SLS-Enhanced Core-Boosted Linear Search for Anytime Maximum Satisfiability

Ole Lübke[1]
Institute for Software Systems, Hamburg University of Technology, Germany

Jeremias Berg
Department of Computer Science, University of Helsinki, Finland

Abstract

Maximum Satisfiability (MaxSAT), the constraint paradigm of minimizing a linear expression over Boolean (0-1) variables subject to a set of propositional clauses, is today used for solving NP-hard combinatorial optimization problems in various domains. Especially anytime MaxSAT solvers that compute low-cost solutions within a limited available computational time have significantly improved in recent years. Such solvers can be divided into SAT-based methods that use sophisticated reasoning, and stochastic local search (SLS) methods that heuristically explore the search space. The two are complementary; roughly speaking, SLS struggles with finding feasible solutions, and SAT-based methods with minimizing cost. Consequently, most state-of-the-art anytime MaxSAT solvers run SLS before a SAT-based algorithm with minimal communication between the two.

In this paper, we aim to harness the complementary strengths of SAT-based, and SLS approaches in the context of anytime MaxSAT. More precisely, we describe several ways to enhance the performance of the so-called core-boosted linear search algorithm for anytime MaxSAT with SLS techniques. Core-boosted linear search is a three-phase algorithm where each phase uses different types of reasoning. Beyond MaxSAT, core-boosted search has also been successful in the related paradigms of pseudo-boolean optimization and constraint programming. We describe how an SLS approach to MaxSAT can be tightly integrated with all three phases of the algorithm, resulting in non-trivial information exchange in both directions between the SLS algorithm and the reasoning methods. We evaluate our techniques on standard benchmarks from the latest MaxSAT Evaluation and demonstrate that our techniques can noticeably improve on implementations of core-boosted search and SLS.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization; Theory of computation → Constraint and logic programming

Keywords and phrases Maximum Satisfiability, MaxSAT, SAT, SLS, Anytime Optimization

Digital Object Identifier 10.4230/LIPIcs.CP.2025.28

Supplementary Material *Software (archive)*: `https://doi.org/10.5281/zenodo.15584837`
Software (repository): `https://github.com/jezberg/loandra/tree/boums-cadical-integration` [9], archived at `swh:1:dir:6babae3bfbb34e637ca99fab6b5f3a85f2263476`

Funding The work is supported by Research Council of Finland under grant 362987.

Acknowledgements The authors thank the Finnish Computing Competence Infrastructure (FCCI) for computational and data storage resources.

1 Introduction

Maximum Satisfiability (MaxSAT) has, over the last few years, matured into a thriving optimization paradigm that is today used for solving NP-hard optimization problems across various domains [6]. While especially theoretical work tends to treat MaxSAT as the problem

[1] Corresponding author

of computing assignments that maximize the (sum-of-weights) of satisfied clauses, most modern MaxSAT solvers treat MaxSAT as the equivalent problem of minimizing a linear cost function over 0-1 variables subject to a set of propositional clauses. Modern MaxSAT solving techniques can roughly be divided into two categories: exact and anytime solvers. Exact solvers aim to compute minimum-cost solutions using as little time and memory as possible. While there has been some recently renewed interest in branch and bound techniques for MaxSAT [33], a clear majority of current exact approaches build on the highly successful conflict-driven clause learning (CDCL) SAT solvers [35] and, essentially, reduce the optimization task into a sequence of decision queries [37, 2].

In this work, we focus on anytime MaxSAT, which aims to compute as low-cost solutions as possible within limited computation time and memory. Existing approaches to anytime MaxSAT can roughly be divided into approaches based on CDCL solvers, called SAT-based approaches [40, 38, 10, 20], and approaches based on stochastic local search (SLS) [30, 31, 32, 18, 17, 28]. SAT-based anytime solvers extensively use CDCL solvers and apply sophisticated reasoning techniques in their search. SLS approaches, in turn, heuristically traverse the search space for solutions as effectively as possible while looking for low-cost solutions. The two approaches are known to exhibit orthogonal performance. Roughly speaking, the reasoning techniques employed by CDCL-based approaches excel at finding feasible solutions or escaping local optima, while SLS approaches excel at quickly finding low-cost solutions in local optima when given a feasible solution from which to start. This orthogonality and the promise of combining reasoning-based CDCL-like and search-based SLS-like methods have previously been realized in related fields. As a notable example related to our work, we note that integrating a local search component in a CDCL solver has been shown to increase performance, especially on satisfiable instances [15].

The orthogonal performance of SAT-based and SLS methods for anytime MaxSAT is well understood by MaxSAT solver developers. Since 2018, practically all anytime solvers participating in the MaxSAT evaluations first apply a SAT solver to obtain any feasible solution, use that solution as a starting point for an SLS algorithm, and finally – if the time limit is not yet reached – attempt to improve the solution returned by the SLS approach through a SAT-based algorithm. In the other direction, some SLS approaches to MaxSAT use the propagation mechanics of a CDCL-type solver to escape local optima [16]. We study further ways to combine the strengths of SAT-based and SLS approaches to anytime MaxSAT. In particular, we work with the so-called core-boosted linear search algorithm to anytime MaxSAT [10] that uses a multitude of CDCL-based reasoning techniques, including preprocessing, lower-bounding core-guided search [41], and upper-bounding solution improving search [42]. Core-boosted linear search has been used in many applications, including interpretable data analysis [24, 44] and scheduling [19]. Similar algorithmic ideas have also successfully been applied in more expressive constraint paradigms like Pseudo-Boolean optimization [21], constraint programming [23] and in more general optimization tasks like multiobjective optimization [26].

More specifically, in this work, we i) identify several novel ways of integrating an SLS approach to all phases of core-boosted search and ii) provide a systematic study on the effect of applying SLS prior to a CDCL-based algorithm in anytime MaxSAT. The main novel integrations of SLS with CDCL-based search we propose are the use of unsatisfiable cores in SLS and the use of SLS interleaved in the so-called increasing precision heuristic (called varying resolution in previous work) [20]. Our experimental evaluation on standard benchmarks demonstrates that sophisticated ways of integrating SLS can result in improved performance of core-boosted search, but the precise effect significantly depends on the specific solver instantiations.

The rest of the paper is structured as follows. In Section 2, we discuss anytime MaxSAT, core-boosted linear search, and SLS for MaxSAT to the extent required for understanding our work. In Section 3, we detail the ways in which SLS can be integrated with all three phases of core-boosted linear search. In Section 4 we report on an experimental evaluation of the effectiveness of the resulting integration of SLS and core-boosted search, before giving concluding remarks in Section 5.

2 Core-Boosted Linear Search and SLS for anytime MaxSAT

In this section, we define Maximum Satisfiability (MaxSAT) and related terms that are used throughout the paper, followed by descriptions of core-boosted linear and stochastic local search (SLS) for MaxSAT.

2.1 Maximum Satisfiability

A literal ℓ is a $\{0,1\}$-valued variable x or its negation $\bar{x}$. A clause $C \equiv \sum_i \ell_i \geq 1$ is an at-least-one constraint and a (CNF) formula F is a set of clauses. We will often treat clauses as the set of literals that appear in them. An objective O is a pseudo-Boolean expression $\sum_i c_i \ell_i$ where each c_i is a positive integer. A (truth) assignment α maps variables to 0 (which we identify with FALSE) or 1 (TRUE). The semantics of assignments are extended to literals, clauses, and formulas in the standard way: $\alpha(\bar{x}) = 1 - \alpha(x)$, $\alpha(C) = \max\{\alpha(\ell) \mid \ell \in C\}$, and $\alpha(F) = \min\{\alpha(C) \mid C \in F\}$. The value (or cost) of O under α is $\alpha(O) = \sum_i c_i \alpha(\ell_i)$. With a slight abuse of notation, we sometimes treat an objective O as a set of its terms, i.e. write $c\ell \in O$ if $O \equiv c\ell + \sum_i c_i \ell_i$. α is a solution of F if $\alpha(F) = 1$. F is satisfiable if it has solutions and unsatisfiable if not.

We adopt two equivalent definitions of Weighted Partial Maximum Satisfiability (WPMS) to present core-boosted linear and stochastic local search (SLS). Specifically, SLS approaches are more naturally described in terms of the so-called *clause-based definition* of MaxSAT in which an instance (F_H, F_S, w) consists of two formulas, the hard clauses F_H, the soft clauses F_S, and a weight function w that associates a positive integer weight $w(C)$ to each soft clause $C \in F_S$. The goal is to compute a solution α of F_H that minimizes the cost $\text{COST}(F_S, w, \alpha) = \sum_{C \in F_S} w(C)(1 - \alpha(C))$, defined as the weights of falsified soft clauses. Core-boosted linear search (and indeed any approach that makes use of a CDCL SAT-solver) for MaxSAT is more natural to describe in terms of the so-called *objective-based* definition of MaxSAT in which an instance (F, O) consists of a formula F and an objective O. The goal is to minimize O subject to F, i.e. compute a solution α of F that minimizes $\alpha(O)$ over all solutions of F. For both definitions of MaxSAT a minimum-cost solution of F is called an optimal solution of F. The *optimum cost* of the instance is the cost of an optimal solution. In *anytime* MaxSAT solving, the goal is to compute a solution of as low cost as possible within some fixed amount of computational resources, typically computation time.

For intuition on the equivalence between the clause-based and the objective-based definitions of MaxSAT, note that a term $c\ell$ of an objective can be viewed as a soft clause $(\bar{\ell})$ with weight c. Thus, any objective-based instance (F, O) can be seen as the clause-based instance $(F, \{(\bar{\ell}) \mid c\ell \in O\}, w)$ where $w((\bar{\ell})) = c$ whenever $c\ell \in O$. In the other direction, a clause-based instance (F_H, F_S, w) can be viewed as an objective-based instance $(F_H \cup \{C \vee b_C \mid C \in F_S\}, \sum_{C \in F_S} w(C) b_C)$ formed by: (i) extending each soft clause C with a fresh variable b_C, (ii) forming a formula that contains the hard clauses and the extended soft clauses, and (iii) forming an objective to be the sum of the weights of soft clauses and their extension variables. For more details on the transformation and the equivalence between the definitions, we refer the reader to [6].

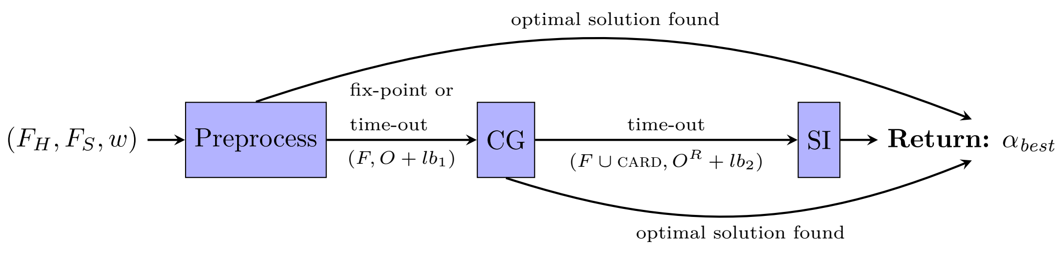


Figure 1 Core-boosted linear search. CG is the core-guided phase, and SI is the solution-improving phase. We assume every step can update the best-known solution α_{best}.

▶ **Example 1.** Consider the clause-based MaxSAT instance (F_H, F_S, w) with: $F_H = \{(x_1 \lor \bar{x}_2), (\bar{x}_1 \lor \bar{x}_2), (x_2 \lor \bar{x}_3)\}$, $F_S = \{(x_1 \lor x_2), (\bar{x}_1 \lor x_2), (x_1 \lor x_2 \lor x_3)\}$, and $w((x_1 \lor x_2)) = 1, w((\bar{x}_1 \lor x_2)) = 2, w((x_1 \lor x_2 \lor x_3)) = 3$. The corresponding objective-based instance (F^o, O) has $F^o = F_H \cup \{(x_1 \lor x_2 \lor b_1), (\bar{x}_1 \lor x_2 \lor b_2), (x_1 \lor x_2 \lor x_3 \lor b_3)\}$ and $O = b_1 + 2b_2 + 3b_3$. An optimal solution α to the instance assigns $\alpha(x_2) = \alpha(x_3) = \alpha(b_1) = \alpha(b_3) = 0$ and $\alpha(x_1) = \alpha(b_2) = 1$ and has $\textsc{cost}(F_S, w, \alpha) = \alpha(O) = 2$.

2.2 Incremental SAT Solving and Unsatisfiable Cores

Core-boosted linear search makes extensive use of incremental SAT-solving under assumptions and unsatisfiable cores. An *unsatisfiable core* of an objective-based MaxSAT instance (F, O) is a clause C that: (i) only contains variables that appear in the objective, and (ii) is satisfied (i.e. assigned to 1) by any solution to F. Notice that satisfying a core requires assigning at least one objective variable to 1, thus incurring cost in O. In this sense, cores provide information on the lower bound of the optimal cost of the instance.

To see the connection between our clause-based definition of a core and the more classical one in terms of unsatisfiable subsets of constraints, note that requirement (ii) that all solutions to F satisfy C is equivalent to the formula $F \land \bigwedge_{\ell \in C} \bar{\ell}$ being unsatisfiable. Thus, our definition of a core could be viewed as the negation of an unsatisfiable subset of F obtained by assigning each literal in C to 0 and simplifying accordingly.

In practice, cores of MaxSAT instances are extracted using incremental SAT solving under assumptions [22, 7]. We abstract the use of an incremental SAT solver (i.e. a decision procedure for propositional logic) into the function `Inc-SAT-solve`$(F, \mathcal{A})$ where F is a CNF-formula and $\mathcal{A}$ is a set of literals, typically called assumptions. The call returns a triplet $(sat?, C, \alpha)$. If *sat?* is true, then α is a solution to F that extends $\mathcal{A}$, i.e. assigns $\alpha(\ell) = 1$ for all $\ell \in \mathcal{A}$. Otherwise (if *sat?* is false), no such solution exists, and C is a clause satisfied by any solution of F containing negations of some of the literals in $\mathcal{A}$. Notice that F has solutions iff `Inc-SAT-solve`$(F, \emptyset)$ returns true. Unsatisfiable cores of MaxSAT instances are then extracted by having $\mathcal{A}$ contain negations of objective variables. Practically all modern SAT solvers offer an assumption interface out of the box [22, 7].

▶ **Example 2.** Consider the objective-based instance (F^o, O) detailed in Example 1. Invoking `Inc-SAT-solve`$(F^o, \{\bar{b}_2\})$ returns true and (for example) the solution α_1 that assigns $\alpha_1(x_1) = \alpha_1(x_2) = \alpha_1(x_3) = \alpha_1(b_2) = 0$ and $\alpha_1(b_1) = \alpha_1(b_3) = 1$. Invoking `Inc-SAT-solve`$(F^o, \{\bar{b}_1, \bar{b}_2\})$ returns false and the core $(b_1 \lor b_2)$.

2.3 Core-Boosted Linear Search for Anytime MaxSAT

Figure 1 overviews core-boosted linear search for computing a low-cost solution to a (clause-based) MaxSAT instance (F_H, F_S, w) [10]. Core-boosted search consists of three separate phases: preprocessing, core-guided search, and solution-improving search. Each phase can – in theory – find and identify an optimal solution to the instance. More importantly, each phase will also modify and simplify the instance, and the following phase is always invoked on the modified instance rather than the original one. Intuitively, this allows core-boosted search never to be much worse than any individual phase, while also allowing subsequent phases to benefit from deductions made in previous ones. We next detail all three phases.

The Preprocessing Phase

The preprocessing phase converts the input clause-based instance (F_H, F_S, w) into an objective-based one using the transformation outlined in Section 2.1. Additionally, the step applies different simplification techniques. These can range from removing tautologies and reusing unit soft clauses as objective terms to more sophisticated preprocessing rules used in SAT and MaxSAT solving, including bounded variable elimination and subsumed label elimination [8, 25]. For the correctness of core-boosted linear search, we assume that the preprocessing techniques are cost-preserving.

▶ **Definition 3.** *Invoke the preprocessing phase on an instance (F_H, F_S, w) and assume it returns the instance $(F, O + lb_1)$ where lb_1 is a constant. The preprocessing phase is cost-preserving if both of the following hold:*

(i) *If α^P is a solution of F then there exists a solution α to F_H for which $\textsc{cost}(F_S, w, \alpha) \leq \alpha^P(O) + lb_1$.*

(ii) *If α is a solution of F_H then there exists a solution α^P to F for which $\alpha^P(O) + lb_1 = \textsc{cost}(F_S, w, \alpha)$.*

Several examples of cost-preserving preprocessing techniques are known. For practically all of them, given a solution α^P to the preprocessed instance $(F, O + lb_1)$, a solution α to the original instance (F_H, F_S, w) for which $\textsc{cost}(F_S, w, \alpha) \leq \alpha^P(O) + lb_1$ can be obtained in linear time with respect to the number of clauses in $F_H \cup F_S$ [8, 25].

The Core-Guided Phase

The next phase of core-boosted linear search applies core-guided search on the preprocessed instance $(F, O + lb_1)$. In the core-boosted setting, core-guided search only runs for a limited time. More precisely, the phase extracts a set $\textsc{cores}$ of cores of $(F, O + lb_1)$ by repeated invocations of an incremental SAT solver in the way described in Section 2.2. When either the time limit for the core-guided phase is reached or no more cores can be found, the core-guided phase forms a new instance $(F \cup \textsc{card}, O^R + lb_2)$ by relaxing all cores in $\textsc{cores}$. For some informal intuition on core-guided search, we have that the difference $lb_2 - lb_1 \geq 0$ is equal to the minimum cost incurred by satisfying the cores in $\textsc{cores}$, and that the objective O^R measures the additional cost on top of satisfying the cores in $\textsc{cores}$ that solutions of $(F \cup \textsc{card})$ incur in O. More precisely, we require that the core relaxation step is solution preserving in the sense of the next definition.

▶ **Definition 4.** *Invoke the core-guided phase on the instance $(F, O + lb_1)$ and let $(F \cup \textsc{card}, O^R + lb_2)$ be the instance returned. The core-guided phase is solution-preserving if the following conditions hold:*

(i) $\alpha^R(O^R) + lb_2 = \alpha^R(O) + lb_1$ *for any solution* α^R *to* $F \cup \textsc{card}$*.*
(ii) *Any solution* α *to* F *can be extended into a solution* α^R *to* $F \cup \textsc{card}$ *for which* $\alpha^R(O^R) + lb_2 = \alpha(O) + lb_1$*.*

In practice, all core-guided algorithms for MaxSAT that we are aware of are solution-preserving and can be used for core-boosted linear search. All of them implement the core relaxation step by introducing new variables of form o^k_C to O^R as indicators for at least $k \geq 2$ variables in the core C being assigned to 1 by solutions of $(F \cup \textsc{card})$. Essentially, such variables ensure that a solution to $F \cup \textsc{card}$ can assign one variable from each core to 1 without incurring cost in O^R, but need to assign one of the new variables to 1 (thus incurring cost) for any subsequent ones. The additional clauses in $\textsc{card}$ contain CNF encodings of reified cardinality constraints equivalent to $\sum_{\ell \in C} \ell \geq k \rightarrow o^k_C$. While existing instantiations of core-boosted search for MaxSAT only extract cores over original objective variables, in the general case, due to core relaxation changing the objective function, subsequent cores could also contain the new variables o^k_C.

The Solution-Improving Phase

The solution-improving phase is invoked on $(F \cup \textsc{card}, O^R + lb_2)$, the final working instance of the core-guided phase. The phase looks for low-cost solutions of $(F \cup \textsc{card}, O^R)$ by invoking a SAT solver on the formula $F \cup \textsc{card} \cup \textsc{as-cnf}(O^R < \alpha_{best}(O^R))$ where α_{best} is the current lowest-cost solution, and $\textsc{as-cnf}(O^R \leq \alpha_{best}(O^R))$ is a CNF encoding of the objective-improving constraint $O^R < \alpha_{best}(O^R)$ that is satisfied by any solution α for which $\alpha(O^R) < \alpha_{best}(O^R)$. If the SAT solver returns a solution, it will cost less than α_{best}. Then, the objective-improving constraint is tightened, and the solver is invoked again. If, instead, the SAT solver determines that no such solution exists, the current best-known solution is returned as optimal. The solution-improving phase runs until either an optimal solution of $(F \cup \textsc{card}, O^R)$ is found, or a time-limit is reached, at which point, the solution α of the input clause-based instance (F_H, F_S, w) for which $\textsc{cost}(F_S, w, \alpha) \leq \alpha_{best}(O) + lb_1$ is returned.

An essential intuition of core-boosted linear search is that any solution of $F \cup \textsc{card}$ that has a low-cost wrt $O^R + lb_2$ will correspond to a solution α of (F_H, F_S, w) for which $\textsc{cost}(F_S, w, \alpha)$ is low. This is because the core-guided phase is solution-preserving, and the preprocessing phase is cost-preserving. Next, we detail two additional heuristics designed for any time solution-improving search that are important for our work.

Increasing Precision. Solution-improving search with increasing precision [20] heuristically picks a constant ρ and runs solution-improving search on the objective $O^\rho = \lfloor O^R/\rho \rfloor$ obtained by: (i) dividing each coefficient in O^R by ρ, (ii) rounding down, and (iii) removing all variables with 0 coefficient. For some intuition on the name, notice how O^ρ is a "lower precision" approximation of O^R for which the solution improving constraint $O^\rho < \alpha_{best}(O^\rho)$ can be encoded with fewer clauses. While the division of an objective obviously can change the cost of solutions, in practice it has been observed that solutions that have a low-cost wrt O^ρ also tend to have a low-cost wrt O^R [20, 10]. If a minimum cost solution wrt O^ρ is found, the value of ρ is lowered – thus increasing the precision – and the search restarted. Solution-improving search with increasing precision ends either when a minimum-cost solution of O^ρ is found when $\rho = 1$, or when a time limit is reached.

Phase Saving. Phase saving refers to overriding the default polarity selection of the SAT solver to guide its search toward solutions of low cost. The intuition underlying phase saving is that a SAT solver, in its default setting, aims to find any solution of its internal formula or prove that one does not exist. In the context of anytime MaxSAT, not all solutions are equal, so phase saving aims to ensure that the first solution that the solver finds is as low-cost as possible. Typical phase-saving schemes employed in anytime MaxSAT [3, 20, 38] involve setting the default polarity of the variables in the objective to not incur cost, and set the polarity of the remaining variables to match the current lowest-cost solution α_{best}.

2.4 Stochastic Local Search for MaxSAT

Algorithm 1 SLS for MaxSAT.

```
Require: $(F_H, F_S, w)$, $\alpha$, max-flips
Ensure: $\alpha_{best}$ is a solution
 1: UPDATE-BEST-KNOWN-SOL
 2: $w^{\texttt{sls}} \leftarrow$ INIT-WEIGHTS
 3: while flip-count $\leq$ max-flips do
 4:     if $\alpha(F_H) = 1$ and COST$(F_S, w, \alpha) <$ best-cost then UPDATE-BEST-KNOWN-SOL
 5:     end if
 6:     if $\nexists$ decreasing variable then $w^{\texttt{sls}} \leftarrow$ UPDATE-WEIGHTS
 7:     end if
 8:     $\alpha \leftarrow$ SELECT-AND-FLIP-VARIABLE
 9:     flip-count $\leftarrow$ flip-count $+ 1$
10: end while
11: return $\alpha_{best}$
12:
13: function UPDATE-BEST-KNOWN-SOL
14:     $\alpha_{best} \leftarrow \alpha$, best-cost $\leftarrow$ COST$(F_S, w, \alpha)$, flip-count $\leftarrow 0$
15: end function
```

Algorithm 1 details the overall structure of a stochastic local search (SLS) algorithm for computing a low-cost solution to a clause-based MaxSAT instance (F_H, F_S, w). In addition to the instance itself, the algorithm expects a user-specified parameter on how long to search for. In contrast to the algorithms that make use of a SAT solver, an SLS approach is, in general, not able to identify optimal solutions, which makes the limit parameter mandatory. A typical way of limiting the run time of SLS approaches is to enforce a maximum number of flips `max-flips` and terminate the search if no improving solutions have been found within `max-flips` flips. Finally, the algorithm also expects a solution α of F_H as input. In the general setting, an SLS algorithm can be invoked using any assignment of the variables in $F_H \cup F_S$. However, prior work has shown that invoking an SLS approach with a solution of F_H leads to much better performance in terms of the cost of the final solution being found. Since we are working on different ways of integrating SLS with SAT-based approaches to MaxSAT, we can assume that we always have access to some solution of the instance being solved. In our setting, such a solution can be obtained, e.g. by invoking the SAT solver on the hard clauses.

Algorithm 1 begins by initializing its best-known solution α_{best} to equal the input solution α and a flip counter to 0 on Line 1. Afterward, the SLS weights $w^{\texttt{sls}}$ for all clauses in $F_H \cup F_S$ are initialized. Importantly, these are not the same as the weights of the soft clauses in the instance given by w. Instead, $w^{\texttt{sls}}$ will be used to define the score of each variable.

▶ **Definition 5.** *Assume an SLS approach has a current assignment α and SLS weights w^{sls} when solving an instance (F_H, F_S, w). For a variable x let α_x be the assignment obtained by flipping the value of x, i.e. α_x agrees with α on all variables except x. The* score *of x wrt α is then $\textsc{score}(x) = \textsc{cost}(F_H \cup F_S, w^{sls}, \alpha) - \textsc{cost}(F_H \cup F_S, w^{sls}, \alpha_x)$.*

In other words, the score of a variable x is the difference in the sum of weights wrt $w^{\mathtt{sls}}$ of falsified clauses (hard or soft) when flipping the variable x. Informally speaking, flipping a variable x with a high positive score will result in the current solution improving. With this intuition, the main loop (Lines 3 to 9) first checks on Line 4 whether the current assignment α' is a solution of lower cost than the best know, and if it is, the best-known solution and `best-cost` are updated. Additionally, the flip counter (`flip-count`) is reset to 0. Note specifically that the flip counter measures the number of flips since the last improvement to the solution was found. Afterward, the existence of so-called decreasing variables is checked. Here a variable is decreasing if its score is positive since flipping variables with a positive score decreases the cost of the assignment wrt $w^{\mathtt{sls}}$ (cf. Definition 5). If no such variables exist, the algorithm has encountered a local optimum, and the SLS weights $w^{\mathtt{sls}}$ are updated to escape it. Note that practical implementations of SLS algorithms directly modify and store the score of affected variables as well. Finally, a variable is selected, and the current assignment is updated by flipping its value on Line 8, and the flip counter is increased on Line 9. Commonly, variable selection employs the Best from Multiple Selection (BMS) heuristic [14], where a fixed number of variables is drawn randomly with replacement, and the variable with the highest score is selected from those.

Different SLS algorithms differ mainly in how the SLS weights are built and updated, and the specific choice of weighting scheme is known to impact the empirical performance of the approach significantly. In our work, we make use of the so-called NuWLS 2.0 weighting scheme for MaxSAT that was introduced in the NuWLS-c-2023 solver [17] that won all four anytime tracks of the MaxSAT evaluation 2023 and has since been integrated into other, well-performing MaxSAT solvers as well [5, 11, 27, 39]. When solving a MaxSAT instance (F_H, F_S, w) NuWLS 2.0 initializes the weight of each hard clause to 1 and the weight of each soft clause to 0. Informally, this encourages Algorithm 1 to initially flip variables more likely to satisfy the hard clauses. The way the weights are updated on Line 6 depends on if the current assignment α is a solution of F_H. If not, the weights of *all falsified* hard clauses are increased by 1. Otherwise, if $\alpha(F_H) = 1$, the weights of *all* soft clauses are increased by their *tuned weight.*

▶ **Definition 6.** *Given an instance (F_H, F_S, w), the tuned weight of a soft clause $c \in F_S$ is equal to $\frac{w(c)|F_S|}{\sum_{d \in F_S} w(d)}$.*

In other words, the tuned weight of a soft clause c is $w(c)$ divided by the average weight of all soft clauses. Dividing by the average effectively distributes the tuned weights around 1: clauses c with smaller-than-average (respectively larger-than-average) $w(c)$ have a tuned weight $\in (0, 1)$ (respectively > 1).

3 Integrating SLS with Core-boosted Linear Search

In this section, we describe the different ways in which we integrate SLS with all phases of core-boosted linear search. Many of the techniques we describe involve invoking an SLS algorithm (i.e. Algorithm 1) on an objective-based instance (F, O). This should be understood as invoking Algorithm 1 on the clause-based instance obtained by treating each term $c\ell \in O$ as a unit soft clause $(\overline{\ell})$ of weight c. In the following sections, assume that core-boosted linear search is invoked on a clause-based instance (F_H, F_S, w).

3.1 SLS in the Preprocessing Phase

A very natural way of integrating SLS with a core-boosted linear search would be to first run the preprocessing phase on the clause-based input instance to obtain the objective-based instance $(F, O + lb_1)$, then invoke a SAT solver on F to obtain an initial solution α, and finally invoke Algorithm 1 on (F, O) with α as the initial assignment and use the α_{best} returned as an initial assignment for the rest of the core-boosted linear search. This idea is well-studied in the context of anytime MaxSAT and corresponds to the de facto way in which most state-of-the-art anytime MaxSAT solvers integrate SLS with a CDCL-based algorithm [39, 28, 17].

While the effectiveness of applying an SLS algorithm before a CDCL-based algorithm is well understood in anytime MaxSAT (as witnessed e.g. by the results of the latest MaxSAT evaluations), much less is known about the relative effectiveness of running an SLS algorithm on a clause-based MaxSAT instance compared to the objective-based MaxSAT instances obtained after preprocessing. In this work, we study the effect of running Algorithm 1 both before and after the preprocessing phase of the core-boosted search. More precisely, given $(F, O + lb_1)$, we first call `Inc-SAT-solve`$(F, \emptyset)$ in order to obtain a solution α of F which we use as a starting point to invoke Algorithm 1 on the instance (F, O), and obtain a solution α' (potentially equal to α). Then, we invoke Algorithm 1 on the instance (F_H, F_S, w) using the solution α^R of F_H that corresponds to the solution α' of F as the initial assignment.

Notice that, especially when employing more advanced preprocessing rules like bounded variable elimination, the clauses in F might significantly differ from those in $F_H \cup F_S$ and the relative effectiveness of Algorithm 1 in terms of the cost of the solution returned is an open question. Another important intuition here is that the solution β returned by Algorithm 1 when invoked on (F_H, F_S, w) need not be a solution of F, thus while the cost $\textsc{cost}(F_S, w, \beta)$ is an upper bound on the optimal cost of $(F, O + lb_1)$, β can not be used for phase saving (cf. Section 2.3). In our implementation, we store β throughout the rest of the core-boosted search, and in the end, compare the best solution found by core-boosted search with β and return the better of the two.

▶ **Example 7.** Consider the clause-based instance (F_H, F_S, w) from Example 1. Assume the preprocessing phase (as a consequence of e.g. applying bounded variable elimination) returns the objective-based instance (F, O) with $F = \{(x_1 \vee b_1), (\bar{x_1} \vee b_2)\}$, and $O = 4b_1 + 2b_2$. Invoking `Inc-SAT-solve`$(F, \emptyset)$ could obtain the solution α that assigns $\alpha(x_1) = \alpha(b_1) = \alpha(b_2) = 1$ with $\alpha(O) = 6$ that can be used as the starting point for SLS (Algorithm 1) invoked on (F, O). α corresponds to the solution β of F_H that assigns $\beta(x_1) = 1, \beta(x_2) = \beta(x_3) = 0$ that has $\textsc{cost}(F_S, w, \beta) = 2$ and can be used as a starting point for running Algorithm 1 on (F_H, F_S, w).

3.2 SLS in the Core-Guided Phase

During the core-guided phase of a core-boosted linear search, most of the SAT solver invocations are expected to return false and a new core. Core extraction essentially boils down to learning new clauses, and since SLS approaches can not learn new clauses nor determine that an instance is unsatisfiable, their applicability during the core-guided phase is limited. For general core-guided algorithms, SLS could conceivably help with heuristics that rely on finding intermediate solutions, most notably the so-called core exhaustion (also called cover optimization) technique [1, 36] that seeks to greedily strengthen the extracted cores with additional SAT calls under relaxed sets of assumptions. Since the PMRES [41] core-guided algorithm that the particular instantiation of core-boosted search we work with uses can not be extended with core exhaustion, we instead study ways in which the cores learned during this phase can be used to improve the performance of the SLS approach in the solution-improving phase where the goal is to compute new solutions, and SLS is expected to perform better. Additionally, we use the SLS approach at the end of the core-guided phase to find a low-cost solution to the instance obtained after the core-guided algorithm performs the core relaxation steps.

Using Cores in SLS

The core-guided phase is invoked on the objective-based instance $(F, O + lb_1)$ obtained after the preprocessing phase. It iteratively extracts a set of cores $\textsc{cores}$ of (F, O) in the form of clauses over the variables in O that are satisfied by all solutions in F. An essential intuition for how we use cores in an SLS algorithm is that the subsequent solution-improving phase of a core-boosted algorithm is invoked on the instance $(F \cup \textsc{card}, O^R)$ that also contains all the clauses in F. Thus, all the solutions of interest during the solution-improving phase will also have to satisfy all the cores extracted during the core-guided phase. Furthermore, such cores are obtained by the sophisticated reasoning performed by the SAT solver and can, as such, represent information that the comparatively much simpler SLS algorithm does not have access to, motivating the inclusion of cores in the SLS algorithm.

A very natural idea of how to incorporate cores into Algorithm 1 during the solution-improving phase is to add all cores as (hard) clauses to the instance, i.e. to run Algorithm 1 on the instance $(F \cup \textsc{card} \cup \textsc{cores}, O^R)$ rather than $(F \cup \textsc{card}, O^R)$.

▶ **Example 8.** Assume the core-guided phase was executed on the objective-based instance (F, O) from Example 7, and found the core $C = (b_1 \lor b_2)$. As C is satisfied by all solutions of F, it can be added to its clauses without changing the set of solutions. It can, however steer the search of the SLS algorithm more effectively toward feasible solutions by effectively communicating that both of the soft clauses $(\bar{b_1})$ and $(\bar{b_2})$ in the clause-based representation of (F, O) that SLS in invoked on, cannot be satisfied simultaneously by any solution.

While the direct addition of the cores is conceptually simple and an easy-to-implement way of communicating information from the cores to the SLS algorithm, adding a potentially large number of new clauses into the instance risks decreasing the overall effectiveness of Algorithm 1 and slowing down the rate at which flips can be performed. Thus, we also consider a way to integrate information in the cores directly in the weighting scheme of Algorithm 1. Recall that the score of a variable (cf. Definition 5) essentially indicates to the algorithm how important it is to flip that variable to steer toward low-cost solutions, and that the score of a variable represents a cost reduction wrt the solver-internal clause weights $w^{\texttt{sls}}$ (cf. Definition 5). The higher the score, the likelier the variable is to get flipped.

Informally speaking, our "core-aware" extension of the NuWLS weighting scheme described in Section 2.4 increases the scores of variables that appear in cores that are assigned to 0 by the current assignment of the SLS algorithm to guide the algorithm toward feasible solutions and decreases the scores of variables appearing in cores that are already assigned to 1 to discourage the search from setting more than one variable to 1 from each core, as doing so would incur more cost in the objective. Additionally, we note that some of the variables in the cores might have had their coefficients set to 0 in O^R. Conceptually, all the cost of assigning such a variable to 1 is already accounted for in the lower bound derived by the core-guided phase. As such, assigning them to 1 during the solution-improving phase incurs no additional cost. With this intuition, we lower the scores of such variables by a larger amount to encourage the SLS algorithm to satisfy the cores on these variables specifically.

Algorithm 2 Modification of SLS scores to incorporate cores.

```
Require: $(F \cup \textsc{card}, O^R)$, $\textsc{cores}$, scores, core-factor $\alpha$
 1: for $C \in \textsc{cores}$ do
 2:     $n \leftarrow$ number of literals in $C$ assigned to 1 by $\alpha$
 3:     for $\ell \in C$ do
 4:         $w \leftarrow$ tuned weight of $\ell$ in $O^R$
 5:         if $w = 0$ then $change \leftarrow$ core-factor $\cdot$ maximum tuned weight in $O^R$
 6:         else $change \leftarrow \frac{1}{w}$
 7:         end if
 8:         if $n = 0$ then add $change$ to scores[$\ell$]
 9:         else subtract $n \cdot change$ from scores[$\ell$]
10:         end if
11:     end for
12: end for
```

Algorithm 2 details the core-based extension of variable scores that we invoke after the weight initialization phase of SLS (cf. Algorithm 1, Line 2). The procedure alters the score of each literal ℓ that appears in a core found during the core-guided phase based on the tuned weight (cf. Definition 6) of ℓ in O^R where O^R is the objective resulting after the core-relaxation steps performed by the core-guided phase. The change in score is based either on the coefficient of ℓ in O^R, or – if the coefficient of ℓ is 0 – by the maximum tuned weight of any literal in O^R.

During the SLS procedure, whenever a variable is flipped, we determine the affected cores (the implementation maintains a mapping from variables to cores), update the count of satisfied literals, and then modify the variable scores of each affected core accordingly, i.e. the variable scores of falsified cores are increased by the inverse corresponding weight, and those of satisfied cores are decreased by n times the inverse corresponding weight. Note, that initially, $n > 0$ for all cores, since we always start SLS from an initial assignment that is a solution, so it satisfies all cores. After a variable flip we do not run Algorithm 2 again, but update the scores so that the resulting values are equal to the values we would obtain by executing Algorithm 2 if the current assignment were the initial assignment. The idea of using information from cores in the SLS-phase bears some resemblance to encoding the constraints of an optimization problem into the optimization criteria via so-called penalty functions [45]. However, as all cores are logically entailed by the constraints in the instance, including any of them in the weighting function does not relax the instance or change the set of feasible solutions to it.

SLS for Obtaining a Solution to the Relaxed Instance

At the end of the core-guided phase, the cores of $(F, O + lb_1)$ are relaxed and the instance $(F \cup \textsc{card}, O^R + lb_2)$ on which the solution improving phase will be invoked is formed. At this point, the SAT solver used by the core-boosted search only contains the clauses in F. As a consequence, the best-known solution α_{best} will only assign the variables in F and O. To obtain an upper bound on the optimal cost of $(F \cup \textsc{card}, O^R + lb_2)$ and a solution for the solution improving phase to use phase saving with, α_{best} needs to be extended into a solution of $(F \cup \textsc{card}, O^R + lb_2)$.

Before our work, the de facto way in which implementations of core-boosted search algorithms for MaxSAT extended α_{best} into a solution of $F \cup \textsc{card}$ was to invoke a SAT solver on the clauses $F \cup \textsc{card}$ with assumptions that correspond to α_{best}, i.e. invoke

`Inc-SAT-solve`$(F \cup \textsc{card}, \{\ell \mid \alpha_{best}(\ell) = 1\})$ and use the obtained solution α as a starting point for the solution-improving phase. In our preliminary experiments, we observed that the solution α obtained this way often is quite poor, i.e. has a high $\alpha(O^R)$. An intuitive explanation for this is that most core-guided MaxSAT algorithms encode the semantics of the new variables in O^R as implications rather than equivalences. As a consequence, there are several extensions of α_{best} into solutions of $F \cup \textsc{card}$ that all have a cost wrt O^R that is at least $\alpha_{best}(O)$ and often much higher. A SAT solver does not differentiate between these extensions and can return any of them. Our solution is to invoke an SLS algorithm on $(F \cup \textsc{card}, O^R)$ using α as a starting point. This allows the SLS algorithm to search not only over the extensions of α_{best}, but also improve on α_{best} itself.

▶ **Example 9.** Consider the instance (F, O) from Example 7 and assume the best assignment α_{best} found so far assigns $\alpha_{best}(x_1) = \alpha_{best}(b_2) = 0$ and $\alpha_{best}(b_1) = 1$. If the core-guided phase finds the core $(b_1 \vee b_2)$ it will form the new objective $O^R = 2b_1 + 2o_C^1$ and new clauses $\textsc{card} = \textsc{as-cnf}((b_1 + b_2 > 1) \to o_C^1)$. Invoking a SAT solver on $F \cup \textsc{card}$ with the assumptions $\{\bar{x_1}, \bar{b_2}, b_1\}$ can obtain the solution α that assigns $\alpha(o_C^1) = 1$, even though a solution that assigns $o_C^1 = 0$ is also a valid extension.

3.3 SLS During the Solution-Improving Phase

In contrast to the core-guided phase, the solution-improving phase of core-boosted linear search focuses on finding solutions of clauses in $F \cup \textsc{card}$ under different precisions (cf. Section 2.3) of O^R. An important intuition for integrating SLS with the phase is that the number of clauses required for encoding an objective improving constraint $\lfloor O^R/\rho \rfloor < B$ of the objective under a precision ρ depends on the bound B. Thus, a low-cost (wrt $\lfloor O^R/\rho \rfloor$) solution can be used to decrease the number of clauses that need to be added to the SAT solver in the solution-improving phase and thus make the SAT calls faster to solve. To obtain such a solution, we invoke SLS on the instance $(F \cup \textsc{card}, \lfloor O^R/\rho \rfloor)$ once for every precision considered during the phase. If the SLS algorithm finds a solution of cost 0, the SAT solver is never invoked, instead the ρ is lowered and the SLS algorithm invoked again.

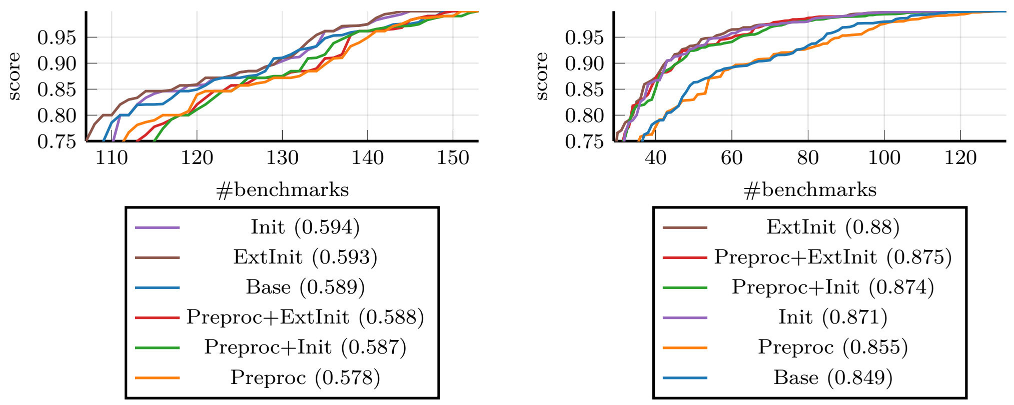
Note, that these more frequent SLS calls can be performed without reinitializing the SLS solver. After the core-guided phase, we initialize the solver once with the instance $(F \cup \textsc{card}, O^R)$. We directly modify the weights inside the SLS solver to apply the increasing precisions.

▶ **Example 10.** Consider the instance (F, O) from Example 7. Assume that the first precision considered in the solution-improving phase is 4. Then $\lfloor O/4 \rfloor = b_1$, initially, only one objective variable is being considered. Executing SLS on $(F, \lfloor O/4 \rfloor)$, could yield an assignment that sets $x_1 = \textsc{true}$, which is a 0-cost solution for $\lfloor O/4 \rfloor$, and thus trivially an optimal solution for $\lfloor O/4 \rfloor$.

4 Experimental Evaluation

We implemented the techniques outlined in Section 3 in the anytime core-boosted linear search solver Loandra from the MaxSAT Evaluation (MSE) 2024 [10, 11]. This version of Loandra uses the SAT solver Glucose 4.1 [4] and runs the local search component of the NuWLS-c solver [17] on the full objective-based instance at the start of each new precision level during solution improving search.

The baseline solver in our evaluation, **Base**, is the 2024 MSE version of Loandra with NuWLS-c exchanged for BouMS 2.1.0 [34] and Glucose for CaDiCaL 2.0 [13]. Since the current experimental evaluation focuses on the algorithmic ideas rather than specific parameter



(a) unweighted. **(b)** weighted.

Figure 2 Anytime scores for different variants of running SLS before core-boosted linear search.

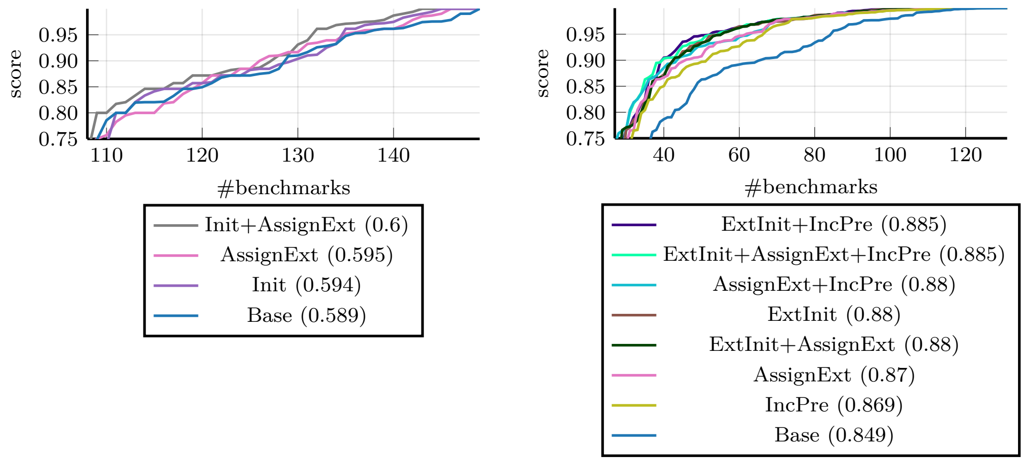
values, we set the parameters of **Base** to equal those used in previous work [17, 11, 42]; the `max-flips` of BouMS was set to 10^7, the preprocessing phase and the core-guided phase are both invoked for a maximum of 30 seconds, the precision for the solution-improving phase was set to 10, and the solution-improving constraints realized with the dynamic polynomial watchdog encoding. The solver implementing all of our techniques is available in open source at: `https://doi.org/10.5281/zenodo.15584837`.

In our experiments, we use the 216 unweighted and 229 weighted instances from the anytime track of the MSE 2024 (`https://maxsat-evaluations.github.io/2024/benchmarks.html`) [12]. We employ the same comparison metric as the MSE, the average (over all instances) of the anytime score that for a solver s on an instance i is calculated as $\frac{1+\text{ score of best known cost for } i}{1+\text{ cost of solution for } i \text{ found by } s}$. The score is 1 if the solver finds a solution whose cost matches the best-known, and 0 if no solution is found. All evaluations were performed on 2.40-GHz Intel Xeon Gold 6148 machines in RHEL under a per-instance 32-GiB memory limit and 15-minute time limit. The choice of a larger (compared to the 5 minutes used in the MaxSAT Evaluations) time limit is motivated by previous work [43] demonstrating that involved search techniques in anytime MaxSAT tend to require a slightly larger time limit to be beneficial.

We first assess the impact of running SLS before the core-guided phase (Section 4.1), followed by an evaluation of running SLS for assignment extension and on different precision levels (Section 4.2). Then we examine the impact of using cores with the previous techniques (Section 4.3). Finally, we compare the resulting solver to the leading state-of-the-art solvers from the MSE 2024 (Section 4.4).

4.1 Impact of Running SLS on the Clause Based Instance

Our first experiment investigates different ways of invoking an SLS solver at the start of the search. For this, we consider two variants (described in more detail in Section 3.1) of the **Base** solver. The `Init` solver invokes SLS on the objective-based instance obtained after the initial conversion. This corresponds to the most common way in which modern SAT-based anytime solvers already use SLS in their search. The `ExtInit` solver extends the `Init` configuration by additionally invoking SLS on the original clause-based instance. Remember



(a) unweighted. **(b)** weighted.

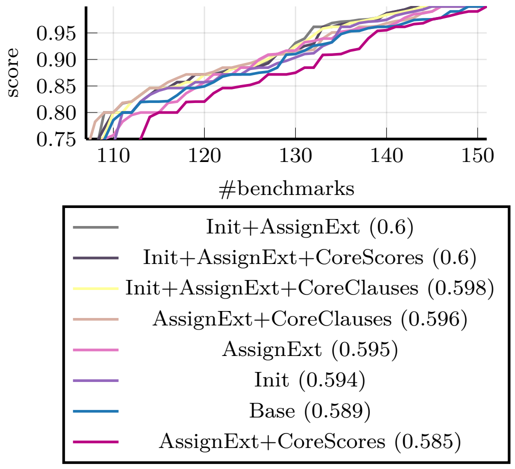
Figure 3 Anytime scores comparing variants of using SLS-enhanced assignment extension and increasing precision.

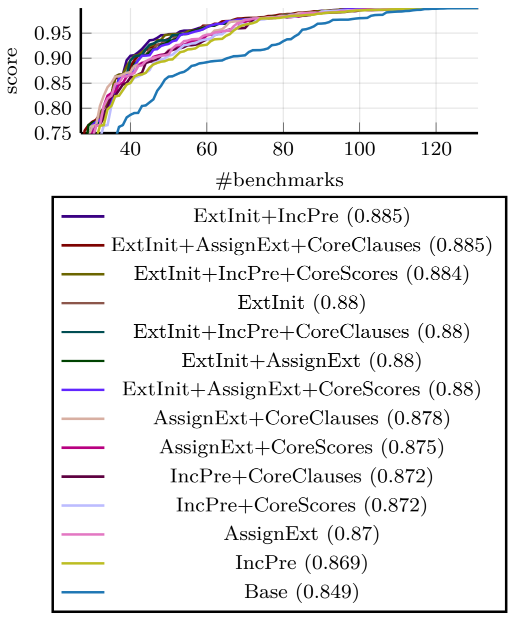
that, when preprocessing (which we denote by `Preproc`) is applied in the conversion from clause-based to objective-based, there can be a significant difference between the clause-based and the objective-based instance.

Figure 2 shows the results of the experiments. On unweighted instances, `Preproc` decreases the performance of **Base**, but `Init` improves the average score by 0.005. `ExtInit`, however, is not as effective as `Init` only. On the weighted instances, `ExtInit` achieves a score 0.009 higher than `Init`, and 0.031 higher than **Base**. In contrast to the unweighted instances, `Preproc` improves the performance of **Base** and `Init`, yet still degrades that of `ExtInit` by 0.005. We conclude that, on weighted instances, invoking an SLS search prior to the search of a SAT-based algorithm can perform noticeably better on the clause-based instance than on the objective-based one to the extent of being more effective than running preprocessing.

4.2 Impact of SLS-Enhanced Assignment Extension and Increasing Precision

For unweighted instances, the linear search phase only has one precision level. Therefore, using SLS to extend a solution to the relaxed instance after the core-guided phase (`AssignExt` as described in Section 3.2) is practically equivalent to running SLS at the start of each precision level (`IncPre` as described in Section 3.3). Hence, we do not evaluate the latter on unweighted instances. Figure 3 shows the results of applying the two techniques in isolation, in combination with each other, and in combination with the best initial SLS routine for each of the two benchmark sets as determined in Section 4.1. `AssignExt` improves the score by 0.006 for unweighted instances and profits from being paired with `Init`; the combination of the techniques yields an improvement of 0.011. `AssignExt`, in isolation, also improves the score on the weighted instances (by 0.021), and performs similarly to `IncPre`. However, when paired with `ExtInit`, `IncPre` outperforms every other possible combination, and yields an improvement of 0.036 compared to **Base**.





(a) unweighted. **(b)** weighted.

Figure 4 Anytime scores comparing the two variants of using cores.

In conclusion, both SLS integration techniques evaluated in this subsection improve the performance of **Base**, and yield the highest scores when paired with running SLS during the initial phase. In such a pairing, on weighted instances, however, `IncPre` outperforms `AssignExt`.

4.3 Impact of Using Cores in SLS

Figure 4 shows the effect of adding cores as clauses (`CoreClauses`) and using them to modify the SLS weighting scheme (`CoreScores`), as described in Section 3.2. For `CoreScores`, the `core-factor` parameter is set to 1. Both `AssignExt` and `IncPre` benefit from `CoreClauses`, improving the score of `AssignExt` by 0.001 on unweighted instances and by 0.008 on weighted instances. The score of `IncPre` is improved by 0.003. The `CoreScores` variant is not effective on unweighted instances, but on weighted instances its performance is comparable to `CoreClauses` when combined with `IncPre`, and slightly worse in conjunction with `AssignExt`.

The benefits of these (simplistic) approaches of using cores are overshadowed by invoking SLS at the beginning of the search; `Init+AssignExt` remains the best variant for unweighted instances and does not benefit from cores. On weighted instances, `ExtInit+ IncPre` wins, and adding a core strategy reduces the score. Yet, `CoreClauses` can help `ExtInit+ AssignExt` to achieve performance equivalent to `ExtInit+ IncPre`.

In conclusion, while SLS can benefit from the additional information generated in the core-guided phase, employing SLS before the search is more effective. Adding the cores as clauses yields higher scores than the weighting scheme modification introduced in this paper. Yet, both perform almost equivalently on weighted instances, making us carefully optimistic that more sophisticated weighting schemes based on cores could be developed.

4.4 Comparison to State-of-the-Art Solvers

Finally, we note that we did also compare our **Base** solver to the best-performing anytime solvers of the 2024 MSE (SPB_MaxSAT_Band and SPB_MaxSAT_FPS [28]) as well as Loandra as it participated in the 2024 MSE. Detailed results are presented in Appendix A, but the takeaway message is that the best-performing variant of our **Base** solver (`Init+AssignExt` for the unweighted instances and `ExtInit+IncPre` for the weighted) were still outperformed by the 2024 MSE version of Loandra by an average score of 0.018 on unweighted and 0.010 on weighted instances. To look into this, we ran the same experiments with a version of **Base** that uses Glucose as a backend. Without preprocessing, the conclusions with glucose as the backend were very similar to those presented in previous sections. When preprocessing with MaxPRE [29] was turned on, the performance of the solver improved significantly to the extent that any benefits from the SLS integration techniques disappeared. In short, we conclude that variants of Loandra that use Glucose as the backend benefit much more from preprocessing before search than from tighter SLS integration. In contrast, preprocessing has a significantly smaller impact when CaDiCaL is the backend, allowing us to observe benefits from SLS integration regardless. Currently, the best performance of Loandra is achieved with Glucose as the backend and preprocessing.

All in all, our results show that more sophisticated methods of integrating SLS with core-boosted linear search can result in increased performance, but the devil is in the details, and developing universally effective methods is a significant research challenge. We also note that the techniques we describe can be expected to work with any SLS approach for MaxSAT and that the approach implemented in BouMS is not state-of-the-art.

5 Conclusions

We studied the potential synergy between stochastic local search and CDCL-based reasoning for computing low-cost solutions to MaxSAT instances. Focusing on the so-called core-boosted linear search algorithm that makes use of many different CDCL-based optimization algorithms and cost-preserving reasoning, we identified several ways an SLS algorithm can enrich the reasoning performed by the SAT-based search. In addition, we studied ways a SAT-based algorithm can provide non-trivial information about the instance being solved in the form of unsatisfiable cores. Our experimental evaluation of the ideas demonstrated some support for the potential benefits of sophisticated methods of integrating SLS with CDCL-based reasoning in the context of MaxSAT. Further interesting work includes studying more sophisticated ways in which cores can be utilized in SLS.

References

1 Carlos Ansótegui, Maria Luisa Bonet, Joel Gabas, and Jordi Levy. Improving WPM2 for (Weighted) Partial MaxSAT. In *Principles and Practice of Constraint Programming*, volume 8124 of *LNCS*, pages 117–132. Springer, 2013. `doi:10.1007/978-3-642-40627-0_12`.

2 Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. SAT-based MaxSAT algorithms. *Artificial Intelligence*, 196:77–105, 2013. `doi:10.1016/j.artint.2013.01.002`.

3 Carlos Ansótegui and Joel Gabàs. WPM3: An (in)complete algorithm for weighted partial MaxSAT. *Artificial Intelligence*, 250:37–57, 2017. `doi:10.1016/j.artint.2017.05.003`.

4 Gilles Audemard and Laurent Simon. Predicting Learnt Clauses Quality in Modern SAT Solvers. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 399–404, 2009. URL: `https://www.ijcai.org/Proceedings/09/Papers/074.pdf`.

5 Florent Avellaneda. EvalMaxSAT 2024. In *MaxSAT Evaluation 2024: Solver and Benchmark Descriptions*, page 8. University of Helsinki, Department of Computer Science, 2024.

6 Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. Maximum Satisfiability. In *Handbook of Satisfiability*, volume 336 of *FAIA*, pages 929–991. IOS Press, 2021. `doi:10.3233/FAIA201008`.

7 Tomáš Balyo, Armin Biere, Markus Iser, and Carsten Sinz. SAT Race 2015. *Artificial Intelligence*, 241:45–65, 2016. `doi:10.1016/j.artint.2016.08.007`.

8 Anton Belov, António Morgado, and Joao Marques-Silva. SAT-Based Preprocessing for MaxSAT. In *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 8312 of *LNCS*, pages 96–111. Springer, 2013. `doi:10.1007/978-3-642-45221-5_7`.

9 Jeremias Berg. The Loandra MaxSAT solver. Software, swhId: `swh:1:dir:6babae3bfbb34e637ca99fab6b5f3a85f2263476` (visited on 2025-07-23). URL: `https://github.com/jezberg/loandra/tree/boums-cadical-integration`, `doi:10.4230/artifacts.24096`.

10 Jeremias Berg, Emir Demirović, and Peter J. Stuckey. Core-Boosted Linear Search for Incomplete MaxSAT. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, volume 11494 of *LNCS*, pages 39–56. Springer, 2019. `doi:10.1007/978-3-030-19212-9_3`.

11 Jeremias Berg, Christoph Jabs, Hannes Ihalainen, and Matti Järvisalo. Loandra in the 2024 MaxSAT Evaluation. In *MaxSAT Evaluation 2024: Solver and Benchmark Descriptions*, pages 9–10. University of Helsinki, Department of Computer Science, 2024.

12 Jeremias Berg, Matti Järvisalo, Ruben Martins, Andreas Niskanen, and Tobias Paxian. *MaxSAT Evaluation 2024: Solver and Benchmark Descriptions.* University of Helsinki, Department of Computer Science, 2024. URL: `http://hdl.handle.net/10138/584878`.

13 Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. CaDiCaL 2.0. In *Computer Aided Verification 36th International Conference*, volume 14681 of *LNCS*, pages 133–152. Springer, 2024. `doi:10.1007/978-3-031-65627-9_7`.

14 Shaowei Cai. Balance between Complexity and Quality: Local Search for Minimum Vertex Cover in Massive Graphs. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, pages 747–753, 2015. URL: `https://www.ijcai.org/Proceedings/15/Papers/111.pdf`.

15 Shaowei Cai, Xindi Zhang, Mathias Fleury, and Armin Biere. Better Decision Heuristics in CDCL through Local Search and Target Phases. *Journal of Artificial Intelligence Research*, 74:1515–1563, 2022. `doi:10.1613/jair.1.13666`.

16 Xiamin Chen, Zhendong Lei, and Pinyan Lu. Deep Cooperation of Local Search and Unit Propagation Techniques. In *30th International Conference on Principles and Practice of Constraint Programming*, volume 307 of *LIPIcs*, pages 6:1–6:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. `doi:10.4230/LIPIcs.CP.2024.6`.

17 Yi Chu, Shaowei Cai, and Chuan Luo. NuWLS-c-2023: Solver Description. In *MaxSAT Evaluation 2023: Solver and Benchmark Descriptions*, pages 23–24. University of Helsinki, Department of Computer Science, 2023.

18 Yi Chu, Shaowei Cai, and Chuan Luo. NuWLS: Improving Local Search for (Weighted) Partial MaxSAT by New Weighting Techniques. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 3915–3923, 2023. `doi:10.1609/aaai.v37i4.25505`.

19 Arnaud De Coster, Nysret Musliu, Andrea Schaerf, Johannes Schoisswohl, and Kate Smith-Miles. Algorithm Selection and Instance Space Analysis for Curriculum-Based Course Timetabling. *Journal of Scheduling*, 25(1):35–58, 2022. `doi:10.1007/s10951-021-00701-x`.

20 Emir Demirović and Peter J. Stuckey. Techniques Inspired by Local Search for Incomplete MaxSAT and the Linear Algorithm: Varying Resolution and Solution-Guided Search. In *Principles and Practice of Constraint Programming*, volume 11802 of *LNCS*, pages 177–194. Springer, 2019. `doi:10.1007/978-3-030-30048-7_11`.

21 Jo Devriendt, Stephan Gocht, Emir Demirović, Jakob Nordström, and Peter J. Stuckey. Cutting to the Core of Pseudo-Boolean Optimization: Combining Core-Guided Search with

Cutting Planes Reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3750–3758, 2021. doi:10.1609/aaai.v35i5.16492.

22 Niklas Eén and Niklas Sörensson. Temporal Induction by Incremental SAT Solving. In *BMC'2003: First International Workshop on Bounded Model Checking*, volume 89 of *ENTCS*, pages 543–560. Elsevier, 2005. doi:10.1016/S1571-0661(05)82542-3.

23 Graeme Gange, Jeremias Berg, Emir Demirović, and Peter J. Stuckey. Core-Guided and Core-Boosted Search for CP. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, volume 12296 of *LNCS*, pages 205–221. Springer, 2020. doi:10.1007/978-3-030-58942-4_14.

24 Hao Hu, Mohamed Siala, Emmanuel Hebrard, and Marie-José Huguet. Learning Optimal Decision Trees with MaxSAT and its Integration in AdaBoost. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence*, pages 1170–1176, 2020. doi:10.24963/ijcai.2020/163.

25 Hannes Ihalainen, Jeremias Berg, and Matti Järvisalo. Clause Redundancy and Preprocessing in Maximum Satisfiability. In *Automated Reasoning*, volume 13385 of *LNCS*, pages 75–94. Springer, 2022. doi:10.1007/978-3-031-10769-6_6.

26 Christoph Jabs, Jeremias Berg, and Matti Järvisalo. Core Boosting in SAT-Based Multi-Objective Optimization. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, volume 14743 of *LNCS*. Springer, 2024. doi:10.1007/978-3-031-60599-4_1.

27 Menghua Jiang and Yin Chen. NuWLS-c-IBR: Solver Description. In *MaxSAT Evaluation 2024: Solver and Benchmark Descriptions*, page 22. University of Helsinki, Department of Computer Science, 2024.

28 Mingming Jin, Kun He, Jiongzhi Zheng, Jinghui Xue, and Zhuo Chen. Combining Band-MaxSAT and FPS with SPB-MaxSAT-c. In *MaxSAT Evaluation 2024: Solver and Benchmark Descriptions*, pages 23–24. University of Helsinki, Department of Computer Science, 2024.

29 Tuukka Korhonen, Jeremias Berg, Paul Saikko, and Matti Järvisalo. MaxPre: An Extended MaxSAT Preprocessor. In *Theory and Practice of Satisfiability Testing – SAT 2017*, volume 10491 of *LNCS*, pages 449–456. Springer, 2017. doi:10.1007/978-3-319-66263-3_28.

30 Zhendong Lei and Shaowei Cai. SATLike-c: Solver Description. In *MaxSAT Evaluation 2018: Solver and Benchmark Descriptions*, page 24. University of Helsinki, Department of Computer Science, 2018.

31 Zhendong Lei and Shaowei Cai. SATLike-c(w): Solver Description. In *MaxSAT Evaluation 2020: Solver and Benchmark Descriptions*, page 15. University of Helsinki, Department of Computer Science, 2020.

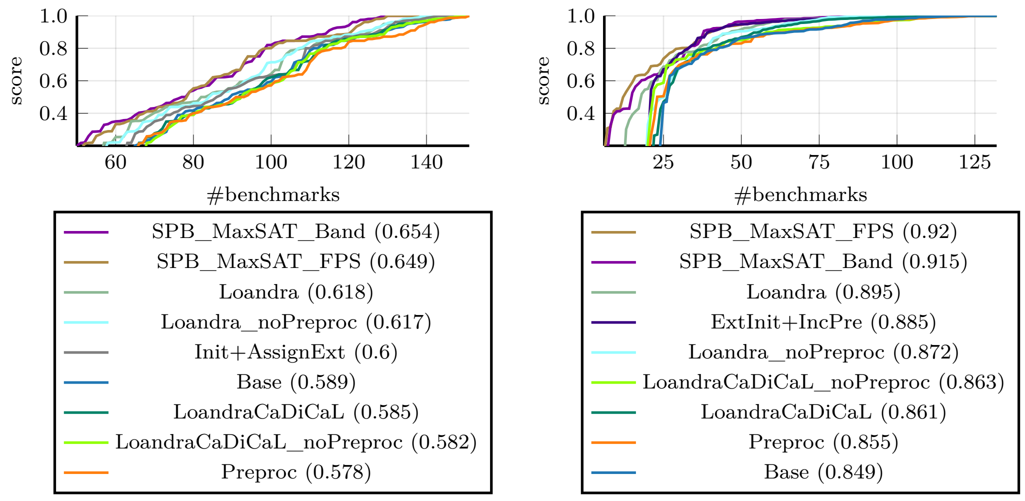
32 Zhendong Lei, Shaowei Cai, Fei Geng, Dongxu Wang, Yongrong Peng, Dongdong Wan, Yiping Deng, and Pinyan Lu. SATLike-c: Solver Description. In *MaxSAT Evaluation 2021: Solver and Benchmark Descriptions*, pages 19–20. University of Helsinki, Department of Computer Science, 2021.

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

34 Ole Lübke. BouMS. Zenodo, March 2025. doi:10.5281/zenodo.14999498.

35 João Marques-Silva, Inês Lynce, and Sharad Malik. Conflict-Driven Clause Learning SAT Solvers. In *Handbook of Satisfiability*, volume 336 of *FAIA*, pages 133–182. IOS Press, 2021. doi:10.3233/FAIA200987.

36 Antonio Morgado, Carmine Dodaro, and Joao Marques-Silva. Core-Guided MaxSAT with Soft Cardinality Constraints. In *Principles and Practice of Constraint Programming*, volume 8656 of *LNCS*, pages 564–573. Springer, 2014. doi:10.1007/978-3-319-10428-7_41.



(a) unweighted. **(b)** weighted.

Figure 5 Anytime scores comparing the best performing configuration to other state-of-the-art solvers.

37 António Morgado, Federico Heras, Mark H. Liffiton, Jordi Planes, and João Marques-Silva. Iterative and core-guided MaxSAT solving: A survey and assessment. *Constraints*, 18(4):478–534, 2013. `doi:10.1007/s10601-013-9146-2`.

38 Alexander Nadel. Polarity and Variable Selection Heuristics for SAT-Based Anytime MaxSAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 12(1):17–22, 2020. `doi:10.3233/SAT-200126`.

39 Alexander Nadel. TT-Open-WBO-Inc-24: An Anytime MaxSAT Solver Entering MSE'24. In *MaxSAT Evaluation 2024: Solver and Benchmark Descriptions*, page 21. University of Helsinki, Department of Computer Science, 2024.

40 Alexander Nadel. TT-Open-WBO-Inc: An Efficient Anytime MaxSAT Solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 15(1):1–7, 2024. `doi:10.3233/SAT-231504`.

41 Nina Narodytska and Fahiem Bacchus. Maximum Satisfiability Using Core-Guided MaxSAT Resolution. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014. `doi:10.1609/aaai.v28i1.9124`.

42 Tobias Paxian, Sven Reimer, and Bernd Becker. Dynamic Polynomial Watchdog Encoding for Solving Weighted MaxSAT. In *Theory and Applications of Satisfiability Testing*, volume 10929 of *LNCS*, pages 37–53. Springer, 2018. `doi:10.1007/978-3-319-94144-8_3`.

43 André Schidler and Stefan Szeider. Structure-guided local improvement for maximum satisfiability. In *CP*, volume 307 of *LIPIcs*, pages 26:1–26:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. `doi:10.4230/LIPICS.CP.2024.26`.

44 Pouya Shati, Eldan Cohen, and Sheila A. McIlraith. SAT-based Optimal Classification Trees for Non-Binary Data. *Constraints*, 28(2):166–202, 2023. `doi:10.1007/s10601-023-09348-1`.

45 Alice E. Smith and David W. Coit. Penalty Functions. In *Handbook of Evolutionary Computation*, pages C5.2:1–C5.2:6. IOP Publishing and Oxford University Press, 1997.

A Detailed Comparison to State-of-the-Art Solvers

Figure 3 shows a comparison of the best combinations of techniques described in this paper to the winners of the MSE 2024, SPB_MaxSAT_Band and SPB_MaxSAT_FPS [28], and Loandra as it participated in the same MSE. To assess the performance of our resulting solvers we include variations of Loandra with sophisticated preprocessing disabled (_noPreproc) and with Glucose exchanged for CaDiCaL (LoandraCaDiCaL). Keep in mind, that Preproc (Base) is LoandraCaDiCaL(_noPreproc) without any SLS.

We can see that Loandra performs worse with CaDiCaL than with Glucose, and preprocessing is also much less effective, or even detrimental for this variant. Given the reduced performance of Base, it is no surprise that our best configurations cannot outperform the best anytime solvers currently available. Still, the substantial improvements over Base show that the techniques described in this paper effectively improve the performance of anytime core-boosted linear search.