



# Integer Linear Programming Techniques for Enhancing Branch and Bound MaxSAT Solvers

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 (MaxSAT) problem is a major optimization challenge with numerous practical applications, including scheduling, hardware and software debugging, and explainable artificial intelligence. MaxSAT can be solved using SAT-based methods, Branch and Bound (BnB), Integer Linear Programming (ILP), and heuristic algorithms. In recent years, several hybrid algorithms have been proposed. Notably, in recent MaxSAT evaluations, most MaxSAT solvers have incorporated ILP solvers as part of their portfolios. This paper investigates the impact of ILP techniques on BnB MaxSAT solvers, particularly ILP preprocessing techniques and various portfolio strategies. Experimental results demonstrate that ILP techniques enable WMaxCDCL-OpenWbo1200 and MaxCDCL-OpenWbo300, the best two solvers in the unweighted track of the MaxSAT evaluation 2024, to solve 27 and 30 additional instances, respectively. Furthermore, although state-of-the-art MaxSAT solvers heavily rely on an ILP solver in their portfolios, our proposed approach uses ILP preprocessing techniques to reduce this dependency. Allocating only a short runtime to the ILP solver within a portfolio that includes (W)MaxCDCL, as proposed in our approach, is sufficient to achieve strong results.

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

**Keywords and phrases** Maximum Satisfiability, Branch and Bound, Integer Linear Programming, Preprocessing.

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

## 1 Introduction

Maximum Satisfiability (MaxSAT) is a natural optimization extension of the Propositional Satisfiability problem (SAT) [12]. 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 both theory and practice, because solving a SAT instance only requires finding a solution, whereas MaxSAT requires finding a solution and proving its optimality, which is more challenging [8, 28]. Many real-world optimization problems can be formulated as MaxSAT instances, including scheduling [15, 17, 43], hardware and software debugging [42, 40], explainable artificial intelligence [22, 23], among many others.

Algorithms for solving MaxSAT can be broadly classified into exact algorithms and heuristic algorithms. Exact algorithms, such as SAT-based, Branch and Bound (BnB), and Integer Linear Programming (ILP), find the optimal solution and prove its optimality. In



© Jialu Zhang, Chu-Min Li, Sami Cherif, Shuolin Li and Zhifei Zheng;  
licensed under Creative Commons License CC-BY 4.0

Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

contrast, heuristic algorithms, including local search and simulated annealing, can also be competitive, but they do not guarantee optimality [28, 32]. It is known that ILP solvers, while they perform well on certain families of instances, are not competitive for most industrial and random instances [5]. Therefore, the common practice observed in recent MaxSAT evaluations<sup>1</sup>, particularly for the most efficient solvers, is to combine ILP solvers in a portfolio with other types of solvers to solve MaxSAT instances. For example, in the MaxSAT evaluation 2024 [10], the total time limit to solve an instance is 3600s; EvalMaxSAT [7] first runs the ILP solver SCIP [3] for 400s and then itself for 3200s; UWMaxSat [39] 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 that leverage ILP, the ILP solver is typically used independently within the portfolio, requiring careful heuristic tuning, such as setting specific time limits.

In this paper, we propose a more integrated approach to enhance BnB MaxSAT solvers with ILP techniques. The process starts by reading the CNF formula and converting it into an integer linear programming problem. The ILP solver is then used to simplify the problem, after which the simplified integer linear constraints are re-encoded into CNF. Finally, the simplified MaxSAT instance is solved using a light portfolio of ILP and MaxSAT solvers. Our approach leverages ILP techniques by incorporating ILP preprocessing into the solving pipeline and combining ILP and BnB MaxSAT solvers through a light portfolio strategy. Experimental results demonstrate that this strategy allows state-of-the-art BnB MaxSAT solvers to solve more instances than the traditional portfolio approach.

The remainder of this paper is organized as follows. Section 2 introduces the MaxSAT problem, the BnB MaxSAT algorithms, and ILP techniques. Section 3 details the methodology for integrating an ILP solver into the MaxSAT solving pipeline. Section 4 presents our experimental results. Finally, Section 5 concludes the paper and outlines directions for future work.

## 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, which is also represented as a set 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  $F$  [12].

MaxSAT is an optimization extension of SAT (more natural than MinSAT, another optimization extension of SAT [30]), encompassing both Partial MaxSAT and Weighted Partial MaxSAT [8, 28]. Partial MaxSAT divides clauses into a subset of *hard* clauses  $H$  and a subset of *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

---

<sup>1</sup> <https://maxsat-evaluations.github.io/>

an optimal assignment that maximizes the sum of weights of satisfied soft clauses while satisfying all the hard clauses. Partial MaxSAT is a particular case of Weighted Partial MaxSAT with  $w_c = 1$  for every soft clause  $c$ .

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, and  $V$  is the set of Boolean decision variables in  $F$ . A binary variable  $y_x$  is introduced for each Boolean variable  $x$  in  $V$ , and a binary variable  $z_c$  is introduced for each soft clause  $c$  in  $S$ . 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.

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

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

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

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

## 2.2 Branch and Bound for MaxSAT

Branch and Bound (BnB) MaxSAT algorithms explore the solution space by incrementally building a binary search tree. During the search, a BnB algorithm continually updates the upper bound (UB), which reflects the cost of the best solution found, and the lower bound (LB), which estimates the minimum achievable cost [14]. If the current branch's LB exceeds the UB, it indicates that no better solution can be found with the current assignment. The algorithm then backtracks to an unexplored branch to continue the search. Solvers such as MaxSatz [29], MiniMaxSAT [20], AHMAXSAT [2], and MaxCDCL [32] are based on BnB MaxSAT algorithms.

The MaxSAT-resolution [36] rule is widely used in BnB MaxSAT solvers to prevent the repeated detection of the same conflicts. Obtaining a tighter lower bound (LB) is also crucial for these solvers. For example, the MaxCDCL solver performs a look-ahead procedure [21] to estimate a more accurate LB for the current branch. Preprocessing techniques are also broadly used in these solvers to reduce the number of variables and clauses, such as bounded variable elimination (BVE), failed literal detection, unit propagation, and self-subsuming resolution [6, 13]. Clause vivification [31] can also be used as preprocessing or inprocessing to simplify hard clauses. These techniques are often based on resolution, which differs significantly from the preprocessing techniques in ILP solvers.

## 2.3 Integer Linear Programming

The Integer Linear Programming solving process can be divided into a preprocessing stage and a solving stage. In the preprocessing stage, the original instance is transformed into an equivalent one that is potentially easier to solve. Then, in the solving stage, the solution space of the transformed instance is explored. Different techniques are applied in each stage.

In the preprocessing stage, techniques such as variable fixing, variable aggregation, redundant constraint elimination, and other advanced inference mechanisms [41] play a

key role in speeding up problem-solving. The variable fixing technique employs a probing algorithm that temporarily assigns a binary variable to 0 or 1 and then propagates the resulting implications [3]. 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 [4].

In the solving stage, the primary methods used are Branch and Bound [9] and Cutting Planes [34] algorithms. The Branch and Bound algorithm systematically explores the ILP solution space, pruning branches that cannot yield optimal solutions. In contrast, the Cutting Planes algorithm iteratively addresses the linear programming relaxation of the ILP by incrementally adding linear constraints to tighten the feasible region until all decision variables satisfy integrality. Duality principles [25] is a technique to enhance both: in Branch and Bound, dual bounds help prune suboptimal branches, while in Cutting Planes, dual solutions guide the formulation of effective constraints. There are also auxiliary techniques, such as restarts, branching node selection heuristics, and others, that help improve solving efficiency [3].

### 3 Methodology

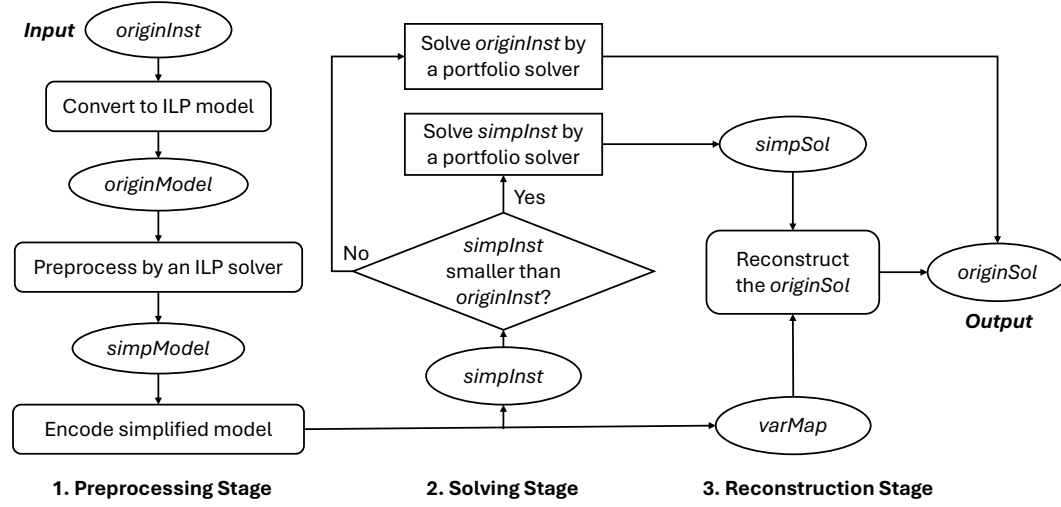
In this section, we propose a three-stage methodology to integrate an ILP solver into the MaxSAT solving pipeline.

#### 3.1 Integrating ILP techniques into MaxSAT solving

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:** At this stage, a portfolio solver is employed to solve either the simplified instance *simpInst* or the original instance *originInst*. If *simpInst* is "smaller" than *originInst*, i.e., if *simpInst* contains fewer variables *and* fewer hard clauses than *originInst*, the portfolio first calls an ILP solver within a limited time to solve *simpInst*; if no optimal solution is obtained, a BnB MaxSAT solver is then called. Otherwise, *originInst* is solved by the portfolio solver. The definition of "smaller" is debatable and deserves future study, as reducing the number of variables and clauses in a MaxSAT instance does not necessarily improve the solving efficiency.
3. **Reconstruction Stage:** At this optional stage, the algorithm constructs an optimal solution *originSol* for *originInst* with *simpSol* and *varMap*. This stage happens only when *simpInst* is "smaller" than *originInst*.

Our methodology, illustrated in Figure 1, integrates both ILP preprocessing and solving techniques into the MaxSAT solving pipeline. During the preprocessing stage, the ILP preprocessing techniques are employed to simplify the MaxSAT instance. Subsequently, in the solving stage, the ILP solving techniques are used alongside MaxSAT solving techniques in a portfolio approach. Note that the ILP solver avoids performing any preprocessing during the solving stage to prevent redundant computation.



■ **Figure 1** A three-stage methodology to integrate an ILP solver into the MaxSAT solving pipeline.

The key aspect in our methodology 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.

After preprocessing by an ILP solver, the original ILP model (*originModel*) is transformed into *simpModel*, in which we distinguish three types of binary decision 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*.

### 3.2 Variable Encoding

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  $i = 1, \dots, n$ . In the case of a simple aggregation, i.e.,  $n = 1$ ,  $c_0 = 0$ ,  $c_1 = 1$  and  $n = 1$ ,  $c_0 = 1$ ,  $c_1 = -1$ , we have  $y_x = y_1$  and  $y_x = 1 - y_1$ , respectively. 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  $(y_1 = 1 - y_2)$  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  $\{y_1 \rightarrow \neg v_1, y_2 \rightarrow v_1, y_3 \rightarrow v_1\}$  when transforming *simpModel* to *simpInst*, which preserves  $(y_1 = 1 - y_2)$  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 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).

■ **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
- 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**

---

### 3.3 Constraint Encoding

We use the SCIP solver [3] 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 [38], respectively. We use the default configuration in PBLib, allowing it to automatically select the most suitable encoding (such as Binary Decision Diagrams (BDD) [1], Adder Networks [19], among others) based on the properties of the constraints. The unsupported constraint type is *orbitope*, which arises from orbitopal fixing [24], a symmetry-breaking technique commonly used in ILP preprocessing. If *simpModel* contains the *orbitope* constraint, the preprocessing is stopped, and the original MaxSAT instance is returned to the solver. In our experiment, we found that about 8% of instances contain the *orbitope* constraint, and we will incorporate this constraint into our methodology in the future.

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

Constraint	Formula	Encoding
<i>Logical OR</i>	$\sum_{i=1}^n x_i \geq 1$	$(x_1 \vee x_2 \vee \dots \vee x_n)$
<i>Logical AND</i>	$\prod_{i=1}^n x_i = y$	$(y \vee \neg x_1 \vee \dots \vee \neg x_n) \wedge \bigwedge_{i=1}^n (\neg y \vee x_i)$
<i>Setppc packing</i>	$\sum_{i=1}^n x_i \leq 1$	At-most-one
<i>Setppc partitioning</i>	$\sum_{i=1}^n x_i = 1$	at-most-one $\wedge (x_1 \vee x_2 \vee \dots \vee x_n)$
<i>Linear</i>	$lhs \leq \sum_{i=1}^n w_i \cdot x_i \leq rhs$	Pseudo-Boolean

The objective function of *simpModel* is  $f'_{(S)} = \text{Maximize } \sum_{c \in S} w'_c \cdot z'_c$ , 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

### 4.1 Test Environment

We use state-of-the-art ILP and BnB MaxSAT solvers in our experiments. Specifically, we employ the best-known open-source ILP solver, SCIP [3] (version 9.1.1)<sup>2</sup>. For the BnB MaxSAT solvers, we select and download WMaxCDCL-OpenWbo1200 [33] and MaxCDCL-OpenWbo300 [27], the two leading open-source MaxSAT solvers from the MaxSAT evaluation 2024 in the unweighted category<sup>3</sup>. WMaxCDCL-OpenWbo1200 (MaxCDCL-OpenWbo300) runs OpenWbo [35] for 1200s (300s) followed by WMaxCDCL (MaxCDCL) for 2400s (3300s) to solve an instance. Note that MaxCDCL-OpenWbo300 supports only unweighted MaxSAT instances, while SCIP and WMaxCDCL-OpenWbo1200 support both weighted and unweighted instances.

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 the duplicated instances twice, we removed from (W)MS $k$  for  $k > 19$  the instances also occurring in previous years from 2019. The unweighted (weighted) instances come from 74 (63) instance families. Each instance family represents a specific optimization problem encoded into MaxSAT, from different fields related to combinatorial optimization and AI, making the tests and our observations more comprehensive and credible.

The computations are performed on a machine equipped with an AMD EPYC 7502 Processor (2.5 GHz) and a Linux system. As in the MaxSAT evaluations, each solver is allocated one CPU, a time limit of 3600 seconds, and 31GB of RAM to solve an instance.

### 4.2 ILP vs. MaxSAT BnB

We first evaluate the performance gap between the ILP solver and the BnB MaxSAT solvers to motivate our approach. As shown in Table 2, the performance of the ILP solver SCIP is significantly worse compared to the two BnB MaxSAT solvers WMaxCDCL-OpenWbo1200 (WMO) and MaxCDCL-OpenWbo300 (MO). SCIP solved only 954 instances in the unweighted category, whereas WMO and MO solved 1660 and 1650 instances, respectively, substantially outperforming SCIP. This performance gap also persists in the weighted category.

From the above evaluation, we make the following observation. On one hand, the SAT/MaxSAT community has made an intensive effort for several decades to improve BnB MaxSAT solvers, so that these solvers reach a high level of maturity, but it is increasingly challenging to achieve further performance improvements using pure MaxSAT/SAT techniques. On the other hand, the ILP solver still demonstrates certain advantages over the BnB MaxSAT solvers in solving specific instances. The rows labeled  $SCIP \setminus (W)MO$  in Table 2 represent instances solved by SCIP but not by (W)MO. In the unweighted category, SCIP solved 68 instances that MO failed to solve and 60 instances that WMO failed to solve. In the weighted

<sup>2</sup> <https://www.scipopt.org>

<sup>3</sup> <https://maxsat-evaluations.github.io/2024/results/exact/unweighted.html>

■ **Table 2** The number of instances solved by the ILP and BnB MaxSAT solvers within 3600s.  $SCIP \setminus (W)MO$  denote instances solved by SCIP but not by MaxCDCL-OpenWbo300 (WMaxCDCL-OpenWbo1200).

Unweighted category	MS19	MS20	MS21	MS22	MS23	MS24	Sum
#Instances	599	401	448	254	260	247	<b>2209</b>
SCIP	235	203	208	102	118	88	<b>954</b>
MO	441	315	344	186	177	197	<b>1660</b>
WMO	446	306	339	183	177	199	<b>1650</b>
$SCIP \setminus MO$	26	12	11	5	9	5	<b>68</b>
$SCIP \setminus WMO$	21	13	10	4	8	4	<b>60</b>
Weighted category	WMS19	WMS20	WMS21	WMS22	WMS23	WMS24	Sum
#Instances	586	433	491	291	218	204	<b>2223</b>
SCIP	228	239	238	122	117	101	<b>1045</b>
WMO	391	337	397	210	154	147	<b>1636</b>
$SCIP \setminus WMO$	10	10	7	3	10	0	<b>40</b>

category, although fewer in number, there are still 40 such instances. This motivates us to transfer some of the ILP solver’s capabilities to BnB MaxSAT solvers, thereby enabling them to handle these particular instances without deteriorating their performance for other instances.

For this purpose, we further investigate the instances where the ILP solver outperforms the BnB MaxSAT solvers and categorize these instances by their respective families. Table 3 lists the three most significant instance families in each category: *optic* [18], *extension-enforcement* [37], and *logic-synthesis* for the unweighted category; *judgment-aggregation* [16], *min-width* [11], and *set-covering* [26] for the weighted category. A portfolio strategy could help BnB MaxSAT solvers handle more instances in the *logic-synthesis*, *optic*, and *set-covering* families, as SCIP solves these particular cases with relatively low computational

■ **Table 3** The top three instance families ranked by the number of instances in  $SCIP \setminus (W)MO$ .  $T_{avg}$  denotes the average CPU time (in seconds) required by SCIP to solve the instances in  $SCIP \setminus (W)MO$ .

Unweighted family	<i>optic</i>	<i>extension-enforcement</i>	<i>logic-synthesis</i>
#Instances	49	39	17
SCIP	37	34	16
MO	19	23	9
WMO	22	28	10
$SCIP \setminus MO$ ( $T_{avg}$ )	<b>22</b> (126.28s)	<b>15</b> (1019.62s)	<b>7</b> (2.18s)
$SCIP \setminus WMO$ ( $T_{avg}$ )	<b>19</b> (144.67s)	<b>10</b> (1331.83s)	<b>6</b> (2.41s)
Weighted family	<i>judgment-aggregation</i>	<i>min-width</i>	<i>set-covering</i>
#Instances	15	53	35
SCIP	15	17	33
WMO	5	12	27
$SCIP \setminus WMO$ ( $T_{avg}$ )	<b>10</b> (367.02s)	<b>9</b> (568.30s)	<b>6</b> (111.46s)



cost. However, creating effective portfolios for the *extension-enforcement*, *min-width*, and *judgment-aggregation* families is challenging, as SCIP requires considerable computation time to solve these instances. Determining optimal time limits for each solver in the portfolio requires careful consideration, because giving more time to SCIP in a portfolio means reducing the time of (W)MO for other instances for which SCIP is not efficient.

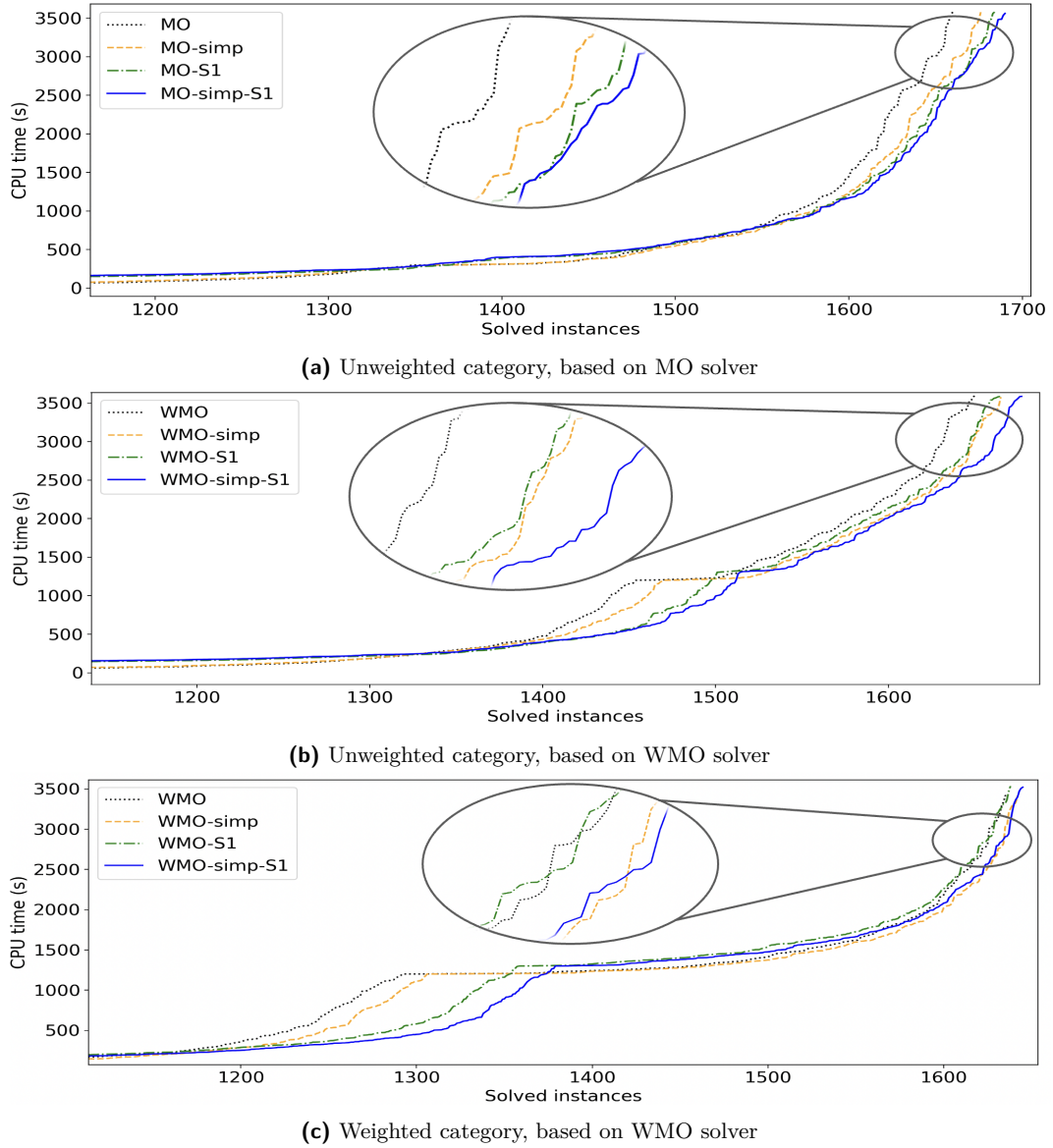
### 4.3 Evaluation of the Methodology

The proposed methodology is evaluated using the MO and WMO baseline solvers. We evaluate two portfolio strategies, ILP preprocessing, and the combination of preprocessing and portfolio. For the two portfolio strategies: (W)MO+S4 and (W)MO+S1, we first run SCIP for 400 seconds and 100 seconds, followed by (W)MO for 3200 seconds and 3500 seconds, respectively. For the ILP preprocessing: (W)MO+simp solver first uses SCIP to preprocess the original instance, then uses the (W)MO solver to solve the simplified instance. For the combination of preprocessing and portfolio: (W)MO+simp+S4 and (W)MO+simp+S1, first preprocess the original instance, then solve the simplified instance with their corresponding portfolio solvers.

Table 4 presents the detailed test results, and Figure 2 illustrates the relationship between the number of solved instances and CPU time for selected solvers. First, we observe that the portfolio solvers (W)MO+S4 solve fewer instances compared to (W)MO+S1, indicating that increasing the runtime allocated to SCIP in the portfolio strategy can produce negative side effects. Second, we find that applying only SCIP preprocessing techniques helps MO and WMO solve 16 and 15 additional instances, respectively, underscoring the importance of ILP preprocessing. This effect is particularly pronounced for weighted instances, where WMO+simp solves more instances than either WMO+S4 or WMO+S1. Third, (W)MO+simp+S1 and (W)MO+simp+S4 solve more instances than their respective counterparts, (W)MO+S1

**Table 4** Number of instances solved by (W)MO with different strategies based on our methodology. *Inc* denotes the additional instances solved compared to the corresponding (W)MO baseline solver.

Unweighted category	MS19	MS20	MS21	MS22	MS23	MS24	Sum( <i>Inc</i> )
MO+S4	453	319	345	186	179	200	<b>1682(+22)</b>
MO+S1	454	319	347	186	178	200	<b>1684(+24)</b>
MO+simp	451	314	348	187	178	198	<b>1676(+16)</b>
MO+simp+S4	457	317	344	185	180	200	<b>1683(+23)</b>
MO+simp+S1	459	318	348	186	180	199	<b>1690(+30)</b>
WMO+S4	452	307	339	181	178	201	<b>1658(+8)</b>
WMO+S1	455	308	342	182	178	200	<b>1665(+15)</b>
WMO+simp	456	307	341	184	178	199	<b>1665(+15)</b>
WMO+simp+S4	460	309	339	183	179	200	<b>1670(+20)</b>
WMO+simp+S1	462	310	343	184	178	200	<b>1677(+27)</b>
Weighted category	WMS19	WMS20	WMS21	WMS22	WMS23	WMS24	Sum( <i>Inc</i> )
WMO+S4	395	335	396	211	154	146	<b>1637(+1)</b>
WMO+S1	395	336	397	210	154	146	<b>1638(+2)</b>
WMO+simp	392	341	399	209	154	147	<b>1642(+6)</b>
WMO+simp+S4	394	339	399	210	154	146	<b>1642(+6)</b>
WMO+simp+S1	393	342	399	210	154	147	<b>1645(+9)</b>



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

and (W)MO+S4. This suggests that combining ILP preprocessing with a portfolio strategy outperforms the classic portfolio approach, with the best configuration enabling MO to solve 30 additional instances and WMO to solve 27 additional instances.

Table 5 shows the impact of our methodology on specific instance families, as discussed in Section 4.2. For the unweighted category: (W)MO+S1 solves more instances than (W)MO within the *optic* and *logic-synthesis* families, although this advantage does not extend to the *extension-enforcement* family. (W)MO+simp improves the performance of (W)MO across all three families; however, the improvement in the *optic* and *logic-synthesis* families is less pronounced than with (W)MO+S1. (W)MO+simp+S1 combines the advantages of ILP preprocessing and portfolio techniques, thereby achieving the greatest improvement among all these solvers. For the weighted category, the impact on individual families is less relevant

■ **Table 5** The number of solved instances on specific families. *Inc* denotes the additional instances solved compared to the corresponding MO or WMO solver.

Unweighted family	<i>optic</i>	<i>extension-enforcement</i>	<i>logic-synthesis</i>
MO+S1 ( <i>Inc</i> )	34 (+15)	23 (0)	16 (+7)
MO+simp ( <i>Inc</i> )	25 (+6)	27 (+4)	13 (+4)
MO+simp+S1 ( <i>Inc</i> )	35 (+16)	27 (+4)	16 (+7)
WMO+S1 ( <i>Inc</i> )	35 (+13)	27 (-1)	16 (+6)
WMO+simp ( <i>Inc</i> )	26 (+4)	36 (+8)	12 (+2)
WMO+simp+S1 ( <i>Inc</i> )	36 (+14)	36 (+8)	16 (+6)
Weighted family	<i>judgment-aggregation</i>	<i>min-width</i>	<i>set-covering</i>
WMO+S1 ( <i>Inc</i> )	5(0)	12(0)	27(0)
WMO+simp ( <i>Inc</i> )	5(0)	14(2)	27(0)
WMO+simp+S1 ( <i>Inc</i> )	5(0)	15(3)	27(0)

as SCIP initially managed to solve fewer instances independently, but our methodology does not have any negative impact on these families and even manages to achieve better results on *min-width*.

To analyze the impact of ILP preprocessing, we divide the MaxSAT benchmark instances into three categories: Smaller, Bigger, and Failed. An instance is placed in Smaller or Bigger only if a simplified instance, *simpInst*, is successfully generated. If *simpInst* contains both fewer variables and fewer hard clauses than the original instance *originInst*, the instance is classified as Smaller; otherwise, it is classified as Bigger. An instance is marked as Failed when *simpModel* contains the *orbitope* constraint, which is not yet supported. Extremely large instances with more than 200,000 variables or 1,000,000 clauses are skipped during preprocessing and are not included in the data. Table 6 summarizes the statistics for the three instance groups. We observe 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. Furthermore, 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).

■ **Table 6** Statistics of three groups of instances w.r.t. the SCIP preprocessing.

States	Unweighted category			Weighted category		
	Smaller	Bigger	Failed	Smaller	Bigger	Failed
#Instances	1085	436	182	980	508	201
<i>PreprocessingTime</i>	15.36s	30.0s	6.20s	14.26s	19.65s	24.86s
<i>FixedVarsRate</i>	18.66%	3.64%	22.81%	19.61%	16.30%	43.48%
<i>AggregatedVarsRate</i>	26.92%	21.52%	30.56%	27.68%	18.36%	29.67%
<i>simpleAggregationRatio</i>	99.49%	79.54%	-	99.22%	96.64%	-

## 5 Conclusion

This paper investigates the impact of ILP techniques to improve MaxSAT solving. We show that ILP solvers are generally less efficient for MaxSAT than the leading solvers in the exact tracks of the MaxSAT Evaluations. Nevertheless, integrating ILP techniques into WMaxCDCL-OpenWbo1200 and MaxCDCL-OpenWbo300—the winners of the unweighted track of the MaxSAT evaluation 2024—for preprocessing and solving with a short 100s time limit enables us to solve 27 and 30 additional instances, respectively. These results are significant because MaxSAT solving has achieved a high level of maturity, and winning a track typically requires solving only a handful of additional instances (often just three) more than the runner-up. Our results also suggest that ILP preprocessing techniques are effective at reducing the number of variables in most instances. This reduction likely explains why they enable the solver to tackle more instances.

Future work will focus on two main aspects. First, to improve our methodology, we will add support for the currently unsupported constraint type, explore additional encoding algorithms, and fine-tune our heuristic parameters. Second, our approach has not yet fully leveraged the potential of ILP techniques. We plan to conduct further investigations into these techniques to enable ILP to outperform branch and bound MaxSAT solvers on specific instance families. We will then incorporate the most effective preprocessing and solving strategies into these solvers to improve their efficiency.

---

## References

- 1 Ignasi Abío, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Bdds for pseudo-boolean constraints – revisited. In Laurent Sakallah, Karem A. and Simon, editor, *Theory and Applications of Satisfiability Testing - SAT 2011*, pages 61–75, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
- 2 André Abramé and Djamel Habet. Ahmaxsat: Description and evaluation of a branch and bound max-sat solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 9, 12 2015. doi:10.3233/SAT190104.
- 3 Tobias Achterberg. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, July 2009. doi:10.1007/s12532-008-0001-1.
- 4 Tobias Achterberg, Robert Bixby, Zonghao Gu, Edward Rothberg, and Dieter Weninger. Presolve reductions in mixed integer programming. *INFORMS Journal on Computing*, 32, 11 2019. doi:10.1287/ijoc.2018.0857.
- 5 Carlos Ansótegui and Joel Gabàs. Solving (weighted) partial maxsat with ILP. In *Integration of AI and OR Techniques in Constraint Programming*, 2013.
- 6 Josep Argelich, Chu Min Li, and Felip Manyà. A preprocessor for max-sat solvers. In Hans Kleine Büning and Xishun Zhao, editors, *Theory and Applications of Satisfiability Testing – SAT 2008*, pages 15–20, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
- 7 Florent Avellaneda. Evalmaxsat 2024. *MaxSAT Evaluation 2024 Solver and Benchmark Descriptions*, page 8, 2024.
- 8 Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. *Maximum Satisfiability*, pages 929 – 991. Frontiers in Artificial Intelligence and Applications. IOS PRESS, Netherlands, 2 edition, 2021. doi:10.3233/FAIA201008.
- 9 E. M. L. Beale and R. E. Small. Mixed integer programming by a branch and bound technique. In W. Kalench, editor, *Proceedings of the IFIP Congress*, volume 2, pages 450–451, London, 1965. Macmillan.
- 10 Jeremias Berg, Matti Järvisalo, Ruben Martins, Andreas Niskanen, and Tobias Paxian, editors. *MaxSAT Evaluation 2024: Solver and Benchmark Descriptions*. Department of Computer

- Science Series of Publications B. Department of Computer Science, University of Helsinki, Finland, 2024.
- 11 Jeremias Berg, Emilia Oikarinen, Matti Järvisalo, and Kai Puolamäki. Maxsat benchmarks from the minimum-width confidence band problem. *MaxSAT Evaluation 2017 Solver and Benchmark Descriptions*, page 38, 2017.
  - 12 A. Biere, M. Heule, and H. van Maaren. *Handbook of Satisfiability: Second Edition*. Frontiers in Artificial Intelligence and Applications. IOS Press, 2021.
  - 13 Armin Biere, Matti Järvisalo, and Benjamin Kiesl. *Preprocessing in SAT Solving*, pages 391 – 435. Frontiers in Artificial Intelligence and Applications. IOS PRESS, Netherlands, 2 edition, 2021. doi:10.3233/FAIA200992.
  - 14 Mohamed Sami Cherif, Djamal Habet, and André Abramé. Understanding the power of max-sat resolution through up-resilience. *Artificial Intelligence*, 289:103397, 2020. doi:10.1016/j.artint.2020.103397.
  - 15 Sami Cherif, Heythem Sattoutah, Chu-Min Li, Corinne Lucet, and Laure Brisoux Devendeville. Minimizing working-group conflicts in conference session scheduling through maximum satisfiability (short paper). In Paul Shaw, editor, *30th International Conference on Principles and Practice of Constraint Programming, CP 2024, September 2-6, 2024, Girona, Spain*, volume 307 of *LIPICs*, pages 34:1–34:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICs.CP.2024.34.
  - 16 Ari Conati, Andreas Niskanen, and Matti Järvisalo. Maxsat encodings for judgment aggregation. *MaxSAT Evaluation 2023 Solver and Benchmark Descriptions*, page 33, 2023.
  - 17 Emir Demirović and Nysret Musliu. Maxsat-based large neighborhood search for high school timetabling. *Computers & Operations Research*, 78:172–180, 2017. doi:10.1016/j.cor.2016.08.004.
  - 18 Rüdiger Ehlers. Approximately propagation complete and approximately conflict propagating sat encoding computation maxsat benchmarks. *MaxSAT Evaluation 2018 Solver and Benchmark Descriptions*, page 38, 2018.
  - 19 Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into sat. *Journal on Satisfiability, Boolean Modelling and Computation*, 2(1-4):1–26, 2006. doi:10.3233/SAT190014.
  - 20 Federico Heras, Javier Larrosa, and Albert Oliveras. Minimaxsat: A new weighted max-sat solver. In João Marques-Silva and Karem A. Sakallah, editors, *Theory and Applications of Satisfiability Testing – SAT 2007*, pages 41–55, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
  - 21 Marijn J.H. Heule and Hans van Maaren. *Chapter 5: Look-ahead based SAT solvers*, pages 183–212. Frontiers in Artificial Intelligence and Applications. IOS Press BV, 2021. Publisher Copyright: © 2021 The authors and IOS Press. All rights reserved. doi:10.3233/FAIA200987.
  - 22 Hao Hu, Marie-José Huguet, and Mohamed Siala. Optimizing Binary Decision Diagrams with MaxSAT for Classification. In *36th AAAI Conference on Artificial Intelligence*, Vancouver, Canada, February 2022.
  - 23 Alexey Ignatiev and Joao Marques-Silva. Xai-mindset2: Explainable ai with maxsat. *MaxSAT Evaluation 2018: Solver and Benchmark Descriptions*, page 43, 2018.
  - 24 Volker Kaibel, Matthias Peinhardt, and Marc E. Pfetsch. Orbitopal fixing. *Discrete Optimization*, 8(4):595–610, 2011. doi:10.1016/j.disopt.2011.07.001.
  - 25 Nikos Komodakis and Jean-Christophe Pesquet. Playing with duality: An overview of recent primal? dual approaches for solving large-scale optimization problems. *IEEE Signal Processing Magazine*, 32(6):31–54, 2015. doi:10.1109/MSP.2014.2377273.
  - 26 Zhendong Lei and Shaowei Cai. Solving set cover and dominating set via maximum satisfiability. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34:1569–1576, 04 2020. doi:10.1609/aaai.v34i02.5517.
  - 27 Chu Min Li, Shuolin Li, Jordi Coll, Djamal Habet, Felip Manyà, and Kun He. Maxcdcl in maxsat evaluation 2024. *MaxSAT Evaluation 2024 Solver and Benchmark Descriptions*, pages 15–16, 2024.

- 28 Chu Min Li and Felip Manyà. Chapter 23. MaxSAT, Hard and Soft Constraints. In *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications. IOS Press, February 2021. doi:10.3233/FAIA201007.
- 29 Chu Min Li, Felip Manyà, and Jordi Planes. New inference rules for Max-SAT. *Journal of Artificial Intelligence Research*, 30:321–359, 2007.
- 30 Chu-Min Li, Felip Manyà, Zhe Quan, and Zhu Zhu. Exact minsat solving. In *International Conference on Theory and Applications of Satisfiability Testing (SAT-2010)*, pages 363–368, 2010.
- 31 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.
- 32 Chu-Min Li, Zhenxing Xu, Jordi Coll, Felip Manyà, Djamel Habet, and Kun He. Combining Clause Learning and Branch and Bound for MaxSAT. *LIPICs, Volume 210, CP 2021*, 210:38:1–38:18, 2021. doi:10.4230/LIPICs.CP.2021.38.
- 33 Shuolin Li, Chu Min Li, Jordi Coll, Djamel Habet, Felip Manyà, and Kun He. Wmaxcdcl in maxsat evaluation 2024. *MaxSAT Evaluation 2024 Solver and Benchmark Descriptions*, pages 17–18, 2024.
- 34 Hugues Marchand, Alexander Martin, Robert Weismantel, and Laurence Wolsey. Cutting planes in integer and mixed integer programming. *Discrete Applied Mathematics*, 123(1):397–446, 2002. doi:10.1016/S0166-218X(01)00348-1.
- 35 Ruben Martins, Vasco Manquinho, and Inês Lynce. Open-wbo: A modular maxsat solver,. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing – SAT 2014*, pages 438–445, Cham, 2014. Springer International Publishing.
- 36 María Luisa Bonet and Jordi Levy and Felip Manyà. Resolution for max-sat. *Artificial Intelligence*, 171(8):606–618, 2007. doi:10.1016/j.artint.2007.03.001.
- 37 Andreas Niskanen, Johannes P. Wallner, and Matti Järvisalo. Discrete optimization problems in dynamics of abstract argumentation: Maxsat benchmarks. *MaxSAT Evaluation 2017 Solver and Benchmark Descriptions*, pages 23–24, 2017.
- 38 Tobias Philipp and Peter Steinke. Pblib – a library for encoding pseudo-boolean constraints into cnf. In Marijn Heule and Sean Weaver, editors, *Theory and Applications of Satisfiability Testing – SAT 2015*, pages 9–16, Cham, 2015. Springer International Publishing.
- 39 Marek Piotrów. Uwrmaxsat entering the maxsat evaluation 2024. *MaxSAT Evaluation 2024 Solver and Benchmark Descriptions*, pages 27–28, 2024.
- 40 Sean Safarpour, Hratch Mangassarian, Andreas Veneris, Mark H. Liffiton, and Karem A. Sakallah. Improved design debugging using maximum satisfiability. In *Formal Methods in Computer Aided Design (FMCAD’07)*, pages 13–19, 2007. doi:10.1109/FAMCAD.2007.26.
- 41 M. W. P. Savelsbergh. Preprocessing and Probing Techniques for Mixed Integer Programming Problems. *ORSA Journal on Computing*, 6(4):445–454, November 1994. doi:10.1287/ijoc.6.4.445.
- 42 Yibin Chen and Sean Safarpour and Joao Marques-Silva and Andreas G. Veneris. Automated design debugging with maximum satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29:1804–1817, 2010.
- 43 Zhifei Zheng, Sami Cherif, and Rui Sa Shibasaki. Optimizing power peaks in simple assembly line balancing through maximum satisfiability. In *36th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2024, Herndon, VA, USA, October 28-30, 2024*, pages 363–370. IEEE, 2024. doi:10.1109/ICTAI62512.2024.00060.