Certified Branch-and-Bound MaxSAT Solving

Dieter Vandesande
Vrije Universiteit Brussel, Belgium

Jordi Coll
Universitat de Girona, Spain

Chu-Min Li
Université de Picardie Jules Verne, France

Bart Bogaerts
Vrije Universiteit Brussel, Belgium

Abstract

Over the past few decades, there has been a remarkable improvement in the performance of combinatorial solvers, which has led to their practical usage in real-world applications. In some of these applications, the correctness of the solver's result is of utmost importance. Unfortunately, the reality is different: the algorithmic complexity of modern solvers enables bugs to sneak into the source code. For Satisfiability Checking (SAT), this problem was mitigated by letting the solver write down a formal proof of correctness of the obtained solution, which is also known as proof logging. However, for more expressive fields such as MaxSAT, which is the optimization variant of SAT, proof logging had not yet had its breakthrough until recently, when the proof system VeriPB was put forward as a good candidate to serve as a general-purpose proof system for MaxSAT solvers. In this paper, we show how VeriPB can be used as a proof system to let the Branch-and-Bound MaxSAT solver MaxCDCL produce proofs of optimality for its solutions. This is the first state-of-the-art MaxSAT solver that implements the Branch-and-Bound approach, opposed to the SAT-based solvers with proof logging in earlier work. We also show how to use VeriPB to add proof logging for an encoding of the model improving constraint into CNF based on Multi-valued Decision Diagrams (MDD).

2012 ACM Subject Classification Theory of computation → Logic

Keywords and phrases Combinatorial optimization, maximum satisfiability, branch-and-bound, certifying algorithms, proof logging

Digital Object Identifier 10.4230/LIPIcs.Soft 2024.2024.

Funding This work was partially supported by Fonds Wetenschappelijk Onderzoek – Vlaanderen (project G070521N). This work is also supported by grant PID2021-122274OB-I00 funded by grant MICIU/AEI/10.13039/501100011033 and by ERDF, EU.

1 Introduction

With ever increasingly efficient solvers being developed in various areas concerned with combinatorial search and optimization, we have now effectively arrived in a situation where NP hard problems are routinely tackled in practice. This maturation has resulted in solvers being deployed in various practical use cases, including safety-critical applications or making life-affecting decisions (e.g., verifying software that drives our transportation infrastructure [16], checking correctness of plans for the operation of the reaction control system of the space shuttle [32], or matching donors and recipients for kidney transplants [29]). For this reason, it is of utmost importance that the results produced by such solvers are guaranteed to be correct. Unfortunately, this is not always the case; in fact, there have been numerous reports of solvers outputting infeasible solutions, or falsely claiming optimality or the absence of solutions [2, 8, 9, 13, 18, 26].

First International Workshop on Discrete Optimization with Soft Constraints.
Editors: Simon de Givry and Javier Larrosa; Article No. ; pp. :1–:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

This problem calls for a systematic solution. The most evident solution would be to use *formal verification*; that is, using a proof assistant [3, 31, 34, 15] to prove that a solver is correct. However, in practice, using formal verification means sacrificing performance [17, 22], which is precisely what has led to the success of combinatorial optimization.

Instead, what we believe to be the way forward is to use *certifying algorithms* [30], an idea that is also known as *proof logging* in the setting of combinatorial optimization. With proof logging, the solver at hand does not only produce an answer (e.g., an optimal solution for optimization problems), but also a *proof of correctness* of this answer that can easily (in terms of the size of the proof) be verified by an independent tool (known as the *proof checker*). Next to guaranteeing correctness, proof logging is also very useful as a software development methodology: it facilitates advanced *testing* and *debugging* of solver code. Moreover, the produced proofs can be seen as an auditable record of how and why a certain conclusion was reached.

Proof logging has been pioneered in the field of Boolean satisfiability (SAT), where numerous proof formats and proof checkers (including formally verified checkers) have seen the light of day [4, 21, 23, 37, 14], with a breakthrough moment when it was used to resolve the Pythagorean triple problem, resulting in the "largest math proof ever" [24]. Moreover, for years, proof logging has been mandatory in (the main tracks) of the yearly solving competition, reflecting the fact that the community effectively demands that all SAT solvers be certifying.

In this paper, we are concerned with *maximum satisfiability* (MaxSAT), the optimization variant of SAT. In MaxSAT, proof logging is less well-spread. Only recently was the VeriPB proof system proposed as a general-purpose proof logging methodology for MaxSAT [36]. VeriPB has been successfully applied for adding proof logging to MaxSAT solvers employing *solution-improving search* [36, 35](where a SAT oracle is repeatedly queried each time searching for a solution that improves upon the previously best found solution) as well as for *core-guided search* (where a SAT oracle is repeatedly queried under ever-relaxing optimistic assumptions). Next to these two search paradigms, in MaxSAT also *implicit hitting set* and *branch-and-bound* are used in state-of-the-art MaxSAT solvers.

In this paper, we are concerned with bringing VeriPB-based certification to branch-and-bound search [28], thereby covering another search paradigm for MaxSAT solving. Modern branch-and-bound solvers combine conflict-driven clause learning (CDCL) [33] with a clever bounding function that determines whether the current search branch is "hopeless", meaning that there are no assignments that improve upon the best solution found so far. Since it is well-known how to certify CDCL search with VeriPB, the main challenge is to certify the conclusions of the bounding function. This bounding function makes use of look-ahead search to generate (conditional) unsatisfiable cores (literals that cannot be true together). These cores are then combined to get an estimate of the best possible objective value that is still achievable. Especially in the unweighted case, this combination of cores relies on very subtle arguments, which we clearly spell out and formalize in the VeriPB proof system.

In order to bring this certification to practice, there are more hurdles to overcome: state-of-the-art branch-and-bound solvers employ several other clever tricks to speed up the solving process, such as pre-processing methods as well as integrating ideas from other search paradigms. One particular technique that proved to be challenging is the use of *multi-valued decision diagrams* (MDDs) [6] in order to create a CNF encoding of a solution-improving constraint (which is enabled or disabled heuristically depending on the size of the instance at hand). The MDD-based encoding used by MaxCDCL generalizes the encoding of Binary Decision Diagrams (BDDs) (by allowing splits on sets of variables instead of a single variable) proposed in [1]. From the perspective of proof logging, the main challenge here is to prove that these constraints are indeed equivalent. As an example, consider the constraints

$$12x_1 + 5x_2 + 4x_3 \geq 6$$

and

$$12x_1 + 5x_2 + 4x_3 \geq 9.$$

Taking into account that the variables take values in $\{0, 1\}$, it is not hard to see that these constraints are equivalent: all combinations of truth assignments that lead to the left-hand side taking a value at least six, must assign it a value of at least nine. In other words: the left-hand side cannot take values 6, 7, or 8. In general, checking whether such a linear expression can take a specific value (e.g., 7) is well-known to be NP hard, but BDD (and MDD) construction algorithms will often detect this efficiently (in the worst-case this can take exponential time, which is why such MDD-based encodings will only be enabled for constraints with few variables and small coefficients). The main question of interest for us is how to convince a proof checker of the fact that these two constraints are equivalent, without doing a substantial amount of additional work. We achieve this using an algorithm that proves this property for all nodes in an MDD in a linear pass over its representation.

We implemented some of our algorithms in MaxCDCL, which is the only branch-and-bound solver that participated in last year's MaxSAT evaluation [25], where it proved to be one of the top performing solvers (it won the weighted track and ended second on the unweighted track). We intend to experimentally evaluate this work, but currently it is still work in progress: while the theory has been worked out and a partial implementation is available, we have no concrete results to report yet.

The rest of this paper is structured as follows. In Section 2, we recall some preliminaries about MaxSAT solving and VeriPB-based proof logging. Section 3 is devoted to presenting the core of the MaxCDCL algorithm, focusing on look-ahead–based bounding, as well as explaining how to get a certifying version of this. In Section 4, we explain how BDDs and MDDs are used to encode PB constraints. Section 5 concludes the paper. Formal details and proofs are often omitted, but can be found in the supplementary material.

2 Preliminaries

We first recall some concepts from pseudo-Boolean optimization and MaxSAT solving. Afterwards, we introduce the VeriPB proof system. For a full exposition, we refer the reader to [11, 27, 7].

In this paper, all variables are assumed to be *Boolean*; meaning they take a value in $\{0, 1\}$. A *literal* ℓ is a Boolean variable x or its negation $\overline{x}$. A *pseudo-Boolean constraint* C is a 0–1 integer linear inequality $\sum_i w_i \ell_i \geq A$. Without loss of generality, we will often assume our constraints to be *normalized*, meaning that the ℓ_i are different literals and all coefficients w_i and the degree A are non-negative. A *formula* is a conjunction of PB constraints. A *clause* is a special case of a PB constraint having all w_i and A equal to one. A *cardinality constraint* is a PB constraint where all w_i are one. If L is a set of literals, we will sometimes simply write L as a shorthand for $\sum_{\ell \in L} \ell$ and thus write constraints such as $L + 3K \geq 42$, meaning $\sum_{\ell \in L} \ell + \sum_{\ell \in K} 3 \cdot \ell \geq 42$.

A *(partial) assignment* α is a (partial) function from the set of variables to $\{0, 1\}$; it is extended to literals in the obvious way. We write $C\restriction_\alpha$ for the constraint obtained from C by substituting all assigned variables x by their assigned value $\alpha(x)$. A constraint C is *satisfied* under α if $\sum_{\alpha(\ell_i)=1} w_i \geq A$. A formula F is satisfied under α if all of its constraints are. We say that F *implies* C (and write $F \models C$) if all assignments that satisfy F also satisfy C.

A *pseudo-Boolean optimization* instance consists of a formula F and a linear term $\mathcal{O} = \sum_i v_i b_i$ (called the *objective*) to be minimized, where v_i are integers and b_i are different literals. If ℓ is a literal, we write $w_{\mathcal{O}}(\ell)$ for the weight of ℓ in $\mathcal{O}$, i.e., $w_{\mathcal{O}}(b_i) = v_i$ for all i and $w_{\mathcal{O}}(\ell) = 0$ for all other literals. In this paper, we view MaxSAT as the special case of pseudo-Boolean optimization where F is a conjunction of clauses.

For a pseudo-Boolean optimization instance $(F, \mathcal{O})$, the VeriPB proof system maintains a *proof configuration* $(\mathcal{C}, \mathcal{D})$ where $\mathcal{C}$ (standing for *core*) and $\mathcal{D}$ (standing for *derived*) are sets of constraints (initialized as F and $\emptyset$, respectively). Constraints can be moved from $\mathcal{D}$ to $\mathcal{C}$ but not vice versa. We are allowed to update the configuration using the cutting planes proof system [12]:

Literal Axioms: For any literal, we can add $\ell_i \geq 0$ to $\mathcal{D}$.

Linear Combination: Given two PB constraints C_1 and C_2 in $\mathcal{C} \cup \mathcal{D}$, we can add a positive linear combination of C_1 and C_2 to $\mathcal{D}$.

Division: Given the normalized PB constraint $\sum_i w_i \ell_i \geq A$ in $\mathcal{C} \cup \mathcal{D}$ and a positive integer c, we can add the constraint $\sum_i \lceil w_i/c \rceil \ell_i \geq \lceil A/c \rceil$ to $\mathcal{D}$.

VeriPB has some additional rules, which are briefly discussed below. We refer to [7, 20] for more details on these rules. There is a rule for dealing with optimization statements:

Objective Improvement: Given an assignment α that satisfies $\mathcal{C}$, we can add the constraint $\mathcal{O} < \mathcal{O}{\restriction_\alpha}$ to $\mathcal{C}$.

This rule states that once a solution is found, we search for strictly better solutions. It is also possible to rewrite the objective:

Objective Reformulation Given a linear term $\mathcal{O}'$ in $\mathcal{O}$, $\mathcal{O}$ can be rewritten by replacing $\mathcal{O}'$ with $\mathcal{O}'_{\text{new}}$ if we have shown that $\mathcal{O}' \geq \mathcal{O}'_{\text{new}}$ and $\mathcal{O}' \leq \mathcal{O}'_{\text{new}}$.

VeriPB allows deriving non-implied constraints with a generalization of the RAT rule [26] (which is common in proof systems for SAT). This rule makes use of a *substitution*, which maps every variable to 0, 1, or a literal. Applying a substitution ω on a constraint C results in the constraint $C{\restriction_\omega}$ obtained from C by replacing each x by $\omega(x)$.

Redundance-based strengthening: If $\mathcal{C} \wedge \mathcal{D} \wedge \neg C \models \mathcal{O} \leq \mathcal{O}{\restriction_\omega} \wedge (\mathcal{C} \wedge C){\restriction_\omega}$, we add C to $\mathcal{C}$ if the strengthening-to-core mode is enabled and to $\mathcal{D}$ if it is disabled.

Intuitively, this rule can be used to show that ω, when applied to assignments instead of formulas, maps any solution of $\mathcal{C}$ that does not satisfy C to a solution of $\mathcal{C}$ that does satisfy C and that has an objective value that is at least as good. The most important use case of the redundance-rule is **reification**: for any PB constraint C and any fresh variable v, not used before, two applications of the redundance rule allow us to derive PB constraints that express $v \Rightarrow C$ and $v \Leftarrow C$. Finally, VeriPB has rules for deleting constraints in a way that guarantees that no better-than-optimal value can be found:

Deletion A constraint in $\mathcal{D}$ can be deleted at any time. For the deletion of a constraint $C \in \mathcal{C}$ it has to be shown that C can be derived by redundance-based strengthening from $\mathcal{C} \setminus \{C\}$.

3 Certification for MaxCDCL

In this section, we present the MaxCDCL algorithm. We start with the core version of the algorithm and afterwards discuss various extensions to it. With each algorithm/extension, we immediately discuss what needs to be done to integrate proof logging support for it.

3.1 The Core of MaxCDCL

MaxCDCL combines branch-and-bound search and conflict-driven clause learning (CDCL). It maintains the objective value v^* of the best found solution so far, which is initialized as $+\infty$. It performs standard CDCL search (branch). However, when a partial assignment is visited where it is clear that the value of the objective will be at least v^*, the search is interrupted and a new clause forcing the solver to backtrack is learned (bound).

VeriPB-based certification for CDCL-based SAT solvers is easy to obtain since VeriPB extends the DRAT proof system. Moreover, this has been done several times before [36, 19, 7], making the certification of the *branch* part straightforward. The *bound* part, which lies at the heart of MaxCDCL, on the other hand, has never been certified before. It consists of a clever way of detecting that it is indeed "clear" that all extensions of the current assignment that satisfy F have an objective value that is too high. To this end, in the general weighted case, given a current assignment α, it uses a lookahead mechanism that is designed to generate a set $\mathcal{K}$ of *weighted local cores*: tuples (w, K) where $w \in \mathbb{N}$ is the weight of the core, and K is a core relative to α: a set of negations of objective literals such that $F \wedge \alpha \wedge K \models \bot$, where $\bot$ denotes the trivially false constraint $0 \geq 1$. In other words a local core guarantees that, given the assignments in α, the underlying objective literals cannot all be false (or in other words: at least one of the underlying objective literals incurs cost). The set $\mathcal{K}$ should moreover satisfy the properties that

1. For each objective literal ℓ,

$$\sum_{(w,K)\in\mathcal{K}\wedge\bar{\ell}\in K} w \quad \leq w_{\mathcal{O}}(\ell),$$

   i.e., the total weight of all cores containing a literal does not exceed the weight of the underlying objective literal in the objective.

2. The total weight of $\mathcal{K}$ exceeds the current upper bound:

$$\sum_{(w,K)\in\mathcal{K}} w \geq v^*.$$

When these conditions are satisfied, since in each assignment that extends α at least one literal in each K is false, in each such assignment

$$v^* \leq \sum_{(w,K)\in\mathcal{K}} w \leq \sum_{(w,K)\in\mathcal{K}} w \cdot \left(\sum_{\bar{\ell}\in K} \ell\right) = \sum_{\ell} \left(\sum_{(w,K)\in\mathcal{K}\wedge\bar{\ell}\in K} w\right) \cdot \ell \leq \mathcal{O} \tag{1}$$

and we indeed see that the value of the objective in such an assignment can never (strictly) improve upon the currently best-found value. This situation is referred to as a *soft conflict*.

At this point, the solver can learn the clause $\neg\alpha$, stating that the current assignment is hopeless, but in practice, from the way the local cores are generated, we get more precise information. The way this is done is as follows. During the lookahead phase, the solver maintains a temporary objective $\mathcal{O}^t$ that is initialized to be equal to $\mathcal{O}$. The set of cores is initialized with trivially falsified unit cores of the form $(w(\ell), \{\bar{\ell}\})$, for all objective literals ℓ such that $\ell \in \alpha$. These represent the cost incurred by the current assignment α. Starting from the current assignment α, in the lookahead phase, all unassigned objective literals $\ell_1, \ell_2, \ldots$ are one at a time falsified (set to their non-cost-incurring polarity) followed by application of unit propagation. As soon as

- either an objective literal ℓ' is propagated to be cost-incurring,

- or a conflict is found

we can learn a new clause that tells us that under α, the literals ℓ_i that were falsified (and in the first case also the literal ℓ') cannot be false together. At this point, we can apply standard conflict analysis techniques to minimize this learned clause C_K corresponding to the local core K we will learn. C_K consists of *(1)* a set of negations of literals in α (denoted $\mathit{reason}(K)$) and *(2)* a set of objective literals, where K is the set of the negations of these literals. For the initial cores of the form $(w(\ell), \{\bar{\ell}\})$, we define $\mathit{reason}(K) = \{\bar{l}\}$. The literals in $\mathit{reason}(K)$ are called the reason for K since they tell us precisely which part of the current assignment was relevant for deriving the local core K. The weight w_K of cores K is chosen to be the $\min_{\bar{\ell} \in K} w_{\mathcal{O}^t}(\ell)$ and $\mathcal{O}^t$ is rewritten to $\mathcal{O}^t - w_K \sum_{\bar{\ell} \in K} \ell$. The lookahead then starts again from scratch, but this time only setting variables still in $\mathcal{O}^t$ to false (thus excluding objective literals whose weights have been completely "consumed" by the previous core). This procedure is guaranteed to indeed produce a set of local cores that satisfy condition 1 above. If condition 2 is not satisfied (i.e., the total weight of the produced cores does not exceed the upper bound), then MaxCDCL continues search as if nothing happened. Otherwise, it learns a clause $C_{\mathcal{K}} \subseteq \neg\alpha$, which is

$$\bigcup_{(w,K) \in \mathcal{K}} \mathit{reason}(K).$$

Indeed, the literals in $\neg C_{\mathcal{K}}$ are the literals in the current assignment that prevent the solver from finding a (strictly) improving solution. After applying standard conflict analysis techniques on the clause $C_{\mathcal{K}}$ for further simplification, it is learned by the solver, the solver backtracks and continues its search.

Hardening

If at some point, we end up in a state where for some literal ℓ currently still unassigned,

$$w_{\mathcal{O}^t}(\ell) + \sum_{(w,K) \in \mathcal{K}} w \geq v^*,$$

then a clause can be learned that propagates ℓ to be false. The reasoning for this is that in the assignment $\alpha \cup \ell$, we would have found a soft conflict. From the proof logging perspective, hardening does not require special treatment, so in what follows we focus on soft conflicts.

Proof-Logging for Lookahead-Based Bounding with Soft Conflicts

The VeriPB proof of lookeahead-based bounding we produce is heavily inspired by equation (1). Essentially, what happens is that we will reproduce this derivation inside the VeriPB proof, and as such we will derive that $\alpha \Rightarrow \mathcal{O} \geq v^*$. Combining this with the model-improving constraint then allows to derive a clause that excludes the current assignment α. All further minimizations that happen to this clause are done using standard conflict analysis (i.e., using resolution and hence, can be de done using explicit cutting planes derivations).

3.2 Literal Unlocking for Cardinality Reasoning

In the unweighted case, during the lookahead-based bounding a smarter mechanism is employed. In this case, the objective to minimize is of the form

$$\sum_{\ell \in objLits} \ell.$$

Given a previous upper bound v^*, in this case the lookahead procedure described above would yield a set of v^* disjoint cores. Instead of searching for a set of disjoint cores, MaxCDCL would in this situation search for a set of disjoint *cardinality constraints*: a set $\mathcal{L}$ of tuples (b, L) such that:

1. Each L is a set of objective literals s
2. For each (b, L) in $\mathcal{L}$, it holds that $F \wedge \alpha \models L \geq b$ (we remind the reader that we write $L \geq b$ for $\sum_{\ell \in L} \ell \geq b$).
3. For each pair (b, L) and (b', L') in $\mathcal{L}$, $L \cap L' = \emptyset$.
4. The total weight exceeds the current upper bound: $\sum_{(b,L)\in\mathcal{L}} b \geq v^*$.

If these four conditions are satisfied, we can see that indeed

$$F \wedge \alpha \models \sum_{\ell \in objLits} \ell \geq \sum_{(b,L)\in\mathcal{L}, \ell \in L} \ell \geq \sum_{(b,L)\in\mathcal{L}} b \geq v^*.$$

In other words, each assignment more precise than α will have an objective value that does not improve upon the best found so far and we can learn a clause that excludes the current assignment α.

The construction of such a set of disjoint cardinality constraints is achieved building on the following result.

▶ **Definition 1.** *Let (b, L) be a cardinality constraint, U a subset of L, and L' a set of objective literals disjoint from L. We say that L'* unlocks (b, L) on U *if $|U| \geq b$ and $F \wedge \alpha \wedge \bigwedge_{\ell \in L'} \overline{\ell} \models \ell'$ for each $\ell' \in U$.*

▶ **Proposition 2.** *Assume $\langle (b_i, L_i) \rangle|_{i\in[1,k]}$ is a sequence of cardinality constraints such that $F \wedge \alpha \models L_i \geq b_i$ for each i. Furthermore let L be a set of objective literals disjoint from all the L_i and let $U_i \subseteq L_i$ be such that $L \cup \bigcup_{j<i} L_j \setminus U_j$ unlocks U_i for each i. If additionally*

$$F \wedge \alpha \models L + \sum_i (L_i \setminus U_i) \geq 1,$$

then

$$F \wedge \alpha \models L + \sum_i L_i \geq \sum_i b_i + 1.$$

This proposition tells us that instead of searching for a new core that is disjoint from all the previously found cores (or more general, cardinality constraints), we can search for one that overlaps with some of the previous constraints, provided that all these constraints get unlocked. In practice this leads to a procedure whereby we falsify one by one all as-of-yet unassigned objective literals. As soon as unit propagation allows us to conclude that one of the previous cardinality constraints is unlocked, we are again allowed to assign the remaining literals from that cardinality constraint. If at some point, this yields a conflict, we can cut to the core of the conflict using standard conflict analysis techniques and learn a new cardinality constraint that increases the total bound by one. Let us illustrate this on a small example.

▶ **Example 3.** Assume $x_1, \ldots, x_{10}$ are objective literals (to be minimized). During the lookahead phase, the following happens:

- Assigning $x_1 = x_2 = x_3 = x_4 = 0$ results in a conflict. After analyzing this conflict the cardinality constraint (clause, in this case)

  $$x_1 + x_2 + x_4 \geq 1 \quad (2)$$

  is learned.

- Assigning $x_3 = x_5 = 0$ propagates $x_6 = 1$, meaning that the cardinality constraint
$$x_3 + x_5 + x_6 \geq 1 \tag{3}$$
is learned.
- Assigning $x_7 = x_8 = 0$ and running unit propagation yields $x_1 = 1$. At this point, the core (2) is unlocked. Further assigning $x_2 = x_4 = 0$ unit propagates to a conflict. As such we can replace core (2) by
$$x_1 + x_2 + x_4 + x_7 + x_8 \geq 2. \tag{4}$$
In combination with (3), this tells us that the value of the objective is at least three.
- Assigning $x_9 = 0$ unit propagates that $x_1 = x_3 = 1$. This is not enough to unlock (4), but (3) is unlocked at this point. Assigning $x_5 = x_6 = 0$ propagates that $x_7 = 1$, meaning that also (4) gets unlocked. Further assigning $x_2 = 0$ results in a conflict, meaning that we can replace (3) and (4) by
$$\sum_{1 \leq i \leq 9} x_i \geq 4.$$

Proof-Logging for Literal Unlocking

To prove correctness of these reasoning steps we will, as before make use of the fact that whenever unit propagation derives a new literal or contradiction, a clause can be learned that expresses precisely this propagation. The final clause to be learned can then be derived following the next proposition. In this proposition, we make abstraction of the current assignment α, but the proposition directly generalizes to that case by appending $\sum_{\ell \in \alpha} M\bar{\ell}$ to the left-hand side of every constraint (for a large enough constant M). The effect of this is that the constraint is made conditional on α.

▶ **Proposition 4.** *Let $L_i|_{1 \leq i \leq k}$ and L be pairwise disjoint sets of objective literals and $b_i|_{1 \leq i \leq k}$ natural numbers. Assume $U_i \subseteq L_i$ with $|U_i| = b_i$ for each i and write R_i for $L_i \setminus U_i$. From the constraints*
$$L_i \geq b_i \qquad \textit{for each } i \tag{5}$$
$$L + \sum_{j<i} R_j + \ell \geq 1 \qquad \textit{for each } i \textit{ and each } \ell \in U_i \tag{6}$$
$$L + \sum_{j} R_j \geq 1 \tag{7}$$
there is a cutting planes derivation that derives
$$L + \sum_{j>i} U_j + \sum_{j} R_j \geq 1 + \sum_{j>i} b_j \tag{8}$$
for each $i \in \{0, \ldots, k\}$. In particular, taking $i = 0$, there is a cutting planes derivation of
$$L + \sum_{j} L_j \geq 1 + \sum_{j} b_j. \tag{9}$$

Please note that (5) represents the previously derived cardinality constraints, (6) are the constraints that guarantee that $L \cup \bigcup_{j<i} R_j$ unlocks (b_i, L_i) on U_i and (7) is the constraint representing the final contradiction that is derived. The constraint (9), in case the sum of all the b_i is large enough, allows us indeed to conclude that there are no assignments that improve upon the best value found so far. While Proposition 4 only specifies the existence of a cutting planes derivation, the proof is constructive and provides all the details necessary to automate this construction.

4 Certified BDD-Based Encodings of PB Constraints

A Binary Decision Diagram (BDD) is a (node- and edge-)labeled graph with

- Two leaves, labeled true ($\mathbf{t}$) and false ($\mathbf{f}$), respectively
- Each internal node is labeled with a variable and has two outgoing edges, labeled true ($\mathbf{t}$) and false ($\mathbf{f}$), respectively.

Each node in a BDD represents a Boolean function. If η is a node labeled x with true child $\eta_{\mathbf{t}}$ and false child $\eta_{\mathbf{f}}$, it maps any (total) interpretation I to $n(I)$, which is defined as $\eta_{\mathbf{t}}(I)$ if $x \in I$ and as $\eta_{\mathbf{f}}(I)$ if $\overline{x} \in I$. The true and false leaf nodes represent a tautology and contradiction respectively.

A BDD is *ordered* if there is a total order of the variables such that each path through the BDD respects this order and it is *reduced* if two conditions hold: *(1)* no node has two identical children and *(2)* no two nodes have the same label, $\mathbf{t}$-child and $\mathbf{f}$-child. For a fixed variable ordering, each Boolean function has a unique ordered and reduced representation as a BDD, i.e., ordered and reduced BDDs form a canonical representation of Boolean functions. From now on, we will fix the standard variable ordering (ordering the x_i based on their index), but the ideas presented here work for an arbitrary ordering. When we write $\mathrm{bdd}(x, \eta_{\mathbf{t}}, \eta_{\mathbf{f}})$ we mean a node in the BDD labeled x with true child $\eta_{\mathbf{t}}$ and false child $\eta_{\mathbf{f}}$. If the BDD is reduced, this node is unique if it exists.

In MaxCDCL, BDDs are used, heuristically, to encode the solution-improving constraint $\mathcal{O} \leq v^*$ in SAT. This is done in two phases: first a BDD is constructed, then a set of clauses is generated from this BDD.

Phase 1: Construction of a BDD for $\mathcal{O} \leq v^*$

For simplicity of the presentation, we will assume here without loss of generality that $\mathcal{O}$ is of the form $\sum_{i=1}^{n} a_i x_i$ (with all the a_i positive and all the x_i variables). The standard way to create a reduced and ordered BDD for $\mathcal{O} \leq v^*$ is to create BDDs $\eta_{\mathbf{t}}$ for $\sum_{i>1}^{n} a_i x_i \leq v^* - a_1$ and $\eta_{\mathbf{f}}$ for $\sum_{i>1}^{n} a_i x_i \leq v^*$, while making sure the diagram remains reduced, and combine them into the node $\mathrm{bdd}(x_1, \eta_{\mathbf{t}}, \eta_{\mathbf{f}})$. However, this approach will always require an exponential number of calls; to avoid this in many cases (but not in the worst-case), a dynamic programming approach was developed. All the PB constraints for which we will create a BDD node are of the form

$$\sum_{i \geq k}^{n} a_i x_i \leq A. \tag{10}$$

The key observation is that in case $\sum_{i \geq k}^{n} a_i x_i$ cannot take the value A (because of the x_i being Boolean variables), then this is in fact equivalent to $\sum_{i \geq k}^{n} a_i x_i \leq A - 1$. In fact, for each such pseudo-Boolean constraint there is a (possibly unbounded) interval $[l, u]$ such that for all $b \in [l, u]$, (10) is equivalent to $\sum_{i \geq k}^{n} a_i x_i \leq b$, and moreover, this interval can be computed "bottom-up" from the BDD structure. We will denote this as

$$\sum_{i \geq k}^{n} a_i x_i \leq [l, u]. \tag{11}$$

The dynamic programming approach now consists in keeping track of this interval for each translated PB constraint and reusing already created BDDs whenever possible. In other words, memoization is used for two things: (1) for looking up whenever a new BDD needs to be created for a formula of the form (10) whether that combination of k and A is already captured by a previously created BDD and (2) for looking up whether a node with two specified children already exists. To illustrate this last point, note that the formulas

$$2x + 7y + 7z + 7u \leq [9, 13]$$

and

$$7y + 7z + 7u \leq [7, 13]$$

are equivalent and hence should have the same respresentation in a reduced BDD-based representation.

Phase 2: A CNF Encoding From the BDD

Given a BDD that represents a PB constraint, constructed as discussed above, we can get a CNF encoding as follows:

- For each internal node η in the BDD, a new variable v_η is created; intuitively this variable is true only when the Boolean function is true. In practice for the two leaf nodes no variable is created but their truth value is filled in directly. However, in the proofs below we will, to avoid case splitting pretend that a variable exists for each node.
- For each internal node $\eta = \mathrm{bdd}(x, \eta_{\mathbf{t}}, \eta_{\mathbf{f}})$, the clauses

  $$\overline{v}_{\eta_{\mathbf{t}}} \wedge x \Rightarrow \overline{v}_\eta \tag{12}$$

  and

  $$\overline{v}_{\eta_{\mathbf{f}}} \wedge \overline{x} \Rightarrow \overline{v}_\eta \tag{13}$$

  are added (encoding precisely the two cases in which the Boolean function in this node can be violated). If all the coefficients are positive, the second clause can be further simplified to $\overline{v}_{\eta_{\mathbf{f}}} \Rightarrow \overline{v}_\eta$.
- Finally, for the topf node $v_\top$ representing $\mathcal{O} \leq v^*$, the unit clause $v_\top$ is added.

Certifying the BDD-Based CNF Encoding

Our strategy for certifying this works as follows.

First we introduce the new variables using their pseudo-Boolean definitions and immediately prove that the lower and upper bound in the interval yield the same Boolean function. This can be done by structural induction on the BDD. Formally, for each node η, representing the PB constraints (11), we will show that we can derive

$$v_\eta \Rightarrow \sum_{i \geq k}^{n} a_i x_i \leq l, \qquad \text{which is} \quad \left(\sum_{i \geq k}^{n} a_i - l\right) \cdot \overline{v}_\eta + \sum_{i \geq k}^{n} -a_i x_i \geq -l \tag{14}$$

and

$$v_\eta \Leftarrow \sum_{i \geq k}^{n} a_i x_i \leq u, \qquad \text{which is} \quad (u+1) \cdot v_\eta + \sum_{i \geq k}^{n} a_i x_i \geq u + 1. \tag{15}$$

Showing this is the hard part of the derivation.

Secondly, we derive the desired clauses from the BDD. This can be done using a straightforward cutting planes derivation.

Certifying Reduced BDDs

Since we are working with *reduced* BDDs, it can happen that different constraints are represented by the same node because they are the same underlying Boolean function. In this case, what we need to show is that these Boolean functions are indeed equivalent. Specifically, the situation we can arrive at is that we will find a node that represents both the following constraints

$$\sum_{i=\beta}^{n} a_i x_i \leq [l_1, u] \qquad\qquad \sum_{i=\alpha}^{n} a_i x_i \leq [l_2, u]$$

with $\alpha < \beta$ and with $l_2 = l_1 + \sum_{i=\alpha}^{\beta-1} a_i$. What we show then is that there is a cutting planes derivation that from the constraints

$$v \Rightarrow \sum_{i=\beta}^{n} a_i x_i \leq l_1 \tag{16}$$

$$v \Leftarrow \sum_{i=\beta}^{n} a_i x_i \leq u \tag{17}$$

we can derive the constraints

$$v \Rightarrow \sum_{i=\alpha}^{n} a_i x_i \leq l_2 \tag{18}$$

$$v \Leftarrow \sum_{i=\alpha}^{n} a_i x_i \leq u \tag{19}$$

In other words, we obtain two instantiations of the constraints (14) and (15) for a single variable v, but for the different constraints. Since the rest of our proof logging procedures only rely on having derived these two constraints, this is sufficient.

Using Multi-Valued Decision Diagrams (MDD)

MaxCDCL not only makes use of BDDs for encoding the solution-improving constraint, but also of MDDs. The idea is as follows. In some cases, MaxCDCL can infer implicit at-most-one constraints. These are constraints of the form $\sum_{i \in I} x_i \leq 1$ where the x_i are literals in the objective $\mathcal{O}$. The detection of such constraints is common in MaxSAT solvers, and certification for it has been described in [2], so we will not repeat this here. Now assume that a set of disjoint at-most-one constraints has been found. In an MDD, instead of branching on *single variable* in each node, we will branch on a set of variables for which an at-most-one constraint has previous been derived. This means a node does not have two, but $|I| + 1$ children: one for each variable in the set and one for the case where none of them is true. Otherwise, the construction and ideas remain the same. As far as *certification* is concerned: essentially all proofs continue to hold; the main difference is that there where case splitting is used, we will now split into $|I| + 1$ cases instead of two (splitting on whether the variable decided on in that node is true or false) and then use the at most one constraint to derive that the conclusion must hold in any case.

5 Conclusions and Future Work

In this paper, we have for the first time presented certification for branch-and-bound MaxSAT solving. This work fits in an ongoing effort to show that VeriPB-based proof logging is feasible for different MaxSAT solving paradigms. Together with previous work, three out of the four major paradigms are now covered, the major omission being implicit hitting-set search. While we see no major theoretic obstacles, an important hurdle preventing proof logging for this last paradigm is the fact that state-of-the art solvers make use of commercial closed source MIP solvers for computing their hitting sets. Unless these solvers are equipped with proof logging capabilities, or if they are replaced by an open-source alternative, certification will remain out of reach.

Our quest to extend MaxCDCL with proof logging capabilities also resulted in an investigation of BDD-based CNF encodings of pseudo-Boolean constraints. This relates to the work [10], who has used BDDs for DRAT-based proof logging. In contrast, the focus of our work is not on developing general proof logging methods for arbitrary BDD operations, but specialized procedures for the constructions occurring in the constructions of BDDs (and more generally MDDs) for encoding PB constraints. For this pseudo-Boolean proof logging turned out to be very practical as it enabled us to reason about intervals of bounds.

In the near future, we plan to finish our implementation and experimentally evaluate the performance of our proof logging procedures.

References

1. Ignasi Abío, Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell, and Valentin Mayer-Eichberger. A new look at BDDs for pseudo-Boolean constraints. *J. Artif. Intell. Res.*, 45:443–480, 2012. URL: `https://doi.org/10.1613/jair.3653`, `doi:10.1613/JAIR.3653`.
2. Jeremias Berg, Bart Bogaerts, Jakob Nordström, Andy Oertel, and Dieter Vandesande. Certified core-guided MaxSAT solving. In Brigitte Pientka and Cesare Tinelli, editors, *Automated Deduction - CADE 29 - 29th International Conference on Automated Deduction, Rome, Italy, July 1-4, 2023, Proceedings*, volume 14132 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2023. `doi:10.1007/978-3-031-38499-8\_1`.
3. Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. `doi:10.1007/978-3-662-07964-5`.
4. Armin Biere. Tracecheck. `http://fmv.jku.at/tracecheck/`, 2006.
5. Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2021. `doi:10.3233/FAIA336`.
6. Miquel Bofill, Jordi Coll, Josep Suy, and Mateu Villaret. An mdd-based SAT encoding for pseudo-Boolean constraints with at-most-one relations. *Artif. Intell. Rev.*, 53(7):5157–5188, 2020. URL: `https://doi.org/10.1007/s10462-020-09817-6`, `doi:10.1007/S10462-020-09817-6`.
7. Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Certified dominance and symmetry breaking for combinatorial optimisation. *J. Artif. Intell. Res.*, 77:1539–1589, 2023. `doi:10.1613/jair.1.14296`.
8. Robert Brummayer and Armin Biere. Fuzzing and delta-debugging SMT solvers. In Ofer Strichman Bruno Dutertre, editor, *Proceedings of the 7th International Workshop on Satisfiability Modulo Theories*, SMT '09, page 1–5, New York, NY, USA, 2009. Association for Computing Machinery. `doi:10.1145/1670412.1670413`.
9. Robert Brummayer, Florian Lonsing, and Armin Biere. Automated testing and debugging of SAT and QBF solvers. In Ofer Strichman and Stefan Szeider, editors, *Theory and Applications of Satisfiability Testing - SAT 2010, 13th International Conference, SAT 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, volume 6175 of *Lecture Notes in Computer Science*, pages 44–57. Springer, 2010. `doi:10.1007/978-3-642-14186-7\_6`.

10 Randal E. Bryant. Tbuddy: A proof-generating BDD package. In Alberto Griggio and Neha Rungta, editors, *22nd Formal Methods in Computer-Aided Design, FMCAD 2022, Trento, Italy, October 17-21, 2022*, pages 49–58. IEEE, 2022. URL: `https://doi.org/10.34727/2022/isbn.978-3-85448-053-2_10`, `doi:10.34727/2022/ISBN.978-3-85448-053-2\_10`.

11 Sam Buss and Jakob Nordström. Proof complexity and SAT solving. In Biere et al. [5], pages 233–350. `doi:10.3233/FAIA200990`.

12 William J. Cook, Collette R. Coullard, and György Turán. On the complexity of cutting-plane proofs. *Discret. Appl. Math.*, 18(1):25–38, 1987. `doi:10.1016/0166-218X(87)90039-4`.

13 William J. Cook, Thorsten Koch, Daniel E. Steffy, and Kati Wolter. A hybrid branch-and-bound approach for exact rational mixed-integer programming. *Math. Program. Comput.*, 5(3):305–344, 2013. `doi:10.1007/s12532-013-0055-6`.

14 Luís Cruz-Filipe, Marijn J. H. Heule, Warren A. Hunt Jr., Matt Kaufmann, and Peter Schneider-Kamp. Efficient certified RAT verification. In Leonardo de Moura, editor, *Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings*, volume 10395 of *Lecture Notes in Computer Science*, pages 220–236. Springer, 2017. `doi:10.1007/978-3-319-63046-5\_14`.

15 Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In André Platzer and Geoff Sutcliffe, editors, *Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings*, volume 12699 of *Lecture Notes in Computer Science*, pages 625–635. Springer, 2021. `doi:10.1007/978-3-030-79876-5\_37`.

16 Nelson Guimarães Ferreira and Paulo Sérgio Muniz Silva. Automatic verification of safety rules for a subway control software. In Alexandre Mota and Arnaldo V. Moura, editors, *Proceedings of the Seventh Brazilian Symposium on Formal Methods, SBMF 2004, Recife, Pernambuco, Brazil, November 29 - December 1, 2004*, volume 130 of *Electronic Notes in Theoretical Computer Science*, pages 323–343. Elsevier, 2004. URL: `https://doi.org/10.1016/j.entcs.2005.03.017`, `doi:10.1016/J.ENTCS.2005.03.017`.

17 Mathias Fleury. *Formalization of logical calculi in Isabelle/HOL*. PhD thesis, Saarland University, Saarbrücken, Germany, 2020. URL: `https://tel.archives-ouvertes.fr/tel-02963301`.

18 Xavier Gillard, Pierre Schaus, and Yves Deville. Solvercheck: Declarative testing of constraints. In Thomas Schiex and Simon de Givry, editors, *Principles and Practice of Constraint Programming - 25th International Conference, CP 2019, Stamford, CT, USA, September 30 - October 4, 2019, Proceedings*, volume 11802 of *Lecture Notes in Computer Science*, pages 565–582. Springer, 2019. `doi:10.1007/978-3-030-30048-7\_33`.

19 Stephan Gocht, Ruben Martins, Jakob Nordström, and Andy Oertel. Certified CNF translations for pseudo-Boolean solving. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2-5, 2022, Haifa, Israel*, volume 236 of *LIPIcs*, pages 16:1–16:25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. `doi:10.4230/LIPIcs.SAT.2022.16`.

20 Stephan Gocht and Jakob Nordström. Certifying parity reasoning efficiently using pseudo-Boolean proofs. 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, February 2-9, 2021*, pages 3768–3777. AAAI Press, 2021. URL: `https://ojs.aaai.org/index.php/AAAI/article/view/16494`.

21 Evguenii I. Goldberg and Yakov Novikov. Verification of proofs of unsatisfiability for CNF formulas. In *2003 Design, Automation and Test in Europe Conference and Exposition (DATE 2003), 3-7 March 2003, Munich, Germany*, pages 10886–10891. IEEE Computer Society, 2003. URL: `https://doi.ieeecomputersociety.org/10.1109/DATE.2003.10008`, `doi:10.1109/DATE.2003.10008`.

22 Marijn Heule, Matti Järvisalo, Martin Suda, Markus Iser, and Tomáš Balyo. The 2022 international SAT competition. https://satcompetition.github.io/2022/, 2022.

23 Marijn Heule, Warren A. Hunt Jr., and Nathan Wetzler. Trimming while checking clausal proofs. In *Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013*, pages 181–188. IEEE, 2013. URL: https://ieeexplore.ieee.org/document/6679408/.

24 Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the Boolean Pythagorean triples problem via cube-and-conquer. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 228–245. Springer, 2016. doi:10.1007/978-3-319-40970-2_15.

25 Matti Järvisalo, Jeremias Berg, Ruben Martins, and Andreas Niskanen. MaxSAT evaluation 2023. https://maxsat-evaluations.github.io/2023/, 2023.

26 Matti Järvisalo, Marijn Heule, and Armin Biere. Inprocessing rules. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, volume 7364 of *Lecture Notes in Computer Science*, pages 355–370. Springer, 2012. doi:10.1007/978-3-642-31365-3_28.

27 Chu Min Li and Felip Manyà. MaxSAT, hard and soft constraints. In Biere et al. [5], pages 903–927. doi:10.3233/FAIA201007.

28 Chu-Min Li, Zhenxing Xu, Jordi Coll, Felip Manyà, Djamal Habet, and Kun He. Boosting branch-and-bound MaxSAT solvers with clause learning. *AI Commun.*, 35(2):131–151, 2022. doi:10.3233/AIC-210178.

29 David F. Manlove and Gregg O'Malley. Paired and altruistic kidney donation in the UK: algorithms and experimentation. *ACM J. Exp. Algorithmics*, 19(1), 2014. doi:10.1145/2670129.

30 Ross M. McConnell, Kurt Mehlhorn, Stefan Näher, and Pascal Schweitzer. Certifying algorithms. *Comput. Sci. Rev.*, 5(2):119–161, 2011. doi:10.1016/j.cosrev.2010.09.009.

31 Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002. doi:10.1007/3-540-45949-9.

32 Monica L. Nogueira, Marcello Balduccini, Michael Gelfond, Richard Watson, and Matthew Barry. An a-Prolog decision support system for the space shuttle. In I. V. Ramakrishnan, editor, *Practical Aspects of Declarative Languages, Third International Symposium, PADL 2001, Las Vegas, Nevada, USA, March 11-12, 2001, Proceedings*, volume 1990 of *Lecture Notes in Computer Science*, pages 169–183. Springer, 2001. doi:10.1007/3-540-45241-9_12.

33 João P. Marques Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, 48(5):506–521, 1999. doi:10.1109/12.769433.

34 Konrad Slind and Michael Norrish. A brief overview of HOL4. In Otmane Aït Mohamed, César A. Muñoz, and Sofiène Tahar, editors, *Theorem Proving in Higher Order Logics, 21st International Conference, TPHOLs 2008, Montreal, Canada, August 18-21, 2008. Proceedings*, volume 5170 of *Lecture Notes in Computer Science*, pages 28–32. Springer, 2008. doi:10.1007/978-3-540-71067-7_6.

35 Dieter Vandesande. Towards certified MaxSAT solving: Certified MaxSAT solving with SAT oracles and encodings of pseudo-Boolean constraints. Master's thesis, Vrije Universiteit Brussel (VUB), 2023. URL: https://researchportal.vub.be/nl/studentTheses/towards-certified-maxsat-solving.

36 Dieter Vandesande, Wolf De Wulf, and Bart Bogaerts. QMaxSATpb: A certified MaxSAT solver. In Georg Gottlob, Daniela Inclezan, and Marco Maratea, editors, *Logic Programming and Nonmonotonic Reasoning - 16th International Conference, LPNMR 2022, Genova, Italy, September 5-9, 2022, Proceedings*, volume 13416 of *Lecture Notes in Computer Science*, pages 429–442. Springer, 2022. doi:10.1007/978-3-031-15707-3_33.

37 Nathan Wetzler, Marijn Heule, and Warren A. Hunt Jr. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429. Springer, 2014. doi:10.1007/978-3-319-09284-3_31.