} 1 + \lambda, & \text{for } \lambda > 0 \\ 1, & \text{for } \lambda < 0 \end{cases} \tag{4.79}$$

If the image of y over x_j and x^k contains no holes:

$$\{x_j + \lambda x_k : x_j \in \{l_j, \cdots, u_j\}, x_k \in \{l_k, \cdots, u_k\}\} = \{l_y, \cdots, u_y\} \tag{4.80}$$

Since all variables should have integral bound $[0, 1]$ in Pseudo-Boolean setting, then every $|\lambda| \neq 0$ will make y non-binary, in this case this technique is not suitable for both PBS and PBO.

4.4.3 Dominated columns

The origin of *dominated columns elimination* dates back to [Gam+15]. Given two variables x_i^σ and x_j^σ with objective coefficients w_i^σ and w_j^σ, assume all constraints are encoded in normalized form, say x_i^σ *dominates* x_j^σ written as $x_i^\sigma \succ x_j^\sigma$, if following criteria hold:

1. $w_j^\sigma \leq w_i^\sigma$;
2. $a_i^\sigma \geq a_j^\sigma, a_i^\sigma \neq 0$ holds for all constraints.

If two variables have a dominance relationship, then we can fix one's value based on their lower and upper bounds, for example:

- If $u_j^\sigma = \infty$ and $x_i^\sigma \succ x_j^\sigma$, then x_j^σ can be set to l_j^σ.

The reason is: assume the optimal solution for x_j^σ is $X_j^\sigma > l_j^\sigma$, we can set $\Delta = X_j^\sigma - l_j^\sigma$, after increasing X_i^σ by Δ and decreasing X_j^σ by Δ, the original constraint will remain feasible:

$$\sum_{k \in N \setminus \{i,j\}} a_k^\sigma X_k^\sigma + a_i^\sigma (X_i^\sigma + \Delta) + a_j^\sigma (X_j^\sigma - \Delta) = \sum_{k \in N} a_k^\sigma X_k^\sigma + (a_i^\sigma - a_j^\sigma)\Delta \geq A \tag{4.81}$$

since $\sum_{k\in N} a_k^\sigma X_k^\sigma \geq A$ already satisfied in an optimal solution and the objective of the formula deduced as well. However, in Pseudo-Boolean setting every variable has bound $[0, 1]$, which leads to that if $X_i^\sigma = X_j^\sigma = 1$ then there is no "increment space" for x_i^σ to compensate the decrements of x_j^σ, therefore this method may not suitable for PBO nor PBS.

4.4.4 Parity fixing

Consider equal constraints:

$$\sum_{i=1}^{k} a_i^\sigma x_i^\sigma = \beta, \quad a_i \in \mathbb{Z}, \beta \in \mathbb{Z}, x \in \{0, 1\} \tag{4.82}$$

we can partition x_i^σ into two sets depends on the parity of their coefficients:

$$\sum_{j=1}^{d} b_j^\sigma y_j^\sigma + \sum_{i=d+1}^{k} a_i^\sigma x_i^\sigma = \beta \tag{4.83}$$

where a_i^σ and b_j^σ are even and odd integers respectively. Now if β is an even number then even number of y_j^σ must be set to true and vice versa, namely:

$$\sum_{j=1}^{d} y_j^\sigma = \beta \qquad \text{mod } 2 \tag{4.84}$$

When $d = 1$ or $d = 2$ above equation degenerated to the analysis mentioned in [BHP08].[5]

An alternative one-hot encoding representation of relation 4.84 is: assume β is an even number and set of indices $S = \{0, 1, \cdots, (d-1)/2\}$, let the binary indicator variable l_i^σ to be:

$$l_i = 1 \Rightarrow \sum_{j=1}^{d} y_j^\sigma = 2i, \quad i \in S \tag{4.85}$$

and adding following two constraints:

$$\sum_{j=1}^{d} y_j^\sigma + M_i l_i^\sigma \leq M_i + 2i, \quad i \in S \tag{4.86}$$

$$\sum_{j=1}^{d} y_j^\sigma + L_i l_i^\sigma \geq L_i + 2i, \quad i \in S \tag{4.87}$$

[5] If exists exactly one i with a_i^σ is odd, then $x_i^\sigma = 0$ if and only if β is even, which implies that we can fix $x_i = \beta$ mod 2; if there exists two odd number $a_i^\sigma, a_j^\sigma, i \neq j$, then $x_i^\sigma = x_j^\sigma$ if and only if β is odd. Hence, one of the variables can be substituted.

where M_i and L_i is the upper and lower bound for $(\sum_{j=1}^{d} y_j^\sigma - 2i), i \in S$ respectively. Finally, we have:

$$\sum_{i \in S} l_i^\sigma \geq 1 \tag{4.88}$$

the analysis when β is an odd number is exactly the same.

However, above parity reasoning seems not hold for inequalities, e.g., consider a general constraint,

$$\sum_{j=1}^{d} b_j^\sigma y_j^\sigma + \sum_{i=d+1}^{k} a_i^\sigma x_i^\sigma \geq \beta \tag{4.89}$$

where b_j^σ are odd integers and a_i^σ are evens. We can set either odd or even number of b_j^σ to true, regardless of the parity of β, only if it could satisfy the constraints.

4.5 Reductions that consider the whole problem

4.5.1 Aggregate pairs of symmetric variables

The symmetry is an important feature that makes the IP problems challenging. In many situations, there exists a set of the indistinguishable object for which individual decision variables must be denied. Given any solution to a model for such a problem, several equivalent "symmetric solutions" can be obtained by simply re-indexing these indistinguishable objects. As a result, the following solving procedure might explore many wasted symmetric equivalent solutions.

The definition of symmetry w.r.t MIP can be found in [Ach+20] and we make suitable adjustment into Pseudo-Boolean form: let $\pi : N \to N$ be a permutation of the index set N. We call π a *symmetry generator* for a Pseudo-Boolean formula (with all constraints in normalized form) if

1. $w_i^\sigma = w_{\pi(i)}^\sigma, \forall i \in N, \sigma \in \{0, 1\}$;
2. there exists a constraints' permutation $\mathcal{P} : M \to M$ such that two constraints C_j and $C_{\mathcal{P}(j)}$ have same degree of falsity, and for all variables $x_i^\sigma \in C_i$ there is a corresponding variable $x_{\pi(i)}^\sigma \in C_{\mathcal{P}(j)}$ with same coefficient $a_i^\sigma = a_{\pi(i)}^\sigma$ and vice versa.

The first condition imposes that the two variables in symmetry are "same", i.e., have the same objective coefficients; the second condition ensures that exchange them will result in the same constraint system. For example, consider a formula:

$$\begin{aligned} \min . \quad & x_1 + x_2 + 3x_3 \\ & x_1 + x_2 + x_3 \geq 1 \end{aligned}$$

there exists a symmetry generator $\pi = (2, 1, 3)$, since $x_1 := 1 \Leftrightarrow x_2 := 1$. Such a generator containing symmetry variables in the same constraint is called an *overlapping system generator*; in contrast, if all pairs of symmetry variables are located in different constraints, then this symmetry generator is called a *non-overlapping system generator*. For example:

$$\begin{aligned} \min . \quad & x_1 + x_2 + 2x_3 + 2x_4 + 3x_5 \\ & x_1 + x_3 + x_5 \geq 1 \\ & x_2 + x_4 + x_5 \geq 1 \end{aligned}$$

One can find there exists a symmetric generator $\pi = (2, 1, 4, 3, 5)$: if we exchange x_1, x_2 and x_3, x_4 we will have same set of constraints. Neither x_1, x_2 nor x_3, x_4 lies on same constraints therefore it is non-overlapping. If replace x_2 by x_1 and x_4 by x_3, we will have:

$$\begin{aligned} \min \quad & 2x_1 + 4x_3 + 3x_5 \\ & x_1 + x_3 + x_5 \leq 1 \\ & x_1 + x_3 + x_5 \leq 1 \end{aligned}$$

Then the parallel constraint detection will remove one of the constraints.

4.5.2 Probing

The idea of *probing* [Sav94; Ach07] is to temporarily set binary variables to true or false and detect whether any variable's bound can be strengthened or whether it is possible to derive stronger inequalities.

Given a variable x_i^σ we can tentatively set it to 0 and 1 and test whether this will propagate some variables x_j^σ to true. For example, consider constraints C in form:

$$C \doteq a_i^\sigma x_i^\sigma + a_j^\sigma x_j^\sigma + \sum_{k,\sigma} a_k^\sigma x_k^\sigma \geq A \tag{4.90}$$

where $i, j, k \in [n], i \neq j \neq k, \sigma \in \{0, 1\}$. The negative slack caused by probing on x_i^σ will incur:

$$slack(C, \rho(x_i^\sigma)) < 0 \Rightarrow x_i^\sigma := 0 \tag{4.91}$$

$$slack(C, \rho(x_i^{1-\sigma})) < 0 \Rightarrow x_i^\sigma := 1 \tag{4.92}$$

if both cases appear then the formula is infeasible. Otherwise, we can propagate x_j^σ to true if its coefficient satisfies either:

$$0 < slack(C, \rho(x_i^\sigma)) < a_j^\sigma \tag{4.93}$$

$$\text{or:} \quad 0 < slack(C, \rho(x_i^{1-\sigma})) < a_j^\sigma \tag{4.94}$$

4.5.3 Biconnected components

Given a variable x_k^σ, two sets of variable indices $U \cup V = N \setminus \{k\}, U \cap V = \emptyset$ and two sets of constraint indices $R \cup S = M, R \cap S = \emptyset, |R| \geq |S|$; consider a Pseudo-Boolean formula F:

$$F \doteq \min . \quad w_k^\sigma x_k^\sigma + w_U^\sigma x_U^\sigma + w_V^\sigma x_V^\sigma \tag{4.95}$$

$$\sum_{u \in U, \sigma} a_u^\sigma x_u^\sigma + a_k^\sigma x_k^\sigma \geq A_R \tag{4.96}$$

$$\sum_{v \in V, \sigma} a_v^\sigma x_v^\sigma + a_k^\sigma x_k^\sigma \geq A_S \tag{4.97}$$

where $\sigma \in \{0, 1\}$. By fixing $x_k^\sigma = X_k^\sigma \in \{0, 1\}$ above formula splits into two sub-formulas which is called *biconnected components* [Ach+20]:

$$\begin{aligned} F(x_U^\sigma, A_R) &\doteq \min \left\{ w_U^\sigma x_U^\sigma + w_k^\sigma X_k^\sigma : \textstyle\sum_{u \in U, \sigma} a_u^\sigma x_u^\sigma \geq A_R - a_k^\sigma X_k^\sigma \right\} \\ F(x_V^\sigma, A_S) &\doteq \min \left\{ w_V^\sigma x_V^\sigma + w_k^\sigma X_k^\sigma : \textstyle\sum_{v \in V, \sigma} a_v^\sigma x_v^\sigma \geq A_S - a_k^\sigma X_k^\sigma \right\} \end{aligned} \tag{4.98}$$

The idea here is to solve the smaller components $F(x_V^\sigma, A_S)$ for each setting of the x_k^σ to obtain the optimal solution of X_V^σ in each setting:

$$X_k^\sigma := 0 \Longrightarrow X_V^{0,\sigma} \tag{4.99}$$

$$X_k^\sigma := 1 \Longrightarrow X_V^{1,\sigma} \tag{4.100}$$

If both sub-formulas in 4.98 have been solved to optimality, we can deduce following implicaton:

1. if $X_k^\sigma := 0$ implies $F(x_V^\sigma, A_S)$ infeasible, then $X_k^\sigma := 1$ and vice versa;
2. if $X_v^{0,\sigma} = X_v^{1,\sigma}, \forall v \in V$ hold, then we can fix $x_v^\sigma := X_v^{0,\sigma}$;
3. if $X_v^{0,\sigma} \neq X_v^{1,\sigma}, \forall v \in V$ hold, then we can substitute $x_v^\sigma := X_v^{0,\sigma} + (X_v^{1,\sigma} - X_v^{0,\sigma}) x_k^\sigma$

Chapter 5

Experimental evaluation

This section aims to deliver the performance evaluation of several preprocessing techniques presented above concerning the state-of-the-art PBS/PBO solver RoundingSat [EN18]. Specifically, we have implemented the *Pure Literal Elimination*, *Hyper Binary Resolution*, and *Subsumption Detection* from SAT community and use the open-sourced presolver PaPILO 2.0 [Bes+21] as an external library for testing MIP presolving techniques.

The first section Implementation detail below introduces the underlying structure of the implementation and the second section Performance evaluation summarizes the experimental statistics.

5.1 Implementation detail

The name of our presolver is simply called Pre[1] by its functionality, which is formed by two different sub-presolvers SAT-Pre and MIP-Pre utilizing the SAT/MIP presolving techniques respectively. The former is the novel presented presolver and the latter provides the port to run PaPILO as an external library and fetch the presolved instances. These two sub-presolvers could run independently and if one wish to perform both SAT and MIP presolving, the instances will be firstly fed to SAT-Pre and then to MIP-Pre. Besides, both two presolvers supports large integer operation, i.e., arithmetic operation on 128-bits signed integers.

The SAT-Pre uses hash set storing constraints and variables to support efficient insert and delete operation within expected constant time. By using external library `boost::hash_combine`[2] algorithm which implements the hash scheme presented in [Aus05], Algorithm 1 hashes every Pseudo-Boolean constraints into an unsigned integer.

Regarding the implementation of SAT presolving techniques: the *Pure Literal Elimination* scheme in Pre is implemented within PaPILO suite and running as part of MIP-Pre. The implementation avoids explicitly transforming all constraints into normalized form but checks the feasibility of value fixing for every variable by ensuring its

[1] https://github.com/RomaLzhih/Pseudo-Boolean-Presolver

[2] https://www.boost.org/doc/libs/1_79_0/libs/container_hash/doc/html/hash.html

Algorithm 1: Hash algorithm for Pseudo-Boolean Constraints

Input: A constraint C contains variables x_U^σ, coefficients c_U and degree d, where $U \subseteq N$.

Output: An unsinged integer hash value h

```
1  begin
2      h = |U|
3      forall x_i^σ ∈ x_N^σ do
4          h ← boost::hash_combine(h, i)
5          h ← boost::hash_combine(h, c_i)
6      h ← boost::hash_combine(h, d)
7      return h
```

coefficients have the same sign, and the rows it presents have the same operator. The parallelism is added by checking every column at the same time. Algorithm 2 shows the detail.

Algorithm 2: Pure Literal Elimination

Input: A variable x_i^σ, its matrix coefficient vector C, objective coefficient vector w and sparse row vector R where it appears, assume all constraints have only operators $\{\geq, \leq, =\}$;

Output: A value v which x_i^σ could be fixed, if it is not possible return -1.

```
1  begin
2      Initialize four Boolean variables op, rop and sign, rsign
3      if the first row r_1^σ ∈ R is an equation then
4          return -1
5      op ← whether r_1^σ has greater-or-equal operator
6      sign ← whether c_1^σ > 0
7      forall rows r_i^σ ∈ R do
8          rop ← whether r_i^σ has same operator as r_1^σ
9          rsign ← whether c_i^σ > 0
10         if r_i^σ is an equation OR rop ⊕ rsign then
11             return -1
12     if op ⊕ sign AND w_i^σ ≥ 0 then
13         return v ← 0
14     else if (NOT op ⊕ sign) AND w_i^σ ≤ 0 then
15         return v ← 1
16     else
17         return -1
```

The *Hyper Binary Resolution* in SAT-Pre implements a relatively weak version, which constructs the implication graph using only $2-ary$ constraints and detects whether the variables in other constraints have a common neighbor in the graph. Algorithm 3 illustrates the detail. Among constructing the implication graph, assume in total N

variables appear in the formula, every variable $x_i^\sigma, i \in N$ will have a corresponding node v_i in the graph and its negation $x_i^{1-\sigma}$ is corresponding to node v_{i+N}. During the edge insertion in line 4, we firstly normalize the constraints and then add the edge, for example, given a constraint $\bar{x}_1 + \bar{x}_2 \geq 1$ and $N = 3$, we will add edges (v_1, v_5) and (v_2, v_4). Also note in line 7, instead of calculating the intersection of the neighbors for every x_i, we pick the x_i with the smallest neighbor cardinality and check whether these neighbors are presented among others as well. This avoids the value copy during the intersection operation and shows good performance in experiments.

Algorithm 3: Hyper Binary Resolution

Input: Constraint pool P in SAT-PRE, a empty graph $G(V, E)$.
Output: Set of new constraints Q.

1 **begin**
2 **forall** $2-ary$ *constraint* $C \ni \{x_1, x_2\}$ *in* P **do**
3 $V = V \cup \{x_1, x_2, \bar{x}_1, \bar{x}_2\}$
4 $E = E \cup (\bar{x}_1, x_2)$ if $x_1 := 0$ implies $x_2 := 1$ and vice versa
5 maps C to the newly added edge.
6 **forall** *constraints* $D \ni x_U^\sigma$ *in* P *where* $|Vars(D)| \geq 3, U \subseteq N$ **do**
7 **if** *all variables* $x_i^\sigma, i \in U \backslash \{j\}, j \in U$ *has a common neighbor* y *in* G **then**
8 perform canceling addition between D and constraints associated to the edges (x_i^σ, y) over variable x_i^σ
9 add resolvent constraint $D' \ni \{x_j^\sigma, y\}$ to Q
10 **return** Q

The *Subsumption Detection* calls the ROUNDINGSAT as an external tool for every pair of constraints to detect whether the formula 3.18 holds or not and delete the subsumed constraints immediately once it is found.

5.2 Performance evaluation

As performed in [Dev+21], we use `PB16`[3] as the test benchmark. Specifically, we test all 1600 PBO instances in `OPT-SMALL-INT` and 2118 PBS instances in `DEC-SMALL-INT`. As for hardware, each instance runs on a single exclusive node with 6 CPU (for utilization of presolving only) and 8192 memory per core running on Linux version 3.10.0. The default time limit for the whole program is set to be 1800s and the time budget allocated for presolving is set to 10% (in total 180s) as used in [MLM09].

We evaluate the impact of 13 different presolving techniques w.r.t to the Pseudo-Boolean solver, namely: the *Coefficient Strengthen*, *Singleton Column*, *Doubleton Equation*, *Dual Fixing*, *Parallel Row*, *Probing*, *Unit Propagation*, *Pure Literal Elimination*, *Simple Probing*, *Simplify Inequalities*, *Sparsify*, *Column Stuffing* and *Hyper Binary Resolution (HBR)* in terms of the number of solved instances, average time and instances reduction size.

[3] http://www.cril.univ-artois.fr/PB16/bench/PB16-used.tar

The PaPILO suite is formed by underlying presolving scheme and provides user switches for certain presolving techniques. We do not modify the underlying scheme since we wish to use PaPILO as an external library. Note that even if all switches has been turned off, the underlying scheme of PaPILO will still simplify some instances. Based on this observation, before we start evaluate MIP techniques which we can explicitly switch on/off, we will run PaPILO using only underlying scheme, and after that, every technique will be turned on with all others off within one round for evaluation.

Given the benchmark we use, we write down the number of solved instances, average solving time for running roundingSat and average presolving time for the technique, the time calculation is based on those instances which do not trigger the time limit. We also save the number of faster and slower instances affected by the application of presolving; among those instances which are solved and trigger the presolving, if they are solved by roundingSat as well, we calculate the performance improvement mean ϵ_u, median δ_u and decrease mean ϵ_d, median δ_d respectively, otherwise this instance would only increase the total number of faster/slower instances but won't contribute to the calculation of ϵ and δ. We also write down the number of exclusively solved/unsolved instances by every technique, i.e., in Table 5.1, 32 instances which are not solved by RoundingSat become solvable after applying the *Pure Literal Elimination*. Besides, for each technique, we count its average calling frequency and the successfully applied ratio [4]. Finally, the average reduction ratio[5] from every technique is appended at the last column of table. Table 5.1 and 5.2 summarizes the difference of using presolving in PBS and PBO respectively.

Table 5.1: Evaluation of presolving techniques with respect to PBS

Name	nSolved(s)	Sol.Time(s)	Pre.Time(s)	nFaster	ext.Solved	ϵ_u(%)	δ_u(%)	nSlower	ext.Unsolved	ϵ_d(%)	δ_d(%)	nCall	appl(%)	red(%)
PureLit	**1725**	52.380	0.319	361	32	42.69	37.38	103	**31**	**140.80**	27.82	**2.37**	24.26	9.47
roundingSat	1724	51.577	NA	NA	NA	NA	**NA**	NA	NA	NA	NA	NA	NA	NA
ParallelRow	1709	55.905	0.322	**514**	32	35.03	28.90	268	47	482.79	50.62	1.00	**43.03**	15.00
SimpleProbing	1705	55.007	0.279	46	**34**	49.99	41.05	66	53	13844.41	372.77	1.03	2.33	2.85
CoeffStrengthen	1703	54.358	0.303	76	29	30.52	29.99	66	50	38.18	**23.06**	1.06	3.77	18.63
DualFix	1702	53.469	0.284	29	29	NA	NA	**51**	51	NA	NA	1.00	0.00	NA
defaultPaPILO	1702	53.636	**0.278**	29	29	NA	NA	**51**	51	NA	NA	NA	NA	NA
HBR	1701	54.826	1.816	238	19	33.69	31.63	252	42	151.34	25.31	1.00	25.57	0.18
Probing	1698	55.433	18.426	327	37	53.53	48.97	168	63	478.20	48.99	2.14	25.31	**29.82**
Propagation	1698	53.450	0.308	89	28	68.17	**79.76**	80	54	1026.53	140.14	1.18	6.06	12.90
ColSingleton	1696	48.973	0.279	41	25	**68.24**	69.79	63	53	6842.71	297.65	1.02	1.86	2.22
Doubletoneq	1693	51.383	0.281	58	29	45.99	30.80	86	60	944.08	110.44	1.07	3.97	0.51
SimplifyIneq	1691	53.491	0.333	79	28	25.28	23.88	64	61	174.37	**151.01**	1.03	3.31	2.88
Stuffing	1688	**48.828**	**0.278**	41	25	68.19	69.91	71	61	6854.40	426.36	1.02	1.87	2.22
Sparsify	1684	53.383	0.282	28	27	30.19	30.19	67	67	NA	NA	1.00	0.06	0.17

It is revealed from the table that applying the presolving, such as *Probing*, *Pure literal* and *Parallel Row*, could accelerate the solving for a large group of instances,

[4]We use the reduction principle in PaPILO here. Define the transaction as the number of reduced/added variable/constraint among the execution of a technique. For example, if a presolving technique deletes a row from an instance, then the transaction is 1. The applied ratio is calculated as the number of instances where a technique produce non-empty transaction divides the number of instances where this technique is called, e.g., a technique produces non-empty transaction for 1 instance among 100 instances will have applied ratio 1%

[5]The reduction ratio is calculated as the number of applied transaction divides instances' size. For example, if a presolving technique reports 4 rows deletion from an instance that has 100 rows, but only 2 row deletion is applied, then the applied transaction is 2 and the reduction ratio is 2%.

Table 5.2: Evaluation of presolving techniques with respect to PBO

Name	nSolved(s)	Sol.Time(s)	Pre.Time	nFaster	ext.Faster	ϵ_u(%)	δ_u(%)	nSlower	ext.Slower	ϵ_d(%)	δ_d(%)	nCall	appl(%)	red(%)
RoundingSat	**1031**	51.801	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
Doubletoneq	1023	51.733	0.269	67	6	40.48	**42.67**	28	14	116.63	80.14	1.08	7.44	2.53
Propagation	1023	55.546	0.322	292	10	**42.16**	40.07	62	18	174.27	69.24	1.65	32.94	**67.56**
SimpleProbing	1022	50.924	0.264	6	6	NA	NA	15	15	NA	NA	1.00	0.00	NA
HBR	1022	53.415	0.749	356	3	33.26	32.73	152	**12**	**59.86**	**12.52**	1.00	48.43	0.25
DualFix	1019	51.829	0.280	117	7	20.65	19.90	24	19	3803.15	66.43	2.34	11.57	0.33
Stuffing	1019	**49.603**	0.271	23	7	34.54	25.80	32	19	474.74	75.12	1.03	2.99	0.64
CoeffStrengthen	1018	50.505	0.288	117	7	28.91	26.36	47	20	985.10	27.82	1.19	13.76	31.16
ColSingleton	1017	49.162	0.268	22	7	36.36	26.80	35	21	439.20	75.49	1.03	2.99	0.64
ParallelRow	1017	57.602	0.316	309	9	30.48	21.88	112	23	257.96	60.02	1.00	39.52	6.39
SimplifyIneq	1017	51.139	0.302	36	6	31.33	28.67	31	20	296.16	91.21	1.05	4.09	8.29
Sparsify	1016	49.496	1.104	51	7	39.47	38.42	40	22	673.77	147.58	1.00	6.29	2.36
defaultPAPILO	1016	50.736	**0.262**	6	6	NA	NA	**21**	21	NA	NA	NA	NA	NA
PureLit	1015	49.618	0.304	193	7	29.54	23.15	42	23	1863.73	55.57	2.59	20.90	8.24
Probing	998	49.900	50.502	**497**	**12**	38.42	35.90	131	45	2118.60	91.83	**2.80**	**59.10**	20.16

while the companied performance degeneration can not be a negligible issue either. Unexpectedly, nearly all techniques lead to fewer instances solved within the given time limit. For most of the techniques, the average negative impact could be much larger than the corresponding improvement. However, the number of solved faster instances is still considerably more than the slower ones, which can also be reflected by the shorter average solving time.

Besides evaluating a single presolving technique, we also perform the test by turning on several presolving techniques. Specifically, for PBS, we pick those techniques that have a smaller δ_d or larger ϵ_u based on Table 5.1, including *Singleton Column*, *Doubleton Equation*, *Parallel Rows*, *Probing* and *Pure Literal Elimination*; as for PBO, we use same criteria to turn on *Hyper Binary Resolution*, *Doubleton Equation*, *Unit Propagation*, *Parallel Row* and *Simplify Inequality* according to Table 5.2. Figure 5.1 and Figure 5.2 demonstrate the cumulative number of solved instances with/without applying the presolving techniques. For some cases, simply running the presolving is enough to solve them, which leads to the isolated points on the left-bottom corner of the figure.

For PBS, in total 1707 instances are solved after using presolving compared with 1724 solved without applying presolving. Among 1094 instances where some presolving techniques are applied, 783 of them obtain an average 47.91% acceleration (median 38.97%) and 307 instances are slowed down by 312.5% (median 52.68%). For PBO, RoundingSat manages to solve 1031 instances without presolving and 1030 after integrating with presolving; among those 840 instances where presolving techniques are applied, 679 of them are solved with an average faster ratio 36.9% (median 33.86%) and 161 instances are solved with an average slower ratio of 181.1% (median 47.12%). Figure 5.3 illustrates the relative solving time for instances with/without presolving.

As shown in Table 5.3, the *Probing* causes most of the problem reduction to those PBS instances which it applies to among all presolving techniques. For PBO instances, the *Unit propagation* performs this role. Besides, nearly all techniques are called more than once and reduce the problem size to some extent.

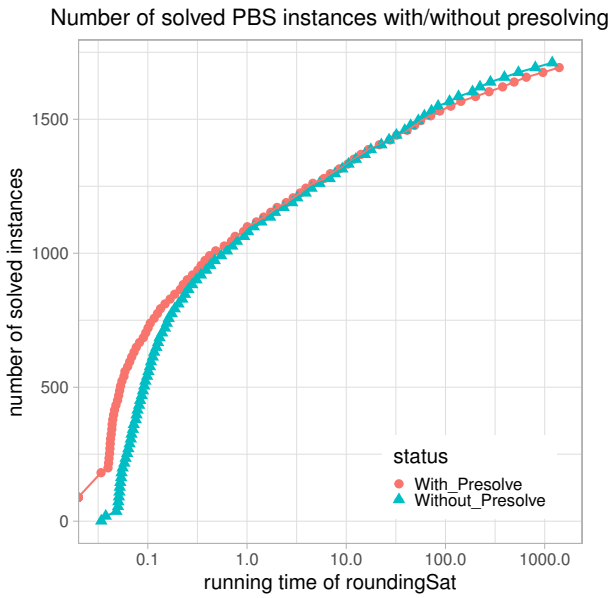


Figure 5.1: Solving time for PBS and PBO instances with both axes draw on log scale.

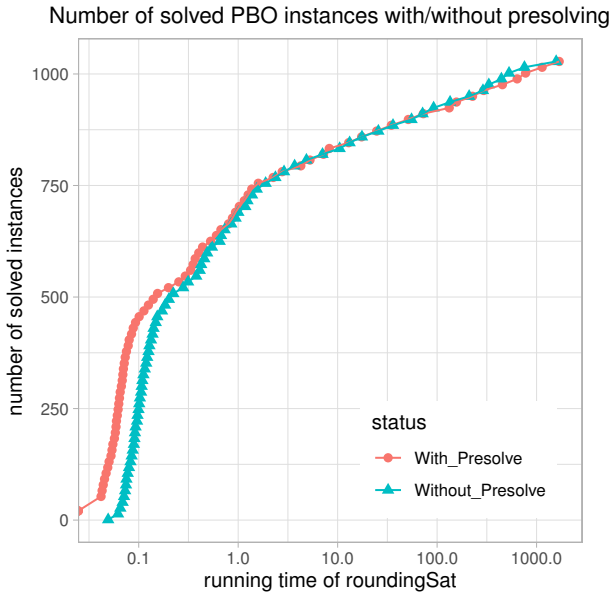


Figure 5.2: Solving time for PBS and PBO instances with both axes draw on log scale.

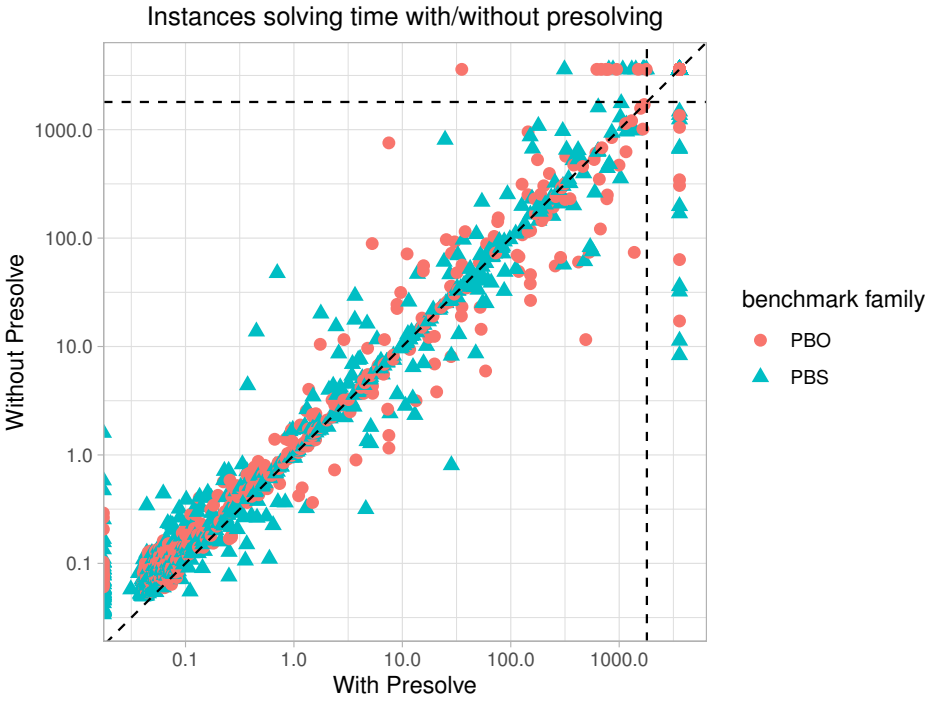


Figure 5.3: Solving time for PBS and PBO instances with both axes draw on log scale.

Table 5.3: Information for applied techniques when solving PBS and PBO instances where dash means this technique are disabled for evaluating the corresponding benchmark family.

Name	PBS				PBO			
	nCalls	nApplied(%)	Pre.Time(s)	red(%)	nCalls	nApplied(%)	Pre.Time(s)	red(%)
ColSingleton	4.57	14.98	0.002	1.33	-	-	-	-
Doubletoneq	2.17	17.83	0.004	3.20	1.08	6.80	0.001	1.81
ParallelRows	1.44	42.80	0.038	15.65	1.01	59.70	0.048	4.32
Probing	1.93	24.11	13.098	16.78	-	-	-	-
PureLit	4.57	27.03	0.020	8.57	-	-	-	-
HBR	-	-	-	-	1.00	48.74	0.809	0.26
Propagation	-	-	-	-	2.35	35.70	0.049	62.57
SimplifyIneq	-	-	-	-	1.24	22.70	0.049	0.79

Chapter 6

Conclusions and future work

We have tried to adapt/lift some preprocessing techniques in SAT and MIP communities into Pseudo-Boolean configurations or figured out some could be trivial or unsuitable for PBS/PBO. Concretely, among the techniques we have discussed, Parallel columns and Dominated columns are not suitable for Pseudo-Boolean configuration: the former tries to merge two parallel columns into a non-binary variable, and the latter requires applied variables to have an infinite bound.

The techniques we run experiments on have shown that presolving is beneficial for the following PB solver's solving phase w.r.t considerably large number of instances, while the negative impact it poses still prevents more instances from being solved within a specific time limit, which could be the target of future research work. The techniques we implement, such as Pure variable and Hyper binary resolution have been shown to be valuable for reducing a problem size efficiently and accelerating the following solving.

From our side, we are preparing to investigate further the following topics:

1. Complete the implementation of HBR, such as adding edges from unit propagation and including the edges implied by the resolvent binary constraints dynamically;

2. Explore more about the heuristics for subsumption, e.g., divide constraints into different blocks and check whether a group of constraints would imply more subsumption;

3. Implement more SAT preprocessing techniques into Pseudo-Boolean configurations, such as General implication graph and Extended hyper resolution, Blocked clause elimination, etc., and evaluate whether these techniques would bring experimental performance improvement for the following solving phase in PB solver;

4. Currently, our presolver PRE is run as a stand-alone library and the I/O consumption between communication and binded PB solver is large. One possible solution is to integrate its presolving functionality into the PB solver's suite directly;

5. From our empirical observation, it seems that applying a presolving method might slow down the solving phase for a problem but not too much, meaning that if we

lose the time limit appropriately, more presolved instances might be solved. For example, if the time limit for solving an instance is set to 500s, we observe roundingSat could solve 1683 PBS instances, but only 1666 PBS instances are solvable after presolving using *Pure Literal Elimination* and 1644 for *Parallel Row*. However, when we set the time limit to 1800s, as reported in Table 5.1, roundingSat solved 1724 instances while the number of solvable instances after applying *Pure Literal Elimination* and *Parallel Row* became 1725 and 1709 respectively. In this case, one possible experimental topic is to increase the time limit and see what will happen.

Chapter 7

Acknowledgement

The author would like to express his sincere appreciation to his supervisors, Jakob Nordström and Stephan Gocht for their invaluable advice, continuous support, and patience during his thesis study. The author would also like to thank his parents and sister, whom without this would have not been possible. The author wishes to thank his friends Hu and Binbin for the happy dinner every Saturday, which could be the author's most enjoyable time within a week. The author would also like to show his thankfulness to Jinke and Yichen, football field and fitness room are some rare places where the author could find someone to chat with in his everyday life.

Life can always be hard, especially when one is suffering from intermittent Covid sequelae, inner foot ankle tearing and eczema successively within four months. There was one day when his head was groggy, skin was itchy and the severe ankle hurt from every step he took, but he had to continue work since there was not much time left for this thesis. He felt sad and hopeless, but just to feel and then, nothing. The author doesn't know what supports him to finish the long time running without relaxing, maybe the intriguing words from books, the beautiful syllables from music, the sound of wind and water in the lovely nature, or everyone whom once has passed his life.

Or maybe just a dream from childhood.

Bibliography

[Bla37] Archie Blake. "Canonical expressions in Boolean algebra". PhD thesis. The University of Chicago, 1937.

[DP60] Martin Davis and Hilary Putnam. "A computing procedure for quantification theory". In: *Journal of the ACM (JACM)* 7.3 (1960), pp. 201–215.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. "A machine program for theorem-proving". In: *Communications of the ACM* 5.7 (1962), pp. 394–397.

[Gom63] Ralph E Gomory. "An algorithm for integer solutions to linear programs". In: *Recent advances in mathematical programming* 64.260-302 (1963), p. 14.

[Rob65] John Alan Robinson. "A machine-oriented logic based on the resolution principle". In: *Journal of the ACM (JACM)* 12.1 (1965), pp. 23–41.

[Dun+67] B Dunham et al. "A non-heuristic program for proving elementary logical theorems". In: *Journal of Symbolic Logic* 32.2 (1967).

[HR69] Peter L Hammer and Sergiu Rudeanu. "Pseudo-boolean programming". In: *Operations Research* 17.2 (1969), pp. 233–261.

[Coo71] Stephen A Cook. "The complexity of theorem-proving procedures". In: *Proceedings of the third annual ACM symposium on Theory of computing*. 1971, pp. 151–158.

[Dan72] George B Dantzig. *Fourier-Motzkin elimination and its dual*. Tech. rep. STANFORD UNIV CA DEPT OF OPERATIONS RESEARCH, 1972.

[Chv73a] Vasek Chvátal. "Edmonds polytopes and a hierarchy of combinatorial problems". In: *Discrete mathematics* 4.4 (1973), pp. 305–337.

[Chv73b] Vasek Chvátal. "Edmonds polytopes and a hierarchy of combinatorial problems". In: *Discrete mathematics* 4.4 (1973), pp. 305–337.

[Lev73] Leonid Anatolevich Levin. "Universal sequential search problems". In: *Problemy peredachi informatsii* 9.3 (1973), pp. 115–116.

[BMW75] AL Brearley, Gautam Mitra, and H Paul Williams. "Analysis of mathematical programming problems prior to applying the simplex algorithm". In: *Mathematical programming* 8.1 (1975), pp. 54–83.

[APT79] Bengt Aspvall, Michael F Plass, and Robert Endre Tarjan. "A linear-time algorithm for testing the truth of certain quantified boolean formulas". In: *Information processing letters* 8.3 (1979), pp. 121–123.

[JS80] Ellis L Johnson and Uwe H Suhl. "Experiments in integer programming". In: *Discrete Applied Mathematics* 2.1 (1980), pp. 39–55.

[GS81] Monique Guignard and Kurt Spielberg. "Logical reduction methods in zero-one programming—minimal preferred variables". In: *Operations Research* 29.1 (1981), pp. 49–74.

[JP82] Ellis L Johnson and Manfred W Padberg. "Degree-two inequalities, clique facets, and biperfect graphs". In: *North-Holland Mathematics Studies*. Vol. 66. Elsevier, 1982, pp. 169–187.

[BBG83] Gordon H Bradley, Gerald G Brown, and Glenn W Graves. "Structural redundancy in large-scale optimization models". In: *Redundancy in Mathematical Programming*. Springer, 1983, pp. 145–169.

[CJP83] Harlan Crowder, Ellis L Johnson, and Manfred Padberg. "Solving large-scale zero-one linear programming problems". In: *Operations Research* 31.5 (1983), pp. 803–834.

[CCT87] William Cook, Collette R Coullard, and Gy Turán. "On the complexity of cutting-plane proofs". In: *Discrete Applied Mathematics* 18.1 (1987), pp. 25–38.

[Man87] CPLEX User's Manual. "Ibm ilog cplex optimization studio". In: *Version* 12 (1987), pp. 1987–2018.

[ZM88] Ramin Zabih and David A McAllester. "A Rearrangement Search Strategy for Determining Propositional Satisfiability." In: *AAAI*. Vol. 88. 1988, pp. 155–160.

[HP91] Karla L Hoffman and Manfred Padberg. "Improving LP-representations of zero-one linear programs for branch-and-cut". In: *ORSA Journal on Computing* 3.2 (1991), pp. 121–134.

[CM93] S Frank Chang and S Thomas McCormick. "Implementation and computational results for the hierarchical algorithm for making sparse matrices sparser". In: *ACM Transactions on Mathematical Software (TOMS)* 19.3 (1993), pp. 419–441.

[Sav94] Martin WP Savelsbergh. "Preprocessing and probing techniques for mixed integer programming problems". In: *ORSA Journal on Computing* 6.4 (1994), pp. 445–454.

[AA95] Erling D Andersen and Knud D Andersen. "Presolving in linear programming". In: *Mathematical Programming* 71.2 (1995), pp. 221–245.

[VT95] Allen Van Gelder and Yumi Tsuji. *Satisfiability testing with more reasoning and less guessing*. Citeseer, 1995.

[BS97] Roberto J Bayardo Jr and Robert Schrag. "Using CSP look-back techniques to solve real-world SAT instances". In: *Aaai/iaai*. Providence, RI. 1997, pp. 203–208.

[ABS99] Gilles Audemard, Belaid Benhamou, and Pierre Siegel. "La méthode d'avalanche AVAL: une méthode énumérative pour SAT". In: *5èmes Journées Nationales sur la Résolution Pratique des Problèmes NP-Complets (JNPC'99)*. 1999, pp. 17–25.

[GRA99] Marques-Silva JP GRASP. "A search algorithm for propositional satisfiability/JP Marques-Silva, KA Sakallah". In: *IEEE Transactions on Computers* 48.5 (1999), pp. 506–521.

[Kul99] O. Kullmann. "On a generalization of extended resolution". In: *Discrete Applied Mathematics* 96-97 (1999), pp. 149–176. ISSN: 0166-218X. DOI: https://doi.org/10.1016/S0166-218X(99)00037-2. URL: https://www.sciencedirect.com/science/article/pii/S0166218X99000372.

[MS99] Joao P Marques-Silva and Karem A Sakallah. "GRASP: A search algorithm for propositional satisfiability". In: *IEEE Transactions on Computers* 48.5 (1999), pp. 506–521.

[Mar99] Alexander Martin. "Integer programs with block structure". PhD thesis. 1999.

[ANS00] Alper Atamtürk, George L Nemhauser, and Martin WP Savelsbergh. "Conflict graphs in solving integer programming problems". In: *European Journal of Operational Research* 121.1 (2000), pp. 40–55.

[Li00] Chu Min Li. "Integrating equivalency reasoning into Davis-Putnam procedure". In: *AAAI/IAAI* 2000 (2000), pp. 291–296.

[DR01] Stefan Dantchev and Søren Riis. "" Planar" tautologies hard for resolution". In: *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE. 2001, pp. 220–229.

[IP01] Russell Impagliazzo and Ramamohan Paturi. "On the complexity of k-SAT". In: *Journal of Computer and System Sciences* 62.2 (2001), pp. 367–375.

[Le 01] Daniel Le Berre. "Exploiting the real power of unit propagation lookahead". In: *Electronic Notes in Discrete Mathematics* 9 (2001), pp. 59–80.

[Mos+01a] Matthew W Moskewicz et al. "Chaff: Engineering an efficient SAT solver". In: *Proceedings of the 38th annual Design Automation Conference*. 2001, pp. 530–535.

[Mos+01b] Matthew W Moskewicz et al. "Chaff: Engineering an efficient SAT solver". In: *Proceedings of the 38th annual Design Automation Conference*. 2001, pp. 530–535.

[Mos+01c] Matthew W Moskewicz et al. “Chaff: Engineering an efficient SAT solver”. In: *Proceedings of the 38th annual Design Automation Conference*. 2001, pp. 530–535.

[Bac02] Fahiem Bacchus. “Enhancing Davis Putnam with extended binary clause reasoning”. In: *AAAI/IAAI* 2002 (2002), pp. 613–619.

[BH02] Endre Boros and Peter L Hammer. “Pseudo-boolean optimization”. In: *Discrete applied mathematics* 123.1-3 (2002), pp. 155–225.

[DG02] Heidi E Dixon and Matthew L Ginsberg. “Inference methods for a pseudo-boolean satisfiability solver”. In: *AAAI/IAAI*. 2002, pp. 635–640.

[SNS02] Patrik Simons, Ilkka Niemelä, and Timo Soininen. “Extending and implementing the stable model semantics”. In: *Artificial Intelligence* 138.1-2 (2002), pp. 181–234.

[Bie04a] Armin Biere. “Resolve and expand”. In: *International conference on theory and applications of satisfiability testing*. Springer. 2004, pp. 59–70.

[Bie04b] Armin Biere. “The evolution from Limmat to Nanosat”. In: *Technical reports* 444 (2004).

[SP04a] Sathiamoorthy Subbarayan and Dhiraj K Pradhan. “NiVER: Non-increasing variable elimination resolution for preprocessing SAT instances”. In: *International conference on theory and applications of satisfiability testing*. Springer. 2004, pp. 276–291.

[SP04b] Sathiamoorthy Subbarayan and Dhiraj K Pradhan. “NiVER: Non-increasing variable elimination resolution for preprocessing SAT instances”. In: *International conference on theory and applications of satisfiability testing*. Springer. 2004, pp. 276–291.

[Aus05] Matt Austern. “Draft technical report on c++ library extensions”. In: *ISO/IEC DTR* 19768 (2005).

[CK05] Donald Chai and Andreas Kuehlmann. “A fast pseudo-boolean constraint solver”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24.3 (2005), pp. 305–317.

[FM05a] Armin Fügenschuh and Alexander Martin. “Computational integer programming and cutting planes”. In: *Handbooks in Operations Research and Management Science* 12 (2005), pp. 69–121.

[FM05b] Armin Fügenschuh and Alexander Martin. “Computational integer programming and cutting planes”. In: *Handbooks in Operations Research and Management Science* 12 (2005), pp. 69–121.

[SE05] Niklas Sorensson and Niklas Een. “Minisat v1. 13-a sat solver with conflict-clause minimization”. In: *SAT* 2005.53 (2005), pp. 1–2.

[BS06] Armin Biere and Carsten Sinz. “Decomposing SAT problems into connected components”. In: *Journal on Satisfiability, Boolean Modeling and Computation* 2.1-4 (2006), pp. 201–208.

[MM06] Vasco M Manquinho and Joao Marques-Silva. "On using cutting planes in pseudo-boolean optimization". In: *Journal on Satisfiability, Boolean Modeling and Computation* 2.1-4 (2006), pp. 209–219.

[SS06] Hossein M Sheini and Karem A Sakallah. "Pueblo: A hybrid pseudo-boolean SAT solver". In: *Journal on Satisfiability, Boolean Modeling and Computation* 2.1-4 (2006), pp. 165–189.

[Ach07] Tobias Achterberg. "Constraint integer programming". In: (2007).

[Bix07] Bob Bixby. "The gurobi optimizer". In: *Transp. Re-search Part B* 41.2 (2007), pp. 159–178.

[AJS08] Gilles Audemard, Said Jabbour, and Lakhdar Sais. "SAT graph-based representation: A new perspective". In: *Journal of Algorithms* 63.1-3 (2008), pp. 17–33.

[BHP08] Timo Berthold, Stefan Heinz, and Marc E Pfetsch. "Solving pseudo-Boolean problems with SCIP". In: (2008).

[Ach09] Tobias Achterberg. "SCIP: solving constraint integer programs". In: *Mathematical Programming Computation* 1.1 (2009), pp. 1–41.

[CIP09] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. "The complexity of satisfiability of small depth circuits". In: *International Workshop on Parameterized and Exact Computation*. Springer. 2009, pp. 75–85.

[MLM09] Ruben Martins, Inês Lynce, and Vasco Manquinho. "Preprocessing in pseudo-boolean optimization: An experimental evaluation". In: *Eighth International Workshop on Constraint Modelling and Reformulation*. Citeseer. 2009.

[HJB10] Marijn Heule, Matti Järvisalo, and Armin Biere. "Clause elimination procedures for CNF formulas". In: *International Conference on Logic for Programming Artificial Intelligence and Reasoning*. Springer. 2010, pp. 357–371.

[JBH10a] Matti Järvisalo, Armin Biere, and Marijn Heule. "Blocked clause elimination". In: *International conference on tools and algorithms for the construction and analysis of systems*. Springer. 2010, pp. 129–144.

[JBH10b] Matti Järvisalo, Armin Biere, and Marijn Heule. "Blocked clause elimination". In: *International conference on tools and algorithms for the construction and analysis of systems*. Springer. 2010, pp. 129–144.

[LP10] Daniel Le Berre and Anne Parrain. "The Sat4j library, release 2.2". In: *Journal on Satisfiability, Boolean Modeling and Computation* 7.2-3 (2010), pp. 59–64.

[MML10] Vasco Manquinho, Ruben Martins, and Inês Lynce. "Improving unsatisfiability-based algorithms for boolean optimization". In: *International conference on theory and applications of satisfiability testing*. Springer. 2010, pp. 181–193.

[AW13] Tobias Achterberg and Roland Wunderling. "Mixed integer programming: Analyzing 12 years of progress". In: *Facets of combinatorial optimization*. Springer, 2013, pp. 449–481.

[HJB13] Marijn JH Heule, Matti Järvisalo, and Armin Biere. "Revisiting hyper binary resolution". In: *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer. 2013, pp. 77–93.

[BBL14] Christian Bliek1ú, Pierre Bonami, and Andrea Lodi. "Solving mixed-integer quadratic programming problems with IBM-CPLEX: a progress report". In: *Proceedings of the twenty-sixth RAMP symposium*. 2014, pp. 16–17.

[Gam+15] Gerald Gamrath et al. "Progress in presolving for mixed integer programming". In: *Mathematical Programming Computation* 7.4 (2015), pp. 367–398.

[Bal+16] Tomáš Balyo et al. "SAT race 2015". In: *Artificial Intelligence* 241 (2016), pp. 45–65.

[HKB17] Marijn JH Heule, Benjamin Kiesl, and Armin Biere. "Short proofs without new variables". In: *International Conference on Automated Deduction*. Springer. 2017, pp. 130–147.

[Kor+17] Tuukka Korhonen et al. "MaxPre: An Extended MaxSAT Preprocessor". In: *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT 2017)*. Ed. by Serge Gaspers and Toby Walsh. Vol. 10491. Lecture Notes in Computer Science. Springer, 2017, pp. 449–456.

[EN18] Jan Elffers and Jakob Nordström. "Divide and Conquer: Towards Faster Pseudo-Boolean Solving." In: *IJCAI*. Vol. 18. 2018, pp. 1291–1299.

[BT19] Sam Buss and Neil Thapen. "DRAT proofs, propagation redundancy, and extended resolution". In: *International Conference on Theory and Applications of Satisfiability Testing*. Springer. 2019, pp. 71–89.

[Ach+20] Tobias Achterberg et al. "Presolve reductions in mixed integer programming". In: *INFORMS Journal on Computing* 32.2 (2020), pp. 473–506.

[Pio20] Marek Piotrów. "Uwrmaxsat: Efficient solver for maxsat and pseudo-boolean problems". In: *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE. 2020, pp. 132–136.

[Bes+21] Ksenia Bestuzheva et al. "The SCIP Optimization Suite 8.0". In: *arXiv preprint arXiv:2112.08872* (2021).

[BJK21] Armin Biere, Matti Järvisalo, and Benjamin Kiesl. "Preprocessing in SAT solving". In: *Handbook of Satisfiability* (2021), pp. 391–435.

[BN21] Samuel R Buss and Jakob Nordström. "Proof complexity and SAT solving". In: *Handbook of Satisfiability* 336 (2021), pp. 233–350.

[Dev+21] Jo Devriendt et al. "Cutting to the core of pseudo-Boolean optimization: combining core-guided search with cutting planes reasoning". In: *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event*. 2021, pp. 3750–3758.

[GN21] Stephan Gocht and Jakob Nordström. "Certifying parity reasoning efficiently using pseudo-Boolean proofs". In: *35th AAAI Conference on Artificial Intelligence. AAAI Press*. 2021.