Core-Guided Linear Programming-Based Maximum Satisfiability

George Katsirelos
Université Paris-Saclay, AgroParisTech, INRAE, UMR MIA Paris-Saclay, 91120, Palaiseau, France

Abstract

The core-guided algorithm OLL is the basis of some of the most successful algorithms for MaxSAT in recent evaluations. It works by iteratively finding cores of the formula and transforming it so that it exhibits a higher lower bound. It has recently been shown to implicitly discover cores of the original formula, as well as a compact representation of its reasoning within a linear program. In this paper, we use and extend these results to design a practical MaxSAT solver. We show an explicit linear program which matches and usually exceeds the bound computed by OLL. We show that OLL can be restated as an algorithm that explicitly computes a feasible dual solution of this linear program. This restated algorithm naturally works with an arbitrary dual solution. It can in fact be used to improve *any* LP representation of the MaxSAT instance. This presents a large increase of the potential design space for such algorithms. We describe some potential improvements from this insight and show that an implementation outperforms the state of the art algorithms on the set of instances from the latest MaxSAT evaluation.

2012 ACM Subject Classification Theory of computation → Discrete optimization; Mathematics of computing → Combinatorial optimization; Theory of computation → Logic; Mathematics of computing → Solvers

Keywords and phrases maximum satisfiability, core-guided solvers, linear programming

Digital Object Identifier 10.4230/LIPIcs.SAT.2025.17

Supplementary Material *Software (Source Code)*: `https://github.com/gkatsi/OLLLP` archived at `swh:1:dir:85dd891d9fd590dabd0fb6c2900c1877e0b17b81`

Funding Part of this work was funded by the AI Interdisciplinary Institute ANITI. ANITI is funded by the French "Investing for the Future – PIA3" program under the Grant agreement n°ANR-23-IACL-0002.

1 Introduction

MaxSAT is the optimization variant of the Boolean propositional satisfiability problem. The performance of MaxSAT solvers has increased significantly in the last few years, as evidenced by the results of recent MaxSAT evaluations. There exist two main classes of algorithms: implicit hitting set (IHS) solvers [9, 11, 10, 4, 5] and core-guided solvers [14, 3, 21, 20, 19, 15]. However, despite the improvement in performance over the last few years, there are few new algorithms. The main core-guided algorithm, OLL [1, 19], was introduced more than 10 years ago.

In this paper, we do not quite propose a completely different algorithm. We propose a new algorithm, OL^{3}P, which is quite similar to OLL. However, it uses a linear program as its main reasoning engine for bounds. The flexibility of the linear program allows us to ignore the subtleties of figuring out a correct reformulation and rely on an LP solver to do it. This means that the OL^{3}P algorithm can accommodate constraints coming from different contexts, which opens up possibilities for further algorithmic development. Specifically, our contributions are:

- We show an LP that can capture the lower bound computed by OLL. More formally, it admits a dual solution which gives the same lower bound. While the existence of this was known [17], we give here an explicit and compact LP.

- We discuss the necessary condition for using an LP to compute bounds in this algorithmic framework, namely that the LP is in augmented form.
- We discuss the subtleties and surprises of using LP reduced costs to reformulate MaxSAT instances.
- We show how to integrate a new source of constraints, by seeding the LP with clauses from the original formula.
- We integrate all these into a solver that clearly outperforms the previous state of the art of core-guided MaxSAT solving.

2 Background

A propositional formula F in conjunctive normal form (CNF) is a conjunction of clauses, where each clause c is a disjunction of literals. A literal is either a propositional variable or its negation. We write $vars(c)$ for the set of variables whose literals appear in a clause c and $vars(F)$ for the union of $vars(c)$ over all the clauses of F. Given an assignment A of variables to true or false, a clause c is satisfied by an assignment if it maps at least one of its literals to true, which we write $A \models c$. Otherwise it is falsified. The CNFSAT (or simply SAT) problem asks to determine whether there exists an assignment such that all clauses are satisfied.

The Weighted Partial MaxSAT (WPMS) problem is an extension of SAT to optimization. A WPMS instance is a tuple $\langle H, S, w\rangle$ where H is a set of *hard* clauses, S is a set of *soft* clauses and $w : S \to \mathbb{R}_{\geq 0}$ is a weight function mapping soft clauses to non-negative weights. The cost of an assignment A, written $w(A)$ is infinite if it falsifies any of the hard clauses, otherwise it is the sum of the weights of the soft clauses it falsifies: $\sum_{c \in S, A \not\models c} w(c)$. The objective of WPMS is to find an assignment whose cost is finite and minimum among all assignments.

Equivalently, we can define WPMS as a pair $\langle H, w\rangle$, where H is a set of hard clauses and w is a function $w : vars(H) \to \mathbb{R}_{\geq 0}$. Alternatively, we can write w as an n-dimensional vector, where n is the number of variables, so that we can write the WPMS problem as

$$\begin{aligned} & \min w^T x \\ s.t. & \\ & x \models H \end{aligned}$$

This restatement of WPMS is consistent with recent work [17, 16]. Moreover, it is equivalent to the formulation with soft clauses. To convert from soft clauses to this formulation, we can reify each soft clause using so-called *blocking* variables, i.e., introduce a fresh variable $b \iff c$ for each soft clause $c \in S$ and set the cost of b to be the weight of c. This is what most WPMS solvers do anyway in order to use assumption-based SAT solvers.

A core C of a WPMS is a subset of its soft clauses such that the CNF formula $C \cup H$ is unsatisfiable. In terms of WPMS solving, this means that at least one of the soft clauses in C must be satisfied. In our preferred formulation of WPMS, where an instance is $\langle H, w\rangle$, a core is a set of literals such that at least one of these literals must be true in any feasible solution, i.e., any solution that satisfies H. In other words, a core is a positive clause over blocking variables.

As we shall see later, it is useful to ignore the correspondence between literals of blocking variables and soft clauses. Indeed, the "cores" extracted by OL^{3}P need not contain exclusively positive literals of blocking variables. In fact, both of these conditions are violated: the clauses extracted by OL^{3}P may contain literals of any sign; moreover, those may be literals

of blocking variables, of sum variables [1], and even of variables of the original formula that are not blocking variables. Despite the difference with standard terminology, we use the term core throughout, as it is the main piece of information given by the SAT solver to the MaxSAT algorithm. We emphasize however, that in our setting a core is simply a clause or, equivalently, a linear inequality.

2.1 OLL

OLL is an algorithm for solving MaxSAT that belongs to a class of iterative SAT-based algorithms known as core-guided. It was introduced by Andres et al. [1] in the context of ASP and adapted to MaxSAT by Morgado et al. [19]. Its pseudocode is shown in Algorithm 1.

OLL optimistically tries to find a solution with 0 cost at each iteration, by asking a SAT solver to solve the CNF instance that results from setting all literals in the objective function to false (line 5). If that succeeds, this is optimal and it returns in line 7. If it fails, it extracts a core from that formula and uses it to increase the lower bound by c^i, the minimum cost of any variable in the core (line 8). It then reformulates the instance in line 9 so that each feasible assignment of F has its cost reduced by exactly c^i in F^{i+1}. The reformulation involves introducing new variables in the objective function, modifying the costs of existing variables, as well as building a subformula f^i_{OLL} which logically defines the new variables. As long as the reformulation step is correct, OLL increases the lower bound at each iteration, hence it is guaranteed to eventually reach the true optimum and terminate.

In order to avoid confusion between cores of the original formula and cores of the reformulated formulas - which can be different - we follow [22] and refer to cores of $H \cup f^i_{OLL}$ as metas.

Algorithm 1 Core-guided MaxSAT.

```
1 Procedure Core-Guided(F = ⟨H, w⟩)
2     lb = 0
3     f^0_OLL = ∅
4     for iteration i = 1, ... do
5         (a, m^i) = solve((H ∪ f^{i-1}_OLL) |_{w=0})
6         if a ≠ ∅ then
7             return a
8         lb ← lb + min{w^{i-1}(j) | x_j ∈ m^i}
9         (f^i_OLL, w^i) = Reformulate(f^{i-1}_OLL, w^{i-1}, m^i)
```

We complete the description of OLL with a description of the reformulation step. Let the meta at iteration i be $m^i = \{x^i_1, x^i_2, \ldots, x^i_{k^i}\}$. By the definition of $F^i|_{w=0}$, all literals that appear in m^i have positive cost. We abuse notation and write $w^{i-1}(m^i) = \min_{x \in m^i} w^{i-1}(x)$ for the minimum cost among the literals that appear in m^i. OLL increases the lower bound by $w^{i-1}(m^i)$ and introduces *sum variables*, defined as $o^i_j \iff \sum_{x \in m^i} \geq j$ for $2 \leq j \leq k^i$. Finally, it reduces by $w^{i-1}(m^i)$ the cost of all literals in m^i and sets to $w^{i-1}(m^i)$ the cost of the positive literals of all o^i_j. Formally,

[1] Sum variables are introduced by OLL and will be explained in section 2.1

$$w^i(x) = \begin{cases} w^{i-1}(x) - w^{i-1}(m^i) & \text{if } x \in m^i \\ w^{i-1}(m^i) & \text{if } x \in \{o^i_2, \ldots, o^i_{k^i}\} \\ w^{i-1}(x) & \text{otherwise} \end{cases}$$

To see why this is a reformulation, observe that we can combine the fact that $\sum_{x \in m^i} x \geq 1$ with the definition of the sum variables to get

$$\sum_{x \in m^i} x \quad = \quad 1 + \sum_{j=2}^{k^i} o^i_j \ , \tag{1}$$

because when d of the k^i variables in m^i are true, exactly the $d-1$ sum variables $o^i_2, \ldots, o^i_d$ will be true, by their definition. To complete the reformulation, we multiply equation (1) by $w^{i-1}(m^i)$ and replace its left-hand side in w^{i-1} by its right hand side, which gives w^i. Note that this differs from the usual proof of correctness of OLL, but is useful to better understand the new algorithm OL^{3}P.

2.2 Linear programming

Linear programs are problems of the form $\min c^T x : Ax = b \wedge x \in \mathbb{R}^n_{\geq 0}$, where x is a vector of n real (rational) valued variables, $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m \times n}$. The problem can be solved in polynomial time. We restrict our attention here to the case where $x \in [0,1]^n$.

This form of linear programs, where all constraints are equalities, is called the augmented form. It is possible to convert any linear program which includes inequalities by introducing *slack variables*. For example the constraint $a^T x \geq b$ becomes $[a \ -1]^T [xs] = b$, where s is a fresh variable and is the slack variable of this constraint.

For each linear program P (called the *primal*), we can define a *dual* problem $D(P)$, which is $\max b^T y : A^T y \leq c \wedge y \in \mathbb{R}^m$, where y is a vector of m real variables, called the dual variables or *dual multipliers*. An important theorem in linear programming is *strong duality*: the optima of P and $D(P)$ are exactly the same. In fact, for every feasible solution y of $D(P)$, its cost is a lower bound on the optimum of P.

There exists a correspondence between variables of the primal and constraints of the dual. Indeed, the i^{th} column of P contains the coefficients of the i^{th} primal variable and also describes the i^{th} dual constraint. From each dual solution y, we can define the *reduced cost* of each primal variable, $rc^y(x)$. This is the slack of the corresponding dual constraint, i.e., $c_i - A_i^T y$. The theorem of *complementary slackness* states that for every pair of optimal solutions x, y, it holds $x_i(c_i - A_i^T y) = 0$, i.e., if the reduced cost of x is non-zero, then x must be 0 in the primal and vice versa.

Reduced costs are used typically for *reduced cost fixing* when the linear program is a relaxation of a discrete optimization problem. For a given dual solution y, $c^T y + c_i - A_i^T y$ is a lower bound on the cost of any solution that assigns x_i a non-zero value. If that matches or exceeds the cost of an incumbent solution of the discrete problem, we can fix x_i to 0. By convention, if a variable is fixed to its 1 in the primal, its reduced cost is non-positive and $c^T y - c_i + A_i^T y$ is a lower bound on the cost of any solution that assigns x_i less than 1 [2].

[2] Negative reduced costs are not true reduced costs. The would imply negative slack of the corresponding dual constraint, which means a non-feasible dual solution. Instead, they are the negated reduced cost of a slack variable and LP solvers use this convention to avoid exposing this slack variable.

In the case of augmented form linear programs, we can make a stronger statement on reduced costs: for any feasible y, the function $b^T y + (rc^y)^T x$ is equal to $c^T x$ in every feasible point of the primal. We can therefore reformulate P using the reduced costs. We give an elementary proof of this fact below.

▶ **Theorem 1.** *Let P be a linear program $\min c^T x : Ax = b$. For a given dual feasible solution y, let $P' = \min b^T y + (rc^y)^T x : Ax = b$. Then $P = P'$, i.e., they have identical sets of feasible solutions and each feasible solution has the same cost in P and P'.*

Proof. Since the constraints of both problems are identical, we only need to show that the objectives are identical, subject only to $Ax = b$.

$$b^T y + (rc^y)^T x = b^T y + (c^T - A^T y)^T x = c^T x + b^T y - y^T Ax \tag{2}$$

Finally, since $Ax = b$, we have $y^T Ax = y^T b = b^T y$, so $b^T y + (rc^y)^T x = c^T x$, as required. ◀

▶ **Example 2.** Consider the following pair of primal and dual LPs:

(Primal)	(Dual)
$\min x_1 + x_2$	$\max y$
s.t.	
$x_1 + x_2 \geq 1$	$y \geq 0$
$x_1 \geq 0$	$y \leq 1$
$x_2 \geq 0$	$y \leq 1$

The optimum of this is 1, witnessed by the primal solution $x_1 = 1$, $x_2 = 0$ and the dual solution $y = 1$. The reduced cost of both x_1 and x_2 under this dual solution is 0, since the corresponding dual constraint for both is $y \leq 1$, which has slack 0. Suppose that we try to use reduced costs as described in theorem 1. This gives the objective function $\min 1$, which is constant. This is not equal to the objective $\min x_1 + x_2$ at all points. For example, the feasible primal assignment $x_1 = x_2 = 1$ has cost 2 under the original objective. Therefore, the scheme of theorem 1 is not a reformulation scheme for LPs not in augmented form.

Suppose now that we convert this LP to augmented form by adding the slack variable s in the sole constraint:

(Primal)	(Dual)
$\min x_1 + x_2$	$\max y$
s.t.	
$x_1 + x_2 - s = 1$	
$x_1 \geq 0$	$y \leq 1$
$x_2 \geq 0$	$y \leq 1$
$s \geq 0$	$-y \leq 0$

This has the same optimum, witnessed by the primal solution $x_1 = 1, x_2 = 0, s = 0$ and the dual solution $y = 1$. The reduced costs of x_1 and x_2 are still 0, but the reduced cost of s is 1. Therefore the scheme of theorem 1 gives the objective $\min 1 + s$, which we can confirm preserves the costs of all feasible primal assignments. Indeed, the assignment $x_1 = x_2 = 1$ above corresponds to the assignment with $x_1 = x_2 = s = 1$ in the augmented LP. This evaluates to 2 under the new objective, as it does under the original objective. ⌟

3 OL^{3}P

In this section, we describe OL^{3}P. First, we note that it has been previously shown [17] that at iteration i of a run of OLL, we can construct an LP from the metas $m^1, \ldots, m^i$, which is logically equivalent to f^i_{OLL} and whose optimum is at least the lower bound computed by OLL. However, that LP is only given implicitly, as the local polytope of a weighted CSP with desirable properties. Here, we give an explicit construction for such an LP. It has the same size asymptotically - linear in the size of the meta - but it is smaller. A similar construction has been proposed by Berg et al. in the context of proof logging for OLL [6].

A crucial requirement is for the LP to be in augmented form, i.e., with equalities only, so that dual solutions give reformulations. Our starting point is equation (1):

$$LP_{OLL}(m^i) \equiv \sum_{x \in m^i} x = 1 + \sum_{j=2}^{k^i} o^i_j \tag{3}$$

$$\cup \quad o^i_j - o^i_{j+1} - e^i_j = 0 \qquad \forall j \in [2, k^i - 1] \tag{4}$$

$$\cup \quad e^i_{k^i} = o^i_{k^i} \tag{5}$$

We name y^i_{sum} the dual multipliers corresponding to constraint (3) for m^i.

▶ **Theorem 3.** *For any execution of* OLL*, at its i^{th} iteration, there exists an LP which (a) has size linear in f^i_{OLL}, (b) is logically equivalent to f^i_{OLL}, (c) has optimum that is at least as great as the lower bound computed by* OLL*, (d) admits a dual feasible solution such that the reduced costs match the objective function computed by* OLL*.*

Proof. The LP is given by constraints (3)–(5).

(a) is obvious, as the total number of non-zeros is $4k_i - 3$: each of the k_i literals of m^i appears once, each of the $k^i - 1$ sum variables appears twice, and each of the $k^i - 1$ equality variables appears once.

(b) For (3), the fact that it follows from f^i_{OLL} was explained in section 2. The constraints (4)–(5) encode that $o^i_j \geq o^i_{j+1}$. The variable e^i_j is a slack variable that also encodes the fact that $\sum_{x \in m^i} x = j$. Indeed, if $e^i_j = 1$ then $o^i_j = 1$ and $o^i_{j+1} = 0$, so the sum is exactly j. If $e^i_j = 0$, then $o^i_j = o^i_{j+1}$, so the sum is either strictly less or strictly greater than j. The last constraint (5) exists only for uniformity. The proof that these entail the SAT encoding of the sum constraints is similar.

(c), (d). Set the dual variable y^i_{sum} for each iteration i to $w^{i-1}(m^i)$ and every other dual variable to 0. We show that this has the required properties by induction. For the first meta, m^1, $w^0(m^1)$ is at most equal to $w^0(x)$ for the original variables $x \in m^1$ and they appear only in a single new constraint, constraint (3) for m^1. Hence the dual constraint of each variable $x \in m^1$ is $y^1_{sum} \leq w^0(x)$, which is satisfied. Moreover, its reduced cost under this dual assignment is $w^0(x) - w^0(m_i)$. For the sum variables the dual constraint is $-y^1_{sum} \leq w^0(x) \leq 0$, which is satisfied since $w^0(m_i)$ is positive and the reduced cost is exactly $w^0(m_i)$, as required.

For the inductive step, at iteration i observe that the only change made to the dual with the addition of the new constraint is that its dual variable y^i_{sum} is added to the dual constraint of each $x \in m^i$. By the inductive hypothesis, these constraints have slack equal to the cost assigned to the corresponding primal variables by OLL. Therefore, by setting $y^i_{sum} = w^{i-1}(m^i)$, we see with the same reasoning as above that all dual constraints are satisfied and the reduced costs are set as OLL. ◀

▶ **Example 4.** Suppose that OLL runs on an instance with objective function

$$\min 5x_1 + 4x_2 + 3x_3 + 2x_4 + x_5$$

and that it finds the core $x_1 + x_2 + x_3 \geq 1$ at iteration 1 and the meta $o_2^1 + x_2 + x_4 \geq 1$ at iteration 2. Then the LP after iteration 1 is

(Primal)	(Dual)
$\min 5x_1 + 4x_2 + 3x_3 + 2x_2 + x_5$	$\max y^1_{sum}$
s.t.	
$x_1 + x_2 + x_3 - o_2^1 - o_3^1 = 1$	
$o_2^1 - o_3^1 - e_2^1 = 0$	
$x_1 \geq 0$	$y^1_{sum} \leq 5$
$x_2 \geq 0$	$y^1_{sum} \leq 4$
$x_3 \geq 0$	$y^1_{sum} \leq 3$
$o_2^1 \geq 0$	$-y^1_{sum} \leq 0$
$o_3^1 \geq 0$	$-y^1_{sum} \leq 0$

And after iteration 2 it will be

(Primal)	(Dual)
$\min 5x_1 + 4x_2 + 3x_3 + 2x_2 + x_5$	$\max y^1_{sum} + y^2_{sum}$
s.t.	
$x_1 + x_2 + x_3 - o_2^1 - o_3^1 = 1$	
$o_2^1 + x_2 + x_4 - o_2^2 - o_3^2 = 1$	
$o_2^1 - o_3^1 - e_2^1 = 0$	
$o_2^2 - o_3^2 - e_2^2 = 0$	
$x_1 \geq 0$	$y^1_{sum} \leq 5$
$x_2 \geq 0$	$y^1_{sum} + y^2_{sum} \leq 4$
$x_3 \geq 0$	$y^1_{sum} \leq 3$
$o_2^1 \geq 0$	$-y^1_{sum} + y^2_{sum} \leq 0$
$o_3^1 \geq 0$	$-y^1_{sum} \leq 0$
$o_2^2 \geq 0$	$-y^2_{sum} \leq 0$
$o_3^2 \geq 0$	$-y^2_{sum} \leq 0$

All variables in the primal are non-negative, but we only show above those that have a corresponding dual constraint. In both cases, we have in the same line a primal constraint and the bounds of its dual variable (unbounded here because the primal constraints are equalities), and the bounds of primal variable with the corresponding dual constraints. We can confirm that the dual solution $y^1_{sum} = 3, y^2_{sum} = 1$ is feasible and the reduced costs of the variables match what is computed by OLL. ⌟

One consequence of theorem 3 and theorem 1 is that we can change the reformulation step of OLL so that it uses an optimal dual solution to compute a reformulation, rather than that which is statically computed. Indeed, the reformulation step can be discarded altogether and the algorithm can use reduced costs directly.

Algorithm 2 OL^{3}P.

```
1  Procedure OL^3P (F = ⟨H, w⟩)
2      lb = 0
3      f^0_OLL = ∅
4      LP^0 = min(w^0)^T x
5      for iteration i = 1, ... do
6          (a, m^i) = solve((H ∪ f^{i-1}_OLL) |_{rc=0})
7          if a ≠ ∅ then
8              return a
9          f^i_OLL = f^{i-1}_OLL ∪ CNF(o^i_j ⟺ Σ_{x∈m^i} ≥ j) ∪ CNF(e^i_j ⟺ Σ_{x∈m^i} = j)
10         LP^i = LP^{i-1} ∪ LP_OLL(m^i)
11         lb = opt(LP^i)
```

We show the pseudocode for this algorithm, which we call OL^{3}P, in Algorithm 2. This follows the outline of OLL. It starts with lower bound zero and initializes the auxiliary formula f^0_{OLL} and corresponding LP LP^0 to the empty formula. Then, in each iteration of the loop in lines 5-11, it tests satisfiability of the formula $(H \cup f^{i-1}_{OLL})\,|_{rc=0}$ in line 6. This formula sets to false all variables which have non-zero reduced cost. In the first iteration, models of this formula correspond to feasible assignments of $\langle H, w\rangle$ that have cost 0 overall. In subsequent iterations, they correspond to assignments whose cost matches the current lower bound. If that formula is satisfiable, it reports optimality and exits in line 8. Otherwise, the SAT solver gives a meta m^i. In line 9, it adds to the subformula f^{i+1}_{OLL} the standard OLL constraints which introduce variables o^i_j to count how many variables of m^i are true, so that o^i_j is true if at least j variables of m^i are true. It additionally introduces the variables required by OL^{3}P, where e^i_j is true if exactly j variables of m^i are true. It also adds the corresponding linear constraints (3)-(5) to the LP in line 10. In constrast to OLL, it does not update the lower bound using the minimum cost of variables in m^i, nor does it update variable costs. Instead, it resolves the LP at line 11 to get an updated bound, as well as updated reduced costs for the next iteration.

▶ **Theorem 5.** OL^{3}P *is sound and complete*

Proof. For soundness, note that lower bounds are derived from the solution of an LP, which is itself built using constraints given by a SAT solver. Hence, all lower bounds are correct.

For completeness, note that since the LP is in augmented form, reduced costs define an equivalent objective. Hence, if the SAT solver determines that the instance $(H \cup f^{i-1}_{OLL})\,|_{rc=0}$ is satisfiable, i.e., it is satisfiable using only literals with 0 reduced cost, the cost of that assignment is exactly the optimum of the LP. Finally, since the lower bound increases at each iteration and the optimum is finite, the algorithm eventually exits with an optimum solution. ◀

While the transition from precomputed reformulation to one driven by LP dual solutions is straightforward, we need to highlight some subtlety in the notation. The instance given to the SAT solver is $(H \cup f^{i-1}_{OLL})\,|_{rc=0}$, which does not require that literals are positive. If a variable is fixed to 1 and has negative reduced cost, it means that it must be true in solutions. While this is not generally problematic with our formulation of MaxSAT, it is different when solving MaxSAT instances in the WCNF format. Specifically, if this happens

for a blocking variable, we require that the clause be falsified, which means that we must encode full reification, not just $\neg c \implies b$, as is typically done to improve performance. Similarly, if this happens for a sum variable o^i_j, it is required that the sum is *at least* j.

▶ **Example 6.** Suppose OL^{3}P is working on an instance with objective

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

and it has discovered the constraint $x_1 + x_2 + x_3 \geq 2$. The usual OLL reformulation would use this constraint to increase the lower bound to 4. However, we can see that there is no way to make two of these variables true and pay any less than 5. Indeed, the LP solver will find a dual solution in the reduced cost of x_1 is -2. This means that x_1 must be true. If it is false, the true variables must be x_2 and x_3, which have joint cost 7.

Now suppose that x_1, x_2, x_3 are blocking variables for soft clauses. If we set x_1 to false but the clause is not fully reified, the SAT solver can discover a solution that satisfies the clause of x_1, nevertheless sets x_1 to true, and its cost is higher than the bound computed by the LP. It will then terminate with LB $<$ UB, which is erroneous behavior. ⌟

The encoding used by many OLL-based solvers, totalizers [15, 23], is not fully reified. Instead, we have to use a different encoding, which can be made to encode full reification, in this case an encoding based on sorting networks [12]. A feature of the totalizer encoding is that it is incremental. For each meta m^i, only the variable o^i_2 needs to be introduced immediately, because any meta that contains o^i_3 can be transformed into one that contains o^i_2, as long as both have non-zero reduced cost. Therefore, practical implementations of OLL only encode o^i_{j+1} when the cost of o^i_j becomes zero. Sorting networks do not have this feature. This is a drawback of our approach, because it means that the SAT solver needs to solve a more complicated formula earlier than it would if it was possible to use an incremental encoding.

3.1 Seeding the LP

MaxHS [9], an implicit hitting set solver, which encodes the optimization part of MaxSAT as an integer linear program (ILP), makes extensive use of *seeding*. Given a MaxSAT formula $F = \langle H, w \rangle$, before it enters the main loop, MaxHS looks for clauses $c \in H$ which contain only literals of variables that appear in the objective. Then, it adds c to the ILP representation of the problem it solves. In addition, when there is a small number of additional variables that must be added to the ILP in order to give it some clauses, MaxHS adds those variables and corresponding clauses. While this may make the ILP larger and hence more difficult to solve, it can also provide useful information which improves both the bounds and the efficiency of the ILP solver.

We can do the same with OL^{3}P. However, where MaxHS can simply add clauses to the ILP problem it maintains, we are restricted to having LPs in augmented form. Hence, when we find a clause c that is suitable for seeding, we add $LP_{OLL}(c)$ to LP^0. The problem with this is that, while in MaxHS adding a clause to the ILP is just a single constraint, in OL^{3}P we need to add 4 times as many non-zeros and introduce $2|c| - 1$ new variables. This encoding has to be mirrored on the SAT side. However, only a relatively small number of constraints get a non-zero dual multiplier. Even if the LP and SAT solvers can determine that all the extra variables that appear in constraints with 0 dual multiplier are redundant, it still increases preprocessing time.

We can do better than that by reconsidering the proof of theorem 1. We can see that it is not necessary for the LP to have all constraints in the form $Ax = b$. Indeed, suppose that an LP has both equality constraints $Ax = b$ and inequality constraints $A'x \geq b'$ and suppose that the dual multiplier of every constraint in A' is 0. Then we can rewrite equation (2) as follows:

$$b^T y + (rc^y)^T x = b^T y + (c^T - A^T y - A'^T y)^T x \tag{6}$$

But since y assigns 0 to the multiplier of every constraint in A', we have $A'^T y = 0$ and the proof proceeds as before.

This allows us to make a simple but important optimization in seeding: when we initially seed the LP, we can leave all clauses as inequality constraints. This reduces the total number of variables in the LP and removes the need to add the corresponding sum constraints on the SAT side. When we actually solve an LP and find an inequality with non-zero dual multiplier, we can convert it to an equality, add all the necessary slack variables and corresponding sum constraint on the SAT side, then reuse the same dual solution to continue execution. But the number of constraints with non-zero dual multiplier is typically much smaller than the entire set of constraints. For instances where we can seed many or even all the initial clauses of the problem, this technique can represent significant savings.

3.2 Combining with implicit hitting set

Algorithm 2 is presented as extracting reduced cost-based metas at each iteration. As our argument in section 3.1 showed, however, the correctness of the algorithm does not depend in any way on the specific type of constraints. To this end, recall that when viewed as an ILP, LP_{OLL} encodes a hitting set problem over a set of cores derived from the metas discovered so far [17]. Hence, if we solve this minimum hitting set problem, we get a new source of cores. These cores are easier to discover than the metas discovered by the main OL^{3}P loop.

Unfortunately, in early experiments, CPLEX proved to be very poor at solving ILP_{OLL}, the integer version of LP_{OLL}. Even in cases where the optimum of the LP matches the optimum of the ILP, CPLEX struggles to find the optimum solution. Exploring this direction further remains future work.

3.3 Reduced cost fixing

As mentioned earlier, reduced cost fixing is a technique that is widely used in ILP solving, as well as in constraint programming [13] and in MaxHS [4]. In the literature of core-guided solvers, the closest analogue is hardening [15]. In particular, if we use exactly the potentially suboptimal dual solutions described previously [17] that simulate OLL, reduced cost fixing is exactly equivalent to hardening. Here, we use an optimal solution, which means we can potentially fix more variables. Note, however, that finding optimal reduced costs is itself a non-trivial problem [8].

There are two considerations for using reduced cost fixing with OL^{3}P. First, note that a practical implementation of OL^{3}P must implement techniques like stratification and weight-aware core extraction (covered in later sections). These split the execution of the solver into distinct phases of core (meta) extraction, each of which finishes with a satisfiable instance, generating a solution, hence an upper bound. This enables reduced cost fixing in the first place. However, it is now no longer necessary that the execution of the solver finishes with a SAT instance.

▶ **Example 7.** Suppose that a variable x always appears positively in every optimal solution of an instance F. Suppose further that we have already found an optimal solution of F and that the reduced cost of x is equal to its actual marginal cost. Then, x will be set to false by reduced cost fixing. There are three possible outcomes from this: either the LP reports a lower bound that exceeds the upper bound, the LP is infeasible, or the SAT solver reports an empty conflict. While seemingly erroneous, these results are correct. We are looking for a solution that improves on the optimum and we have pruned a literal that appears in all optimum solutions. The system is unsatisfiable, so any inference is correct. ⌟

A second consideration is that, knowing the specific relationship among the variables $o^i_2, \ldots, o^i_{k^i}$, we can compute better reduced costs. Consider o^i_3 for some i. When that is true, it implies also o^i_2, therefore we incur the cost of the reduced cost of both o^i_2 and o^i_3. In other words, the true marginal cost of o^i_3 is $mc(o^i_3) \geq rc^y(o^i_2) + rc^y(o^i_3)$. This generalizes to all indices of a sum. In preliminary experiments, we found that this technique can be very effective in fixing a large number of sum variables, but ends up hindering the performance of the SAT solver. We therefore do not use this technique in section 4.

3.4 Stratification

Stratification [15, 2] is a technique for improving solving of instances with diverse sets of costs. It splits variables of the objective into buckets, such that the higher the bucket index, the smaller the cost of the variable. It then solves the instance ignoring all but the first bucket of variables, i.e., treating all the other variables as if they have reduced cost 0. When that finishes, it adds variables from the next bucket and so on.

Stratification is that much more important for OL^{3}P. The dual solution is not guaranteed to yield integer reduced costs and in fact there typically exist many variables with very small reduced cost. Without stratification, OL^{3}P will be left to find metas with very small cost and make slow progress towards the optimum.

In addition, we implement rank-based stratification. We define the *rank* of an original variable of the instance to be $rank(x) = 0$. For all sum or equality variables corresponding to meta m^i, let r be the maximum rank among all $x \in m_i$. Then we set $rank(o^i_j) = rank(e^i_j) = r + 1$ for $j \in [2, k^i]$. For each cost-based bucket, we first extract all metas involving variables of rank 0, then 1, and so on. Note that metas of rank 0 are cores of the original formula.

The reasoning for this heuristic is that the higher the rank of the variables in a core, the harder it is for the SAT solver to discover, hence we want to extract the easier cores first.

4 Experimental evaluation

In this section, we describe the implementation details of a practical implementation of OL^{3}P and evaluate its performance.

4.1 Experimental Setup

We implemented OL^{3}P as an alternative algorithm on top of MaxHS [3]. We evaluate two variants. The first, OL^{3}P, implements the base algorithm OL^{3}P shown in Algorithm 2, with rank-based stratification, reduced cost fixing, weight aware cost extraction, and seeding as described in section 3.1. The second, OL^{3}P-S is the same but performs no seeding. This variant is the closest to base OLL, as the only major difference is that it uses LP for

[3] https://github.com/gkatsi/OLLLP

reformulation. Both variants use IBM CPLEX version 22.1.1.0 to solve the LP and CaDiCaL version "sc-2021" as a SAT solver. The ILP solver of CPLEX is not used. Additionally, both variants use the MaxPRE preprocessor [18].

We compare against CashWMaxSAT version "DisjCom-S6" [23], the winner of the 2024 MaxSAT evaluation in the weighted track. By default, CashWMaxSAT uses a portfolio strategy whereby, before launching OLL, it gives an instance to the SCIP ILP solver [7] and lets it run for some time. Only if SCIP does not solve the instance, does it execute the OLL algorithm. On the other hand, it does not use MaxPRE by default. We tested four configurations of CashWMaxSAT, varying the use of this feature and the use of MaxPRE:

1. As run in the 2024 evaluation, where the SCIP timeout is 600 seconds
2. As above, but with the SCIP timeout set to 300 seconds to match the lower timeout we used (see below)
3. With SCIP disabled, but using MaxPRE
4. Without either SCIP or MaxPRE

The latter two are more directly comparable to OL^{3}P-S, as they are not portfolios. They all use CaDiCaL version 2.0.0 as a SAT solver.

Finally, we compared against MaxHS, in its configuration from the 2022 MaxSAT evaluation, which is the last time it participated, using IBM CPLEX version 22.1.1.0 to solve the ILP and CaDiCaL version "sc-2021" as a SAT solver. MaxHS is also a portfolio solver. When the number of soft clauses is relatively small, it launches LSU instead. When the problem is small or the number of variables that do not appear in soft clauses is small, it delegates to CPLEX.

We compare the solvers on the instances from the weighted track of the 2024 MaxSAT evaluation. We ran all solvers for 1800 seconds of CPU time on a cluster of 14-core Intel Xeon E5-2695 v3 CPUs running at 2.30GHz.

4.2 Results

Surprisingly, the four different configurations of CashWMaxSAT varied little in their performance, solving 405, 409, 410, and 406 instances, respectively. It is possible that the shorter timeout compared to the evaluation made the portfolio ineffective. We retained configurations 2 and 3, as those exhibiting the best performance.

We show in Figure 1 a plot of the relative performance of the four solvers. Comparing only the core-guided solvers, we see that even though the solvers end at similar numbers of solved instances, both variants of OL^{3}P-S are better than CashWMaxSAT. We give a more detailed breakdown of the number of instances solved by family in Table 1. We see that most of the gains of OL^{3}P come from the JUDGEMENT-AGGREGATION family. There, the seeded LP computes a near optimum bound, then it takes just a few iterations to find the optimum solution and prove optimality. In OL^{3}P-S and CashWMaxSAT, this information is not available and they make slow progress, ending with a significant optimality gap. In the rest of the families, OL^{3}P sometimes exhibits a small advantage, but not consistently.

We also give in Table 2 the average time needed to solve instances (ignoring instances that timed out), as well as the PAR2 score of each solver (the sum of the runtimes over all instances, with instances that timed out or ran out of memory counting as twice the timeout, hence 3600 here). We see that both OL^{3}P-S and OL^{3}P have significantly reduced average solving time and lower PAR2 compared to CashWMaxSAT. This means that, not only does OL^{3}P solve more instances, it does so more quickly on average. The CashWMaxSAT +SCIP configuration does particularly poorly in these metrics because of the sequential portfolio it uses: for every instance that SCIP is unable to solve, it pays a penalty of 300 seconds. These

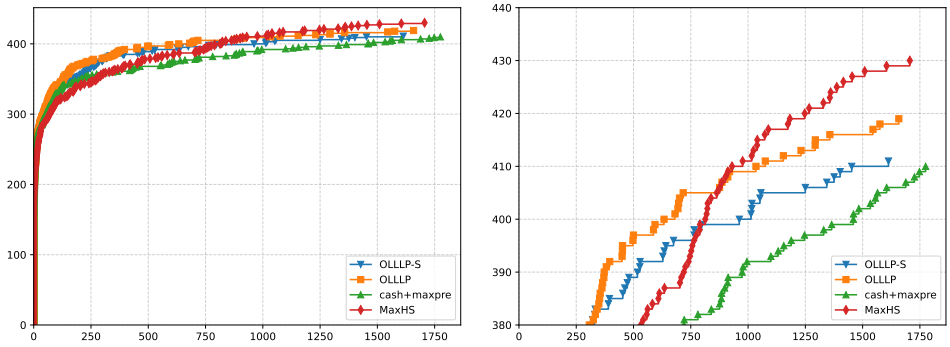

Figure 1 Results on exact weighted track of 2024 MaxSAT evaluation. On the right, zoomed in on the y axis so that the differences are highlighted.

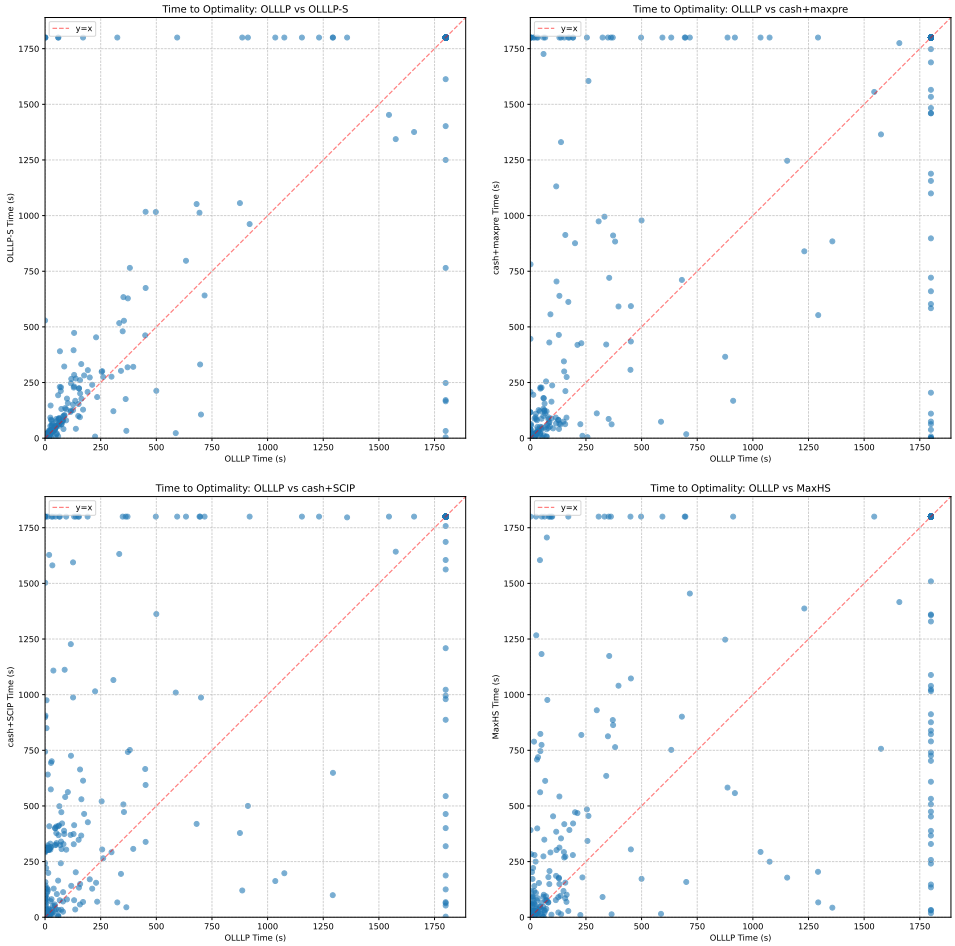

Figure 2 Scatter plots comparing OL^{3}P against the other three solvers. Dots over the $y = x$ line indicate that OL^{3}P was faster.

Table 1 Number of instances solved by each solver per family.

Family	#	Cash	Cash +SCIP	MaxHS	OL^{3}P-S	OL^{3}P
abstraction-refinement	11	11	10	11	11	11
af-synthesis	15	8	4	12	3	4
auctions	15	14	15	15	14	14
BTBNSL	15	3	2	7	3	3
causal-discovery	15	14	14	12	14	14
correlation-clustering	15	9	9	9	9	9
CSG	10	10	10	10	10	10
css-refactoring	11	11	11	8	11	11
dalculus	15	15	13	15	15	15
decision-tree	15	0	0	0	0	0
drmx-cryptogen	15	15	11	15	15	13
frb	15	14	11	14	15	15
haplotyping-pedigrees	15	15	15	15	15	15
IMLIB	16	16	16	16	16	16
judgment-aggregation	15	1	12	14	0	10
lisbon-wedding	15	3	3	1	3	4
max-realizability	15	14	14	13	13	12
MaxSATQueriesinInterpretableClassifiers	15	14	14	13	13	13
metro	15	15	15	15	15	15
MinimumWeightDominatingSetProblem	10	0	0	2	0	0
mpe	15	13	13	13	14	12
mpmcs	15	15	15	15	15	15
ParametricRBACMaintenance_mse20	15	0	0	0	3	3
planning	33	33	33	33	33	33
preference_planning	12	12	12	12	12	12
protein_ins	12	12	12	12	12	12
pseudoBoolean	15	14	14	14	14	14
qcp	15	15	15	15	15	15
quantum-circuit	15	15	15	14	15	15
railway-transport	5	2	2	1	2	1
ramsey	4	2	2	1	2	2
relational-inference	8	4	5	6	5	5
rna-alignment	15	10	10	15	15	15
setcover	6	0	3	3	0	0
spot5	15	13	10	11	11	11
switchingactivitymaximization	9	0	0	1	1	1
synplicate	40	15	15	16	15	15
tcp	15	13	13	13	13	14
timetabling	13	10	10	7	11	9
upgradeability	15	15	15	15	15	15
warehouses	8	3	8	8	5	8

observations are confirmed in the scatter plots in Figure 2. However, these plots additionally show that no solver dominates another, as there exist several instances that are solved by only one solver in each pair.

In comparison to MaxHS, all core-guided solvers perform worse overall. However, in some families, like JUDGEMENT-AGGREGATION and WAREHOUSES, OL^{3}P reduces the performance gap. Additionally, the average solving time of both OL^{3}P-S and OL^{3}P is signficantly lower in solved instances where it does prove optimality, although this is not sufficient to cover the gap in PAR2 score caused by the difference in number of instances solved.

Table 2 PAR2 scores.

Solver	PAR2 score	avg time
MaxHS	576598	160.462 s
CashWMaxSAT	640293	148.031 s
CashWMaxSAT +SCIP	670618	213.735 s
OL^{3}P-S	614729	94.2319 s
OL^{3}P	586971	94.9198 s

4.3 LP overhead

A natural question for both OL^{3}P-S and OL^{3}P is on the overhead of LP solving, since we replace the very cheap reformulation procedure of OLL by solving an LP. For OL^{3}P-S, this overhead is negligible. The solver spent just 16s solving the LP over the entire set of solved instances. The maximum amount of time spent in the LP solver in any one instances was just 1.15 s. In only 3 instances was the LP solver time more than 1 second, and they all were instances where the total time to solve them was more than 300 seconds. For OL^{3}P, the overhead is much greater. Cumulatively, OL^{3}P spent 8163.91 seconds in the LP solver (25 seconds per solved instance on average). The maximum amount of time in the LP solver in any one instance was 1343 seconds. The JUDGEMENT-AGGREGATION instances consume most of that time, with 4 of them spending more than 1000 seconds in the LP solver, each. Part of the reason for this overhead is that the current implementation is not particularly careful to be efficient with its use of the LP solver. These results suggest that more attention should be given to this, especially if other sources of constraints are used to populate the LP. Techniques like column and row generation can also speed up LP solving.

5 Conclusion

We have presented OL^{3}P-S, a core-guided algorithm for MaxSAT that uses a linear program to compute reformulations. This has advantages with respect to the behavior of the algorithm itself, as it computes stronger lower bounds from the same information. Moreover, since it can use constraints generated in any way, it opens many algorithmic possibilities. We explored here two: one that worked fairly well, seeding the LP with constraints of the MaxSAT formula; and one that did not, which was to try to solve the hitting set problem described by the LP as an integer program. Regardless, it is likely that more possible ways of enriching and using the linear program will emerge.

References

1 Benjamin Andres, Benjamin Kaufmann, Oliver Matheis, and Torsten Schaub. Unsatisfiability-based optimization in clasp. In *Technical Communications of the 28th International Conference on Logic Programming (ICLP'12)(2012)*, pages 212–221. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.ICLP.2012.211.

2 Carlos Ansótegui, Maria Luisa Bonet, Joel Gabàs, and Jordi Levy. Improving sat-based weighted maxsat solvers. In Michela Milano, editor, *Principles and Practice of Constraint Programming – 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, volume 7514 of *Lecture Notes in Computer Science*, pages 86–101. Springer, 2012. doi:10.1007/978-3-642-33558-7_9.

3 Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. Solving (weighted) partial maxsat through satisfiability testing. In *International conference on theory and applications of satisfiability testing*, pages 427–440. Springer, 2009. doi:10.1007/978-3-642-02777-2_39.

4 Fahiem Bacchus, Antti Hyttinen, Matti Järvisalo, and Paul Saikko. Reduced cost fixing in maxsat. In J. Christopher Beck, editor, *Principles and Practice of Constraint Programming – 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 – September 1, 2017, Proceedings*, volume 10416 of *Lecture Notes in Computer Science*, pages 641–651. Springer, 2017. doi:10.1007/978-3-319-66158-2_41.

5 Jeremias Berg, Fahiem Bacchus, and Alex Poole. Abstract cores in implicit hitting set maxsat solving. In Luca Pulina and Martina Seidl, editors, *Theory and Applications of Satisfiability Testing – SAT 2020 – 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 277–294. Springer, 2020. doi:10.1007/978-3-030-51825-7_20.

6 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-29)*, volume 14132 of *Lecture Notes in Computer Science*, pages 1–22, 2023. doi:10.1007/978-3-031-38499-8_1.

7 Suresh Bolusani, Mathieu Besançon, Ksenia Bestuzheva, Antonia Chmiela, João Dionísio, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Mohammed Ghannam, Ambros Gleixner, Christoph Graczyk, Katrin Halbig, Ivo Hedtke, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Dominik Kamp, Thorsten Koch, Kevin Kofler, Jurgen Lentz, Julian Manns, Gioni Mexi, Erik Mühmer, Marc E. Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Mark Turner, Stefan Vigerske, Dieter Weninger, and Lixing Xu. The SCIP Optimization Suite 9.0. Technical report, Optimization Online, February 2024. URL: https://optimization-online.org/2024/02/the-scip-optimization-suite-9-0/.

8 Guillaume Claus, Hadrien Cambazard, and Vincent Jost. Arc-consistency and linear programming duality: an analysis of reduced cost based filtering, 2022. doi:10.48550/arxiv.2207.10325.

9 Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In Jimmy Ho-Man Lee, editor, *Principles and Practice of Constraint Programming – CP 2011 – 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings*, volume 6876 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 2011. doi:10.1007/978-3-642-23786-7_19.

10 Jessica Davies and Fahiem Bacchus. Exploiting the power of mip solvers in maxsat. In Matti Järvisalo and Allen Van Gelder, editors, *Theory and Applications of Satisfiability Testing – SAT 2013 – 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings*, volume 7962 of *Lecture Notes in Computer Science*, pages 166–181. Springer, 2013. doi:10.1007/978-3-642-39071-5_13.

11 Jessica Davies and Fahiem Bacchus. Postponing optimization to speed up MAXSAT solving. In Christian Schulte, editor, *Principles and Practice of Constraint Programming – 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, volume 8124 of *Lecture Notes in Computer Science*, pages 247–262. Springer, 2013. doi:10.1007/978-3-642-40627-0_21.

12 Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into SAT. *J. Satisf. Boolean Model. Comput.*, 2(1-4):1–26, 2006. doi:10.3233/sat190014.

13 Filippo Focacci, Andrea Lodi, and Michela Milano. Cost-based domain filtering. In Joxan Jaffar, editor, *Principles and Practice of Constraint Programming – CP'99, 5th International Conference, Alexandria, Virginia, USA, October 11-14, 1999, Proceedings*, volume 1713 of *Lecture Notes in Computer Science*, pages 189–203. Springer, 1999. doi:10.1007/978-3-540-48085-3_14.

14 Zhaohui Fu and Sharad Malik. On solving the partial MAX-SAT problem. In Armin Biere and Carla P. Gomes, editors, *Theory and Applications of Satisfiability Testing – SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, volume 4121 of *Lecture Notes in Computer Science*, pages 252–265. Springer, 2006. doi:10.1007/11814948_25.

15 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.

16 Hannes Ihalainen, Jeremias Berg, and Matti Järvisalo. Unifying SAT-based approaches to maximum satisfiability solving. *Journal of Artificial Intelligence Research*, 2024. doi:10.1613/jair.1.15986.

17 George Katsirelos. An Analysis of Core-Guided Maximum Satisfiability Solvers Using Linear Programming. In Meena Mahajan and Friedrich Slivovsky, editors, *26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023)*, volume 271 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:19, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SAT.2023.12.

18 Tuukka Korhonen, Jeremias Berg, Paul Saikko, and Matti Järvisalo. Clause redundancy and preprocessing in maximum satisfiability. In Serge Gaspers and Toby Walsh, editors, *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing, (SAT '17)*, volume 10491 of *Lecture Notes in Computer Science*, pages 449–456. Springer, 2017. doi:10.1007/978-3-031-10769-6_6.

19 António Morgado, Carmine Dodaro, and João Marques-Silva. Core-guided maxsat with soft cardinality constraints. In Barry O'Sullivan, editor, *Principles and Practice of Constraint Programming – 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, volume 8656 of *Lecture Notes in Computer Science*, pages 564–573. Springer, 2014. doi:10.1007/978-3-319-10428-7_41.

20 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.

21 Nina Narodytska and Fahiem Bacchus. Maximum satisfiability using core-guided maxsat resolution. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 2717–2723, 2014. doi:10.1609/aaai.v28i1.9124.

22 Nina Narodytska and Nikolaj S. Bjørner. Analysis of core-guided maxsat using cores and correction sets. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2-5, 2022, Haifa, Israel*, volume 236 of *LIPIcs*, pages 26:1–26:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.SAT.2022.26.

23 Shiwei Pan, Yiyuan Wang, Shaowei Cai, Jiangman Li, Wenbo Zhu, and Minghao Yin. CASHWMaxSAT-DisjCad: Solver description. Technical report, Department of Computer Science, University of Helsinki, Helsinki, 2024. URL: http://hdl.handle.net/10138/584878.