Beyond Core-Guided MaxSAT

Ilario Bonacina
Universitat Politècnica de Catalunya, Barcelona, Spain

Jordi Levy
IIIA, CSIC, Barcelona, Spain

Ion Mikel Liberal
IIIA, CSIC, Barcelona, Spain

Abstract

Several proof systems for MaxSAT have been proposed in the literature, including MaxSAT resolution and, more recently, systems based on polynomial calculus and tableaux. Although these systems are sound and complete and have varying strengths, they fail to capture the specific inferential strategies used by practical MaxSAT solvers, particularly those used in core-guided approaches. As a result, a formula that is hard to prove in these proof systems may not be hard for a solver, and vice versa.

In this paper, we describe a new proof system for MaxSAT, the *Comparator Calculus* (CC), which models the inferential strategies used in core-guided MaxSAT solvers. Inspired by this formalism, we introduce two new MaxSAT algorithms: a core-guided one (CSimple) and one non-core-guided (CSat), which uses heuristics to construct new soft formulas and calls a SAT solver on a single soft formula. We also define a hybrid mechanism (core-guided CSat) that uses cores to guide the heuristics.

We evaluate and compare our solvers with OLL on instances from the MaxSAT evaluation 2024, random 2-CNFs, and PHP formulas. Experimental results suggest that, in general, the performance of the different MaxSAT solvers depends on the structure of the instances. On the set of industrial instances of the MaxSAT Evaluation 2024, CSimple performs better than the others (including OLL).

2012 ACM Subject Classification Theory of computation → Proof complexity; Mathematics of computing → Solvers

Keywords and phrases MaxSAT, Proof Systems, Solvers, Optimization

Digital Object Identifier 10.4230/LIPIcs.SAT.2026.9

Supplementary Material *Software (Source Code)*: `https://github.com/Equipomaxsat/Beyond-Core-Guided-MaxSAT` [15]; archived at `swh:1:dir:a682af082413755925b3b91d4df82ca14d10dd84`

Funding *Ilario Bonacina*: Funded by the AEI with the grant number PID2022-138506NB-C22.
Jordi Levy: Funded by the AEI with the grant number PID2022-138506NB-C21.
Ion Mikel Liberal: Funded by the AEI with the grant number PID2022-138506NB-C21.

1 Introduction

MaxSAT is the optimization version of the SAT problem. Unlike SAT, which asks whether a CNF formula is satisfiable or not, in MaxSAT, we are interested in finding the maximum number of clauses that are satisfiable.

Roughly speaking, there are three families of MaxSAT solvers in the literature: *branch-and-bound* [30, 32], *core-guided* [21, 3, 34, 33, 36, 5, 35], and *implicit hitting set* [20, 39] MaxSAT solvers. Whereas branch-and-bound MaxSAT solvers represent the state-of-the-art for random formulas, core-guided and implicit hitting set solvers are preferred for industrial instances. Recently, Ihalainen et al. [26] showed how core-guided and implicit hitting set solvers can be seen as particular instances of a more general scheme. In both cases, these solvers call a SAT solver (sometimes with additional assumptions) to check the satisfiability of a CNF formula or to obtain unsatisfiable subsets of clauses (cores) to guide the search

toward an optimal solution. Recent advances have focused on certifying the correctness of core-guided MaxSAT solvers, ensuring their reliability in practical applications [11] and the integrity of preprocessing steps [27].

Proof systems for MaxSAT include MaxSAT resolution [16, 17] and versions of the polynomial calculus [13, 14] and tableaux calculi [31, 8]. However, they all fail to model the inferential strategies practical MaxSAT solvers use. In this work, we show how a simple proof system, the *Comparator Calculus* (CC), can model the type of calls core-guided solvers make to a SAT solver. A relevant feature of the Comparator Calculus is that it can also model uses of SAT solvers that differ significantly from core extraction. We illustrate this by presenting an algorithm (CSat) for MaxSAT, whose behavior can be modeled by the Comparator Calculus. Despite relying on SAT solver calls, it does not follow the core-guided approach.

The Comparator Calculus consists essentially of two substitution rules: the *Comparator* (COMP) rule and the *Contradiction* (CONTR) rule. Without going into technical details, the COMP rule replaces two formulas by their conjunction and disjunction, while the contradiction rule substitutes an unsatisfiable formula with an immediate contradiction. In this case, to ensure the rule is polynomially checkable, the witness of the unsatisfiability is given by a refutation in some propositional proof system. This rule aims to capture the use of SAT solvers for solving MaxSAT. The COMP rule is intended to capture the nature of the networks that are usually used to encode the cardinality constraints that come up in runs of core-guided solvers [38, 19, 9]. Indeed, CC polynomially simulates the behavior of core-guided MaxSAT solvers such as Fu&Malik and OLL when the cardinality constraints are encoded using sorting networks (see Section 3.2).

Core-guided MaxSAT solvers were first introduced by Fu and Malik in [21], where the authors proposed a solver for partial MaxSAT that can be informally described as follows. The algorithm Fu&Malik calls a SAT solver with all the clauses (hard and soft) and takes advantage of the set of unsatisfiable clauses returned by the SAT solver by *weakening* the clauses in the core with new variables and imposing the condition that only one of those new variables must be falsified (cardinality constraint). Then, it repeats the same procedure until it gets a satisfiable formula. This algorithm is guided by the unsatisfiable sets of clauses returned by the SAT solvers. After this, several modifications and refinements of Fu&Malik's scheme were proposed and, roughly speaking, the main differences lie in how these different approaches encode and use cardinality constraints [3, 33, 20, 36]. Nowadays, the OLL algorithm [33, 25] is a good representative of the state-of-the-art MaxSAT solvers.

In this work, we present some algorithms for MaxSAT inspired by the CC calculus. The first is *CSimple*. Unlike OLL, which orders[1] all the clauses in an unsatisfiable core, CSimple constructs a simpler network using comparators that just compute the conjunction of all the clauses in the core.

The second algorithm is CSat, which uses calls to a SAT solver but cannot be classified as core-guided. CSat works with a partial MaxSAT instance with unary soft clauses. Initially, it obtains several satisfying assignments (models) of the hard part of the problem. Then, instead of searching for cores in the soft part, it tries to *infer* them. Taking advantage of the models, it constructs binary comparator circuits between soft clauses. When a soft clause is found that is falsified in all current models (a *candidate*), the SAT solver is called to certify its unsatisfiability. Depending on the result, the candidate is either removed or the model

[1] The order does not refer to the clauses but to the output of the network. We mean that in OLL we compute all soft cardinality constraints (using totalizers or sorting networks). But clearly, the order of the soft clauses influences the cardinality constraints.

returned by the SAT solver is added to the set of models. The algorithm terminates when a model satisfying all remaining soft clauses is obtained. The third type of algorithms we consider is a core-guided version of CSat. They use some core-guided reasoning to reduce the search space for the heuristic.

We compare the performance of the introduced algorithms (CSimple, CSat with two different heuristics, and the core-guided version of both heuristics) with OLL [33] on instances (weighted and unweighted) from the MaxSAT Evaluation 2024 [12], random 2-CNF formulas, and the Pigeonhole principle formulas (see Section 5). In general, we observe that the performance of the different MaxSAT solvers depends on the structure of the instances. On the industrial instances, CSimple performs better than all others. We compare the algorithms on the time spent in SAT solver calls.

On the theoretical side, the definition of a relatively weak proof system that simulates the behavior of practical MaxSAT solvers opens up the possibility of applying tools from proof complexity. For instance, to derive concrete performance limits for specific classes of formulas or encodings, such as cardinality constraints or random CNF instances.

Structure of the article. In Section 2 we recall all the necessary preliminaries for this work. Section 3 contains the definition of the Comparator Calculus CC, its soundness and completeness, how the calculus specializes to partial MaxSAT with hard and soft clauses, its connection with core-guided MaxSAT solvers, and a brief discussion of the strengths and limitations of CC. Section 4 contains the high-level description and the pseudocode of CSimple and CSat. Section 5 contains the experimental evaluation of CSimple and the variations of CSat, comparing them with OLL. Finally, in Section 6, we make some concluding remarks.

2 Preliminaries

This section contains standard notions and notations used throughout the paper. Let X be a set of Boolean variables. A *literal* ℓ is a variable x from X or its negation $\overline{x}$. Propositional formulas are constructed recursively from literals using conjunctions ($\wedge$) and disjunctions ($\vee$). In particular, a *clause* is a disjunction of literals, and a formula in CNF is a conjunction of clauses. We denote the empty clause with $\bot$.

A (total) *assignment* is a mapping $\alpha : X \to \{0, 1\}$. We extend assignments to propositional formulas in the usual way: setting $\alpha(\overline{x}) = 1 - \alpha(x)$, $\alpha(A \vee B) = \max\{\alpha(A), \alpha(B)\}$, and $\alpha(A \wedge B) = \min\{\alpha(A), \alpha(B)\}$, together with all the properties of classical logic ($1 \vee C = 1$, $0 \vee C = C$, etc). An assignment α *satisfies* a multiset of formulas Σ (denoted $\alpha \models \Sigma$) if, for every formula $F \in \Sigma$, $\alpha(F) = 1$. The multiset Σ is *satisfiable* if there exists an assignment α such that $\alpha \models \Sigma$. We say that α is a *model* of Σ. Otherwise, Σ is *unsatisfiable*. Any subset of Σ which is unsatisfiable is a *core*. Given a multiset of formulas Σ and an assignment α, the *cost* of Σ under α is

$$\mathrm{cost}_\alpha(\Sigma) = \sum_{F \in \Sigma} (1 - \alpha(F)) \ ,$$

i.e. $\mathrm{cost}_\alpha(\Sigma)$ is the number of formulas in Σ falsified by α. We consider (partial) MaxSAT instances of the form $\mathcal{F} = \mathcal{H} \cup \mathcal{S}$ where $\mathcal{H}$ is a set of *hard* formulas and $\mathcal{S}$ is a multiset of soft formulas. The *cost* of $\mathcal{F}$ is

$$\mathrm{cost}(\mathcal{F}) = \mathrm{cost}_{\mathcal{H}}(\mathcal{S}) = \min_{\alpha \colon \alpha \models \mathcal{H}} \mathrm{cost}_\alpha(\mathcal{S}),$$

i.e., $\text{cost}(\mathcal{F})$ is the minimum number of falsified formulas in $\mathcal{S}$ by any assignment satisfying all the hard formulas in $\mathcal{H}$. W.l.o.g. we can assume the soft formulas to be literals $\overline{b}_1, \ldots, \overline{b}_m$: indeed, any soft formula $F_i \in \mathcal{S}$ can be equivalently represented by a hard formula $F_i \vee b_i$ and a soft literal $\overline{b}_i$ (where b_i is a fresh new variable). This is the *blocking literals encoding* and it is how MaxSAT instances are usually encoded in practice.

Notice that, in SAT solving, formulas are assumed to be sets or conjunctions of clauses, i.e., in CNF. Similarly, in MaxSAT solving, formulas are assumed to be multisets of clauses. Here, we deal with arbitrary formulas, and MaxSAT problems are multisets of arbitrary formulas. Moreover, we also deal with sets $\mathcal{A}$ of assignments.

3 The Comparator Calculus

In this section, we introduce the *Comparator Calculus* (CC) for MaxSAT. For simplicity, we only consider unweighted formulas, although the calculus can be easily extended to the weighted case by adding *fold* and *unfold* rules to it (see [17] and [14] for a detailed description of these rules). First, we describe the calculus on multisets of soft formulas and then we show the minor adaptations for the calculus to deal with hard and soft formulas.

The *Comparator Calculus* manipulates multisets of propositional formulas with two substitution rules: the *comparator* (COMP) rule and *contradiction* (CONTR) rule. That is, the following inference rules are applied by replacing the premises with the conclusions:

$$\frac{A \quad B}{A \wedge B \quad A \vee B} \quad (\text{COMP}) \qquad \frac{A}{\bot} \ \{A \text{ unsatisfiable}\} \quad (\text{CONTR}) \ . \tag{1}$$

Notice that, in the CONTR rule, there is a *side condition* that requires the unsatisfiability of the premise A in order to apply the rule. To certify applications of this rule, we provide a refutation of the premise A (or a suitable encoding of A) in some propositional proof system P. For instance, we could consider as P the propositional proof system Frege (see, for instance [29]), or we could provide the proof of unsatisfiability of a suitable encoding $\text{enc}(A)$ produced by a SAT solver. Since most SAT solvers we could use to certify the unsatisfiability of A only handle CNF formulas, an encoding is necessary in this case.

Notice that both the COMP rule and the CONTR rule preserve the cost: for every assignment α, we have

$$\begin{aligned} &\text{cost}_\alpha(\{A, B\}) = \text{cost}_\alpha(\{A \wedge B, A \vee B\}) \text{ and} \\ &\text{cost}_\alpha(A') = \text{cost}_\alpha(\bot) \end{aligned} \tag{2}$$

when A' is unsatisfiable.

The *comparator* (COMP) rule is probably the simplest sound rule we can get in a cost-preserving calculus for MaxSAT,[2] but it has the disadvantage that, even if the premises are clauses, the conclusions have higher logical depth; in particular, they are not clauses. This forces us to work with arbitrary propositional formulas. The name of the rule (*comparator*) is borrowed from the terminology of the circuits used to build sorting networks. These circuits have two inputs and two outputs. They sort the values on the inputs and can be

[2] Indeed, this rule and its soundness for MaxSAT derivations were already mentioned in [14] in the context of polynomial calculus for MaxSAT.

implemented using AND and OR gates. Sorting networks are also a standard method to encode cardinality constraints, used commonly in core-guided MaxSAT solvers.[3] Before the formal definition of the Comparator Calculus, we give a simple example of derivation.

▶ **Example 3.1.** Given the multiset of formulas $\{x_1 \wedge x_2,\ \overline{x}_1 \wedge x_3,\ \overline{x}_2 \wedge \overline{x}_3\}$, we can perform the following substitutions using the COMP and CONTR rules:

$$\dfrac{\dfrac{\dfrac{\dfrac{x_1 \wedge x_2,\quad \overline{x}_1 \wedge x_3,\quad \overline{x}_2 \wedge \overline{x}_3}{x_1 \wedge x_2 \wedge \overline{x}_1 \wedge x_3,\quad (x_1 \wedge x_2) \vee (\overline{x}_1 \wedge x_3),\quad \overline{x}_2 \wedge \overline{x}_3}\ \text{COMP}}{\bot,\quad (x_1 \wedge x_2) \vee (\overline{x}_1 \wedge x_3),\quad \overline{x}_2 \wedge \overline{x}_3}\ \text{CONTR}}{\bot,\quad ((x_1 \wedge x_2) \vee (\overline{x}_1 \wedge x_3)) \wedge \overline{x}_2 \wedge \overline{x}_3,\quad (x_1 \wedge x_2) \vee (\overline{x}_1 \wedge x_3) \vee (\overline{x}_2 \wedge \overline{x}_3)}\ \text{COMP}}{\bot,\quad \bot,\quad (x_1 \wedge x_2) \vee (\overline{x}_1 \wedge x_3) \vee (\overline{x}_2 \wedge \overline{x}_3)}\ \text{CONTR}$$

In the first application of CONTR, a side proof is, for instance, a Resolution refutation of $x_1 \wedge x_2 \wedge \overline{x}_1 \wedge x_3$, i.e. a refutation of the set of clauses $\{x_1, x_2, \overline{x}_1, x_3\}$, and in the second it is a refutation of $((x_1 \wedge x_2) \vee (\overline{x}_1 \wedge x_3)) \wedge \overline{x}_2 \wedge \overline{x}_3$, i.e. a refutation of the set of clauses $\{x_1 \vee \overline{x}_1, x_1 \vee x_3, x_2 \vee \overline{x}_1, x_2 \vee x_3, \overline{x}_2, \overline{x}_3\}$. In this simple example, we apply distributivity to transform formulas to CNF. In general, this process may lead to an exponential blow-up in the size of formulas. Therefore, we will use more efficient encodings (for instance Tseitin encoding) at the expense of introducing fresh variables.

Formally, derivations in the Comparator Calculus in the context of partial MaxSAT are defined as follows.

3.1 The Comparator Calculus on Partial MaxSAT

We consider partial MaxSAT instances of the form $\mathcal{F} = \mathcal{H} \cup \mathcal{S}$, where $\mathcal{H}$ is a set of hard formulas and $\mathcal{S}$ is a multiset of soft formulas and w.l.o.g. we assume $\mathcal{S}$ only contains literals (or $\bot$s) and we denote hard clauses using a subscript ∞, i.e. to denote that a clause C is hard we write it as C_∞. In adapting the rules of CC to this context, the COMP rule becomes

$$\dfrac{\ell_1 \quad \ell_2}{y_1 \quad y_2 \quad \mathsf{CNF}(y_1 \leftrightarrow \ell_1 \wedge \ell_2,\ y_2 \leftrightarrow \ell_1 \vee \ell_2)_\infty}\ (\text{COMP}')$$

where ℓ_1, ℓ_2 are soft literals, y_1, y_2 are fresh new variables (also soft literals) and

$$\mathsf{CNF}(y_1 \leftrightarrow \ell_1 \wedge \ell_2,\ y_2 \leftrightarrow \ell_1 \vee \ell_2)_\infty$$

denotes the natural CNF encoding of $y_1 \leftrightarrow \ell_1 \wedge \ell_2$ and $y_2 \leftrightarrow \ell_1 \vee \ell_2$ as hard clauses.

▶ **Definition 3.2** (CC in partial MaxSAT). *A* CC *derivation from $\mathcal{F} = \mathcal{H} \cup \mathcal{S}$ is a sequence of pairs $\pi = \langle (\mathcal{H}_1; \mathcal{S}_1), \ldots, (\mathcal{H}_s; \mathcal{S}_s) \rangle$ where $\mathcal{H}_1 = \mathcal{H}$, $\mathcal{S}_1 = \mathcal{S}$ and for each i one of the following two cases occurs:*

1. *If $\mathcal{S}_i = \mathcal{S}'_i \cup \{\ell_1, \ell_2\}$ then introduce two fresh new variables y_1, y_2 and let*

$$\mathcal{H}_{i+1} = \mathcal{H}_i \cup \{\mathsf{CNF}(y_1 \leftrightarrow \ell_1 \wedge \ell_2,\ y_2 \leftrightarrow \ell_1 \vee \ell_2)\}$$

and

$$\mathcal{S}_{i+1} = \mathcal{S}'_i \cup \{y_1, y_2\}\ .$$

[3] Totalizers are reported to work better than Batcher's odd-even sorting networks in OLL MaxSAT solvers [35, 28]. However, in this paper, we focus on the Boolean formulas used as soft restrictions, trying to make an abstraction of their encoding.

2. *If* $\mathcal{S}_i = \mathcal{S}'_i \cup \{\ell\}$ *and* $\mathcal{H}_i \cup \{\ell\}$ *is unsatisfiable (which is certified by a refutation in a proof system* P*), then*

$$\mathcal{H}_{i+1} = \mathcal{H}_i \quad \textit{and} \quad \mathcal{S}_{i+1} = \{\bot\} \cup \mathcal{S}'_i \ .$$

The option in Item 1 corresponds to the COMP *rule, while the option in Item 2 corresponds to an application of the* CONTR *rule. The* size *of a proof is the total number of bits needed to write the derivation, including the refutations in* P *appearing in Item 2. Notice that we count towards the size also the hard formulas. For a lighter notation, we usually omit the proof system* P *in the notation for* CC*.*

The calculus we just described is sound and complete for MaxSAT in the following sense.

▶ **Theorem 3.3** (Soundness). *For every multiset of formulas* $\mathcal{F} = \mathcal{H} \cup \mathcal{S}$ *and* CC *derivation* $\pi = \langle(\mathcal{H}_1;\mathcal{S}_1),\ldots,(\mathcal{H}_s;\mathcal{S}_s)\rangle$ *with* $\mathcal{H}_1 = \mathcal{H}$*,* $\mathcal{S}_1 = \mathcal{S}$ *and* $\{\bot,\overset{k}{\ldots},\bot\} \subseteq \mathcal{S}_s$*, it holds that* $\mathrm{cost}(\mathcal{F}) = \mathrm{cost}_{\mathcal{H}}(\mathcal{S}) \geq k$*.*

▶ **Theorem 3.4** (Completeness). *For every multiset of formulas* $\mathcal{F} = \mathcal{H} \cup \mathcal{S}$ *with* $\mathrm{cost}(\mathcal{F}) = \mathrm{cost}_{\mathcal{H}}(\mathcal{S}) = k$ *there exists a* CC *derivation* $\pi = \langle(\mathcal{H}_1;\mathcal{S}_1),\ldots,(\mathcal{H}_s;\mathcal{S}_s)\rangle$ *with* $\mathcal{H}_1 = \mathcal{H}$*,* $\mathcal{S}_1 = \mathcal{S}$ *and* $\{\bot,\overset{k}{\ldots},\bot\} \subseteq \mathcal{S}_s$*.*

For the sake of clarity, we described the CC calculus on partial MaxSAT, but it is straightforward to extend the CC calculus on *weighted* partial MaxSAT instances. Indeed, the algorithms inspired by the CC calculus are all working with weighted clauses.

3.2 Connection with core-guided MaxSAT Solvers

We show how CC simulates the OLL, the Fu&Malik algorithm, and a version of Fu&Malik with symmetry breaking that better adapts to CC.

The Proof System Behind the OLL Algorithm. Given a MaxSAT instance $\mathcal{F}$, the OLL algorithm replaces the clauses in a core $\mathcal{C}$ by *soft* cardinality constraints. If we focus on the unweighted case and use formulas instead of clauses, this is equivalent to applying the rule:

$$\frac{A_1 \quad \cdots \quad A_r}{\bot \quad \mathrm{enc}(\sum_{i=1}^{r} A_i \geq r-1) \ \cdots \ \mathrm{enc}(\sum_{i=1}^{r} A_i \geq 1)} \tag{3}$$

where $A_1 \wedge \cdots \wedge A_r$ is unsatisfiable. Its unsatisfiability is certified in a propositional proof system P, and $\mathrm{enc}(\sum_{i=1}^{r} A_i \geq j)$ is a propositional formula equivalent to the cardinality constraint $\sum_{i=1}^{r} A_i \geq j$. Notice that if $A_1 \wedge \cdots \wedge A_r$ is unsatisfiable then there is no assignment α satisfying $\mathrm{enc}(\sum_{i=1}^{r} \alpha(A_i) \geq r)$. The OLL rule in eq. (3) is cost-preserving. To emphasize the underlying propositional proof system P, we use the notation OLL_P *calculus* to denote the calculus that uses the rule above.

Among the various methods to encode cardinality constraints, using sorting networks is one of the most commonly used in current solvers.

The main idea behind this method is as follows. Let $y_1,\ldots,y_r = \mathsf{SN}(x_1,\ldots,x_r)$ be a sorting network with inputs $x_1,\ldots,x_r$ and outputs $y_1,\ldots,y_r$, the assignment α satisfies the constraint $x_1 + \cdots + x_r \geq i$ if, and only if, the output y_{r-i+1} in $\mathsf{SN}(\alpha(x_1),\ldots,\alpha(x_r))$ is one.

The Partial MaxSAT version of the OLL rule, using sorting networks, is:

$$\frac{x_1 \quad \cdots \quad x_r}{\bot \quad y_2 \cdots y_r \quad \mathsf{CNF}(y_1,\ldots,y_r = \mathsf{SN}(x_1,\ldots,x_r))_\infty} \tag{4}$$

under the condition that $\mathcal{H} \cup \{x_1,\ldots,x_r\}$ is unsatisfiable.

This calculus is the one modeling the basic OLL algorithm, which uses calls to a SAT solver to find the core. To do this, the formulas describing the sorting network in eq. (4) need to be CNFs (for example via a Tseitin encoding [42]), and are added as hard clauses.

Sorting networks with n inputs can be defined using $\mathcal{O}(n \log n)$ comparators and with depth $\mathcal{O}(\log n)$ [1], where a comparator is a circuit that takes input x, y and has output $x \wedge y, x \vee y$, like our COMP rule. Defining these networks with comparator circuits fits very well in the context of CC.

▶ **Theorem 3.5.** CC_P *polynomially simulates the* OLL_P *calculus (with sorting networks to encode soft cardinality constraints).*

Proof. One application of the OLL rule (in eq. (3)) can be simulated by $\mathcal{O}(n \log n)$ applications of the COMP rule using the construction from [1], plus one application of CONTR rule to replace $A_1 + \cdots + A_r \geq r$ – which is $A_1 \wedge \cdots \wedge A_r$ in any sorting network – by $\bot$. We use the same proof system P to certify the correctness of the CONTR rule and the correctness of the OLL rule. In both cases, we have to certify in P the unsatisfiability of $A_1 \wedge \cdots \wedge A_r$. ◀

The Proof System Behind the Fu&Malik Algorithm. In the Fu&Malik algorithm, every time a core $\{A_1, \ldots, A_r\}$ is found (by calling a SAT solver on the soft clauses), all the soft clauses are relaxed adding a fresh variable x_i and a hard constraint $x_1 + \cdots + x_r \leq 1$. In addition, an empty clause is added to ensure that only one of the original soft clauses is *repaired*. Therefore, in terms of formulas, the derivation can be modeled as:

$$\frac{A_1 \cdots A_r}{\bot \quad A_1 \vee x_1 \;\cdots\; A_r \vee x_r \quad \mathrm{enc}\left(\sum_{i=1}^{r} x_i \leq 1\right)_\infty} \tag{5}$$

under the condition that $A_1 \wedge \cdots \wedge A_r$ is unsatisfiable.

As for the OLL calculus we call the calculus using the rule above the *Fu&Malik calculus* and this calculus clearly models the Fu&Malik algorithm.

In this case, the introduction of hard constraints avoids the possibility of defining a calculus where all formulas are soft. Moreover, this encoding introduces a lot of *symmetries*. If an assignment falsifies several A_i's, we can decide to set any of their x_i's to true, getting several optimal assignments.

In the experimental comparison, we resort to an optimized version of the original Fu&Malik algorithm with symmetry breaking, where, intuitively, we *repair* the first unsatisfied formula. This is inspired by the symmetry-breaking approach in [2].

In other words, we set x_i to true only if all previous formulas $A_1, \ldots, A_{i-1}$ are already satisfied. Therefore, the soft clause $A_i \vee x_i$ is equivalent to the formula $A_i \vee (A_1 \wedge \cdots \wedge A_{i-1})$. With this restriction, the derivation results in:

$$\frac{A_1 \cdots A_r}{\bot \quad A_2 \vee A_1 \quad A_3 \vee (A_1 \wedge A_2) \;\cdots\; A_r \vee (A_1 \wedge \cdots \wedge A_{r-1})} \tag{6}$$

where $\{A_1 \wedge \cdots \wedge A_r$ is unsatisfiable. We call the calculus using the rule above *Fu&Malik calculus with symmetry breaking*. As usual, we use a subscript P to denote the propositional proof system used to certify unsatisfiability.

Notice that with this restriction on which soft clause we repair, we no longer need the cardinality restriction ensuring that only one is repaired. The first soft formula $A_1 \vee x_1$ is always repaired. Therefore, it is a tautology and can be removed from the conclusions.

▶ **Theorem 3.6.** CC_P *linearly simulates the* Fu&Malik$_P$ *calculus with symmetry breaking.*

Proof. One application of the FU&MALIK SYM. BREAK rule can be simulated by $r-1$ applications of the COMP rule plus one application of CONTR to derive $\bot$ from $A_1 \wedge \cdots \wedge A_r$. Like in the OLL case, we assume we are using the same proof system in both calculi to certify the unsatisfiability of $A_1 \wedge \cdots \wedge A_r$. ◀

3.3 Strength, Limitations and Extensions of the Comparator Calculus

The main factor at play in measuring the strength of CC is the underlying propositional proof system used to certify the validity of applications of the CONTR rule. If this proof system P is very strong, say for instance P is Frege, then CC_P seems to become very strong as well. In practice, the correctness of the CONTR rule is certified by executions of SAT solvers, and proof search in state-of-the-art SAT solvers is captured by proof systems such as Resolution or Cutting Planes (or Frege on formulas of bounded logical depth). It is reasonable then to consider in CC_P the proof system P to be one of the systems above. Then, certifying the validity of a CONTR rule, i.e., the unsatisfiability of a formula F, needs to be done by refuting *some encoding* of F in P (in the case of Resolution, a CNF encoding, for instance). The role played by different encodings of the same formula in determining the hardness of refuting it is not yet well understood in proof complexity.

If we consider the proof system bounded-depth Frege (bdFrege) and all the formulas appearing in $\mathsf{CC}_{\text{bdFrege}}$ calculus are restricted to have bounded logical depth, then $\mathsf{CC}_{\text{bdFrege}}$ requires exponential size to certify the unsatisfiability of Tseitin formulas and the Pigeonhole Principle. This is a simple consequence of $\mathsf{CC}_{\text{bdFrege}}$ being no stronger than bdFrege and the existing exponential size lower bounds for the formulas above in bdFrege [24, 23].

Regarding the limitations of CC, note that the system only considers calls to the proof system P (or a SAT solver) to certify the validity of applications of the CONTR rule. There might be a lot of repeated work in this process. Consider, for example, the scenario when $\mathcal{F} = \{A, B\}$ where both A and B are unsatisfiable, i.e. $\text{cost}(\mathcal{F}) = 2$, and both A and B are hard to refute in P, but $A \to B$ is easy to prove in P. (The most extreme case is when $A = B$.) In computing the size of CC we pay both for a P-refutation of A and B, while it might have been more efficient to prove in P that B is unsatisfiable and that $A \to B$.

Under these considerations, a possible way to strengthen CC could be by incorporating ideas from OLL into the calculus. For instance, extending the OLL inference rule to the rule where more than one $\bot$ is concluded, i.e.

$$\frac{A_1 \cdots A_r}{\overset{k}{\bot \cdots \bot} \quad B_{k+1} \cdots B_r} \quad \left\{ \begin{array}{l} B_k \text{ is unsatisfiable} \\ \text{For } i = k, \ldots, r,\ B_i \text{ is equivalent to } A_1 + \cdots + A_r \geq i \end{array} \right\}$$

where B_i is a formula built from conjunctions and disjunctions of the A_i's following the structure of a sorting network. A particular instance of this rule, for $r = k = 2$, is

$$\frac{A_1 \quad A_2}{\bot \quad \bot} \quad \{A_1 \vee A_2 \text{ is unsatisfiable}\} .$$

Notice that if the size of the refutation of $A_1 \vee A_2$ is smaller than the sum of the refutations of A_1 and A_2, then CC cannot efficiently simulate this inference.

A practical implementation of this system could use a heuristic to select the multiset $\{A_1, \ldots, A_r\}$ – possibly the complete set of (soft) formulas – , then using a sorting network structure to derive the formulas $B_1, \ldots, B_r$, and then a binary search with calls to the SAT solver – at most $\log r$ calls – to find the value k.

Building a sorting network on top of all soft clauses and performing a dichotomic search with calls to a SAT solver is arguably the simplest MaxSAT algorithm one can devise. However, it has been shown to perform poorly in practice. A refinement – restricting the dichotomic search to a subset of the soft clauses, such as a core – has been incorporated in some core-guided MaxSAT solvers (see for instance [4, 6]).

4 CC Inspired Algorithms for MaxSAT

This section presents two types of algorithms for solving (weighted) Partial MaxSAT inspired by CC: CSimple and CSat. Both take as input a partial MaxSAT instance encoded with blocking variables, that is, a set $\mathcal{H}$ of hard clauses and a multiset $\mathcal{S}$ of soft unary clauses. In the implementation, $\mathcal{S}$ is a set of pairs (weight, literal).

For clarity, we only cover the unweighted case, but a minor adaptation of the description would work for weighted partial MaxSAT as well (and the implementations work for the weighted case as well). In case of acceptance, the source code of the implementations of the algorithms will be made publicly available.

4.1 CSimple

CSimple is a core-guided MaxSAT solver. After obtaining an unsatisfiable core, CSimple computes the conjunction of all the soft formulas in the core (which is unsatisfiable, but not necessarily minimal), applying the comparator rule. The conjunction of all formulas is removed, while the remaining formulas continue being soft but not necessarily ordered (as in OLL). The soundness of CC implies the soundness of CSimple, therefore, the computation of the optimal cost is also ensured. Compared with OLL, whereas OLL needs to introduce $\mathcal{O}(n \log^2 n)$ comparators to deal with a core of size n (using the Batcher's sorting network), CSimple only needs $n - 1$ comparators. The construction of the conjunction is done with a comparator circuit of minimal depth $\log n$.

4.2 CSat and core-guided CSat

Both CSat and core-guided MaxSAT solvers (see, for instance, [3, 34]) call SAT solvers as functions, but there are crucial differences. The main difference is that core-guided MaxSAT solvers use the SAT solver to get unsatisfiable cores to infer an empty clause (or equivalently to increase the lower bound for the cost). In contrast, CSat tries to indirectly obtain a *candidate* for a core using some heuristics and calls a SAT solver only to certify the correctness of the guessed candidate core. This candidate is the conjunction of some original soft clauses, obtained by the application of the COMP rule, and represented as a unique soft formula. Moreover, CSat keeps track of a *set of assignments* $\mathcal{A}$ satisfying the hard clauses $\mathcal{H}$, instead of just one (partial) assignment as in common SAT solvers. As a consequence, instead of invoking the SAT solver on the full set of soft clauses, as typical core-guided solvers do, CSat evaluates the satisfiability of $\mathcal{H}$ clauses and just *one* soft formula. This soft formula, or candidate, is falsified in all models in $\mathcal{A}$. With this approach, we try to make executions of the SAT solver more efficient.

For the description of CSat we refer to the pseudocode in Algorithm 1. The parts in orange in the pseudocode refer to the core-guided variant of CSat. We will discuss them later, and for the moment, discuss Algorithm 1 without the parts in green.

Initialization. At the beginning of the execution, CSat makes k calls to the SAT solver over the hard clauses $\mathcal{H}$, to obtain some models to initialize $\mathcal{A}$. (If no model of $\mathcal{H}$ exists, the algorithm terminates). It also initializes the *remaining cost upper bound rub* as the minimum cost among all the assignments obtained so far. It also has a counter, or *cost lower bound lb*, of all empty clauses generated so far.

Main Loop. After this initialization part, CSat enters the main loop. While the remaining upper bound is greater than zero, i.e., no model of all remaining soft clauses has been found, the algorithm checks if there is some unary soft clause c falsified by all assignments in $\mathcal{A}$. This is a *candidate* to encode an unsatisfiable formula. If such a clause exists, it calls the SAT solver with the hard clauses $\mathcal{H}$ and the candidate c. If the solver returns SAT, it will also give a model, so CSat incorporates the new model into $\mathcal{A}$ and updates the remaining upper bound accordingly. Otherwise, it applies the CONTR rule: it removes the soft clause from the current set of soft clauses, increases the lower bound, and decreases the remaining upper bound. Notice that this has to be done because we do not introduce $\bot$ into $\mathcal{S}$.

If no soft clause evaluates to false over all assignments, CSat invokes some heuristic that selects two soft clauses b_1 and b_2. CSat constructs a comparator between them, that is, applying the COMP rule, we replace b_1 and b_2 by $b_1 \wedge b_2$ and $b_1 \vee b_2$. Since $b_1 \wedge b_2$ is not a clause, CSat introduces new hard clauses corresponding to $\mathsf{CNF}(x \leftrightarrow b_1 \wedge b_2)$ and $\mathsf{CNF}(y \leftrightarrow b_1 \vee b_2)$, and replaces b_1 and b_2 by x and y in the soft clauses. Notice that we have to ensure that the heuristic leads to obtaining some candidate (some clause evaluated to false by all assignments) and, eventually, the termination of the algorithm. This is proved in Theorems 4.3.

Finally, when a model of all remaining clauses in $\mathcal{S}$ is obtained, i.e., when $rub = 0$, the algorithm terminates and outputs the number of obtained empty clauses lb and the optimal assignment $\alpha \in \mathcal{A}$ satisfying all remaining soft clauses.

▶ Remark 4.1. There is a more visual way of representing the execution of CSat, which is in fact closer to the way it is actually implemented. The main data structure of the CSat implementation is a Boolean matrix $M = (M_{b,\alpha})$, where rows are indexed by the current soft unit clauses $b \in \mathcal{S}$, columns by assignments $\alpha \in \mathcal{A}$, and $M_{b,\alpha} = \alpha(b)$. The remaining upper bound rub is the minimal number of zeros found in the columns of M. Therefore, the algorithm terminates when a column of ones is obtained. When the CONTR rule is applied, one of the rows is removed (and the lb counter is incremented by 1). When the COMP rule is applied, two rows in the matrix are replaced by their conjunction and disjunction, respectively. The heuristic selects two rows of the matrix, performing some operations on them. Therefore, the data structure used in its implementation has to be very efficient in performing bit-wise operations on rows, like pair-wise conjunction and disjunction of two rows, and counting the number of zeros in a row.

▶ **Theorem 4.2** (Correctness). *The algorithm CSat on input $\mathcal{F} = \mathcal{H} \cup \mathcal{S}$, if it terminates, returns* $\mathrm{cost}_{\mathcal{H}}(\mathcal{S})$.

In general, it is not true that CSat always terminates. It depends on the heuristic function we use to select the two soft clauses to apply COMP. Suppose, for instance, that `heuristic` were selecting always the last two soft clauses. Since the application of COMP to the formulas $A \wedge B$ and $A \vee B$ results in $(A \wedge B) \wedge (A \vee B) = A \wedge B$ and $(A \wedge B) \vee (A \vee B) = A \vee B$, we enter an infinite loop.

Algorithm 1 The CSat algorithm.

Input: A set of hard clauses $\mathcal{H}$ and a multiset of unary soft clauses $\mathcal{S}$
Output: $\text{cost}_{\mathcal{H}}(\mathcal{S})$ and an optimal assignment α s.t. $\alpha \models \mathcal{H}$ and $\text{cost}_{\alpha}(\mathcal{S}) = \text{cost}_{\mathcal{H}}(\mathcal{S})$

```
A ← {α | sat, α ← SAT(H)}                    ▷ Set of assignments s.t. ∀α ∈ A. α ⊨ H
lb = 0                                        ▷ Lower bound for cost_H(S)
rub = min_{α∈A} {cost_α(S)}                   ▷ Remaining upper bound for cost_H(S)
C ← SAT(H ∪ S)
while rub > 0 do
    if C = {c} or ∃c ∈ S ∀α ∈ A, α(c) = 0 then   ▷ Exists a candidate c of empty clause
        sat, α ← SAT(H ∪ {c})                 ▷ if C = {c} this call is skipped and jump to the else
        if |C| > 1 or sat then                ▷ Introduce new model in A
            A ← A ∪ {α}
            rub ← min {rub, cost_α(S)}
        else                                  ▷ Apply CONTR rule
            lb ← lb + 1
            rub ← rub − 1
            S ← S \ {c}
            sat, α, C ← SAT(H ∪ S)
            if sat then
                A ← A ∪ {α}
                rub ← min {rub, cost_α(S)}
            end if
        end if
    else                                      ▷ Apply COMP rule
        b1, b2 ← heuristic(H, S, A, C)
        H ← H ∪ CNF(x ↔ b1 ∧ b2, y ↔ b1 ∨ b2)   ▷ x and y are fresh variables
        S ← S \ {b1, b2} ∪ {x, y}
        C ← C \ {b1, b2} ∪ {x}
    end if
end while
return lb and α ∈ A s.t. cost_α(S) = 0
```

Heuristics. We describe two possible heuristics for the implementation of CSat. Both heuristics ensure termination of CSat. Given a set of assignments $\mathcal{A}$ and a formula F, let $\text{count}_{\mathcal{A}}(F)$ be the number of assignments in $\mathcal{A}$ falsifying F.

To describe the first heuristic `heuristic1` (Algorithm 2) we use the language of Remark 4.1, i.e. a matrix $M = (M_{b,\alpha})$. Recall that the rows range over the soft literals $\mathcal{S}$ and the columns over the set of assignments $\mathcal{A}$ and $M_{b,\alpha} = \alpha(b)$.

The function `heuristic1` first finds a set of rows B_1 with the maximum number of 0s. Then, it finds a set of pairs B_2 of the form (x, y) where $x \in B_1$ and $y \in \mathcal{S}$, $y \neq x$ and maximizing the number of 0s of the row $x \wedge y$ if it were added to M. If there is more than one y, then it breaks ties by taking $B_3 \subseteq B_2$ as the set of pairs of literals (x, y) that maximize the number of 0s in the row $x \vee y$ if it were added to M.

The second heuristic `heuristic2` (Algorithm 3), intuitively, tries to optimize (maximize) the minimum number of bits we can assure the comparator contributes to order.

▶ **Theorem 4.3.** *The algorithm CSat with the* `heuristic1` *in Algorithm 2 or the* `heuristic2` *in Algorithm 3 both terminate in* $\mathcal{O}(|\mathcal{S}|^2)$ *iterations.*

Algorithm 2 The `heuristic1` function.

Input: $\mathcal{S}$, $\mathcal{A}$, $\mathcal{C}$
Output: $x, y \in \mathcal{S}$ (or $\mathcal{C}$)
$B_1 = \arg\max_{x \in \mathcal{S}} \text{count}_{\mathcal{A}}(x)$
$B_2 = \arg\max_{x \in B_1, y \in \mathcal{S} \setminus \{x\}} \text{count}_{\mathcal{A}}(x \wedge y)$
$B_3 = \arg\max_{(x,y) \in B_2} \text{count}_{\mathcal{A}}(x \vee y)$
return any $(x, y) \in B_3$

Algorithm 3 The `heuristic2` function.

Input: $\mathcal{S}$, $\mathcal{A}$, $\mathcal{C}$
Output: $x, y \in \mathcal{S}$ (or $\mathcal{C}$)
$B = \arg\max_{(x,y) \in \mathcal{S}} \min\{\text{count}_{\mathcal{A}}(x \vee \overline{y}), \text{count}_{\mathcal{A}}(\overline{x} \vee y)\}$
return any $(x, y) \in B$

Notice that Theorem 4.3 only allows us to bound the number of COMP iterations as $\mathcal{O}(|\mathcal{S}|^2)$. However, it is known that there exist sorting networks with $\mathcal{O}(|\mathcal{S}| \log |\mathcal{S}|)$ comparators. Therefore, although both heuristics ensure termination, they are not optimal in terms of the number of comparisons.

Core-guided CSat. The difference between the core-guided variation of CSat and CSat is marked in orange in Algorithm 1. The main difference is that the core-guided CSat maintains a core and uses it to guide the heuristic. Instead of considering all the soft clauses, the heuristic only takes into account the clauses in the core. Doing this we restrict considerably (depending on the quality of the core, of course) the number of pairs of soft clauses to be considered in the application of the heuristics. Notice also that, in the core-guided versions, we only introduce conjunctions in the core, the heuristic terminates in $|\mathcal{C}|$ steps.

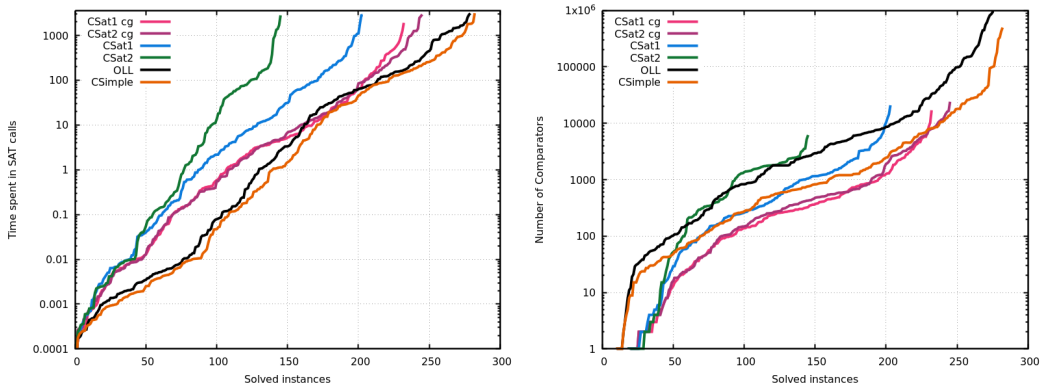
5 Experimental Results

We have conducted some experiments with implementations of the weighted version of the OLL, CSimple, and the CSat algorithms (with both heuristics and with/without the core-guided option, denoted CSat1, CSat2, CSat1 cg, and CSat2 cg, respectively). We also implemented Fu&Malik and Fu&Malik with symmetry breaking, but we don't show results because their performance is much worse than OLL.

We compare the algorithms on three families of benchmarks: the benchmarks from the MaxSAT Evaluation 2024 [12] (weighted and unweighted), the pigeonhole principle (with many more pigeons than holes), and random 2-CNFs (with a sufficiently large density of clauses). The goal of the last two experiments is to test any difference in behavior when all the unsatisfiable cores are expensive to produce (the case of PHP), and the case when the first cores are fast to produce while the last ones are not (random 2CNFs).

All the algorithms are implemented in Python, using the Optilog [7] library, and calling the same SAT solver, Glucose 4.1 [10]. We use the SAT solver incrementally, i.e., it is not restarted between different calls, so that the learned clauses are not removed. However, our MaxSAT solvers are not incremental in the sense of [41, 37]. All experiments have been executed in the cluster Ars Magna, composed of 11 nodes with two Intel Xeon CPUs at 2.2

GHz with 10 cores per CPU (20 cores per node). The timeout was $3600s$, and the memory was limited to $8GB$ per process. We only consider the time consumed in the calls to the SAT solver.



(a) (Non-cumulative) Time spent in SAT solver calls. **(b)** Number of comparators.

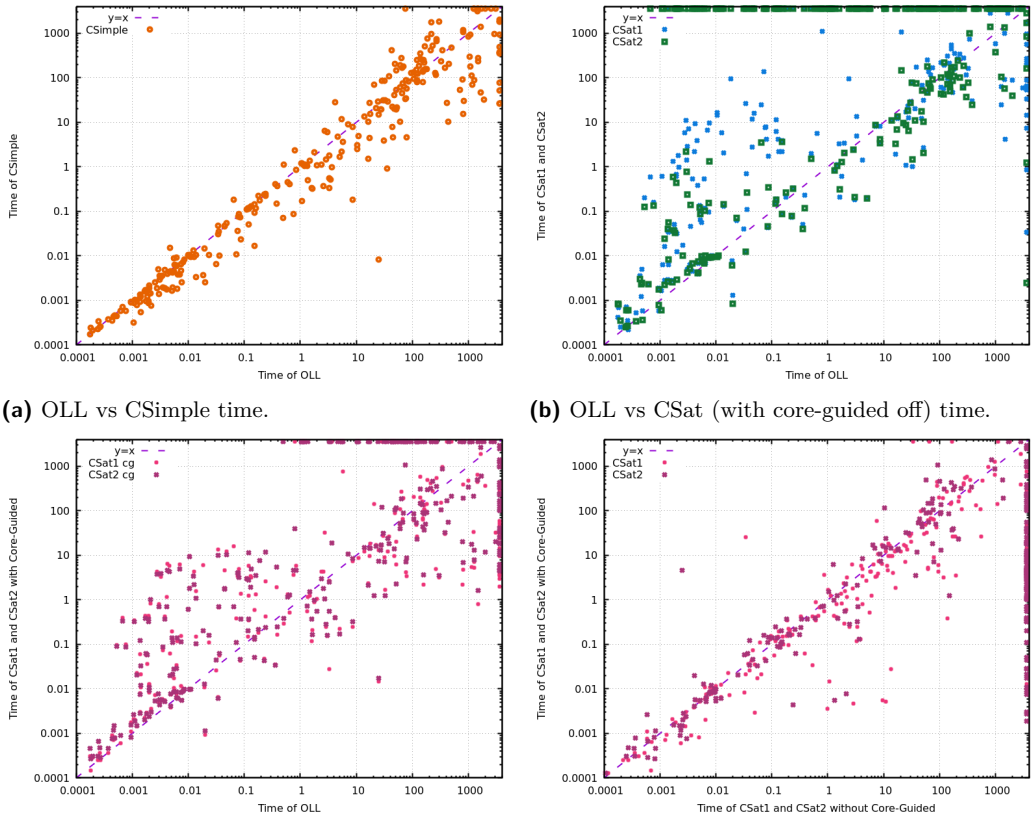
Figure 1 Cactus plot comparing the MaxSAT algorithms on the weighted category of the MaxSAT Evaluation 2024.

Experiment 1: Industrial Instances. We evaluate OLL, CSimple and both heuristics of CSat (and the core-guided option on and off) on instances of the MaxSAT Evaluation 2024 [12] (see Figure 1 and Figure 2). We show the results for the weighted instances. We also compare the algorithms w.r.t. the number of comparators (see Figure 1b). Every time we add a comparator, we have to add 6 hard clauses to the formula, so this number indicates how the formula grows during the execution.

We observe that CSimple solves more instances than all the other algorithms, OLL included. However, although core-guided CSat uses fewer comparators, this does not make the search more efficient. If cores are not minimal, CSat might increase the cost using fewer soft clauses and, in some cases, win. Otherwise, it might need a lot of SAT-solver calls answering SAT before obtaining UNSAT. We also observe that the core-guided option improves significantly CSat for both heuristics and makes them perform similarly. When the core-guided option is off, heuristic 2 is better than heuristic 1 (w.r.t. time and number of comparators).

In Figure 2, we pairwise compare the proposed algorithms with OLL. We observe a lot of variability in the behavior, in particular, there are certain instances where CSat performs considerably better than OLL, and others where it is the other way around. This phenomenon suggests that the structure of the instances matters when choosing a solver. We also observe that a direct comparison of CSat and core-guided CSat shows that on almost all instances the core-guided version performs better. OLL correlates quite clearly with CSimple (although CSimple performs better).

Experiment 2: Pigeonhole Principle. The *pigeonhole principle* PHP_n^m states that m pigeons fly to n pigeonholes without overlapping. This is encoded saying that each pigeon flies to at least one pigeonhole, and that each pigeonhole can have at most one pigeon occupying it. We consider the at-least restrictions as soft constraints and the at-most as hard. Therefore, the optimal cost is $m - n$. In the experimental evaluation we consider PHP_n^m with $m = 25$ and $n = 5$ (see Figure 3a), since such formulas are exponentially hard for resolution, and require large proofs even for the first cores.



(a) OLL vs CSimple time.

(b) OLL vs CSat (with core-guided off) time.

(c) OLL vs CSat (with core-guided on) time.

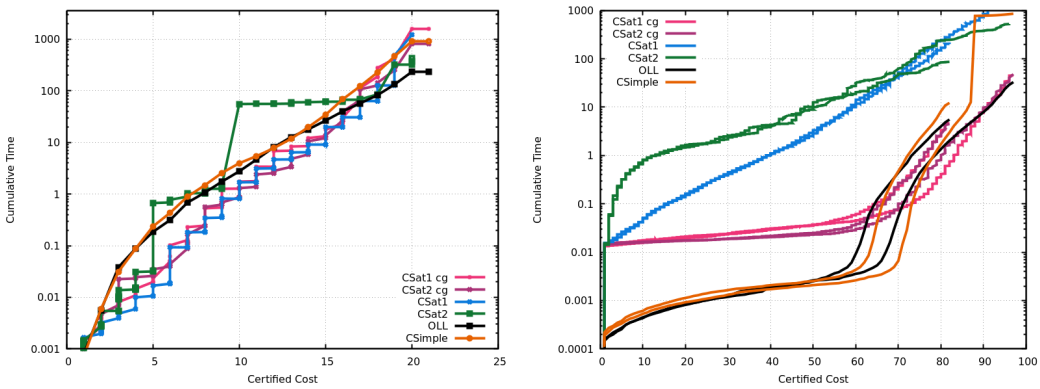
(d) CSat (with core-guided off) vs CSat (with core-guided on) time.

Figure 2 Direct comparison of OLL with the other algorithms for MaxSAT on the *weighted* industrial instances from the MaxSAT Evaluation 2024. Comparison of CSat and core-guided CSat on the same instances.

We observe that the OLL is the solver that spends the least time in the SAT solver calls. However, we can also see that almost all the other solvers are progressing faster than OLL until some advanced point in the certification of the optimal cost. It is also notable that in these formulas, the core-guided option in CSat does not improve the time in comparison with the non-core-guided version. Besides these observations, the solvers perform quite similarly aside from CSat2 (heuristic 2 and core-guided off). The time of this solver shows a ladder progress where incrementing the certificated cost is very expensive at some points, but very inexpensive in the rest.

Experiment 3: Random 2-CNFs. *Random 2-CNF* formulas are constructed by independently selecting m clauses, where each clause is constructed first choosing a set of 2 variables uniformly at random among the n available, and then tossing an unbiased coin to determine the polarity of the variables. A random 2-CNF is unsatisfiable with high probability when it has $m = n(1+\epsilon)$ clauses for every $\epsilon > 0$ and where n is the number of variables [18, 22]. We give all clauses in the random 2-CNF as soft constraints.

In the experimental evaluation, we consider random 2-CNFs with $n = 23$ and $m = 529$, and with $n = 24$ and $m = 576$ (see Figure 3b). Random 2-CNFs have the property that the extraction of the first cores is fast: they are just proofs of unsatisfiability of 2SAT problems. Whereas, for $m \gg n$, we expect the certification of the optimal cost to require super-polynomial time due to Max2SAT hardness.



(a) PHP_n^m with $n = 5$ and $m = 25$.

(b) Random 2-CNF formulas with $n = 23, 24$ and $m = 529, 576$.

Figure 3 Evolution of cumulative time needed to certify a partial cost in PHP_n^m formulas, and random 2-CNF formulas.

In the experimental results, we observe that CSat with the core-guided approach is better w.r.t. the time spent in SAT solver calls than the non-core-guided version for both heuristics. Moreover, core-guided CSat performs better than CSimple and similarly to OLL. From the experiments, we also observe that for all core-guided algorithms, including the core-guided version of CSat, the time needed to obtain the first cores is small, but at a certain abrupt point, the time consumed by the SAT solver grows notoriously. However, this phase-transition phenomenon is not observed for the non-core-guided CSat regardless of the heuristic.

6 Conclusions and Further Work

This paper introduces the Comparator Calculus (CC), a sound and complete proof system for MaxSAT designed to reflect the inferential behavior of core-guided MaxSAT solvers and built from two simple cost-preserving substitution rules. CC simulates key solving strategies used by algorithms such as Fu&Malik and OLL. In contrast to prior approaches rooted in resolution or algebraic methods, CC offers a lightweight and practically motivated model that can serve both as a tool for theoretical analysis and as inspiration for new solver designs.

Inspired by CC, we proposed a new core-guided MaxSAT solver CSimple and a new SAT-based MaxSAT solver (CSat) that departs from the core-guided paradigm. Rather than relying on unsatisfiable core extraction, CSat incrementally constructs candidate formulas and tests their consistency. This approach helps to mitigate the known limitations of core-guided solvers near the optimality certification. We also consider a core-guided variant of CSat, trying to join the two approaches.

Experimental results suggest that, in general, the performance of the different MaxSAT solvers depends on the structure of the instances [40]. On the set of industrial instances of the MaxSAT Evaluation 2024, CSimple performs better than the others (including OLL). Regarding CSat versions, we have observed that regardless of the heuristic, the core-guided option improves considerably its performance both on industrial instances and random 2-CNFs formulas.

Future work includes exploring proof complexity lower bounds within the calculus and improving the efficiency of CSat heuristics. It also remains open to study how different CNF encodings of formulas in the CONTR rule affect derivation size and tractability.

References

1 Miklós Ajtai, János Komlós, and Endre Szemerédi. An $O(n \log n)$ sorting network. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1–9, 1983. doi:10.1145/800061.808726.

2 Carlos Ansótegui, Maria Luisa Bonet, Joel Gabàs, and Jordi Levy. Improving WPM2 for (weighted) partial MaxSAT. In *19th International Conference on Principles and Practice of Constraint Programming, CP 2013*, volume 8124 of *Lecture Notes in Computer Science*, pages 117–132. Springer, 2013. doi:10.1007/978-3-642-40627-0_12.

3 Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. SAT-based MaxSAT algorithms. *Artif. Intell.*, 196:77–105, 2013. doi:10.1016/J.ARTINT.2013.01.002.

4 Carlos Ansótegui, Frédéric Didier, and Joel Gabàs. Exploiting the structure of unsatisfiable cores in MaxSAT. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015*, pages 283–289. AAAI Press, 2015. URL: http://ijcai.org/Abstract/15/046.

5 Carlos Ansótegui and Joel Gabàs. WPM3: an (in)complete algorithm for weighted partial MaxSAT. *Artif. Intell.*, 250:37–57, 2017. doi:10.1016/J.ARTINT.2017.05.003.

6 Carlos Ansótegui, Joel Gabàs, and Jordi Levy. Exploiting subproblem optimization in SAT-based MaxSAT algorithms. *J. Heuristics*, 22(1):1–53, 2016. doi:10.1007/S10732-015-9300-7.

7 Carlos Ansótegui, Jesus Ojeda, António Pacheco, Josep Pon, Josep M. Salvia, and Eduard Torres. Optilog: A framework for SAT-based systems. In *Proceedings of the 24th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 1–10, 2021. doi:10.1007/978-3-030-80223-3_1.

8 Josep Argelich, Chu Min Li, Felip Manyà, and Joan Ramon Soler. Clause tableaux for maximum and minimum satisfiability. *Log. J. IGPL*, 29(1):7–27, 2021. doi:10.1093/JIGPAL/JZZ025.

9 Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality networks: a theoretical and empirical study. *Constraints An Int. J.*, 16(2):195–221, 2011. doi:10.1007/S10601-010-9105-0.

10 Gilles Audemard and Laurent Simon. On the Glucose SAT solver. *Int. J. Artif. Intell. Tools*, 27(1):1840001:1–1840001:25, 2018. doi:10.1142/S0218213018400018.

11 Jeremias Berg, Bart Bogaerts, Jakob Nordström, Andy Oertel, and Dieter Vandesande. Certified core-guided MaxSAT solving. In *Proceedings of the 29th International Conference on Automated Deduction (CADE)*, pages 1–22, 2023. doi:10.1007/978-3-031-38499-8_1.

12 Jeremias Berg, Matti Järvisalo, Ruben Martins, Andreas Niskanen, and Tobias Paxian, editors. *MaxSAT Evaluation 2024: Solver and Benchmark Descriptions*. Department of Computer Science Series of Publications B. Department of Computer Science, University of Helsinki, Finland, 2024.

13 Ilario Bonacina, Maria Luisa Bonet, and Jordi Levy. Polynomial calculus for MaxSAT. In *26th International Conference on Theory and Applications of Satisfiability Testing, SAT 2023*, volume 271 of *LIPIcs*, pages 5:1–5:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.SAT.2023.5.

14 Ilario Bonacina, Maria Luisa Bonet, and Jordi Levy. Polynomial calculus for optimization. *Artif. Intell.*, 337:104208, 2024. doi:10.1016/J.ARTINT.2024.104208.

15 Ilario Bonacina, Jordi Levy, and Ion Mikel Liberal. Beyond Core-Guided MaxSAT. Software, Funded by the AEI with the grant number PID2022-138506NB-C21 and PID2022-138506NB-C22., swhId: swh:1:dir:a682af082413755925b3b91d4df82ca14d10dd84 (visited on 2026-06-30). URL: https://github.com/Equipomaxsat/Beyond-Core-Guided-MaxSAT.git, doi:10.4230/artifacts.26940.

16 Maria Luisa Bonet, Jordi Levy, and Felip Manyà. A complete calculus for Max-SAT. In *9th International Conference on Theory and Applications of Satisfiability Testing - SAT 2006*, volume 4121 of *Lecture Notes in Computer Science*, pages 240–251. Springer, 2006. doi:10.1007/11814948_24.

17 Maria Luisa Bonet, Jordi Levy, and Felip Manyà. Resolution for Max-SAT. *Artif. Intell.*, 171(8-9):606–618, 2007. doi:10.1016/J.ARTINT.2007.03.001.

18 Vasek Chvátal and Bruce A. Reed. Mick gets some (the odds are on his side). In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 620–627, 1992. doi:10.1109/SFCS.1992.267789.

19 Michael Codish and Moshe Zazon-Ivry. Pairwise cardinality networks. In *Proceedings of the 16th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, pages 154–172, 2010. doi:10.1007/978-3-642-17511-4_10.

20 Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 225–239, 2011. doi:10.1007/978-3-642-23786-7_19.

21 Zhaohui Fu and Sharad Malik. On solving the partial MAX-SAT problem. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 252–265, 2006. doi:10.1007/11814948_25.

22 Andreas Goerdt. A threshold for unsatisfiability. *J. Comput. Syst. Sci.*, 53(3):469–486, 1996. doi:10.1006/JCSS.1996.0081.

23 Johan Håstad. On small-depth Frege proofs for PHP. In *Proceedings of 2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 37–49, 2023. doi:10.1109/FOCS57990.2023.00010.

24 Johan Håstad. On small-depth Frege proofs for tseitin for grids. *J. ACM*, 68(1), 2020. doi:10.1145/3425606.

25 Alexey Ignatiev, António Morgado, and João Marques-Silva. RC2: an efficient MaxSAT solver. *J. Satisf. Boolean Model. Comput.*, 11(1):53–64, 2019. doi:10.3233/SAT190116.

26 Hannes Ihalainen, Jeremias Berg, and Matti Järvisalo. Unifying core-guided and implicit hitting set based optimization. In *Proceedings of the 32nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1935–1943, 2023. doi:10.24963/IJCAI.2023/215.

27 Hannes Ihalainen, Andy Oertel, Yong Kiam Tan, Jeremias Berg, Matti Järvisalo, Magnus O. Myreen, and Jakob Nordström. Certified MaxSAT preprocessing. In *Proceedings of the 12th International Joint Conference on Automated Reasoning (IJCAR)*, pages 396–418, 2024. doi:10.1007/978-3-031-63498-7_24.

28 George Katsirelos. Core-guided linear programming-based maximum satisfiability. In *28th International Conference on Theory and Applications of Satisfiability Testing, SAT 2025, Glasgow, Scotland, August 12-15, 2025*, volume 341 of *LIPIcs*, pages 17:1–17:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.SAT.2025.17.

29 Jan Krajíček. *Proof Complexity*. Cambridge University Press, 2019.

30 Chu Min Li, Felip Manyà, Nouredine Ould Mohamedou, and Jordi Planes. Resolution-based lower bounds in MaxSAT. *Constraints An Int. J.*, 15(4):456–484, 2010. doi:10.1007/S10601-010-9097-9.

31 Chu Min Li, Felip Manyà, and Joan Ramon Soler. A clause tableau calculus for MaxSAT. In Subbarao Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016*, pages 766–772. IJCAI/AAAI Press, 2016. URL: http://www.ijcai.org/Abstract/16/114.

32 Chu-Min Li, Zhenxing Xu, Jordi Coll, Felip Manyà, Djamal Habet, and Kun He. Combining clause learning and branch and bound for MaxSAT. In *27th International Conference on Principles and Practice of Constraint Programming, CP 2021*, volume 210 of *LIPIcs*, pages 38:1–38:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CP.2021.38.

33 António Morgado, Carmine Dodaro, and João Marques-Silva. Core-guided MaxSAT with soft cardinality constraints. In *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 564–573, 2014. doi:10.1007/978-3-319-10428-7_41.

34 António Morgado, Federico Heras, Mark H. Liffiton, Jordi Planes, and João Marques-Silva. Iterative and core-guided MaxSAT solving: A survey and assessment. *Constraints An Int. J.*, 18(4):478–534, 2013. doi:10.1007/S10601-013-9146-2.

35 António Morgado, Alexey Ignatiev, and João Marques-Silva. MSCG: robust core-guided MaxSAT solving. *J. Satisf. Boolean Model. Comput.*, 9(1):129–134, 2014. doi:10.3233/SAT190105.

36 Nina Narodytska and Fahiem Bacchus. Maximum satisfiability using core-guided MaxSAT resolution. In *Proceedings of the 28th Conference on Artificial Intelligence (AAAI)*, pages 2717–2723, 2014. doi:10.1609/AAAI.V28I1.9124.

37 Andreas Niskanen, Jeremias Berg, and Matti Järvisalo. Incremental Maximum Satisfiability. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing (SAT 2022)*, volume 236 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:19, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SAT.2022.14.

38 Toru Ogawa, Yangyang Liu, Ryuzo Hasegawa, Miyuki Koshimura, and Hiroshi Fujita. Modulo based CNF encoding of cardinality constraints and its application to MaxSAT solvers. In *Proceedings of the 25th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 9–17, 2013. doi:10.1109/ICTAI.2013.13.

39 Paul Saikko, Jeremias Berg, and Matti Järvisalo. LMHS: a SAT-IP hybrid MaxSAT solver. In *Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 539–546, 2016. doi:10.1007/978-3-319-40970-2_34.

40 André Schidler and Stefan Szeider. Analyzing reformulation performance in core-guided maxsat solving. In *28th International Conference on Theory and Applications of Satisfiability Testing, SAT 2025, Glasgow, Scotland, August 12-15, 2025*, volume 341 of *LIPIcs*, pages 26:1–26:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.SAT.2025.26.

41 Xujie Si, Xin Zhang, Vasco Manquinho, Mikoláš Janota, Alexey Ignatiev, and Mayur Naik. On incremental core-guided maxsat solving. In Michel Rueher, editor, *Principles and Practice of Constraint Programming*, pages 473–482, Cham, 2016. Springer International Publishing. doi:10.1007/978-3-319-44953-1_30.

42 G. S. Tseitin. *On the Complexity of Derivation in Propositional Calculus*, pages 466–483. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983. doi:10.1007/978-3-642-81955-1_28.