SAT-Based Leximax Optimisation Algorithms

Miguel Cabral
INESC-ID, IST, University of Lisbon, Portugal

Mikoláš Janota
Czech Technical University in Prague, Czech Republic

Vasco Manquinho
INESC-ID, IST, University of Lisbon, Portugal

Abstract

In several real-world problems, it is often the case that the goal is to optimise several objective functions. However, usually there is not a single optimal objective vector. Instead, there are many optimal objective vectors known as Pareto-optima. Finding all Pareto-optima is computationally expensive and the number of Pareto-optima can be too large for a user to analyse. A compromise can be made by defining an optimisation criterion that integrates all objective functions.

In this paper we propose several SAT-based algorithms to solve multi-objective optimisation problems using the leximax criterion. The leximax criterion is used to obtain a Pareto-optimal solution with a small trade-off between the objective functions, which is suitable in problems where there is an absence of priorities between the objective functions. Experimental results on the Multi-Objective Package Upgradeability Optimisation problem show that the SAT-based algorithms are able to outperform the Integer Linear Programming (ILP) approach when using non-commercial ILP solvers. Additionally, experimental results on selected instances from the MaxSAT evaluation adapted to the multi-objective domain show that our approach outperforms the ILP approach using commercial solvers.

2012 ACM Subject Classification Computing methodologies → Optimization algorithms

Keywords and phrases Multi-Objective Optimisation, Leximax, Sorting Networks

Digital Object Identifier 10.4230/LIPIcs.SAT.2022.29

Supplementary Material *Software (Source Code)*: `https://github.com/miguelcabral/leximaxIST`

Funding This work was partially supported by Portuguese national funds through FCT, under projects UIDB/50021/2020, PTDC/CCI-COM/31198/2017, PTDC/CCI-COM/32378/2017 and DSAIPA/AI/0044/2018. The results were supported by the Ministry of Education, Youth and Sports within the dedicated program ERC CZ under the project *POSTMAN* no. LL1902. This article is part of the *RICAIP* project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857306.

1 Introduction

In many real-world problems such as Virtual Machine Consolidation [46], trip planning [50], automated program repair [70] or Package Upgradeability [39], there are several objective functions to minimise. The challenge with having more than one objective function is that the objective functions may be conflicting. Decreasing one objective function may lead to an increase of another objective function. Despite this trade-off, it is possible to discard feasible solutions for which there exists another feasible solution that is able to decrease all objective functions at the same time. For example, suppose we have two objective functions f_1 and f_2 and two feasible solutions α, α' with objective vectors $(f_1(\alpha), f_2(\alpha)) = (10, 10)$ and $(f_1(\alpha'), f_2(\alpha')) = (20, 20)$. In this case there is no trade-off and α is clearly preferred over α'. This motivates the well-known notion of Pareto-optimality. A Pareto-optimal solution is such that there does not exist another feasible solution that decreases all objective functions at the same time.

When solving a Multi-Objective Boolean Optimisation problem, one can try to enumerate or approximate the set of Pareto-optima (also known as Pareto frontier) and leave it to an expert to choose one of those solutions. The expert does not have to analyse all Pareto-optima, since there are criteria to select a representative subset of Pareto-optimal solutions [33]. A different approach is to transform the multi-objective problem into a single-objective one by using a linear combination of the objective functions. However, defining the weight for each objective function is often unclear for users.

One can also compute a Pareto-optimal solution that is minimal according to a certain order. For example, in the lexicographic order [52], users define priorities to the objective functions. In this case, an optimal solution corresponds to minimising the highest priority objective function, then the second highest priority objective function, and so on. For instance, the vector $(20, 50)$ is lexicographically smaller than the vector $(40, 30)$, assuming an order of priorities from left to right. However, defining a lexicographic order corresponds to selecting one feasible solution from the extremes of the Pareto frontier. As a result, the lexicographic-optimum will likely be very unbalanced, i.e. some objective functions will have very small values and other objective functions will have very large values.

Unlike the lexicographic order, the leximax relation tends to provide a small trade-off between the several objective functions. Moreover, the leximax relation does not require the user to predefine an order of priority between the objective functions. Hence, computing a leximax-optimal solution is suitable in problems where there is an absence of priorities between the objective functions. In practice, the leximax-optimum corresponds to minimising the maximum value among all objective functions, then the second maximum of the objective functions, and so on. Therefore, besides minimising the worst value among all objective functions, the solution found using the leximax criterion is usually balanced. Thus, leximax is a natural criterion for solving problems where an order among the objective functions is not defined, returning a balanced solution with good performance. Finally, observe that the lexicographic-optimum and the leximax-optimum are both a Pareto-optimum [28].

In this paper, we propose several SAT-based leximax optimisation algorithms. The main contributions of our work are:

1. new incremental SAT-UNSAT and UNSAT-SAT leximax optimisation algorithms,
2. the use of unsatisfiable cores to improve the performance of UNSAT-SAT leximax optimisation algorithms,
3. dynamical construction of the CNF representation of the objective functions, and
4. an extensive empirical evaluation of our algorithms on the Multi-Objective Package Upgradeability Optimisation problem [49] and on randomly generated multi-objective instances based on the MaxSAT Evaluation 2021 benchmarks [55].

This paper is organised as follows. In Section 2 we formally define the Multi-Objective Boolean Optimisation problem using the leximax criterion and highlight related work in Multi-Objective Boolean Optimisation and leximax optimisation. Section 3 describes the new SAT-based leximax optimisation algorithms. In Section 4 we evaluate our leximax optimisation algorithms against other state of the art leximax optimisation algorithms. Finally, Section 5 summarises the main contribution of our work.

2 Background

This paper assumes the standard definitions and notation of propositional logic, including the notions of Boolean variable, literal, clause, conjunctive normal form (CNF) and the Boolean satisfiability (SAT) problem [16]. Given a set of m literals $l_1, \ldots, l_m$ and respective coefficients $\omega_1, \ldots, \omega_m \in \mathbb{N}$, a Pseudo-Boolean (PB) expression is a weighted sum of literals $\sum_{i=1}^{m} \omega_i \cdot l_i$. Given an integer $k \in \mathbb{N}$, a linear PB constraint has the form $\sum_{i=1}^{m} \omega_i \cdot l_i \bowtie k, \quad \bowtie \in \{\leq, \geq, =\}$.

▶ **Definition 1** (Multi-Objective Boolean Optimisation problem (MOBO)). *An instance of MOBO is defined by a vector $f(X) = (f_1(X), \ldots, f_n(X))$ of n PB expressions and a set of PB constraints defined over a set X of Boolean variables. We assume, without loss of generality, that each objective function is a weighted sum of Boolean variables, with positive weights.*

▶ **Definition 2** (Objective vector). *Let $\alpha : X \to \{0,1\}$ be a complete assignment that satisfies all PB constraints in a Multi-Objective Boolean Optimisation instance. Given an assignment α and objective functions $f = (f_1, \ldots, f_n)$, the vector $\vec{f}(\alpha) = (f_1(\alpha), \ldots, f_n(\alpha))$ is called the objective vector where $f_i(\alpha)$ is the value of function f_i considering the assignment α.*

▶ **Definition 3** (Pareto-optimal). *Let $\vec{a} = (a_1, \ldots, a_n) \in \mathbb{N}^n$ and $\vec{b} = (b_1, \ldots, b_n) \in \mathbb{N}^n$. We write $\vec{a} \prec_{Par} \vec{b}$, if for all $i \in \{1, \ldots, k\}$, $a_i \leq b_i$ and there exists $j \in \{1, \ldots, k\}$ such that $a_j < b_j$. A feasible solution α is Pareto-optimal if there does not exist another feasible solution α' such that $\vec{f}(\alpha') \prec_{Par} \vec{f}(\alpha)$.*

▶ **Example 4.** Consider two objective vectors $(20, 20, 20)$ and $(40, 40, 40)$. In this case we have that $(20, 20, 20) \prec_{\text{Par}} (40, 40, 40)$.

▶ **Example 5.** Consider two objective vectors $(20, 20, 20)$ and $(10, 40, 40)$. In this case, we have that $(20, 20, 20) \not\prec_{\text{Par}} (10, 40, 40)$ and $(10, 40, 40) \not\prec_{\text{Par}} (20, 20, 20)$. That is, the two vectors are not comparable using the relation $\prec_{\text{Par}}$.

▶ **Definition 6** (Lexicographically optimal). *Let $\vec{a} = (a_1, \ldots, a_n) \in \mathbb{N}^n$ and $\vec{b} = (b_1, \ldots, b_n) \in \mathbb{N}^n$. We define the lexicographic relation, $\prec_{lexico}$, as follows. We write $\vec{a} \prec_{lexico} \vec{b}$ whenever there exists $i \in \{1, \ldots, n\}$ such that $a_i < b_i$ and, for all $j \in \{1, \ldots, i-1\}$, $a_j = b_j$. A feasible solution α is lexicographically optimal if there does not exist a feasible solution α' such that $\vec{f}(\alpha') \prec_{lexico} \vec{f}(\alpha)$.*

▶ **Example 7.** We have that $(10, 40, 40) \prec_{\text{lexico}} (20, 20, 20)$, since $10 < 20$.

Observe that the lexicographic relation is a strict total order. Thus, there exists exactly one lexicographically optimal objective vector.

▶ **Proposition 8.** *Every lexicographically optimal solution is Pareto-optimal [28].*

▶ **Definition 9** (Leximax-optimal). *Let $\vec{a} = (a_1, \ldots, a_n) \in \mathbb{N}^n$ and $\vec{b} = (b_1, \ldots, b_n) \in \mathbb{N}^n$. The leximax relation $\prec_{leximax}$ is defined as follows. Let $\vec{a}^{\downarrow}$ denote the n-tuple with the elements of $\vec{a}$ sorted in decreasing order. We call the i-th component of $\vec{a}^{\downarrow}$ the i-th maximum of $\vec{a}$. The tuples $\vec{a}$ and $\vec{b}$ are leximax-indistinguishable if $\vec{a}^{\downarrow} = \vec{b}^{\downarrow}$. We write $\vec{a} \prec_{leximax} \vec{b}$, if $\vec{a}^{\downarrow} \prec_{lexico} \vec{b}^{\downarrow}$. A feasible solution α is leximax-optimal if there does not exist a feasible solution α' such that $\vec{f}(\alpha') \prec_{leximax} \vec{f}(\alpha)$.*

▶ **Example 10.** We have that $(20, 20, 20) \prec_{\text{leximax}} (10, 40, 40)$, since the following holds for their sorted versions: $(20, 20, 20) \prec_{\text{lexico}} (40, 40, 10)$.

The leximax relation is not trichotomous, since any permutation of the components of a vector is indistinguishable (e.g. $(10, 20)^{\downarrow} = (20, 10)^{\downarrow}$). However, since the lexicographic relation is a strict total order, there is exactly one leximax-optimal *sorted* objective vector. Any permutation of the values of a leximax-optimal objective vector is also leximax-optimal. So there are at most $n!$ leximax-optimal objective vectors, where n is the number of objectives.

▶ **Proposition 11.** *Every leximax-optimal solution is Pareto-optimal [28].*

Algorithm 1 ILP-based leximax optimisation algorithm.

```
Input: Integer Linear Programming constraints $C$ and objective functions $f_1, \ldots, f_n$.
Output: A leximax-optimal solution of the problem, $\alpha$.
1 for $i \leftarrow 1$ to $n$ do
2     $C \leftarrow C \cup \{0 \leq r_j^i \leq 1 : j = 1, \ldots, n\}$
3     $C \leftarrow C \cup \{f_j \leq v_i + r_j^i M : j = 1, \ldots, n\}$
4     $C \leftarrow C \cup \left\{\sum_{j=1}^{n} r_j^i \leq i - 1\right\}$
5     $\alpha \leftarrow \min(v_i, C)$
6     $C \leftarrow C \cup \{v_i = \alpha(v_i)\}$
7 return $\alpha$
```

Related Work in Multi-Objective Boolean Optimisation

There are several frameworks based on stochastic search in order to approximate MOBO [24, 71]. These stochastic solvers can sometimes be complemented with the selective integration of constraint solvers [36, 69].

In recent years, several SAT-based algorithms have been proposed that enumerate all Pareto-optimal solutions. For instance, Neves et al. [68] showed that one can find all Pareto-optimal solutions by enumerating all Minimal Correction Subsets of a Boolean formula. Moreover, Soh et al. [67] show that there is a one to one correspondence between p-minimal models and Pareto-optimal solutions. More recently, a hitting set based approach has also been proposed [40] for Multi-Objective optimisation with two objective functions where one function is a black box.

There are several algorithms and applications using lexicographic optimisation [52]. In Answer Set Programming, the tool `asprin` [19, 7] uses a general algorithm that computes optimal solutions according to several criteria, including Pareto-optimal, lexicographically optimal and minmax-optimal solutions. In the context of leximax optimisation of discrete problems, we highlight the work of Bouveret and Lemaître [18]. One of the algorithms has been adapted to Integer Linear Programming (ILP) and is implemented in the Package Upgradeability solver `mccs` [56, 35, 57].

The pseudo-code is shown in Algorithm 1. Given the problem constraints C and the n objective functions, the algorithm iteratively solves single-objective ILP instances. It iterates over the number of objective functions n, and in each iteration, i, it finds the value of the i-th maximum of the objective vector, using the integer variable v_i. To find the i-th maximum it defines n new Boolean variables $r_1^i, \ldots, r_n^i$ (line 2) in order to relax the constraints on the maximum value of each objective function (line 3). The idea is that for all objective functions f_j, $j = 1, \ldots, n$, we have that $f_j \leq v_i$ is enforced if and only if r_j^i is false. Observe that the constant M must be large enough so that $f_j \leq v_i + M$ is trivially satisfied (e.g., M can be an upper bound of $f_1, \ldots, f_n$). Note also that in the first iteration all objective functions must be bounded by the first maximum. In the second iteration, only $n - 1$ objective functions are bounded by the second maximum. In general, in the i-th iteration, $n - i + 1$ objective functions are bounded by the i-th maximum. This is achieved with the constraint in line 4 that limits the number of relaxation variables that can be assigned to true. The minimisation of v_i subject to C is done through an ILP solver call in line 5. At the end of iteration i, a constraint is added that fixes the value of the i-th maximum to the optimum found by the ILP call (line 6).

3 Algorithms for Leximax Optimisation

In the following sections, we present the new SAT-based leximax optimisation algorithms. The algorithms are constructed by adapting the ILP Algorithm 1 to the Boolean domain, by using several techniques available in the MaxSAT solving literature. In Section 3.1, we show how the ILP-based algorithm can be adapted to the Pseudo-Boolean domain by replacing each ILP solver call by an iterative search on the value of the objective function, using linear search SAT-UNSAT, UNSAT-SAT and binary search [45, 44, 3, 29]. In Section 3.2, we describe the SAT-based algorithms that result from the encoding of the Pseudo-Boolean constraints to CNF [27, 42, 37, 11, 1]. We focus on the translation through sorting networks [27]. Finally, in Sections 3.3 and 3.4, we present the algorithms based on an UNSAT-SAT search using unsatisfiable cores, also a technique already studied in the context of MaxSAT [51, 4, 23]. We refer to the literature on MaxSAT solving for more details on these algorithms [58, 8].

3.1 Iterative Pseudo-Boolean Algorithm

The ILP-based approach for leximax optimisation presented in Algorithm 1 can be adapted to only use Boolean variables. In particular, one can replace each ILP solver call with an iterative PB solving procedure that refines lower bounds and/or upper bounds on the i-th maximum, $i = 1, \dots, n$. In this case, the constraints $f_j \leq v_i + r^i_j M$ (line 3) are changed to $f_j \leq k + r^i_j M$, $j = 1, \dots, n$, where $k \in \mathbb{N}$. Then, instead of calling an ILP solver and minimising v_i, we repeatedly solve a PB satisfiability instance until the minimum value of k is found. For instance, one can use a linear search SAT-UNSAT procedure where, in each iteration i, a sequence of satisfiable PB instances are solved. Whenever a new feasible solution is found, a tighter upper bound UB on the value of the i-th maximum is determined. In the next PB instance, this value is decreased, by setting k to $\mathrm{UB} - 1$, and adding the PB constraints $f_j \leq k + r^i_j M$, $j = 1, \dots, n$. When the optimal value of the i-th maximum is found, we fix it by adding $f_j \leq k + r^i_j M$, $j = 1, \dots, n$ to the set of hard constraints, where k equals the optimal value of the i-th maximum. Note that this linear search SAT-UNSAT can also be replaced with a linear search UNSAT-SAT or a binary search procedure. With linear search UNSAT-SAT, we start with $k = 0$ and increase its value until the PB instance becomes satisfiable.

▶ **Example 12.** Consider a MOBO instance with the following hard constraints $\mathcal{H}$

$$\mathcal{H} = \quad x_1 + x_2 \geq 1 \quad \wedge \quad x_4 + x_5 \geq 1 \quad \wedge \quad x_3 + x_6 \geq 1.$$

Suppose we have the following two objective functions to minimise:

$$f_1 = x_1 + x_2 + x_3; \quad f_2 = x_4 + x_5 + x_6.$$

In this example, we show the execution of the PB-based linear search SAT-UNSAT algorithm on this instance using the leximax criterion. We begin by minimising $\max(f_1, f_2)$. First, we call the PB solver on the hard constraints $\mathcal{H}$. Any feasible solution α provides us with an upper bound on the value of $\max(f_1, f_2)$. In this case, our upper bound is $\mathrm{UB} = \max(f_1(\alpha), f_2(\alpha))$. Next, we check if there exists another feasible solution such that $\max(f_1, f_2) < \mathrm{UB}$. Suppose we have a satisfiable assignment α where all variables are assigned value 1 and the objective vector of α is $(3, 3)$. Hence, $\mathrm{UB} = 3$. We solve the PB instance with the following constraints:

$$\mathcal{H} \quad \wedge \quad x_1 + x_2 + x_3 \leq 2 \quad \wedge \quad x_4 + x_5 + x_6 \leq 2.$$

The instance is satisfiable and suppose we have a new solution α' where x_2 and x_4 are assigned to 0, while the other variables are assigned value 1. In this case, the objective vector is $(2, 2)$. Then, we update the upper bound of $\max(f_1, f_2)$ to $\mathrm{UB} = 2$. Once more, we check if there is a feasible solution such that $\max(f_1, f_2) < \mathrm{UB}$, by solving the PB instance with constraints:

$$\mathcal{H} \quad \wedge \quad x_1 + x_2 + x_3 \leq 1 \quad \wedge \quad x_4 + x_5 + x_6 \leq 1.$$

The formula is unsatisfiable. We conclude that the current value of the upper bound, 2, is the minimum value of $\max(f_1, f_2)$. Next, we run the second iteration, where we minimise the second maximum. First, we fix the value of $\max(f_1, f_2)$ to the optimum, 2, by adding to $\mathcal{H}$ the PB constraints:

$$x_1 + x_2 + x_3 \leq 2 \quad \wedge \quad x_4 + x_5 + x_6 \leq 2.$$

From the previous objective vector $(2, 2)$ we obtain an upper bound on the second maximum of (f_1, f_2) of $\mathrm{UB} = 2$. We define two new Boolean variables, r_1 and r_2, such that, the PB constraint $f_j \leq \mathrm{UB} - 1$ is enforced if and only if r_j is false, $j = 1, 2$. Hence, a new call is made with additional PB constraints:

$$\mathcal{H} \quad \wedge \quad x_1 + x_2 + x_3 \leq 1 + Mr_1 \quad \wedge \quad x_4 + x_5 + x_6 \leq 1 + Mr_2 \quad \wedge \quad r_1 + r_2 \leq 1, \quad (1)$$

where M is a large constant. The value of M must be large enough so that when r_j is true the constraints $f_j \leq \mathrm{UB} - 1 + Mr_j$ do not restrict more than the remaining constraints, $j = 1, 2$. For example, M can be 2, since $f_j \leq 2$, for $j = 1, 2$, because we have fixed the value of the first maximum.

The formula (1) is satisfiable. Let α'' be a satisfiable assignment to (1) with variables x_1, x_3, x_4, r_1 assigned value 1 and the remaining variables assigned value 0. In this case we have the objective vector $(2, 1)$ and the new upper bound for the second maximum is 1. Finally, a new call is made on the following formula:

$$\mathcal{H} \quad \wedge \quad x_1 + x_2 + x_3 \leq 0 + Mr_1 \quad \wedge \quad x_4 + x_5 + x_6 \leq 0 + Mr_2 \quad \wedge \quad r_1 + r_2 \leq 1.$$

Since this formula is unsatisfiable, then α'' is a leximax-optimal solution. Note that having x_1, x_4, x_6 assigned value 1 is also a leximax-optimal solution with the objective vector $(1, 2)$.

3.2 Iterative SAT-based Algorithm using Sorting Networks

The algorithm described in the previous section uses a PB solver. However, one can replace the PB solver with a SAT solver. For that, all constraints of the MOBO instance must be translated to CNF using one of the many available encodings [9, 66, 6, 62, 37, 12, 10, 27, 2].

Besides the original PB constraints, one also needs to encode into CNF the additional constraints added during the search procedure. First, note that a cardinality constraint is added on the relaxation variables (see line 4 in Algorithm 1). These cardinality constraints are usually small, since the size is bounded by the number of objective functions.

Finally, one also needs to deal with the constraints that bound the value of the objective functions. For that, we use an encoding based on sorting networks [43, 27]. In the following, we assume the objective functions to be cardinality expressions for ease of explanation. In case we have a PB expression, a unary encoding from PB to CNF could be used.

Instead of encoding to CNF constraints of the form $f_j \leq k + Mr_j^i$, where f_j is the objective function and r_j^i is the relaxation variable, we encode to CNF constraints of the form $f_j \leq k$. As a result, the relaxation of these constraints is dealt with in a different way, without requiring a large constant M.

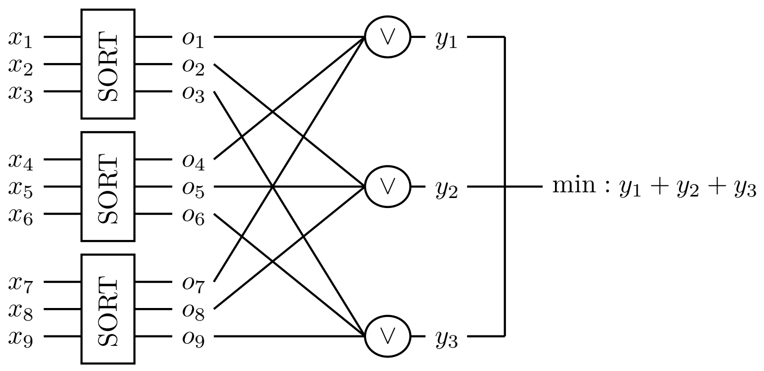

Figure 1 Schematic depiction of the encoding of the 1st iteration of the SAT-based algorithm.

Figure 1 illustrates our encoding for finding the first maximum for three objective functions. The encoding starts by adding to the set of hard of constraints, $\mathcal{H}$, a sorting network encoding the expression for each objective function. Therefore, for each objective function f_j, we get fresh Boolean variables $o_{1,j}, \ldots, o_{m_j,j}$, corresponding to the output of the sorting network of f_j, $j = 1, \ldots, n$, where m_j is the value of f_j when all its variables are true. Hence, $f_j = \sum_{i=1}^{m_j} o_{i,j}$, and $o_{1,j}, \ldots, o_{m_j,j}$ is sorted in increasing order. Effectively, we obtain unary representations of the value of the objective functions. If a given variable $o_{v,j}$ is true, this means that the value of objective function f_j is at least $m_j - v + 1$. Otherwise, if $o_{v,j}$ is false, then the value of objective function f_j is smaller than $m_j - v + 1$.

An important property of the sorting network encoding is that the componentwise disjunction between the outputs of the sorting networks gives us $\max(f_1, \ldots, f_n)$. For example, the componentwise disjunction between the sorted vectors $(0, 1, 1)$ and $(0, 0, 1)$ is $(0 \vee 0, 1 \vee 0, 1 \vee 1) = (0, 1, 1)$, which corresponds to the vector with the largest number of ones. Then, we add fresh Boolean variables $y_1, \ldots, y_m$ such that $(y_1, \ldots, y_m)$ is the sorted vector of the componentwise disjunction, where $m = \max(m_1, \ldots, m_n)$. The PB constraints enforcing that the i-th maximum is upper bounded by k are replaced by the unit clause $\neg y_{m-k}$, if $k < m$. In the case of Figure 1, we have three objective functions $f_1 = x_1 + x_2 + x_3, f_2 = x_4 + x_5 + x_6, f_3 = x_7 + x_8 + x_9$.

In the iterations where $i > 1$, it is necessary to consider that some objective functions are relaxed. Hence, for each objective function f_j, we add fresh Boolean variables $s^i_{1,j}, \ldots, s^i_{m_j,j}$, $j = 1, \ldots, n$. In addition, we add to $\mathcal{H}$ clauses enforcing that if r^i_j is false (i.e. objective function f_j is not relaxed when finding the i-th maximum), then $(s^i_{1,j}, \ldots, s^i_{m_j,j})$ is equal to $(o_{1,j}, \ldots, o_{m_j,j})$, for $j = 1, \ldots, n$. Finally, instead of doing the componentwise disjunction between the outputs of the sorting networks, we do it between the new vectors $(s^i_{1,j}, \ldots, s^i_{m_j,j})$, $j = 1, \ldots, n$. Note that the relaxation variables allows some of these vectors to assume arbitrary values. However, since we are minimising, those vectors can safely be assigned to false and will not affect the componentwise disjunction. As a result, the vector resulting from the componentwise disjunction, $(y^i_1, \ldots, y^i_m)$, will correspond to the maximum of $n - i + 1$ objective functions. Figure 2 illustrates the encoding used for the second iteration. The next iterations follow the same schema. Observe that the sorting networks do not have to be rebuilt between iterations. The encoding of the objective function expressions using the sorting networks is done only once.

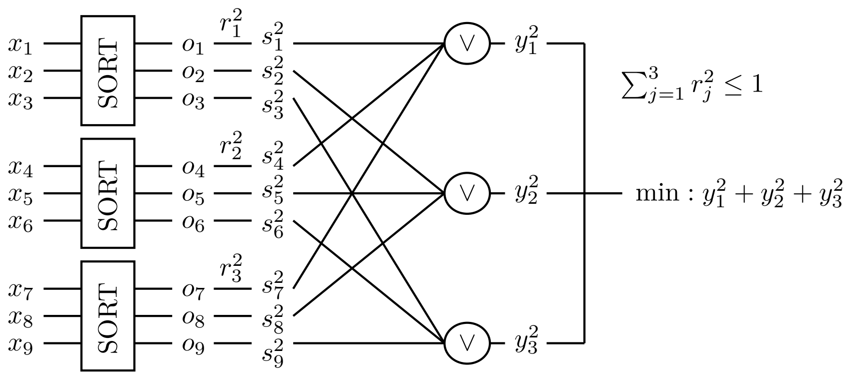

Figure 2 Schematic depiction of the encoding of the 2nd iteration of the SAT-based algorithm.

Algorithm 2 Core-guided leximax optimisation algorithm.

```
Input: A set of hard constraints $\mathcal{H}$ and objective functions $f_1, \ldots, f_n$.
Output: A leximax-optimal solution $\alpha$.
1  $\mathcal{X} \leftarrow \bigcup_{j=1}^{n} \{\neg x : x \in f_j\}$
2  for $i \leftarrow 1$ to $n$ do
3      $\mathcal{H} \leftarrow \mathcal{H} \cup \left\{\sum_{j=1}^{n} r_j^i \leq i - 1\right\}$
4      LB $\leftarrow 0$
5      repeat
6          $(st, \alpha, \mathcal{C}) \leftarrow \texttt{SAT}(\mathcal{H} \cup \{(\texttt{EncodeCNF}(f_j \leq \text{LB}) \vee r_j^i) : j = 1, \ldots, n\} \cup \mathcal{X})$
7          if $st$ = False then
8              if $\mathcal{C} \cap \mathcal{X} = \emptyset$ then LB $\leftarrow$ LB $+ 1$
9              else $\mathcal{X} \leftarrow \mathcal{X} \setminus \mathcal{C}$
10     until $st$
11     $\mathcal{H} \leftarrow \mathcal{H} \cup \{(\texttt{EncodeCNF}(f_j \leq \text{LB}) \vee r_j^i) : j = 1, \ldots, n\}$
12 return $\alpha$
```

3.3 Core-guided Algorithm

In this section, we present the structure of our core-guided algorithm for leximax Boolean optimisation. The main goal of the core-guided algorithm is to only consider a subset of the variables in each objective function, thus improving the performance of each call.

Given a CNF formula φ, we assume that a call to a SAT solver to check the satisfiability of φ returns a triple $(st, \alpha, \mathcal{C})$, where st is a Boolean that is true if and only if φ is satisfiable. If φ is satisfiable, then α contains a satisfying assignment. Otherwise, if φ is unsatisfiable, then $\mathcal{C}$ contains an unsatisfiable subformula (also known as an unsatisfiable core) of φ.

Algorithm 2 shows the pseudocode of the core-guided algorithm. First, we define a set $\mathcal{X}$ with unit clauses that are the negation of the variables in all objective functions $f_1, \ldots, f_n$. As in Algorithm 1, we iterate over the number of objective functions and in each iteration we find the smallest i-th maximum considering $n - i + 1$ objective functions. The main difference between Algorithm 1 and Algorithm 2 is that, in each iteration, we perform an UNSAT-SAT search procedure on the value of the i-th maximum initially assuming all variables in the objective functions are false, i.e. initially all are set to the optimum value. For finding the minimum i-th maximum, we start with a lower bound LB of 0 (line 4). Then, a SAT call bounds the value of the non-relaxed objective functions to LB. If the formula in line 6 is unsatisfiable and it does not depend on the assumed values in $\mathcal{X}$, then we can safely increase the lower bound (line 8). Otherwise, we remove from $\mathcal{X}$ the unit clauses that appear in the unsatisfiable core $\mathcal{C}$ (line 9). At the end of iteration i, the variable LB contains the smallest possible value for the i-maximum of the objective functions. Hence, we can fix this value (line 11).

3.4 SAT-based Core-guided Algorithm with Sorting Networks

Algorithm 2 can be implemented using different CNF encodings for bounding the objective functions (lines 6 and 11). In this section, we detail Algorithm 2 using the sorting networks representation proposed in Section 3.2.

Note that in the core-guided algorithm proposed in Algorithm 2, the CNF encoding of PB constraints that bound the objective functions is simplified due to the unit clauses in the set $\mathcal{X}$ that fix the values of some variables. In developing this algorithm, the same technique using sorting networks from Section 3.2 can be used to encode the objective functions, and use the set $\mathcal{X}$ to unit propagate the fixed values from the input of the sorting networks to the output. However, we propose to improve on this. Instead of representing the entire objective functions in the beginning of the algorithm, we propose to construct sorting networks only for the objective variables that are not negated in $\mathcal{X}$. As a result, the sorting networks are built dynamically and incrementally. At first, the sorting networks are empty and grow as more variables from the objective functions are removed from $\mathcal{X}$ due to the unsatisfiable cores found during the SAT calls.

Using dynamic sorting networks raises some challenges. There are two main options for growing the sorting networks: (1) build a new sorting network from scratch or (2) grow the previous sorting network incrementally. However, the first case of rebuilding the sorting network might result in losing incremental SAT solving between iterations. An alternative would be to add a new blocking variable to the entire encoding of a sorting network and control if a given sorting network is active or not using the blocking variable and SAT solver assumptions. Nevertheless, this last alternative still implies rebuilding the sorting networks and the size of the SAT formula might become too large to handle efficiently. Due to these drawbacks, we propose to reuse the sorting network and make it grow in an incremental fashion. The incremental encoding to CNF has been the subject of previous research work [27, 53] but it is new in the context of leximax optimisation.

Consider there is a sorting network that encodes some objective function and a new set of variables is to be added. In that case, we can build a sorting network for the new variables and apply a merge encoding between both sorting networks. Figure 3 illustrates the proposed schema for growing a sorting network where the Batcher's odd-even merge construction [13, 43] can be used.

A sorting network with k inputs with Batcher's odd-even merge sorting network uses $\Theta(k(\log(k))^2)$ comparators. There are no known constructions that produce sorting networks with smaller size complexity that can be used in practice. Thus, a sorting network with k inputs is encoded with $\Omega(k(\log(k))^2)$ clauses.

There are situations where the proposed sort-and-merge approach may lead to a larger growth of the number of comparators (and thus the number of clauses) than expected. The issue is that if, each time we obtain a new unsatisfiable core, we remove the variables from $\mathcal{X}$ and add them straight away to the sorting networks, the number of comparators may grow with k^2, instead of $k(\log(k))^2$. The odd-even merge construction produces $\Theta(k\log(k))$ comparators when merging two sequences of size k, and produces $\Theta(k)$ comparators when

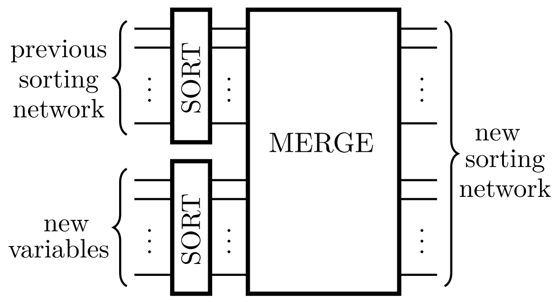

Figure 3 General schema of growing a sorting network using sort and merge.

merging two sequences with size k and 1. Therefore, the odd-even merge sorting network, which works by recursively sorting two subsequences and then merging them, only produces $\Theta(k(\log(k))^2)$ comparators when, in each recursive step, the sequence is cut in half. However, if we have the scenario of merging a sequence of $k-1$ elements and a sequence of 1 element, the final sorting network will contain $\Theta(k^2)$ comparators. In the core-guided algorithm proposed in Algorithm 2, in the worst case, the unsatisfiable cores contain exactly one of the clauses in $\mathcal{X}$. As a result, it is possible to end up with sorting networks with k inputs having $\Theta(k^2)$ comparators.

In order to try to limit the worst-case scenario where the encoding of the objective functions becomes quadratic, we propose to delay the merge process between the previous sorting network and the new variables to be added. Hence, when an unsatisfiable core is identified in the SAT call (line 6 of Algorithm 2), we remove from $\mathcal{X}$ the clauses in the core, but the respective variables are not immediately added to the sorting networks. Instead, the variables from the clauses in the unsatisfiable core are added to a set $\mathcal{S}$. Next, the SAT solver is repeatedly called with a smaller set $\mathcal{X}$ of unit clauses. For each new unsatisfiable core $\mathcal{C}$, clauses from $\mathcal{C} \cap \mathcal{X}$ are removed from $\mathcal{X}$ and their respective variables are added to $\mathcal{S}$. When the SAT call becomes satisfiable, then we add the variables in $\mathcal{S}$ to the sorting networks and the set $\mathcal{S}$ is emptied.

This procedure consists in finding as many cores as possible of a formula that are disjoint with regard to $\mathcal{X}$. From now on, we refer to this strategy as the *disjoint cores strategy*. The use of disjoint cores is another idea borrowed from the MaxSAT solving literature [22, 14]. Note that this option may reduce the number of times the sorting networks are extended during the algorithm's execution. When extending the sorting networks by sort-and-merge, the disjoint cores strategy can prevent the worst case quadratic growth of the number of comparators.

▶ **Example 13.** This example focuses solely on the delay of including literals from unsatisfiable cores into the sorting network representation of the objective functions.

Consider a MOBO instance with hard clauses $\mathcal{H}$ and two objective functions:

$$f_1 = x_1 + x_2 + x_3 + x_4; \quad f_2 = x_5 + x_6 + x_7 + x_8.$$

Assume at some point in the execution of the core-guided algorithm with dynamic sorting networks $\{x_1, x_2\}$ and $\{x_5, x_6\}$ are in the sorting networks and $\mathcal{X} = \{\neg x_3, \neg x_4, \neg x_7, \neg x_8\}$. Suppose we are minimising $\max(f_1, f_2)$ and the lower bound is currently 1. The next call to the SAT solver is testing if there exists a feasible solution for the following formula:

$$\mathcal{H} \quad \wedge \quad \bigwedge_{l \in \mathcal{X}} l \quad \wedge \quad x_1 + x_2 \leq 1 \quad \wedge \quad x_5 + x_6 \leq 1. \tag{2}$$

Assume the formula is unsatisfiable and the core $\mathcal{C}$ is such that $\mathcal{C} \cap \mathcal{X} = \{\neg x_3\}$. In this case, $\mathcal{X}$ is updated to $\{\neg x_4, \neg x_7, \neg x_8\}$. There are two options. The first option is to add x_3 to the sorting network of f_1 and continue with the algorithm by calling the SAT solver on the formula

$$\mathcal{H} \quad \wedge \quad \bigwedge_{l \in \mathcal{X}} l \quad \wedge \quad x_1 + x_2 + x_3 \leq 1 \quad \wedge \quad x_5 + x_6 \leq 1. \tag{3}$$

However, as previously explained, this option may lead to an undesirable growth of the number of clauses in the objective function CNF representation. Hence, using the disjoint cores strategy, x_3 is not added to the sorting network right away. Instead we store x_3 in a set $\mathcal{S}$, but we still remove $\neg x_3$ from $\mathcal{X}$. Then, we test once again if (2) is true, but now x_3 can be true because it is no longer in $\mathcal{X}$.

Assume the formula is still unsatisfiable and the new unsatisfiable core $\mathcal{C}$ is such that $\mathcal{C} \cap \mathcal{X} = \{\neg x_7\}$. We remove $\neg x_7$ from $\mathcal{X}$ and add x_7 to $\mathcal{S}$, so now $\mathcal{S} = \{x_3, x_7\}$ and $\mathcal{X} = \{\neg x_4, \neg x_8\}$. We check again the formula (2) where x_7 can now be true. Assume the formula is still unsatisfiable and the new core $\mathcal{C}$ is such that $\mathcal{C} \cap \mathcal{X} = \{\neg x_4\}$. Then, $\neg x_4$ is removed from $\mathcal{X}$ and x_4 is added to $\mathcal{S}$. Assume that (2) becomes satisfiable. Notice that this does not mean the optimal solution has been found, as $\max(f_1, f_2)$ may not be 1 for the feasible solution. We need to add the variables in $\mathcal{S}$ to the sorting networks and then check if there exists a feasible solution such that

$$\mathcal{H} \quad \wedge \quad \bigwedge_{l \in \mathcal{X}} l \quad \wedge \quad x_1 + x_2 + x_3 + x_4 \leq 1 \quad \wedge \quad x_5 + x_6 + x_7 \leq 1, \tag{4}$$

with $\mathcal{X} = \{\neg x_8\}$. If the formula (4) is unsatisfiable, we repeat the previous steps of removing literals from $\mathcal{X}$ (in this case only $\neg x_8$ is left) until the formula becomes satisfiable.

4 Evaluation

This section evaluates the proposed SAT-based algorithms for leximax optimisation. Besides comparing the different strategies of our algorithms, we compare against the state of the art ILP-based approach. The algorithms based on Constraint Programming [18] are not included since we were unable to find a publicly available implementation.

4.1 Use Case: Package Upgradeability

The Package Upgradeability problem is an NP-complete problem [26] that arises when a user of a software system (e.g. Linux distributions) wants to install, remove or upgrade software packages. In the final installation, for each package p_i that is installed, all its dependencies must be satisfied and there cannot be any other installed package p_j such that p_i and p_j are conflicting. In addition, the user can define several objective functions to be minimised, such as the number of newly installed packages, the number of removed packages, or the number of not up-to-date packages. Therefore, the Package Upgradeability problem can be modelled as a MOBO formula. Furthermore, it is often the case that the user is unable to define a proper order of preferences among the several objectives. Hence, our evaluation is focused on finding a leximax-optimal solution for the Package Upgradeability problem.

There are several Package Upgradeability solvers that rely on encodings to Answer Set Programming [31], Maximum Satisfiability [41, 38], Pseudo-Boolean Optimisation [5] and ILP [57, 35]. However, most tools only compute lexicographically optimal solutions of the problem. Only `mccs` [56, 57] (version 1.1) is able to compute leximax-optimal solutions, using the ILP-based algorithm described in Algorithm 1. `mccs` can be used with different ILP solvers. Hence, we configured and tested `mccs` with the following ILP solvers: `CPLEX` [21] (version 12.10.0), `Gurobi` [34] (version 9.0.3), `SCIP` [30, 65] (version 7.0.1), `Cbc` [20] (version devel, build Jan 14 2021), `GLPK` [32] (version 4.65) and `lpsolve` [47] (version 5.5.2.5).

4.2 Implementation and Benchmarks

The new SAT-based algorithms and the existing ILP-based algorithm were implemented in a tool for leximax optimisation that is publicly available [17]. In the implementation of the SAT-based algorithms, the Batcher's odd–even merge procedure [13, 43] was used for constructing and merging sorting networks. The sorting network encoding of MiniSat+ [27] was used without coefficient decomposition and interconnected sorting networks since we focus on solving unweighted instances (e.g. Package Upgradeability). To evaluate the SAT-based algorithms on the Package Upgradeability domain, the `packup` [41, 38, 64] Package Upgradeability solver was linked with our tool. The algorithms were evaluated using the incremental SAT solving library of `CaDiCaL` [15] (version 1.3.1).

The Package Upgradeability solvers, `packup` and `mccs`, were executed on a set of 142 Package Upgradeability benchmarks [48] from the Mancoosi International Solver Competition [49]. The objective functions defined in the Mancoosi competition are the following: *removed*, *notuptodate*, *changed*, *unsat_recommends* and *new*. For each of the 142 benchmarks we generated instances considering all 26 combinations of two, three, four and five objective functions, resulting in 3692 instances. Of the 3692 instances, 122 instances correspond to unsatisfiable instances or instances that are reduced to the single-objective case. As a result, the final benchmark set contains 3570 instances to find a leximax-optimal solution.

In order to broaden the experimental evaluation, we adapted a subset of benchmark instances from the MaxSAT Evaluation 2021 [55] to the multi-objective domain. From the 561 benchmarks of the unweighted track of the MaxSAT Evaluation 2021, we selected 100 instances that encode problems from different domains, such that the MaxSAT solver `Open-WBO` [54, 63] outperforms `Gurobi` within a 10 s timeout. For each of those benchmarks, a MOBO instance was generated by keeping the same set of constraints and by randomly partitioning the original set of soft clauses into multiple sets of soft clauses. We considered partitions into two, three and four objective functions, resulting in a set of 300 instances [59].

All algorithms were executed on a single thread, with 180 seconds CPU time limit for the Package Upgradeability instances and 3600 seconds CPU time limit in the case of MaxSAT-based instances. The experiments were run on Intel(R) Xeon(R) E5-2630 v2 CPU 2.60GHz machines with Debian Linux operating system with 4 GB memory limit for each instance.

4.3 Evaluation of the SAT-based Algorithms

This section analyses the performance of the different versions of the proposed SAT-based algorithms on the set of Package Upgradeability benchmarks.

First we describe the abbreviations used when presenting the results for these versions. There are three versions of Algorithm 1 using the SAT encoding as described in Section 3.2, each corresponding to different types of search for the i-th maximum: the linear search SAT-UNSAT algorithm is `linear-su`, linear search UNSAT-SAT is `linear-us` and binary search is `binary`. There are several versions of the core-guided algorithm proposed in Section 3.4. The version where the sorting networks are rebuilt with non-incremental SAT solving is named `core-rebuild`. If the sorting networks are rebuilt, but using incremental SAT solving then we use `core-rebuild-incr`. If the objective functions are statically built

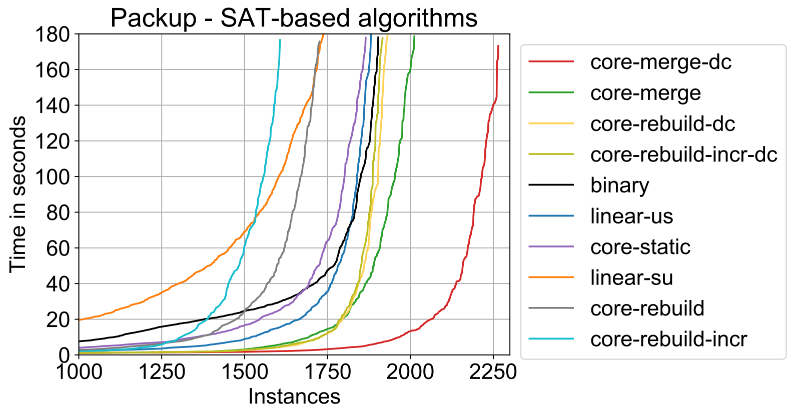

Figure 4 Run times of SAT-based algorithms on Package Upgradeability instances (180 seconds).

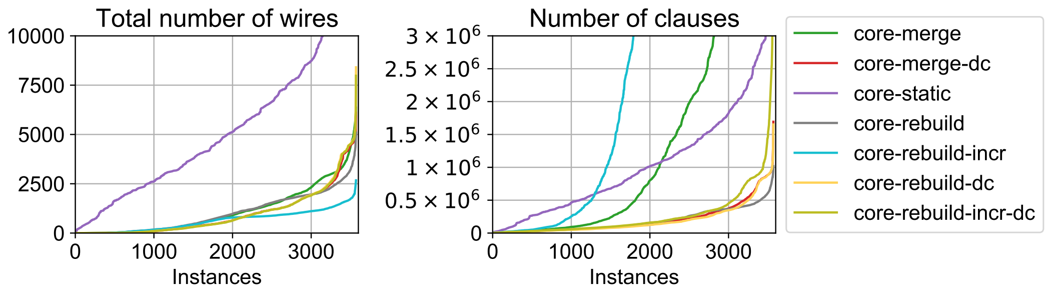

Figure 5 Cactus plots of the sum of the number of wires of the sorting networks (on the left) and the number of clauses (on the right) at the time of the last SAT call.

at the beginning of the algorithm we use `core-static`. The version with dynamic sorting networks that extends the objective functions by sorting the new variables and merging with the previous sorting network is `core-merge`. If the additional disjoint cores strategy is used, then the algorithms' names are appended with '`-dc`'.

Figure 4 shows a cactus plot with the run times of solved Package Upgradeability instances by SAT-based algorithms. Among the different versions of the iterative SAT-based algorithm, the best performing was the binary search. Nevertheless, despite being able to solve more instances within the time limit of 180 seconds, it is usually slower than the UNSAT-SAT search when this approach is able to solve the instance. For example, the plot clearly shows that the number of instances solved by the binary search within 20 seconds is much smaller.

The results from Figure 4 also show that the core-guided approach using the proposed incremental merge of sorting networks is the best performing algorithm. Moreover, delaying the merge operation by using the disjoint cores strategy results in a significant performance boost. Observe that the disjoint cores strategy also greatly improves the algorithms that rebuild the sorting networks (both the incremental and non-incremental versions).

To better understand the difference in the performance of the algorithms, we analyse the number of wires of the sorting networks and the number of clauses of the formula. Figure 5 contains two cactus plots. Both plots correspond to the Package Upgradeability benchmarks. The plot on the left shows the total number of wires of the last sorting networks produced by each algorithm. The plot on the right shows the number of clauses of the last call to the SAT solver. As expected, using dynamic sorting networks allows to significantly reduce the

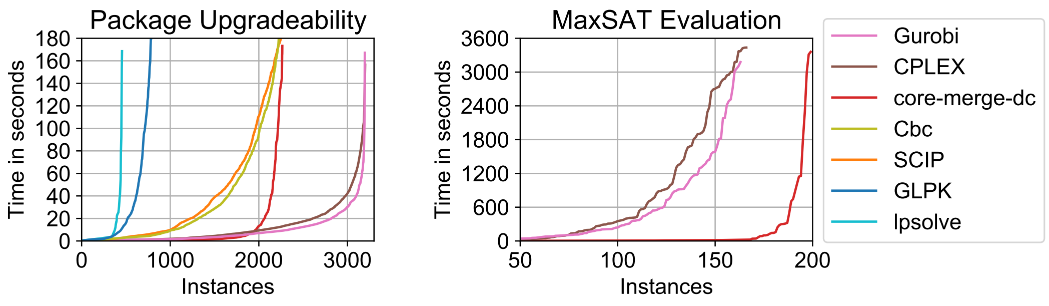

Figure 6 Run times of solved instances on Package Upgradeability benchmarks (left) and generated multi-objective instances based on the MaxSAT Evaluation 2021 (right).

number of wires of the sorting networks when compared with the static representation of the objective functions. This occurs because not all variables are represented in the objective functions. The goal is that the reduction in the number of wires allows a reduction in the number of comparators and ultimately in the number of clauses. Indeed, we observe a significant reduction in the number of clauses with dynamic sorting networks, except for `core-rebuild-incr`, because it does not delete the previous sorting network encodings, and `core-merge`, because of the growth of the number of comparators when merging. Since there are many small unsatisfiable cores, the growth in `core-merge` tends to approximate the worst case scenario explained in Section 3.4. Due to this growth on the size of the formula, `core-rebuild-incr` and `core-merge` exceeded the 4 GB memory limit in around 30% and 4% of the instances, respectively. The right plot of Figure 5 clearly shows the impact of using the disjoint cores strategy, resulting in a significant reduction in the size of the final sorting networks. Note that `core-merge-dc` and `core-rebuild-incr-dc` both use incremental SAT solving, and produce a similar number of clauses. However, the sort-and-merge approach proved to be more effective than the rebuild approach, as `core-merge-dc` solved more instances with overall faster run times than `core-rebuild-incr-dc`.

4.4 Comparison with ILP solvers

This section provides a comparison between the best performing core-guided SAT approach `core-merge-dc` and the iterative ILP algorithm when using different ILP solvers.

Figure 6 shows the cactus plots of the run times of solved instances, for each solver. On the left we the results for the Package Upgradeability benchmarks and on the right the results on the instances generated from the MaxSAT Evaluation 2021.

In the case of Package Upgradeability, the overall results show that when using commercial solvers `Gurobi` and `CPLEX`, the iterative ILP approach solves more instances. However, the plot also shows that our approach can outperform the iterative ILP algorithm when using non-commercial solvers in both the number of instances and run-times. We note that in many real-world cases, such as the Package Upgradeability problem in open-source Linux distributions, the usage of commercial solvers is not feasible.

On the MaxSAT Evaluation 2021 instances, we tested the ILP approach using commercial solvers `Gurobi` and `CPLEX`. These results suggest that our proposed algorithm can potentially outperform the ILP commercial solvers in other application domains, e.g. the Placement Fixer problem, which is not well-suited for ILP, because of the type of constraints [61].

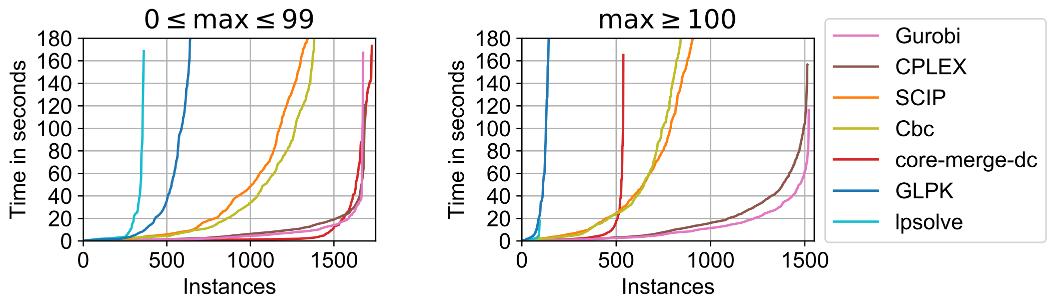

Figure 7 Run times of `core-merge-dc` and the ILP solvers on Package Upgradeability benchmarks, with an optimal maximum of the objective vector ≤ 99 on the left and ≥ 100 on the right.

Figure 7 shows two cactus plots that provide an analysis on the algorithms' performance depending on the values in the leximax-optimal objective vector in the Package Upgradeability instances. The plot on the left shows the run times for instances where the largest value in the leximax-optimal solution is smaller than 100. On the other hand, the plot on the right shows the run times for instances where the largest value is larger than or equal to 100.

Observe that when the largest value is smaller than 100, `core-merge-dc` has a similar performance to the iterative ILP algorithm when using commercial solvers `CPLEX` or `Gurobi` and clearly outperforms non-commercial solvers. On the other hand, the iterative ILP algorithm has a stronger performance for larger values in the leximax-optimal solution. This occurs since our core-guided approach is slower to converge in these situations [25]. Nevertheless, in the context of package upgradeability, it is often the case that real-world users perform incremental adjustments to a current installation. Hence, in these situations, optimal solutions tend to have a small cardinality and our proposed `core-merge-dc` algorithm excels in those scenarios.

5 Conclusions

This paper introduces the first SAT-based leximax optimisation algorithms. Besides iterative SAT-based algorithms, we also propose a new core-guided algorithm for leximax optimisation. The algorithms are built upon the effective encoding of PB constraints to CNF and a CNF encoding to determine the maximum value among several objective functions. Moreover, we explore the translation of PB constraints to CNF using sorting networks. We use a merging technique that allows to dynamically and incrementally extend the representation of objective functions. Additionally, a strategy based on the identification of disjoint cores in the core-guided algorithm allows to delay the merging process, thus resulting in a more effective encoding of the objective functions.

An experimental evaluation is carried out on the Multi-Objective Package Upgradeability Optimisation problem and on generated MOBO instances based on the MaxSAT Evaluation benchmarks. The core-guided algorithm outperforms the iterative ILP algorithm when using non-commercial solvers on the Package Upgradeability benchmarks. The results on the MaxSAT Evaluation instances suggest that our SAT-based algorithms may be competitive with the ILP algorithm with commercial solvers on other application domains.

As a future direction of research, we highlight the integration of our leximax optimisation algorithms with incomplete algorithms, such as `Polosat` [60], in the same way they are currently integrated to MaxSAT solvers. Another interesting direction of research is the development of hybrid approaches using SAT and ILP solvers for leximax optimisation, e.g. by adapting the approach of the MaxSAT solver `MaxHS` [22].

References

1 Ignasi Abío, Valentin Mayer-Eichberger, and Peter J. Stuckey. Encoding linear constraints into SAT. *CoRR*, 2020. `arXiv:2005.02073`.

2 Ignasi Abío, Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell, and Valentin Mayer-Eichberger. A New Look at BDDs for Pseudo-Boolean Constraints. *Journal Artificial Intelligence Research*, 45:443–480, 2012.

3 Fadi A Aloul, Arathi Ramani, Igor Markov, and Karem Sakallah. PBS: a backtrack-search pseudo-boolean solver and optimizer. In *Proceedings of the 5th International Symposium on Theory and Applications of Satisfiability*, pages 346–353, 2002.

4 Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. Solving (weighted) partial maxsat through satisfiability testing. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 427–440. Springer, 2009. `doi:10.1007/978-3-642-02777-2_39`.

5 Josep Argelich, Daniel Le Berre, Inês Lynce, João P. Marques Silva, and Pascal Rapicault. Solving linux upgradeability problems using boolean optimization. In Inês Lynce and Ralf Treinen, editors, *Proceedings First International Workshop on Logics for Component Configuration, LoCoCo 2010, Edinburgh, UK, 10th July 2010*, volume 29 of *EPTCS*, pages 11–22, 2010. `doi:10.4204/EPTCS.29.2`.

6 Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality Networks: a theoretical and empirical study. *Constraints*, 16(2):195–221, 2011.

7 asprin webpage. `https://potassco.org/asprin/`.

8 Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. Maximum satisfiability. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications, pages 929–991. IOS PRESS, Netherlands, 2 edition, 2021. `doi:10.3233/FAIA201008`.

9 Olivier Bailleux and Yacine Boufkhad. Efficient CNF Encoding of Boolean Cardinality Constraints. In *Principles and Practice of Constraint Programming*, volume 2833 of *LNCS*, pages 108–122. Springer, 2003.

10 Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel. A Translation of Pseudo Boolean Constraints to SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):191–200, 2006.

11 Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel. New encodings of pseudo-boolean constraints into cnf. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 181–194. Springer, 2009.

12 Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel. New Encodings of Pseudo-Boolean Constraints into CNF. In *International Conference on Theory and Applications of Satisfiability Testing*, volume 5584 of *LNCS*, pages 181–194. Springer, 2009.

13 K.E. Batcher. Sorting networks and their applications. *Proceedings of AFIPS Spring Joint Computer Conference*, 32:307–314, 1968.

14 Jeremias Berg and Matti Järvisalo. Weight-Aware Core Extraction in SAT-Based MaxSAT Solving. In J. Christopher Beck, editor, *Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10416 of *Lecture Notes in Computer Science*, pages 652–670. Springer, 2017. `doi:10.1007/978-3-319-66158-2_42`.

15 Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020, 2020.

16 Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

17 Boolean Leximax Optimisation solver based on iterative SAT solving, source code. `https://github.com/miguelcabral/leximaxIST`.

18 Sylvain Bouveret and Michel Lemaître. Computing leximin-optimal solutions in constraint networks. *Artificial Intelligence*, 173(2):343–364, 2009.

19 Gerhard Brewka, James Delgrande, Javier Romero, and Torsten Schaub. asprin: Customizing answer set preferences without a headache. *AAAI*, 2015.

20 Cbc solver webpage. `https://github.com/coin-or/Cbc`.

21 CPLEX Optimizer, IBM webpage. `https://www.ibm.com/analytics/cplex-optimizer`.

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

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

24 K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimisation: NSGA-II. In *International Conference on Parallel Problem Solving from Nature*, pages 849–858. Springer, 2000.

25 Detailed experimental results of the experimental evaluation on the package upgradeability problem. `https://sat.inesc-id.pt/~mcabral/tables/detailed/`.

26 Roberto Di Cosmo, Berke Durak, Xavier Leroy, Fabio Mancinelli, and Jérôme Vouillon. Maintaining large software distributions: new challenges from the FOSS era. In *Proceedings of the FRCSS 2006 workshop*, 2006. EASST Newsletter.

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

28 Matthias Ehrgott. *Multicriteria Optimization*. Springer-Verlag, Berlin, Heidelberg, 2005.

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

30 Gerald Gamrath, Daniel Anderson, Ksenia Bestuzheva, Wei-Kun Chen, Leon Eifler, Maxime Gasse, Patrick Gemander, Ambros Gleixner, Leona Gottwald, Katrin Halbig, Gregor Hendel, Christopher Hojny, Thorsten Koch, Pierre Le Bodic, Stephen J. Maher, Frederic Matter, Matthias Miltenberger, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Christine Tawfik, Stefan Vigerske, Fabian Wegscheider, Dieter Weninger, and Jakob Witzig. The SCIP Optimization Suite 7.0. Technical report, Optimization Online, March 2020. URL: `http://www.optimization-online.org/DB_HTML/2020/03/7705.html`.

31 Martin Gebser, Roland Kaminski, and Torsten Schaub. aspcud: A linux package configuration tool based on answer set programming. *Electronic Proceedings in Theoretical Computer Science*, 65, September 2011. `doi:10.4204/EPTCS.65.2`.

32 GLPK (GNU Linear Programming Kit) webpage. `https://www.gnu.org/software/glpk/`.

33 Andreia P. Guerreiro, Vasco M. Manquinho, and José Rui Figueira. Exact hypervolume subset selection through incremental computations. *Comput. Oper. Res.*, 136:105471, 2021. `doi:10.1016/j.cor.2021.105471`.

34 Gurobi webpage. `https://www.gurobi.com/`.

35 Gustavo Guttierez, Mikoláš Janota, Inês Lynce, Olivier Lhomme, Vasco Manquinho, João Marques-Silva, and Claude Michel. Mancoosi deliverable 4.3: Final version of the optimization algorithms and tools. Technical report, Mancoosi, 2011. URL: `https://www.mancoosi.org/reports/d4.3.pdf`.

36 C. Henard, M. Papadakis, M. Harman, and Y. Le Traon. Combining Multi-Objective Search and Constraint Solving for Configuring Large Software Product Lines. In *International Conference on Software Engineering*, pages 517–528, 2015.

37 Steffen Hölldobler, Norbert Manthey, and Peter Steinke. A compact encoding of pseudo-boolean constraints into SAT. In *KI 2012: Advances in Artificial Intelligence*, volume 7526 of *LNCS*, pages 107–118. Springer, 2012.

38 Alexey Ignatiev, Mikolás Janota, and João Marques-Silva. Towards efficient optimization in package management systems. In Pankaj Jalote, Lionel C. Briand, and André van der Hoek, editors, *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, pages 745–755. ACM, 2014. `doi:10.1145/2568225.2568306`.

39 Mikolás Janota, Inês Lynce, Vasco M. Manquinho, and João Marques-Silva. Packup: Tools for package upgradability solving. *J. Satisf. Boolean Model. Comput.*, 8(1/2):89–94, 2012. `doi:10.3233/sat190090`.

40 Mikolás Janota, António Morgado, José Fragoso Santos, and Vasco M. Manquinho. The seesaw algorithm: Function optimization using implicit hitting sets. In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*, volume 210 of *LIPIcs*, pages 31:1–31:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. `doi:10.4230/LIPIcs.CP.2021.31`.

41 Mikoláš Janota, Inês Lynce, Vasco Manquinho, and Joao Marques-Silva. PackUp: Tools for Package Upgradability Solving: System description. *Journal on Satisfiability, Boolean Modeling and Computation*, 8, January 2012. `doi:10.3233/SAT190090`.

42 Saurabh Joshi, Ruben Martins, and Vasco M. Manquinho. Generalized totalizer encoding for pseudo-boolean constraints. In Gilles Pesant, editor, *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings*, volume 9255 of *Lecture Notes in Computer Science*, pages 200–209. Springer, 2015. `doi:10.1007/978-3-319-23219-5_15`.

43 D.E. Knuth. *The Art of Computer Programming: Fundamental algorithms*. Number vol. 1 in Addison-Wesley series in computer science and information processing. Addison-Wesley, 1997.

44 Miyuki Koshimura, Tong Zhang, Hiroshi Fujita, and Ryuzo Hasegawa. Qmaxsat: A partial max-sat solver system description. *Journal on Satisfiability, Boolean Modeling and Computation*, 8, 2012. `doi:10.3233/SAT190091`.

45 Daniel Le Berre and Anne Parrain. The sat4j library, release 2.2. *Journal on Satisfiability Boolean Modeling and Computation*, 7:59–64, July 2010. `doi:10.3233/SAT190075`.

46 Rui Li, Qinghua Zheng, Xiuqi Li, and Zheng Yan. Multi-objective optimization for rebalancing virtual machine placement. *Future Gener. Comput. Syst.*, 105:824–842, 2020. `doi:10.1016/j.future.2017.08.027`.

47 lpsolve webpage. `https://sourceforge.net/projects/lpsolve/`.

48 Benchmarks from the Mancoosi International Solver Competition 2011. `http://data.mancoosi.org/misc2011/problems/`.

49 Mancoosi international solver competition 2011. `https://www.mancoosi.org/misc-2011/index.html`.

50 Rafael Marques, Luís M. S. Russo, and Nuno Roma. Flying tourist problem: Flight time and cost minimization in complex routes. *Expert Syst. Appl.*, 130:172–187, 2019. `doi:10.1016/j.eswa.2019.04.024`.

51 João Marques-Silva and Jordi Planes. On using unsatisfiability for solving maximum satisfiability. *CoRR*, abs/0712.1097, 2007. `arXiv:0712.1097`.

52 João Marques-Silva, Josep Argelich, Ana Graça, and Inês Lynce. Boolean lexicographic optimization: Algorithms & applications. *Ann. Math. Artif. Intell.*, 62:317–343, July 2011. `doi:10.1007/s10472-011-9233-2`.

53 Ruben Martins, Saurabh Joshi, Vasco M. Manquinho, and Inês Lynce. Incremental cardinality constraints for maxsat. In Barry O'Sullivan, editor, *Principles and Practice of Constraint Programming - 20th International Conference, CP*, volume 8656 of *Lecture Notes in Computer Science*, pages 531–548. Springer, 2014. `doi:10.1007/978-3-319-10428-7_39`.

54 Ruben Martins, Vasco M. 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 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 438–445. Springer, 2014. `doi:10.1007/978-3-319-09284-3_33`.

55 MaxSAT Evaluation 2021. `https://maxsat-evaluations.github.io/2021/`.

56 mccs package upgradeability solver webpage. `https://www.i3s.unice.fr/~cpjm/misc/mccs.html`.

57 Claude Michel and Michel Rueher. Handling software upgradeability problems with MILP solvers. In Inês Lynce and Ralf Treinen, editors, *Proceedings First International Workshop on Logics for Component Configuration, LoCoCo 2010, Edinburgh, UK, 10th July 2010*, volume 29 of *EPTCS*, pages 1–10, 2010. `doi:10.4204/EPTCS.29.1`.

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

59 Multi-Objective Boolean Optimisation benchmarks generated from the unweighted track of the MaxSAT Evaluation 2021. `https://sat.inesc-id.pt/~mcabral/benchmarks/mse21_pbmo.zip`.

60 Alexander Nadel. On optimizing a generic function in SAT. In *2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel, September 21-24, 2020*, pages 205–213. IEEE, 2020. `doi:10.34727/2020/isbn.978-3-85448-042-6_28`.

61 Alexander Nadel and Vadim Ryvchin. Bit-Vector Optimization. In Marsha Chechik and Jean-François Raskin, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS*, volume 9636 of *Lecture Notes in Computer Science*, pages 851–867. Springer, 2016. `doi:10.1007/978-3-662-49674-9_53`.

62 Toru Ogawa, YangYang Liu, Ryuzo Hasegawa, Miyuki Koshimura, and Hiroshi Fujita. Modulo Based CNF Encoding of Cardinality Constraints and Its Application to MaxSAT Solvers. In *International Conference on Tools with Artificial Intelligence*, pages 9–17. IEEE, 2013.

63 Open-WBO source code. `https://github.com/sat-group/open-wbo`.

64 packup package upgradeability solver webpage. `http://sat.inesc-id.pt/~mikolas/sw/packup/`.

65 SCIP solver webpage. `https://www.scipopt.org/`.

66 Carsten Sinz. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In *Principles and Practice of Constraint Programming*, volume 3709 of *LNCS*, pages 827–831. Springer, 2005.

67 T. Soh, M. Banbara, N. Tamura, and D. Le Berre. Solving Multiobjective Discrete Optimization Problems with Propositional Minimal Model Generation. In *International Conference on Principles and Practice of Constraint Programming*, pages 596–614. Springer, 2017.

68 Miguel Terra-Neves, Inês Lynce, and Vasco Manquinho. Introducing pareto minimal correction subsets. *Proceedings of the Twentieth International Conference Theory and Applications of Satisfiability Testing*, pages 195–211, 2017. `doi:10.1007/978-3-319-66263-3_13`.

69 Y. Xiang, Y. Zhou, Z. Zheng, and M. Li. Configuring Software Product Lines by Combining Many-Objective Optimization and SAT Solvers. *ACM Transactions on Software Engineering and Methodology*, 26(4):14:1–14:46, 2018.

70 Yuan Yuan and Wolfgang Banzhaf. ARJA: automated repair of java programs via multi-objective genetic programming. *IEEE Trans. Software Eng.*, 46(10):1040–1067, 2020. `doi:10.1109/TSE.2018.2874648`.

71 Q. Zhang and H. Li. MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, 2007.