Certified Symmetry and Dominance Breaking for Combinatorial Optimisation

Bart Bogaerts,[1] **Stephan Gocht,**[2,3] **Ciaran McCreesh,**[4] **Jakob Nordström**[3,2]

[1] Vrije Universiteit Brussel
[2] Lund University, Lund, Sweden
[3] University of Copenhagen, Copenhagen, Denmark
[4] University of Glasgow, Glasgow, UK
bart.bogaerts@vub.be, stephan.gocht@cs.lth.se, Ciaran.McCreesh@glasgow.ac.uk, jn@di.ku.dk

Abstract

Symmetry and dominance breaking can be crucial for solving hard combinatorial search and optimisation problems, but the correctness of these techniques sometimes relies on subtle arguments. For this reason, it is desirable to produce efficient, machine-verifiable certificates that solutions have been computed correctly. Building on the cutting planes proof system, we develop a certification method for optimisation problems in which symmetry and dominance breaking are easily expressible. Our experimental evaluation demonstrates that we can efficiently verify fully general symmetry breaking in Boolean satisfiability (SAT) solving, thus providing, for the first time, a unified method to certify a range of advanced SAT techniques that also includes XOR and cardinality reasoning. In addition, we apply our method to maximum clique solving and constraint programming as a proof of concept that the approach applies to a wider range of combinatorial problems.

1 Introduction

Symmetries pose a challenge when solving hard combinatorial problems. For example, consider the Crystal Maze puzzle[1] shown in Figure 1, which is often used in introductory constraint modelling courses. A human modeller might notice that the puzzle is the same under a vertical mirror symmetry, and could introduce the constraint $A < G$ to eliminate this. Or, they may notice a horizontal mirror symmetry, which could be broken with $A < B$. Alternatively, they might spot that the *values* are symmetrical, and that we can interchange 1 and 8, 2 and 7, and so on; this can be eliminated by saying that $A \leq 4$. In each case a constraint is being added that preserves satisfiability overall, but that restricts a solver to finding (ideally) just one witness from each equivalence class of solutions—the hope is that this will improve solver performance. However, although we may be reasonably sure that any of these three constraints is correct individually, are combinations of these constraints valid simultaneously? What if we had said $F < C$ instead of $A < B$? And what if we could use numbers more than once? Getting symmetry elimination constraints right can be error-prone even for experienced modellers, and when dealing with larger problems with many constraints and interacting symmetries it can be hard to tell whether an instance is genuinely unsatisfiable, or was made so by an incorrect symmetry constraint.

Despite these difficulties, symmetry elimination using both manual and automatic techniques has been key to many successes across modern combinatorial optimisation paradigms such as constraint programming (CP) (Garcia de la Banda et al. 2014), Boolean satisfiability (SAT) (Biere et al. 2021), and mixed-integer programming (MIP) (Achterberg and Wunderling 2013). As these optimisation technologies are increasingly being used for high-value and life-affecting decision-making processes, it becomes vital that we can trust their outputs—and unfortunately, current solvers do not always produce the correct answer (Brummayer, Lonsing, and Biere 2010; Cook et al. 2013; Akgün et al. 2018; Gillard, Schaus, and Deville 2019). The most promising way to address this problem appears to be to use *certification*, or *proof logging*, where a solver must produce an efficiently machine-verifiable certificate that the solution given is correct (Alkassar et al. 2011; McConnell et al. 2011). This approach has been successfully used in the SAT community, with numerous proof logging formats such as *RUP* (Goldberg and Novikov 2003), *TraceCheck* (Biere 2006), *DRAT* (Heule, Hunt Jr., and Wetzler 2013a,b; Wetzler, Heule, and Hunt Jr. 2014), *GRIT* (Cruz-Filipe, Marques-Silva, and Schneider-Kamp 2017), and *LRAT* (Cruz-Filipe et al. 2017). However, currently used methods work only for decision problems, and do not support the full range of SAT solving techniques, let alone CP and MIP solving. As a case in point, there is no efficient proof logging for symmetry breaking, except

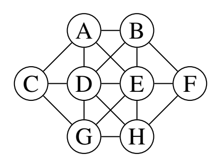

Figure 1: The Crystal Maze puzzle. Place numbers 1 to 8 in the circles, with every circle getting a different number, so that adjacent circles do not have consecutive numbers.

[1]https://theconversation.com/what-problems-will-ai-solve-in-future-an-old-british-gameshow-can-help-explain-49080

for limited cases with small symmetries which can interact only in simple ways (Heule, Hunt Jr., and Wetzler 2015). Tchinda and Djamégni (2020) recently proposed a proof logging method *DSRUP* for symmetric learning of variants of derived clauses, but this format does not support symmetry breaking (in the sense just discussed) and is also inherently unable to support pre- and inprocessing techniques, which are crucial in state-of-the-art SAT solvers.

In this work, we develop a proof logging method for *optimisation* problems, where we are given a formula F and an objective function f, that can deal with *dominance*, a generalization of symmetry. Dominance breaking starts from the observation that we can strengthen F by imposing a constraint C if every solution of F that does not satisfy C is dominated by another solution of F. This technique is used in many fields of combinatorial optimisation (Walsh 2006, 2012; Gent, Petrie, and Puget 2006; McCreesh and Prosser 2016; Jouglet and Carlier 2011; Gebser, Kaminski, and Schaub 2011; Bulhões, Sadykov, and Uchoa 2018; Hoogeboom et al. 2020; Baptiste and Pape 1997; Demeulemeester and Herroelen 2002). The core idea of our method is to present an *explicit construction* of the dominating solution, so that a verifier can check that this construction strictly improves the objective value and preserves satisfaction of F. This constructed solution might itself be dominated, and hence not satisfy C, but since the objective value decreases with every application, the process must eventually terminate. Importantly, verification does not require construction of an assignment satisfying C, and can be performed efficiently even when multiple constraints are to be added; this resolves a practical issue with earlier approaches like (Heule, Hunt Jr., and Wetzler 2015), which struggle with large or overlapping symmetries. Following preliminaries in Section 2, we describe this method in full detail in Section 3.

We have developed a proof format and verifier on top of *VeriPB* (Elffers et al. 2020; Gocht and Nordström 2021; Gocht, McCreesh, and Nordström 2020; Gocht et al. 2020). The pseudo-Boolean constraints and cutting planes proof system (Cook, Coullard, and Turán 1987) used by *VeriPB* are convenient to express and reason with dominance inequalities, and moreover also make it possible to certify XOR and cardinality reasoning (Gocht and Nordström 2021), two other advanced techniques which previous SAT proof logging methods have not been able to support efficiently. In Section 4, we demonstrate that our new verifier can efficiently check *automated static symmetry breaking* in SAT, *manual static symmetry breaking* in CP, and *automated dynamic dominance handling* in maximum clique solving. While the latter two applications are proofs of concept, for static symmetry breaking we show in full generality, and for the first time, that proof logging is practical by running experiments on SAT competition benchmarks. We conclude the paper with some brief remarks in Section 5.

2 Preliminaries

Let us briefly review some standard material, referring the reader to, e.g., Buss and Nordström (2021) for more details. A *literal* ℓ over a Boolean variable x is x itself or its negation $\overline{x} = 1 - x$, where variables take values 0 (false) or 1 (true). A *pseudo-Boolean (PB) constraint* is a 0–1 linear inequality

$$C \doteq \textstyle\sum_i a_i \ell_i \geq A \ , \tag{1}$$

where a_i and A are integers (and $\doteq$ denotes syntactic equality). We can assume without loss of generality that PB constraints are *normalized*; i.e., that all literals ℓ_i are over distinct variables and that the *coefficients* a_i and the *degree (of falsity)* A are non-negative, but most of the time we will not need this. Instead, we will write PB constraints in more relaxed form as $\sum_i a_i \ell_i \geq A + \sum_j b_j \ell_j$ or $\sum_i a_i \ell_i \leq A + \sum_j b_j \ell_j$ when convenient, or even use equality $\sum_i a_i \ell_i = A$ as syntactic sugar for the pair of inequalities $\sum_i a_i \ell_i \geq A$ and $\sum_i -a_i \ell_i \geq -A$, assuming that all constraints are implicitly normalized if needed. The *negation* $\neg C$ of the constraint C in (1) is

$$\neg C \doteq \textstyle\sum_i - a_i \ell_i \geq -A + 1 \ . \tag{2}$$

A *pseudo-Boolean formula* is a conjunction $F \doteq \bigwedge_j C_j$ of PB constraints, which we can also think of as the set $\bigcup_j \{C_j\}$ of constraints in the formula, choosing whichever viewpoint seems most convenient. Note that a *(disjunctive) clause* $\ell_1 \vee \cdots \vee \ell_k$ is equivalent to the PB constraint $\ell_1 + \cdots + \ell_k \geq 1$, so formulas in *conjunctive normal form (CNF)* are special cases of PB formulas.

A *(partial) assignment* is a (partial) function from variables to $\{0, 1\}$; a *substitution* can also map variables to literals. We extend an assignment or substitution ρ from variables to literals in the natural way by respecting the meaning of negation, and for literals ℓ over variables x not in the domain of ρ, denoted $x \notin \mathit{dom}(\rho)$, we use the convention $\rho(\ell) = \ell$. (That is, we can consider all assignments and substitution to be total, but to be the identity outside of their specified domains. Strictly speaking, we also require that all substitutions be defined on the truth constants $\{0, 1\}$ and be the identity on these constants.) We sometimes write $x \mapsto b$ when $\rho(x) = b$, for b a literal or truth value.

We write $\rho \circ \omega$ to denote the composed substitution resulting from applying first ω and then ρ, i.e., $\rho \circ \omega(x) = \rho(\omega(x))$. As an example, for $\omega = \{x_1 \mapsto 0, x_3 \mapsto \overline{x}_4, x_4 \mapsto x_3\}$ and $\rho = \{x_1 \mapsto 1, x_2 \mapsto 1, x_3 \mapsto 0, x_4 \mapsto 0\}$ we have $\rho \circ \omega = \{x_1 \mapsto 0, x_2 \mapsto 1, x_3 \mapsto 1, x_4 \mapsto 0\}$. Applying ω to a constraint C as in (1) yields

$$C{\restriction_\omega} \doteq \textstyle\sum_i a_i \omega(\ell_i) \geq A \ , \tag{3}$$

substituting literals or values as specified by ω. For a formula F we define $F{\restriction_\omega} \doteq \bigwedge_j C_j{\restriction_\omega}$.

Since we will sometimes have to make fairly elaborate use of substitutions, let us discuss some further notational conventions. If F is a formula over variables $\vec{x} = \{x_1, \ldots, x_m\}$, we can write $F(\vec{x})$ when we want to stress the set of variables over which F is defined. For a substitution ω with domain (contained in) $\vec{x}$, the notation $F\big(\vec{x}{\restriction_\omega}\big)$ is understood to be a synonym of $F{\restriction_\omega}$. For the same formula F and $\vec{y} = \{y_1, \ldots, y_m\}$, the notation $F(\vec{y})$ is syntactic sugar for $F{\restriction_\omega}$ with ω denoting the substitution (implicitly) defined by $\omega(x_i) = y_i$ for $i = 1, \ldots, n$. Finally, for a formula $G = G(\vec{x}, \vec{y})$ over $\vec{x} \cup \vec{y}$ and substitutions α and β defined on $\vec{z} = \{z_1, \ldots, z_n\}$ (either of which could be the identity), the notation $G(\vec{z}{\restriction}_\alpha, \vec{z}{\restriction}_\beta)$ should be understood as $G{\restriction}_\omega$ for ω defined by $\omega(x_i) = \alpha(z_i)$ and $\omega(y_i) = \beta(z_i)$ for $i = 1, \ldots, n$.

The (normalized) constraint C in (1) is *satisfied* by ρ if $\sum_{\rho(\ell_i)=1} a_i \geq A$. A PB formula F is satisfied by ρ if all constraints in it are, in which case it is *satisfiable*. If there is no satisfying assignment, F is *unsatisfiable*. Two formulas are *equisatisfiable* if they are both satisfiable or both unsatisfiable. We also consider optimisation problems, where in addition to F we are given an integer linear objective function $f \doteq \sum_i w_i \ell_i$ and the task is to find an assignment that satisfies F and minimizes f. (To deal with maximization problems we can just negate the objective function.)

Cutting planes (Cook, Coullard, and Turán 1987) is a method for iteratively deriving constraints C from a pseudo-Boolean formula F. We write $F \vdash C$ for any constraint C derivable as follows. Any *axiom constraint* $C \in F$ is trivially derivable, as is any *literal axiom* $\ell \geq 0$. If $F \vdash C$ and $F \vdash D$, then any positive integer *linear combination* of these constraints is derivable. Finally, from a constraint in normalized form $\sum_i a_i \ell_i \geq A$ we can use *division* by a positive integer d to derive $\sum_i \lceil a_i/d \rceil \ell_i \geq \lceil A/d \rceil$, dividing and rounding up the degree and coefficients. For a set of PB constraints F' we write $F \vdash F'$ if $F \vdash C$ for all $C \in F'$.

For PB formulas F, F' and constraints C, C', we say that F *implies* or *models* C, denoted $F \models C$, if any assignment satisfying F also satisfies C, and write $F \models F'$ if $F \models C'$ for all $C' \in F'$. It is easy to see that if $F \vdash F'$ then $F \models F'$, and so F and $F \wedge F'$ are equisatisfiable. A constraint C is said to *literal-axiom-imply* another constraint C' if C' can be derived from C by addition of literal axioms $\ell \geq 0$.

A constraint C *unit propagates* the literal ℓ under ρ if $C{\restriction}_\rho$ cannot be satisfied unless $\ell \mapsto 1$. During *unit propagation* on F under ρ, ρ is extended iteratively by any propagated literals until an assignment ρ' is reached under which no constraint $C \in F$ is propagating, or under which some constraint C would propagate a literal had it not already been assigned to the opposite value. The latter scenario is referred to as a *conflict*, since ρ' *violates* the constraint C in this case, and ρ' is called a *conflicting* assignment. Using the generalization of (Goldberg and Novikov 2003) in (Elffers et al. 2020), we say that F implies C by *reverse unit propagation (RUP)*, and that C is a *RUP constraint* with respect to F, if $F \wedge \neg C$ unit propagates to conflict under the empty assignment. If C is a RUP constraint with respect to F, then it can be proven that there is also a derivation $F \vdash C$. More generally, it can be shown that $F \vdash C$ if and only if $F \wedge \neg C \vdash \bot$, where $\bot$ is a shorthand for the *trivially false constraint* $0 \geq 1$. Therefore, we will extend the notation and write $F \vdash C$ also when C is derivable from F by RUP or by contradiction. It is worth noting here again that, as shown in (2), the negation of any PB constraint can also be expressed syntactically as a PB constraint—this fact will be convenient in what follows.

3 A Proof System for Dominance Breaking

We proceed to develop our formal proof system for verifying dominance breaking, which we have implemented on top of the version of *VeriPB* in (Gocht and Nordström 2021). We remark that for applications it is absolutely crucial not only that the proof system be sound, but that all proofs be efficiently machine-verifiable. There are significant challenges involved in making proof logging and verification efficient, but in this section we mostly ignore these aspects of our work and focus on the theoretical underpinnings.

Our foundation is the cutting planes proof system described in Section 2. However, in a proof in our system for (F, f), where f is a linear objective function to be minimized under the pseudo-Boolean formula F (or where $f \doteq 0$ for decision problems), we also allow strengthening F by adding constraints C that are not implied by the formula. Pragmatically, adding C should be in order as long as we keep some optimal solution, i.e., a satisfying assignment to F that minimizes f, which we will refer to as an *f-minimal solution* of F. We will formalize this idea by allowing the use of an additional pseudo-Boolean formula $\mathcal{O}_\preceq(\vec{u}, \vec{v})$ that, together with a sequence of variables $\vec{z}$, defines a relation $\alpha \preceq \beta$ to hold between assignments α and β if $\mathcal{O}_\preceq(\vec{z}{\restriction}_\alpha, \vec{z}{\restriction}_\beta)$ evaluates to true. We require (a cutting planes proof) that $\mathcal{O}_\preceq$ is such that this defines a preorder, i.e., a reflexive and transitive relation. Adding new constraints C will be valid as long as we guarantee to preserve some f-minimal solution that is also minimal with respect to $\preceq$. In other words, $\preceq$ can be combined with f to define a preorder $\preceq_f$ on assignments by

$$\alpha \preceq_f \beta \quad \text{if} \quad \alpha \preceq \beta \text{ and } f{\restriction}_\alpha \leq f{\restriction}_\beta \ , \tag{4}$$

and we require that all derivation steps in the proof should preserve some solution that is minimal with respect to $\preceq_f$. The preorder defined by $\mathcal{O}_\preceq(\vec{u}, \vec{v})$ will only become important once we introduce our new *dominance-based strengthening rule* later in this section. For simplicity, up until that point the reader can assume that the pseudo-Boolean formula is $\mathcal{O}_\top \doteq \emptyset$ inducing the trivial preorder relating all assignments, though all proofs presented below work in full generality for the orders that will be introduced later.

A proof for (F, f) in our proof system consists of a sequence of *proof configurations* $(\mathscr{C}, \mathscr{D}, \mathcal{O}_\preceq, \vec{z}, v)$, where

- $\mathscr{C}$ is a set of pseudo-Boolean *core constraints*;
- $\mathscr{D}$ is another set of pseudo-Boolean *derived constraints*;
- $\mathcal{O}_\preceq$ is a PB formula encoding a preorder and $\vec{z}$ a set of literals on which this preorder will be applied; and
- v is the best value found so far for f.

The initial configuration is $(F, \emptyset, \mathcal{O}_\top, \emptyset, \infty)$. The distinction between $\mathscr{C}$ and $\mathscr{D}$ is only relevant when a nontrivial preorder is used; we will elaborate on this when discussing dominance. The intended semantics of f and v is that if $v < \infty$, then there exists a solution α satisfying F such that $f{\restriction}_\alpha \leq v$, and in this case the proof can make use of the constraint $f \leq v - 1$ in the search for better solutions. As long as the optimal solution has not been found, it should hold that f-minimal solutions of $\mathscr{C} \cup \mathscr{D}$ have the same objective value as f-minimal solutions of F. The precise relation is formalized in the notion of *valid configurations* as defined next.

Definition 1. *A configuration* $(\mathscr{C}, \mathscr{D}, \mathcal{O}_\preceq, \vec{z}, v)$ *is* (F, f)-valid *if the following conditions hold:*

1. *If $v < \infty$, then there is a total assignment ρ satisfying F such that $f{\restriction}_\rho \leq v$.*
2. *For every $v' < v$, it holds that the sets $F \cup \{f \leq v'\}$ and $\mathscr{C} \cup \{f \leq v'\}$ are equisatisfiable.*
3. *For every total assignment ρ satisfying the constraints $\mathscr{C} \cup \{f \leq v-1\}$, there exists a total assignment $\rho' \preceq_f \rho$ satisfying $\mathscr{C} \cup \mathscr{D} \cup \{f \leq v-1\}$.*

We will show that (F, f)-validity is an invariant of our proof system, i.e., that it is preserved by all derivation rules. Note that the two last items together imply that if the configuration $(\mathscr{C}, \mathscr{D}, \mathscr{O}_\preceq, \vec{z}, v)$ is such that v is not yet the value of an optimal solution, then f-minimal solutions of F and of $\mathscr{C} \cup \mathscr{D}$ have the same objective value, just as desired.

A proof in our proof system ends when the configuration $(\mathscr{C}, \mathscr{D}, \mathscr{O}_\preceq, \vec{z}, v^*)$ is such that $\mathscr{C} \cup \mathscr{D}$ contains contradiction $\bot \doteq 0 \geq 1$. In that case, either $v^* = \infty$ and F is unsatisfiable, or v^* is the optimal value (or $v^* = 0$ for a satisfiable decision problem). We state this as a formal theorem (but due to space constraints, proofs of all statements in this section can be found in (Bogaerts et al. 2022)).

Theorem 2. *Let F be a pseudo-Boolean formula and f an objective function. If $(\mathscr{C}, \mathscr{D}, \mathscr{O}_\preceq, \vec{z}, v^*)$ is an (F, f)-valid configuration with $\{0 \geq 1\} \subseteq \mathscr{C} \cup \mathscr{D}$, then*

- *F is unsatisfiable if and only if $v^* = \infty$; and*
- *if F is satisfiable, then there is an f-minimal solution α of F with objective value $f{\restriction}_\alpha = v^*$.*

We are now ready to give a formal description of the rules in our proof system.

Implicational Derivation Rule

If we can exhibit a derivation of the pseudo-Boolean constraint C from $\mathscr{C} \cup \mathscr{D} \cup \{f \leq v-1\}$ in our (slightly extended) version of cutting planes as described in Section 2 (i.e., in formal notation, if $\mathscr{C} \cup \mathscr{D} \cup \{f \leq v-1\} \vdash C$), then we can go from the configuration $(\mathscr{C}, \mathscr{D}, \mathscr{O}_\preceq, \vec{z}, v)$ to the configuration $(\mathscr{C}, \mathscr{D} \cup \{C\}, \mathscr{O}_\preceq, \vec{z}, v)$ by the *implicational derivation rule*. By the soundness of the cutting planes proof system, this means that $\mathscr{C} \cup \mathscr{D} \cup \{f \leq v-1\} \models C$, and so (F, f)-validity is preserved, but, more importantly, the cutting planes derivation provides a simple and efficient way for an algorithm to *verify* that this implication holds. This is a key feature of all rules in our proof system—not only are they sound, but the soundness of every rule application can be efficiently verified by checking a simple, syntactic object.

When doing proof logging, the solver would need to specify by which sequence of cutting planes derivation rules C was obtained. For practical purposes, though, it greatly simplifies matters that in many cases the verifier can figure out the required proof details automatically, meaning that the proof logger can just state the desired constraint without any further information. One important example of this is when C is a reverse unit propagation (RUP) constraint with respect to $\mathscr{C} \cup \mathscr{D} \cup \{f \leq v-1\}$. Another case is when C is literal-axiom-implied by some other constraint.

Objective Bound Update Rule

The *objective bound update rule* allows improving the estimate of what value can be achieved for the objective function f. We can go from $(\mathscr{C}, \mathscr{D}, \mathscr{O}_\preceq, \vec{z}, v)$ to $(\mathscr{C}, \mathscr{D}, \mathscr{O}_\preceq, \vec{z}, v')$ if we know an assignment α satisfying $\mathscr{C}$ such that $f{\restriction}_\alpha = v' < v$. When actually doing proof logging, the solver would specify such an assignment α, which would then be checked by the proof verifier (in our case *VeriPB*).

To argue that this rule preserves (F, f)-validity, we note that the last two items are trivially satisfied (they are weaker after applying the rule than before). The first item is satisfied since item 2 guarantees the existence of an α' satisfying F with an objective value that is at least as good as v'. Note that we have no guarantee that α' will be a solution to F. However, although we will not emphasize this point here, it follows from our formal treatment below that the proof system guarantees that such an f-minimal solution α' to the original formula F can be efficiently reconstructed from the proof (where efficiency is measured in the size of the proof).

Redundance-Based Strengthening Rule

The redundance-based strengthening rule allows deriving a constraint C from $\mathscr{C} \cup \mathscr{D}$ even if C is not implied, provided that it can be shown that any assignment α that satisfies $\mathscr{C} \cup \mathscr{D}$ can be transformed into another assignment $\alpha' \preceq_f \alpha$ that satisfies both $\mathscr{C} \cup \mathscr{D}$ and C (in case $\mathscr{O}_\preceq = \mathscr{O}_\top$, the condition $\alpha' \preceq_f \alpha$ just means that $f{\restriction}_{\alpha'} \leq f{\restriction}_\alpha$). This rule is borrowed from (Gocht and Nordström 2021), which in turn relies heavily on (Heule, Kiesl, and Biere 2017; Buss and Thapen 2019). We extend this rule here from decision problems to optimization problems in the natural way.

Formally, we say that C can be derived from $(\mathscr{C}, \mathscr{D}, \mathscr{O}_\preceq, \vec{z}, v)$ by *redundance-based strengthening*, or just *redundance* for brevity, if there is a substitution ω (which we will refer to as the *witness*) such that

$$\begin{aligned} &\mathscr{C} \cup \mathscr{D} \cup \{\neg C\} \vdash \\ &\quad (\mathscr{C} \cup \mathscr{D} \cup C){\restriction}_\omega \cup \{f{\restriction}_\omega \leq f\} \cup \mathscr{O}_\preceq(\vec{z}{\restriction}_\omega, \vec{z}) \ . \end{aligned} \tag{5}$$

Intuitively, (5) says that if some assignment α satisfies $\mathscr{C} \cup \mathscr{D}$ but falsifies C, then $\alpha' = \alpha \circ \omega$ still satisfies $\mathscr{C} \cup \mathscr{D}$ and also satisfies C. In addition, the condition $f{\restriction}_\omega \leq f$ ensures that $\alpha \circ \omega$ achieves an objective function value that is at least as good as that for α. This together with the constraints $\mathscr{O}_\preceq(\vec{z}{\restriction}_\omega, \vec{z})$ guarantees that $\alpha' \preceq_f \alpha$. For proof logging purposes, the witness ω as well as any non-immediate cutting planes derivations of constraints on the right-hand side of (5) would have to be specified, but, e.g., all RUP constraints or literal-axiom-implied constraints can be left to the verifier to check.

Proposition 3. *If C is derivable from an (F, f)-valid configuration $(\mathscr{C}, \mathscr{D}, \mathscr{O}_\preceq, \vec{z}, v)$ by redundance-based strengthening, then $(\mathscr{C}, \mathscr{D} \cup \{C\}, \mathscr{O}_\preceq, \vec{z}, v)$ is (F, f)-valid as well.*

Deletion Rule

We also need to be able to delete previously derived constraints. From a configuration $(\mathscr{C}, \mathscr{D}, \mathscr{O}_\preceq, \vec{z}, v)$ we can transition to $(\mathscr{C}', \mathscr{D}', \mathscr{O}_\preceq, \vec{z}, v)$ using the *deletion rule* if

1. $\mathscr{D}' \subseteq \mathscr{D}$ and

2. $\mathscr{C}' = \mathscr{C}$ or $\mathscr{C}' = \mathscr{C} \setminus \{C\}$ for some constraint C derivable via the redundance rule from $(\mathscr{C}', \emptyset, \mathscr{O}_{\preceq}, \vec{z}, v)$.

This last condition above perhaps seems slightly odd, but it is there since deleting arbitrary constraints could violate (F, f)-validity in two different ways. Firstly, it would allow finding better-than-optimal solutions. Secondly, and perhaps surprisingly, in combination with the dominance-based strengthening rule, which we will discuss below, arbitrary deletion is unsound, as it can turn satisfiable instances into unsatisfiable ones. This is illustrated in Example 5 further below.

To see that deletion preserves (F, f)-validity, it is clear that item 1 remains satisfied by deletion, as does the direction of item 2 that claims satisfiability of $\mathscr{C} \cup \{f \leq v'\}$. For the other direction of item 2 and for item 3, intuitively the redundance rule guarantees that solutions of the configuration after deletion can be mapped to solutions of the configuration before deletion that are at least as good.

An alternative to condition 2 would be to enforce the more restrictive demand $\mathscr{C}' \vdash \mathscr{C}$. However, this would prevent the use of some SAT preprocessing techniques such as bounded variable elimination (Eén and Biere 2005).

Transfer Rule

Constraints can always be moved from the derived set $\mathscr{D}$ to the core set $\mathscr{C}$ using the *transfer rule*, which allows a transition from $(\mathscr{C}, \mathscr{D}, \mathscr{O}_{\preceq}, \vec{z}, v)$ to $(\mathscr{C}', \mathscr{D}, \mathscr{O}_{\preceq}, \vec{z}, v)$ if $\mathscr{C} \subseteq \mathscr{C}' \subseteq \mathscr{C} \cup \mathscr{D}$. This clearly preserves (F, f)-validity.

The transfer rule together with deletion allows replacing constraints in the original formula with stronger constraints. For example, assume that $x + y \geq 1$ is in $\mathscr{C}$ and that we derive $x \geq 1$. Then we can move $x \geq 1$ from $\mathscr{D}$ to $\mathscr{C}$ and then delete $x + y \geq 1$. The required redundance check $\{x \geq 1, \neg(x + y \geq 1)\} \vdash \perp$ is immediate.

The rules discussed so far do not change $\mathscr{O}_{\preceq}$, and so any derivation using these rules only will operate with the trivial preorder $\mathscr{O}_{\top}$ imposing no conditions. The proof system defined in terms of these rules is a straightforward extension of *VeriPB* as developed in (Elffers et al. 2020; Gocht, McCreesh, and Nordström 2020; Gocht et al. 2020; Gocht and Nordström 2021) to an optimization setting. We next discuss the main contribution of this paper, namely the new dominance rule making use of the preorder $\mathscr{O}_{\preceq}$.

Dominance-Based Strengthening Rule

Any preorder $\preceq$ induces a strict order $\prec$ defined by $\alpha \prec \beta$ if $\alpha \preceq \beta$ and $\beta \not\preceq \alpha$. The relation $\prec_f$ obtained in this way from the preorder (4) coincides with what Chu and Stuckey (2015) call a *dominance relation* in the context of constraint optimisation. Our dominance rule allows deriving a constraint C from $\mathscr{C} \cup \mathscr{D}$ even if C is not implied, similar to the redundance rule. However, for the dominance rule an assignment α satisfying $\mathscr{C} \cup \mathscr{D}$ but falsifying C need only to be mapped to an assignment α' that satisfies $\mathscr{C}$, but not necessarily $\mathscr{D}$ or C. On the other hand, the new assignment α' should satisfy the strict inequality $\alpha' \prec_f \alpha$ and not just $\alpha' \preceq_f \alpha$ as in the redundance rule. To show that this new dominance rule preserves (F, f)-validity, we will prove that it is possible to construct an assignment that satisfies $\mathscr{C} \cup \mathscr{D} \cup \{C\}$ by iteratively applying the witness of the dominance rule, in combination with (F, f)-validity of the configuration before application of the dominance rule. As our base case, if α' satisfies $\mathscr{C} \cup \mathscr{D} \cup \{C\}$, we are done. Otherwise, since α' satisfies $\mathscr{C}$, by (F, f)-validity we are guaranteed the existence of an assignment α'' satisfying $\mathscr{C} \cup \mathscr{D}$ for which $\alpha'' \prec_f \alpha' \prec_f \alpha$ holds. If α'' still does not satisfy C, we can repeat the argument. In this way, we get a strictly decreasing sequence (with respect to $\prec_f$) of assignments. Since the set of possible assignments is finite, this sequence will eventually terminate.

Formally, we can derive C by dominance-based strengthening provided that there exists a substitution ω such that

$$\begin{aligned}&\mathscr{C} \cup \mathscr{D} \cup \{\neg C\} \vdash \\ &\quad \mathscr{C}\restriction_\omega \cup \mathscr{O}_{\preceq}(\vec{z}\restriction_\omega, \vec{z}) \cup \neg\mathscr{O}_{\preceq}(\vec{z}, \vec{z}\restriction_\omega) \cup \{f\restriction_\omega \leq f\} \ ,\end{aligned} \tag{6}$$

where $\mathscr{O}_{\preceq}(\vec{z}\restriction_\omega, \vec{z})$ and $\neg\mathscr{O}_{\preceq}(\vec{z}, \vec{z}\restriction_\omega)$ together state that $\alpha \circ \omega \prec \alpha$ for any assignment α. A minor technical problem is that the pseudo-Boolean formula $\mathscr{O}_{\preceq}(\vec{z}, \vec{z}\restriction_\omega)$ may contain multiple constraints, so that the negation of it is no longer a PB formula. To get around this, we split (6) into two separate conditions and shift $\neg\mathscr{O}_{\preceq}(\vec{z}, \vec{z}\restriction_\omega)$ to the premise of the implication, which eliminates the negation. Thus, the formal version of our *dominance-based strengthening rule*, or just *dominance rule* for brevity, says that we can go from $(\mathscr{C}, \mathscr{D}, \mathscr{O}_{\preceq}, \vec{z}, v)$ to $(\mathscr{C}, \mathscr{D} \cup \{C\}, \mathscr{O}_{\preceq}, \vec{z}, v)$ if there is a substitution ω such that the conditions

$$\mathscr{C} \cup \mathscr{D} \cup \{\neg C\} \vdash \mathscr{C}\restriction_\omega \cup \mathscr{O}_{\preceq}(\vec{z}\restriction_\omega, \vec{z}) \cup \{f\restriction_\omega \leq f\} \tag{7a}$$

$$\mathscr{C} \cup \mathscr{D} \cup \{\neg C\} \cup \mathscr{O}_{\preceq}(\vec{z}, \vec{z}\restriction_\omega) \vdash \perp \tag{7b}$$

are satisfied. Just as for the redundance rule, the witness ω as well as any non-immediate derivations would have to be specified in the proof log.

Proposition 4. *If C is derivable from an (F, f)-valid configuration $(\mathscr{C}, \mathscr{D}, \mathscr{O}_{\preceq}, \vec{z}, v)$ by dominance-based strengthening, then $(\mathscr{C}, \mathscr{D} \cup \{C\}, \mathscr{O}_{\preceq}, \vec{z}, v)$ is also (F, f)-valid.*

When introducing the deletion rule, we already mentioned that deleting arbitrary constraints can be unsound in combination with dominance-based strengthening. We now illustrate this phenomenon.

Example 5. *Consider the formula $F = \{p \geq 1\}$ with objective $f \doteq 0$ and the configuration*

$$(\mathscr{C}_1 = \{p \geq 1\}, \mathscr{D}_1 = \{p \geq 1\}, \mathscr{O}_{\preceq}, \{p\}, \infty) \ , \tag{8}$$

where $\mathscr{O}_{\preceq}(u, v)$ is defined as $\{v + \overline{u} \geq 1\}$. This configuration is (F, f)-valid and $\mathscr{C} \cup \mathscr{D}$ is satisfiable. If *we were allowed to delete constraints arbitrarily from $\mathscr{C}$, we could derive a configuration with $\mathscr{C}_2 = \emptyset$ and $\mathscr{D}_2 = \{p \geq 1\}$. However, now the dominance rule can derive $C \doteq \overline{p} \geq 1$, using the witness $\omega = \{p \mapsto 0\}$. To see that all conditions for applying dominance-based strengthening are indeed satisfied, we notice that conditions* (7a)–(7b) *simplify to*

$$\emptyset \cup \{p \geq 1\} \cup \{p \geq 1\} \vdash \emptyset \cup \{p + 1 \geq 1\} \cup \emptyset \tag{9a}$$

$$\emptyset \cup \{p \geq 1\} \cup \{p \geq 1\} \cup \{0 + \overline{p} \geq 1\} \vdash \perp \tag{9b}$$

respectively. Both claims clearly hold, meaning that we arrive at a configuration that contains both $p \geq 1$ and $\overline{p} \geq 1$.

Preorder Encodings

As mentioned before, $\mathcal{O}_{\preceq}$ is shorthand for a pseudo-Boolean formula $\mathcal{O}_{\preceq}(\vec{u}, \vec{v})$ over two sets of formal placeholder variables $\vec{u} = \{u_1, \ldots, u_n\}$ and $\vec{v} = \{v_1, \ldots, v_n\}$ of equal size, which should also match the size of $\vec{z}$ in the configuration. To use $\mathcal{O}_{\preceq}$ in a proof, it is required to show that this formula encodes a preorder. This is done by providing (in a proof preamble) cutting planes derivations establishing

$$\emptyset \vdash \mathcal{O}_{\preceq}(\vec{u}, \vec{u}) \tag{10a}$$

$$\mathcal{O}_{\preceq}(\vec{u}, \vec{v}) \cup \mathcal{O}_{\preceq}(\vec{v}, \vec{w}) \vdash \mathcal{O}_{\preceq}(\vec{u}, \vec{w}) \tag{10b}$$

where (10a) formalizes reflexivity and (10b) transitivity (and where notation like $\mathcal{O}_{\preceq}(\vec{v}, \vec{w})$ is shorthand for applying to $\mathcal{O}_{\preceq}(\vec{u}, \vec{v})$ the substitution ω that maps u_i to v_i and v_i to w_i, as discussed in Section 2). These two conditions guarantee that the relation $\preceq$ defined by $\alpha \preceq \beta$ if $\mathcal{O}_{\preceq}(\vec{z}\!\upharpoonright_{\alpha}, \vec{z}\!\upharpoonright_{\beta})$ forms a preorder on the set of assignments.

By way of example, to encode the lexicographic order $u_1 u_2 \ldots u_n \preceq_{\text{lex}} v_1 v_2 \ldots v_n$, we can use a single constraint

$$\mathcal{O}_{\preceq_{\text{lex}}}(\vec{u}, \vec{v}) \doteq \textstyle\sum_{i=1}^{n} 2^{n-i} \cdot (v_i - u_i) \geq 0 \ . \tag{11}$$

Reflexivity is vacuously true since $\mathcal{O}_{\preceq_{\text{lex}}}(\vec{u}, \vec{u}) \doteq 0 \geq 0$, and transitivity also follows easily since adding $\mathcal{O}_{\preceq_{\text{lex}}}(\vec{u}, \vec{v})$ and $\mathcal{O}_{\preceq_{\text{lex}}}(\vec{v}, \vec{w})$ yields $\mathcal{O}_{\preceq_{\text{lex}}}(\vec{u}, \vec{w})$.

A potential concern with encodings such as (11) is that coefficients can become very large as the number of variables in the order grows. It is perfectly possible to address this by allowing order encodings using auxiliary variables in addition to $\vec{u}$ and $\vec{v}$. We have chosen not to develop the theory for this in the current paper, however, since we feel that it makes the exposition unnecessarily complicated without adding anything of real significance to the scientific contribution.

Order Change Rule

The final proof rule that we need is a rule for introducing a nontrivial order, and it turns out that it can also be convenient to be able to use different orders at different points in the proof. Switching orders is possible, but to maintain soundness it is important to first clear the set $\mathcal{D}$ (after transferring the constraints we want to keep to $\mathcal{C}$). The reason for this is simple: if we allow arbitrary order changes, then the third item of (F, f)-validity would no longer hold, but when $\mathcal{D} = \emptyset$, it is trivially true.

Formally, provided that $\mathcal{O}_{\preceq^2}$ has been established to be a preorder (via cutting planes proofs for (10a) and (10b)), and provided that $\vec{z}_2$ is a list of variables of the size required by this order, it is allowed to go from the configuration $(\mathcal{C}, \emptyset, \mathcal{O}_{\preceq^1}, \vec{z}_1, v)$ to the configuration $(\mathcal{C}, \emptyset, \mathcal{O}_{\preceq^2}, \vec{z}_2, v)$ using the *order change* rule. As explained above, it is clear that this rule preserves (F, f)-validity.

This concludes the presentation of our proof system. Each rule has been shown to preserve (F, f)-validity, and the initial configuration is clearly (F, f)-valid. Therefore, by Theorem 2 our proof system is sound: whenever we can derive a configuration $(\mathcal{C}, \mathcal{D}, \mathcal{O}_{\preceq}, \vec{z}, v)$ such that $\mathcal{C} \cup \mathcal{D}$ contains $0 \geq 1$, it holds that v is the value of f in any f-minimal solution of F (or, for a decision problem, we have $v < \infty$ precisely when F is satisfiable). As mentioned above, in this case the full sequence of configurations $(\mathcal{C}, \mathcal{D}, \mathcal{O}_{\preceq}, \vec{z}, v)$ together with annotations about the derivation steps—including, in particular, any witnesses ω—contains all information needed to efficiently reconstruct such an f-minimal solution of F. It is also straightforward to show that our proof system is complete: after using the bound update rule to log an optimal solution v^*, it follows from the implicational completeness of cutting planes that contradiction can be derived from $F \cup \{f \leq v^* - 1\}$.

4 Applications

We now exhibit three applications that have not previously admitted efficient certification, and demonstrate that our new method can support simple, practical proof logging in each case. We first show that, by enhancing the *BreakID* tool for SAT solving (Devriendt et al. 2016) with *VeriPB* proof logging, we can cover the entire solving toolchain when symmetries are involved. We then revisit the Crystal Maze example from the introduction. Finally, we discuss how dominance-based strengthening can be used to support vertex domination reasoning in a maximum clique solver. All code for our implementations and experiments, as well as data and scripts for all plots, can be found at https://doi.org/10.5281/zenodo.6373986.

Symmetry Breaking in SAT Solvers

Symmetry handling has a long and successful history in SAT solving, with a wide variety of techniques considered by, e.g., Aloul, Sakallah, and Markov (2006); Benhamou and Saïs (1994); Benhamou et al. (2010); Devriendt et al. (2012); Devriendt, Bogaerts, and Bruynooghe (2017); Metin, Baarir, and Kordon (2019); Sabharwal (2009). These techniques were used to great effect in, e.g., the 2013 and 2016 editions of the *SAT competition*,[2] where the *SAT+UNSAT hard combinatorial track* and the *no-limit* track, respectively, were won by solvers employing symmetry breaking. However, the victory in 2013 can partly be explained by a small parser bug. For reasons such as this, proof logging is now obligatory in the main track of the SAT competition. While it is hard to overemphasize the importance of this development, it unfortunately means that symmetry breaking can no longer be used, since there is no way of efficiently certifying the correctness of such reasoning in *DRAT*. We will now explain how pseudo-Boolean reasoning with the dominance rule can provide proof logging for the *static symmetry breaking* techniques of Devriendt et al. (2016).

Let π be a permutation of the set of literals in a given CNF formula F (i.e., , a bijection on the set of literals), extended to (sets of) clauses in the obvious way. We say that π is a *symmetry* of F if it commutes with negation, i.e., $\pi(\overline{\ell}) = \overline{\pi(\ell)}$, and preserves satisfaction of F, i.e., $\alpha \circ \pi$ satisfies F if and only if α does. A *syntactic symmetry* in addition satisfies that $\pi(F) \doteq F\!\upharpoonright_{\pi} \doteq F$. As is standard, we only consider syntactic symmetries.

The most common way of breaking symmetries is by adding *lex-leader constraints* (Crawford et al. 1996). We

[2] www.satcompetition.org

here use $\preceq_{\text{lex}}$ to denote the lexicographic order on assignments induced by the sequence of variables $x_1, \ldots, x_m$. Given a set G of symmetries of F, a lex-leader constraint is a formula ψ_{LL} such that α satisfies ψ_{LL} if and only if $\alpha \preceq_{\text{lex}} \alpha \circ \pi$ for each $\pi \in G$. Let $\{x_{i_1}, \ldots, x_{i_n}\}$ be the *support* of π (i.e., all variables x such that $\pi(x) \neq x$), ordered so that $i_j \leq i_k$ if and only if $j \leq k$. Then the constraints

$$y_0 \geq 1 \tag{12a}$$

$$\overline{y}_{j-1} + \overline{x}_{i_j} + \pi(x_{i_j}) \geq 1 \qquad 1 \leq j \leq n \tag{12b}$$

$$\overline{y}_j + y_{j-1} \geq 1 \qquad 1 \leq j < n \tag{12c}$$

$$\overline{y}_j + \overline{\pi(x_{i_j})} + x_{i_j} \geq 1 \qquad 1 \leq j < n \tag{12d}$$

$$y_j + \overline{y}_{j-1} + \overline{x}_{i_j} \geq 1 \qquad 1 \leq j < n \tag{12e}$$

$$y_j + \overline{y}_{j-1} + \pi(x_{i_j}) \geq 1 \qquad 1 \leq j < n \tag{12f}$$

form a lex-leader constraint for π, where each y_j is a fresh variable representing that α and $\alpha \circ \pi$ are equal up to x_{i_j}, and where (12b) does the actual breaking.

To derive this in our proof system, assume that we have a configuration $(\mathscr{C}, \mathscr{D}, \mathcal{O}_{\preceq}, \vec{x}, v)$ where assignments are compared lexicographically on $\vec{x} = \{x_1, \ldots, x_m\}$ according to $\mathcal{O}_{\preceq}$ as in (11). Let π be a syntactic symmetry of $\mathscr{C}$ (i.e., such that $\mathscr{C}\restriction_\pi \doteq \mathscr{C}$) with support contained in $\vec{x}$. In this case

$$C_{LL} \doteq \textstyle\sum_{i=1}^{m} 2^{m-i} \cdot (\pi(x_i) - x_i) \geq 0 \tag{13}$$

expresses that $\pi(\vec{x})$ is greater than or equal to $\vec{x}$. Noting that SAT problems lack an objective function, we can apply the dominance rule with $\omega = \pi$ to derive C_{LL}. To see that (7a) holds, we note that $\neg C_{LL}$ expresses that $\vec{x}$ is strictly larger than $\pi(\vec{x})$, and hence this implies $\mathcal{O}_{\preceq}(\vec{x}\restriction_\pi, \vec{x})$. Clearly, (7b) is true as well, since its premise contains both C_{LL} and its negation. Since the y-variables are fresh, we can also derive the constraints (12a) and (12c)–(12f) as explained by Gocht and Nordström (2021). It remains to show how to deduce the constraints (12b) from C_{LL}.

As before, assume that the support of π is $\{x_{i_1}, \ldots, x_{i_n}\}$ with $i_j \leq i_k$ if and only if $j \leq k$. Note first that for all x_i that are not in the support of π, the term $\pi(x_i) - x_i$ disappears since $\pi(x_i) = x_i$ and thus C_{LL} simplifies to

$$\textstyle\sum_{j=1}^{n} 2^{m-i_j} \cdot (\pi(x_{i_j}) - x_{i_j}) \geq 0 \ , \tag{14}$$

which can only hold if the term with the largest coefficient is non-negative. It follows that C_{LL} implies $\pi(x_{i_1}) - x_{i_1} \geq 0$ by reverse unit propagation (RUP), and hence can be derived from our current configuration with the implicational rule, also yielding the weaker constraint (12b) with $j = 1$.

To deal with $j > 1$, we define

$$C_{LL}(0) \doteq C_{LL} \tag{15a}$$

$$C_{LL}(k) \doteq C_{LL}(k-1) + 2^{m-i_k} \cdot (12d\,[j = k]) \tag{15b}$$

where $(12d\,[j = k])$ denotes substitution of j by k in (12d). Simplifying $C_{LL}(k)$ yields

$$\textstyle\sum_{i=1}^{k} 2^{m-i}\overline{y_j} + \sum_{i=k+1}^{m} 2^{m-i} \cdot (\pi(x_i) - x_i) \geq 0 \ , \tag{16}$$

which, in combination with all constraints (12c), directly entails (12b) with $j = k$. To see this, note that if y_k is false, then (12b) is trivially true for $j = k + 1$. On the other hand, if y_k is true, then so are all the preceding y-variables, and the dominant term in $C_{LL}(k)$ becomes $\pi(x_{i_k}) - x_{i_k}$, which implies (12b) for $j = k$ analogously to the case for $j = 1$.

It is important to note here that the order is set once and is the same for all symmetries $\pi \in G$ to be broken. Since constraints are added only to $\mathscr{D}$, dominance rule applications for different symmetries will not interfere with each other. Furthermore, contrary to the symmetry logging approach of Heule, Hunt Jr., and Wetzler (2015), handling a symmetry once is enough to guarantee complete breaking. See (Bogaerts et al. 2022) for a worked-out *VeriPB* example of symmetry breaking together with explanations of how the proof logging syntax matches rules in our proof system.

To validate our approach, we implemented *VeriPB* proof logging for the symmetry breaking method in *BreakID*, and modified *Kissat*[3] to output *VeriPB*-proofs (since the redundance rule is a generalization of the RAT rule, this required only minor changes). We used a simpler version of the deletion rule that only guarantees to prove a lower bound on the objective value—if this lower bound is infinity, this certifies that decision problems are unsatisfiable (see the discussion of *weak* (F, f)*-validity* in (Bogaerts et al. 2022)).

Out of all the benchmark instances from all the SAT competitions since 2016, we selected all instances in which at least one symmetry was detected; there were 1089 such instances in total. We performed our experiments on machines with dual Intel Xeon E5-2697A v4 processors with 512GBytes RAM and solid-state drive (SSD), running Ubuntu 20.04. We ran twenty instances in parallel on each machine, limiting each instance to 16GBytes RAM, and with a timeout of 5,000s for solving and 100,000s for verification.

The left plot in Figure 2 displays the performance overhead for symmetry breaking, comparing for each instance the running time with and without proof logging. For most instances, the overhead is negligible (99% of instances are at most 32% slower). The other two plots in Figure 2 display the relationship between the time needed to generate a proof (both for SAT and UNSAT instances) and to verify the correctness of this proof. When only considering verification of the symmetry breaking (middle plot), 1058 instances out of 1089 could be verified, 2 timed out, and 29 terminated due to running out of memory. 75% of the instances could be verified within 3.2 times the solving time and 95% within a factor 20. The time needed for verification is thus considerably longer than solving time, but still practical in the majority of cases. After symmetry breaking, 721 instances could be solved with the SAT solver (right plot) and we could verify 671 instances, while for 33 instances verification timed out and for 17 instances the verifier ran out of memory. Notably, 84 instances could only be solved with symmetry breaking, out of which we could verify 81.

Symmetries in Constraint Programming

In the general setting considered in constraint programming, we must deal with variables with larger (non-Boolean) do-

[3]http://fmv.jku.at/kissat/

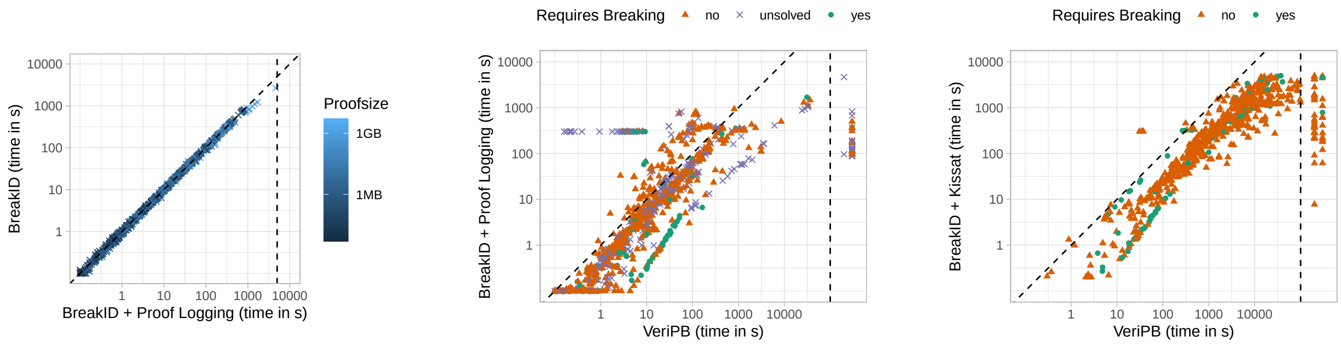

Figure 2: On the left, performance overhead due to proof logging symmetry breaking. In the center, performance of verifying symmetry breaking. On the right, performance of verifying symmetry breaking and SAT solving. Points behind the vertical dashed line indicate timeouts (left) and out of memory (right).

mains and with rich constraints supported by propagation algorithms. One might think that a proof system based upon Boolean variables and linear inequalities would not be suitable for this larger class of problem. However, Elffers et al. (2020) showed how to use *VeriPB* for constraint satisfaction problems by first encoding variables and constraints in pseudo-Boolean form, and then constructing cutting planes proofs to justify the behaviour of propagators such as *alldifferent*. Similarly, the work we present here can also be applied to constraint satisfaction and optimisation problems.

Recall the symmetry breaking constraints proposed for the Crystal Maze puzzle in the introduction. Given the difficulties in knowing which combinations of constraints are valid, it would be desirable if these constraints could be *introduced as part of a proof*, rather than taken as axioms. This would give a modeller immediate feedback as to whether the constraints have been chosen correctly. Our proof system is indeed powerful enough to express all three of the examples we presented, and we have implemented a small tool which can write out the appropriate proof fragments; this allows the entire Crystal Maze example to be verified with *VeriPB*. Interestingly, although symmetries can be broken in different ways in high-level CP models (including through lexicographic and value precedence constraints), when we encode the problem in pseudo-Boolean form these differences largely disappear, and after creating a suitable order we can re-use the SAT techniques just discussed. So, although a full proof-logging constraint solver does not yet exist, we can confidently claim that symmetries no longer block this goal.

Lazy Global Domination in Maximum Clique

Gocht et al. (2020) showed how *VeriPB* can be used to implement proof logging for a wide range of maximum clique algorithms, observing that the cutting planes proof system is rich enough to justify a wide range of bound and inference functions used by various solvers (despite cutting planes not knowing what a graph or clique is). However, there is one clique-solving technique in the literature that is *not* amenable to cutting planes reasoning. In order to solve problem instances that arise from a distance-relaxed clique-finding problem, McCreesh and Prosser (2016) enhanced their maximum clique algorithm with a *lazy global domination* rule that works as follows. Suppose that the solver has constructed a candidate clique C and is considering to extend C by two vertices v and w, where the neighbourhood of v excluding w is a (non-strict) superset of the neighbourhood of w excluding v. Then if the solver first tries v and rejects it, there is no need to branch on w as well.

In principle, it should be possible to introduce additional constraints justifying this kind of reasoning in advance using redundance-based strengthening, without the need for the full dominance breaking framework in Section 3 (with some technicalities involving consistent orderings for tiebreaking). However, due to the prohibitive cost of computing the full vertex dominance relation in advance, McCreesh and Prosser instead implement a form of *lazy* dominance detection, which only triggers following a backtrack.

To provide proof logging for this, we must instead be able to introduce vertex dominance constraints precisely when they are used. It is hard to see how to achieve this with the redundance rule, but it is possible using dominance-based strengthening: we have implemented this in the proof logging maximum clique solver in (Gocht et al. 2020), as discussed in more detail in (Bogaerts et al. 2022).

5 Conclusion

In this paper, we show that the pseudo-Boolean proof logging method in *VeriPB* (Gocht and Nordström 2021) can be extended with a rule for dominance breaking so as to efficiently certify unlimited symmetry breaking in SAT solving, even when combined with XOR and cardinality reasoning. A natural next question is whether our method is strong enough to capture other techniques such as those used for MaxSAT; several such techniques, such as the *dominating unit-clause rule* (Niedermeier and Rossmanith 2000) and *group subsumed label elimination* (Leivo, Berg, and Järvisalo 2020), appear to be special cases of dominance, making this a promising direction. Our work also contributes towards extending proof logging techniques from SAT to other combinatorial solving paradigms such as constraint programming and dedicated graph solving algorithms.

Acknowledgements

Bart Bogaerts was partially supported by Fonds Wetenschappelijk Onderzoek – Vlaanderen (project G070521N). Ciaran McCreesh was supported by a Royal Academy of Engineering Research Fellowship. Stephan Gocht and Jakob Nordström were supported by the Swedish Research Council grant 2016-00782, and Jakob Nordström also received funding from the Independent Research Fund Denmark grant 9040-00389B. Part of this work was carried out while taking part in the semester program *Satisfiability: Theory, Practice, and Beyond* in the spring of 2021 at the Simons Institute for the Theory of Computing at UC Berkeley.

References

Achterberg, T.; and Wunderling, R. 2013. Mixed Integer Programming: Analyzing 12 Years of Progress. In Jünger, M.; and Reinelt, G., eds., *Facets of Combinatorial Optimization*, 449–481. Springer.

Akgün, Ö.; Gent, I. P.; Jefferson, C.; Miguel, I.; and Nightingale, P. 2018. Metamorphic Testing of Constraint Solvers. In *Proceedings of the 24th International Conference on Principles and Practice of Constraint Programming (CP '18)*, volume 11008 of *LNCS*, 727–736. Springer.

Alkassar, E.; Böhme, S.; Mehlhorn, K.; Rizkallah, C.; and Schweitzer, P. 2011. An Introduction to Certifying Algorithms. *it - Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik*, 53(6): 287–293.

Aloul, F. A.; Sakallah, K. A.; and Markov, I. L. 2006. Efficient symmetry breaking for Boolean satisfiability. *IEEE Transactions on Computers*, 55(5): 549–558.

Baptiste, P.; and Pape, C. L. 1997. Constraint Propagation and Decomposition Techniques for Highly Disjunctive and Highly Cumulative Project Scheduling Problems. In *Proceedings of the 3rd International Conference on Principles and Practice of Constraint Programming (CP '97)*, volume 1330 of *LNCS*, 375–389. Springer.

Benhamou, B.; Nabhani, T.; Ostrowski, R.; and Saïdi, M. R. 2010. Enhancing Clause Learning by Symmetry in SAT Solvers. In *Proceedings of the 22nd IEEE International Conference on Tools with Artificial Intelligence (ICTAI '10)*, volume 1, 329–335.

Benhamou, B.; and Saïs, L. 1994. Tractability through Symmetries in Propositional Calculus. *Journal of Automated Reasoning*, 12(1): 89–102.

Biere, A. 2006. TraceCheck. http://fmv.jku.at/tracecheck/.

Biere, A.; Heule, M. J. H.; van Maaren, H.; and Walsh, T., eds. 2021. *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2nd edition.

Bogaerts, B.; Gocht, S.; McCreesh, C.; and Nordström, J. 2022. Certified Symmetry and Dominance Breaking for Combinatorial Optimisation. https://arxiv.org/abs/2203.12275. Full-length version.

Brummayer, R.; Lonsing, F.; and Biere, A. 2010. Automated Testing and Debugging of SAT and QBF Solvers. In *Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing (SAT '10)*, volume 6175 of *LNCS*, 44–57. Springer.

Bulhões, T.; Sadykov, R.; and Uchoa, E. 2018. A Branch-and-price Algorithm for the Minimum Latency Problem. *Computers & Operations Research*, 93: 66–78.

Buss, S. R.; and Nordström, J. 2021. Proof Complexity and SAT Solving. In (Biere et al. 2021), chapter 7, 233–350.

Buss, S. R.; and Thapen, N. 2019. DRAT Proofs, Propagation Redundancy, and Extended Resolution. In *Proceedings of the 22nd International Conference on Theory and Applications of Satisfiability Testing (SAT '19)*, volume 11628 of *LNCS*, 71–89. Springer.

Chu, G.; and Stuckey, P. J. 2015. Dominance Breaking Constraints. *Constraints*, 20(2): 155–182. Preliminary version in *CP '12*.

Cook, W.; Coullard, C. R.; and Turán, G. 1987. On the Complexity of Cutting-Plane Proofs. *Discrete Applied Mathematics*, 18(1): 25–38.

Cook, W.; Koch, T.; Steffy, D. E.; and Wolter, K. 2013. A Hybrid Branch-and-Bound Approach for Exact Rational Mixed-Integer Programming. *Mathematical Programming Computation*, 5(3): 305–344.

Crawford, J. M.; Ginsberg, M. L.; Luks, E. M.; and Roy, A. 1996. Symmetry-Breaking Predicates for Search Problems. In *Proceedings of the 5th International Conference on Principles of Knowledge Representation and Reasoning (KR '96)*, 148–159.

Cruz-Filipe, L.; Heule, M. J. H.; Hunt, W. A.; Kaufmann, M.; and Schneider-Kamp, P. 2017. Efficient Certified RAT Verification. In *Proceedings of the 26th International Conference on Automated Deduction (CADE-26)*, volume 10395 of *LNCS*, 220–236. Springer.

Cruz-Filipe, L.; Marques-Silva, J.; and Schneider-Kamp, P. 2017. Efficient Certified Resolution Proof Checking. In *Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '17)*, volume 10205 of *LNCS*, 118–135. Springer.

Demeulemeester, E. L.; and Herroelen, W. S. 2002. *Project Scheduling: A Research Handbook*, volume 49 of *International Series in Operations Research & Management Science*. Kluwer Academic Publishers.

Devriendt, J.; Bogaerts, B.; and Bruynooghe, M. 2017. Symmetric Explanation Learning: Effective Dynamic Symmetry Handling for SAT. In *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT '17)*, volume 10491 of *LNCS*, 83–100. Springer.

Devriendt, J.; Bogaerts, B.; Bruynooghe, M.; and Denecker, M. 2016. Improved Static Symmetry Breaking for SAT. In *Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing (SAT '16)*, volume 9710 of *LNCS*, 104–122. Springer.

Devriendt, J.; Bogaerts, B.; De Cat, B.; Denecker, M.; and Mears, C. 2012. Symmetry Propagation: Improved Dynamic Symmetry Breaking in SAT. In *Proceedings of the 24th*

IEEE International Conference on Tools with Artificial Intelligence (ICTAI '12), 49–56.

Eén, N.; and Biere, A. 2005. Effective Preprocessing in SAT Through Variable and Clause Elimination. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT '05)*, volume 3569 of *LNCS*, 61–75. Springer.

Elffers, J.; Gocht, S.; McCreesh, C.; and Nordström, J. 2020. Justifying All Differences Using Pseudo-Boolean Reasoning. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI '20)*, 1486–1494.

Garcia de la Banda, M.; Stuckey, P. J.; Van Hentenryck, P.; and Wallace, M. 2014. The future of optimization technology. *Constraints*, 19(2): 126–138.

Gebser, M.; Kaminski, R.; and Schaub, T. 2011. Complex Optimization in Answer Set Programming. *Theory and Practice of Logic Programming*, 11(4–5): 821–839.

Gent, I. P.; Petrie, K. E.; and Puget, J. 2006. Symmetry in Constraint Programming. In Rossi, F.; van Beek, P.; and Walsh, T., eds., *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, 329–376. Elsevier.

Gillard, X.; Schaus, P.; and Deville, Y. 2019. SolverCheck: Declarative Testing of Constraints. In *Proceedings of the 25th International Conference on Principles and Practice of Constraint Programming (CP '19)*, volume 11802 of *LNCS*, 565–582. Springer.

Gocht, S.; McBride, R.; McCreesh, C.; Nordström, J.; Prosser, P.; and Trimble, J. 2020. Certifying Solvers for Clique and Maximum Common (Connected) Subgraph Problems. In *Proceedings of the 26th International Conference on Principles and Practice of Constraint Programming (CP '20)*, volume 12333 of *LNCS*, 338–357. Springer.

Gocht, S.; McCreesh, C.; and Nordström, J. 2020. Subgraph Isomorphism Meets Cutting Planes: Solving With Certified Solutions. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI '20)*, 1134–1140.

Gocht, S.; and Nordström, J. 2021. Certifying Parity Reasoning Efficiently Using Pseudo-Boolean Proofs. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI '21)*, 3768–3777.

Goldberg, E.; and Novikov, Y. 2003. Verification of Proofs of Unsatisfiability for CNF Formulas. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '03)*, 886–891.

Heule, M. J. H.; Hunt Jr., W. A.; and Wetzler, N. 2013a. Trimming While Checking Clausal Proofs. In *Proceedings of the 13th International Conference on Formal Methods in Computer-Aided Design (FMCAD '13)*, 181–188.

Heule, M. J. H.; Hunt Jr., W. A.; and Wetzler, N. 2013b. Verifying Refutations with Extended Resolution. In *Proceedings of the 24th International Conference on Automated Deduction (CADE-24)*, volume 7898 of *LNCS*, 345–359. Springer.

Heule, M. J. H.; Hunt Jr., W. A.; and Wetzler, N. 2015. Expressing Symmetry Breaking in DRAT Proofs. In *Proceedings of the 25th International Conference on Automated Deduction (CADE-25)*, volume 9195 of *LNCS*, 591–606. Springer.

Heule, M. J. H.; Kiesl, B.; and Biere, A. 2017. Short Proofs Without New Variables. In *Proceedings of the 26th International Conference on Automated Deduction (CADE-26)*, volume 10395 of *LNCS*, 130–147. Springer.

Hoogeboom, M.; Dullaert, W.; Lai, D.; and Vigo, D. 2020. Efficient Neighborhood Evaluations for the Vehicle Routing Problem with Multiple Time Windows. *Transportation Science*, 54(2): 400–416.

Jouglet, A.; and Carlier, J. 2011. Dominance Rules In Combinatorial Optimization Problems. *European Journal of Operational Research*, 212(3): 433–444.

Leivo, M.; Berg, J.; and Järvisalo, M. 2020. Preprocessing in Incomplete MaxSAT Solving. In *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI '20)*, 347–354.

McConnell, R. M.; Mehlhorn, K.; Näher, S.; and Schweitzer, P. 2011. Certifying Algorithms. *Computer Science Review*, 5(2): 119–161.

McCreesh, C.; and Prosser, P. 2016. Finding Maximum k-Cliques Faster Using Lazy Global Domination. In *Proceedings of the 9th Annual Symposium on Combinatorial Search (SOCS '16)*, 72–80.

Metin, H.; Baarir, S.; and Kordon, F. 2019. Composing Symmetry Propagation and Effective Symmetry Breaking for SAT Solving. In *Proceedings of the 11th International NASA Formal Methods Symposium (NFM '19)*, volume 11460 of *LNCS*, 316–332. Springer.

Niedermeier, R.; and Rossmanith, P. 2000. New Upper Bounds for Maximum Satisfiability. *Journal of Algorithms*, 36(1): 63–88.

Sabharwal, A. 2009. SymChaff: Exploiting Symmetry in a Structure-Aware Satisfiability Solver. *Constraints*, 14(4): 478–505. Preliminary version in *AAAI '05*.

Tchinda, R. K.; and Djamégni, C. T. 2020. On Certifying the UNSAT Result of Dynamic Symmetry-Handling-Based SAT Solvers. *Constraints*, 25(3–4): 251–279.

Walsh, T. 2006. General Symmetry Breaking Constraints. In *Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming (CP '06)*, volume 4204 of *LNCS*, 650–664. Springer.

Walsh, T. 2012. Symmetry Breaking Constraints: Recent Results. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI '12)*, 2192–2198.

Wetzler, N.; Heule, M. J. H.; and Hunt Jr., W. A. 2014. DRAT-trim: Efficient Checking and Trimming Using Expressive Clausal Proofs. In *Proceedings of the 17th Internatjuional Conference on Theory and Applications of Satisfiability Testing (SAT '14)*, volume 8561 of *LNCS*, 422–429. Springer.