Weight-Aware Transformations for MaxSAT

Jesse Looney
Computer Science
Amherst College
Amherst, USA
jlooney27@amherst.edu

Nikolaj Bjørner
Microsoft Research
Redmond, USA
nbjorner@microsoft.com

Haoze Wu
Computer Science
Amherst College
Amherst, USA
hwu@amherst.edu

Nina Narodytska
VMware Research by Broadcom
Palo Alto, USA
nina.narodytska@broadcom.com

***Abstract*—The core-guided MaxSat algorithms are one of the most successful approaches for solving MaxSat problems. Several of these algorithms are based on the idea of core relaxation, where all clauses in the core must have the same weight. For weighted instances, such cores are obtained by using cloning clauses whose weights are larger than the minimum weight. While cloning is widely used, the question of whether it is necessary has only recently been raised. We investigate this question and propose a novel core transformation procedure that takes weights structure into account. In particular, we consider two MaxSat algorithms, OLL and MaxRes, and show how they can be made weight-aware; we can reuse internal encoding variables in the case of OLL and exploit the MaxSat resolution rule in the case of MaxRes. We implemented our algorithm on top of OLL with sequential counters encoding of cardinality constraints and show that it has competitive performance compared to this baseline. Moreover, there are several families of benchmarks where it significantly outperforms the baseline.**

I. Introduction

Weighted Maximum Satisfiability (MaxSat) is an optimization version of satisfiability that contains both hard clauses and weighted soft clauses that cannot be satisfied together. Solving a weighted partial MaxSat formula involves finding a truth assignment that satisfies the hard clauses along with a subset of soft clauses such that the total weight of violated clauses is minimized. Many industrial applications can be naturally encoded in MaxSat, and successful application areas include software package upgrade and debugging, bioinformatics, timetabling, planning, and scheduling [1]–[7].

Due to its practical importance, several effective algorithms have been developed [7]–[17]. One of the most successful approaches to solving large industrial problems has been core-guided approaches, where a sequence of unsatisfiable cores is extracted using a SAT solver. Each time an UNSAT core is extracted, it performs a preprocessing step where all clauses in the core are made to be the same weight. Then the core is relaxed, and the lower bound of the optimal cost is increased by the value of the minimum weight clause in the core. Such a procedure does not take into account weights structure in the core. In this work, we revisit core relaxation procedures for two classes of core-guided solvers: OLL and MaxRes. We analyze their relaxation algorithms and show that there are new ways to perform core relaxation using transformations that exploit the weight structure of the core.

We make the following contributions. We start by analyzing the OLL algorithm and propose an alternative relaxation procedure. We prove several interesting properties of the proposed transformation and show that it can reduce the number of SAT calls, as it can increase the lower bound of the optimal cost by more than the minimum weight of soft clauses by relaxing multiple, possibly overlapping, cores. We demonstrate how to efficiently implement it by softening auxiliary variables of sequential counters encoding of the reified cardinality constraints.

Next, we investigate the MaxRes algorithm and show that it can naturally extend to cores with heterogeneous weights if we use a slightly less restrictive resolution rule compared to the one used in [18]. This demonstrates that weight-aware relaxation is not tied to OLL but applies to resolution-based core-guided solving.

Finally, we implement the proposed algorithm on top of OLL with an incremental version of sequential counters and develop several hybrid strategies that use both cloning and the new proposed relaxation procedure. We show that we perform comparably to the base version and outperform it on more than half of the benchmark families from the MaxSat Evaluation 2024. In particular, we investigated a number of configurations and showed that our best hybrid solves 405 instances, compared with 401 for the clone-based baseline using the same encoding. Adding our configurations to a portfolio of state-of-the-art core-guided solvers raises the virtual best solver result from 414 to 425 instances, showing that weight-aware relaxation captures structure that the other solvers do not exploit. The main open problem is how to select the transformation for each core.

II. Background

Basic definitions. A maximum satisfiability problem consists of a set of soft clauses $F_s = \{(C_1, w_1), \ldots, (C_m, w_m)\}$, where w_i is the violation cost of the clause C_i, and a set of hard clauses $F_h = \{C_{m+1}, \ldots, C_{m+m'}\}$ over a set of Boolean variables. We denote $\mathsf{Vars}(\psi)$ a set of variables in clauses of ψ. W.l.o.g., we assume that F_h is SAT and $F_s \wedge F_h$ is UNSAT. A literal l is either a variable $x \in \mathsf{Vars}(F_s \cup F_h)$ or its negation $\overline{x}$. A clause C is a disjunction of literals $(l_1 \vee \cdots \vee l_n)$. An assignment I is a mapping $\mathsf{Vars}(F_s \cup F_h) \mapsto \{0, 1\}$. A clause C is satisfied by an assignment, $I(C) = 1$, iff $I(l) = 1$ for some $l \in C$, otherwise C is falsified by I and $I(C) = 0$. A set of clauses F is satisfied by an assignment, $I(F) = 1$, iff $I(C) = 1$ for all $C \in F$. I is a solution of MaxSat if

https://doi.org/

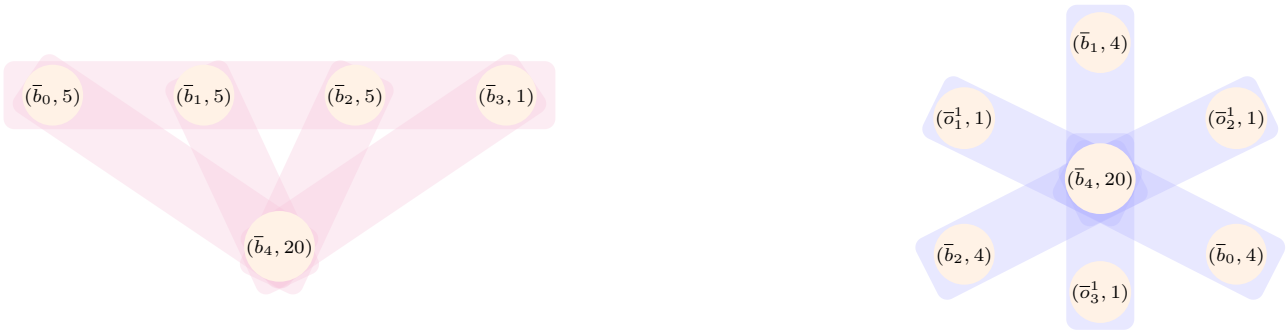

Fig. 1: (a) Left figure: initial cores structure in Exm 2; (b) right figure: cores structure after the 1st core is relaxed in Exm 3.

it satisfies all hard clauses F_h. I is an optimal solution if it minimizes the total weight of violated clauses.

Definition 1 (Core/Minimal core)**.** *An unsatisfiable* core *is a subset of soft clauses $core \subseteq F_s$ such that $core \wedge F_h$ is unsatisfiable. A core is* minimal *if no proper subset is a core.*

Example 1. *Suppose we have four soft clauses $F_s : \{(x_1), (\bar{x}_1), (x_2), (\bar{x}_1 \vee \bar{x}_2)\}$. An example of a core is $core = \{(x_1), (\bar{x}_1), (x_2)\}$ as it is an unsatisfiable subset of clauses. An instance of a minimal core is $\{(x_1), (\bar{x}_1)\}$.* □

Core-guided search. The core-guided solver is one of the main classes of MaxSat solvers. In a nutshell, a core-guided solver starts by assuming that all soft clauses can be satisfied. If that is not the case, it finds an unsatisfiable core and relaxes the formula by allowing one clause in the core to be violated. To perform such transformations, a solver introduces relaxation variables b that are added to the original clauses. For example, an original soft clause (C_1, w_1) is replaced with a hard clause $C = (C_1 \vee b_1)$ and a soft clause (b_1, w_1), where b_1 is a relaxation variable. We assume that this transformation has been performed for all soft clauses. For convenience, we assume that clauses are sorted by their weights in decreasing order in a core: $core = [(b_0, w_0), \ldots, (b_k, w_k)]$, $w_i \geq w_{i+1}, i \in [0, k)$, and use a list instead of a set to describe a core. Next, we introduce our running example.

Example 2 (Running example)**.** *Consider a problem with five soft weighted clauses $F_s = [(\bar{b}_0, 5), (\bar{b}_1, 5), (\bar{b}_2, 5), (\bar{b}_3, 1), (\bar{b}_4, 20)]$. We assume a set of hard clauses F_h that induces the core structure below; its exact form is not relevent to the example, so we omit it. Figure 1(a) shows core structures with 5 minimal unsatisfiable cores. The optimal solution must violate clauses $[(\bar{b}_0, 5), (\bar{b}_1, 5), (\bar{b}_2, 5), (\bar{b}_3, 1)]$. Hence, the optimal cost is 16.* □

III. Motivation

Modern core-guided relaxation-based algorithms perform cloning operations on the core to ensure that all weights are equal to the minimum weight in the core [9], [18], [19]. However, such transformations do not take into account the weight structure of the core, as all core clauses are trimmed to the same weight, and can weaken the derived lower bound [17]. Moreover, cloning a clause means that the same clause may participate in the transformation procedure multiple times. Finally, the weight structure of the clauses in the core may carry important information that is currently ignored. Hence, an interesting question is whether there are alternative transformations to the standard clone-based procedure [6]. Recently, [6], [20] pioneered investigating non-cloning procedures for the core-guided algorithm OLL and proposed the first weight-aware transformation algorithm, AbstCG. We investigate this direction further, showing how weight-aware transformations can be performed incrementally and establishing several new properties of such transformations. In particular, these properties allow the lower bound on the solution cost to be increased by more than the minimum weight in some cases.

We start by reviewing the core-guided algorithm OLL, which uses a clone-based transformation in Section IV. Then, we consider alternative transformations and their properties. In Section VII, we consider the MaxRes algorithm and propose an alternative transformation.

IV. The clone-based core transformation (OLL)

We consider a core-guided algorithm that employs soft cardinality constraints to perform a transformation of a core (OLL). Algorithm 1 shows the core transformation procedure. Given a core $[(\bar{b}_0, w_0), \ldots, (\bar{b}_k, w_k)]$, it starts by computing the minimum weight among all clauses in the core, w (line 1). The first step of the algorithm is *cloning* (lines 4–8). It creates a clone of each soft clause $(\bar{b}_h, w_h)$ that has a weight greater than w into two soft clauses $(\bar{b}_h, w_h - w)$ and $(\bar{b}_h, w)$. This gives it a core with soft clauses of the same weight, $(\bar{b}_i, w)$, $i \in [0, k]$. Second, it introduces soft cardinality constraints to count the number of violated clauses (lines 10–12). After each step, it updates hard F_h and soft F_s clauses. In practice, the incremental version of the core transformation is used, where only the first reified cardinality constraint is introduced, and the remaining constraints are postponed, meaning that they are introduced only if $\bar{o}_i$ is part of a future core.

Example 3. *Let us continue with the running example and the first core $[(\bar{b}_0, 5), (\bar{b}_1, 5), (\bar{b}_2, 5), (\bar{b}_3, 1)]$. Table I (see the first core) shows all cardinality constraints and soft constraints that Algorithm 1 introduces. The first step is cloning, where we clone clauses in the core to make sure that all soft clauses have*

Algorithm 1 The clone-based core transformation

Require: $[(\bar{b}_0, w_0), \ldots, (\bar{b}_k, w_k)]$, F_h, F_s
Ensure: Transformed hard and soft clauses
1: $w \leftarrow \min\{w_0, \ldots, w_k\}$
2: $F'_s = F_s, F'_h = F_h$
3: ▷ Step 1: Cloning
4: **for** each $(\bar{b}_h, w_h) \in core$ **do**
5: **if** $w_h > w$ **then**
6: Replace $(\bar{b}_h, w_h)$ with $(\bar{b}_h, w_h - w)$ and $(\bar{b}_h, w)$
7: $F'_s = F'_s \cup \{(\bar{b}_h, w_h - w)\}$
8: $F'_s = F'_s \setminus \{(\bar{b}_h, w)\}$
9: ▷ Step 2: Introduce soft cardinality
10: **for** $i = 1$ to $k - 1$ **do**
11: $F'_h = F'_h \cup \{\sum_{h \in [0,k]} b_h \leq i \leftrightarrow \bar{o}_i\}$
12: $F'_s = F'_s \cup \{(\bar{o}_i, w)\}$
13: **return** F'_h, F'_s

Algorithm 2 CusCus core transformation

Require: $[(\bar{b}_0, w_0), \ldots, (\bar{b}_k, w_k)]$, $w_i \geq w_{i+1}, i \in [0, k)$, F_h, F_s
Ensure: Transformed hard and soft clauses
1: $W = [w_0, \ldots, w_k, 0]$; $q = |\text{distinct}([w_0, \ldots, w_k])| - 1$
2: $P = [p_0, \ldots, p_q] = [0, \ldots, 0]$; $\delta = [\delta_0, \ldots, \delta_{q-1}] = [0, \ldots, 0]$
3: $r = 0$
4: ▷ Step 1: Prepare auxiliary data
5: **for** $h = 0$ to k **do**
6: **if** $W_h > W_{h+1}$ **then**
7: $p_r = h$
8: $\delta_r = W_h - W_{h+1}$
9: $r = r + 1$
10: $F'_s = F_s \setminus \cup_{h \in [0,k]} \{(\bar{b}_h, w_h)\}$, $F'_h = F_h$
11: ▷ Step 2: Transforming the core
12: **for** $r = 0$ to q **do**
13: $\alpha_r = [0, 1, \ldots, p_r]$
14: **for** $i \in \alpha_r \mid i \neq 0 \vee r \neq q$ **do**
15: $F'_h = F'_h \cup \{\sum_{h \in \alpha_r} b_h \leq i \leftrightarrow \bar{o}^r_i\}$
16: $F'_s = F'_s \cup \{(\bar{o}^r_i, \delta_r)\}$
17: **return** F'_h, F'_s

the same weights $[(\bar{b}_0, 1), (\bar{b}_1, 1), (\bar{b}_2, 1), (\bar{b}_3, 1)]$ *and clones* $[(\bar{b}_0, 4), (\bar{b}_1, 4), (\bar{b}_2, 4)]$. *Next, we introduce reified cardinality constraints using soft reification variables* o^1_i *(the superscript 1 stands for the first iteration) and introduces three reified cardinality constraints. In the case of the incremental version, all but the first cardinality constraint can be added in later iterations [9] (highlighted in blue in Table I).* □

For completeness, we explicitly write down the transformed hard and soft clauses after a core transformation, assuming w_k is minimum weight.

$$F'_h = F_h \bigcup_{i \in (0,k)} \Big[\sum_{h \in [0,k]} b_h \leq i \leftrightarrow \bar{o}_i \Big] \quad (1)$$

$$F'_s = \Big(F_s \setminus \bigcup_{h \in [0,k]} (\bar{b}_h, w_h)\Big) \cup \bigcup_{h \in [0,k], w_h - w_k > 0} (\bar{b}_h, w_h - w_k) \bigcup_{i \in (0,k)} (\bar{o}_i, w_k) \quad (2)$$

$$F_{\text{OLL}} = F'_h \cup F'_s \quad (3)$$

We consider partial execution of OLL on our running example using clone-based transformation.

Example 4. *We will show an execution of the algorithm for three steps in Table I. We record the number of calls to the SAT solver, #, and the current lower bound,* LB. *Suppose the first core is* $[(\bar{b}_0, 5), (\bar{b}_1, 5), (\bar{b}_2, 5), (\bar{b}_3, 1)]$ *which is the same as in Example 3. After relaxing the first core, the core structure is shown in (Figure 1(b)). This core structure is a pathological case for core-guided solvers, as they will have to discover all 6 cores to obtain the optimal cost in the best possible execution. Suppose the second core is* $[(\bar{b}_4, 20), (\bar{o}^1_1, 1)]$. *The minimum weight is 1, so we have to clone* $(\bar{b}_4, 20)$ *into* $(\bar{b}_4, 19)$ *and* $(\bar{b}_4, 1)$. *Hence, we introduce one reified cardinality constraint. The third core is* $[(\bar{b}_4, 19), (\bar{b}_0, 4)]$. *We proceed similarly to the second core. In any execution, there will be at least 4 more calls to the SAT solver to discover all cores in the star structure of the cores that we obtain in Figure 1(b) and obtain the optimal cost which is 16 in this example.* □

#	LB	Reified cardinality	Introduced softs
1	1	$core_1 = [(\bar{b}_0, 5), (\bar{b}_1, 5), (\bar{b}_2, 5), (\bar{b}_3, 1)]$	
		$b_0 + b_1 + b_2 + b_3 \leq 1 \leftrightarrow \bar{o}^1_1$	$(\bar{o}^1_1, 1), (\bar{b}_h, 4), h \in [0, 2]$
		$b_0 + b_1 + b_2 + b_3 \leq 2 \leftrightarrow \bar{o}^1_2$	$(\bar{o}^1_2, 1)$
		$b_0 + b_1 + b_2 + b_3 \leq 3 \leftrightarrow \bar{o}^1_3$	$(\bar{o}^1_3, 1)$
2	2	$core_2 = [(\bar{b}_4, 20), (\bar{o}^1_1, 1)]$	
		$b_4 + o^1_1 \leq 1 \leftrightarrow \bar{o}^2_1$	$(\bar{o}^2_1, 1), (\bar{b}_4, 19)$
3	6	$core_3 = [(\bar{b}_4, 19), (\bar{b}_0, 4)]$	
		$\bar{b}_4 + \bar{b}_0 \leq 1 \leftrightarrow \bar{o}^{1,3}_1$	$(\bar{o}^{1,3}_1, 4), (\bar{b}_4, 15)$

TABLE I: Three iterations from Example 4.

Core: $[(\bar{b}_0, 5), (\bar{b}_1, 5), (\bar{b}_2, 5), (\bar{b}_3, 1)]$.		
Aux. data	Reified cardinality	New soft
$r = 1$:	$b_0 + b_1 + b_2 + b_3 \leq 1 \leftrightarrow \bar{o}^1_1$	$(\bar{o}^1_1, 1)$
$\alpha_1 = [0, 1, 2, 3]$	$b_0 + b_1 + b_2 + b_3 \leq 2 \leftrightarrow \bar{o}^1_2$	$(\bar{o}^1_2, 1)$
and $\delta_1 = 1$	$b_0 + b_1 + b_2 + b_3 \leq 3 \leftrightarrow \bar{o}^1_3$	$(\bar{o}^1_3, 1)$
$r = 0$:	$b_0 + b_1 + b_2 \leq 0 \leftrightarrow \bar{o}^0_0$	$(\bar{o}^0_0, 4)$
$\alpha_0 = [0, 1, 2]$	$b_0 + b_1 + b_2 \leq 1 \leftrightarrow \bar{o}^0_1$	$(\bar{o}^0_1, 4)$
and $\delta_0 = 4$	$b_0 + b_1 + b_2 \leq 2 \leftrightarrow \bar{o}^0_2$	$(\bar{o}^0_2, 4)$

TABLE II: The CusCus transformation on processing the core $[(\bar{b}_0, 5), (\bar{b}_1, 5), (\bar{b}_2, 5), (\bar{b}_3, 1)]$.

V. The CusCus core transformation

The main idea of CusCus is to use parts of the cardinality constraint as soft constraints. We will show that our transformation is cost-preserving and allows deriving a stronger lower bound using simple implied constraints. Algorithm 2 presents our core transformation. It takes a core with clauses sorted by weights in decreasing order $[(\bar{b}_0, w_0), \ldots, (\bar{b}_k, w_k)]$, where $w_i \geq w_{i+1}, i \in [0, k)$, as input. Next, it introduces auxiliary arrays. First, it defines an array of sorted weights $W = [w_0, \ldots, w_k, 0]$, where $W_h = W[h]$, $h \in [0, k+1]$, are the weights of the clauses in the core, and for convenience the last value is zero (line 1). Next, it defines an array of indices over W to capture the positions in W where the weight value changes, $P = [p_0, \ldots, p_q]$. In line 7, we set $p_r = h$ if $W_h > W_{h+1}$, so we have a change of weights in W. We also record the difference between consecutive non-equal weight values as $\delta_r = W_h - W_{h+1} =$ [by construction] $= W_{p_r} - W_{p_{r+1}}$, for $r \in [0, q]$ (line 8), where we set $p_{q+1} = k+1$

so that $W_{p_{q+1}} = 0$ and $\delta_q = W_k$. At the second step, we compute updates of soft and hard clauses. We use the prefix notation on indices: $\alpha_r = [0, 1, \ldots, p_r],\ r \in [0, q]$ (line 13). For each prefix α_r of the sequence of variables $[b_0, \ldots, b_k]$, we introduce soft cardinality constraints (lines 12–16). We exclude $b_0 + \ldots + b_k \leq 0 \leftrightarrow \overline{o}_0^q$ as $\overline{o}_0^q$ must be falsified due to existence of *core*.

Example 5. *We continue with the running example, and let the first core be* $[(\overline{b}_0, 5), (\overline{b}_1, 5), (\overline{b}_2, 5), (\overline{b}_3, 1)]$. *The sorted weight array is* $W = [w_0, w_1, w_2, w_3, w_4] = [5, 5, 5, 1, 0]$. *The value of* $q = 1$*, as it is equal to the number of distinct weights in core minus one:* $q = |\{5, 1\}| - 1 = 1$. *There are two index positions in* W *where the weight changes:* 2*, as* $w_2 > w_3$*, and* 3*, as* $w_3 > w_4$. *Hence,* $P = [p_0, p_1] = [2, 3]$. *The array of differences is* $\delta = [4, 1]$*, as* $\delta_0 = w_2 - w_3 = 4$*, and* $\delta_1 = w_3 - w_4 = 1$. *Table II shows all cardinality constraints and soft constraints we introduce. As* $q = 1$*, we have two prefixes,* $\alpha_0 = [0, 1, 2]$*,* $\alpha_1 = [0, 1, 2, 3]$. *So, we will have two sets of cardinality constraints to count the number of ones per prefix of weights with the same value.* □

We explicitly write down the transformed hard F_h and soft F_s clauses.

$$F_h' = F_h \bigcup_{r \in [0,q]} \bigcup_{i \in \alpha_r | i \neq 0 \vee r \neq q} \big[\sum_{h \in \alpha_r} b_h \leq i \ \leftrightarrow \ \overline{o}_i^r \big] \quad (4)$$

$$F_s' = \big(F_s \setminus \bigcup_{h=0}^{k} (\overline{b}_h, w_h)\big) \bigcup_{r \in [0,q]} \bigcup_{i \in \alpha_r | i \neq 0 \vee r \neq q} (\overline{o}_i^r, \delta_r) \quad (5)$$

$$F_{\text{CusCus}} = F_h' \cup F_s' \quad (6)$$

In the next section, we prove that Algorithm 2 performs a cost-preserving transformation.

A. Theoretical properties of CusCus

First, we prove the correctness property that the transformation is cost-preserving. We say that α_r is shorter than α_d iff $|\alpha_r| < |\alpha_d|$. We denote $spr(j)$ as the index of the shortest prefix α_r, $spr(j) = r$, that contains the jth index, so that $j \in (p_{r-1}, p_r]$. In Example 5, if $j = 3$ then $spr(3)$ is equal to 1, as $\alpha_1 = [0, 1, 2, 3]$ is the shortest prefix that contains 3. We also denote $l_r = p_r - p_{r-1}$, $r \in (0, q]$, the number of indices that α_r adds over α_{r-1}.

Proposition 1. *Consider a clause* $(\overline{b}_j, w_j)$ *in core* $= [(\overline{b}_0, w_0), \ldots, (\overline{b}_k, w_k)]$ *and the transformation defined by Algorithm 2. Then* $w_j = \sum_{h=r}^{q} \delta_h$*, where* $r = spr(j)$.

Proof. As $j \in (p_{r-1}, p_r]$, we get $w_j = w_{p_r}$ as all weights w_g, $g \in (p_{r-1}, p_r]$, have the same value. Therefore, by construction, we have $w_j = w_{p_r} = \underbrace{w_{p_r} - w_{p_{r+1}}}_{\delta_r} + \underbrace{w_{p_{r+1}} - w_{p_{r+2}}}_{\delta_{r+1}} + \ldots + \underbrace{w_{p_{q-1}} - w_{p_q}}_{\delta_{q-1}} + \underbrace{w_{p_q} - 0}_{\delta_q} = \sum_{h=r}^{q} \delta_h$. □

Theorem 1. *Consider a core* $= [(\overline{b}_0, w_0), \ldots, (\overline{b}_k, w_k)]$ *and the transformation defined by Algorithm 2. The transformation is cost-preserving.*

Proof. Algorithm 2 transformation is explicitly described by Formula 6. To prove the result, we need to show that (a) if I is a solution of $F_h \cup F_s$ (the formula before the transformation is applied), then I is also a solution of $F_h' \cup F_s'$ (Formula 6 after the transformation is applied), and (b) the cost of I is the same in $F_h \cup F_s$ and $F_h' \cup F_s'$.

First, we observe that Formula 6 does not change the set of solutions of the formula before transformation $F_h \cup F_s$, as we only add hard constraints that are reified, and each reification variable forms a soft unit clause. Hence, we only need to prove that the cost of each solution remains unchanged after the transformation.

Let I be a solution of the formula $F_h \cup F_s$ with cost $cost(I)$. Consider clauses in the core that are violated by I. We only need to consider clauses in the core, as we do not change other soft clauses during the transformation. Let B be the violated clauses sorted by weight: $B = [(\overline{b}_{t_1}, w_{t_1}), \ldots, (\overline{b}_{t_s}, w_{t_s})]$. Hence, the cost is $cost(I) = \sum_{j=1}^{s} cost(b_{t_j}) = \sum_{j=1}^{s} w_{t_j}$.

Next, we need to prove that the total violation cost $\sum_{j=1}^{s} cost(b_{t_j})$ is the same in the transformed formula for I. The total cost of violation for I in F_{CusCus} is equal to the number of violated soft clauses in each prefix times the cost of this prefix. More formally,

$$\sum_{r \in [0,q]} \delta_r \times \sum_{j=1}^{s} (t_j \in \alpha_r) = \sum_{r \in [0,q]} \delta_r \times |\{t_1, \ldots, t_s\} \cap [0, p_r]|$$

$$= \sum_{j=1}^{s} \sum_{r \in [0,q]} \delta_r \times |\{t_j\} \cap [0, p_r]|$$

$$= \sum_{j=1}^{s} \sum_{r=spr(t_j)}^{q} \delta_r = \text{ by Proposition 1} = \sum_{j=1}^{s} w_{t_j}.$$

Hence, the violation cost of I is identical in the transformed formula. As we considered an arbitrary solution of $F_h \cup F_s$, this proves that the transformation preserves the cost of any solution. □

We prove a few useful properties of the transformation.

Proposition 2. *The following properties hold after* CusCus *core transformation:*

1) $\overline{o}_0^q = 0$
2) $\overline{o}_i^r = 0 \quad \Rightarrow \quad \overline{o}_{i-1}^r = 0,\ i \in \alpha_r, i \neq 0,\ r \in [0, q]$
3) $\overline{o}_i^r = 0 \quad \Rightarrow \quad \overline{o}_{i-l_r}^{r-1} = 0,\ i \in \alpha_r, i \geq l_r,\ r \in (0, q]$

Proof. The first two properties hold for clone-based transformation and the same arguments hold for CusCus. So, we only need to prove the third property. By definition, $\overline{o}_i^r = 0$ is equivalent to $b_0 + \ldots + b_{p_r} > i$. First, we subtract l_r from both sides: $b_0 + \ldots + b_{p_r} - l_r > i - l_r$. We rewrite as $[b_0 + \ldots + b_{p_{r-1}}] + [b_{p_{r-1}+1} + \ldots + b_{p_r} - l_r] > i - l_r$. By construction, we know that $b_{p_{r-1}+1} + \ldots + b_{p_r} - l_r \leq 0$

as $|\{b_{p_{r-1}+1}, \ldots, b_{p_r}\}| \leq l_r$. By subtracting the last two inequalities, we get $b_0 + \ldots + b_{p_{r-1}} > i - l_r$. Hence, $\overline{o}_{i-l_r}^{r-1} = 0$. □

Next, we prove an interesting property of CUSCUS. We show that, given a core, we can derive the existence of a different core, and we can transform both of them in a certain order. This allows us to increase the lower bound of the cost by more than the minimal weight of the clauses in the core without calling a SAT solver. Let us denote A as a set of soft clauses $\{(\overline{b}_0, w_0), \ldots, (\overline{b}_f, w_f)\}$.

Proposition 3. *Let F_h and F_s be the hard and soft clauses at some iteration of the algorithm. If $A \cup \{(\overline{o}_i^r, \delta_r)\}$ is a core of the formula $F_h \cup F_s$, then $A \cup \{(\overline{o}_{i-l_r}^{r-1}, \delta_{r-1})\}$ is also a core of $F_h \cup F_s$.*

Proof. We prove by contradiction. As $A \cup \{(\overline{o}_i^r, \delta_r)\}$ is a core, by definition of a core, we have the following unsatisfiable formula $Q = F_h \wedge_{\overline{b}_h \in A} \overline{b}_h \wedge \overline{o}_i^r$. Suppose $A \cup \{(\overline{o}_{i-l_r}^{r-1}, \delta_{r-1})\}$ is not a core. So we have the satisfiable following formula $R = F_h \wedge_{\overline{b}_h \in A} \overline{b}_h \wedge \overline{o}_{i-l_r}^{r-1}$.

Both Q and R share a subformula $F_h \wedge_{\overline{b}_h \in A} \overline{b}_h$. We consider two cases. First, if this sub-formula has no solutions, we have a contradiction to R is SAT. Second, we suppose that $F_h \wedge_{\overline{b}_h \in A} \overline{b}_h$ has a solution I that can be extended to a solution of R. Note that $\overline{o}_{i-l_r}^{r-1}$ must be set to *true* in I. By Proposition 2, Property 3, $\overline{o}_i^r$ must be *true* as well. Hence, I is a solution of Q. This leads to a contradiction. □

Using Proposition 3, we can prove that *both* cores that satisfy Proposition 3 can be relaxed consecutively. Lemma 1 proves this important property and allows us to *increase the lower bound multiple times without calling the SAT solver*. Let $(\overline{o}_i^r, \delta_r)$ and $(\overline{o}_{i-l_r}^{r-1}, \delta_{r-1})$ be two soft clauses introduced by processing some core before the current iteration.

Lemma 1. *Suppose, we have a core $= [(\overline{c}_0, w_0), \ldots, (\overline{c}_k, w_k), (\overline{o}_i^r, \delta_r)]$, where $w_k > \delta_r$. Consider the* CUSCUS *transformation $F_{\text{CUSCUS}} = F_h' \cup F_s'$ after processing this core. If a soft clause $(\overline{o}_{i-l_r}^{r-1}, \delta_{r-1}) \in F_s'$, then $\{(\overline{o'}_0^{q-1}, w_k - \delta_r), (\overline{o}_{i-l_r}^{r-1}, \delta_{r-1})\}$ is a core of F_{CUSCUS}, where $\overline{o'}_0^{q-1}$ is a reification variable introduced at the current iteration.*

Proof. We prove by contradiction. As we have a *core* $= [(\overline{c}_0, w_0), \ldots, (\overline{c}_k, w_k), (\overline{o}_i^r, \delta_r)]$ the following formula is unsatisfiable: $Q = F_h \wedge \overline{c}_0 \wedge \ldots \wedge \overline{c}_k \wedge \overline{o}_i^r$, where F_h denotes the hard clauses at the start of the current iteration, so it already contains the reified constraint defining $\overline{o}_i^r$.

Suppose $\{(\overline{o'}_0^{q-1}, w_k - \delta_r), (\overline{o}_{i-l_r}^{r-1}, \delta_{r-1})\}$ is a not core of F_{CUSCUS}. Consider a formula $R = F_h' \wedge \overline{o'}_0^{q-1} \wedge \overline{o}_{i-l_r}^{r-1}$. This formula is satisfiable by our assumption. Then there exists a solution I. Note that $\overline{o}_{i-l_r}^{r-1}$ must be *true* and $\overline{o'}_0^{q-1}$ must be *true* in this solution.

The sorted weights of the current *core* are $[w_0, \ldots, w_k, \delta_r, 0]$. As $w_k > \delta_r$, position k is a position where the weight changes, and it is the second to last such position, since the last one is $k+1$, where δ_r drops to 0. Hence $p_{q-1} = k$ and $\alpha_{q-1} = [0, \ldots, k]$, which also gives $\delta_{q-1} = w_k - \delta_r$ as the weight of the introduced soft clause. So, we have $c_0 + c_1 + c_2 + \ldots + c_k \leq 0 \leftrightarrow \overline{o'}_0^{q-1}$.

As $\overline{o'}_0^{q-1}$ is *true* in I, we conclude that $I(c_0) + I(c_1) + I(c_2) + \ldots + I(c_k) = 0$. As $\overline{o}_{i-l_r}^{r-1}$ is *true* in I, we conclude by Proposition 2, Property 3, that $I(\overline{o}_i^r) = 1$. Hence, $F_h' \wedge \overline{c}_0 \wedge \ldots \wedge \overline{c}_k \wedge \overline{o}_i^r$ must be *true*. As F_h' only adds definitions of fresh variables to F_h, every solution of F_h' restricted to $\mathsf{Vars}(F_h)$ is a solution of F_h, we can conclude that $F_h \wedge \overline{c}_0 \wedge \ldots \wedge \overline{c}_k \wedge \overline{o}_i^r$ is *true*. Hence, Q is satisfiable. This leads to a contradiction. □

An execution of OLL with CUSCUS. Next, we consider partial execution of CUSCUS on our running example. To avoid writing multiple sets of reified constraints per core, we assume that there exists an encoding `ReifiedCard` for all cardinality constraints introduced by CUSCUS. It takes unit soft clauses and a value r as inputs and produces a SAT encoding of all reified cardinality constraints and additionally makes available all reification variables. More concretely, given a core $[(\overline{b}_0, w_0), \ldots, (\overline{b}_k, w_k)]$ and $r, r \in [0, q]$, we can assume the existence of the following SAT encoding which allows us to simplify Formula 4.

$$\texttt{ReifiedCard}\left([(\overline{b}_0, w_0), \ldots, (\overline{b}_k, w_k)], r\right) = \\ = \bigcup_{i \in \alpha_r \mid i \neq 0 \vee r \neq q} \left[\sum_{h \in \alpha_r} b_h \leq i \;\leftrightarrow\; \overline{o}_i^r\right], \quad (7)$$

$$F_h = F_h \bigcup_{r \in [0,q]} \texttt{ReifiedCard}\left([(\overline{b}_0, w_0), .., (\overline{b}_k, w_k)], r\right) \quad (8)$$

Example 6. *Let us consider an execution of the running example using* CUSCUS*. We will demonstrate the algorithm's execution over two steps. Table III presents reified cardinality constraints and soft constraints introduced upon encountering each core.*

Suppose the first core is $[(\overline{b}_0, 5), (\overline{b}_1, 5), (\overline{b}_2, 5), (\overline{b}_3, 1)]$, which is the same as in Example 5. Thus, we introduce two sets of reified cardinality constraints. Now, we write them in a compact form using `ReifiedCard`*. Soft clauses are obtained using* `ReifiedCard`*. Note that we have an additional superscript to distinguish the iteration when the reification variable is introduced. For example, the second superscript in $o_1^{1,\mathbf{1}}$ shows the iteration number. We again highlight in blue the soft clauses that can be postponed in the incremental version.*

Suppose the second core is $[(\overline{b}_4, 20), (\overline{o}_1^{1,1}, 1)]$. We have $W = [w_0, w_1, w_2] = [20, 1, 0]$. The value of $q = 1$, as it is equal to the number of distinct weights in core (which is 2) minus one. There are two index positions in W where the weight changes: 0, as $w_0 > w_1$, and 1, as $w_1 > w_2$. Hence, $P = [p_0, p_1] = [0, 1]$. The array of differences is $\delta = [19, 1]$, as $\delta_0 = w_0 - w_1 = 19$, and $\delta_1 = w_1 - w_2 = 1$. Two prefixes, $\alpha_0 = [0]$, $\alpha_1 = [0, 1]$. We introduce two sets of reified cardinality constraints over the variables in the core. Since the cardinality constraint over α_0 contains only one variable, we

#	LB	Reified cardinality	New soft
1	1	$core_1 = [(\bar{b}_0, 5), (\bar{b}_1, 5), (\bar{b}_2, 5), (\bar{b}_3, 1)]$;	
		`ReifiedCard`$(core_1, 1)$	$(\bar{o}_1^{1,1}, 1), (\bar{o}_2^{1,1}, 1), \bar{o}_3^{1,1}, 1)$
		`ReifiedCard`$(core_1, 0)$	$(\bar{o}_0^{0,1}, 4), (\bar{o}_1^{0,1}, 4), (\bar{o}_2^{0,1}, 4)$
2	2	$core_2 = [(\bar{b}_4, 20), (\bar{o}_1^{1,1}, 1)]$	
		`ReifiedCard`$(core_2, 1)$	$(\bar{o}_1^{1,2}, 1)$
		`ReifiedCard`$(core_2, 0)$	$(\bar{o}_0^{0,2}, 19) \leftrightarrow (\bar{b}_4, 19)$
	6	By Lemma 1: as $[(\bar{b}_4, 20), (\bar{o}_1^{1,1}, 1)]$ is a core then $[(\bar{b}_4, 19), (\bar{o}_0^{0,1}, 4)]$ is a core. Hence: $core_3 = [(\bar{b}_4, 19), (\bar{o}_0^{0,1}, 4)]$	
		`ReifiedCard`$(core_3, 1)$	$(\bar{o}_1^{1,3}, 4)$
		`ReifiedCard`$(core_3, 0)$	$(\bar{o}_0^{0,3}, 15) \leftrightarrow (\bar{b}_4, 15)$

TABLE III: Two iterations with CUSCUS (Exm. 6).

k\p	0	1	2		3	
				$r = 0, p_1 = 2$		$r = 1, p_0 = 3$
3					s_3^3	$\leftrightarrow o_3^1$
2			s_2^2	$\leftrightarrow o_2^0$	s_2^3	$\leftrightarrow o_2^1$
1		s_1^1	s_1^2	$\leftrightarrow o_1^0$	s_1^3	$\leftrightarrow o_1^1$
0	s_0^0	s_0^1	s_0^2	$\leftrightarrow o_0^0$	s_0^3	$\leftrightarrow o_0^1$
	b_0	b_1	b_2		b_3	

TABLE IV: Sequential counter encoding variables and their correspondence to the reification o variables for the first core (Example 7).

have a trivial constraint: $b_4 \leq 0 \leftrightarrow \bar{o}_0^{0,2}$. *Hence,* $\bar{o}_0^{0,2} = \bar{b}_4$, *and we retain the original variable name for simplicity.*

Finally, we can apply Lemma 1 and save *a SAT call. We have* $core = [(\bar{b}_4, 20), (\bar{o}_i^{r,1}, \delta_r) := (\bar{o}_1^{1,1}, 1)]$. *Thus,* $r = 1$, $i = 1$, *and* $\delta_r = 1$. *We consider* $l_r = l_1 = p_1 - p_0 = 1$. *Next we check if* $(\bar{o}_{i-l_r}^{r-1,1}, \delta_{r-1}) = (\bar{o}_0^{0,1}, \delta_{r-1})$ *belong to soft clauses. The answer is yes, as we have* $(\bar{o}_0^{0,1}, 4)$, *so* $\delta_{r-1} = 4$. *By Lemma 1, it forms a core with* $(\bar{o}_0^{q-1,2}, w_k - \delta_r) = (\bar{o}_0^{0,2}, 19)$. *As explained above in this example,* $(\bar{o}_0^{0,2}, 19) = (\bar{b}_4, 19)$. *Hence,* $[(\bar{b}_4, 19), (\bar{o}_0^{0,1}, 4)]$ *is a derived core, and we can process it using Algorithm 2. There exists an execution with* only *two additional calls to the SAT solver to discover all cores and finds the optimal bound 16. In total, it saves three calls compared to the best execution in Example 4.* □

VI. ENCODINGS OF REIFIED CARDINALITY CONSTRAINTS

We consider a concrete implementation of the `ReifiedCard` encoding. Interestingly, we can simply exploit the incremental sequential counter encoding without any changes to the hard constraints, but reuse internal variables of the encoding to encode soft constraints.

Let us recap the sequential counters encoding for the cardinality constraint [21]. It introduces $O(nk)$ auxiliary variables to track the number of ones in the cardinality constraint. The new variables count the sum of the input variables $b_0, \ldots, b_k$ in unary for each cumulative sum $\sum_{h=0}^{p} b_h$: $i \in [0, k], \quad p \in [i, k]$, $s_i^p \leftrightarrow \sum_{h=0}^{p} b_h \geq i + 1$. If s_i^k is *true*, then we know that the sum of variables $b_0, \ldots, b_k$ is greater than i.

Proposition 4. *Let a core be* $[(\bar{b}_0, w_0), \ldots, (\bar{b}_k, w_k)]$ *and* `ReifiedCard`$\left([(\bar{b}_0, w_0), \ldots, (\bar{b}_k, w_k)], r\right)$ *be an encoding of reified cardinality constraint defined by Formula 7. The following holds:* $o_i^r = s_i^{p_r}$, $i \in \alpha_r$.

Proof. By definition, for any p_r, we have the corresponding prefix α_r such that $\sum_{h \in \alpha_r} b_h \leq i \leftrightarrow \bar{o}_i^r$. On the other hand, by definition of sequential counter variables we have that $\sum_{h=0}^{p_r} b_h = \sum_{h \in \alpha_r} b_h \geq i + 1 \leftrightarrow s_i^{p_r}$. Taking negation of the first expression gives us $\sum_{h \in \alpha_r} b_h > i \leftrightarrow o_i^r$. Hence, $o_i^r = s_i^{p_r}$. □

Using Proposition 4 we can implement `ReifiedCard` easily using internal variables of the sequential counters encoding.

Example 7. *Consider the running example and the first core* $[(\bar{b}_0, 5), (\bar{b}_1, 5), (\bar{b}_2, 5), (\bar{b}_3, 1)]$. *Table IV shows all auxiliary variables in the sequential counters encoding for* $b_0 + b_1 + b_2 + b_3 \leq 3$. *We show how they correspond to the reification variables for* $p_0 = 2$ *and* $p_1 = 3$. □

Abstract CG. Finally, we discuss another weight-aware transformation procedure, AbstCG [6], and contrast it with CUSCUS. To the best of our knowledge, AbstCG is the only existing core-guided transformation that relaxes a core without flattening all weights to the minimum. It *abstracts* each group of equal-weight literals into a counter whose output feeds the next group, processing groups from the largest coefficient down. In particular, AbstCG first partitions $core$ into m disjoint sets $core = G_1 \vee \cdots \vee G_m$ so that all variables in the same set G_i have the same coefficient. Starting from G_1, which corresponds to the largest coefficient in the core, AbstCG introduces a set of constraints for each G_i. $ab_i = \left(in_i, \left\{\sum_{x \in in_i} x \geq j \leftrightarrow o_i^j \mid 1 \leq j \leq |in_i|\right\}, out_i\right)$. The inputs $in_i = G_i \cup out_{i-1}$ consist of the variables in G_i and the outputs of ab_{i-1}, and the objective is updated by processing each abstraction set $ab_i = (in_i, D_i, out_i)$ in order, starting from $i = 1$. The coefficient of each $x \in in_i$ is decreased by $w_i = \min(\{w_j \mid x_j \in in_i\})$, and each output $x \in out_i$ is included in the objective with coefficient w_i. After processing each ab_i, a constant w_m is added to the lower bound. The key difference for CUSCUS is that we do not have dependencies between the relaxation variables of different groups, which allows us to *incrementally* introduce cardinality constraints. In AbstCG the inputs of ab_i include the outputs out_{i-1}, so the counter for group i cannot be built before that for group $i - 1$. In CUSCUS each prefix constraint is over $b_0, \ldots, b_{p_r}$ directly, with no reference to another prefix, so it can be postponed until its reification variable appears in a core. For instance, in Example 6, we only introduce two relaxation variables, whereas AbstCG introduces four variables, as it has to introduce three variables for the first group. We will see in the experimental evaluation that the different encodings lead to complementary performance.

VII. RESOLUTION-BASED CORE TRANSFORMATION

Next, we focus on a different core-guided algorithm, MAXRES [18]. We consider its core transformation procedure

and show that it can easily incorporate clauses with different weights without a cloning step and keeps a compact encoding as in the original MAXRES formulation. We focus on a *theoretical* contribution here to show that weight-aware core relaxation is not specific to OLL but extends to resolution-based core-guided solving. MAXRES is not a state-of-the-art solver at present and has not been actively developed in recent years, so a competitive implementation would require substantial engineering orthogonal to our question; most recent core-guided work, including every solver we compare against, builds on OLL.

A. MAXRES *original core transformation*

MAXRES approach to solving MAXSAT that is based on restricted MAXSAT resolution [22]:

$$\frac{(x \vee A, w_1) \qquad (\overline{x}, w_2)}{(A, w_2), (x \vee A, w_1 - w_2), (\overline{x} \vee \overline{A}, w_2)} \tag{9}$$

We call $(A,\ w_2)$ the *resolvant* and $\{(x \vee A, w_1 - w_2), (\overline{x} \vee \overline{A},\ w_2)\}$ the *compensation clauses*. In [18], it was assumed that $w_1 = w_2$ as cloning was performed before a resolution step, so the clause $(x \vee A,\ w_1 - w_2)$ vanishes.

Consider how MAXRES works. Given a core $[(\overline{b}_0, w_0), \ldots, (\overline{b}_k, w_k)]$, we first perform cloning: $[(\overline{b}_0, w), \ldots, (\overline{b}_k, w)]$, $w = w_k$ where w_k is the minimum weight. Given the core, we know that the hard clause $(b_0 \vee \cdots \vee b_k)$ is implied by the formula. Hence, we resolve this hard clause with the unit soft clauses in the core $(\overline{b}_0, w), \ldots, (\overline{b}_k, w)$, deriving the empty clause $(\square, w)$ using $k+1$ applications of restricted MAXRES. The first step is to resolve $(b_0 \vee \cdots \vee b_k, \infty)$ and $(\overline{b}_0, w)$.

$$\frac{(b_0 \vee \cdots \vee b_k, \infty) \qquad (\overline{b}_0, w)}{(b_1 \vee .. \vee b_k, w)(b_0 \vee .. \vee b_k, \infty)(\overline{b}_0 \vee \overline{b_1 \vee .. \vee b_k}, w)} \tag{10}$$

We must remove $(\overline{b}_0, w)$ and keep the resolvant and the compensation clause. In the second step we perform the resolution with the second clause in a core $(\overline{b}_1, w)$ with $(b_1 \vee \cdots \vee b_k, w)$, obtain two clauses $(b_2 \vee \cdots \vee b_k, w)$ and $(\overline{b}_1 \vee \overline{b_2 \vee \cdots \vee b_k}, w)$ and remove both premises. Each subsequent step is similar. Finally, when $i = k$ we resolve (b_k, w) and $(\overline{b}_k, w)$ to obtain $(\square, w)$. The clauses remaining after $k+1$ MAXRES steps are performed:

$$(\square, w) \quad (b_0 \vee .. \vee b_k, \infty)(\overline{b_i} \vee \overline{b_{i+1} \vee .. \vee b_k}, w), \tag{11}$$

where $0 \leq i < k$. The compensation clauses are not in clausal form, so new variables d_i with $d_i \leftrightarrow (b_i \vee \cdots \vee b_k)$ are introduced to encode them. With d_i we can convert compensation clauses as follows (assume $d_k = 1$): $d_i \leftrightarrow (b_i \vee d_{i+1})$, $1 \leq i < k$, $(\overline{b_i} \vee \overline{d_{i+1}}, w)$, $0 \leq i < k$. Note that the empty clause is removed from the formula to increase the lower bound by w. Correctness of this transformation follows from correctness of the MAXSAT resolution rule.

B. MAXRES *new core transformation*

We show that the MAXRES approach can be easily modified to incorporate weights. It requires several minor modifications. The first modification is that soft clauses can have different weight. Hence, each application of the resolution rule (Equation 9) produces two compensation clauses. The second modification we *must* resolve in the order of the decreasing weights; otherwise, we get clauses with negative weight.

We are given a core $[(\overline{b}_0, w_0), \ldots, (\overline{b}_k, w_k)]$ and we assume that it is *sorted* by weights, so $w_i \geq w_{i+1}, i \in [0, k)$. The first step is to resolve $(b_0 \vee \cdots \vee b_k, \infty)$ and $(\overline{b}_0, w_0)$ as in Equation 10. In the second step we resolve $(b_1 \vee \cdots \vee b_k, w_0)$ and $(\overline{b}_1, w_1)$, obtain the resolvant $(b_2 \vee \cdots \vee b_k, w_1)$ and *two* compensation clauses $(b_1 \vee \cdots \vee b_k, w_0 - w_1), (\overline{b}_1 \vee \overline{b_2 \vee \cdots \vee b_k}, w_1)$. Each subsequent step is similar. Finally, when $i = k$ we resolve (b_k, w_k) and $(\overline{b}_k, w_k)$ to obtain $(\square, w_k)$. The clauses remaining after all MAXRES steps are $(\square, w_k)$, $(b_0 \vee \ldots \vee b_k, \infty)$ and compensation clauses: $(\overline{b_i} \vee \overline{b_{i+1} \vee \ldots \vee b_k}, w_i), i \in [0, k); (\mathbf{b_i} \vee \ldots \vee \mathbf{b_k}, \mathbf{w_{i-1}} - \mathbf{w_i}), \mathbf{i} \in [\mathbf{0}, \mathbf{k})$.

In bold, we highlight additional compensation clauses compared to Equations 11. Again, new variables d_i with $d_i \leftrightarrow (b_i \vee \cdots \vee b_k)$ are introduced. With d_i we can then convert each compensation clause as follows (we assume $d_k = 1$): $(\overline{b_i} \vee \overline{d_{i+1}}, w_i),\ i \in [0, k),\ d_i \leftrightarrow (b_i \vee d_{i+1}), (\mathbf{d_i}, \mathbf{w_{i-1}} - \mathbf{w_i}),\ i \in [1, k)$. As above, the empty clause is removed from the formula to increase the lower bound by w_k. Note that if $w_i = w_{i-1}$ for some i, then we do not need to introduce soft clause $(d_i, w_{i-1} - w_i) \leftrightarrow (d_i, 0)$.

VIII. RELATED WORK

There are a number of core-guided MAXSAT solvers for the weighted MAXSAT problem [5]–[9], [17], [18], [23]–[25]. In [24], the authors observed that stronger bounds can be derived if one keeps track of clause weights in a core as a set of pseudo-Boolean constraints. The main difference in our work compared to clone-based techniques for core relaxation is that we perform a different transformation that takes into account weights structure. In [17], the authors also observed that clone-based relaxation loses information about weights and proposed using linear programming to derive a set of new soft constraints and a stronger bound. We do not use any additional procedure to obtain stronger bounds; however, we do not guarantee that we find the optimal lower bound. In [6], an alternative procedure was proposed that uses a different encoding. As we discuss in Section VI, CUSCUS allows relaxation variables to be introduced incrementally.

IX. EXPERIMENTAL EVALUATION

Implementation. We implemented CUSCUS on top of the OLL-based MaxSAT solver RC2 [26], which won the exact weighted track of the MaxSAT Evaluation in 2018–2019 [27]. By default, RC2 uses an incremental totalizer encoding `ITotalizer` from the PySAT Python library [28]–[30] to encode cardinality constraints. We extended PySAT with an incremental sequential counters encoding `ISeqCounter`

family	#	cgss2		cgss2-dyn		clone		cuscus		h4		s4		s2c4	
		Slv.	Time	Slv.	Time	Slv.	Time	Slv.	Time	Slv.	Time	Slv.	Time	Slv.	Time
abstraction	11	11	209.5	11	**184.1**	6	524.2	6	618.3	6	592.2	6	584.1	6	561.1
af-synthesis	15	14	**1323.8**	14	1431.4	13	1864.0	6	2071.6	13	1581.9	12	1500.7	11	1642.6
auctions	15	14	4.4	15	**45.8**	14	2.1	14	2.5	15	164.2	14	2.0	14	2.0
BTBNSL	15	0		0		0		0		0		1	**412.9**	1	1847.9
causal	15	11	851.5	9	298.6	12	134.9	13	421.2	12	109.9	12	146.2	14	**253.1**
correlation	15	3	150.7	3	181.6	7	436.0	6	863.5	7	779.0	7	402.6	7	**244.7**
CSG	10	10	467.7	9	64.8	10	19.4	10	51.5	10	21.1	10	**17.9**	10	19.3
css	11	11	36.0	11	34.5	11	**7.1**	11	8.2	11	7.4	11	7.4	11	7.2
dalculus	15	15	0.7	15	0.6	15	**0.4**	15	**0.4**	15	**0.4**	15	**0.4**	15	**0.4**
decision	15	0		0		0		0		0		0		0	
drmx	15	15	**110.0**	15	116.3	15	323.1	14	90.6	14	99.6	15	336.7	14	92.3
frb	15	13	137.7	13	361.3	15	83.6	15	76.1	15	**74.6**	15	88.5	15	76.6
haplotyping	15	15	23.1	15	**4.2**	15	22.3	15	23.7	15	21.5	15	26.3	15	22.9
IMLIB	16	16	**0.0**	16	**0.0**	16	0.3	16	0.3	16	0.3	16	0.3	16	0.3
judgment	15	0		0		0		0		0		0		0	
lisbon	15	3	563.4	3	872.2	6	553.7	6	**458.1**	6	461.9	6	599.5	6	502.6
max	15	12	**67.5**	12	236.9	12	89.1	12	116.6	12	133.3	12	125.4	12	116.1
MaxSATQueries	15	13	122.4	12	0.2	13	52.3	13	40.8	13	40.3	13	48.1	14	**250.1**
metro	15	15	127.1	15	**112.4**	15	140.0	15	114.5	15	115.7	15	139.3	15	113.8
MinimumWeight	10	0		0		0		0		0		0		0	
mpe	15	10	355.6	11	125.2	12	17.2	11	278.0	12	17.6	13	138.8	14	**297.2**
mpmcs	15	15	0.1	15	**0.0**	15	2.1	15	2.1	15	2.1	15	2.1	15	2.1
ParametricRBAC	15	2	**315.8**	2	541.1	1	686.8	1	490.7	1	694.9	1	890.6	1	1558.3
planning-bnn	6	6	**27.3**	6	29.0	6	61.3	6	59.1	6	60.8	6	62.1	6	60.5
planning	15	15	**0.2**	15	1.1	15	0.3	15	0.3	15	0.3	15	0.3	15	0.3
preference	12	12	10.5	12	13.0	12	8.6	12	8.0	12	7.7	12	**7.3**	12	8.6
protein_ins	12	12	**16.7**	12	16.8	12	76.5	12	75.2	12	76.3	12	81.2	12	76.4
pseudoBoolean	15	14	**0.0**	14	**0.0**	14	0.2	14	0.2	14	0.2	14	0.2	14	0.2
qcp	15	15	**0.0**	15	**0.0**	15	0.3	15	0.3	15	0.3	15	0.3	15	0.3
quantum	15	13	299.9	13	290.9	12	246.3	13	135.1	13	**127.8**	13	151.1	13	131.3
railway	5	1	188.4	1	275.3	1	223.1	1	162.3	1	**119.1**	1	211.9	1	212.1
ramsey	4	0		1	2938.5	0		0		0		0		1	**1100.9**
relational	8	5	130.7	5	**55.3**	5	606.3	5	353.3	5	393.6	5	415.0	5	351.1
rna	15	15	**1.0**	15	**1.0**	15	3.8	15	3.8	15	3.7	15	3.8	15	3.7
setcover	6	0		0		0		0		0		0		0	
spot5	15	12	**22.0**	11	0.2	12	136.2	12	161.0	12	113.0	12	137.9	11	0.4
switching	9	0		0		0		0		1	**3184.7**	0		0	
synplicate	40	16	**47.6**	16	107.9	15	106.7	15	150.9	15	37.7	15	70.2	15	44.4
tcp	15	14	234.0	13	92.0	14	89.7	14	122.0	14	**77.7**	14	146.0	14	146.0
timetabling	13	8	136.5	8	491.4	7	73.8	8	**79.5**	8	502.4	8	481.6	7	68.5
upgradeability	15	15	0.9	15	**0.5**	15	2.0	15	2.3	15	1.9	15	2.0	15	2.0
warehouses	8	4	10.8	5	166.5	8	7.9	6	4.1	8	**7.2**	8	21.9	8	43.6
overall	571	395	153.1	393	149.1	401	148.8	392	132.4	404	147.4	404	145.2	405	**140.3**

TABLE V: Number of solved instances and average runtime in seconds on solved instances by considered solver configurations.

which provides access to its internal variables s_i^p so it may serve as an implementation of `ReifiedCard`, and then we modified RC2 to use this encoding. We refer to this RC2 configuration with sequential counter and cloning as `clone`. We then implemented CUSCUS on top of `ISeqCounter` using the method described in Sec. VI and refer to this configuration as `cuscus`.[1]

We observed that CUSCUS has the most to gain when the core contains long "stairs", i.e., subsequences of soft clauses in a core with the same weight, which can be grouped together rather than individually cloned. On the other hand, when all weights in the core are distinct, CUSCUS cannot effectively take advantage of grouping and also produces typically lower-weight clauses due to using the marginal weight instead of the difference with the minimum weight. Therefore, we developed hybrid configurations that heuristically decide the transformation method to perform on any given core. We consider the following three dynamic heuristics:

- **Homogeneity:** The homogeneity heuristic computes the *homogeneity* of a core. This is defined as the ratio $\#constraints/\#weights$ of the size of the core to number of distinct weights in the core. It can be thought of as the mean length of the "stairs" in the core. By `h`X, we denote a solver that applies the CUSCUS transformation only on cores of homogeneity at least X, and otherwise

[1]Code is available at https://github.com/jesselooney/cuscus.

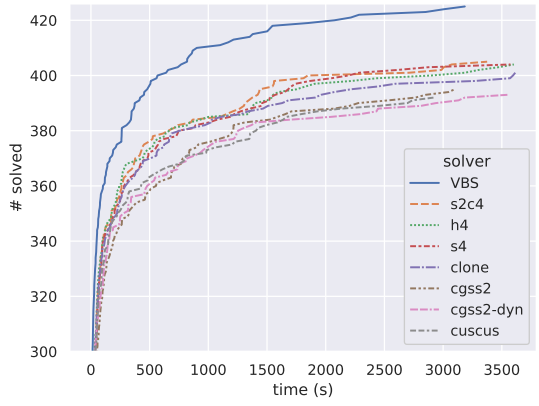

Fig. 2: Cactus plot of all configs. in Table V.

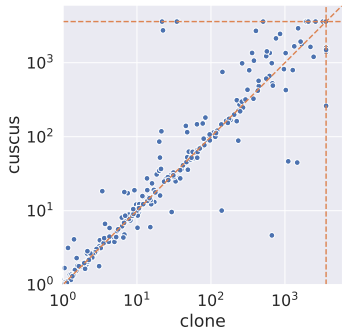

Fig. 3: Runtime of `cuscus` vs. `clone`.

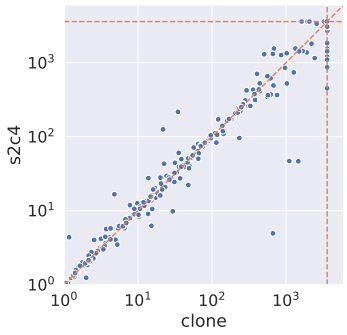

Fig. 4: Runtime of `s2c4` vs. `clone`.

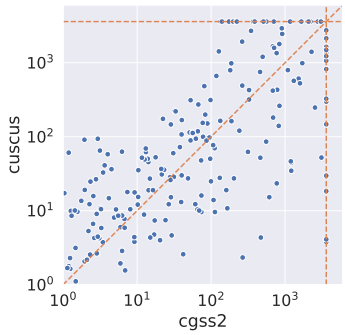

Fig. 5: Runtime of `cuscus` vs. `cgss2`.

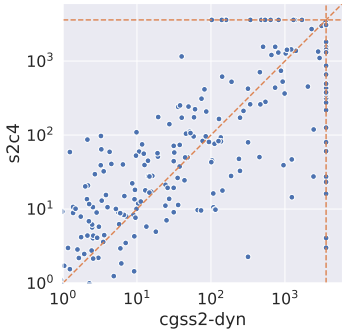

Fig. 6: Runtime of `s2c4` vs. `cgss2-dyn`.

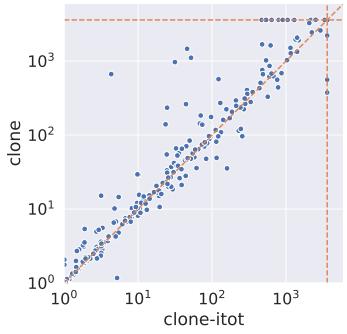

Fig. 7: Runtime of `clone` vs. `clone-itot`.

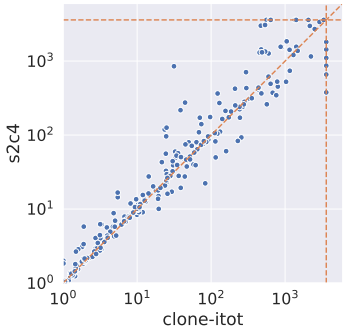

Fig. 8: Runtime of `s2c4` vs. `clone-itot`.

applies cloning.

- **Stair size:** Another heuristic decides whether to apply CusCus to individual stairs, rather than the whole core. To perform a partial CusCus transformation, we first clone until only the desired weights are present in the core, then apply CusCus. By `s`X, we denote a solver that applies CusCus only to stairs of length at least X. That is, we select weights that occur at least X times in the core and clone until only these weights (along with the minimum weight) remain before applying CusCus. For example, a core with weights $[5, 5, 5, 4, 4, 1]$ is first transformed into $[5, 5, 5, 1, 1, 1]$ (with two clones of weight 3) under the `s3` heuristic, before CusCus is applied to the two prefixes of this new core.
- **Core size:** We can improve the stair size heuristic by taking into account core size as well. By `s`X`c`Y, we denote a solver that applies the `s`X heuristic/transformation only on cores of size at least Y, and otherwise applies cloning.

Experimental setup. We ran our solvers on benchmarks from the exact weighted track of MaxSAT Evaluation 2024. These experiments were run on a computing cluster consisting of Dell PowerEdge R6525 rack servers with 2.6-GHz AMD CPUs. Each job was given a single physical core, a one-hour CPU time limit, and 8GB memory. For each of the RC2-based solvers (`clone`, `cuscus`, `h`X, `s`X, `s`X`c`Y), we used the same configuration of parameters available from the underlying implementation. We enabled intrinsic at-most-one constraint processing (`-a`), core minimization (`-m`), core exhaustion (`-x`), and diversity-based stratification (`--blo=div`), and we used Glucose 4 (`-s g4`) as the SAT oracle. We compare our approach with the original RC2 implementation, denoted by `clone-itot`, as well as AbstCG [6], [20], which is implemented in `cgss2`. We consider both the static version (`cgss2` described in Section VI) and the dynamic version (`cgss2-dyn`) of `cgss2` where the solver dynamically decides whether to apply the original clone-based transformation or AbstCG, depending on the size of the encoding.

Choice of baselines. We consider `clone` our main baseline, as it differs from `cuscus` only in the core transformation. Our main goal is to measure the effect of the transformation, so we keep the surrounding machinery fixed: core minimization, upper-bound search, and MILP preprocessing all have to be co-tuned to obtain a state-of-the-art solver. We also compared with additional baselines including `cgss2`, UWrMaxSat, and RC2, which are top-performing core-guided solvers.

Stratification/Hardening heuristics. Some care was required to make CusCus work with stratification and stratification-based constraint hardening; RC2 dynamically computes stratification levels using information about the number and weights of constraints with smaller weight than the current level. This information is also used at the end of each stratification level to harden clauses whose weights exceed the sum of weights among clauses in lower levels. When using CusCus, because the prefix constraints can have weights smaller than the minimum weight in the core (and indeed in the stratification level), it is possible that some of the deferred constraints will belong to lower stratification levels. Hence, it is necessary to track the numbers of these constraints for each weight in order to inform stratification and hardening correctly.

Experimental results. Table V shows the number of solved

instances and the average runtime on solved instances for tested solvers. For each benchmark family, we highlight the best average time among those that solved the greatest number of problems. We first compare `clone` and `cuscus`. We observe that `cuscus` solves more instances on 3 families and loses instances on 5 families. In particular, it loses 7 instances on one family `af-synthesis`. As shown in the scatter plot in Figure 3, the two core transformation procedures are highly complementary and there are a significant number of instances where `cuscus` is faster. However, `cuscus` can struggle on instances with many cores of mostly distinct weights. For example, in a core where each weight is distinct, CUSCUS will result in the same number of added clauses as simply cloning, but their weights will generally be larger, being the difference to the *next* weight rather than all the way to the minimum. This result motivated the development of hybrid configurations that try to be more selective with the application of CUSCUS, a choice that cloning does not offer, since trimming every core to its minimum weight treats all cores alike.

Table VI reports results for all instantiations of the heuristics we proposed under different parameter settings. For each parameter choice, we select the best-performing configuration by the number of solved instances and report these results in Table V. Based on the analysis in Table VI, we also conclude that `s2c4` is the best configuration, as it performed best among the heuristics we considered in terms of the number of solved instances over the whole track, with ties broken by average runtime. In particular, `s2c4` performs especially well against `clone`: on more than half of the families, it either solves more instances or the same number of instances in less time.

Figure 2 shows the cactus plot of the configurations considered in Table V. Among the individual solvers, `s2c4` shows the best performance in terms of the number of solved instances and performance across benchmark instances. We note that the virtual best solver (VBS) solves significantly more instances than the baseline. This shows that there is complementarity between the CUSCUS-based approach, `clone`, and `cgss2`. To further investigate the contribution of CUSCUS to VBS, we perform additional analysis. The VBS of `cgss2` and `clone` solves 414 instances and the VBS of all CUSCUS-based configurations solves 415, while their combination solves 425. The CUSCUS-based configurations therefore add 11 instances that neither baseline solves.

Next, we perform pair-wise comparison between solvers. We start by comparing `cuscus` and `s2c4` with `clone` (Figure 3 and Figure 4). One can see that `s2c4` performs much better than `cuscus` and outperforms `clone` on many instances. Next, we compare `cuscus` and `s2c4` with `cgss2` and `cgss2-dyn`, respectively. As can be seen from Figure 5 and Figure 6, the results show surprising complementary performance. This is interesting, as both algorithms are weight-aware, but they use different transformation procedures. Finally, we compare against the original RC2 implementation, denoted by `clone-itot`, which performs cloning but uses `ITotalizer` instead of `ISeqCounter` for cardinality constraints. Note that `clone-itot` differs from our configurations in two respects: the transformation and the cardinality encoding. Therefore, it is difficult to determine whether the improvements come from the more succinct totalizer encoding or from the transformation itself. This configuration solved 406 instances overall. Figures 7 and 8 compare the performance of `clone-itot` with `clone` and `s2c4`, respectively. Despite using a sequential counter encoding of cardinality constraints, `s2c4` performed competitively against `clone-itot` and contributed 6 unique solutions. We have also attempted a comparison with UWrMaxSAT [31], a top performer in MaxSAT Evaluation 2024. The competition configuration of UWrMaxSAT first runs a MILP solver. We turned off this feature for fair comparison. After doing so, we found that UWrMaxSAT terminated on 427 instances, but its answers were inconsistent with the known results on 24 of them.

Correctness. Finally, we validated every configuration against the known results of the MaxSAT Evaluation 2024 and further ensured that each passes the MaxSAT Regression Suite[2] [32]. We also ran MaxSAT-Fuzzer with the `Paxian`, `PaxianPyTiny`, `PaxianPyGMBO`, `Pollitt`, `Manthey` and `Soos` fuzzers on every final configuration, with answers checked against the known-robust solver EvalMaxSAT[3] [33]. We found no runtime errors or incorrect answers in any solver tested over the course of a 12-hour fuzzing session.

X. DISCUSSION AND FUTURE WORK

We proposed a novel core transformation procedure that performs a weight-aware transformation of the core and allows the lower bound on the optimal cost to be increased by more than the minimum weight. We proved its correctness and several useful properties of this transformation. We also proved that it can be implemented on top of the sequential counters encoding by reusing auxiliary variables of the encoding to form soft constraints, and we provided a prototype implementation. We showed that this procedure can be combined with clone-based approaches, leading to an improvement of the baseline solver, which was OLL with incremental sequential counters. However, there are a number of open questions to explore.

The first question is whether we can modify the totalizer encoding so that we can perform the CUSCUS procedure incrementally, similar to what we showed for sequential counters. It is not as natural as in the case of sequential counters, as totalizer encoding works with trees built on top of the relaxation variables in the core, rather than subsequences as sequential counters use. The second interesting question is to identify whether we can improve further by discovering additional properties of CUSCUS that could derive more implied constraints and identify additional cores that can be relaxed without calling the SAT solver. Finally, hybridization is also an interesting problem for further investigation, particularly in terms of how weights can be taken into account during the transformation procedure.

[2]https://github.com/tobipaxe/MaxSATRegressionSuite
[3]https://github.com/FlorentAvellaneda/EvalMaxSAT

family	#	clone		cuscus		h2		h4		h15		s2		s4		s15		s1c4		s2c4		s4c4	
abstraction	11	6	524.2	6	618.3	6	613.0	6	592.2	6	518.2	6	598.0	6	584.1	6	576.9	6	630.3	6	561.1	6	532.3
af-synthesis	15	13	1864.0	6	2071.6	7	1480.7	13	1581.9	13	1687.9	10	1736.2	12	1500.7	11	1593.7	7	1966.0	11	1642.6	12	1369.7
auctions	15	14	2.1	14	2.5	15	99.6	15	164.2	14	2.3	14	2.3	14	2.0	14	2.3	14	2.2	14	2.0	14	1.9
BTBNSL	15	0		0		1	535.9	0		0		1	2435.9	1	412.9	0		0		1	1847.9	1	315.6
causal	15	12	134.9	13	421.2	13	317.2	12	109.9	12	140.2	14	300.8	12	146.2	12	146.4	13	384.1	14	253.1	13	342.0
correlation	15	7	436.0	6	863.5	7	686.2	7	779.0	7	489.7	7	261.4	7	402.6	7	487.0	5	375.1	7	244.7	7	370.4
CSG	10	10	19.4	10	51.5	10	19.0	10	21.1	10	21.5	10	22.3	10	17.9	10	21.6	10	103.5	10	19.3	10	19.5
css	11	11	7.1	11	8.2	11	7.2	11	7.4	11	7.0	11	7.6	11	7.4	11	7.5	11	7.4	11	7.2	11	7.1
dalculus	15	15	0.4	15	0.4	15	0.4	15	0.4	15	0.4	15	0.4	15	0.4	15	0.4	15	0.4	15	0.4	15	0.4
decision	15	0		0		0		0		0		0		0		0		0		0		0	
drmx	15	15	323.1	14	90.6	14	89.7	14	99.6	15	295.7	15	335.5	15	336.7	14	104.9	14	98.7	14	92.3	15	325.3
frb	15	15	83.6	15	76.1	15	77.9	15	74.6	15	74.8	15	82.4	15	88.5	15	84.9	15	76.3	15	76.6	15	84.2
haplotyping	15	15	22.3	15	23.7	15	21.4	15	21.5	15	21.3	15	25.2	15	26.3	15	25.0	15	20.8	15	22.9	15	23.9
IMLIB	16	16	0.3	16	0.3	16	0.3	16	0.3	16	0.3	16	0.3	16	0.3	16	0.3	16	0.3	16	0.3	16	0.3
judgment	15	0		0		0		0		0		0		0		0		0		0		0	
lisbon	15	6	553.7	6	458.1	6	441.1	6	461.9	6	525.9	6	578.2	6	599.5	6	523.9	6	514.9	6	502.6	6	488.6
max-realizability	15	12	89.1	12	116.6	12	97.1	12	133.3	12	96.5	12	119.1	12	125.4	12	91.5	12	124.8	12	116.1	12	113.2
MaxSATQueries	15	13	52.3	13	40.8	13	39.3	13	40.3	13	41.5	13	42.7	13	48.1	13	64.7	13	46.2	14	250.1	13	43.9
metro	15	15	140.0	15	114.5	15	135.5	15	115.7	15	129.8	15	128.5	15	139.3	15	176.2	15	107.4	15	113.8	15	107.9
MinimumWeight	10	0		0		0		0		0		0		0		0		0		0		0	
mpe	15	12	17.2	11	278.0	12	20.2	12	17.6	12	17.6	13	141.3	13	138.8	13	183.4	11	86.3	14	297.2	13	111.7
mpmcs	15	15	2.1	15	2.1	15	2.0	15	2.1	15	2.1	15	2.1	15	2.1	15	2.1	15	2.1	15	2.1	15	2.1
ParametricRBAC	15	1	686.8	1	490.7	1	1030.6	1	694.9	1	1604.8	1	1262.5	1	890.6	1	686.0	1	1433.3	1	1558.3	1	842.9
planning-bnn	6	6	61.3	6	59.1	6	59.9	6	60.8	6	59.9	6	62.3	6	62.1	6	63.0	6	60.1	6	60.5	6	62.0
planning	15	15	0.3	15	0.3	15	0.3	15	0.3	15	0.3	15	0.3	15	0.3	15	0.3	15	0.3	15	0.3	15	0.3
preference_planning	12	12	8.6	12	8.0	12	9.1	12	7.7	12	7.8	12	9.6	12	7.3	12	7.8	12	5.6	12	8.6	12	7.4
protein_ins	12	12	76.5	12	75.2	12	76.0	12	76.3	12	75.7	12	79.6	12	81.2	12	78.3	12	81.5	12	76.4	12	82.5
pseudoBoolean	15	14	0.2	14	0.2	14	0.2	14	0.2	14	0.2	14	0.2	14	0.2	14	0.2	14	0.2	14	0.2	14	0.2
qcp	15	15	0.3	15	0.3	15	0.3	15	0.3	15	0.3	15	0.3	15	0.3	15	0.3	15	0.3	15	0.3	15	0.3
quantum	15	12	246.3	13	135.1	13	124.5	13	127.8	13	151.6	13	153.2	13	151.1	13	133.9	13	134.5	13	131.3	13	149.3
railway	5	1	223.1	1	162.3	1	226.5	1	119.1	1	223.0	1	193.3	1	211.9	1	221.1	1	163.3	1	212.1	1	217.5
ramsey	4	0		0		0		0		0		1	1222.6	0		0		1	2144.3	1	1100.9	0	
relational	8	5	606.3	5	353.3	5	391.5	5	393.6	5	469.6	5	377.2	5	415.0	5	481.9	5	395.9	5	351.1	5	426.5
rna	15	15	3.8	15	3.8	15	3.7	15	3.7	15	3.6	15	3.7	15	3.8	15	4.5	15	3.7	15	3.7	15	3.7
setcover	6	0		0		0		0		0		0		0		0		0		0		0	
spot5	15	12	136.2	12	161.0	11	0.3	12	113.0	12	129.6	11	0.3	12	137.9	12	141.5	11	0.3	11	0.4	12	119.7
switchingactivity	9	0		0		1	3266.8	1	3184.7	0		0		0		0		0		0		0	
synplicate	40	15	106.7	15	150.9	15	35.7	15	37.7	15	124.2	15	48.5	15	70.2	15	117.5	16	247.0	15	44.4	15	81.0
tcp	15	14	89.7	14	122.0	14	141.0	14	77.7	14	85.2	14	135.9	14	146.0	14	103.0	14	110.3	14	146.0	14	140.5
timetabling	13	7	73.8	8	79.5	7	65.0	8	502.4	8	328.4	7	74.0	8	481.6	7	77.0	7	59.0	7	68.5	8	486.7
upgradeability	15	15	2.0	15	2.3	15	2.0	15	1.9	15	2.0	15	2.0	15	2.0	15	2.1	15	2.1	15	2.0	15	2.0
warehouses	8	8	7.9	6	4.1	8	8.0	8	7.2	8	8.0	8	45.4	8	21.9	8	8.8	6	1.7	8	43.6	8	22.9
overall	571	401	148.8	392	132.4	398	118.8	404	147.4	403	144.7	403	142.2	404	145.2	400	129.6	392	130.3	405	140.3	405	141.1

TABLE VI: Solve times by solver and benchmark family.

REFERENCES

[1] R. Reiter, "A theory of diagnosis from first principles," *Artif. Intell.*, vol. 32, no. 1, pp. 57–95, 1987.

[2] M. H. Liffiton and K. A. Sakallah, "Algorithms for computing minimal unsatisfiable subsets of constraints," *J. Autom. Reasoning*, vol. 40, no. 1, pp. 1–33, 2008.

[3] M. H. Liffiton, A. Previti, A. Malik, and J. Marques-Silva, "Fast, flexible MUS enumeration," *Constraints*, vol. 21, no. 2, pp. 223–250, 2016.

[4] A. Morgado, F. Heras, M. Liffiton, J. Planes, and J. Marques-Silva, "Iterative and core-guided maxsat solving: A survey and assessment," *Constraints*, vol. 18, no. 4, p. 478–534, oct 2013. [Online]. Available: https://doi.org/10.1007/s10601-013-9146-2

[5] J. Alòs, C. Ansótegui, and E. Torres, "Revisiting sat-based solvers: Maxsat rules and core sequences," *J. Artif. Int. Res.*, vol. 85, Mar. 2026. [Online]. Available: https://doi.org/10.1613/jair.1.19525

[6] H. Ihalainen, J. Berg, and M. Järvisalo, "Unifying sat-based approaches to maximum satisfiability solving," *Journal of Artificial Intelligence Research*, vol. 80, pp. 931–976, 2024. [Online]. Available: https://www.jair.org/index.php/jair/article/view/15986

[7] J. Berg and M. Järvisalo, "Weight-aware core extraction in SAT-based MaxSAT solving," in *Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, 2017, pp. 652–670. [Online]. Available: https://doi.org/10.1007/978-3-319-66158-2_42

[8] C. Ansótegui, M. L. Bonet, and J. Levy, "Solving (weighted) partial maxsat through satisfiability testing," in *Proc. of the 20th Int. Conf. on Theory and Applications of Satisfiability Testing, SAT'09*, O. Kullmann, Ed., vol. 5584, Springer-Verlag. Springer-Verlag, 2009, pp. 427–440.

[9] A. Ignatiev, A. Morgado, and J. Marques-Silva, "RC2: an efficient maxsat solver," *J. Satisf. Boolean Model. Comput.*, vol. 11, no. 1, pp. 53–64, 2019. [Online]. Available: https://doi.org/10.3233/SAT190116

[10] S. Cai and Z. Lei, "Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability," *Artif. Intell.*, vol. 287, p. 103354, 2020. [Online]. Available: https://doi.org/10.1016/j.artint.2020.103354

[11] M. Piotrow, "UWrMaxSat in maxsat evaluation 2021," https://maxsat-evaluations.github.io/2021/mse21-solver-src/complete/uwrmaxsat.zip, 2021.

[12] Z. Lei, S. Cai, D. Wang, Y. Peng, F. Geng, D. Wan, Y. Deng, and P. Lu, "CASHWMaxSAT: Solver description, 2021," https://maxsat-evaluations.github.io/2021/mse21-solver-src/complete/cashmaxsat.zip, 2021.

[13] F. Avellaneda, "A short description of the solver EvalMaxSAT," https://maxsat-evaluations.github.io/2021/mse21-solver-src/complete/evalmaxsat.zip, 2021.

[14] J. Davies and F. Bacchus, "Solving MAXSAT by solving a sequence of simpler SAT instances," in *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings*, ser. Lecture Notes in Computer Science, J. H. Lee, Ed., vol. 6876. Springer, 2011, pp. 225–239. [Online]. Available: https://doi.org/10.1007/978-3-642-23786-7_19

[15] H. Ihalainen, J. Berg, and M. Järvisalo, "Refined core relaxation for core-guided maxsat solving," in *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*, ser. LIPIcs, L. D. Michel, Ed., vol. 210. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 28:1–28:19. [Online]. Available: https://doi.org/10.4230/LIPIcs.CP.2021.28

[16] J. Berg, F. Bacchus, and A. Poole, "Abstract cores in implicit hitting set maxsat solving (extended abstract)," in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, Z. Zhou, Ed. ijcai.org, 2021, pp. 4745–4749. [Online]. Available: https://doi.org/10.24963/ijcai.2021/643

[17] G. Katsirelos, "Core-Guided Linear Programming-Based Maximum Satisfiability," in *28th International Conference on Theory and Applications of Satisfiability Testing (SAT 2025)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), J. Berg and J. Nordström, Eds., vol. 341. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025, pp. 17:1–17:17. [Online]. Available: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.SAT.2025.17

[18] N. Narodytska and F. Bacchus, "Maximum satisfiability using core-guided maxsat resolution," in *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, 2014*. AAAI Press, 2014, pp. 2717–2723.

[19] Z. Fu and S. Malik, "On solving the partial MAX-SAT problem," in *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, ser. Lecture Notes in Computer Science, A. Biere and C. P. Gomes, Eds., vol. 4121. Springer, 2006, pp. 252–265. [Online]. Available: https://doi.org/10.1007/11814948_25

[20] "MaxSAT evaluation 2024: Solver and benchmark descriptions," https://helda.helsinki.fi/server/api/core/bitstreams/5066b1c4-72e6-465e-b5c4-c52a5d7d6837/content, 2024, see the section on CGSS2 and CGSS2-AbstCG.

[21] C. Sinz, "Towards an optimal cnf encoding of boolean cardinality constraints," in *Principles and Practice of Constraint Programming - CP 2005*, P. van Beek, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 827–831.

[22] M. L. Bonet, J. Levy, and F. Manyà, "Resolution for max-sat," *Artificial Intelligence*, vol. 171, no. 8, pp. 606–618, 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370207000422

[23] C. Ansótegui, M. L. Bonet, J. Gabàs, and J. Levy, "Improving sat-based weighted maxsat solvers," in *Principles and Practice of Constraint Programming*, M. Milano, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 86–101.

[24] C. Ansótegui, M. L. Bonet, and J. Levy, "A new algorithm for weighted partial maxsat," in *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, ser. AAAI'10. AAAI Press, 2010, p. 3–8.

[25] A. Morgado, A. Ignatiev, and J. Marques-Silva, "MSCG: robust core-guided maxsat solving," *J. Satisf. Boolean Model. Comput.*, vol. 9, no. 1, pp. 129–134, 2014. [Online]. Available: https://doi.org/10.3233/sat190105

[26] A. Ignatiev, A. Morgado, and J. Marques-Silva, "RC2: an efficient maxsat solver," *J. Satisf. Boolean Model. Comput.*, vol. 11, no. 1, pp. 53–64, 2019. [Online]. Available: https://doi.org/10.3233/SAT190116

[27] F. Bacchus, M. Järvisalo, and R. Martins, "Maxsat evaluation 2018: New developments and detailed results," *Journal on Satisfiability, Boolean Modelling and Computation*, vol. 11, no. 1, pp. 99–131, 2019. [Online]. Available: https://journals.sagepub.com/doi/abs/10.3233/SAT190119

[28] O. Bailleux and Y. Boufkhad, "Efficient cnf encoding of boolean cardinality constraints," in *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming*, ser. CP'03. Berlin, Heidelberg: Springer-Verlag, 2003, p. 108–122. [Online]. Available: https://doi.org/10.1007/978-3-540-45193-8_8

[29] A. Ignatiev, A. Morgado, and J. Marques-Silva, "PySAT: A Python toolkit for prototyping with SAT oracles," in *SAT*, 2018, pp. 428–437. [Online]. Available: https://doi.org/10.1007/978-3-319-94144-8_26

[30] A. Ignatiev, Z. L. Tan, and C. Karamanos, "Towards universally accessible SAT technology," in *SAT*, 2024, pp. 4:1–4:11. [Online]. Available: https://doi.org/10.4230/LIPIcs.SAT.2024.16

[31] M. Piotrów, "Uwrmaxsat: Efficient solver for maxsat and pseudo-boolean problems," in *2020 IEEE 32nd international conference on tools with artificial intelligence (ICTAI)*. IEEE, 2020, pp. 132–136.

[32] T. Paxian and A. Biere, "MaxSAT fuzzing and delta debugging," *Journal of Artificial Intelligence Research*, vol. 85, 2026. [Online]. Available: https://doi.org/10.1613/jair.1.19716

[33] F. Avellaneda, "A short description of the solver EvalMaxSAT," in *MaxSAT Evaluation 2020: Solver and Benchmark Descriptions*, 2020, p. 8.

family	h2	h4	h15	s2	s4	s15	s1c4	s2c4	s4c4
abstraction	84.3	38.7	0.0	2.2	0.6	0.4	85.3	2.2	0.6
af-synthesis	80.5	22.9	0.0	73.2	19.0	0.0	96.3	73.2	19.0
auctions	4.3	1.0	0.0	10.2	0.3	0.0	28.2	9.6	0.3
BTBNSL	1.7			14.2	10.8			15.3	10.8
causal	0.2	0.0	0.0	2.2	0.0	0.0	47.9	2.2	0.8
correlation	6.8	1.3	0.0	14.6	7.0	0.1	49.4	14.5	7.0
CSG	0.0	0.0	0.0	1.9	0.0	0.0	45.8	1.8	0.0
css	4.6	0.2	0.0	9.8	0.8	0.2	26.1	6.9	0.8
dalculus									
drmx									
frb									
haplotyping									
IMLIB	78.0	71.4	56.5	53.0	18.5	0.0	79.8	47.6	18.5
lisbon	100.0	54.9	0.0	48.3	0.0	0.0	100.0	48.3	0.0
max-realizability	50.0	0.0	0.0	100.0	0.0	0.0	50.0	0.0	0.0
MaxSATQueries	78.1	71.9	2.5	49.1	43.9	0.0	87.5	53.3	43.9
metro	75.6	31.3	0.5	66.7	29.7	5.4	88.9	64.6	29.7
mpe	0.0	0.0	0.0	9.7	0.5	0.1	59.1	10.6	0.5
mpmcs	0.0	0.0	0.0	0.0	0.0	0.0	100.0	0.0	0.0
ParametricRBAC	72.5	22.2	10.8	34.0	6.7	0.0	75.0	34.0	6.7
planning-bnn	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
planning	16.2	5.2	0.0	27.6	9.2	0.0	38.7	27.6	9.2
preference_planning	23.6	0.5	0.0	27.6	5.6	0.0	48.1	24.5	5.6
protein_ins									
pseudoBoolean									
qcp	100.0	100.0	66.7	100.0	100.0	80.0	100.0	100.0	100.0
quantum	100.0	99.1	87.1	99.1	97.4	85.8	100.0	99.1	97.4
railway	6.3	0.0	0.0	14.0	1.8	0.0	17.9	7.1	1.8
ramsey				21.7			73.8	23.2	
relational	28.7	16.0	1.8	37.4	30.5	14.8	38.7	35.5	30.5
rna	83.3	33.3	0.0	50.0	0.0	0.0	83.3	50.0	0.0
spot5	23.7	2.1	0.0	9.7	2.1	0.0	23.7	7.9	2.1
switchingactivity	100.0	50.0							
synplicate	54.6	10.3	0.9	18.9	4.0	0.0	60.5	18.5	4.0
tcp	86.6	52.0	22.8	84.9	61.5	25.3	86.6	80.1	61.5
timetabling	96.0	68.1	29.4	30.8	29.7	4.6	96.0	30.8	29.7
upgradeability	1.5	0.0	0.0	10.7	0.0	0.0	29.1	8.2	0.0
warehouses	3.7	0.7	0.0	10.2	4.3	0.0	44.9	10.2	4.3
overall	44.8	28.5	14.5	36.8	22.2	11.8	67.6	35.4	22.2

TABLE VII: Mean frequency of CUSCUS usage by solver. Empty cells indicate that either none of the benchmarks were solved or none of the solved benchmarks had any cores suitable for applying CUSCUS. Families with no solved instances are omitted to save space.

APPENDIX

Additional evaluation results.

We calculated the frequency[4] of `cuscus` application by the hybrid configurations in Table VII. Table VII provides a per-family breakdown for various hybrid configurations. For `h4`, the CUSCUS transformation was used 28.5% of the time, `s4` was used 22.2%, and for `s2c4` it was used 35.4%.

[4]This is computed over all cores that are not unit and do not have identical weights. The transformation algorithm is the same as in unweighted case for such cores.