MaxSAT-Based Bi-Objective Boolean Optimization

Christoph Jabs
HIIT, Department of Computer Science, University of Helsinki, Finland

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

Andreas Niskanen
HIIT, Department of Computer Science, University of Helsinki, Finland

Matti Järvisalo
HIIT, Department of Computer Science, University of Helsinki, Finland

Abstract

We explore a maximum satisfiability (MaxSAT) based approach to bi-objective optimization. Bi-objective optimization refers to the task of finding so-called Pareto-optimal solutions in terms of two objective functions. Bi-objective optimization problems naturally arise in various real-world settings. For example, in the context of learning interpretable representations, such as decision rules, from data, one wishes to balance between two objectives, the classification error and the size of the representation. Our approach is generally applicable to bi-objective optimizations which allow for propositional encodings. The approach makes heavy use of incremental Boolean satisfiability (SAT) solving and draws inspiration from modern MaxSAT solving approaches. In particular, we describe several variants of the approach which arise from different approaches to MaxSAT solving. In addition to computing a single representative solution per each point of the Pareto front, the approach allows for enumerating all Pareto-optimal solutions. We empirically compare the efficiency of the approach to recent competing approaches, showing practical benefits of our approach in the contexts of learning interpretable classification rules and bi-objective set covering.

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

Keywords and phrases Multi-objective optimization, Pareto front enumeration, bi-objective optimization, maximum satisfiability, incremental SAT

Digital Object Identifier 10.4230/LIPIcs.SAT.2022.12

Supplementary Material *Software (Source Code and Data)*: `https://bitbucket.org/coreo-group/bioptsat`; archived at `swh:1:dir:3bb8b3ab49f2c36cfeb99211ccd60ac0f8b9bb15`

Funding Work financially supported by Academy of Finland under grants 322869, 328718 and 342145.

Acknowledgements The authors wish to thank the Finnish Computing Competence Infrastructure (FCCI) for supporting this project with computational and data storage resources.

1 Introduction

Recent years have witnessed significant progress in Boolean satisfiability (SAT) based optimization, maximum satisfiability (MaxSAT) solving in particular [8]. Much like the success of SAT solvers, MaxSAT allows for succinctly encoding a wide range of NP-hard real-world optimization problems, and modern MaxSAT solvers today can scale up to finding provably optimal solutions to instances of very significant size.

As typical for constraint optimization solvers, MaxSAT allows for finding optimal solutions with respect to a single cost function. However, various real-world settings give rise to multiple, often conflicting objectives [17]. In such multi-objective settings, the answer to the question of what constitutes an optimal solution becomes less evident. A standard notion of "optimality" in the multi-objective case is that of Pareto optimality (also called Pareto efficiency in some contexts) [7]. Intuitively, a Pareto-optimal solution is one which cannot be improved wrt any single objective without making it worse wrt another objective.

In this work, we focus in particular on bi-objective optimization, that is, the task of finding the Pareto-optimal solutions – or in other words, computing a representative solution for each point on the so-called the Pareto front – under two conflicting objectives. While the handle on the solutions of interest can quickly become hard to grasp when the number of objectives is increased, bi-objective problems naturally arise in the real-world. One topical setting is that of learning interpretable classifiers [27, 37, 22, 30, 40, 21, 51, 52, 24] such as decision rules (or other logically-oriented representations) from data. In this context, interpretability – often understood as the size of a representation, with the intuition that the smaller the representation, the easier it is for humans to interpret – is intrinsically conflicting with the objective of accurately representing the data at hand; hence the two objectives of minimizing size of the representation and minimizing classification error give naturally rise to combinatorial bi-optimization problems.

In this work, we develop an approach to SAT-based bi-objective optimization. More precisely, the approach we develop, which can be viewed as an instantiation of the lexicographic method [31] via SAT solving, allows for taking advantage of advances in MaxSAT solving algorithms. Instead of using MaxSAT solvers as black-boxes, however, we make use of incremental SAT solving [15, 33] directly in implementing the approach. As the approach allows for making use of a MaxSAT algorithm of choice, we study the effectiveness of different algorithmic choices, both solution-improving (sometimes called SAT/UNSAT) [8, 11, 16] and core-guided [34, 5, 38, 23] variants. The approach allows for computing representatives for each point on the Pareto front in an ordered fashion, and extends naturally also to enumerating *all* solutions at each point of the Pareto front. In terms of earlier work on SAT-based multi-objective optimization, it should be noted that we go beyond the multi-*level* setting [32] of lexicographic optimization which assumes a preference order among the objectives.

What comes to competing approaches, we implement for the exact same setting two recent approaches: enumeration of so-called P-minimal solutions [47] (as arguably the closest one to ours) originally proposed in the context of SAT-based constraint optimization [28], and an implicit hitting set style approach in the flavour of the recently-proposed Seesaw approach [26] (for more discussion on related work, see Section 5). While there are no evident standard benchmark sets in the context of multi-objective optimization, we empirically evaluate the performance of these approaches in two problem settings, learning Pareto-optimal interpretable decision rules (as a generalization of settings for which MaxSAT-based solutions have been proposed [30]) and bi-objective set covering (as earlier considered in the work presenting enumeration of P-minimal solutions [47]). The empirical results suggest that our approach outperforms these competing approaches and that its efficiency is impacted by the choice of the integrated MaxSAT algorithm within the approach.

2 Preliminaries

For a Boolean variable x there are two literals, the positive x and the negative $\neg x$. A clause C is a set of (disjunction over) literals and a CNF formula F is a set of (conjunction over) clauses. The set of variables and literals appearing in F are $\textsc{var}(F)$ and $\textsc{lit}(F)$, respectively. A truth assignment τ maps Boolean variables to 1 (true) or 0 (false). The semantics of

$$F = \{\texttt{As-CNF}\left(\sum_{x \in \mathrm{O_I} \cup \mathrm{O_D}} x \geq 3\right), (i_1 \vee i_2), (i_2 \vee i_3), (d_1 \vee d_2), (d_2 \vee d_3)\}$$

$$\mathrm{O_D} = \{d_1, d_2, d_3\}$$
$$\mathrm{O_I} = \{i_1, i_2, i_3\}$$

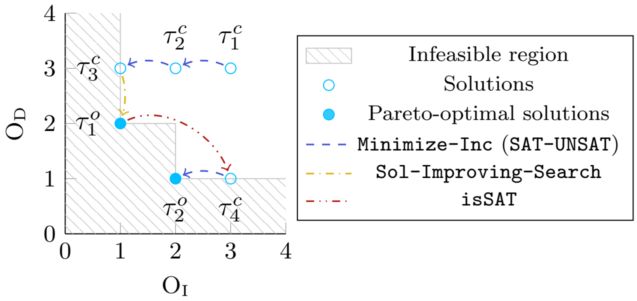

Figure 1 Left: An example CNF formula F and two objectives $\mathrm{O_I}$ and $\mathrm{O_D}$. Right: the solution space of F wrt $\mathrm{O_I}$ and $\mathrm{O_D}$. The solutions τ_1^o and τ_2^o (solid points) are Pareto-optimal, while τ_i^c for $i = 1, \ldots, 4$ are not.

truth assignments are extended to a negated variable $\neg v$, a clause C and a formula F in the standard way: $\tau(\neg v) = 1 - \tau(v)$, $\tau(C) = \max\{\tau(l) \mid l \in C\}$, and $\tau(F) = \min\{\tau(C) \mid C \in F\}$. When convenient, we view assignments τ over a set $\mathrm{VAR}(F)$ of variables as sets of literals $\tau = \{l \mid l \in \mathrm{VAR}(F), \tau(l) = 1\} \cup \{\neg l \mid l \in \mathrm{VAR}(F), \tau(l) = 0\}$. An assignment τ for which $\tau(F) = 1$ is a solution to F. A CNF formula F is satisfiable if it has solutions, otherwise it is unsatisfiable. In this work, wlog we assume that all CNF formulas we deal with are satisfiable. For a set L of literals and a bound $k \in \mathbb{N}$, $\texttt{As-CNF}\left(\sum_{l \in L} l \geq k\right)$ denotes a CNF formula that encodes the linear inequality $\sum_{l \in L} l \geq k$.

An objective O is a multiset of literals. The value $\mathrm{O}(\tau)$ of a truth assignment τ under O is $\mathrm{O}(\tau) = \sum_{l \in \mathrm{O}} \tau(l)$, i.e., the number of the literals in O that τ assigns to 1. Treating O as a multiset allows for representing objective functions with non-unit coefficients by adding a literal multiple times.

Given a CNF formula F, two objectives $\mathrm{O}_1, \mathrm{O}_2 \subset \mathrm{LIT}(F)$ and solutions τ_1, τ_2 to F, we say that τ_1 dominates τ_2 if (i) $\mathrm{O}_i(\tau_1) \leq \mathrm{O}_i(\tau_2)$ for $i = 1, 2$, and (ii) either $\mathrm{O}_1(\tau_1) < \mathrm{O}_1(\tau_2)$ or $\mathrm{O}_2(\tau_1) < \mathrm{O}_2(\tau_2)$. A solution τ is Pareto-optimal if no other solution dominates it. The Pareto front of F wrt $\mathrm{O}_1, \mathrm{O}_2$ consists of all solutions of F that are Pareto-optimal wrt O_1 and O_2. When the objectives are clear from context, we will simply say that a solution τ is a Pareto-optimal solution of F. The pair $(\mathrm{O}_1(\tau), \mathrm{O}_2(\tau))$ of a Pareto-optimal τ is a Pareto point (of F wrt O_1 and O_2). Note that there may be multiple solutions that correspond to the same Pareto point. We consider the task of computing a representative solution for each Pareto point as well as the task of enumerating all solutions in the Pareto front.

▶ **Example 1.** An example CNF formula F and two objectives $\mathrm{O_I}$ and $\mathrm{O_D}$ are shown on the left in Figure 1. The solution space is illustrated on the right. The two solid dots correspond to the two Pareto points of F wrt $\mathrm{O_I}$ and $\mathrm{O_D}$. Examples of Pareto-optimal solutions corresponding to these points are $\tau_1^o = \{d_1, d_3, i_2, \neg d_2, \neg i_1, \neg i_3\}$ and $\tau_2^o = \{i_1, i_3, d_2, \neg i_2, \neg d_1, \neg d_3\}$.

An important property of Pareto-optimal solutions to bi-objective problems is summarized by the next observation.

▶ **Observation 2** (Adapted from [20]). *Sorting the Pareto-optimal solutions of F wrt increasing values of O_1 amounts to sorting them wrt decreasing values of O_2, and vice-versa.*

▶ **Example 3.** Consider the CNF formula F, the objectives $\mathrm{O_I}$ and $\mathrm{O_D}$ and the two Pareto-optimal solutions τ_1^o and τ_2^o from Figure 1 and Example 1. By the definition of Pareto-optimality, lowering the value of one objective of a Pareto-optimal solution has to increase the value of the other; we have $\mathrm{O_I}(\tau_1^o) = 1 < 2 = \mathrm{O_I}(\tau_2^o)$ and $\mathrm{O_D}(\tau_1^o) = 2 > 1 = \mathrm{O_D}(\tau_2^o)$.

Algorithm 1 BiOptSat: MaxSAT-based bi-objective optimization.

Input: CNF formula F, objectives $\mathrm{O_I}$ and $\mathrm{O_D}$.
Output: Either one or all Pareto-optimal solution corresponding to each Pareto point of F.

InitSATsolver(F)
(res, τ) $\leftarrow$ isSAT($\emptyset$) {Invokes the SAT solver on the formula.}
if res = UNSAT **then**
  return "no solutions"
$b_\mathrm{D} \leftarrow \infty, b_\mathrm{I} \leftarrow 0$
while res = SAT **do**
  $(b_\mathrm{I}, \tau) \leftarrow$ Minimize-Inc($b_\mathrm{D}, \mathrm{O_I}(\tau)$) {Maintains $\textsc{Tot}(\mathrm{O_I})$ (or similar)}
  $(b_\mathrm{D}, \tau) \leftarrow$ Solution-Improving-Search($b_\mathrm{I}, \mathrm{O_D}(\tau)$) {Builds $\textsc{Tot}(\mathrm{O_D})$}
  yield τ {Optionally: **yield** EnumSols($b_\mathrm{D}, b_\mathrm{I}$)}
  (res, τ) $\leftarrow$ isSAT($\{\langle \mathrm{O_D} < b_\mathrm{D}\rangle\}$)

Incremental SAT Solving under Assumptions [15, 33]. When the underlying CNF formula F is clear from context, the call isSAT($\mathcal{A}$) invokes a SAT solver on the formula under the assumptions specified by the set $\mathcal{A}$ of literals. The call either returns "satisfiable" (SAT) and a solution $\tau \supset \mathcal{A}$, or "unsatisfiable" (UNSAT) and a subset $\mathcal{A}_s \subset \{\neg l \mid l \in \mathcal{A}\}$ such that $F \wedge \bigwedge_{l\in\mathcal{A}_s}(\neg l)$ is unsatisfiable, i.e., an unsatisfiable core of F.

Totalizers. Given a set L of n input literals and a bound $k = 1, \dots, n$, the (incremental) totalizer encoding [9, 35] produces a CNF formula $\textsc{Tot}(L, k)$ that defines a set $\{\langle L < 1\rangle, \dots, \langle L < k\rangle\} \subset \textsc{var}(\textsc{Tot}(L, k))$ of *output literals* that – informally speaking – count the number of literals in L assigned to true by solutions to $\textsc{Tot}(L, k)$: If τ is an assignment that satisfies $\textsc{Tot}(L, k)$, then $\tau(\langle L < b\rangle) = 1$ if $\sum_{l\in L} \tau(l) < b$. The incremental totalizer supports both increasing the bound k and adding new input literals without having to rebuild the whole formula: we have that $\textsc{Tot}(L, k) \subset \textsc{Tot}(L, k')$ and $\textsc{Tot}(L, k) \subset \textsc{Tot}(L \cup L', k)$ hold for any bound $k' > k$ and set L' of literals. If the bound k is clear from context or $k = |L|$ we will simply write $\textsc{Tot}(L)$. Additionally, we use $\langle L \leq b\rangle$ as a shorthand for the literal $\langle L < b + 1\rangle$. We note that the assignments of the auxiliary variables of the totalizer encoding are functionally defined by the assignment of the input and output variables. As such we will leave them out from the solutions we describe in favour of brevity and clarity of examples.

3 The Approach

We detail the MaxSAT-based approach to bi-objective optimization developed in this work together with its variants.

3.1 Overview of the Algorithm

Algorithm 1, which we refer to as BiOptSat, details our framework for computing the Pareto-optimal solutions of a given CNF formula F wrt two given objectives $\mathrm{O_I}$ and $\mathrm{O_D}$. The framework is an instantiation of the general lexicographic optimization method [31] instantiated with a SAT solver. More specifically, all subroutines of the procedure are implemented using a single instantiation of a SAT solver that is invoked incrementally and preserved (i.e., not reset) during the whole search. BiOptSat maintains the bounds b_I and b_D on the two objectives $\mathrm{O_I}$ and $\mathrm{O_D}$, respectively. In each iteration, the value of b_I is set to the smallest value for which there is a still-undiscovered Pareto-optimal solution

τ for which $\mathrm{O_I}(\tau) = b_\mathrm{I}$ by the `Minimize-Inc` procedure. The value of b_D is then set to $\mathrm{O_D}(\tau)$ by the `Solution-Improving-Search` procedure. In case one wishes to enumerate all Pareto-optimal solutions (in contrast to a single representative solution for each Pareto point), the `EnumSols` procedure then enumerates all Pareto-optimal solutions τ^o for which $\mathrm{O_I}(\tau^o) = b_\mathrm{I}$ and $\mathrm{O_D}(\tau^o) = b_\mathrm{D}$.

Importantly, since the value of $\mathrm{O_I}$ is always minimized first, the value b_I returned each iteration is monotonically increasing. We therefore call $\mathrm{O_I}$ the *increasing objective*. By Observation 2, this means that the sequence of values b_D is monotonically decreasing, leading us to calling $\mathrm{O_D}$ *decreasing*. By these observations, BiOptSat performs search in an ordered fashion along the Pareto front.

In detail, given a CNF formula F and two objectives $\mathrm{O_I}$ and $\mathrm{O_D}$, the search of BiOptSat in Algorithm 1 starts by initializing a SAT solver with all clauses in F on Line 1. Satisfiability (i.e., the existence of any Pareto-optimal solutions) is checked by invoking the SAT solver on its internal formula without assumptions via the `isSAT`($\emptyset$) function (Line 2). If the formula is unsatisfiable, there are no Pareto-optimal solutions and the algorithm terminates. Otherwise, τ is an assignment that satisfies the formula. Before the main enumeration procedure starts, the bounds b_I and b_D on $\mathrm{O_I}$ and $\mathrm{O_D}$ are set to 0 and ∞, respectively.

The main search loop (Lines 6–10) iterates as long as there are Pareto-optimal solutions of F that have not been enumerated yet. This is the case if there is a solution τ for which $\mathrm{O_D}(\tau) < b_\mathrm{D}$, which is checked by invoking the SAT solver under the assumption $\langle \mathrm{O_D} < b_\mathrm{D} \rangle$ on Line 10. In the beginning of each main loop iteration, the procedure `Minimize-Inc` is employed to minimize the increasing objective, i.e., to compute the smallest value b_I for which there is a solution τ for which $\mathrm{O_D}(\tau) < b_\mathrm{D}$ and $\mathrm{O_I}(\tau) = b_\mathrm{I}$ (Line 7). We assume that `Minimize-Inc` maintains a way to enforce that $\mathrm{O_I}(\tau) < k$, e.g., through a totalizer $\mathrm{Tot}(\mathrm{O_I})$, and that BiOptSat and all of its subroutines have access to a set of assumptions to enforce this bound for any k.

Next, the algorithm employs *solution-improving search* [8, 11, 16] to minimize the decreasing objective, i.e., to compute the smallest b_D for which there is a solution τ for which $\mathrm{O_D}(\tau) = b_\mathrm{D}$ and $\mathrm{O_I}(\tau) = b_\mathrm{I}$ (Line 8). The totalizer $\mathrm{Tot}(\mathrm{O_D}, \mathrm{O_D}(\tau))$ is built at the first time this subroutine is invoked. Building the totalizer at this point allows for only building it up to bound $\mathrm{O_D}(\tau)$, since all Pareto-optimal solutions are known to have at most that value for $\mathrm{O_D}$. Solution-improving search works by – starting from $k = \mathrm{O_D}(\tau)$ – iteratively invoking the SAT solver under the assumptions $\{\langle \mathrm{O_D} < k \rangle, \langle \mathrm{O_I} \leq b_\mathrm{I} \rangle\}$ for decreasing values of k until the solver reports UNSAT, and returns b_D and a solution τ for which $\mathrm{O_D}(\tau) = b_\mathrm{D}$ and $\mathrm{O_I}(\tau) = b_\mathrm{I}$. At this point we know that there is no solution of F that dominates τ, so τ is returned as Pareto-optimal on Line 9. If one wants to enumerate all solutions τ^o that correspond to the Pareto point $(b_\mathrm{I}, b_\mathrm{D})$, the `EnumSols` procedure repeatedly invokes the SAT solver with the assumptions $\{\langle \mathrm{O_D} \leq b_\mathrm{D} \rangle, \langle \mathrm{O_I} \leq b_\mathrm{I} \rangle\}$ and blocks each found solution until no more solutions are found.

▶ **Example 4.** Invoke BiOptSat on the CNF formula F and objectives $\mathrm{O_I}$, $\mathrm{O_D}$ detailed in Figure 1. The search starts by invoking a SAT solver on F. The call returns a solution, say $\tau_1^c = \{i_1, i_2, i_3, d_1, d_2, d_3\}$ for which $\mathrm{O_I}(\tau_1^c) = \mathrm{O_D}(\tau_1^c) = 3$. The first iteration of the main search loop starts with a call to `Minimize-Inc`. This returns $b_\mathrm{I} = 1$ and (e.g.) the solution $\tau_3^c = \{i_2, d_1, d_2, d_3, \neg i_1, \neg i_3,\}$ for which $\mathrm{O_I}(\tau_3^c) = 1$ and $\mathrm{O_D}(\tau_3^c) = 3$. BiOptSat then proceeds to the `Solution-Improving-Search` subroutine that initializes a totalizer $\mathrm{Tot}(\mathrm{O_D}, 3)$. The first call to the SAT solver is made under the assumptions $\mathcal{A} = \{\langle \mathrm{O_I} \leq 1 \rangle, \langle \mathrm{O_D} < 3 \rangle\}$. The query is satisfiable. Say that the solver returns the solution $\tau_1^o = \{d_1, d_3, i_2, \neg i_1, \neg i_3, \neg d_2\}$. Then, the solver is invoked with the assumptions $\mathcal{A} = \{\langle \mathrm{O_I} \leq 1 \rangle, \langle \mathrm{O_D} < 2 \rangle\}$. The query is unsatisfiable, so the procedure returns the Pareto-optimal τ_1^o and $b_\mathrm{D} = \mathrm{O_D}(\tau_1^o) = 2$. At the end of the iteration, the SAT solver is queried under the assumption $\{\langle \mathrm{O_D} < 2\rangle\}$. As the query is satisfiable, the solver returns, e.g., the solution $\tau_4^c = \{d_2, i_1, i_2, i_3, \neg d_1, \neg d_2\}$ and the algorithm starts a new iteration.

The next iteration of BiOptSat proceeds similarly to the first. The procedure `Minimize-Inc` returns $b_\mathrm{I} = 2$ and, e.g., the solution $\tau_2^o = \{i_1, i_3, d_2, \neg d_1, \neg d_3, \neg i_2\}$. `Solution-Improving-Search` cannot improve on the decreasing objective, so the solution τ_2^o is proven to be Pareto-optimal. At the end of the iteration, on Line 10 the SAT solver is invoked under the assumption $\{\langle \mathrm{O_D} < 1\rangle\}$. The solver returns unsatisfiable, terminating the algorithm.

3.2 Approaches to Minimizing the Increasing Objective

We consider five different instantiations of the `Minimize-Inc` procedure for minimizing the increasing objective, inspired by MaxSAT algorithms.

SAT-UNSAT. `SAT-UNSAT` is a variant of solution-improving search that is used for minimizing $\mathrm{O_D}$. The procedure gets as input the current bound b_D on $\mathrm{O_D}$ and the value $\mathrm{O_I}(\tau)$ obtained by the solution τ computed during the last SAT solver call. Since the last call is made on Line 10 under the assumption $\langle \mathrm{O_D} < b_\mathrm{D}\rangle$, the solution τ will have $\mathrm{O_D}(\tau) < b_\mathrm{D}$. As such, the value $\mathrm{O_I}(\tau)$ is an upper bound for the smallest value of $\mathrm{O_I}$ obtained by any solution τ' having $\mathrm{O_D}(\tau') < b_\mathrm{D}$.

The procedure `SAT-UNSAT` maintains the totalizer $\mathrm{Tot}(\mathrm{O_I})$ and begins by checking, if the current upper bound on that totalizer is at least $\mathrm{O_I}(\tau)$, extending it if not. Then the SAT solver is iteratively invoked under the assumptions $\{\langle \mathrm{O_D} < b_\mathrm{D}\rangle, \langle \mathrm{O_I} < k\rangle\}$ for decreasing values of k starting from $\mathrm{O_I}(\tau)$. The procedure terminates when the query is unsatisfiable, after which the value of k and the solution obtained during the final satisfiable call are returned as b_I and τ.

▶ **Example 5.** Consider the invocation of BiOptSat detailed in Example 4. We detail the invocation of `Minimize-Inc` instantiated as `SAT-UNSAT`. The full progression of the search of BiOptSat with `Minimize-Inc` instantiated as `SAT-UNSAT` is illustrated in Figure 1. In the first iteration, `SAT-UNSAT` is invoked with $b_\mathrm{D} = \infty$ and $\mathrm{O_I}(\tau) = 3$. At this point, the totalizer over $\mathrm{O_I}$ has not been built, so the procedure starts by adding $\mathrm{Tot}(\mathrm{O_I}, 3)$ to the solver. The first call to the SAT solver is made under the assumptions $\{\langle \mathrm{O_I} < 3\rangle\}$, since $b_\mathrm{D} = \infty$ and therefore no assumption constraining $\mathrm{O_D}$ is needed. The query is satisfiable, the solver returns, e.g., the solution $\tau_2^c = \{d_1, d_2, d_3, i_1, i_2, \neg i_3\}$. In the next iteration, the set of assumptions is $\{\langle \mathrm{O_I} < 2\rangle\}$. The query is again satisfiable, returning, e.g., the solution $\tau_3^c = \{d_1, d_2, d_3, i_2, \neg i_1, \neg i_3\}$. The SAT solver is then invoked under the assumptions $\{\langle \mathrm{O_I} < 1\rangle\}$. Now the query is unsatisfiable, so the procedure terminates and returns τ_3^c and $b_\mathrm{I} = 1$. In the second (and last) iteration of BiOptSat, `SAT-UNSAT` is invoked with $b_\mathrm{D} = 2$ and $\mathrm{O_I}(\tau) = 3$. The first call to the SAT solver is made under the assumptions $\{\langle \mathrm{O_D} < 2\rangle, \langle \mathrm{O_I} < 3\rangle\}$. The query is satisfiable and the solver returns, e.g., the solution $\tau_2^o = \{i_1, i_3, d_2, \neg d_1, \neg d_3, \neg i_2\}$. `SAT-UNSAT` invokes the SAT solver again under the assumptions $\{\langle \mathrm{O_D} < 2\rangle, \langle \mathrm{O_I} < 2\rangle\}$. The query is unsatisfiable, so the procedure returns $b_\mathrm{I} = 2$ and τ_2^o.

UNSAT-SAT. `UNSAT-SAT` takes a similar approach to `SAT-UNSAT` search but searches for the smallest value by lower-bounding instead of upper-bounding. It also maintains a totalizer $\mathrm{Tot}(\mathrm{O_I})$. For finding the next solution, the bound k is set to the last known value of

b_{I} and the solver is then iteratively queried for a new solution under the assumptions $\{\langle \mathrm{O_I} \leq k+1\rangle, \langle \mathrm{O_D} < b_{\mathrm{D}}\rangle\}$. If the query is unsatisfiable, the bound k is increased by 1 and the solver is queried again. The search ends once the solver returns satisfiable and in this case, the solution, and the bound are returned. Since the bound of this lower bounding search procedure will only monotonically increase, it is enough if the totalizer $\textsc{Tot}(\mathrm{O_I})$ is at every step built up to the bound $k+1$ and extended to the next bound in the next iteration. This way, the SAT solver is always loaded with a minimum number of clauses.

MSU3. `MSU3` implements a core-guided approach [34, 5], maintaining a set $\texttt{Act} \subset \mathrm{O_I}$ of *active* objective literals and a totalizer $\textsc{Tot}(\texttt{Act})$ built over them. Initially, $\texttt{Act} = \emptyset$, i.e., all literals of $\mathrm{O_I}$ are inactive. Informally speaking, an inactive literal $l \in \mathrm{O_I} \setminus \texttt{Act}$ is assumed to the value 0 in every invocation of the SAT solver until it is returned as part of a core. More precisely, on input b_{D} and $\mathrm{O_I}(\tau)$, the algorithm starts from the value b_{I} computed in the previous iteration and invokes the SAT solver under the assumptions $\mathcal{A} = \{\langle \texttt{Act} \leq b_{\mathrm{I}}\rangle, \langle \mathrm{O_D} < b_{\mathrm{D}}\rangle\} \cup \{\neg l \mid l \in \mathrm{O_I} \setminus \texttt{Act}\}$. If the query is unsatisfiable, the SAT solver returns a core $\mathcal{A}_s \subset \{\neg l \mid l \in \mathcal{A}\}$. Next, the bound b_{I} is increased by one, the inactive literals in $\mathcal{A}_s$ are added to `Act` and the totalizer $\textsc{Tot}(\texttt{Act})$ is extended. The procedure continues until the SAT solver returns satisfiable, and a solution τ which sets $\mathrm{O_I}(\tau) \leq b_{\mathrm{I}}$ and $\mathrm{O_D}(\tau) < b_{\mathrm{D}}$. At that point the value b_{I} is the minimum value of $\mathrm{O_I}(\tau)$ subject to $\mathrm{O_D}(\tau) < b_{\mathrm{D}}$. This is because the value of b_{I} is increased monotonically, and the solver returned unsatisfiable in the second-to-last iteration.

For enforcing $\langle \mathrm{O_I} \leq b\rangle$ when employing `MSU3`, consider an invocation of $\texttt{MSU3}(b_{\mathrm{D}}, \mathrm{O_I}(\tau))$ made during $\textsc{BiOptSat}$ and assume it returns the tuple (b_{I}, τ). In the next call to `Solution-Improving-Search`, the number of literals in $\mathrm{O_I}$ set to 1 needs to be restricted to at most b_{I}. Since the totalizer maintained by `MSU3` only has $\texttt{Act} \subset \mathrm{O_I}$ as inputs, we do not have access to an output literal of form $\langle \mathrm{O_I} \leq b_{\mathrm{I}}\rangle$. Instead, we use the assumptions $\{\langle \texttt{Act} \leq b_{\mathrm{I}}\rangle\} \cup \{\neg l \mid l \in \mathrm{O_I} \setminus \texttt{Act}\}$, i.e., restrict the number of literals in `Act` set to 1 to b_{I} and assume the value of each inactive literal $l \in \mathrm{O_I} \setminus \texttt{Act}$ to 0. In the following proposition, we prove that doing so does not remove any Pareto-optimal solutions from consideration.

▶ **Proposition 6.** *Let τ be a Pareto-optimal solution of F for which $O_I(\tau) = b_I$. Then $\tau(l) = 0$ for all $l \in O_I \setminus \texttt{Act}$.*

Proof (Sketch). Since, b_{I} was returned by `MSU3`, we know that there is a Pareto-optimal τ^o for which $\mathrm{O_I}(\tau^o) = b_{\mathrm{I}}$ and $\mathrm{O_D}(\tau^o) < b_{\mathrm{D}}$. By the properties of cores, we also know that *any* solution τ^s of F for which $\mathrm{O_D}(\tau^s) < b_{\mathrm{D}}$ assigns at least b_{I} literals in `Act` to 1. Thus, any τ^n that assigns $\tau^n(l) = 1$ for an inactive literal $l \in \mathrm{O_I} \setminus \texttt{Act}$ will have $\mathrm{O_I}(\tau^n) > b_{\mathrm{I}}$. ◀

▶ **Example 7.** Consider the invocation of $\textsc{BiOptSat}$ detailed in Example 4. Here we detail the invocations of `Minimize-Inc` instantiated as `MSU3`. In the first iteration of $\textsc{BiOptSat}$, `MSU3` is invoked with $b_{\mathrm{D}} = \infty$ and $\mathrm{O_I}(\tau) = 3$. Initially, the set $\texttt{Act} = \emptyset$ of active literals is empty, so the first call to the SAT solver is made under the assumptions $\mathcal{A} = \{\neg i_1, \neg i_2, \neg i_3\}$. The query is unsatisfiable and the solver returns, e.g., $\mathcal{A}_s = \{i_1, i_2\}$. The literals in $\mathcal{A}_s$ are marked as active and the totalizer $\textsc{Tot}(\texttt{Act})$ is initialized. The SAT solver is then invoked under the assumptions $\mathcal{A} = \{\neg i_3, \langle \texttt{Act} \leq 1\rangle\}$. The query is satisfiable so the solver returns (e.g.) the solution $\tau_3^c = \{d_1, d_2, d_3, i_2, \neg i_1, \neg i_3\}$ and $b_{\mathrm{I}} = 1$. In the next iteration of $\textsc{BiOptSat}$, `MSU3` is invoked with $b_{\mathrm{D}} = 2$ and $\mathrm{O_I}(\tau) = 2$. The set $\texttt{Act} = \{i_1, i_2\}$ is kept from the previous iterations, so the first call to the SAT solver is made under the assumptions $\mathcal{A} = \{\neg i_3, \langle \texttt{Act} \leq 1\rangle, \langle \mathrm{O_D} < 2\rangle\}$. The query is unsatisfiable. If i_3 is a part of the core $\mathcal{A}_s$ returned by the solver, it is marked as active and the totalizer $\textsc{Tot}(\texttt{Act})$ extended accordingly. Next, the SAT solver is invoked under the assumptions $\mathcal{A} = \{\langle \texttt{Act} \leq 2\rangle, \langle \mathrm{O_D} < 2\rangle\}$. The call returns SAT, obtaining the solution $\tau_2^o = \{i_1, i_3, d_2, \neg d_1, \neg d_3, \neg i_2\}$ and $b_{\mathrm{I}} = 2$.

OLL. `OLL` is another core-guided procedure (originally proposed in the context of ASP [2] and also successfully applied in MaxSAT [38, 23]) that handles the cardinality constraint over the literals in O_I differently to `MSU3`. Instead of a single totalizer over all literals in `Act`, a separate totalizer is built for every core returned after the unsatisfiable SAT solver calls. In each iteration, the assumptions given to the SAT solver consist of (i) the inactive literals of O_I, (ii) the outputs of previously built totalizers corresponding to the lowest number of input literals that should be assigned to 1 in any possible satisfying assignment and (iii) the bound $\langle O_D < b_D \rangle$. The procedure terminates when the SAT solver returns a solution τ. Similarly to `MSU3`, the assumptions for enforcing a bound on O_I in the other subroutines of Algorithm 1 need to be adapted when using `OLL`.

MSHybrid. `MSHybrid` is a hybrid between `MSU3` and `SAT-UNSAT`, with the following intuition. If `MSU3` reaches the stage where all literals of the objective are active, its search will become equivalent to `UNSAT-SAT`. However, `SAT-UNSAT` search may be a significantly better approach compared to `UNSAT-SAT`. If this is the case, `MSU3` might have an advantage over `SAT-UNSAT` as long as not all literals are active, but as soon as all literals are active, it looses its advantage. Furthermore, if a problem instance has literals in O_I that are not constrained by F, these literals will never appear in any core making `MSU3` behave like `UNSAT-SAT` even before the totalizer is fully built.

With this intuition, we propose `MSHybrid`, a – to the best of our understanding – previously unstudied variant that starts with `MSU3` search and switches over to `SAT-UNSAT` as soon as a certain percentage of the literals in O_I have been added to the totalizer $\textsc{Tot}(\texttt{Act})$. Before switching over to `SAT-UNSAT`, the remaining literals are added to the totalizer to build $\textsc{Tot}(O_I)$, which is needed for `SAT-UNSAT`. With this, the advantages of both `MSU3` and `SAT-UNSAT` can in the best case be combined.

▶ **Example 8.** Consider the invocation of BiOptSat detailed in Example 4. We detail the invocations of `Minimize-Inc` instantiated as `MSHybrid`. Since `MSHybrid` starts out as `MSU3`, the first invocation starts by following the description in Example 7. Assume `MSHybrid` is configured to switch as soon as 50% of the literals in O_I are active. This is reached when the core $\mathcal{A}_s = \{i_1, i_2\}$ is returned and i_1, i_2 become active. At this moment, `MSHybrid` stops the `MSU3` search procedure, finishes building $\textsc{Tot}(O_I)$ by adding i_3 to $\textsc{Tot}(\texttt{Act})$, and starts `SAT-UNSAT` search as in the first iteration detailed in Example 5. Since the second iteration is after the switch to `SAT-UNSAT`, it will be identical to the second iteration in Example 5.

3.3 Refinements

We consider a number of refinements to BiOptSat.

Lazily building Tot(O_D). Assume that BiOptSat is invoked on a CNF formula F and a pair of overlapping objectives O_I and O_D for which $O_I \cap O_D \neq \emptyset$ with `Minimize-Inc` instantiated as `MSU3` or `OLL`. Let `Act` be the set of active literals of O_I as maintained by `Minimize-Inc`. Lazy building of $\textsc{Tot}(O_D)$ refers to only having $(O_D \setminus O_I) \cup (\texttt{Act} \cap O_D)$ as input to the totalizer (incrementally extending the totalizer as the set `Act` grows), and assuming the value of each literal $l \in (O_D \cap O_I) \setminus \texttt{Act}$ to 0 in each SAT call made during invocations of `Solution-Improving-Search`. The soundness of doing so follows by an argument very similar to the one we made in Proposition 6.

Lazy building of $\textsc{Tot}(O_D)$ requires a minor adaption to the termination criterion of BiOptSat (i.e., Algorithm 1). As the totalizer maintained by `Solution-Improving-Search` might not have all literals of O_D as inputs, the algorithm does not have a (straight-forward) way of checking if there exists a solution τ for which $\mathrm{O_D}(\tau) < b_\mathrm{D}$. However, the lack of further Pareto-optimal solutions is instead detected in the next call to `Minimize-Inc` by the SAT solver returning a core that only contains the assumption used for bounding the value of $\mathrm{O_D}$.

Domain-Specific Solution Blocking. If multiple representatives of the same Pareto point are of interest, the procedure `EnumSols` needs to block all obtained solutions. While this can be done in a straight-forward manner on the CNF-level, we will in later sections give examples of how domain-specific knowledge can be used in order to derive stronger clauses that block not only a specific solution obtained, but also other, symmetric solutions.

Refinements to Core-Guided Variants. Our implementation of the variants BiOptSat with `MSU3` or `OLL` make use of refinements commonly used in core-guided MaxSAT solving. More specifically, we employ core minimization [23] (either exact or heuristic) and core-exhaustion [23, 4]. Given a core $\mathcal{A}_s$ returned by the SAT solver, heuristic core minimization refers to reinvoking the SAT solver with $\{\neg l \mid l \in \mathcal{A}_s\}$ as the assumptions hoping that the solver returns a smaller set of assumptions. Exact core minimization refers to iteratively finding a minimal unsatisfiable subset by attempting to remove each assumption separately. Core exhaustion is an OLL specific technique that seeks to improve the lower bound of each totalizer being added.

4 Experiments

We implemented all variants and refinements of BiOptSat described in Section 3 in C++. The open-source implementation and empirical data are available at `https://bitbucket.org/coreo-group/bioptsat`. Our implementations of MSU3 and OLL were inspired by their implementations in Open-WBO [36], the other variants were implemented from scratch. We used CaDiCaL v1.5.2 [12] as the internal SAT solver. We also reimplemented the competing approaches P-minimal and Seesaw (see Sect. 4.2), since no reference implementations were available. For ParetoMCS, we used the publicly-available Sat4j-based [11] implementation from `https://gitlab.ow2.org/sat4j/moco`. We evaluate the relative runtime performance of the BiOptSat variants against these two competing approaches, as well as the impact of the specific refinements (recall Section 3.3; employed as applicable, by default with heuristic core minimization) to BiOptSat on their runtime performance. As a parametric detail, in its default `MSHybrid` is configured to switch between `MSU3` and `SAT-UNSAT` once 70% of the literals in $\mathrm{O_I}$ have been added to $\textsc{Tot}(\texttt{Act})$. In preliminary experiments we observed that this threshold is low enough to prevent the `MSU3` search phase from behaving like `UNSAT-SAT`. Furthermore, varying the threshold slightly does not have significant impact on performance. All experiments were run on 2.60-GHz Intel Xeon E5-2670 machines with 64-GB RAM in RHEL under a 1.5-hour per-instance time and 16-GB memory limit.

4.1 Benchmarks

For the experiments, we consider two bi-objective optimization problems as benchmark domains: learning of interpretable decision rules and the bi-objective set covering problem.

Learning Interpretable Decision Rules (LIDR). Recently, a variety of SAT and MaxSAT-based approaches have been developed for learning interpretable classifiers from data [22, 30, 40, 21, 51, 52, 24]. The two objectives of minimizing size ("the smaller, the more interpretable") and classification error (when there is no perfect classifier, as typical for real-world data) are conflicting, hence giving naturally rise to bi-objective optimization problems. Here we consider learning of interpretable decision rules as a representative benchmark domain from this line of work, building on the encoding presented in [30]. In short, here a decision rule is a binary classifier in the form of a CNF formula over Boolean features. In [30] a linear combination of the two objectives, using a parameter $\lambda \geq 0$, was proposed in order to directly apply a MaxSAT solver to find decision rules under a pre-fixed value for λ. While this allows for finding a Pareto-optimal decision rule under a specific value of λ, MaxSAT solving multiple times under different choices of λ does not guarantee finding a representative Pareto-optimal decision rule for each Pareto point [31]. In contrast, here we address directly the problem of computing all Pareto-optimal solutions wrt the two objectives. For a given set of n data samples over m features, the encoding uses two sets of variables: s_l^j for $l = 1, \dots, k$, $j = 1, \dots, m$ and η_i for $i = 1, \dots, n$ for a specific number k of clauses in the decision rules to be learned, with the interpretation that $s_l^j = 1$ iff the jth feature is included in the lth clause of the decision rule, and $\eta_i = 1$ if the ith data sample is misclassified.

We represent the sample with index i with a Boolean class y_i and the Boolean features x_i^j where $j = 1, \dots, m$. With this, the encoding is $\neg\eta_i \rightarrow (y_i \leftrightarrow \bigwedge_{l=1}^{k} \bigvee_{j=1}^{m} (x_i^j \wedge s_l^j))$. We use this encoding, literals s_l^j as $\mathrm{O_I}$ and literals η_i as $\mathrm{O_D}$. This corresponds to finding Pareto-optimal solutions wrt the size of the decision rule as the total number of literals and its classification error. (In preliminary experiments we observed that using the classification error as the increasing objective leads to worse performance.) Since decision rules in CNF contain many symmetric solutions obtained by changing the order of clauses, we add additional clauses to the encoding to break these symmetries by enforcing a lexicographic ordering on the bit-strings representing the clauses; see Appendix A for details.

As the basis of benchmark instances, we used 24 standard UCI [14] and Kaggle datasets used in [30]; see Appendix B for details. We randomly and independently sampled subsets of $n \in \{50, 100, 1000, 5000, 10000\}$ data samples from the datasets, four of each size (when applicable), resulting in a total of 372 datasets. All experiments on these datasets were run with the encoding from [30] configured to learn CNF decision rules consisting of two clauses.

When enumerating multiple solutions corresponding to the same Pareto point, the blocking clauses for BiOptSat (as well as the P-minimal approach compared to in the experiments) can be strengthened to find solutions mapping to distinct rules: blocking over the variables s_l^j is sufficient and blocks multiple symmetric solutions that only differ in the assignment to auxiliary variables. Furthermore, making use of the algorithm-specific fact that BiOptSat is guaranteed to enumerate Pareto-optimal solutions in order of increasing size, for BiOptSat it is sufficient to block a solution over all s_l^j that are assigned to false.

Bi-Objective Set Covering. In the set covering problem, given a collection $\mathcal{S}$ of subsets of a set of elements $\{1, \dots, n\}$, the task is to find a smallest possible subset $\mathcal{C}$ of the elements $\{1, \dots, n\}$ such that $\mathcal{C}$ covers all sets in $\mathcal{S}$, i.e., $\mathcal{C} \cap S \neq \emptyset$, $\forall S \in \mathcal{S}$. In the weighted bi-objective set covering problem we consider here, two integer weights c_1^e and c_2^e are associated with each element e. The two objectives are to minimize $\sum_{e \in \mathcal{C}} c_1^e$ and $\sum_{e \in \mathcal{C}} c_2^e$. When encoding bi-objective set covering in propositional logic, every set $S \in \mathcal{S}$ forms one clause in the encoding, i.e., the clauses are $\{l_e \mid e \in S\}$ with l_e being a literal representing if element e is in $\mathcal{C}$. Furthermore, the integer values for the cost c^e associated with element e can be represented by adding l_e to the objective set c^e times. Note that multi-objective set covering was also used originally in an empirical evaluation of the P-minimal approach [47].

We generated two types of bi-objective set covering problem instances: (i) using a fixed probability p for an element appearing in a set (SetCovering-EP), and (ii) using fixed set cardinality s, with elements in a set chosen uniformly at random without replacement (SetCovering-SC). We generated both types of instances using combinations of the following parameters: number of elements $n \in \{100, 150, 200\}$, number of sets $m \in \{20, 40, 60, 80\}$, element probability $p \in \{0.1, 0.2\}$ and set cardinality $s \in \{5, 10\}$. For each combination, we generated five instances, leading to 120 instances of each type. The integer cost values c for the two objectives were chosen uniformly at random from the range $c \in [1, 100]$. We note that – since both objectives are randomly generated by the same process – the two objectives can be swapped without a noticeable impact on overall runtime performance of solvers when run on many instances.

The blocking clauses used in BiOptSat for enumerating all Pareto-optimal solutions can be strengthened also for set covering. Due to the fact that BiOptSat is guaranteed to enumerate the Pareto-optimal solutions so that one of the objectives will monotonically decrease, it is enough to block in BiOptSat the solution over all l_e that are assigned to true.

4.2 Competing Approaches

We consider in our experiments two competing SAT-based approaches for enumerating Pareto-optimal solutions.

***P*-minimal.** The approach perhaps closest to ours is solving multi-objective constraint optimization problems by enumerating so-called P-minimal solutions [47, 28]. We were unable to obtain an implementation of the approach from the authors. For a fair comparison with BiOptSat, we hence reimplemented the approach similarly as BiOptSat. In more detail, the P-minimal approach corresponds to enumerating the solutions of $F^{\mathrm{W}} = F \wedge \textsc{Tot}(\mathrm{O_I}) \wedge \textsc{Tot}(\mathrm{O_D})$ that are subset-minimal wrt the set of outputs of the totalizers. More precisely, if P is the set of output literals of $\textsc{Tot}(\mathrm{O_I}) \wedge \textsc{Tot}(\mathrm{O_D})$, then the goal is to enumerate solutions τ_m such that no other solution τ has $\{b \mid b \in P \wedge \tau(b) = 0\} \subsetneq \{b \mid b \in P \wedge \tau_m(b) = 0\}$. The procedure for enumerating such solutions (detailed in [28]) works by (i) using a solver to obtain any solution τ of F^{W}, (ii) iteratively minimizing the subset of variables of P set to true by the solution, and, once a minimal solution τ_m has been found, (iii) adding the clause $(\langle \mathrm{O_I} < k_1 \rangle \vee \langle \mathrm{O_D} < k_2 \rangle)$ containing the output variables corresponding to the lowest index set to true by τ_m.

▶ **Example 9.** Consider the CNF formula F and two objectives $\mathrm{O_I}$ and $\mathrm{O_D}$ from Figure 1. P-minimal starts by building two totalizers $\textsc{Tot}(\mathrm{O_I})$ and $\textsc{Tot}(\mathrm{O_D})$ and invoking the SAT solver on $F^{\mathrm{W}} = F \wedge \textsc{Tot}(\mathrm{O_I}) \wedge \textsc{Tot}(\mathrm{O_D})$. The query is satisfiable, assume the first solution obtained is $\tau_1^c = \{i_1, i_2, i_3, d_1, d_2, d_3\}$. In order to subset-minimize τ_1^c, the clause $(\langle \mathrm{O_I} < 3 \rangle \vee \langle \mathrm{O_D} < 3 \rangle)$ is added to the SAT solver, and the solver is invoked again under the assumptions $\{\langle \mathrm{O_I} \leq 3 \rangle, \langle \mathrm{O_D} \leq 3 \rangle\}$. The added clause blocks τ_1^c and all solutions dominated by τ_1^c from the search space. Assume the next solution obtained is $\tau_5^c = \{d_1, d_3, i_1, i_3, \neg d_2, \neg i_2\}$. Again, a clause $(\langle \mathrm{O_I} < 2 \rangle \vee \langle \mathrm{O_D} < 2 \rangle)$ is added, and the SAT solver is queried with assumptions $\{\langle \mathrm{O_I} \leq 2 \rangle, \langle \mathrm{O_D} \leq 2 \rangle\}$. The query is satisfiable. Assume the solution obtained is $\tau_2^o = \{i_1, i_3, d_2, \neg i_2, \neg d_2, \neg d_3\}$. P-minimal then adds the clause $(\langle \mathrm{O_I} < 2 \rangle \vee \langle \mathrm{O_D} < 1 \rangle)$ and invokes the solver again under the assumptions $\{\langle \mathrm{O_I} \leq 2 \rangle, \langle \mathrm{O_D} \leq 1 \rangle\}$. The query is unsatisfiable, which proves that τ_2^o is Pareto-optimal. To find a next Pareto-optimal solution, the solver is queried without any assumptions for a new solution to start the minimization process from.

Note that P-minimal has no guarantee on the order that the solutions are enumerated in. Intuitively, when an intermediate solution τ is found, the following SAT solver call either provides another solution that dominates τ, or proves that τ is Pareto-optimal.

In our implementation we extended P-minimal to the task of enumerating all solutions on the Pareto front. Specifically, we add a new relaxation variable r to the clause added in each iteration for use as an assumption to enumerate all solutions at that Pareto point in a standard way. If the next call provides a solution that dominates the previous one, we harden the clause added in the previous iteration by adding $\neg r$ as a unit clause. Also, once all solutions for a Pareto point are enumerated, the clause is hardened.

Seesaw. Seesaw [26] was recently proposed as a framework for bi-objective optimization as a generalization of the so-called implicit hitting set approach [13, 25, 45, 19, 44]. In contrast to our work, a main ingredient in Seesaw is the idea of treating one of the objectives as a black box. This allows for – but also requires – problem-specific instantiations of the black box; no generic Seesaw implementation applicable generally to bi-objective optimization is available. That said, to enable a comparison with (an instantiation of) Seesaw, we instantiated the approach for the LIDR problem. (For bi-objective set covering, both objectives are monotone over the chosen cover. As such, instantiating Seesaw is not feasible because the refined core extraction method from [26] cannot be used, resulting in Seesaw enumerating all possible solutions of the input formula.)

While the original paper presents Seesaw in general terms, in our context the Seesaw algorithm computes Pareto-optimal solutions of a CNF formula F by maintaining a collection $\mathcal{C}$ of subsets of $\mathrm{O_I}$ that are called *cores*. Informally speaking, every solution τ that improves on $\mathrm{O_D}$ needs to assign at least one literal from each core to 1. The algorithm works iteratively by computing a hitting set $\mathtt{hs} \subset \mathrm{O_I}$, i.e., a subset-minimal set of literals of $\mathrm{O_I}$ that intersects with each core in $\mathcal{C}$, and then a solution τ that sets $\tau(o) = 1$ for each $o \in \mathtt{hs}$ and $\tau(o) = 0$ for each $o \in \mathrm{O_I} \setminus \mathtt{hs}$ and for which $\mathrm{O_D}(\tau)$ is the smallest possible value for all such solutions if one exists. The iteration then extracts a new core that $\mathtt{hs}$ does not intersect with. The Pareto-optimal solutions of F are identified by the size of the hitting set increasing. More precisely, if the hitting set is found to increase from size $|\mathtt{hs}|$ to size $|\mathtt{hs}_2|$ with $|\mathtt{hs}_2| > |\mathtt{hs}|$, the solution τ found with a hitting set of size $|\mathtt{hs}|$ that has the smallest minimal value $\mathrm{O_D}(\tau)$ is Pareto-optimal [26].

We instantiated Seesaw for LIDR by using misclassifications as the objective over which cores are extracted and the integer programming solver CPLEX v20.10 for computing a hitting set $\mathtt{hs}$ over these cores. In the second step, the number of literals in the smallest rule misclassifying the examples in $\mathtt{hs}$ or a subset of it is found. This function is implemented as a solution-improving search in CaDiCaL. This instantiation was chosen because finding the smallest rule misclassifying $\mathtt{hs}$ is an anti-monotone function and the refined version of core extraction presented in [26] can therefore be used, making Seesaw feasible in the first place.

Note that, in contrast to BiOptSat and P-minimal, extending Seesaw as it is presented in [26] to support the enumeration of all Pareto-optimal solutions seems non-trivial. For a non-formal intuition note that, while Seesaw is guaranteed to find at least one solution obtaining the objective values of each Pareto-optimal point, the non-deterministic hitting set computation might steer the algorithm past other solutions that obtain the same values.

ParetoMCS. In [49, 48, 50] an approach for computing Pareto-optimal solutions via so-called Pareto-minimal correction sets (ParetoMCSes) was proposed. Using our notation, the approach works by enumerating all subsets $S \subset (\mathrm{O_I} \cup \mathrm{O_D})$ for which (i) $F \wedge \bigwedge_{l \in (\mathrm{O_I} \cup \mathrm{O_D}) \setminus S} (\neg l)$

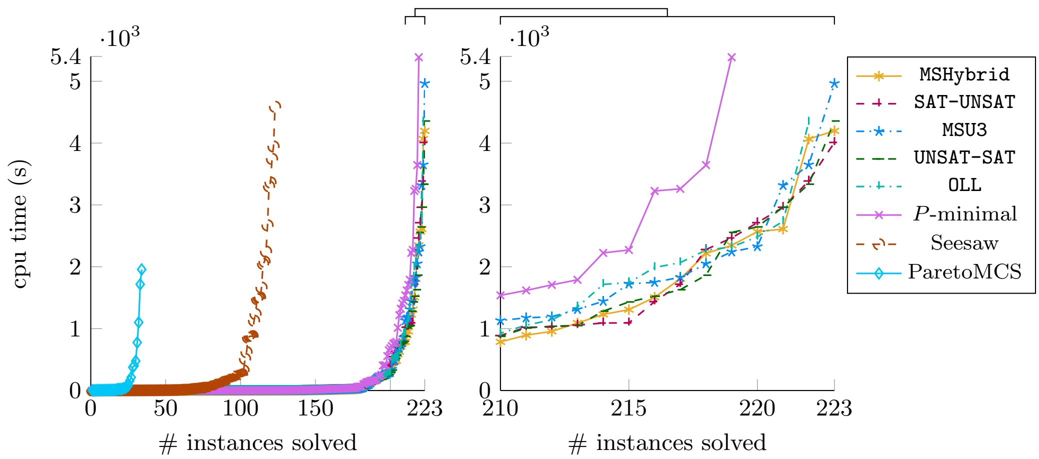

Figure 2 Runtime comparison of variants of BiOptSat and competitors for LIDR; the plot on the right shows a magnification for comparing the best-performing approaches.

is satisfiable and (ii) $F \wedge \bigwedge_{l \in (O_I \cup O_D) \setminus S'} (\neg l)$ is unsatisfiable for all $S' \subsetneq S$. Let $\mathcal{S}$ be the collection of all such sets. The computation of $\mathcal{S}$ corresponds to MCS enumeration to which numerous algorithms have been proposed [10, 39, 42]. The Pareto-optimal solutions are obtained by extracting the solutions satisfying $F \wedge \bigwedge_{l \in (O_I \cup O_D) \setminus S} (\neg l)$ for all $S \in \mathcal{S}$ and removing the dominated ones [49]. The ParetoMCS approach to multi-objective optimization is approximative in that it can only guarantee that a solution is Pareto-optimal once the full set $\mathcal{S}$ has been computed. In contrast, every minimal solution found during the P-minimal approach of [47] and every solution returned by the `EnumSols` subroutine of Algorithm 1 is immediately known to be Pareto-optimal.

▶ **Example 10.** Consider the CNF formula F and two objectives O_I and O_D from Example 1. The ParetoMCS enumeration procedure will return the solution $\tau = \{d_1, d_3, i_1, i_3, \neg d_2, \neg i_2\}$ since no solution τ_s of F has $\{x \in O_I \cup O_D \mid \tau_s(x) = 1\} \subsetneq \{d_1, d_3, i_1, i_3\}$. The fact that the solution τ is not Pareto-optimal can only be discovered when a solution that dominates it is enumerated. However, there are no guarantees on when such a dominating solution is found. This means that τ is guaranteed to be Pareto-optimal only after all solutions in $\mathcal{S}$ have been enumerated.

We refer to this approach of enumerating Pareto-optimal solutions as ParetoMCS for short and only consider an instance solved once all MCSes have been enumerated and the solutions therefore have been proven optimal.

4.3 Results

We start with a comparison of the runtime performance of different variants of BiOptSat, P-minimal and (for LIDR) Seesaw. For LIDR, Figure 2 shows the number of instances solved (x-axis) for different per-instance time limits (y-axis) for the task of computing a single representative solution for each Pareto point. The best-performing approaches are the BiOptSat variants `MSHybrid`, `SAT-UNSAT`, `UNSAT-SAT` and `MSU3` solving 223 instances, while P-minimal solves 219 instances. All variants of BiOptSat outperform P-minimal to some extent. Seesaw and ParetoMCS, solving only 123 and 34 instances, respectively, within the resource constraints, are clearly outperformed by BiOptSat. Figure 3 shows a similar comparison for the two variants of bi-objective set covering. Here `MSHybrid` is

Table 1 Solved instances by approach and benchmark family.

Instance Type	SAT-UNSAT		UNSAT-SAT		MSU3		OLL		MSHybrid		P-minimal	
	single	all	single	all	single	all	single	all	single	all	single	all
Decision Rules	**223**	**215**	**223**	**215**	**223**	**215**	222	213	**223**	**215**	219	213
SetCovering-EP	77	75	71	71	71	70	58	58	**83**	**81**	71	68
SetCovering-SC	35	35	29	29	36	36	34	34	**40**	**40**	38	26

the best-performing variant of BiOptSat, outperforming P-minimal: P-minimal solved 71 (resp. 38) fixed element probability (resp., fixed set cardinality) instances, whereas MSHybrid solved 83 (resp. 40) instances. ParetoMCS did not solve a single one of the set covering instances while Seesaw cannot be feasibly instantiated for this benchmark domain. Similar plots for the task of enumerating all solutions on the Pareto front are provided in Appendix C.

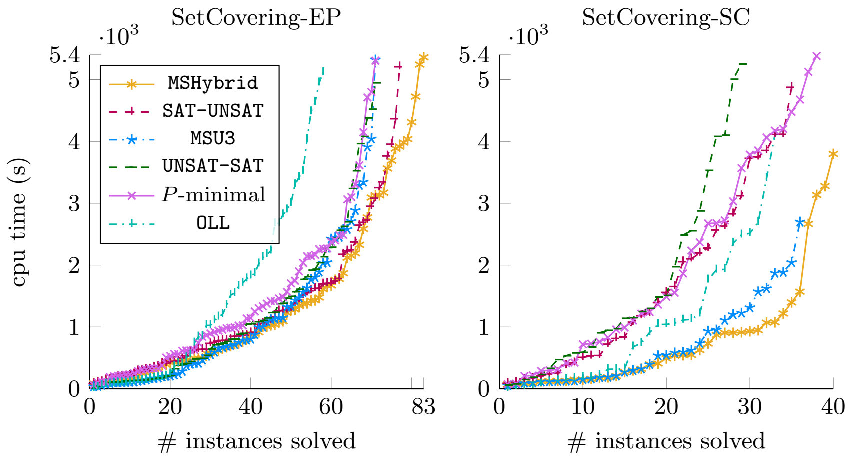

Figure 3 Runtime comparison of variants of BiOptSat and competitors for bi-objective set covering problem.

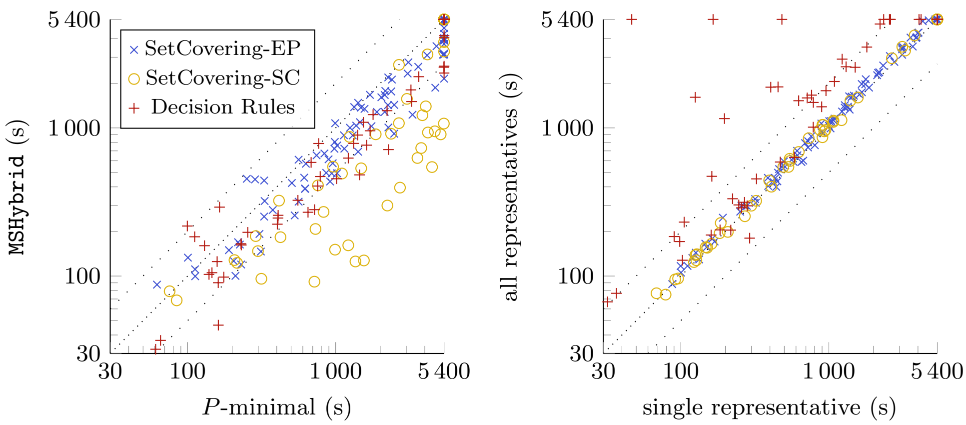

Figure 4 Left: Runtime comparison between P-minimal and BiOptSat in the MSHybrid variant. Right: Runtime comparison between enumerating a single representative vs all solutions per Pareto point with MSHybrid.

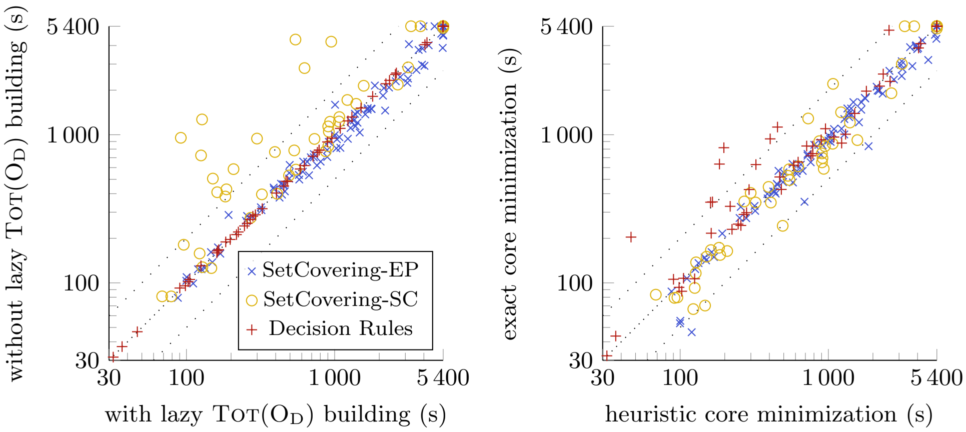

Figure 5 Instance runtime comparisons for the two refinements lazily building the totalizer for the decreasing objective (left) and exact core minimization (right).

The numbers of solved instances for the well-performing approaches are summarized in Table 1, both for enumerating a single representative solution per Pareto point and for enumerating *all* Pareto-optimal solutions. `MSHybrid` is the best-performing BiOptSat variant overall, outperforming P-minimal in all cases. The performance difference is greater when enumerating all Pareto-optimal solutions. For more details, Figure 4 (left) shows a per-instance runtime comparison between `MSHybrid` and P-minimal. We note that P-minimal did not solve a single instance that was not solved by `MSHybrid`. In general, `MSHybrid` was outperformed by P-minimal on only 31 instances while `MSHybrid` solved 297 instances in less time. Both BiOptSat and our implementation of P-minimal make fully incremental use of the SAT solver, never resetting it during search. This suggests that the advantage BiOptSat has over P-minimal lies in the search of BiOptSat being more structured. Figure 4 (right) shows a runtime comparison between enumerating a single representative solution per Pareto point and enumerating all Pareto-optimal solutions with `MSHybrid`. Overall, the approach scales well also for the latter task, although there understandably is an overhead when the number of solutions required to be enumerated grows significantly; this is the case for LIDR where some instances have more than $10,000$ solutions per Pareto point. This is in contrast to the set covering instances, which tend to have only a single (or few) solutions per Pareto point.

Finally, we evaluated the impact of the proposed refinements on the runtime efficiency of the best-performing approach, `MSHybrid`. Figure 5 shows the impact of lazily building $\text{Tot}(O_D)$ (left) and exact vs heuristic core minimization (right). Lazily building $\text{Tot}(O_D)$ has no evident impact on LIDR, as expected (the literals from O_D do not appear in O_I and $\text{Tot}(O_D)$ can therefore not be lazily built). For fixed set cardinality set covering, however, we see a strong positive effect. Heuristic core minimization appears to have a positive effect on LIDR as well as on harder set covering instances, although the difference to exact minimization is smaller than that of lazily building $\text{Tot}(O_D)$.

5 Related Work

We overview other most closely related approaches proposed for multi-objective constraint optimization.

There is earlier work on SAT-based lexicographic optimization [18, 6, 32]. Given a CNF formula F and two objectives O_1 and O_2, a solution τ dominates another solution τ^s in the lexicographic sense if (a) $O_1(\tau) < O_1(\tau^s)$, or (b) $O_1(\tau) = O_1(\tau^s)$ and $O_2(\tau) < O_2(\tau^s)$. Informally speaking, in contrast to Pareto-optimality, lexicographic optimization imposes an explicit preference over the objectives and asks to compute a solution that minimizes O_1 using O_2 as a tie-breaker. The problem is closely related to the so-called multi-level optimization problem. In particular, both can be cast as a single objective weighted optimization problem and solved with a MaxSAT solver [6, 32]. In fact, many modern MaxSAT solvers exploit multilevel properties of input instances in order to improve search efficiency [41, 3].

Beyond SAT-based approaches, multi-objective optimization has been studied in other declarative optimization paradigms. For example, in constraint programming, a filtering algorithm for the bi-objective Pareto constraint was proposed [20]. The resulting search algorithm is similar to ParetoMCS in that it maintains a set $\mathcal{S}$ of solutions that do not dominate each other. When a new solution is found, any solution it dominates is removed from $\mathcal{S}$. Multi-objective optimization has also been studied in the context of mixed integer programming (see, e.g., [43, 29, 46, 1]). Our focus in this work was to develop MaxSAT-based bi-objective optimization problems, especially suited for problems naturally represented in propositional logic, such that the ones we employed in our empirical evaluation.

6 Conclusions

We proposed an approach to bi-objective optimization based on tightly integrating algorithmic ideas from the realm of MaxSAT solving, allowing for instantiations through the integration of different MaxSAT solving algorithms. The approach allows for provably finding all Pareto-optimal solutions. Search in the approach is performed in an ordered way along the Pareto front, which allows for, e.g., employing tighter blocking of earlier found solutions. The approach is generally applicable to bi-objective optimizations which allow for propositional encodings. As examples of such problems, we empirically evaluated several variations and refinements of the approach on two different types of bi-objective optimization problem domains, namely, learning interpretable decision rules from data and bi-objective set covering. Going beyond variants based on well-known MaxSAT solving algorithms, we proposed a hybrid variant of the approach employing both core-guided and solution-improving search. We showed empirically that the hybrid variant achieves the best performance, surpassing also the efficiency of two recent competing approaches.

References

1 Maria João Alves and João C. N. Clímaco. A review of interactive methods for multiobjective integer and mixed-integer programming. *European Journal of Operational Research*, 180(1):99–115, 2007. doi:10.1016/j.ejor.2006.02.033.

2 Benjamin Andres, Benjamin Kaufmann, Oliver Matheis, and Torsten Schaub. Unsatisfiability-based optimization in clasp. In Agostino Dovier and Vítor Santos Costa, editors, *Technical Communications of the 28th International Conference on Logic Programming, ICLP 2012, September 4–8, 2012, Budapest, Hungary*, volume 17 of *LIPIcs*, pages 211–221. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.ICLP.2012.211.

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

4 Carlos Ansótegui, Maria Luisa Bonet, Joel Gabàs, and Jordi Levy. Improving WPM2 for (weighted) partial MaxSAT. In Christian Schulte, editor, *Principles and Practice of Constraint Programming – 19th International Conference, CP 2013, Uppsala, Sweden, September 16–20, 2013. Proceedings*, volume 8124 of *Lecture Notes in Computer Science*, pages 117–132. Springer, 2013. `doi:10.1007/978-3-642-40627-0_12`.

5 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`.

6 Josep Argelich, Inês Lynce, and João P. Marques Silva. On solving boolean multilevel optimization problems. In Craig Boutilier, editor, *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11–17, 2009*, pages 393–398, 2009. URL: `http://ijcai.org/Proceedings/09/Papers/073.pdf`.

7 Jasbir S. Arora. Multiobjective optimum design concepts and methods. In Jasbir S. Arora, editor, *Introduction to Optimum Design (Second Edition)*, pages 543–563. Academic Press, San Diego, second edition edition, 2004. URL: `https://www.sciencedirect.com/science/article/pii/B9780120641550500173`.

8 Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. Maximum satisfiability. In *Handbook of Satisfiability*, volume 336 of *FAIA*, chapter 24, pages 929–991. IOS Press, 2021. `doi:10.3233/FAIA201008`.

9 Olivier Bailleux and Yacine Boufkhad. Efficient CNF encoding of boolean cardinality constraints. In Francesca Rossi, editor, *Principles and Practice of Constraint Programming – CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 – October 3, 2003, Proceedings*, volume 2833 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2003. `doi:10.1007/978-3-540-45193-8_8`.

10 Jaroslav Bendík and Ivana Cerna. Rotation based MSS/MCS enumeration. In Elvira Albert and Laura Kovács, editors, *LPAR 2020: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Alicante, Spain, May 22–27, 2020*, volume 73 of *EPiC Series in Computing*, pages 120–137. EasyChair, 2020. `doi:10.29007/8btb`.

11 Daniel Le Berre and Anne Parrain. The Sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(2–3):59–6, 2010. `doi:10.3233/sat190075`.

12 Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proceedings of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020.

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

14 Dheeru Dua and Casey Graff. UCI machine learning repository, 2021. URL: `http://archive.ics.uci.edu/ml`.

15 Niklas Eén and Niklas Sörensson. Temporal induction by incremental SAT solving. *Electronic Notes in Theoretical Computer Science*, 89(4):543–560, 2003. `doi:10.1016/S1571-0661(05)82542-3`.

16 Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1–4):1–26, 2006. `doi:10.3233/sat190014`.

17 Matthias Ehrgott. *Multicriteria Optimization (2. ed.)*. Springer, 2005. `doi:10.1007/3-540-27659-9`.

18 Matthias Ehrgott and Xavier Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spectrum*, 22(4):425–460, 2000. doi:10.1007/s002910000046.

19 Katalin Fazekas, Fahiem Bacchus, and Armin Biere. Implicit hitting set algorithms for maximum satisfiability modulo theories. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *Automated Reasoning – 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14–17, 2018, Proceedings*, volume 10900 of *Lecture Notes in Computer Science*, pages 134–151. Springer, 2018. doi:10.1007/978-3-319-94205-6_10.

20 Renaud Hartert and Pierre Schaus. A support-based algorithm for the bi-objective pareto constraint. In Carla E. Brodley and Peter Stone, editors, *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27–31, 2014, Québec City, Québec, Canada*, pages 2674–2679. AAAI Press, 2014. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8337.

21 Hao Hu, Mohamed Siala, Emmanuel Hebrard, and Marie-José Huguet. Learning optimal decision trees with MaxSAT and its integration in AdaBoost. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 1170–1176. ijcai.org, 2020. doi:10.24963/ijcai.2020/163.

22 Alexey Ignatiev, João Marques-Silva, Nina Narodytska, and Peter J. Stuckey. Reasoning-based learning of interpretable ML models. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19–27 August 2021*, pages 4458–4465. ijcai.org, 2021. doi:10.24963/ijcai.2021/608.

23 Alexey Ignatiev, António Morgado, and João Marques-Silva. RC2: an efficient MaxSAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 11(1):53–64, 2019. doi:10.3233/SAT190116.

24 Alexey Ignatiev, Filipe Pereira, Nina Narodytska, and João Marques-Silva. A SAT-based approach to learn explainable decision sets. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *Automated Reasoning – 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14–17, 2018, Proceedings*, volume 10900 of *Lecture Notes in Computer Science*, pages 627–645. Springer, 2018. doi:10.1007/978-3-319-94205-6_41.

25 Alexey Ignatiev, Alessandro Previti, Mark H. Liffiton, and João Marques-Silva. Smallest MUS extraction with minimal hitting set dualization. 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 173–182. Springer, 2015. doi:10.1007/978-3-319-23219-5_13.

26 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.

27 Yaochu Jin and Bernhard Sendhoff. Pareto-based multiobjective machine learning: An overview and case studies. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 38(3):397–415, 2008. doi:10.1109/TSMCC.2008.919172.

28 Miyuki Koshimura, Hidetomo Nabeshima, Hiroshi Fujita, and Ryuzo Hasegawa. Minimal model generation with respect to an atom set. In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *Proceedings of the 7th International Workshop on First-Order Theorem Proving, FTP 2009, Oslo, Norway, July 6–7, 2009*, volume 556 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009. URL: http://ceur-ws.org/Vol-556/paper06.pdf.

29 Kuan Lu, Shinji Mizuno, and Jianming Shi. A new mixed integer programming approach for optimization over the efficient set of a multiobjective linear programming problem. *Optimization Letters*, 14(8):2323–2333, 2020. doi:10.1007/s11590-020-01554-7.

30 Dmitry Malioutov and Kuldeep S. Meel. MLIC: A MaxSAT-based framework for learning interpretable classification rules. In John N. Hooker, editor, *Principles and Practice of Constraint Programming – 24th International Conference, CP 2018, Lille, France, August 27–31, 2018, Proceedings*, volume 11008 of *Lecture Notes in Computer Science*, pages 312–327. Springer, 2018. doi:10.1007/978-3-319-98334-9_21.

31 R. Marler and Jasbir Arora. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26:369–395, April 2004. doi:10.1007/s00158-003-0368-6.

32 João Marques-Silva, Josep Argelich, Ana Graça, and Inês Lynce. Boolean lexicographic optimization: algorithms & applications. *Annals of Mathematics and Artificial Intelligence*, 62(3–4):317–343, 2011. doi:10.1007/s10472-011-9233-2.

33 Joao Marques-Silva, Ines Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In *Handbook of Satisfiability*, volume 336 of *FAIA*, chapter 4, pages 133–182. IOS Press, 2021. doi:10.3233/FAIA200987.

34 João Marques-Silva and Jordi Planes. On using unsatisfiability for solving maximum satisfiability. *Computing Research Repository*, abs/0712.1097, 2007. arXiv:0712.1097.

35 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 2014, Lyon, France, September 8–12, 2014. Proceedings*, volume 8656 of *Lecture Notes in Computer Science*, pages 531–548. Springer, 2014. doi:10.1007/978-3-319-10428-7_39.

36 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.

37 Christoph Molnar. *Interpretable Machine Learning*. Lulu.com, 2 edition, 2022. URL: christophm.github.io/interpretable-ml-book/.

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

39 António Morgado, Mark H. Liffiton, and João Marques-Silva. MaxSAT-based MCS enumeration. In Armin Biere, Amir Nahir, and Tanja E. J. Vos, editors, *Hardware and Software: Verification and Testing – 8th International Haifa Verification Conference, HVC 2012, Haifa, Israel, November 6–8, 2012. Revised Selected Papers*, volume 7857 of *Lecture Notes in Computer Science*, pages 86–101. Springer, 2012. doi:10.1007/978-3-642-39611-3_13.

40 Nina Narodytska, Alexey Ignatiev, Filipe Pereira, and João Marques-Silva. Learning optimal decision trees with SAT. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13–19, 2018, Stockholm, Sweden*, pages 1362–1368. ijcai.org, 2018. doi:10.24963/ijcai.2018/189.

41 Tobias Paxian, Pascal Raiola, and Bernd Becker. On preprocessing for weighted MaxSAT. In Fritz Henglein, Sharon Shoham, and Yakir Vizel, editors, *Verification, Model Checking, and Abstract Interpretation – 22nd International Conference, VMCAI 2021, Copenhagen, Denmark, January 17–19, 2021, Proceedings*, volume 12597 of *Lecture Notes in Computer Science*, pages 556–577. Springer, 2021. doi:10.1007/978-3-030-67067-2_25.

42 Alessandro Previti, Carlos Mencía, Matti Järvisalo, and João Marques-Silva. Improving MCS enumeration via caching. In Serge Gaspers and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing – SAT 2017 – 20th International Conference, Melbourne, VIC, Australia, August 28 – September 1, 2017, Proceedings*, volume 10491 of *Lecture Notes in Computer Science*, pages 184–194. Springer, 2017. doi:10.1007/978-3-319-66263-3_12.

43 L.M. Rasmussen. Zero—one programming with multiple criteria. *European Journal of Operational Research*, 26(1):83–95, 1986. URL: https://www.sciencedirect.com/science/article/pii/037722178690161X.

44 Paul Saikko, Carmine Dodaro, Mario Alviano, and Matti Järvisalo. A hybrid approach to optimization in answer set programming. In Michael Thielscher, Francesca Toni, and Frank Wolter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference, KR 2018, Tempe, Arizona, 30 October – 2 November 2018*, pages 32–41. AAAI Press, 2018. URL: https://aaai.org/ocs/index.php/KR/KR18/paper/view/18021.

45 Paul Saikko, Johannes Peter Wallner, and Matti Järvisalo. Implicit hitting set algorithms for reasoning beyond NP. In Chitta Baral, James P. Delgrande, and Frank Wolter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25–29, 2016*, pages 104–113. AAAI Press, 2016. URL: http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12812.

46 Marianna De Santis, Gabriele Eichfelder, Julia Niebling, and Stefan Rocktäschel. Solving multiobjective mixed integer convex optimization problems. *SIAM Journal on Optimization*, 30(4):3122–3145, 2020. doi:10.1137/19M1264709.

47 Takehide Soh, Mutsunori Banbara, Naoyuki Tamura, and Daniel Le Berre. Solving multiobjective discrete optimization problems with propositional minimal model generation. 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 596–614. Springer, 2017. doi:10.1007/978-3-319-66158-2_38.

48 Miguel Terra-Neves, Inês Lynce, and Vasco M. Manquinho. Enhancing constraint-based multi-objective combinatorial optimization. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2–7, 2018*, pages 6649–6656. AAAI Press, 2018. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17227.

49 Miguel Terra-Neves, Inês Lynce, and Vasco M. Manquinho. Multi-objective optimization through pareto minimal correction subsets. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13–19, 2018, Stockholm, Sweden*, pages 5379–5383. ijcai.org, 2018. doi:10.24963/ijcai.2018/757.

50 Miguel Terra-Neves, Inês Lynce, and Vasco M. Manquinho. Stratification for constraint-based multi-objective combinatorial optimization. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13–19, 2018, Stockholm, Sweden*, pages 1376–1382. ijcai.org, 2018. doi:10.24963/ijcai.2018/191.

51 Jinqiang Yu, Alexey Ignatiev, Pierre Le Bodic, and Peter J. Stuckey. Optimal decision lists using SAT. *Computing Research Repository*, abs/2010.09919, 2020. arXiv:2010.09919.

52 Jinqiang Yu, Alexey Ignatiev, Peter J. Stuckey, and Pierre Le Bodic. Computing optimal decision sets with SAT. In Helmut Simonis, editor, *Principles and Practice of Constraint Programming – 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7–11, 2020, Proceedings*, volume 12333 of *Lecture Notes in Computer Science*, pages 952–970. Springer, 2020. doi:10.1007/978-3-030-58475-7_55.

A Symmetry Breaking in the Decision Rule Encoding

To not enumerate multiple decision rules that only differ in the order of their clauses, we added the following symmetry breaking clauses to the encoding from [30]: The idea behind the symmetry breaking is that the bit-strings $\tau(s_l^1)\tau(s_l^2)\ldots\tau(s_l^m)$ are forced to be in lexicographic ordering. In addition to the s variables, we introduce variables e_l^j for $j = 1,\ldots,m$ and $l = 2,\ldots,k$ that represent whether the bit-strings of the clauses with index $(l-1)$ and l are equal for the first j bits. The semantics of this representation are encoded as follows: $e_l^1 \leftrightarrow (s_{l-1}^1 \leftrightarrow s_l^1)$ and $e_l^j \leftrightarrow (e_l^{j-1} \wedge (s_{l-1}^f \leftrightarrow s_l^j))$ for $j = 2,\ldots,m$. The lexicographic ordering is then enforced by adding the constraints $\neg e_l^1 \rightarrow (s_{l-1}^1 \wedge \neg s_{l-1}^1)$ and $(e_l^{j-1} \wedge \neg e_l^j) \rightarrow (s_{l-1}^j \wedge \neg s_l^j)$ for $j = 2,\ldots,m$, enforcing that the bit with the smallest index in which the clauses differ should be 1 in the clause with index $(l-1)$ and 0 in the clause with index l.

B Details About the Datasets Used for Decision Rule Learning

Table 2 summarizes the datasets used in the empirical evaluations, including their origin and statistics, as well as the sizes of CNF formulas obtained from them with the encoding from [30]. The original files were downloaded from the UCI Machine Learning Repository [14] and from Kaggle (`https://www.kaggle.com`). Links to the original datasets as well as the files we used will are available with the implementation. We randomly and independently sampled subsets of $n \in \{50, 100, 1000, 5000, 10000\}$ data samples from the datasets, four of each size (when applicable), resulting in a total of 372 datasets, and discretized the data as in [30]: categorical features are one-hot encoded, continuous features discretized by comparing to a collection of thresholds.

In addition to the name and the source of the datasets, the table shows the number of data samples as well as the number of features before and after discretization. The last two columns give some statistics on the formulas generated with the encoding from [30] for two clauses based on the full datasets. We report both the number of clauses and the number of variables in these formulas.

For the decision rule instances, the instance that took the longest time to solve that did not time out for the `MSHybrid` variant was a subset of 100 examples of the Connect 4 dataset. The CNF formula for this dataset has 678 variables and 4152 clauses. The largest instance in terms of the number of examples that our algorithm was able to find a representative for every Pareto point for was a subset of the Travel Insurance dataset with 10000 samples. When looking at the number of features, the largest solvable dataset was a subset of the Twitter dataset with 50 examples and 1511 discretized features.

C Additional Empirical Detail

Figure 6 shows how many instances could be solved for a specific time limit for the decision rule learning benchmarks. In this case, all approaches (except for Seesaw) enumerate *all* solutions for each Pareto point. Figure 7 shows the same for the set covering benchmarks.

Table 2 The datasets used in the decision rule experiments and summary statistics on them and the CNF formulas generated from them.

Dataset	Source	# samples	# features	# disc. feat.	# clauses (10^3)	# vars (10^3)
Adult	UCI	32 561	14	144	635	98.1
Bank Marketing	UCI	45 211	16	88	1329	136
Banknote Authentication	UCI	372	4	16	6.67	4.16
Connect 4	UCI	67 557	42	126	2052	203
Default of Credit Card Clients	UCI	30 000	23	110	878	90.3
Dota 2 Games Results	UCI	92 650	115	345	11 164	279
FIFA 2018 Man of the Match	Kaggle	128	26	106	3.00	0.708
Heart Disease	Kaggle	303	13	31	3.72	1.00
Indian Liver Patient Dataset	UCI	583	10	14	6.67	1.79
Ionosphere	UCI	351	33	144	9.90	1.49
Iris	UCI	150	4	11	1.08	0.483
MAGIC Gamma Telescope	UCI	19 020	10	79	273	57.3
Medical Hospital Readmissions	Kaggle	25 000	64	125	1641	75.4
Mushroom	UCI	8124	22	115	190	24.7
Parkinsons	UCI	195	22	51	2.81	0.738
Pima Indians Diabetes	Kaggle	768	8	30	7.25	2.39
Skin Segmentation	UCI	245 057	3	119	745	736
Tic-Tac-Toe Endgame	UCI	958	9	27	7.75	2.96
Buzz in Social Media (Toms Hardware)	UCI	28 179	96	910	3712	87.3
Buzz in Social Media (Twitter)	UCI	49 999	77	1511	5406	155
Blood Transfusion Service Center	UCI	748	4	6	4.39	2.26
Travel Insurance	Kaggle	63 326	10	211	1188	191
Wisconsin Diagnostic Breast Cancer	UCI	569	30	88	20.7	1.97
Rain in Australia	Kaggle	107 696	16	141	2952	339

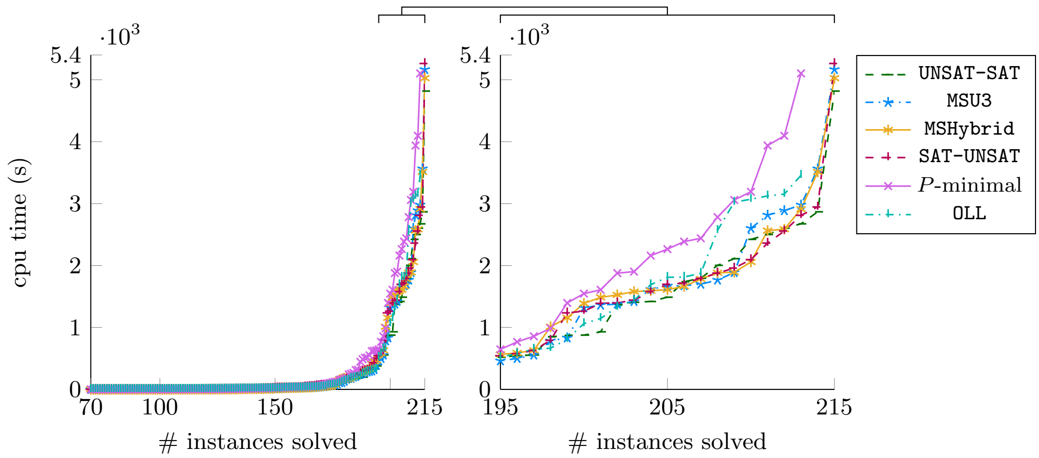

Figure 6 Runtime comparison of P-minimal and variants of BiOptSat for LIDR on the task of enumerating all solutions on the Pareto front.

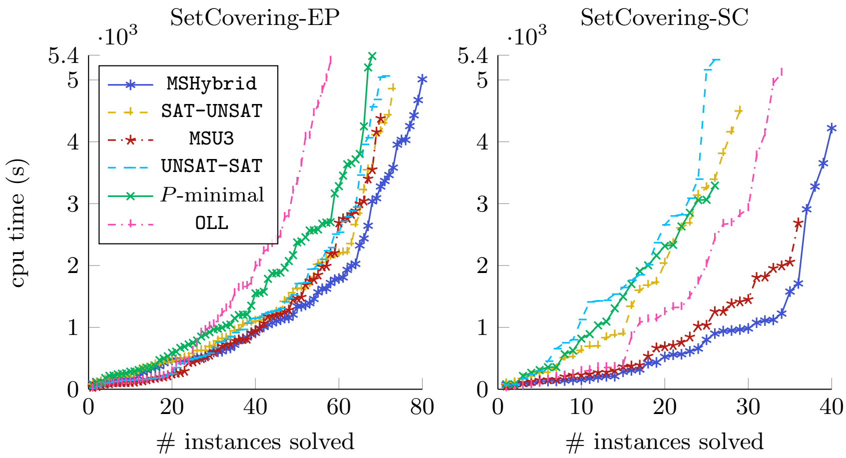

Figure 7 Runtime comparison of P-minimal and variants of BiOptSat for bi-objective set covering problem on the task of enumerating all solutions on the Pareto front.