# Integer Linear Programming Preprocessing for Maximum Satisfiability


**Jialu Zhang**
Laboratoire MIS UR 4290, Université de Picardie Jules Verne, Amiens, France

**Chu-Min Li**
Laboratoire MIS UR 4290, Université de Picardie Jules Verne, Amiens, France

**Sami Cherif**
Laboratoire MIS UR 4290, Université de Picardie Jules Verne, Amiens, France

**Shuolin Li**
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

**Zhifei Zheng**
Laboratoire MIS UR 4290, Université de Picardie Jules Verne, Amiens, France



## Abstract

The Maximum Satisfiability problem (MaxSAT) is a major optimization challenge with numerous practical applications. In recent MaxSAT evaluations, most MaxSAT solvers have adopted an ILP solver as part of their portfolios. This paper investigates the impact of Integer Linear Programming (ILP) preprocessing techniques on MaxSAT solving. Experimental results show that ILP preprocessing techniques help WMaxCDCL-OpenWbo1200, the winner of the MaxSAT evaluation 2024 in the unweighted track, solve 15 additional instances. Moreover, current state-of-the-art MaxSAT solvers heavily use an ILP solver in their portfolios, while our proposed approach reduces the need to call an ILP solver in a portfolio including WMaxCDCL or MaxCDCL.



**2012 ACM Subject Classification** Theory of computation → Constraint and logic programming

**Keywords and phrases** Maximum Satisfiability, ILP, Preprocessing.

**Digital Object Identifier** 10.4230/LIPIcs.SATCPDP.2025.23


## 1 Introduction

Maximum Satisfiability (MaxSAT) is a natural optimization extension of the Propositional Satisfiability problem (SAT) [7]. While SAT consists of determining an assignment that satisfies the clausal constraints in a given formula under Conjunctive Normal Form (CNF), the goal in MaxSAT shifts to finding a solution satisfying the maximum number of clauses in the formula. MaxSAT is harder to solve than SAT in theory and practice because it is more difficult to find and prove optimal solutions [6, 14]. Many real-world optimization problems can be formulated as MaxSAT instances, including scheduling [26], hardware and software debugging [25], explainable artificial intelligence [11], and so on.

Algorithms for solving the MaxSAT problem can be broadly classified into exact algorithms and heuristic algorithms. Exact algorithms, such as SAT-based (e.g., RC2 [12]), Branch and Bound (e.g., MaxCDCL [16]), and Integer Linear Programming (ILP), find the optimal solution and prove its optimality. In contrast, heuristic algorithms, such as stochastic local search [24], can also be competitive, but they do not guarantee optimality. It is known that ILP solvers, while they perform well on certain families of instances, are not competitive for most industrial and random instances [3]. Therefore, the common practice observed in recent MaxSAT evaluations[1], particularly for the most efficient solvers, is to combine ILP solvers in a

[1] https://maxsat-evaluations.github.io/



Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

portfolio with other types of solvers to solve MaxSAT instances. For example, in the MaxSAT evaluation 2024 [1], the total time limit to solve an instance is 3600s; EvalMaxSAT [5] first runs the ILP solver SCIP [2] for 400s and then itself for 3200s; UWrMaxSat [19] runs SCIP and itself alternatively each with a possibly different time limit, and compares its upper and lower bounds with SCIP to improve them. As such, in all these portfolio MaxSAT solvers taking advantage of ILP, the ILP solver is typically used independently in a portfolio setting, requiring careful heuristic tuning, such as setting specific time limits.

In this paper, we propose a more integrated approach, where an ILP solver is used as a preprocessing step, fully incorporated into the solving pipeline. Our approach consists of first reading the CNF formula to convert it using integer linear constraints with an objective function, then the ILP solver is called to simplify the problem, finally, the simplified integer linear constraints are re-encoded into CNF to be solved using a MaxSAT solver. Experimental results show that this approach allows solving more instances than the current state-of-the-art MaxSAT solvers. Note that our approach does not require setting a heuristic time limit for ILP solvers, allowing them to run until they fully preprocess the instance, which is different from other approaches, such as EvalMaxSAT and UWrMaxSat.

The remainder of this paper is organized as follows. Section 2 introduces the MaxSAT problem and ILP preprocessing techniques. Section 3 presents the methodology for integrating ILP-based preprocessing techniques into MaxSAT solvers. Section 4 discusses our experimental results. Finally, we conclude and discuss future work in Section 5.

## 2 Preliminaries

### 2.1 Maximum Satisfiability

Given a set of Boolean variables, a literal $l$ is either a variable $x$ or its negation $\neg x$, a clause $c$ is a disjunction of literals and can be represented as a set of literals. A formula $F$ in Conjunctive Normal Form (CNF) is a conjunction of clauses. A variable $x$ is assigned if it takes a value in $\{True, False\}$ (i.e., $\{1, 0\}$). A literal $x$ ($\neg x$) is assigned to $True$ ($False$) if variable $x$ is assigned $True$, and to $False$ ($True$) otherwise. A clause $c$ is satisfied if at least one of its literals is assigned to $True$. A formula $F$ is satisfied if all its clauses are satisfied. The SAT problem consists of finding an assignment that satisfies a given CNF formula $F$ [7].

MaxSAT is an optimization extension of SAT (more natural than MinSAT, another optimization extension of SAT [10]), encompassing both Partial MaxSAT and Weighted Partial MaxSAT [6, 14]. Partial MaxSAT divides clauses into *hard* clauses $H$ and $soft$ clauses $S$, i.e., $F = H \cup S$, and the goal is to find an assignment that satisfies all hard clauses in $H$ while maximizing the number of satisfied soft clauses in $S$. In Weighted Partial MaxSAT, a soft clause $c \in S$ can be falsified with an integer penalty $w_c$, also called the weight of $c$. The objective for Weighted Partial MaxSAT is thus to find an optimal assignment that maximizes the sum of weights of satisfied soft clauses while satisfying all the hard clauses.

### 2.2 Preprocessing Techniques

Preprocessing in problem-solving usually amounts to transforming a given instance into an equivalent one that would potentially be easier to solve. In ILP solvers, preprocessing techniques are a key factor to speed up problem-solving, which includes variable fixing, variable aggregation, redundant constraint elimination, and other advanced inference mechanisms [22]. The variable fixing technique employs a probing algorithm that temporarily assigns a binary variable to 0 or 1 and then propagates the resulting implications [2]. Variable aggregation exploits equations and constraint relationships within the model, as well as cluster or symmetry detection algorithms, to merge multiple variables into a single one. Meanwhile, redundant constraint elimination checks the bounds of each constraint, removes constraints that are proved to be satisfied by all variable values satisfying other constraints, or detects constraints implying infeasibility of the problem [23].

In SAT and MaxSAT solvers, preprocessing techniques are broadly used to reduce the number of variables and clauses, such as bounded variable elimination (BVE), failed literal detection, unit propagation, and self-subsuming resolution [4, 8]. Clause vivification [15] can also be used as preprocessing or inprocessing among hard clauses. There are also tools, such as MaxPre [13], that integrate SAT and MaxSAT preprocessing techniques into a program.

A MaxSAT problem can be naturally converted into an ILP problem. Equations (1)-(4) give an ILP model for the weighted partial MaxSAT problem $F = H \cup S$, where $H$ ($S$) is the set of hard (soft) clauses, respectively. $V$ is the set of boolean decision variables in $F$. A binary variable $z_c$ is introduced for every soft clause $c \in S$ with weight $w_c$, a binary variable $y_x$ is introduced for each Boolean variable $x$ in $V$, and a hard (soft) clause $c$ is written as $H_c^- \vee H_c^+$ ($S_c^- \vee S_c^+$), where $H_c^-$ ($H_c^+$) is a disjunction of negative (positive) literals. Equation (2) ensures that every hard clause is satisfied, and Equation (3) entails that if a soft clause $c$ is satisfied, then its weight $w_c$ can contribute to the objective function. However, the encoding of an ILP problem into MaxSAT is not so straightforward, as many ILP constraints require sophisticated techniques to be efficiently encoded into MaxSAT. Fortunately, tools such as the PBLib Library [18] have been developed in the literature to facilitate these transformations by providing efficient encoding techniques.

$$\textbf{Objective:} \quad \text{Maximize} \sum_{c \in S} w_c \cdot z_c \tag{1}$$

$$\textbf{Subject to:} \quad \sum_{x \in H_c^+} y_x + \sum_{x \in H_c^-} (1 - y_x) \geq 1, \quad \forall c \in H \tag{2}$$

$$z_c \leq \sum_{x \in S_c^+} y_x + \sum_{x \in S_c^-} (1 - y_x), \quad \forall c \in S \tag{3}$$

$$z_c \in \{0, 1\}, \quad \forall c \in S; \quad y_x \in \{0, 1\}, \quad \forall x \in V \tag{4}$$

# 3 Preprocessing a MaxSAT instance through ILP

We propose a methodology to integrate ILP preprocessing techniques into MaxSAT solvers. This section presents our methodology as well as the variable and constraint encodings.

## 3.1 Methodology

Our three-stage methodology can be described as follows:

1. **Preprocessing Stage:** Given a MaxSAT instance (*originInst*), an ILP model (*originModel*) is constructed based on Equations (1) to (4). Preprocessing techniques are then applied to *originModel* using an ILP solver, yielding a hopefully simplified model (*simpModel*). The *simpModel* is subsequently encoded into a simplified MaxSAT instance (*simpInst*), while the mapping between variables in *originInst* and *simpInst* is recorded in *varMap*.
2. **Solving Stage:** If *simpInst* is "smaller" than *originInst*, i.e., if *simpInst* contains fewer variables *and* fewer hard clauses than *originInst*, a MaxSAT solver is applied to solve *simpInst* to obtain an optimal solution (*simpSol*) of *simpInst*. Otherwise, the

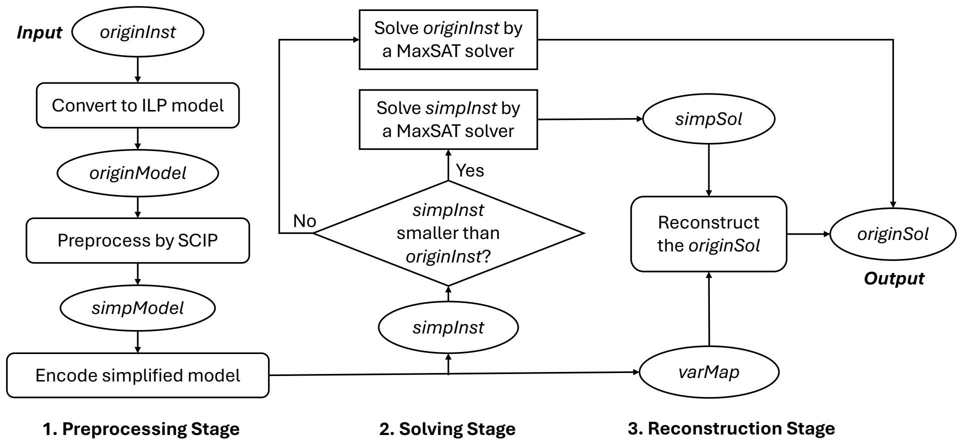



**Figure 1** Integrating ILP preprocessing techniques into MaxSAT solvers

original instance ($originInst$) is solved by the MaxSAT solver. However, the definition of "smaller" is debatable and deserves future study.

3. **Reconstruction Stage:** In this stage, the algorithm constructs an optimal solution $originSol$ for $originInst$ with $simpSol$ and $varMap$. This stage happens only when $simpInst$ is "smaller" than $originInst$.

The three stages are illustrated in Figure 1, in which the key aspect is to convert $simpModel$ into $simpInst$. We first check the variables and constraints in $simpModel$ and then try to encode them to MaxSAT. The encoding involves mapping variables from $simpModel$ to $simpInst$, encoding constraints as hard clauses, and representing the objective function as soft clauses. The details are described in the following subsections.

## 3.2 Variable Encoding

After preprocessing by an ILP solver, the original ILP model ($originModel$) encoding the original MaxSAT instance is transformed into $simpModel$, in which we distinguish three types of binary variables: fixed, aggregated, and free. A fixed variable in $simpModel$ means that it is assigned a fixed value because the other value is proven to falsify at least one constraint in $originModel$. Algorithm 1 records the values of the fixed variables in $varMap$ (line 4) for the reconstruction of $originSol$.

A variable $y_x$ in $simpModel$ is referred to as aggregated when there is a relation of the form $y_x = c_0 + \sum_{i=1}^{n} c_i \cdot y_i$ in $simpModel$. This entails that the value of $y_x$ depends on other variables $y_i$ for $i = 1, \ldots, n$. In the case of a simple aggregation, i.e., $n = 1$, $c_0 = 0$ and $c_1 \in \{1, -1\}$, we have $y_x = y_1$ or $y_x = -y_1$. Algorithm 1 thus traverses the aggregation chain and creates a unique new Boolean variable to represent all variables in the chain by preserving their relations (lines 8-10). For example, consider three variables in $simpModel$ with the aggregation relationships ($y_1 = \neg y_2$) and ($y_2 = y_3$). In this case, only one new Boolean variable $v_1$ is created in $simpInst$ to represent $y_1$, $y_2$ and $y_3$, by implementing the mapping $\{y_1 \to \neg v_1, y_2 \to v_1, y_3 \to v_1\}$ when transforming $simpModel$ to $simpInst$, which preserves ($y_1 = -y_2$) and ($y_2 = y_3$). Together with variable fixing, this operation often significantly reduces the number of variables in $simpInst$ w.r.t. $originInst$, as will be showcased empirically in Section 4. In the general case, Algorithm 1 encodes the aggregation constraint as a Pseudo-Boolean formula $-y_x + \sum_{i=1}^{n} c_i \cdot y_i = -c_0$ and translates it into hard

**Algorithm 1** Encoding Variables

**Require:** *originInst*, *simpModel*, *varMap*
1: **for** each variable $x$ **in** *originInst* **do**
2: $y_x \leftarrow$ corresponding variable of $x$ in *simpModel*
3: **if** $y_x$ is a fixed variable in *simpModel* **then**
4: $varMap[x] \leftarrow$ the fixed value of $y_x$ in *simpModel*
5: **else if** $y_x$ is a free variable in *simpModel* **then**
6: $varMap[x] \leftarrow$ new Boolean variable in *simpInst*
7: **else if** $y_x$ is a simple aggregated variable in *simpModel* **then**
8: $y_z \leftarrow$ final variable in the aggregation chain $//y_z$ should be a free variable
9: create $varMap[z]$ if it was not created
10: $varMap[x] \leftarrow varMap(z)$ or $\neg varMap(z)$ according to the aggregation relation
11: **else if** $y_x$ is a multiple aggregated variable in *simpModel* **then**
12: $varMap[x] \leftarrow$ create a new Boolean variable in *simpInst*
13: Encode the aggregation constraint with a Pseudo-Boolean encoding
14: **end if**
15: **end for**

clauses in *simpInst* (lines 12-13).

A variable $y_x$ is referred to as free if it is neither fixed nor aggregated. Algorithm 1 creates a new Boolean variable in *simpInst* for each free variable in *simpModel* (line 6).

## 3.3 Constraint Encoding

We use the SCIP solver [2] to preprocess *originModel* as it is an open-source mixed-integer programming solver broadly used in MaxSAT evaluations. The obtained *simpModel* usually contains various types of constraints, as listed in Table 1. *Logical OR* and *Logical AND* constraints are directly encoded into CNF. *Setppc* and *Linear* constraints are encoded into CNF using the methods for at-most-one and pseudo-Boolean constraints in the PBLib library [18], respectively. We use the default configuration in PBLib, allowing it to automatically select the most suitable encoding (such as Binary Decision Diagrams (BDD) or Adder Networks, among others) based on the properties of the constraints. The *orbitope* constraint, which arises from the symmetry-breaking technique used in SCIP preprocessing, is not listed in Table 1. Given the complexity of encoding the orbitope constraint in CNF, instances containing it are directly handled by the MaxSAT solver.

**Table 1** Encodings of different constraints in *simpModel*

| Constraint | Formula | Encoding |
|---|---|---|
| Logical OR | $\sum_{i=1}^{n} x_i \geq 1$ | $(x_1 \lor x_2 \lor ... \lor x_n)$ |
| Logical AND | $\prod_{i=1}^{n} x_i = y$ | $(y \lor \neg x_1 \lor ... \lor \neg x_n) \land \bigwedge_{i=1}^{n} (\neg y \lor x_i)$ |
| Setppc packing | $\sum_{i=1}^{n} x_i \leq 1$ | at-most-one |
| Setppc partitioning | $\sum_{i=1}^{n} x_i = 1$ | at-most-one $\land (x_1 \lor x_2 \lor ... \lor x_n)$ |
| Linear | $lhs \leq \sum_{i=1}^{n} w_i \cdot x_i \leq rhs$ | Pseudo-Boolean |

The objective function of *simpModel* is $f'_{(S)} = Maximize \sum_{c \in S} w'_c \cdot z'_c$, where $S$ is the set of soft clauses, $z'_c$ is the decision variable in *simpModel*, and $w'_c$ is the corresponding coefficient. We encode $f'_{(S)}$ into soft clauses using the following method: if a coefficient $w'_c$ of a decision variable $z'_c$ is positive, then $z'_c$ is added as a soft clause with weight $w'_c$, otherwise, $\neg z'_c$ is added with weight $-w'_c$.

## 4 Experimental Results

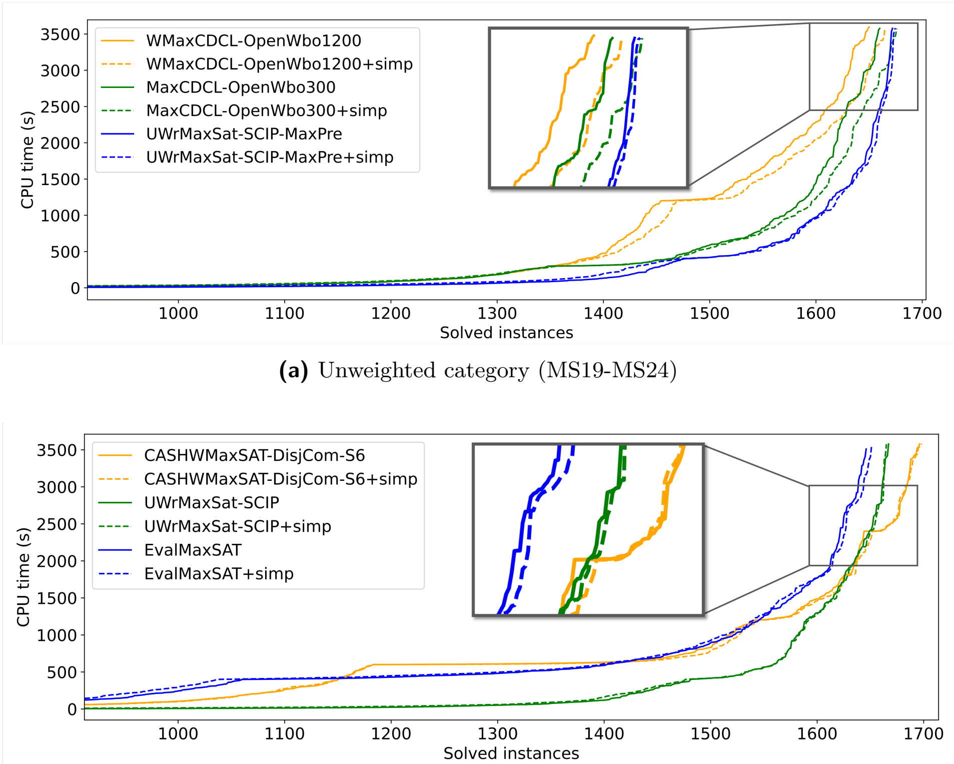
We use state-of-the-art ILP and MaxSAT solvers for our experiments. More specifically, SCIP (version 9.1.1) is used as the ILP solver for preprocessing [2]. For MaxSAT, we select the top-performing solvers from the MaxSAT evaluation 2024. In the unweighted category, the leading solvers are WMaxCDCL-OpenWbo1200 [20], MaxCDCL-OpenWbo300 [9], and UWrMaxSat-SCIP-MaxPre [19], which are ranked as the top three. In the weighted category, the top three solvers include CASHWMaxSAT-DisjCom-S6 [21], UWrMaxSat-SCIP [19], and EvalMaxSAT [5]. The benchmark MaxSAT instances are sourced from the unweighted and weighted categories of MaxSAT evaluations from 2019 to 2024 (MS19-MS24 and WMS19-WMS24, respectively). To avoid counting instances twice in the table below, we removed from (W)MS$k$ for $k > 19$ the instances also occurring in previous years from 2019. The computations are performed on an AMD EPYC 7502 Processor (2.5GHz) and 31GB of RAM under Linux. Each solver is allocated 3600 seconds to solve an instance, as in MaxSAT evaluations, including preprocessing.

Table 2 compares the number of solved instances by each solver with the SCIP preprocessing (+simp) and without. We report the number of instances solved without the SCIP preprocessing and, between parentheses, we indicate the number of additional instances solved with the preprocessing both within the same 3600s timeout. For example, MaxCDCL-OpenWbo300 solves 1660 instances in total without the SCIP preprocessing, but 1676 instances in total with the SCIP preprocessing within 3600s. Figure 2 presents the number of solved instances along with their CPU time costs for each solver.

WMaxCDCL-OpenWbo1200 (MaxCDCL-OpenWbo300) runs OpenWbo [17] for 1200s (300s) followed by WMaxCDCL (MaxCDCL) for 2400s (3300s) to solve an instance. They do not use SCIP in their portfolios. In the unweighted category, the SCIP preprocessing significantly improves their performance, allowing them to solve 15 (resp. 16) additional instances. This technique also allows MaxCDCL-OpenWbo300 to bridge the gap with

**Table 2** Number of instances solved by each solver within 3600s with (+simp) or without the SCIP preprocessing in each subset of instances. The highlighted solvers use SCIP in their portfolios.

| Unweighted category | MS19 | MS20 | MS21 | MS22 | MS23 | MS24 | **Total** |
|---|---|---|---|---|---|---|---|
| #Instances | 599 | 401 | 448 | 254 | 260 | 247 | **2209** |
| SCIP | 235 | 203 | 208 | 102 | 118 | 88 | **954** |
| WMaxCDCL-OpenWbo1200(+simp) | 446(+10) | 306(+1) | 339(+2) | 183(+1) | 177(+1) | 199(0) | **1650(+15)** |
| MaxCDCL-OpenWbo300(+simp) | 441(+10) | 315(-1) | 344(+4) | 186(+1) | 177(+1) | 197(+1) | **1660(+16)** |
| **UWrMaxSat-SCIP-Maxpre**(+simp) | 445(0) | 329(0) | 352(0) | 183(0) | 188(+2) | 175(+1) | **1672(+3)** |
| Weighted category | WMS19 | WMS20 | WMS21 | WMS22 | WMS23 | WMS24 | **Total** |
| #Instances | 586 | 433 | 491 | 291 | 218 | 204 | **2223** |
| SCIP | 228 | 239 | 238 | 122 | 117 | 101 | **1045** |
| **CASHWMaxSAT-DisjCom-S6**(+simp) | 410(+4) | 349(-1) | 405(-2) | 214(0) | 167(+1) | 151(0) | **1696(+2)** |
| **UWrMaxSat-SCIP**(+simp) | 404(-3) | 347(0) | 398(0) | 209(+1) | 163(0) | 146(0) | **1667(-2)** |
| **EvalMaxSAT**(+simp) | 390(+3) | 338(+1) | 396(-2) | 212(+1) | 162(+1) | 148(+1) | **1646(+5)** |




**(a)** Unweighted category (MS19-MS24)

**(b)** Weighted category (WMS19-WMS24)

**Figure 2** Number of solved instances vs. CPU time

UWrMaxSat-SCIP-Maxpre, by solving 1 instance more than UWrMaxSat-SCIP-Maxpre with the technique, but 12 instances less than UWrMaxSat-SCIP-Maxpre without the technique.

These results are significant because MaxSAT solving has reached a high level of maturity, making further improvements increasingly hard. In fact, in the unweighted category of the MaxSAT evaluation 2024, the winner WMaxCDCL-OpenWbo1200 solves only 2 instances more than MaxCDCL-OpenWbo300, which solves in turn 4 instances more than UWrMaxSat-SCIP-Maxpre, out of a total of 553 instances.

In the weighted category, CashWMaxSAT and UWrMaxSAT are already deeply combined with SCIP in their portfolio settings. The SCIP preprocessing does not improve or deteriorate their performance significantly. For EvalMaxSAT, which applies SCIP during 400s only to instances for which the weight of each soft clause is under 5,000,000, the effect of the SCIP preprocessing is still positive. These results suggest that the SCIP preprocessing is compatible with a portfolio running SCIP as an entire solver.

To analyse the impact of preprocessing, we partition the MaxSAT instances into 4 groups: "Smaller", "Bigger", "Failed", and "Skipped". An instance is in the "Smaller" or "Bigger" group if *simpInst* is created. If *simpInst* contains fewer variables *and* fewer hard clauses than *originInst*, the instance is in the "Smaller" group. Otherwise, it is in the "Bigger" group. An instance is in the "Failed" or "Skipped" group if *simpInst* is not created. It is in the "Failed" group if the SCIP preprocessing produces a constraint type not listed in Table 1, and in the "Skipped" group if *originInst* contains more than 200,000 variables or 1,000,000 clauses, for which the SCIP preprocessing is not performed. In our experiments, each MaxSAT solver solves the simplified instance *simpInst* only if the instance is in the "Smaller" group.

**Table 3** Statistics of four groups of instances w.r.t. the SCIP preprocessing.

| States | Unweighted category (MS19-MS24) | | | | Weighted category (WMS19-WMS24) | | | |
|---|---|---|---|---|---|---|---|---|
| | Smaller | Bigger | Failed | Skipped | Smaller | Bigger | Failed | Skipped |
| #Instances | 1085 | 436 | 182 | 506 | 980 | 508 | 201 | 534 |
| *PreprocessingTime* | 15.36s | 30.0s | 6.20s | - | 14.26s | 19.65s | 24.86s | - |
| *FixedVarsRate* | 18.66% | 3.64% | 22.81% | - | 19.61% | 16.30% | 43.48% | - |
| *AggregatedVarsRate* | 26.92% | 21.52% | 30.56% | - | 27.68% | 18.36% | 29.67% | - |
| *simpleAggregationRatio* | 99.49% | 79.54% | - | - | 99.22% | 96.64% | - | - |

**Table 4** Number of instances solved by WMaxCDCL-OpenWbo1200 (WMO+simp) or MaxCDCL-OpenWbo300 (MO+simp) with the SCIP solver as a portfolio.

| Unweighted category | MS19 | MS20 | MS21 | MS22 | MS23 | MS24 | **Sum** |
|---|---|---|---|---|---|---|---|
| WMO+simp(+S4/+S1) | 456(+4/+6) | 307(+2/+3) | 341(-2/+2) | 184(-1/0) | 178(+1/0) | 199(+1/+1) | **1665(+5/+12)** |
| MO+simp(+S4/+S1) | 451(+6/+8) | 314(+3/+4) | 348(-4/0) | 187(-2/-1) | 178(+2/+2) | 198(+2/+1) | **1676(+7/+14)** |
| Weighted category | WMS19 | WMS20 | WMS21 | WMS22 | WMS23 | WMS24 | **Sum** |
| WMO+simp(+S4/+S1) | 392(+2/+1) | 341(-2/+1) | 399(0/0) | 209(+1/+1) | 154(0/0) | 147(-1/0) | **1642(0/+3)** |

Otherwise, it is the original instance that is solved. Table 3 shows the statistics of these four groups of instances. We see that the SCIP preprocessing time is negligible compared to the total allocated time of 3600s (less than 1%), and the percentage of fixed ($FixedVarsRate$) or aggregated ($AggregatedVarsRate$) over all variables in $originModel$ is significant, and the percentage of simple aggregation variables ($simpleAggregationRatio$, see lines 8-10 of Algorithm 1) over all aggregated variables is very high (99% for the "Smaller" instances).

Finally, we investigate the relation between a portfolio running SCIP as an entire solver and the SCIP preprocessing, by designing four new portfolios: WMO+simp+S4 (WMO+simp+S1) runs SCIP as an entire solver for 400s (100s), then WMaxCDCL-OpenWbo1200 with the SCIP preprocessing for 3200s (3500s); MO+simp+S4 and MO+simp+S1 are similar but use MaxCDCL instead of WMaxCDCL. Table 4 compares the results of the portfolios with or without SCIP as an entire solver. We see that the versions running SCIP for 100s give very good results, allowing us to solve 12, 14, and 3 more instances, respectively. But the versions running SCIP 400s give less good results. These results show that the SCIP preprocessing does not yet completely bridge the gap due to SCIP as an entire solver in a portfolio, but it significantly reduces the need to call it as an entire solver. Recall that running SCIP 400s as an entire solver in a portfolio means that we generally lose 400s for most instances for which SCIP is not effective, but we just lost 100s if we run SCIP 100s in a portfolio.

# 5 Conclusion

This paper investigates the impact of ILP preprocessing techniques on MaxSAT solving. The experimental results show that the ILP preprocessing techniques enable WMaxCDCL-OpenWbo1200, the winner of the MaxSAT evaluation 2024 in the unweighted track, to solve 15 additional instances in this track. They also show that the ILP preprocessing reduces the need to call the ILP solver in a portfolio including WMaxCDCL or MaxCDCL, because running SCIP 100s gives much better results than running 400s. Future work includes supporting more constraints to enable simplifying more instances and trying more encoding methods for constraints in the preprocessed ILP model. It would also be relevant to enhance the efficiency of the MaxSAT solver by fine-tuning the heuristic parameters.

## References


1 *MaxSAT Evaluation 2024: Solver and Benchmark Descriptions*. Department of Computer Science, University of Helsinki, Finland, 2024.

2 Tobias Achterberg. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, July 2009.

3 Carlos Ansótegui and Joel Gabàs. Solving (weighted) partial maxsat with ILP. In *Integration of AI and OR Techniques in Constraint Programming*, 2013.

4 Josep Argelich, Chu Min Li, and Felip Manyà. A preprocessor for max-sat solvers. In *SAT 2008*, pages 15–20, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

5 Florent Avellaneda. Evalmaxsat 2024. *MaxSAT Evaluation 2024 Solver and Benchmark Descriptions*, page 8, 2024.

6 Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. *Maximum Satisfiability*, pages 929 – 991. Frontiers in Artificial Intelligence and Applications. Netherlands, 2 edition, 2021.

7 A. Biere, M. Heule, and H. van Maaren. *Handbook of Satisfiability: Second Edition*. Frontiers in Artificial Intelligence and Applications. IOS Press, 2021.

8 Armin Biere, Matti Järvisalo, and Benjamin Kiesl. *Preprocessing in SAT Solving*, pages 391 – 435. Frontiers in Artificial Intelligence and Applications. Netherlands, 2 edition, 2021.

9 J. Coll D. Habet F. Manyà C. M. Li, S. Li and K. He. Maxcdcl in maxsat evaluation 2024. *MaxSAT Evaluation 2024 Solver and Benchmark Descriptions*, pages 15–16, 2024.

10 Z. Quan C.-M. Li, F. Manyà and Z. Zhu. Exact minsat solving. In *SAT-2010*, pages 363–368.

11 Alexey Ignatiev and Joao Marques-Silva. Xai-mindset2: Explainable ai with maxsat. *MaxSAT Evaluation 2018: Solver and Benchmark Descriptions*, page 43, 2018.

12 Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. RC2: an Efficient MaxSAT Solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 11(1):53–64, 2019.

13 Tuukka Korhonen, Jeremias Berg, Paul Saikko, and Matti Järvisalo. MaxPre: An Extended MaxSAT Preprocessor. In *SAT 2017*, volume 10491, pages 449–456. 2017.

14 Chu Min Li and Felip Manyà. Chapter 23. MaxSAT, Hard and Soft Constraints. In *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications. , February 2021.

15 Chu-Min Li, Fan Xiao, Mao Luo, Felip Manyà, Zhipeng Lü, and Yu Li. Clause vivification by unit propagation in CDCL SAT solvers. *Artificial Intelligence*, 279:103197, 2020.

16 Chu-Min Li, Zhenxing Xu, Jordi Coll, Felip Manyà, Djamal Habet, and Kun He. Combining Clause Learning and Branch and Bound for MaxSAT. *CP 2021*, 210:38:1–38:18, 2021.

17 Ruben Martins, Vasco Manquinho, and Inês Lynce. Open-wbo: A modular maxsat solver,. In *SAT 2014*, pages 438–445, Cham, 2014. Springer International Publishing.

18 Tobias Philipp and Peter Steinke. Pblib – a library for encoding pseudo-boolean constraints into cnf. In *SAT 2015*, pages 9–16, Cham, 2015. Springer International Publishing.

19 Marek Piotrów. Uwrmaxsat entering the maxsat evaluation 2024. *MaxSAT Evaluation 2024 Solver and Benchmark Descriptions*, pages 27–28, 2024.

20 J. Coll D. Habet F. Manyà S. Li, C. M. Li and K. He. Wmaxcdcl in maxsat evaluation 2024. *MaxSAT Evaluation 2024 Solver and Benchmark Descriptions*, pages 17–18, 2024.

21 S. Cai J. Li W. Zhu S. Pan, Y. Wang and M. Yin. Cashwmaxsat-disjcad: Solver description. *MaxSAT Evaluation 2024 Solver and Benchmark Descriptions*, page 25, 2024.

22 M. W. P. Savelsbergh. Preprocessing and Probing Techniques for Mixed Integer Programming Problems. *ORSA Journal on Computing*, 6(4):445–454, November 1994.

23 Z. Gu E. Rothberg T. Achterberg, R. Bixby and D. Weninger. Presolve reductions in mixed integer programming. *INFORMS Journal on Computing*, 32, 11 2019.

24 Dave A. D. Tompkins and Holger H. Hoos. Ubcsat: An implementation and experimentation environment for sls algorithms for sat and max-sat. In *SAT 2005*, pages 306–320, 2005.

25 J. Marques-Silva Y. Chen, S. Safarpour and A. G. Veneris. Automated design debugging with maximum satisfiability. *IEEE Transactions on Computer*, 29:1804–1817, 2010.

26 Zhifei Zheng, Sami Cherif, and Rui Sa Shibasaki. Optimizing power peaks in simple assembly line balancing through maximum satisfiability. In *ICTAI 2024*, pages 363–370. IEEE, 2024.