Master's thesis

Master's Programme in Computer Science

A Maximum Satisfiability Based Approach to Bi-Objective Boolean Optimization

Christoph Jabs

30th May 2022

Faculty of Science

University of Helsinki

Contact information

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki,Finland

Email address: info@cs.helsinki.fi
URL: http://www.cs.helsinki.fi/

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Study programme
Faculty of Science	Master's Programme in Computer Science

Tekijä — Författare — Author

Christoph Jabs

Työn nimi — Arbetets titel — Title

A Maximum Satisfiability Based Approach to Bi-Objective Boolean Optimization

Ohjaajat — Handledare — Supervisors

Prof. Matti Järvisalo, Dr. Jeremias Berg, Dr. Andreas Niskanen

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Master's thesis	30th May 2022	68 pages, 2 appendix pages

Tiivistelmä — Referat — Abstract

Many real-world problem settings give rise to $\mathcal{NP}$-hard combinatorial optimization problems. This results in a need for efficient algorithmic approaches for finding optimal solutions to such problems. Because of $\mathcal{NP}$ hardness, developing such algorithms is highly non-trivial. Many approaches—ranging from local-search-style algorithms to declarative programming—have been developed for optimization problems under a single objective. However, less work has been done on approaches for optimization problems under *multiple* objectives.

We present BiOptSat, an exact declarative approach for finding so-called Pareto-optimal solutions to bi-objective optimization problems. Bi-objective optimization problems arise for example when learning interpretable classifiers, with size, and classification error of classifiers as the two objectives. Using propositional logic as a declarative programming language, we seek to extend the progress and success in maximum satisfiability (MaxSAT) solving to two objectives. BiOptSat can be viewed as an instantiation of the lexicographic method and makes use of a single SAT solver that is preserved throughout the entire search procedure. Our approach allows for finding a single Pareto-optimal solution, finding one representative solution for each Pareto point, and enumerating all Pareto-optimal solutions.

We provide an open-source implementation of five variants of BiOptSat, building on different algorithms proposed for MaxSAT. We empirically evaluate these five variants, comparing their runtime performance to that of three key competing algorithmic approaches. The empirical comparison in the contexts of learning interpretable decision rules and bi-objective set covering shows practical benefits of our approach. Furthermore, for the best-performing variant of BiOptSat, we empirically study the effects of proposed refinements to BiOptSat.

ACM Computing Classification System (CCS)
Mathematics of computing → Discrete mathematics → Combinatorics → Combinatorial optimization
Theory of computation → Logic → Constraint and logic programming

Avainsanat — Nyckelord — Keywords

Bi-objective optimization, MaxSAT, incremental SAT, interpretable classifiers, bi-objective set covering

Säilytyspaikka — Förvaringsställe — Where deposited

Helsinki University Library

Muita tietoja — övriga uppgifter — Additional information

Algorithms study track

Acknowledgements

This work was done as a part of the Constraint Reasoning and Optimization (CoReO) research group at the Department of Computer Science of the University of Helsinki. I want to sincerely thank my supervisors Professor Matti Järvisalo, Doctor Jeremias Berg and Doctor Andreas Niskanen for guiding me in executing the research work and summarizing it in the form of this thesis. Thank you for giving me the opportunity and the freedom to dive deep into the topic and bring in my own ideas.

The research presented in this thesis was funded by the Academy of Finland under grants 322869 and 328718. In addition, I wish to thank the Finnish Computing Competence Infrastructure (FCCI) for supporting this project with computational and data storage resources.

Helsinki, May 2022

Christoph Jabs

Contents

1 Introduction

In this thesis, we develop an exact declarative programming based algorithmic approach to finding so-called Pareto-optimal solutions to bi-objective optimization problems encoded in propositional logic.

Optimization problems can be summarized as the task of finding a "best" solution out of a collection of feasible ones. For example, when looking for a new flat to buy, most people will be comparing prices with the aim to find the cheapest flat possible that fulfils their requirements. Commonly, the notion of "best" that is used in optimization is that a solution with lowest associated "cost" is considered optimal. For the example above, cost is the price of a flat. If the collection of possible solutions is discrete (as in this example), we speak of *combinatorial* optimization.

For many real-world problems, the set of feasible solutions is too large to represent explicitly. Instead, the feasible solutions are implicitly represented, often declaratively as a set of mathematical constraints. Solving such an implicitly defined optimization problem is typically $\mathcal{NP}$-hard [1] and requires non-trivial algorithmic approaches. Examples of such optimization problems appear in scheduling [2–5], supply chain optimization [6], air traffic management [7, 8], clustering [9, 10], and optimal data representation [11–17], among various others.

Different approaches to $\mathcal{NP}$-hard optimization have been proposed. These approaches can be categorized as either exact or inexact. Inexact approaches provide no guarantee of finding an optimal solution but provide a "*good*" solution within given resource constraints. Examples of such approaches are stochastic local search [18] and evolutionary algorithms [19, 20]. Exact approaches, on the other hand, are guaranteed to find an *optimal* solution, given enough resources. A central method for exact optimization is the so-called declarative approach.

The declarative approach to solving optimization problems, as illustrated in Figure 1.1, consists of first employing an encoding that casts the original problem into a set of mathematical constraints formulated in a declarative language. An encoding is hereby a mapping of each instance of the original problem to a set of constraints in the declarative language, where each optimal solution of the encoded instance corresponds to an optimal solution of the original instance. Having encoded the problem instance, a so-called solver,

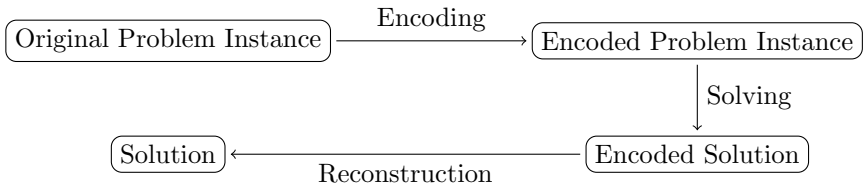

Figure 1.1: The solving pipeline of the declarative approach to optimization.

an algorithm for finding optimal solutions for instances formulated in the encoding language, is invoked on the set of constraints to find a solution with the lowest cost. As a last step, this found solution to the constraints is mapped back to an optimal solution of the original instance.

An advantage of the declarative approach is that it is generally applicable to any problem, as long as a compact encoding for said problem and a practical solver for the chosen declarative language exist. When solving an optimization problem with the declarative approach, the challenge lies in choosing the encoding language that allows for a natural encoding, and finding such an encoding. Determining if a solution to constraints formulated in an encoding language exists is typically $\mathcal{NP}$-complete. For a given declarative language, the existence of solvers that can *efficiently* solve relevant optimization instances of said language is therefore critical as well. Branch-and-cut algorithms for mixed integer linear programming (MILP) [21, 22] are an example of solver technology that achieves good performance on real-world instances. The corresponding language of linear inequalities is arguably the most classical language for modelling hard optimization problems. On the other hand, it might come as a surprise that the encoding language of propositional logic—which is arguably a low-level language—has seen a stark increase in usage as well. This is in particular due to recent advances in propositional satisfiability (SAT) [23] and conflict-driven clause learning solvers [24], the success of which has translated to increasing success of the Boolean optimization paradigm of maximum satisfiability (MaxSAT) [25].

It should be noticed that a clear majority of declarative optimization approaches, including MILP and MaxSAT, work under the assumption that we are dealing with problems which intrinsically have a single objective to optimize. However, this is not always the case. Coming back to the flat search example, we notice that some requirements, like the number of rooms, might be easy to specify, but consider the distance of ones daily commute. Rather than setting a fixed threshold such as "maximum d kilometres distance", what we might actually want to do is minimize this distance at the same time as the cost of the flat. Now

there are two objectives to take into account regarding what constitutes a "best" solution. Two or more objectives give rise to *multi-objective* optimization.

A crucial difference between single- and multi-objective optimization is that there is no single prevalent notion of optimality for two or more objectives. Whereas for a single objective function, there is a clear minimum (or maximum) and objective values can be unambiguously compared, this becomes harder to define naturally for the bi-objective case: in the flat search example, consider a flat A with a cost of 300 000 € and 1-kilometre daily commute and compare it to another flat B that costs 240 000 € and has a 3-kilometre daily commute. It is not immediately clear which one of these options is better, and the choice would depend on ones personal preference over the two objectives. This becomes especially difficult if there is no such preference. Typically, a situation like that occurs when two of the objectives considered are in conflict, as the price of a flat and the corresponding daily commute might be if the commute is towards the city centre and flats in the city centre are more expensive.

In this thesis, we use the commonly-studied concept of *Pareto optimality** [26] as the notion of optimality for multi-objective optimization problems. Intuitively, under Pareto optimality a solution is considered optimal if no solution that improves some objective without worsening the others exists. As an example, this definition considers the two flats A and B from earlier both equally optimal. Under Pareto optimality, the task of solving a bi-objective optimization problem exactly can mean multiple things: finding a single Pareto-optimal solution, finding a representative solution for each Pareto point (i.e., tuple of Pareto-optimal objective values)†, or finding all Pareto-optimal solutions. Many approaches [28–30] to solving multi-objective optimization under Pareto optimality appear to focus on the second task where a single solution per Pareto point is computed. The last task goes one step further and enumerates the full Pareto front (i.e., all Pareto-optimal solutions), even if multiple of the solutions might lead to the same objective values. All three of these tasks can be solved by the algorithmic approach presented in this thesis.

We focus on the common class of *bi*-objective optimization problems, which have exactly two objective functions. Bi-objective optimization problems arise naturally for various real-world settings. For example, when learning interpretable classifiers [11–14, 17, 31–33], the objectives "interpretability" and "classification error" are in conflict because a more complex and therefore less interpretable classifier is typically more accurate. As another

*Pareto optimality is sometimes called *efficiency* [26, 27]

†A Pareto point is also called a *non-dominated point* in literature [26]

example, a bi-objective optimization problem arises when wanting to create a portfolio of algorithms that together solve a set of benchmark instances as fast as possible while also containing as few solvers as possible [29]. There are also bi-objective optimization problems in network routing with the objectives load balancing and latency [34]. In supply chain optimization, in addition to the economic objective, environmental aspects may be of interest to take into consideration as a second objective [35, 36].

The main contribution of this work is the BiOptSat algorithm, a MaxSAT-based bi-objective optimization approach. BiOptSat follows the lexicographic method [37], which works by sequentially minimizing both objectives separately under the additional constraint that the other objective cannot get worse. It terminates once no more solutions can be found, at which point all Pareto-optimal solutions have been enumerated. Note the difference between this lexicographic *method* compared to lexicographic *optimization* to which SAT-based approaches have been proposed earlier [38, 39]. Lexicographic optimization only considers the first Pareto point, found by the lexicographic method, optimal.

BiOptSat builds on advances in MaxSAT solving, allowing for variants based on different so-called solution-improving [25, 40, 41] and core-guided [42–45] algorithms. Our algorithm allows for solving all three tasks for bi-objective optimization: finding a single Pareto-optimal solution, one representative solution for each Pareto point or enumerating all Pareto-optimal solutions. We propose five different variants of BiOptSat that differ in how the minimization of the "increasing" objective is handled. The first four building on the SAT-UNSAT [40], UNSAT-SAT [46], MSU3 [42] and OLL [44] MaxSAT algorithms, modifying them mainly in the fact that a bound on the decreasing objective needs be enforced during optimization. The fifth variant is a hybrid, switching from MSU3 to the SAT-UNSAT-based variant during the search, aiming to combine the advantages of the two approaches. In addition to the five variants of BiOptSat, we also propose multiple refinements for improving its performance: lazily building the cardinality constraints for both objectives to reduce the number of clauses in the solver, blocking dominated solutions to prune the search space, more efficient domain-specific blocking clauses, bound hardening to enable the solver to learn more information and other refinements known from core-guided MaxSAT solving.

We provide an open-source implementation of all five variants of BiOptSat (https://bitbucket.org/coreo-group/bioptsat/) and empirically evaluate its performance on two benchmark domains: learning interpretable decision rules from binary data [11] (as a generalization of settings for which MaxSAT-based single-objective solutions have

been previously proposed) and bi-objective set covering. In the empirical evaluation, we compare BiOptSat to three key SAT-based competitors: enumeration of so-called P-minimal solutions [28], ParetoMCS enumeration [30] and Seesaw [29]. We find that BiOptSat outperformed these competitors in all studied cases. As an additional result of this evaluation, we determine which variant of BiOptSat is the best-performing overall. Furthermore, for the best-performing variant, we study the effects of the proposed refinements to determine their effectiveness.

This thesis is structured as follows. An overview of propositional satisfiability and maximum satisfiability, highlighting the important preliminaries needed to understand the proposed algorithm, is given in Chapter 2. In Chapter 3, we introduce bi-objective optimization, defining the problem and surveying key existing algorithmic approaches to bi-objective optimization. After that, in Chapter 4, we describe BiOptSat, including five distinct variants and some refinements. Details on the empirical evaluation of our implementation are provided in Chapter 5.

Results presented in this thesis have been published in the proceedings of the 25th International Conference on Theory and Applications of Satisfiability Testing (SAT 2022) [47]. In this thesis, we extend on the empirical evaluation and give a more in-depth description of the preliminaries and the algorithm itself, compared to [47].

2 Propositional Satisfiability

In this chapter, we provide an overview of propositional satisfiability (SAT) and maximum satisfiability (MaxSAT). We discuss techniques that our algorithmic approach to bi-objective optimization builds on: incremental SAT solving and cardinality constraints.

2.1 Propositional Satisfiability

For a Boolean variable v there are two literals, the positive v and the negative $\neg v$. A clause C is a set of (i.e., disjunction over) literals, and a CNF formula F is a set of (i.e., conjunction over) clauses [48]. Any propositional formula can be converted in linear time to an equivalent CNF formula of linear size with the standard Tseitin encoding [48, 49]. 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 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 an assignment τ over a set $\textsc{var}(F)$ of variables as the set of literals $\tau = \{v \mid v \in \textsc{var}(F), \tau(v) = 1\} \cup \{\neg v \mid v \in \textsc{var}(F), \tau(v) = 0\}$. An assignment τ for which $\tau(F) = 1$ is a solution to F. The propositional satisfiability (SAT) problem asks to decide whether a given CNF formula F has a solution. A CNF formula F is satisfiable if it has a solution, otherwise it is unsatisfiable.

> **Example 2.1.** Consider the CNF formula $F_1 = a \wedge \neg b$ over variables $\textsc{var}(F_1) = \{a, b\}$. This formula is satisfiable since for $\tau = \{a, \neg b\}$, $\tau(F_1) = 1$. The formula $F_2 = F_1 \wedge (\neg a \vee b)$ on the other hand is not satisfiable. This is because the third clause is the negation of F_1.

The SAT problem was proved to be $\mathcal{NP}$-complete by Cook [50]. This result is central to the modern day use of SAT in the declarative programming approach to solving instances of other $\mathcal{NP}$-complete problems by encoding them as CNF formulas, solving these formulas with a SAT solver and then decoding the solutions to the original problem domain (recall Chapter 1). The advantage of using SAT as a declarative programming language for solving other problems comes from the fact that state-of-the-art SAT solvers are efficient in practice

and can solve real-world instances with up to millions of variables and clauses [24]. For this reason, the SAT-based declarative approach is successful, even though CNF formulas only support a very limited set of constraints (compared to many other declarative languages) natively.

Example 2.2. We give an example of modelling the well-known set covering decision problem [51] in SAT. The set covering problem is to decide whether a *cover* $\mathcal{C}$ of size $|\mathcal{C}| \leq b$ exists that intersects with (i.e., contains an element from) every *set* $S \in \mathcal{S}$. Encoding this problem as a CNF formula can be done by introducing a variable v_e for every distinct element e in the sets. If an assignment τ has $\tau(v_e) = 1$, this encodes that the corresponding cover $\mathcal{C}_\tau$ contains e. Every set is represented as a clause of the corresponding variables. Lastly, assume that $\texttt{As-CNF}\left(\sum_{v \in V} v \leq b\right)$, where V is the set of all variables, encodes that a solution should assign at most b variables to 1. A set of clauses like this is known as a cardinality constraint and possible ways of representing it as a CNF formula will be discussed in Section 2.3. The clauses representing the sets together with the cardinality constraint encode the set covering problem.

As a toy instantiation, assume the following situation. You have three guests coming over and want to make a pizza on which each guest likes at least one topping. Guest A tells you they like pepperoni, courgette, and bell pepper; Guest B likes chicken, avocado, and prawns; Guest C likes mushrooms and chilli pepper. However, you only have enough money to get two toppings from the store. The groups of toppings the guests like form three sets, and you want to know if a cover (another collection of toppings) of at most size two exists. For this example, the encoding described above yields the following three clauses: $C_\text{A} = (v_\text{pepperoni} \vee v_\text{courgette} \vee v_\text{bell pepper})$, $C_\text{B} = (v_\text{chicken} \vee v_\text{avocado} \vee v_\text{prawns})$, $C_\text{C} = (v_\text{mushroom} \vee v_\text{chilli pepper})$. The constraint that the cover should have at most cardinality 2 is encoded as the cardinality constraint $\texttt{As-CNF}\left(\sum_{v \in V} v \leq 2\right)$.

We can now construct the CNF formula $F = C_\text{A} \wedge C_\text{B} \wedge C_\text{C} \wedge \texttt{As-CNF}\left(\sum_{v \in V} v \leq 2\right)$. There is no solution of F, describing that there are no two toppings so that every guest likes at least one of them. Now if Guest C says that they also like prawns, C_C changes to $C_\text{C} = (v_\text{mushrooms} \vee v_\text{chilli pepper} \vee v_\text{prawns})$ and the modified formula is satisfiable. One solution τ of F is $\tau(v_\text{bell pepper}) = \tau(v_\text{prawns}) = 1$ and $\tau(v) = 0$ for all other v. This tells us that on a pizza with bell pepper and prawns, every guest will find something they like.

2.2 Incremental SAT Solving under Assumptions

Many applications of SAT solving—such as algorithms for solving maximum satisfiability [25]—require solving a series of interrelated SAT instances. We discuss state-of-the-art SAT solvers and the incremental interface they provide.

A SAT solver [24] is an implementation of an algorithm that determines the satisfiability of a given CNF formula F. If F is satisfiable, the solver returns "satisfiable" (`SAT`) and a solution τ with $\tau(F) = 1$; if F is unsatisfiable, the return value is "unsatisfiable" (`UNSAT`).

So-called conflict-driven clause learning (CDCL) solvers represent the state of the art in SAT solving. They have been found to solve many $\mathcal{NP}$-hard real-world problem instances significantly more efficiently than the potential exponential worst-case runtime. CDCL solvers work on partial assignments to the variables in a given CNF formula F. During the search for a satisfiable assignment, the partial assignment is extended through so-called decisions and unit propagation. In a decision, the algorithm uses heuristics to select a value for a variable that has no value in the current partial assignment. The deterministic process of unit propagation finds variable assignments that are implied by the partial assignment, i.e., must hold, as otherwise a clause would be falsified by the partial assignment. During the search procedure, a partial assignment might lead to conflicts, meaning that F becomes unsatisfiable. At this stage, CDCL analyses the conflict and learns a clause that ensures that the current partial assignment will not be considered again. After conflict analysis, the solver backtracks non-chronologically, removing some number of the most-recent decisions and the resulting unit propagations. CDCL terminates once a solution is found or the empty clause is learned. Since all learned clauses are logically entailed by F, learning the empty clause proves that F is unsatisfiable.

To solve interrelated instances more efficiently, modern SAT solvers provide an incremental interface that allows for retaining the state of the solver (e.g., learned clauses) from previous solver calls [24, 52]. Retaining learned information this way allows CDCL solvers to determine satisfiability for subsequent calls faster in many cases. Key to incremental SAT solving are *assumptions*. An assumption is a literal that is treated as a temporary unit clause, i.e., a solver call with internal formula F and a set of assumptions $\mathcal{A}$ either returns `SAT` and a solution $\tau \supset \mathcal{A}$, or `UNSAT` and a subset $\kappa \subset \{\neg l \mid l \in \mathcal{A}\}$ such that $F \wedge \bigwedge_{l \in \kappa}(\neg l)$ is unsatisfiable. The subset κ is called an unsatisfiable *core* [24] and implied by F, meaning the assumptions it stems from cannot all be satisfied together with F.

Example 2.3. Recall the toy instance from Example 2.2 after the change of guest C liking prawns. Assume that there is no bell pepper at the store. To check if we can still make a pizza with only two toppings such that every guest likes at least one of the toppings, we can use incremental SAT solving. Instead of adding the fact that bell pepper is not available as a new clause ($\neg v_{\text{bell pepper}}$), we can define assumptions $\mathcal{A} = \{\neg v_{\text{bell pepper}}\}$. Solving with these assumptions, the solver might return the solution τ with $\tau(v_{\text{courgette}}) = \tau(v_{\text{prawns}}) = 1$ and $\tau(v) = 0$ for all other v.

If instead prawns are not available, we can set the set of assumptions to $\mathcal{A} = \{\neg v_{\text{prawns}}\}$. In this case, the solver will return `UNSAT` and the core $\kappa = \{v_{\text{prawns}}\}$. The interpretation of this core is that prawns need to be available so that we can make a pizza that satisfies all constraints.

2.3 Encoding Cardinality Constraints as Totalizers

Many problem applications in the real world require so-called cardinality constraints, enforcing a bound on how many literals in a given set can be assigned to true. In addition, algorithms (for example for MaxSAT [25]) often enforce a bound on a linear objective. The algorithmic approach to bi-objective optimization presented in this thesis also makes heavy use of cardinality constraints. Formally, for a set L of literals and a bound $b \in \mathbb{N}$, `As-CNF` $(\sum_{l \in L} l \circ b)$ denotes a CNF formula encoding the cardinality constraint (i.e., linear (in)equality) $\sum_{l \in L} l \circ b$, where $\circ \in \{<, >, \geq, \leq, =\}$. Numerous methods of forming such CNF formulas are known (e.g., [41, 53, 54]).

In this work we make use of the so-called totalizer encoding [53, 55]. Given a set L of n input literals and a bound $k \in \{1, \ldots, n\}$, the (incremental) totalizer encoding produces a CNF formula $\textsc{Tot}(L, k)$ that defines a set $\{\langle L < 1\rangle, \ldots, \langle L < k+1\rangle\} \subset \textsc{var}(\textsc{Tot}(L, k))$ 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 $\textsc{Tot}(L, k)$ and $b < k$, then $\tau(\langle L < b\rangle) = 1$ if and only if $\sum_{l \in L} \tau(l) < b$. For applications where only either upper or lower bounding of the number of literals assigned to true is needed, the size of the encoded totalizer can be reduced by encoding implications in only one direction instead of both (i.e., $\langle L < b\rangle \leftarrow (\sum_{l \in L} l < b)$ or $\langle L < b\rangle \rightarrow \sum_{l \in L} l < b$, but not both). The algorithmic approach presented in this thesis only uses upper bounding cardinality constraints, therefore we employ the size-reduced version of the totalizer encoding. 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 $k' > k$. This is desirable when making incremental SAT calls where the bound or the set of input literals changes between calls, since it allows for making earlier calls to the SAT solver on a formula with fewer clauses and retaining information from these calls. Extending the totalizer means adding clauses while reusing the ones that were previously added.

We use $\langle L \leq b \rangle$ as a shorthand for the literal $\langle L < b + 1 \rangle$; furthermore, if the maximal bound k of a totalizer $\textsc{Tot}(L, k)$ is clear from context or the bound is $k = |L|$, we omit it and simply write $\textsc{Tot}(L)$.

2.4 Maximum Satisfiability

Maximum satisfiability (MaxSAT) [25] is the optimization variant of SAT. In this work, by MaxSAT we refer to *weighted partial* MaxSAT, in which a set of *hard* clauses F and a set of *soft* literals O are given.* A solution of a MaxSAT instance is any solution (satisfying assignment) of the hard clauses F. Each literal $l \in \mathrm{O}$ has an associated weight w_l. The task is to find an optimal solution, i.e., a solution that minimizes the linear function $\sum_{l \in \mathrm{O}} \tau(l) \cdot w_l$. Since the $\mathcal{NP}$-complete SAT decision problem (recall Section 2.1) can clearly be solved by MaxSAT (its optimization extension), MaxSAT is $\mathcal{NP}$-hard.

In the same way that SAT can be used as a declarative language to solve other decision problems, MaxSAT can be used to solve various types of $\mathcal{NP}$-hard optimization problems declaratively.

> **Example 2.4.** Recall the set covering problem from Example 2.2. By removing the constraint on the size of the cover and asking for a *smallest* cover, we can change the problem from a decision to an optimization problem. For modelling this problem as MaxSAT, we use the clauses representing the sets as hard clauses and each variable as a soft literal. In terms of the toy instance from Example 2.2, we use the clauses C_A, C_B and C_C as hard clauses and all variables as soft literals. In the situation where guest C does *not* like prawns, an optimal solution to this instance is $\tau_1(v_{\text{courgette}}) = \tau_1(v_{\text{chicken}}) = \tau_1(v_{\text{mushroom}}) = 1$ and $\tau_1(v) = 0$ for all other v. This encodes that a pizza

*We note that defining MaxSAT with soft *literals* is non-standard. However, soft clauses can be modelled as soft literals by adding a new relaxation variable to the clause, treating the clause as hard and the relaxation variable as a soft literal.

> needs to include at least three toppings so that every guest likes at least one of them and one possible set of such toppings is courgette, chicken, and mushroom. If guest C likes prawns, an optimal solution is $\tau_2(v_{\text{bell pepper}}) = \tau_2(v_{\text{prawns}}) = 1$ and $\tau_2(v) = 0$ for all other v.

Many algorithms for solving MaxSAT have been proposed in recent years [40, 44–46, 56, 57]. Early approaches were based on the branch-and-bound scheme [58–62]. Over the years, branch-and-bound was mostly displaced by algorithms that solve MaxSAT via solving a number of SAT instances with the help of an underlying SAT solver [24], or solving MaxSAT via integer programming [25]. Recently, there have also been results showing that branch-and-bound combined with clause learning may also achieve good performance [57, 63]. MaxSAT algorithms using SAT solvers can be categorized w.r.t. how the objective function is modelled: many algorithms encode cardinality constraints in CNF whereas the implicit hitting set approach [56, 64–66] employs an integer programming solver to natively handle the objective. This thesis builds on MaxSAT algorithms that solve a sequence of SAT problems. These algorithms can be further categorized as either solution-improving, bound-improving or core-guided algorithms. More detailed descriptions for most of these algorithms can be found in [25].

The two conceptually-simplest algorithms we build on are solution-improving SAT-UNSAT search [40] and bound-improving UNSAT-SAT search [46]. SAT-UNSAT (also known as Linear SAT-UNSAT (LSU) [25]) search solves MaxSAT by starting from a known satisfiable solution for the hard clauses. From this, a cardinality constraint is added to the SAT solver, enforcing that the next found solution achieves a better objective value than the last. If such a solution is found, the cardinality constraint is tightened to the objective value of this new solution. As soon as the SAT solver returns `UNSAT` for a call, the last found solution is known to be optimal. As this search procedure goes through a series of satisfiable calls first, terminating at the first unsatisfiable call, it is known as SAT-UNSAT search. In contrast to SAT-UNSAT, UNSAT-SAT search [46, 67] find the optimal value by lower-bounding the objective value instead. It starts by adding a tight cardinality constraint to the SAT solver—resulting in unsatisfiable queries—and slowly loosening the constraint until a first satisfiable query is reached.

The other two algorithms we are building on in this thesis are MSU3 [42] and OLL [44, 45]. Both of these search procedures are core-guided, meaning they make use of unsatisfiable cores returned by the SAT solver. Core-guided MaxSAT was first proposed with the algorithm now known as Fu-Malik [46]. The central insight behind core-guided MaxSAT is

that an optimal solution must assign at least one of the soft literals in every core to true. In the MSU3 algorithm, the soft literals are assumed to false and cores are extracted over them. When a soft literal appears in a core, this literal is removed from the assumptions and then added to a cardinality constraint allowing some soft literals to be true. Every iteration increases the bound of the cardinality constraint by one. With this, MSU3—as also every other core-guided algorithm—makes a series of unsatisfiable SAT queries, terminating at the first satisfiable one, which yields the optimal solution. OLL, which was first proposed for the paradigm of answer set programming [68] and later applied to MaxSAT [44, 45], differs from MSU3 in how the extracted cores are relaxed. Instead of building one large cardinality constraint over all cores, it builds an individual cardinality constraint for each core. Furthermore, these cardinality constraints are not treated as hard but as additional *soft* clauses, meaning they can be relaxed in subsequent iterations of the algorithm.

3 Bi-Objective Optimization

In this chapter, we provide background on bi-objective optimization, describing the problem setting considered as an instantiation of the more general case of multi-objective optimization. We also provide an overview of other notions of optimality (apart from Pareto optimality) for multiple objectives and existing approaches to solving bi-objective optimization problems.

3.1 Multi-Objective Pareto Optimization

Given p objective functions $f_i : \mathcal{X} \to \mathbb{R}^+$, where $i = 1, \ldots, p$, in multi-objective optimization the task is to find one or more $x \in \mathcal{X}$ that are optimal (under a predetermined notion of optimality) w.r.t. p objective functions [69]. In this work, w.l.o.g. we consider *minimization* problems. Formally, a multi-objective optimization problem (MOOP) is of the form

$$\min(f_1(x), \ldots, f_p(x)), \text{ subject to } x \in \mathcal{X}. \tag{3.1}$$

Problems with $p = 2$ are called *bi*-objective optimization problems.

Since the objectives might be in conflict with each other, for multi-objective optimization there is no single optimal tuple of objective function values. We say that two objectives are in conflict in the likely case that their respective global optima cannot be reached at the same time. One central way to define optimality for multiple objectives is that of Pareto optimality.

Definition 3.1 (Dominated solutions [26])**.** Given a MOOP as defined in Equation (3.1) and two solutions $x, x' \in \mathcal{X}$, x dominates x' (w.r.t. $f_1, \ldots, f_p$) if (i) $f_i(x) \leq f_i(x')$ for all $i = 1, \ldots, p$, and (ii) $f_i(x) < f_i(x')$ for some $i \in \{1, \ldots, p\}$. We represent x dominating x' by $x \prec x'$.

Definition 3.2 (Pareto optimality [26])**.** Given a MOOP as defined in Equation (3.1), a solution $x \in \mathcal{X}$ is Pareto-optimal (w.r.t. $f_1, \ldots, f_p$) if and only if there is no $x' \in \mathcal{X}$ such that $x' \prec x$, i.e., x is not dominated by any other solution.

When the objectives are clear from context, we will simply say that a solution x is Pareto-optimal. Note that there can be multiple Pareto-optimal solutions to a MOOP. The set of all Pareto-optimal solutions is called the Pareto front (w.r.t. $f_1, \ldots, f_p$); the tuple $(f_1(x), \ldots, f_p(x))$ for a Pareto-optimal x—i.e., the image of x in the space defined by the objective functions—is a Pareto point (also called non-dominated point in literature [26]). There can be multiple Pareto-optimal solutions that correspond to the same Pareto point.

We consider three central related tasks for solving multi-objective optimization problems under Pareto optimality.

(i) Finding a single Pareto-optimal solution (e.g., [70]),

(ii) finding a representative solution for every Pareto point (e.g., [28, 29]), and

(iii) finding all Pareto-optimal solutions (e.g., [71]).

Our algorithmic approach can be used for solving all three of these tasks.

3.2 Bi-Objective Optimization in a SAT Context

As in MaxSAT (recall Section 2.4), we consider optimization problems with linear objective functions. We extend the formalization of MaxSAT to bi-objective optimization in the following way. The set of feasible solutions is declaratively represented as the CNF formula F. An objective O is a multiset of literals. Formalizing an objective as multisets allows for representing integer weights by adding a literal multiple times. 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 literals in O that τ assigns to 1. A bi-objective instance in the SAT context is a triple of a CNF formula and two objectives.

Example 3.3. An example formula F and two objectives $\mathrm{O_I}$ and $\mathrm{O_D}$ are shown on the left side in Figure 3.1. The solution space is illustrated on the right. The three solid dots correspond to the three Pareto points of F w.r.t. $\mathrm{O_I}$ and $\mathrm{O_D}$. Examples of Pareto-optimal solutions corresponding to these points are $\tau_1^o = \{i_2, d_1, d_3, d_4, \neg i_1, \neg i_3, \neg i_4, \neg d_2\}$, $\tau_2^o = \{i_1, i_2, d_1, d_2, \neg i_3, \neg i_4, \neg d_3, \neg d_4\}$ and $\tau_3^o = \{i_1, i_3, i_4, d_2, \neg i_2, \neg d_1, \neg d_3, \neg d_4\}$. The solution $\tau_3^c = \{i_2, d_1, d_2, d_3, d_4, \neg i_1, \neg i_3, \neg i_4\}$ is dominated by τ_1^o ($\tau_1^o \prec \tau_3^c$) because $\mathrm{O_I}(\tau_1^o) \leq \mathrm{O_I}(\tau_3^c)$ and $\mathrm{O_D}(\tau_1^o) < \mathrm{O_D}(\tau_3^c)$.

$$F = \Big\{ \texttt{As-CNF}\left(\sum_{l \in \mathrm{O_I} \cup \mathrm{O_D}} l \geq 4 \right),$$
$$(i_1 \vee i_2), (i_2 \vee i_3), (i_2 \vee i_4)$$
$$(d_1 \vee d_2), (d_2 \vee d_3), (d_2 \vee d_4) \Big\},$$
$$\mathrm{O_I} = \{i_1, i_2, i_3, i_4\},$$
$$\mathrm{O_D} = \{d_1, d_2, d_3, d_4\}$$

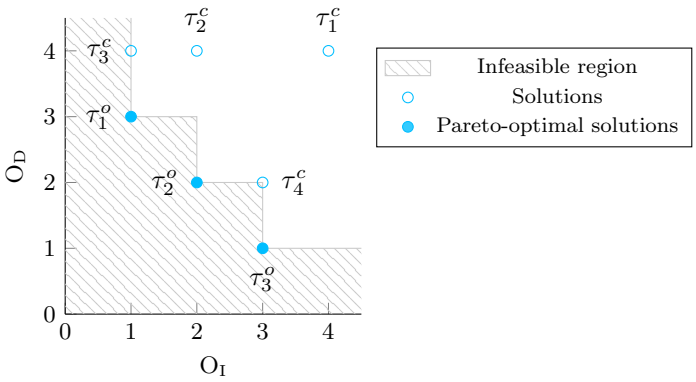

Figure 3.1: Left: An example formula F and two objectives $\mathrm{O_I}$ and $\mathrm{O_D}$. Right: the feasible region of F in the objective space defined by $\mathrm{O_I}$ and $\mathrm{O_D}$. The solutions τ_1^o and τ_2^o (solid points) are Pareto-optimal, while τ_i^c for $i = 1, \ldots, 4$ are not.

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

> **Observation 3.4** (Adapted from [72])**.** Sorting the Pareto-optimal solutions of a bi-objective optimization problem under the objectives f_1 and f_2 w.r.t. increasing values of f_1 amounts to sorting them w.r.t. decreasing values of f_2 and vice-versa.

> **Example 3.5.** Consider the formula F, the objectives $\mathrm{O_I}$ and $\mathrm{O_D}$ and the three Pareto-optimal solutions τ_1^o, τ_2^o and τ_3^o from Figure 3.1 and Example 3.3. By Observation 3.4, sorting the Pareto-optimal solutions by decreasing value for one objective results in sorting them by increasing values for the other objective: we have $\mathrm{O_I}(\tau_1^o) = 1 < \mathrm{O_I}(\tau_2^o) = 2 < \mathrm{O_I}(\tau_3^o) = 3$ and $\mathrm{O_D}(\tau_1^o) = 3 > \mathrm{O_D}(\tau_2^o) = 2 > \mathrm{O_D}(\tau_3^o) = 1$.

3.3 On Other Notions of Optimality

As mentioned in Section 3.1, Pareto optimality is only one—although a central—notion of optimality for multiple objectives. Two other notions of optimality for bi-objective optimization that narrow down the set of solutions considered optimal are *lexicographic optimization* and *lexicographic max-ordering optimization* [73]. These notions of optimality can be seen as a way of specifying in advance which Pareto-optimal solutions are of interest. The solutions considered optimal are a subset of all Pareto-optimal solutions [73] and every algorithm finding all Pareto-optimal solutions will therefore also find all solutions optimal under these other notions. For this reason, finding an optimal solution under these

other notions of optimality can be considered easier than finding a representative solution for every Pareto point or enumerating all Pareto-optimal solutions. Both lexicographic optimization and lexicographic max-ordering optimization are applicable for any number of objectives. Here we describe them in the context of bi-objective optimization.

In lexicographic optimization [73], a preference over the objectives is enforced, considering only one of the "end points" of the Pareto front—i.e., a Pareto-optimal solution with the smallest value for the objective chosen as primary—optimal. Formally, given a feasible set $\mathcal{X}$ and two objectives f_1 and f_2, a solution x dominates another solution x' in the lexicographic sense if (a) $f_1(x) < f_1(x')$, or (b) $f_1(x) = f_1(x')$ and $f_2(x) < f_2(x')$. Intuitively, lexicographic optimization minimizes f_1, using f_2 as a tie-breaker. The comparison criterion can also be seen as lexicographically comparing the string of objective values of two solutions, hence the name of the notion of optimality.

> **Example 3.6.** Consider again the formula F and the objectives $\mathrm{O_I}$ and $\mathrm{O_D}$ from Figure 3.1. Assume the objective $\mathrm{O_I}$ is chosen as the objective with higher priority. In this case, all solutions corresponding to the Pareto point $(3, 1)$ (e.g., $\tau_1^o = \{i_2, d_1, d_3, d_4, \neg i_1, \neg i_3, \neg i_4, \neg d_2\}$) are lexicographically optimal.

Lexicographic optimization can be cast into an optimization problem with a single objective with the help of the weighted sum method [70]. This is a common approach to solving lexicographic optimization, since solving algorithms for optimization problems with one objective are widely available (e.g., MaxSAT [25] or integer linear programming solvers [21]).

Lexicographic max-ordering (leximax) optimization [73] is closely related to lexicographic optimization. The difference between these two is that for leximax optimization, the objective values are sorted in descending order before comparing them lexicographically. This leads to the Pareto points with the smallest maximum objective value being considered optimal. Let $f_{\max}(x) = \max\{f_1(x), f_2(x)\}$ and $f_{\min}(x) = \min\{f_1(x), f_2(x)\}$. Formally, a solution x dominates another solution x' in the leximax sense if (a) $f_{\max}(x) < f_{\max}(x')$, or (b) $f_{\max}(x) = f_{\max}(x')$ and $f_{\min}(x) < f_{\min}(x')$. Informally speaking, this notion of optimality seeks to keep all objective values low by minimizing the maximum value first. All leximax-optimal solutions are contained in the set of Pareto-optimal solutions. However, they might correspond to different Pareto points.

Example 3.7. Consider again the formula F and the objectives $\mathrm{O_I}$ and $\mathrm{O_D}$ in Figure 3.1. The solution $\tau_2^o = \{i_1, i_2, d_1, d_2, \neg i_3, \neg i_4, \neg d_3, \neg d_4\}$ is leximax-optimal since it has the smallest maximum objective value.

3.4 Approaches to Bi-Objective Optimization

In this section, we give an overview of earlier-proposed approaches to solving bi-objective optimization problems. The focus here lies on Section 3.4.1, surveying SAT-based approaches. In addition, we shortly survey exact approaches based on other declarative optimization paradigms, mainly constraint and mixed integer programming. We conclude the section by giving a brief overview of inexact methods and discuss approaches that make use of different optimality definitions.

3.4.1 SAT-Based Approaches

We highlight three key SAT-based approaches to bi-objective optimization under Pareto optimality: enumeration of P-minimal solutions, enumeration of Pareto-minimal correction sets, and Seesaw. Furthermore, we touch on SAT-based optimization under lexicographic optimality.

P-Minimal Solution Enumeration

The approach perhaps closest to the one presented in this thesis reduces finding Pareto points to the enumeration of so-called P-minimal solutions [28, 74]. This approach was originally proposed for solving constraint satisfaction problems [75] encoded in CNF via the order encoding [76]. We note that the approach can be used with any encoding that allows for representing an objective value as a sorted unary number. Here we present enumeration of P-minimal solutions using the totalizer encoding (recall Section 2.3) instead of the order encoding as it also produces a unary representation of the objective values.

Finding the Pareto points to the bi-objective optimization problem defined by F, O_1 and O_2 corresponds to enumerating the solutions of $F^{\mathrm{W}} = F \wedge \textsc{Tot}(\mathrm{O}_1) \wedge \textsc{Tot}(\mathrm{O}_2)$ that are subset-minimal w.r.t. the outputs of the totalizers assigned to 0. More precisely, if P is the set of output literals of $\textsc{Tot}(\mathrm{O}_1) \wedge \textsc{Tot}(\mathrm{O}_2)$, then the goal is to enumerate solutions

τ^m such that no other solution τ has $\{l \mid l \in P, \tau(l) = 0\} \subsetneq \{l \mid l \in P, \tau^m(l) = 0\}$. The procedure for enumerating such solutions [74] 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,

(iii) once a minimal solution τ^m has been found, 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.

We refer to the algorithm for enumerating P-minimal solutions as "P-minimal" for short.

Example 3.8. Consider the formula F and two objectives $\mathrm{O_I}$ and $\mathrm{O_D}$ from Figure 3.1. 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 result is satisfiable; assume the first solution obtained is $\tau_1^c = \{i_1, i_2, i_3, i_4, d_1, d_2, d_3, d_4\}$.* In order to minimize τ_1^c, the clause $(\langle \mathrm{O_I} < 4 \rangle \vee \langle \mathrm{O_D} < 4 \rangle)$ is added to the SAT solver, and the solver is invoked again under the assumptions $\{\langle \mathrm{O_I} \leq 4 \rangle, \langle \mathrm{O_D} \leq 4 \rangle\}$. 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 $\tau_5^c = \{i_1, i_3, i_4, d_1, d_3, d_4, \neg i_2, \neg d_2\}$. Again, a clause $(\langle \mathrm{O_I} < 3 \rangle \vee \langle \mathrm{O_D} < 3 \rangle)$ is added, and the SAT solver is queried with the assumptions $\{\langle \mathrm{O_I} \leq 3 \rangle, \langle \mathrm{O_D} \leq 3 \rangle\}$. The result is `SAT`; assume the solution obtained is $\tau_2^o = \{i_2, d_1, d_3, d_4, \neg i_1, \neg i_3, \neg i_4, \neg d_2\}$. P-minimal then adds the clause $(\langle \mathrm{O_I} < 2 \rangle \vee \langle \mathrm{O_D} < 2 \rangle)$ and invokes the solver again under the assumptions $\{\langle \mathrm{O_I} \leq 2 \rangle, \langle \mathrm{O_D} \leq 2 \rangle\}$. The result is `UNSAT` 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.

As presented in [28], P-minimal enumerates a single representative solution per Pareto point. It can, however, be extended to enumerating all Pareto-optimal solutions by adding a relaxation variable to the clause added at each iteration and iteratively blocking solutions once they have been proven to be Pareto-optimal. More details on this are given later on Section 5.2.

*Note that the assignment of the auxiliary and output variables of the totalizer encoding are implied by the assignment of the input variables. For this reason we omit them when giving the assignment of a solution.

Enumeration of Pareto-Minimal Correction Sets

An approach for computing Pareto-optimal solutions via so-called Pareto-minimal correction sets (ParetoMCSes) has been previously proposed [30, 77, 78]. A ParetoMCS (w.r.t. two objectives O_1 and O_2) consists of two sets of literals (M_1, M_2) such that (i) $M_1 \subset \mathrm{O}_1$ and $M_2 \subset \mathrm{O}_2$, and (ii) there is a Pareto-optimal solution τ that sets $\tau(l) = 1$ for all $l \in M_1 \cup M_2$ and $\tau(l) = 0$ for all other $l \in (\mathrm{O}_1 \cup \mathrm{O}_2) \setminus (M_1 \cup M_2)$. Computing Pareto-optimal solutions can be reduced to the computations of ParetoMCSes [30]. The task of computing ParetoMCSes is accomplished by enumerating all subsets $T \subset (\mathrm{O}_1 \cup \mathrm{O}_2)$ for which (i) $F \wedge \bigwedge_{l \in (\mathrm{O}_1 \cup \mathrm{O}_2) \setminus T}(\neg l)$ is satisfiable, and (ii) $F \wedge \bigwedge_{l \in (\mathrm{O}_1 \cup \mathrm{O}_2) \setminus T'}(\neg l)$ is unsatisfiable for all $T' \subsetneq T$. Let $\mathcal{T}$ be the collection of all such sets. The Pareto-optimal solutions are obtained by extracting the solutions satisfying $F \wedge \bigwedge_{l \in (\mathrm{O}_\mathrm{I} \cup \mathrm{O}_\mathrm{D}) \setminus T}(\neg l)$ for all $T \in \mathcal{T}$ and removing the dominated ones [30]. The computation of $\mathcal{T}$ corresponds to enumeration of minimum correction sets, to which numerous algorithms have been proposed [79–81]. 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 $\mathcal{T}$ has been computed.

> **Example 3.9.** Consider the formula F and two objectives O_I and O_D from Figure 3.1. The ParetoMCS enumeration procedure will return the solution $\tau = \{i_1, i_3, i_4, d_1, d_3, d_4, \neg i_2, \neg d_2\}$ since no solution τ' of F has $\{l \in \mathrm{O}_\mathrm{I} \cup \mathrm{O}_\mathrm{D} \mid \tau'(l) = 1\} \subsetneq \{i_1, i_3, i_4, d_1, d_3, d_4\}$. The solution τ is not Pareto-optimal, but only filtered out when a solution that dominates it is enumerated. However, there are no guarantees on when such a dominating solution is found.

We will refer to this algorithm for enumerating Pareto-optimal solutions via enumerating ParetoMCSes as "ParetoMCS" for short.

Implicit Hitting Set Approach: Seesaw

The implicit hitting set approach [82–86] has, among other formalisms and problems, been successfully applied to MaxSAT [56, 64–66] and other SAT-related applications [87–89]. Recently, Seesaw [29] was proposed as a generalized implicit hitting set framework for bi-objective optimization. The overarching idea is that an optimization problem is modelled as a set $\mathcal{K}$ of so-called *cores* which represent an undesirable or conflicting substructure of the problem. Note that these cores are not necessarily equal to a core in SAT solving as described in Section 2.2; for this reason, we will be referring to them as *optimization cores*

from now on. For multiple objectives, an optimization core is only equivalent to a core if the decreasing objective is constrained to not be worse than the last value. In contrast to our work, in Seesaw one of the objectives is treated as a black box. This black box view does, however, also allow for SAT-based instantiations of Seesaw.

In our context the Seesaw algorithm computes Pareto-optimal solutions of a formula F w.r.t. $\mathrm{O_I}$ and $\mathrm{O_D}$ by maintaining a collection $\mathcal{K}$ of optimization cores that are subsets of $\mathrm{O_I}$. Informally speaking, in the bi-objective setting, 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 $\mathtt{hs} \subset \mathrm{O_I}$ (using an integer programming solver), i.e., a subset-minimal set of literals of $\mathrm{O_I}$ that intersects with each optimization core in $\mathcal{K}$. Next, a solution τ is computed, so that $\tau(l) = 1$ for each $l \in \mathtt{hs}$, $\tau(l) = 0$ for each $l \in \mathrm{O_I} \setminus \mathtt{hs}$, and $\mathrm{O_D}(\tau)$ is the smallest possible value for all such solutions if one exists. This is the step which employs a SAT solver in our instantiations of the algorithm. Seesaw 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 minimum value $\mathrm{O_D}(\tau)$ is Pareto-optimal [29].

Example 3.10. Consider the formula F and two objectives $\mathrm{O_I}$ and $\mathrm{O_D}$ from Figure 3.1. Initially, there are no optimization cores, so $\mathcal{K} = \emptyset$ and $\mathtt{hs} = \emptyset$. Since there is no τ that sets $\tau(l) = 0$ for each $l \in \mathrm{O_I}$, the iteration ends by extracting the optimization core $\mathrm{O_I}$. The intuition is, that any solution τ of F sets at least one variable in $\mathrm{O_I}$ to 1. In the next iteration, a minimum hitting set over $\mathcal{K} = \{\mathrm{O_I}\}$ is computed. There are a number of alternatives; assume $\mathtt{hs} = \{i_1\}$. Since there is no τ that sets $\tau(i_2) = \tau(i_3) = \tau(i_4) = 0$, the iteration ends with extracting the optimization core $\{i_2, i_3, i_4\}$. The same intuition as earlier holds for this core. Assume the next hitting set computed is $\mathtt{hs} = \{i_2\}$. Now there is a τ that sets $\tau(i_1) = \tau(i_3) = \tau(i_4) = 0$; one that also minimizes $\mathrm{O_D}(\tau)$ is $\tau_1^o = \{i_2, d_1, d_3, d_4, \neg i_1, \neg i_3, \neg i_4, \neg d_2\}$. The iteration ends with extracting the optimization core $\kappa = \{i_1, i_3, i_4\}$. Now the intuition is that, since τ_1^o minimizes $\mathrm{O_D}$ over solutions that assign $\tau(l) = 0$ for every $l \notin \mathtt{hs}$, every solution that obtains a lower value of $\mathrm{O_D}$ assigns at least one literal of κ to 1. Assume next we get $\mathtt{hs} = \{i_3\}$ for which no corresponding solution exists; the optimization core $\{i_1, i_2, i_4\}$ is added to $\mathcal{K}$. Now we have $\mathcal{K} = \{\mathrm{O_I}, \{i_2, i_3, i_4\}, \{i_1, i_3, i_4\}, \{i_1, i_2, i_4\}\}$; the only minimum hitting set is $\mathtt{hs} = \{i_4\}$. There is no τ that sets $\tau(i_1) = \tau(i_2) = \tau(i_3) = 0$ so a new optimization core $\{i_1, i_2, i_3\}$ is extracted. Next, one possible hitting set is $\mathtt{hs} = \{i_1, i_3\}$. Since the size of

> the hitting set grew from 1 to 2, the algorithm concludes that τ_1^o is Pareto-optimal. The algorithm continues in this manner, finding the Pareto-optimal τ_2^o and τ_3^o in the process. After computing the hitting set consisting of all literals in $\mathrm{O_I}$, the core extracted is $\emptyset$ at which point the algorithm terminates.

Note that the optimization core extraction strategy that only computes $\mathrm{O_I} \setminus \texttt{hs}$ as the new optimization core detailed in the example corresponds to what is called the weakest possible strategy in the original paper [29]. Seesaw is only feasible in practice when using a stronger optimization core extraction strategy since Seesaw otherwise reduces to enumerating all subsets of $\mathrm{O_I}$ as hitting sets [29]. One such stronger strategy for extracting optimization cores that is generally applicable if the oracle function is anti-monotone was presented in the original paper [29]. When using a SAT-based instantiation of the black box objective in Seesaw, it is also possible to use cores extracted by the SAT solver as the optimization cores for Seesaw. More details on this are given with the concrete instantiations of Seesaw that we implement in Section 5.2.

Also note that, in contrast to BiOptSat and P-minimal, extending Seesaw as it was originally presented [29] 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 point, the non-deterministic hitting set computation might steer the algorithm past other solutions that obtain the same values.

SAT-Based Lexicographic Optimization

There is also earlier work on SAT-based lexicographic optimization (recall Section 3.3) [38, 39, 90]. Lexicographic optimization is closely related to the so-called multi-level optimization problem. In particular, both can be cast as single-objective optimization and solved with a MaxSAT solver [38, 39]. In fact, many modern MaxSAT solvers exploit multilevel properties of input instances in order to improve search efficiency [91, 92].

3.4.2 Other Declarative Optimization Paradigms

Beyond SAT-based approaches, multi-objective optimization has been studied in the context of other declarative optimization paradigms. An early algorithm that can be used in constraint programming [75] is based on the lexicographic method [93]. A branch-and-

bound-based algorithm that outperforms the previous algorithm was presented later [94]. This improved filtering algorithm was improved again by the Pareto constraint [72, 95]. The resulting search algorithm is similar to ParetoMCS in that it maintains a set $\mathcal{T}$ of solutions that do not dominate each other. When a new solution is found, any solution it dominates is removed from $\mathcal{T}$.

Other than approaches for finding all Pareto-optimal solutions, there have also been constraint programming algorithms proposed for finding leximax-optimal solutions. In [96], five algorithms for this problem are given. There is one branch-and-bound-based algorithm and another algorithm based on adding constraints to encode the sorted objective value vector, minimizing multiple times over that.

Multi-objective optimization has also been studied in the context of linear programming, mixed integer programming and zero-one-programming [97–99]. There are different algorithmic approaches in this field, some based on the Simplex algorithm [100, 101], others on branch-and-bound [27, 102] while a last category build on reducing the problem of finding a Pareto-optimal solution to single-objective mixed integer programming [103–105].

3.4.3 Inexact Approaches

Other than the exact algorithms, there are also inexact search algorithms for multi-objective optimization problems [106–110]. These algorithms are *not* guaranteed to return the exact Pareto front, but they will return approximately optimal solutions. Depending on the application, such an approximate solution might be sufficient. Inexact algorithms are mainly used for problems that are too large to solve them with exact methods under given resource constraints.

Literature on inexact optimization algorithms is vast, for a survey see [106]. Algorithms that have been extended to multiple objectives include, e.g., Pareto local search [107–109], simulated annealing [111–113] and evolutionary algorithms [19, 20, 110, 114]. Both of these categories of algorithms can be described as “local search style”, where one or more solutions are iteratively modified to find better solutions.

4 The BiOptSat Algorithm

In this chapter, we detail BiOptSat, the MaxSAT-based approach to bi-objective optimization developed in this work, together with its variants. Section 4.1 gives an overview of the algorithmic framework, while Section 4.2 presents five specific instantiations of one of the subroutines, based on established MaxSAT algorithms. Furthermore, in Section 4.3 we discuss refinements to BiOptSat.

4.1 Overview of the Algorithm

Algorithm 1 details the BiOptSat framework for computing the Pareto-optimal solutions of a given CNF formula F w.r.t. two given objectives $\mathrm{O_I}$ and $\mathrm{O_D}$. BiOptSat is an instantiation of the general lexicographic method [37] instantiated with a SAT solver. To find a Pareto-optimal solution, the lexicographic method for multi-objective optimization defines multiple iterative single-objective optimization problems minimizing the objectives in order, where later calls are under the additional constraint that the objectives already minimized by earlier calls cannot take worse values than the found minimum. Once a first Pareto-optimal solution is found, the search continues by adding a constraint that one of the objectives needs to be improved. The lexicographic method will enumerate all Pareto-optimal solutions in monotonically-increasing order of the first objective. By Observation 3.4, for bi-objective optimization this means that the solutions are enumerated in decreasing order for the second objective. With this intuition, we call objective $\mathrm{O_I}$ *increasing* and $\mathrm{O_D}$ *decreasing*.

In BiOptSat, the lexicographic method is instantiated with a single SAT solver that all subroutines make use of. This single solver instantiation 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 `Minimize-Inc` procedure sets the value of b_I to the smallest value for which there still is an undiscovered Pareto-optimal solution τ^o for which $\mathrm{O_I}(\tau^o) = b_\mathrm{I}$. The value of b_D is then set to $\mathrm{O_D}(\tau^o)$ by the `Solution-Improving-Search` procedure.

In the default configuration shown in Algorithm 1, BiOptSat solves the task of finding a single representative per Pareto point. In case one wishes to enumerate all Pareto-optimal

Algorithm 1 BiOptSat: MaxSAT-based bi-objective optimization

Input: A formula F, two objectives $\mathrm{O_I}$ and $\mathrm{O_D}$.

Output: Either a single representative for each Pareto point of F or the full Pareto front.

1: `InitSATsolver`(F)
2: $(\mathrm{res}, \tau) \leftarrow$ `isSAT`$(\emptyset)$ {Invokes the SAT solver on the formula}
3: **if** res $=$ `UNSAT` **then**
4: **return** "no solutions"
5: $b_\mathrm{D} \leftarrow \infty, b_\mathrm{I} \leftarrow -1$
6: **while** res $=$ `SAT` **do**
7: $(b_\mathrm{I}, \tau) \leftarrow$ `Minimize-Inc`$(b_\mathrm{D}, b_\mathrm{I}, \mathrm{O_I}(\tau))$ {Maintains $\textsc{Tot}(\mathrm{O_I})$ (or similar)}
8: $(b_\mathrm{D}, \tau) \leftarrow$ `Solution-Improving-Search`$(b_\mathrm{I}, \mathrm{O_D}(\tau))$ {Builds $\textsc{Tot}(\mathrm{O_D})$}
9: **yield** τ {Optionally: **yield** `EnumSols`$(b_\mathrm{D}, b_\mathrm{I})$}
10: $(\mathrm{res}, \tau) \leftarrow$ `isSAT`$(\{\langle \mathrm{O_D} < b_\mathrm{D} \rangle\})$

solutions, the `EnumSols` procedure enumerates all Pareto-optimal solutions τ^o for which $\mathrm{O_I}(\tau^o) = b_\mathrm{I}$ and $\mathrm{O_D}(\tau^o) = b_\mathrm{D}$. We note that finding a single Pareto-optimal solution accounts to terminating the algorithm as soon as the first Pareto-optimal solution is returned. This first solution is also guaranteed to be lexicographically optimal with the increasing objective having higher priority.

In detail, given a 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). Here, `isSAT`$(\mathcal{A})$ denotes an incremental invocation of the SAT solver initialized on Line 1, with the set of assumptions $\mathcal{A}$. It has three return parameters, $(\mathrm{res}, \tau, \kappa)$ where the first is either `SAT` or `UNSAT`, indicating if the internal formula is satisfiable with the given assumptions. If res $=$ `SAT`, τ is populated by a satisfying assignment and κ is not modified; if res $=$ `UNSAT`, κ is populated with an unsatisfiable core and τ is not modified. In case the returned core of a specific call is not used, we will omit it as a return parameter. On Line 2, if res is `UNSAT`, the formula has 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 -1 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 determined 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^m with $\mathrm{O_I}(\tau^m) = b_\mathrm{I}$ and $\mathrm{O_D}(\tau^m) < b_\mathrm{D}$ (Line 7). The parameters of the `Minimize-Inc` procedure are the bound b_D that the decreasing objective of the found solution needs to be below, b_I as a known lower and $\mathrm{O_I}(\tau^r)$ as a known upper bound on the minimum increasing objective value. We assume that `Minimize-Inc` maintains a way to enforce that $\mathrm{O_I}(\tau) < b$, e.g., through a totalizer $\textsc{Tot}(\mathrm{O_I})$, and that BiOptSat and all of its subroutines have access to a set of assumptions for enforcing this bound for any b. Details of different instantiations of the `Minimize-Inc` subroutine are discussed in Section 4.2.

Next, the algorithm employs *solution-improving search* [25, 40, 41] to minimize the decreasing objective, i.e., to compute the smallest b_D for which there is a solution τ^o with $\mathrm{O_I}(\tau^o) = b_\mathrm{I}$ and $\mathrm{O_D}(\tau^o) = b_\mathrm{D}$ (Line 8). The totalizer $\textsc{Tot}(\mathrm{O_D}, \mathrm{O_D}(\tau))$ is built 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 the known upper bound $b = \mathrm{O_D}(\tau)$—iteratively invoking the SAT solver under the assumptions $\{\langle \mathrm{O_D} < b \rangle, \langle \mathrm{O_I} \leq b_\mathrm{I} \rangle\}$ for decreasing values of b until the query is unsatisfiable. As soon as unsatisfiability is reached, `Solution-Improving-Search` returns $b_\mathrm{D} = b + 1$ and the last solution τ for which $\mathrm{O_I}(\tau) = b_\mathrm{I}$ and $\mathrm{O_D}(\tau) = b_\mathrm{D}$. 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 $\{\langle \mathrm{O_D} \leq b_\mathrm{D} \rangle, \langle \mathrm{O_I} \leq b_\mathrm{I} \rangle\}$ blocks every found solution by adding a clause that prevents the solver from finding that same solution again. `EnumSols` terminates as soon as not more solutions are found.

> **Example 4.1.** Invoke BiOptSat on the formula F and objectives $\mathrm{O_I}$, $\mathrm{O_D}$ detailed in Figure 4.1. The search starts by invoking a SAT solver on F. This call returns a solution, say $\tau_1^c = \{i_1, i_2, i_3, i_4, d_1, d_2, d_3, d_4\}$ 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 $\tau_3^c = \{i_2, d_1, d_2, d_3, d_4, \neg i_1, \neg i_3, \neg i_4\}$ 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 $\textsc{Tot}(\mathrm{O_D}, 4)$. The first call to the SAT solver is made with the assumptions $\mathcal{A} = \{\langle \mathrm{O_I} \leq 1 \rangle, \langle \mathrm{O_D} < 4 \rangle\}$. The query is satisfiable. Say that the solver

$$F = \Big\{ \texttt{As-CNF}\left(\sum_{l \in \mathrm{O_I} \cup \mathrm{O_D}} l \geq 4 \right),$$
$$(i_1 \vee i_2), (i_2 \vee i_3), (i_2 \vee i_4)$$
$$(d_1 \vee d_2), (d_2 \vee d_3), (d_2 \vee d_4) \Big\},$$
$$\mathrm{O_I} = \{i_1, i_2, i_3, i_4\},$$
$$\mathrm{O_D} = \{d_1, d_2, d_3, d_4\}$$

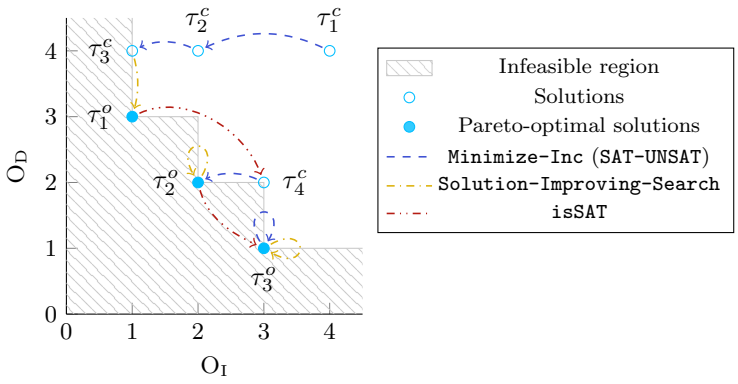

Figure 4.1: Left: The same example formula F and two objectives $\mathrm{O_I}$ and $\mathrm{O_D}$ as in Figure 3.1. Right: the feasible region of F in the objective space defined by $\mathrm{O_I}$ and $\mathrm{O_D}$. Furthermore, the search progression of the `SAT-UNSAT` variant of BiOptSat.

returns the solution $\tau_1^o = \{i_2, d_1, d_3, d_4, \neg i_1, \neg i_3, \neg i_4, \neg d_2\}$. Then, the solver is invoked with the assumptions $\mathcal{A} = \{\langle \mathrm{O_I} \leq 1 \rangle, \langle \mathrm{O_D} < 3 \rangle\}$. The query is unsatisfiable, so the procedure returns the Pareto-optimal τ_1^o and $b_\mathrm{D} = \mathrm{O_D}(\tau_1^o) = 3$. Now optionally, the procedure `EnumSols` can be used to enumerate all other solutions corresponding to the Pareto point $(\mathrm{O_I}(\tau_1^o), \mathrm{O_D}(\tau_1^o))$. At the end of the iteration, the SAT solver is queried with the assumption $\{\langle \mathrm{O_D} < 3 \rangle\}$. As the query is satisfiable and the solver returns, e.g., the solution $\tau_4^c = \{i_1, i_2, i_3, d_1, d_2, \neg i_4, \neg d_3, \neg d_4\}$, 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 $\tau_2^o = \{i_1, i_2, d_1, d_2, \neg i_3, \neg i_4, \neg d_3, \neg d_4\}$. `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 with the assumption $\{\langle \mathrm{O_D} < 2 \rangle\}$. This query is satisfiable and the solver returns, e.g., the solution $\tau_3^o = \{i_1, i_3, i_4, d_2, \neg i_2, \neg d_1, \neg d_3, \neg d_4\}$.

The last iteration starts by calling `Minimize-Inc` which returns $b_\mathrm{I} = 3$ and, e.g., again the solution τ_3^o. `Solution-Improving-Search`, again, cannot improve on the decreasing objective, so τ_3^o is also Pareto-optimal. Lastly, the SAT solver is queried with the assumption $\{\langle \mathrm{O_D} < 1 \rangle\}$. This query is unsatisfiable, terminating the algorithm.

4.2 Variants for Minimizing the Increasing Objective

We consider five different instantiations of the `Minimize-Inc` procedure for minimizing the increasing objective. The first four (`SAT-UNSAT`, `UNSAT-SAT`, `MSU3` and `OLL`) are inspired

Algorithm 2 `SAT-UNSAT` instantiation of `Minimize-Inc`

Input: Last bound b_{D} on O_{D} and known upper bound b on the minimum value of O_{I}.

Output: Solution τ and smallest $b = \mathrm{O}_{\mathrm{I}}(\tau)$ so that $\mathrm{O}_{\mathrm{D}}(\tau) < b_{\mathrm{D}}$.

1: build or extend $\textsc{Tot}(\mathrm{O}_{\mathrm{I}}, b)$ if necessary
2: $(\mathrm{res}, \tau) \leftarrow$ `isSAT`$(\{\langle \mathrm{O}_{\mathrm{D}} < b_{\mathrm{D}} \rangle, \langle \mathrm{O}_{\mathrm{I}} < b \rangle\})$
3: **while** res = `SAT` **do**
4: $\quad b \leftarrow \mathrm{O}_{\mathrm{I}}(\tau)$
5: $\quad (\mathrm{res}, \tau) \leftarrow$ `isSAT`$(\{\langle \mathrm{O}_{\mathrm{D}} < b_{\mathrm{D}} \rangle, \langle \mathrm{O}_{\mathrm{I}} < b \rangle\})$
6: **return** (b, τ)

by existing MaxSAT algorithms already described in Section 2.4 while the last (`MSHybrid`) switches between two MaxSAT-like algorithms to combine their advantages. We do not define similar variants for the `Solution-Improving-Search` procedure, since this procedure needs to perform upper-bounding search in order to be able to incrementally use the SAT solver.

4.2.1 `SAT-UNSAT`

`SAT-UNSAT` is a variant of solution-improving search [25, 40, 41] that is also used for minimizing O_{D}. The inputs to the procedure are the last bound b_{D} on O_{D} and the upper bound $b = \mathrm{O}_{\mathrm{I}}(\tau)$ on the minimum value of the increasing objective known from the last SAT solver call. The known lower bound is not needed for this variant. Since the last call is made on Line 10 with the assumption $\langle \mathrm{O}_{\mathrm{D}} < b_{\mathrm{D}} \rangle$, the solution τ will have $\mathrm{O}_{\mathrm{D}}(\tau) < b_{\mathrm{D}}$.

`SAT-UNSAT` is outlined as Algorithm 2. The procedure maintains the totalizer $\textsc{Tot}(\mathrm{O}_{\mathrm{I}})$ and begins on Line 11 by checking if the current upper bound on $\textsc{Tot}(\mathrm{O}_{\mathrm{I}})$ is at least b, extending the totalizer if not. Then the SAT solver is iteratively invoked with the assumptions $\{\langle \mathrm{O}_{\mathrm{D}} < b_{\mathrm{D}} \rangle, \langle \mathrm{O}_{\mathrm{I}} < b \rangle\}$ for decreasing values of b (Line 15). The procedure terminates when the query is unsatisfiable, after which (on Line 16) the value of b and the solution obtained during the final satisfiable call are returned as b_{I} and τ.

> **Example 4.2.** Consider the invocation of BiOptSat detailed in Example 4.1. 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 4.1. In the first iteration, `SAT-UNSAT` is invoked with $b_{\mathrm{D}} = \infty$ and $b = \mathrm{O}_{\mathrm{I}}(\tau_1^c) = 4$. At this point, the totalizer over O_{I} has not been built, so the procedure starts by adding

Algorithm 3 `UNSAT-SAT` instantiation of `Minimize-Inc`

Input: Last bound b_{D} on O_{D} and last bound b_{I} on O_{I}.

Output: Solution τ and smallest $b = \mathrm{O}_{\mathrm{I}}(\tau)$ so that $\mathrm{O}_{\mathrm{D}}(\tau) < b_{\mathrm{D}}$.

1: $b \leftarrow b_{\mathrm{I}}$
2: build or extend $\textsc{Tot}(\texttt{Act}, b+1)$
3: $(\mathrm{res}, \tau) \leftarrow \texttt{isSAT}(\{\langle \mathrm{O}_{\mathrm{D}} < b_{\mathrm{D}} \rangle, \langle \mathrm{O}_{\mathrm{I}} \leq b+1 \rangle\})$
4: **while** res = `UNSAT` **do**
5: $\quad b \leftarrow b+1$
6: $\quad$ extend $\textsc{Tot}(\mathrm{O}_{\mathrm{I}}, b+1)$
7: $\quad (\mathrm{res}, \tau) \leftarrow \texttt{isSAT}(\{\langle \mathrm{O}_{\mathrm{D}} < b_{\mathrm{D}} \rangle, \langle \mathrm{O}_{\mathrm{I}} \leq b+1 \rangle\})$
8: **return** $(b+1, \tau)$

$\textsc{Tot}(\mathrm{O}_{\mathrm{I}}, 4)$ to the solver. The first call to the SAT solver is made with the assumptions $\{\langle \mathrm{O}_{\mathrm{I}} < 4 \rangle\}$, since $b_{\mathrm{D}} = \infty$ and therefore no assumption constraining O_{D} is needed. The query is satisfiable. Assume that the solver returns the solution $\tau_2^c = \{i_1, i_2, d_1, d_2, d_3, d_4, \neg i_3, \neg i_4\}$. In the next iteration, the set of assumptions is $\{\langle \mathrm{O}_{\mathrm{I}} < 2 \rangle\}$. The query is again satisfiable and the solver returns, e.g., the solution $\tau_3^c = \{i_2, d_1, d_2, d_3, d_4, \neg i_1, \neg i_3, \neg i_4\}$. The SAT solver is then invoked with the assumptions $\{\langle \mathrm{O}_{\mathrm{I}} < 1 \rangle\}$. Now the query is unsatisfiable so the procedure terminates and returns $b_{\mathrm{I}} = 1$ and τ_3^c.

In the second iteration of BiOptSat, `SAT-UNSAT` is invoked with $b_{\mathrm{D}} = 3$ and $b = \mathrm{O}_{\mathrm{I}}(\tau_4^c) = 3$. The first call to the SAT solver is made with the assumptions $\{\langle \mathrm{O}_{\mathrm{D}} < 3 \rangle, \langle \mathrm{O}_{\mathrm{I}} < 3 \rangle\}$. The query is satisfiable. Assume that the solver returns the solution $\tau_2^o = \{i_1, i_2, d_1, d_2, \neg i_3, \neg i_4, \neg d_3, \neg d_4\}$. `SAT-UNSAT` invokes the SAT solver again with the assumptions $\{\langle \mathrm{O}_{\mathrm{D}} < 3 \rangle, \langle \mathrm{O}_{\mathrm{I}} < 2 \rangle\}$. The query is unsatisfiable, so the procedure returns $b_{\mathrm{I}} = 2$ and τ_2^o.

In the third (and last) iteration of BiOptSat, `SAT-UNSAT` is invoked with $b_{\mathrm{D}} = 2$ and $b = \mathrm{O}_{\mathrm{I}}(\tau_3^o) = 3$. The SAT solver is queried with the assumptions $\{\langle \mathrm{O}_{\mathrm{D}} < 2 \rangle, \langle \mathrm{O}_{\mathrm{I}} < 3 \rangle\}$, which is unsatisfiable. Hence, `SAT-UNSAT` returns $b_{\mathrm{I}} = 3$ and τ_3^o.

4.2.2 `UNSAT-SAT`

`UNSAT-SAT` takes an analogous approach to `SAT-UNSAT` search but searches for the smallest value by lower-bounding instead of upper-bounding [46]. As input parameters it receives the last bound b_{D} on O_{D} and the last bound b_{I} on O_{I} as a lower bound on the sought-after

smallest value. The upper bound on the smallest value is not made use of for this variant. `UNSAT-SAT` also maintains a totalizer $\textsc{Tot}(\mathrm{O_I})$.

The `UNSAT-SAT` instantiation of `Minimize-Inc` proceeds as illustrated in Algorithm 3. On Line 17, the bound b is set to the known lower bound b_I and the solver is then iteratively queried on Line 23 under the assumptions $\{\langle \mathrm{O_I} \leq b+1\rangle, \langle \mathrm{O_D} < b_\mathrm{D}\rangle\}$. If the query is unsatisfiable, the bound b is increased by 1 and the solver is queried again. The search ends once the query is satisfiable; in this case, the solution, and the bound are returned on Line 24. 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 $b+1$ (Line 22) and extended to the next bound in the next iteration. This way, the SAT solver is always queried over a minimum number of clauses.

Example 4.3. Consider the invocation of BiOptSat detailed in Example 4.1. Here we detail the invocation of `Minimize-Inc` instantiated as `UNSAT-SAT`. In the first iteration, `UNSAT-SAT` is invoked with $b_\mathrm{D} = \infty$ and $b_\mathrm{I} = -1$. At this point, the totalizer over $\mathrm{O_I}$ has not been built, so the procedure starts by initializing $\textsc{Tot}(\mathrm{O_I}, 0)$ and invokes the SAT solver with the assumptions $\{\langle \mathrm{O_I} \leq 0\rangle\}$. The query is unsatisfiable, so the totalizer is extended to $\textsc{Tot}(\mathrm{O_I}, 1)$ and the SAT solver invoked with the assumptions $\{\langle \mathrm{O_I} \leq 1\rangle\}$. The query is satisfiable. Assume that the returned solution is $\tau_3^c = \{i_2, d_1, d_2, d_3, d_4, \neg i_1, \neg i_3, \neg i_4\}$. Hence, `UNSAT-SAT` returns $b_\mathrm{I} = 1$ and τ_3^c.

In the second iteration of BiOptSat, `UNSAT-SAT` is invoked with $b_\mathrm{D} = 3$ and $b_\mathrm{I} = 1$. The totalizer is extended to $\textsc{Tot}(\mathrm{O_I}, 2)$ and the solver is invoked with the assumptions $\{\langle \mathrm{O_I} \leq 2\rangle, \langle \mathrm{O_D} < 3\rangle\}$. The query is satisfiable. Assume that the returned solution is $\tau_2^o = \{i_1, i_2, d_1, d_2, \neg i_3, \neg i_4, \neg d_3, \neg d_4\}$. Hence, the routine returns $b_\mathrm{I} = 2$ and τ_2^o.

In the last iteration of BiOptSat, `UNSAT-SAT` is invoked with $b_\mathrm{D} = 2$ and $b_\mathrm{I} = 2$. The totalizer is extended to $\textsc{Tot}(\mathrm{O_I}, 3)$ and the solver is invoked with the assumptions $\{\langle \mathrm{O_I} \leq 3\rangle, \langle \mathrm{O_D} < 2\rangle\}$. The query is satisfiable. Assume that the returned solution is $\tau_3^o = \{i_1, i_3, i_4, d_2, \neg i_2, \neg d_1, \neg d_3, \neg d_4\}$. Hence, `UNSAT-SAT` returns $b_\mathrm{I} = 3$ and τ_3^o.

4.2.3 `MSU3`

`MSU3` implements a core-guided approach inspired by the MSU3 MaxSAT algorithm [42]. The input parameters of this subroutine are the last bound b_D on $\mathrm{O_D}$ and the last bound b_I on $\mathrm{O_I}$ as a lower bound on the sought-after smallest value. The upper bound on the smallest value is not needed for this variant.

Algorithm 4 `MSU3` instantiation of `Minimize-Inc`

Input: Last bound b_{D} on O_{D} and last bound b_{I} on O_{I}.

Output: Solution τ and smallest $b = \mathrm{O}_{\mathrm{I}}(\tau)$ so that $\mathrm{O}_{\mathrm{D}}(\tau) < b_{\mathrm{D}}$.

1: $b \leftarrow \max\{b_{\mathrm{I}}, 0\}$
2: $(\mathrm{res}, \tau, \kappa) \leftarrow \mathtt{isSAT}(\{\langle \mathrm{O}_{\mathrm{D}} < b_{\mathrm{D}} \rangle, \langle \mathrm{O}_{\mathrm{I}} \leq b \rangle\} \cup \{\neg l \mid l \in \mathrm{O}_{\mathrm{I}} \setminus \mathtt{Act}\})$
3: **while** $\mathrm{res} = \mathtt{UNSAT}$ **do**
4: $\quad b \leftarrow b + 1$
5: $\quad \kappa \leftarrow \kappa \setminus \{\neg\langle \mathrm{O}_{\mathrm{D}} < b_{\mathrm{D}} \rangle, \neg\langle \mathrm{O}_{\mathrm{I}} \leq b \rangle\}$
6: $\quad \mathtt{Act} \leftarrow \mathtt{Act} \cup \kappa$
7: $\quad$ build or extend $\textsc{Tot}(\mathtt{Act}, b)$
8: $\quad (\mathrm{res}, \tau, \kappa) \leftarrow \mathtt{isSAT}(\{\langle \mathrm{O}_{\mathrm{D}} < b_{\mathrm{D}} \rangle, \langle \mathrm{O}_{\mathrm{I}} \leq b \rangle\} \cup \{\neg l \mid l \in \mathrm{O}_{\mathrm{I}} \setminus \mathtt{Act}\})$
9: **return** (b, τ)

The `MSU3` instantiation does not maintain a totalizer $\textsc{Tot}(\mathrm{O}_{\mathrm{I}})$, but a set $\mathtt{Act} \subset \mathrm{O}_{\mathrm{I}}$ of *active* objective literals and a totalizer $\textsc{Tot}(\mathtt{Act})$ built over them. Initially $\mathtt{Act} = \emptyset$, i.e., all literals of O_{I} are inactive. Informally speaking, an inactive literal $l \in \mathrm{O}_{\mathrm{I}} \setminus \mathtt{Act}$ is assumed to the value 0 in every invocation of the SAT solver until it is returned as part of a core. Algorithm 4 illustrates the search performed by `MSU3`. The algorithm starts from the value $b = b_{\mathrm{I}}$ computed in the previous iteration and invokes the SAT solver with the assumptions $\mathcal{A} = \{\langle \mathtt{Act} \leq b \rangle, \langle \mathrm{O}_{\mathrm{D}} < b_{\mathrm{D}} \rangle\} \cup \{\neg l \mid l \in \mathrm{O}_{\mathrm{I}} \setminus \mathtt{Act}\}$ on Line 26. If the query is unsatisfiable, the SAT solver returns a core $\kappa \subset \{\neg l \mid l \in \mathcal{A}\}$. Next, the bound b is increased by one, the inactive literals in κ are added to `Act` and the totalizer $\textsc{Tot}(\mathtt{Act})$ is extended (Lines 28–31). The procedure continues until the query is satisfiable, and a solution τ which has $\mathrm{O}_{\mathrm{I}}(\tau) = b$ and $\mathrm{O}_{\mathrm{D}}(\tau) < b_{\mathrm{D}}$ is found. At that point the value b is the minimum value $\mathrm{O}_{\mathrm{I}}(\tau)$ for any solution τ subject to $\mathrm{O}_{\mathrm{D}}(\tau) < b_{\mathrm{D}}$. This is because the value of b is increased monotonically, and the solver returned unsatisfiable in the second-to-last iteration.

For enforcing $\mathrm{O}_{\mathrm{I}}(\tau) \leq b_{\mathrm{I}}$ when employing `MSU3`, consider an invocation of $\mathtt{MSU3}(b_{\mathrm{D}}, b_{\mathrm{I}})$ made during BiOptSat and assume it returns the tuple (b_{I}, τ). In the next call to `Solution-Improving-Search`, the number of literals in O_{I} set to 1 needs to be restricted to at most b_{I}. Since the totalizer maintained by `MSU3` only has $\mathtt{Act} \subset \mathrm{O}_{\mathrm{I}}$ as inputs, we do not have access to an output literal of form $\langle \mathrm{O}_{\mathrm{I}} \leq b_{\mathrm{I}} \rangle$. Instead, we use the assumptions $\{\langle \mathtt{Act} \leq b_{\mathrm{I}} \rangle\} \cup \{\neg l \mid l \in \mathrm{O}_{\mathrm{I}} \setminus \mathtt{Act}\}$, i.e., restrict the number of literals in `Act` set to 1 to b_{I} and assume the value of each inactive literal $l \in \mathrm{O}_{\mathrm{I}} \setminus \mathtt{Act}$ to 0. In the following proposition, we prove that doing so can be done without removing any Pareto-optimal solutions from the search.

Proposition 4.4. Let τ^o be a Pareto-optimal solution of F for which $\mathrm{O_I}(\tau^o) = b_\mathrm{I}$. Then $\tau^o(l) = 0$ for all $l \in \mathrm{O_I} \setminus \mathtt{Act}$.

Proof. (Sketch) Since, b_I was returned by `MSU3`, we know that there exists 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 $l \in \mathrm{O_I} \setminus \mathtt{Act}$ will have $\mathrm{O_I}(\tau^n) > b_\mathrm{I}$. □

Example 4.5. Consider the invocation of BiOptSat detailed in Example 4.1. Here we detail the invocations of `Minimize-Inc` instantiated as `MSU3`. In the first iteration of BiOptSat, `MSU3` is invoked with $b_\mathrm{D} = \infty$ and $b_\mathrm{I} = -1$. Initially, the set $\mathtt{Act} = \emptyset$ of active literals is empty, so the first call to the SAT solver is made with the assumptions $\mathcal{A} = \{\neg i_1, \neg i_2, \neg i_3, \neg i_4\}$. The query is unsatisfiable. Assume that the solver returns $\kappa = \{i_1, i_2\}$. The literals in κ are marked as active and the totalizer $\textsc{Tot}(\mathtt{Act}, 1)$ is initialized. The SAT solver is then invoked with the assumptions $\mathcal{A} = \{\neg i_3, \neg i_4, \langle \mathtt{Act} \leq 1 \rangle\}$. The query is satisfiable so the procedure returns, e.g., $b_\mathrm{I} = 1$ and the solution $\tau_3^c = \{i_2, d_1, d_2, d_3, d_4, \neg i_1, \neg i_3, \neg i_4\}$.

In the next iteration of BiOptSat, `MSU3` is invoked with $b_\mathrm{D} = 3$ and $b_\mathrm{I} = 1$. The set $\mathtt{Act} = \{i_1, i_2\}$ is kept from the previous iterations, so the first call to the SAT solver is made with the assumptions $\mathcal{A} = \{\langle \mathrm{O_D} < 3 \rangle, \langle \mathtt{Act} \leq 1 \rangle, \neg i_3, \neg i_4\}$. The query is unsatisfiable. Assume that the core returned by the solver is $\kappa = \{\neg\langle \mathrm{O_D} < 3 \rangle, \neg\langle \mathtt{Act} \leq 1 \rangle, i_3, i_4\}$. The totalizer outputs $\neg\langle \mathrm{O_D} < 3 \rangle$ and $\neg\langle \mathtt{Act} \leq 1 \rangle$ are discarded, i_3 and i_4 are added to the active literals, and the totalizer is extended to $\textsc{Tot}(\mathtt{Act}, 2)$. The SAT solver is queried again with the assumptions $\mathcal{A} = \{\langle \mathrm{O_D} < 2 \rangle, \langle \mathtt{Act} \leq 2 \rangle\}$; the query is satisfiable. Assume that the returned solution is $\tau_2^o = \{i_1, i_2, d_1, d_2, \neg i_3, \neg i_4, \neg d_3, \neg d_4\}$. `MSU3` returns $b_\mathrm{I} = 2$ and τ_2^o.

In the last iteration, `MSU3` is invoked with $b_\mathrm{D} = 2$ and $b_\mathrm{I} = 2$. The SAT solver is queried with the assumptions $\mathcal{A} = \{\langle \mathrm{O_D} < 2 \rangle, \langle \mathtt{Act} \leq 2 \rangle\}$. The query is unsatisfiable; assume that the core is $\kappa = \{\neg\langle \mathrm{O_D} < 2 \rangle, \neg\langle \mathtt{Act} \leq 2 \rangle\}$. Both totalizer outputs are discarded, and the totalizer is extended to $\textsc{Tot}(\mathtt{Act}, 3)$. The solver is queried again with the assumptions $\mathcal{A} = \{\langle \mathrm{O_D} < 2 \rangle, \langle \mathtt{Act} \leq 3 \rangle\}$. The query is satisfiable. Assume that the returned solution is $\tau_3^o = \{i_1, i_3, i_4, d_2, \neg i_2, \neg d_1, \neg d_3, \neg d_4\}$. Then `MSU3` returns $b_\mathrm{I} = 3$ and τ_3^o.

Algorithm 5 `MSHybrid` instantiation of `Minimize-Inc`

Input: Last bound b_{D} on O_{D}, known upper and lower bounds b and b_{I} on min. of O_{I}.

Output: Solution τ and smallest $b = \mathrm{O}_{\mathrm{I}}(\tau)$ so that $\mathrm{O}_{\mathrm{D}}(\tau) < b_{\mathrm{D}}$.

1: **if** $|\texttt{Act}| < \texttt{thr} \cdot |\mathrm{O}_{\mathrm{I}}|$ **then**
2: $\quad(b_{\mathrm{I}}, \tau) \leftarrow \texttt{MSU3}(b_{\mathrm{D}}, b_{\mathrm{I}})$ {Immediately terminates once $|\texttt{Act}| < \texttt{thr} \cdot |\mathrm{O}_{\mathrm{I}}|$ is met}
3: **if** $|\texttt{Act}| \geq \texttt{thr} \cdot |\mathrm{O}_{\mathrm{I}}|$ **then**
4: $\quad$fully build or extend $\textsc{Tot}(\mathrm{O}_{\mathrm{I}}, b)$ if necessary
5: $\quad(b_{\mathrm{I}}, \tau) \leftarrow \texttt{SAT-UNSAT}(b_{\mathrm{D}}, b)$
6: **return** (b_{I}, τ)

4.2.4 `OLL`

As mentioned in Section 2.4, the OLL MaxSAT algorithm [44, 45] builds a cardinality constraint for each core extracted from the formula, treating the totalizer outputs as part of the objective. The same holds for the `OLL` instantiation of `Minimize-Inc`. Assume in the ith iteration, `OLL` extracts the core κ_i. It will build a totalizer $\textsc{Tot}(\kappa_i, 1)$ and enforce $\sum_{l \in \kappa_i} l \leq 1$ as an assumption. For all literals in the extracted core, if the literal is part of the objective, the literal is marked as active. If the literal is the output of a totalizer built over a previous core, the bound enforced on that totalizer is increased by one. 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 $\langle \mathrm{O}_{\mathrm{D}} < b_{\mathrm{D}} \rangle$. 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` by assuming the inactive literals of $l \in \mathrm{O}_{\mathrm{I}} \setminus \texttt{Act}$ to 0. Additionally, a set of assumptions over the totalizer outputs needs to be included.

4.2.5 `MSHybrid`

The final variant proposed in this work, `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 break down to `UNSAT-SAT`, meaning it is a lower-bounding search where the bound on the totalizer $\textsc{Tot}(\mathrm{O}_{\mathrm{I}})$ is increased by one every iteration until the SAT query is satisfiable. Existing work in MaxSAT suggests that `SAT-UNSAT` is a better

approach than using UNSAT-SAT for solving problems arising in real-world domains. 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, this advantage is gone. Furthermore, if a problem instance has literals in O_I that never appear in any cores (i.e., 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 as a hybrid 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 $\textsc{Tot}(\texttt{Act})$. The subroutine—that takes the last bound b_D on O_D, the known upper $b = O_I(\tau)$ and lower bounds b_I on the minimum of O_I as input—is outlined as Algorithm 5. Additionally, MSHybrid has a configuration parameter thr that defines at what percentage of the literals in O_I being active it switches from MSU3 to SAT-UNSAT. Initially MSHybrid will execute MSU3 on Line 35. If MSU3 finds the minimum without meeting the condition $|\texttt{Act}| < \texttt{thr} \cdot |O_I|$, the found minimum b_I and corresponding solution τ will be returned on Line 39. In case $|\texttt{Act}| < \texttt{thr} \cdot |O_I|$ is met during the execution of MSU3, MSU3 will immediately be terminated. On Line 37, $\textsc{Tot}(O_I, k)$ will then be fully built from the existing $\textsc{Tot}(\texttt{Act})$ and SAT-UNSAT invoked on Line 38. From now on, every call to MSHybrid will call SAT-UNSAT on Line 38. With this, the advantages of both MSU3 and SAT-UNSAT can in the best case be combined. A similar approach of combining core-guided and solution-improving search is known in incomplete MaxSAT solving as core-boosted linear search [115].

Example 4.6. Consider the invocation of BiOptSat detailed in Example 4.1. We detail the invocations of Minimize-Inc instantiated as MSHybrid. Since MSHybrid starts out as MSU3, the first invocation follows the description in Example 4.5.

Assume MSHybrid is configured to switch as soon as 70% of the literals in O_I are active ($\texttt{thr} = 0.7$). Since after the first iteration of BiOptSat we have $\texttt{Act} = \{i_1, i_2\}$, the second invocation of MSHybrid also starts as MSU3 since less than 70% of the literals in O_I are active. As soon as i_3 and i_4 become active, with the first core in the second invocation of MSU3, the MSU3 subroutine is terminated since the threshold for switching to SAT-UNSAT is reached. Because all literals in O_I are already active in this example and therefore included in $\textsc{Tot}(\texttt{Act})$, the totalizer does not need to be extended. SAT-UNSAT can directly be invoked as in the second iteration outlined in Example 4.2.

In the third iteration of BiOptSat, MSHybrid will directly invoke SAT-UNSAT, which proceeds as described in the third iteration in Example 4.2.

4.3 Refinements to BiOptSat

Finally, we discuss possible refinements to BiOptSat. Later on, we will empirically evaluate the impact of these refinements on the runtime performance of the algorithm.

4.3.1 Lazily Building Tot(O_D)

Assume that BiOptSat is invoked on a 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 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 Proposition 4.4. Essentially, the properties of cores imply that the Pareto-optimal solutions τ^o of F for which $O_I(\tau^o) = b_I$ assign $\tau^o(l) = 0$ for all $l \in (O_D \cap O_I) \setminus \texttt{Act}$.

The idea behind this refinement is to build the totalizer only over literals that can be assigned to 1 by Pareto-optimal solutions that are currently enumerated. With this, unnecessary clauses are removed from the working formula of the SAT solver.

Lazy building of Tot(O_D) requires a minor adjustment of the termination criterion of Algorithm 1. More specifically, as the totalizer maintained by `Solution-Improving-Search` might not have all literals of O_D as inputs, the algorithm does not have a (straightforward) way of checking if there exists a solution τ for which $O_D(\tau) < b_D$. 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 O_D.

4.3.2 Blocking of Dominated Solutions

Every time in BiOptSat that a candidate solution τ with objective values $b_I = O_I(\tau)$ and $b_D = O_D(\tau)$ is found, the definition of a Pareto-optimal point leads to the conclusion that all solutions τ^d with $O_I(\tau^d) > b_I$ and $O_D(\tau^d) > b_D$ cannot be Pareto-optimal. These points can all be blocked by adding the clause $\{\langle O_I \leq b_I \rangle, \langle O_D \leq b_D \rangle\}$ to the solver. Adding

this refinement might help prune the search space quicker and therefore speed up finding Pareto-optimal solutions.

4.3.3 Domain-Specific Solution Blocking

If multiple representatives of the same Pareto point are of interest, the procedure `EnumSols` needs to block all obtained solutions in order to enumerate all solutions corresponding to the same Pareto point. Naively, a found solution τ can be blocked with the clause $\{\neg l \mid l \in \tau \cap \text{LIT}(F^{\text{orig}})\}$, where F^{orig} is the original formula of the instance without any clauses added by the algorithm. In later sections we 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.

4.3.4 Bound Hardening

Because the values for $\text{O}_\text{D}(\tau)$ monotonically decrease while the algorithm progresses, this information can be passed on to the SAT solver for it to be able to draw logical conclusions from it. This is done by adding the literal $\langle \text{O}_\text{D} \leq b \rangle$ as a unit clause, as soon as the algorithm has verified that all remaining Pareto-optimal solutions τ^o have $\text{O}_\text{D}(\tau^o) \leq b$. Bound hardening could also be done for O_I, since it is known that the corresponding objective values will monotonically increase during the search. However, as mentioned in Section 2.3, the way BiOptSat uses totalizers, only upper bounds can be enforced with them. In preliminary experiments, we found that the advantage of bound hardening for the increasing objective does not justify the additional clauses added by building the full totalizer. For this reason, bound hardening is only employed for the decreasing objective.

4.3.5 Refinements to Core-Guided Variants

As further detailed in Chapter 5, our implementation of the BiOptSat variants `MSU3` and `OLL` make use of refinements commonly used in core-guided MaxSAT solving. More specifically, we employ core minimization [45] (either exact or heuristic) and core exhaustion [45, 116]. Additionally, we consider a disjoint core extraction phase [56].

Given a core κ returned by the SAT solver, heuristic core minimization refers to reinvoking the SAT solver with $\{\neg l \mid l \in \kappa\}$ 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 upper bound of each totalizer being added. For this, it increases the bound and checks if the SAT query is still unsatisfiable. A disjoint core phase refers to iteratively invoking the SAT solver in order to extract several disjoint sets of objective literals to add to the totalizer (when using `MSU3`) or build new totalizers over (when using `OLL`).

5 Experiments

We implemented all variants and refinements of BiOptSat described in Chapter 4 in C++. Our implementations of MSU3 and OLL were inspired by their implementations in Open-WBO v2.1 [117], the other variants were implemented from scratch. We used the state-of-the-art [118] SAT solver CaDiCaL v1.5.2 [119]. Our implementation is available in open source at https://bitbucket.org/coreo-group/bioptsat/. We compare the performance of this implementation to that of P-minimal, ParetoMCS and Seesaw. To the best of our knowledge, there is no general implementation of P-minimal and Seesaw publicly available. Therefore, we also implemented P-minimal and Seesaw from scratch, including them in our implementation.

In terms of refinements (recall Section 4.3), in our implementation, the core-guided variants of BiOptSat use heuristic core minimization by default. Dominated solutions are not blocked in `SAT-UNSAT`. As an additional parametric detail, in its default `MSHybrid` is configured to switch between `MSU3` and `SAT-UNSAT` once 70% of the literals in O_I have been added to Tot(`Act`).

We empirically evaluate the relative runtime performance of the BiOptSat variants against the three competing approaches, as well as the impact of the specific refinements (recall Section 4.3) to BiOptSat on their runtime performance. This comparison is done for the two tasks of finding a single representative solution per Pareto point and enumerating all Pareto-optimal solutions. We also investigate how much CPU time is spent in the `Minimize-Inc` compared to the `Solution-Improving-Search` subroutine of BiOptSat.

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.

5.1 Benchmarks

We empirically evaluate the performance of BiOptSat and the competing approaches on two bi-objective optimization problems: learning interpretable decision rules from data and bi-objective set covering.

5.1.1 Learning Interpretable Decision Rules

Recently, a variety of SAT and MaxSAT-based approaches for learning interpretable classifiers from data [11–14, 17, 31, 32] have been developed. 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 naturally giving 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 by Malioutov and Meel [11]. In short, a decision rule is a binary classifier in the form of a CNF formula over Boolean features. The result of the formula evaluated on the features of a data sample is the binary classification assigned by the classifier to this sample. Since MaxSAT only allows for a single objective, in [11] a linear combination of the two objectives, using a parameter $\lambda \geq 0$, was proposed. This is equivalent to the so-called weighted sum method [70]. 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 [37, 70]. In contrast, here we address directly the problem of computing all Pareto-optimal solutions w.r.t. the two objectives.

Example 5.1. Consider the sample dataset shown on the right with features x_1 and x_2, class label y and three samples. Two example decision rules are $r_1 = (x_1)$, $r_2 = (x_1 \wedge x_2)$. Rule r_1 has size 1 and classification error 1, while r_2 has size 2 and classification error 0. Both, r_1 and r_2, are Pareto-optimal w.r.t size and classification error.

x_1	x_2	y
1	1	1
0	1	0
1	0	0

For a given set of n data samples over m features, the encoding presented in [11] uses two sets of variables: s_l^j for $l = 1, \ldots, k$ and $j = 1, \ldots, m$, and η_i for $i = 1, \ldots, n$, for a specific number k of clauses in the decision rules to be learned. The interpretation of the variables is that $s_l^j = 1$ if and only if 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 label y_i and the Boolean features x_i^j, where $j = 1, \ldots, m$. With this, the encoding is

$$\neg\eta_i \rightarrow (y_i \leftrightarrow \bigwedge_{l=1}^{k} \bigvee_{j=1}^{m} (x_i^j \wedge s_l^j)), \text{ for } i = 1, \ldots, n.$$

We employ this encoding as F, literals s_l^j as $\mathrm{O_I}$ and literals η_i as $\mathrm{O_D}$. The task in this benchmark is therefore to find Pareto-optimal solution w.r.t. the size of the decision rule

(measured as the number of literals in the rule) and its classification error. In preliminary experiments, we found that choosing the objectives as described here leads to better performance than using classification error as the increasing objective.*

Since decision rules in CNF contain many symmetric solutions obtained by changing the order of clauses in the rule, we add additional clauses to the encoding to break these symmetries. The idea behind the symmetry breaking is that the bit-strings $\tau(s_l^1)\tau(s_l^2)\dots\tau(s_l^m)$ are forced to be in lexicographic ordering. In more detail, additionally to the s variables, we introduce variables e_l^j for $j = 1,\dots,m$ and $l = 2,\dots,k$ 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:

$$e_l^1 \leftrightarrow (s_{l-1}^1 \leftrightarrow s_l^1) \text{ and}$$
$$e_l^j \leftrightarrow (e_l^{j-1} \wedge (s_{l-1}^f \leftrightarrow s_l^j)) \text{ for } j = 2,\dots,m.$$

The lexicographic ordering is then enforced by adding the constraints

$$\neg e_l^1 \rightarrow (s_{l-1}^1 \wedge \neg s_{l-1}^1), \text{ and}$$
$$(e_l^{j-1} \wedge \neg e_l^j) \rightarrow (s_{l-1}^j \wedge \neg s_l^j) \text{ for } j = 2,\dots,m,$$

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.

> **Example 5.2.** Consider the rule $r_2 = (x_1 \wedge x_2)$ for the data in Example 5.1. It consists of two clauses, $C_1 = x_1$ and $C_2 = x_2$. In a solution τ to the encoding, C_1 will be represented as the bit-string $\tau(s_l^1)\tau(s_l^2) = 10$ and C_2 as $\tau(s_l^1)\tau(s_l^2) = 01$. Without symmetry breaking, either $\tau_1 = \{s_1^1, \neg s_1^2, \neg s_2^1, s_2^2\}$ or $\tau_2 = \{\neg s_1^1, s_1^2, s_2^1, \neg s_2^2\}$ would be valid solutions, even though they both map to r_2. The symmetry-breaking clauses enforce that the bit-string representing C_1 precedes the bit-string representing C_2. Therefore, only τ_1 is a feasible solution.

As the basis of our benchmark instances, we used 24 standard UCI [120] and Kaggle (https://www.kaggle.com) benchmark datasets, including ones used in the original evaluation of the encoding [11]; see Appendix A for details. We independently at random 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. The datasets were discretized

*This might be because the cores that can be extracted from the size objective are more beneficial for the search procedure.

as in [11]: categorical features are one-hot encoded, continuous features discretized by comparing to a collection of thresholds. All experiments on these datasets were run with the encoding configured to learn CNF decision rules consisting of two clauses ($k = 2$).

When enumerating multiple solutions corresponding to the same Pareto point, the blocking clauses for BiOptSat (as well as P-minimal 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. Further, 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.

5.1.2 Bi-Objective Set Covering

In the set covering problem over sets $\mathcal{S}$, a subset $\mathcal{C}$ of the set of elements $\{1, \ldots, n\}$ needs to be chosen under the constraints that (i) $\mathcal{C}$ covers all sets in $\mathcal{S}$, i.e., $\mathcal{C} \cap S \neq \emptyset$, $\forall S \in \mathcal{S}$, and (ii) $\mathcal{C}$ is minimal regarding some objective. The bi-objective variant assigns each element e two different cost values c_1^e and c_2^e where the two different objectives for the cover $\mathcal{C}$ are to minimize $c_1^{\mathcal{C}} = \sum_{e \in \mathcal{C}} c_1^e$ and $c_2^{\mathcal{C}} = \sum_{e \in \mathcal{C}} c_2^e$. When encoding set covering into 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 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 as a benchmark for evaluating P-minimal [28].

> **Example 5.3.** Consider the sets $\mathcal{S} = \{\{a, b\}, \{b, d\}\}$ and costs $c_1^a = c_1^b = c_1^d = c_2^a = c_2^d = 1$ and $c_2^b = 5$. The two covers $\mathcal{C}_1 = \{b\}$ and $\mathcal{C}_2 = \{a, b\}$ are Pareto-optimal with $\mathcal{C}_1$ having costs $c_1^{\mathcal{C}_1} = 1$ and $c_2^{\mathcal{C}_1} = 5$ and $\mathcal{C}_2$ costs $c_1^{\mathcal{C}_2} = 2$ and $c_2^{\mathcal{C}_2} = 2$.

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),

(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 $c \in [1, 100]$.

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, in BiOptSat it is enough to block the solution over all l_e that are assigned to true.

5.2 Competing Approaches

We compare BiOptSat to the three algorithms overviewed in Section 3.4.1: P-minimal, Seesaw and ParetoMCS. These algorithms were chosen because they are similar in that they are (or, in the case of Seesaw, can be) SAT-based, and they solve the same task of enumerating Pareto-optimal solutions. We implemented P-minimal and Seesaw using CaDiCaL [119] as the SAT solver (similarly as for BiOptSat) and CPLEX v20.10 for hitting set extraction. For ParetoMCS, we use the publicly-available Sat4j-based [40] implementation (https://gitlab.ow2.org/sat4j/moco).

BiOptSat can be used to enumerate both a single representative solution for each Pareto point and the full Pareto front. P-minimal can be extended to solve these same two tasks. For Seesaw, as mentioned before, extending it to enumerating the full Pareto front seems non-trivial, and for ParetoMCS, the given implementation only returns a single solution per Pareto point. We therefore only compare to Seesaw and ParetoMCS on the task of enumerating a single representative solution per Pareto point.

In our implementation we also extended P-minimal to the task of enumerating all solutions in the Pareto front. For achieving this, we add a new relaxation variable r to the clause added at each iteration for use as an assumption to enumerate all solutions at that Pareto point: the next SAT solver query is done including the assumption $\neg r$, if a dominating solution is found, the clause is made permanent, i.e., hardening it, by adding $\neg r$ as a unit clause. If no dominating solution is found, all solutions corresponding to the just discovered Pareto point can be enumerated when removing the assumption $\neg r$—effectively removing the clause that r appears in—by blocking every found solution and querying the solver again until it returns `UNSAT`. Once all solutions for that Pareto point are enumerated, the clause is hardened by adding $\neg r$ as a unit clause. If the next solution found dominates the previous one, we also harden the clause added in the previous iteration.

Example 5.4. Consider the same invocation of P-minimal as in Example 3.8. In order to enumerate all solutions in the Pareto front, the clause added in the first iteration is $(\langle \mathrm{O_I} < 4 \rangle \vee \langle \mathrm{O_D} < 4 \rangle \vee r_1)$ and the solver is queried again with the assumptions $\{\langle \mathrm{O_I} \leq 4 \rangle, \langle \mathrm{O_D} \leq 4 \rangle, \neg r_1\}$. Since the solver will return a dominating solution, the clause added is hardened by adding the unit clause $\neg r_1$ to the solver. The second iteration is modified similarly as the first, adding the relaxation variable r_2. In the third iteration, the added clause is $(\langle \mathrm{O_I} < 2 \rangle \vee \langle \mathrm{O_D} < 2 \rangle \vee r_3)$ and the solver call with assumptions $\{\langle \mathrm{O_I} \leq 2 \rangle, \langle \mathrm{O_D} \leq 2 \rangle, \neg r_3\}$ is unsatisfiable. Now, by iteratively querying the solver with the assumptions $\{\langle \mathrm{O_I} \leq 2 \rangle, \langle \mathrm{O_D} \leq 2 \rangle\}$ and blocking all found solutions, the set of solutions corresponding to the Pareto point $(2, 2)$ are enumerated.

The implementation of ParetoMCS allows for optionally enabling stratification [78]. We use stratification for a second variant of ParetoMCS, and refer to it as ParetoMCS-strat.

Since Seesaw needs to be instantiated for each task separately, we instantiated Seesaw for learning interpretable decision rules by using misclassifications as the objective over which optimization cores are extracted and a hitting set `hs` is found over these optimization cores. In the second step, the number of literals in the smallest rule misclassifying the samples in `hs` (or a subset of it) is found. This function is implemented as a solution-improving search with a SAT solver. This instantiation was chosen because finding the smallest rule misclassifying `hs` is an anti-monotone function and the refined version of core extraction presented in the original paper [29] can therefore be used, making Seesaw feasible in the first place. Furthermore, we implemented two more variants of Seesaw in which optimization cores are extracted in the solution-improving search procedure by the SAT solver. The final SAT query of solution-improving search will always be unsatisfiable; the core returned by the solver for this call can be used as an optimization core for Seesaw. Optionally we apply exact core minimization to these cores before using them in Seesaw.* These two variants will be referred to as Seesaw-SAT and Seesaw-SAT-min.

5.3 Results

We turn to an overview of the empirical results. First, we present a runtime comparison of the different approaches to bi-objective optimization for the task of finding a single

*We found that heuristic core minimization leads to no significant improvement over no minimization at all and therefore don't include it.

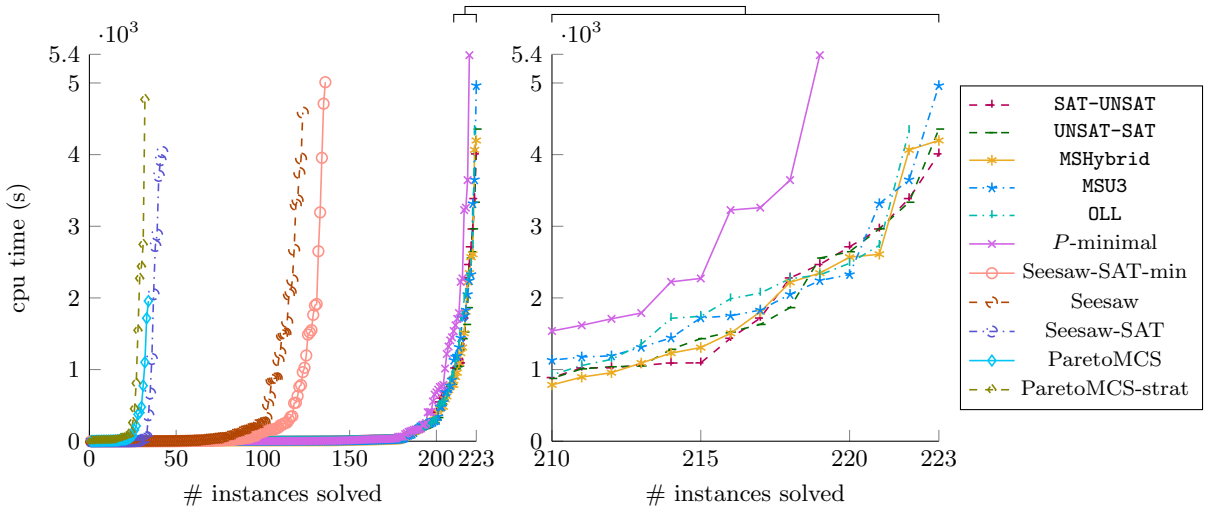

Figure 5.1: Runtime comparison between variants of BiOptSat and competing approaches for learning interpretable decision rules; enumeration of a single representative solution per Pareto point. The plot on the right shows a magnification for comparing the best-performing approaches.

representative solution per Pareto point. Then, we show the same comparison for enumerating all Pareto-optimal solutions. Finally, we provide observations on the impact the refinements presented in Section 4.3 have on the performance of BiOptSat.

5.3.1 Finding a Single Representative Solution per Pareto Point

For comparing the performance of the variants of BiOptSat presented in Section 4.2, *P*-minimal, ParetoMCS and Seesaw, we first discuss at the results for learning a single decision rule per Pareto point. Figure 5.1 shows the number of instances solved (horizontal axis) for different per-instance time limits (vertical axis) for this task. The best-performing approaches are the BiOptSat variants `MSHybrid`, `SAT-UNSAT`, `UNSAT-SAT` and `MSU3`, solving 223 instances, while *P*-minimal solved 219 instances. All variants of BiOptSat outperform *P*-minimal to some extent. Seesaw and ParetoMCS are very clearly outperformed by all other approaches. Even the best variant of Seesaw (Seesaw-SAT-min) only solves 136 instances within the resource constraints. ParetoMCS performs even worse, solving only 34 instances without and 32 with stratification.

Next, we consider a similar comparison for the bi-objective set covering benchmarks. In this comparison, only the Seesaw variants with SAT core extraction are included since the default Seesaw core extraction strategy cannot be applied for this problem. Figure 5.2 shows the number of solved instances given a per-instance time limit for the two generated

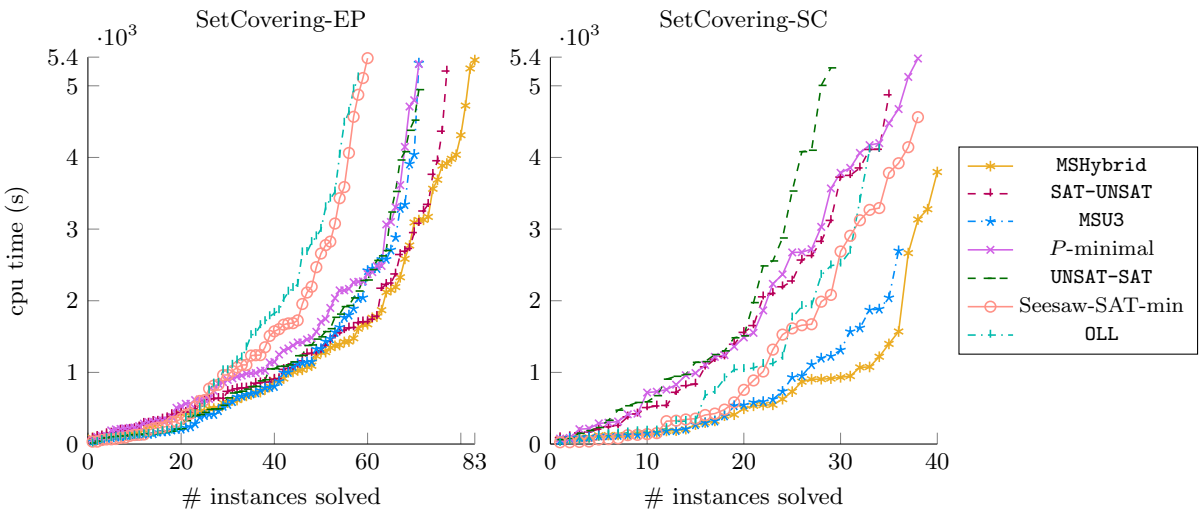

Figure 5.2: Runtime comparison between variants of BiOptSat and competing approaches for bi-objective set covering problem; enumeration of a single representative solution per Pareto point.

sets of bi-objective set covering instances. ParetoMCS with and without stratification, as well as Seesaw-SAT did not manage to solve any of the set covering instances. The best-performing variant of BiOptSat is again `MSHybrid`, considerably outperforming P-minimal and Seesaw-SAT-min: P-minimal solved 71 (respectively 38) fixed element probability (respectively set cardinality) instances, Seesaw-SAT-min solved 60 (respectively 38), whereas `MSHybrid` solved 83 (respectively 40) instances. For this application, not all variants of BiOptSat outperformed P-minimal: `MSU3` and `OLL` were outperformed for both instance variants while `SAT-UNSAT` and `UNSAT-SAT` were only outperformed for the instances generated with fixed set cardinality. The good performance of Seesaw-SAT-min on these instances might be explained by the fact that the weights for one of the objectives in Seesaw are handled by an integer linear programming solver, rather than in propositional logic.

The numbers of solved instances for all approaches are summarized in Table 5.1. The best-performing approach for each benchmark is highlighted in bold. `MSHybrid` is the best-performing BiOptSat variant overall, outperforming P-minimal in all cases. For more details, Figure 5.3 (left) shows a per-instance runtime comparison between `MSHybrid` and P-minimal. We note that P-minimal did not uniquely solve any instance. 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 full 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 being more structured.

Table 5.1: Solved instances by approach and benchmark domain. Results for both tasks, finding a single solution per Pareto point (single) and enumerating all Pareto-optimal solutions (all).

	Decision Rules		SetCovering-EP		SetCovering-SC	
Algorithm	single	all	single	all	single	all
`SAT-UNSAT`	**223**	**215**	77	75	35	35
`UNSAT-SAT`	**223**	**215**	71	71	29	29
`MSU3`	**223**	**215**	71	70	36	36
`OLL`	222	213	58	58	34	34
`MSHybrid`	**223**	**215**	**83**	**81**	**40**	**40**
P-minimal	219	213	71	68	38	26
Seesaw	123	–	–	–	–	–
Seesaw-SAT	42	–	0	–	0	–
Seesaw-SAT-min	136	–	60	–	38	–
ParetoMCS	34	–	0	–	0	–
ParetoMCS-strat	32	–	0	–	0	–

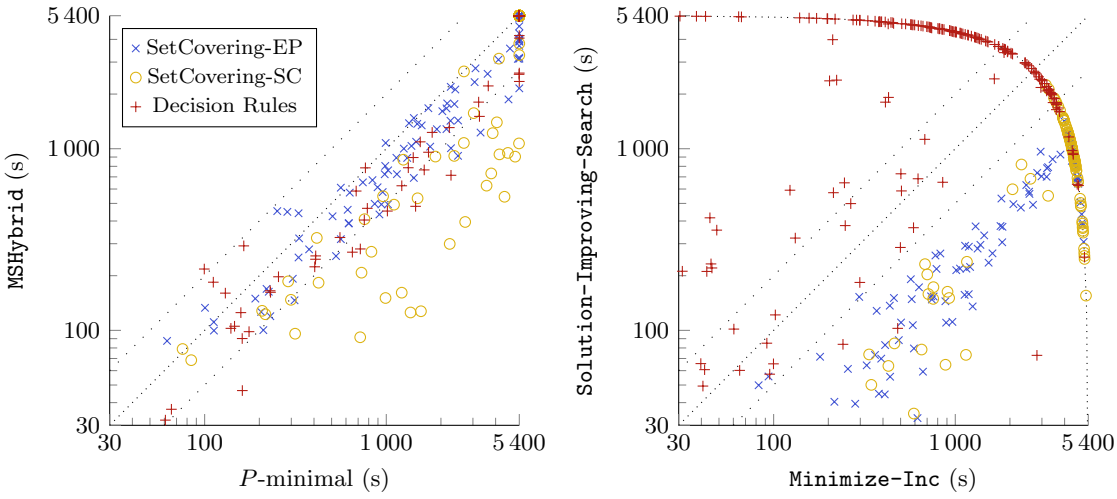

Figure 5.3: Left: Runtime comparison between *P*-minimal and BiOptSat in the `MSHybrid` variant. Right: Time (in seconds) spent in the `Minimize-Inc` and `Solution-Improving-Search` subroutines for BiOptSat in the `MSHybrid` variant. Both plots are for enumeration of a single representative per Pareto point.

Figure 5.3 (right) shows how much time is spent in the `Minimize-Inc` compared to the `Solution-Improving-Search` subroutine for the `MSHybrid` variant of BiOptSat. The arc that forms in this plot is the time limit border, which is where the sum of both axes equals the 5 400-second per-instance time limit. It can be seen that for the decision rule instances, in general more time is spent in the `Solution-Improving-Search` subroutine, while for the set covering instances, more time is spent in the `Minimize-Inc` subroutine. This indicates that the answer to the question of which step of the lexicographic minimization is harder depends notably on the instance. Furthermore, we note that the two objectives for the set covering instances are both generated randomly with the same procedure, therefore swapping the objectives does not change the instances. For the decision rule instances, the objectives are structurally different, but—as mentioned previously—preliminary experiments have shown that this configuration of the objectives leads to better performance.

5.3.2 Enumerating All Pareto-Optimal Solutions

Next we compare the performance for learning all Pareto-optimal interpretable decision rules. Figure 5.4 shows the results for this task, presented in the same way as in Figure 5.1. For this task, the BiOptSat variants `SAT-UNSAT`, `UNSAT-SAT`, `MSU3` and `MSHybrid` all performed the best, solving 215 instances each. With this, each of them outperforms P-minimal, which solved 213 instances. The `OLL` variant of BiOptSat solved the same number of instances as P-minimal. Note that Seesaw and ParetoMCS cannot be used for enumerating all Pareto-optimal solutions and are therefore excluded for these experiments.

Turning to the same comparison for bi-objective set covering, as seen from Figure 5.5, `MSHybrid` is the best-performing approach also here. For the instances with fixed set cardinality, all BiOptSat variants solved at least as many instances as P-minimal, for the instances with fixed element probability only `OLL` performed worse than P-minimal. The best-performing variant, `MSHybrid`, solved 81 (respectively 40) of the fixed element probability (respectively set cardinality) instances while P-minimal solved 68 (respectively 26).

Table 5.1 summarizes the number of solved instances for all approaches applicable to this task. It can be seen that `MSHybrid` is the best-performing approach for this task as well. A per-instance runtime comparison between `MSHybrid` and P-minimal for the task of enumerating all Pareto-optimal solutions is shown in Figure 5.6 (left). P-minimal did also not uniquely solve any instance for enumerating all Pareto-optimal solutions. Furthermore,

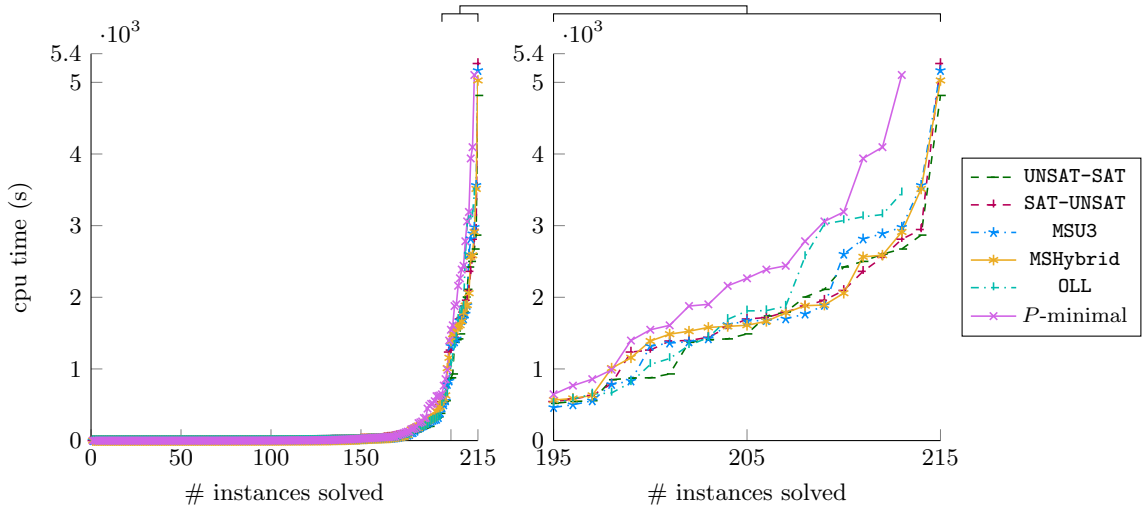

Figure 5.4: Runtime comparison between variants of BiOptSat and competing approaches for learning interpretable decision rules; enumeration of all Pareto-optimal solutions. The plot on the right shows a magnification for comparing the best-performing approaches.

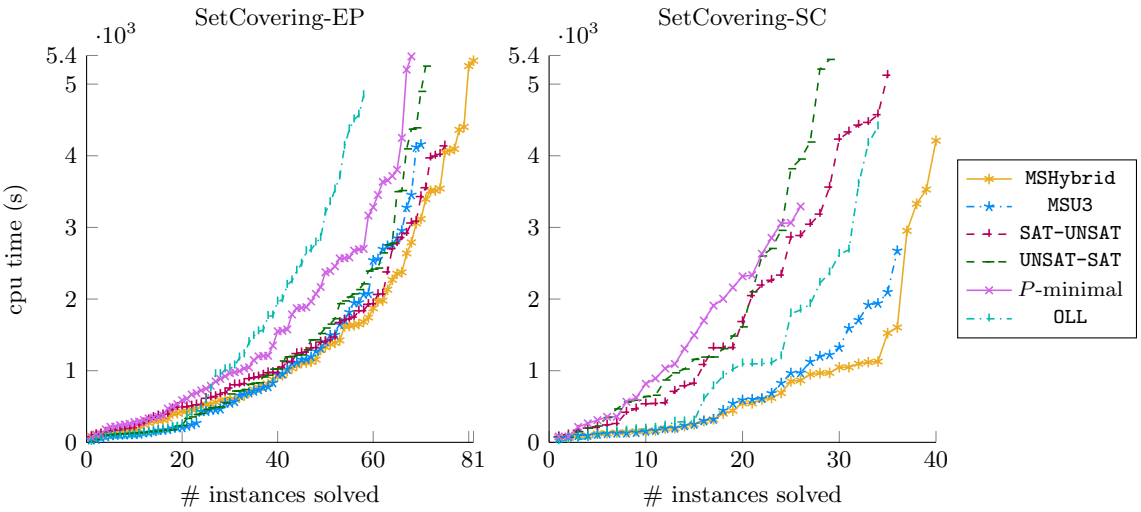

Figure 5.5: Runtime comparison between variants of BiOptSat and competing approaches for bi-objective set covering problem; enumeration of all Pareto-optimal solutions.

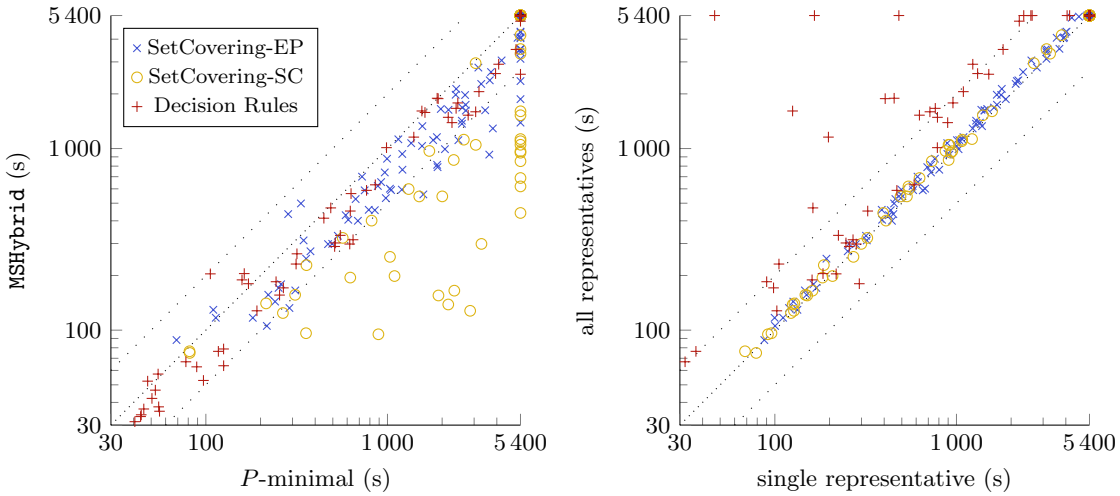

Figure 5.6: Left: Runtime comparison between P-minimal and BiOptSat in the `MSHybrid` variant; enumeration of all Pareto-optimal solutions. Right: Runtime comparison between enumerating a single representative vs. all solutions per Pareto point with `MSHybrid`.

`MSHybrid` was outperformed by P-minimal on 71 instances while `MSHybrid` outperformed P-minimal on 257 instances.

Comparing the number of solved instances in Table 5.1, we can see that the performance difference between BiOptSat and P-minimal is greater when enumerating all Pareto-optimal solutions. Furthermore, Figure 5.6 (right) shows a runtime comparison between enumerating a single representative solution per Pareto point and enumerating all Pareto-optimal solutions with `MSHybrid`. Overall, BiOptSat scales well also for enumerating all Pareto-optimal solutions, although there is an overhead when the number of solutions required to be enumerated grows significantly; this is the case for learning interpretable decision rules, 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 (of few) solutions per Pareto point. The observation that there are fewer solutions per Pareto point for the set covering instances can be intuitively explained by the weighted objectives which make it significantly less likely that two distinct solutions have identical objective function values.

5.3.3 Impact of Refinements

Finally, we evaluated the impact of the refinements proposed in Section 4.3 on the runtime efficiency of the best-performing approach, `MSHybrid`. As the first refinement considered, we evaluate the impact of lazily building the totalizer for the decreasing objective. Figure 5.7

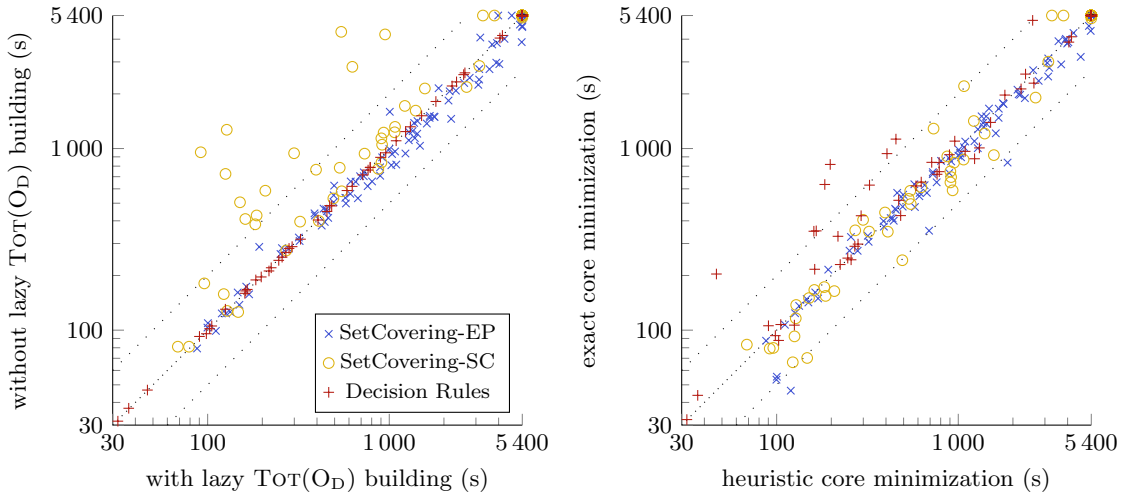

Figure 5.7: Instance runtime comparisons for the two refinements lazily building the totalizer for the decreasing objective (left) and exact core minimization (right).

(left) shows a runtime comparison between `MSHybrid` with and without lazy building of $\textsc{Tot}(O_D)$. It can be seen that for learning interpretable decision rules, this refinement has no evident impact. This is to be expected since for these benchmarks, the literals from O_D do not appear in O_I, so $\textsc{Tot}(O_D)$ cannot be lazily built. For the set covering instances with fixed element probability, the impact of this refinement is small and tends to be slightly negative for most instances. However, for fixed set cardinality set covering, we see a strong positive effect.

Next, we detail the impact of different core minimization strategies. By default, `MSHybrid` employs heuristic core minimization. In Figure 5.7 (right), the performance of this configuration is compared to using exact core minimization instead. Heuristic core minimization appears to have a positive effect for the task of learning interpretable decision rules as well as for harder set covering instances. However, the effect is smaller than that of lazily building $\textsc{Tot}(O_D)$.

Figure 5.8 (left) shows the effect of blocking dominated solutions in the `SAT-UNSAT` phase of `MSHybrid`. The impact of this refinement is negligible on all benchmarks, although there are three instances of the set covering benchmarks with fixed element probability that were only solved when not blocking dominated solutions, giving this configuration a slight advantage. On the decision rule instances, while slight positive effects of the refinement can be seen, the effect is not strong enough to enable solving additional instances.

As the last refinement considered, we consider adding a disjoint core extraction phase to the `MSU3` phase of `MSHybrid`. Figure 5.8 shows the impact of adding this refinement

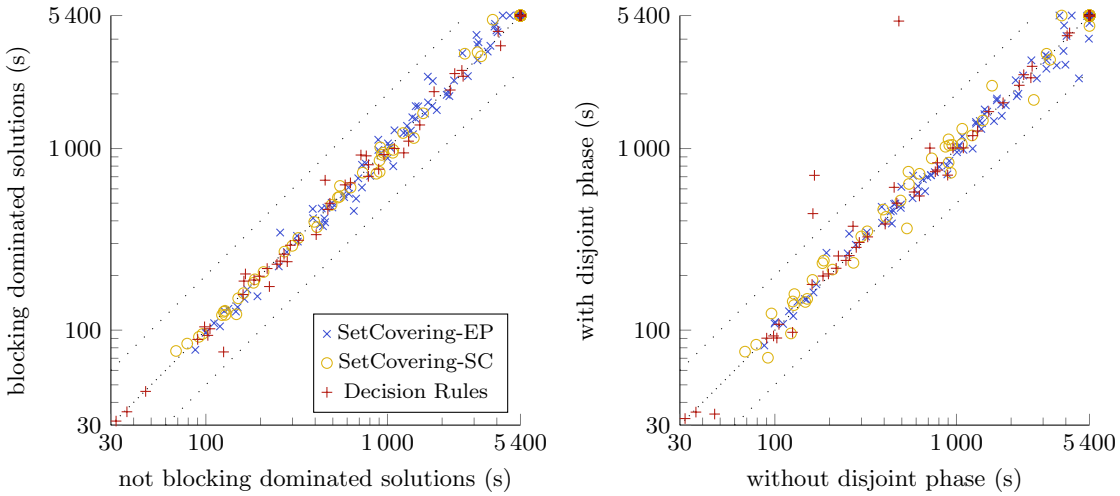

Figure 5.8: Instance runtime comparisons for the two refinements blocking dominated solutions (left) and disjoint core extraction (right).

to the default configuration of `MSHybrid`. It can be seen that adding the disjoint phase does not result in a clear positive or negative effect. However, the impact per instance varies a lot more compared to the impact of blocking dominated solutions. The only clear negative impact can be seen on three instances for learning interpretable decision rules. The strongest outlier is an instance that was solved by the configuration without a disjoint phase in under 500 seconds but only barely under the given time limit of 5 400 seconds by the configuration with a disjoint phase. At present, it is unclear to us why for this specific this refinement has such a strong negative effect.

6 Conclusions

We presented BiOptSat, an algorithm for exact bi-objective optimization under Pareto optimality. The structured search of BiOptSat builds on algorithms for maximum satisfiability (MaxSAT) and makes incremental use of a solver for propositional satisfiability (SAT). BiOptSat can be used to solve three tasks for $\mathcal{NP}$-hard bi-objective optimization problems encoded in propositional logic: finding a single Pareto-optimal solution, finding one representative solution for each Pareto point, and enumerating all Pareto-optimal solutions.

We presented four variants of BiOptSat that are based on algorithms proposed for MaxSAT (`SAT-UNSAT`, `UNSAT-SAT`, `MSU3`, and `OLL`), as well as `MSHybrid`, a novel hybrid between `MSU3` and `SAT-UNSAT`. The main difference between the BiOptSat variants and their MaxSAT inspirations is that an additional constraint over the second objective needs to be enforced during the optimization. An open-source implementation of all five variants and two previously proposed SAT-based approaches is available. We compared BiOptSat to three previously-proposed approaches: P-minimal [28], ParetoMCS [30], and Seesaw [29]. We empirically evaluated the five variants of BiOptSat, comparing them to the three competitors, on two benchmark domains: learning interpretable decision rules [11] and bi-objective set covering. Additionally, we evaluated the algorithms for the two tasks of finding one representative solution for each Pareto point, and of enumerating all Pareto-optimal solutions. In the empirical evaluation we found that the `MSHybrid` variant did not only outperform all other four variants of BiOptSat but also the three competitors. `MSHybrid` achieves improved performance by combining advantages from the MSU3 and the SAT-UNSAT MaxSAT algorithms. When enumerating all Pareto-optimal solutions, the advantage of BiOptSat over its competitors is slightly more pronounced. The good performance of BiOptSat is in part due to the incremental use of the SAT solver, but—since P-minimal also makes fully incremental use of a SAT solver—more important for the good efficiency is the structured nature of the search of BiOptSat.

We also evaluated refinements to the best-performing variant, `MSHybrid`. The refinements found most impactful were lazily building the totalizer for the decreasing objective when the two objectives share literals, and heuristic core minimization. Other refinements that

did not show a significant impact on performance are blocking of solutions dominated by candidates found during the search and the addition of a disjoint core extraction phase.

Going beyond the work presented in this thesis, it is likely that BiOptSat could be further improved and evaluated. For example, the way weighted objectives are handled in BiOptSat at the moment can be considered relatively naive. As shown empirically, the performance of BiOptSat on weighted instances is already competitive. However, applying more sophisticated ways of handling weights promises even better performance. Additionally, a better understanding for what objective should be chosen as increasing to achieve the best performance remains an interesting direction for further research.

Bibliography

[1] S. Arora and B. Barak. *NP and NP completeness*. In *Computational Complexity—A Modern Approach*. Cambridge University Press, 2009. Chapter 2, pages 38–67. ISBN: 978-0-521-42426-4. URL: http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264.

[2] M. Stojadinovic. Air traffic controller shift scheduling by reduction to CSP, SAT and SAT-related problems. In B. 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 886–902. Springer, 2014. DOI: 10.1007/978-3-319-10428-7_63.

[3] M. Bofill, M. Garcia, J. Suy and M. Villaret. MaxSAT-based scheduling of B2B meetings. In L. Michel, editor, *Integration of AI and OR Techniques in Constraint Programming—12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18–22, 2015, Proceedings*, volume 9075 of *Lecture Notes in Computer Science*, pages 65–73. Springer, 2015. DOI: 10.1007/978-3-319-18008-3_5.

[4] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987. DOI: 10.1287/opre.35.2.254.

[5] D. E. Akyol and G. M. Bayhan. A review on evolution of production scheduling with neural networks. *Computers & Industrial Engineering*, 53(1):95–122, 2007. DOI: 10.1016/j.cie.2007.04.006.

[6] L. G. Papageorgiou. Supply chain optimisation for the process industries: Advances and opportunities. *Computer & Chemical Engineering*, 33(12):1931–1938, 2009. DOI: 10.1016/j.compchemeng.2009.06.014.

[7] D. Bertsimas, G. Lulli and A. R. Odoni. An integer optimization approach to large-scale air traffic flow management. *Operations Research*, 59(1):211–227, 2011. DOI: 10.1287/opre.1100.0899.

[8] A. Richards and J. How. Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In *Proceedings of the 2002 American*

Control Conference (IEEE Cat. No.CH37301), volume 3, 1936–1941 vol.3, 2002. DOI: 10.1109/ACC.2002.1023918.

[9] T. Dao, K. Duong and C. Vrain. Constrained clustering by constraint programming. *Artificial Intelligence*, 244:70–94, 2017. DOI: 10.1016/j.artint.2015.05.006.

[10] I. Davidson, S. S. Ravi and L. Shamis. A SAT-based framework for efficient constrained clustering. In *Proceedings of the SIAM International Conference on Data Mining, SDM 2010, April 29 – May 1, 2010, Columbus, Ohio, USA*, pages 94–105. SIAM, 2010. DOI: 10.1137/1.9781611972801.9.

[11] D. Malioutov and K. S. Meel. MLIC: A MaxSAT-based framework for learning interpretable classification rules. In J. 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.

[12] N. Narodytska, A. Ignatiev, F. Pereira and J. Marques-Silva. Learning optimal decision trees with SAT. In J. 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.

[13] H. Hu, M. Siala, E. Hebrard and M. Huguet. Learning optimal decision trees with MaxSAT and its integration in AdaBoost. In C. 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.

[14] J. Yu, A. Ignatiev, P. J. Stuckey and P. L. Bodic. Computing optimal decision sets with SAT. In H. 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.

[15] E. Demirovic and P. J. Stuckey. Optimal decision trees for nonlinear metrics. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2–9, 2021*, pages 3733–3741. AAAI Press, 2021. URL: https://ojs.aaai.org/index.php/AAAI/article/view/16490.

[16] P. Shati, E. Cohen and S. A. McIlraith. SAT-based approach for learning optimal decision trees with non-binary features. In L. 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*, 50:1–50:16. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2021. DOI: 10.4230/LIPIcs.CP.2021.50.

[17] A. Ignatiev, F. Pereira, N. Narodytska and J. Marques-Silva. A sat-based approach to learn explainable decision sets. In D. Galmiche, S. Schulz and R. 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.

[18] W. Michiels, E. H. L. Aarts and J. H. M. Korst. *Theoretical aspects of local search*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2007. ISBN: 978-3-540-35853-4. DOI: 10.1007/978-3-540-35854-1.

[19] D. Dasgupta and Z. Michalewicz. *Evolutionary algorithms in engineering applications*. Springer, 1997. ISBN: 978-3-540-62021-1.

[20] R. Storn and K. V. Price. Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997. DOI: 10.1023/A:1008202821328.

[21] D.-S. Chen, R. G. Batson and Y. Dang. Introduction. In *Applied Integer Programming: Modeling and Solution*, chapter 1, pages 3–20. John Wiley & Sons, December 2010. ISBN: 0470373067. URL: https://www.ebook.de/de/product/9338058/chen_batson_dang_applied_integer_programming.html.

[22] B. Korte and J. Vygen. *Integer programming*. In *Combinatorial Optimization: Theory and Algorithms*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2018. Chapter 5, pages 103–132. ISBN: 978-3-662-56039-6. DOI: 10.1007/978-3-662-56039-6_5.

[23] J. Franco and J. Martin. A history of satisfiability. In *Handbook of Satisfiability*. Volume 336, FAIA, chapter 1, pages 3–74. IOS Press, 2021. DOI: 10.3233/FAIA200984.

[24] J. Marques-Silva, I. Lynce and S. Malik. Conflict-driven clause learning SAT solvers. In *Handbook of Satisfiability*. Volume 336, FAIA, chapter 4, pages 133–182. IOS Press, 2021. DOI: 10.3233/FAIA200987.

[25] F. Bacchus, M. Järvisalo and R. Martins. Maximum satisfiability. In *Handbook of Satisfiability*. Volume 336, FAIA, chapter 24, pages 929–991. IOS Press, 2021. DOI: 10.3233/FAIA201008.

[26] M. Ehrgott. Efficiency and nondominance. In *Multicriteria Optimization (2. ed.)* Chapter 2, pages 22–64. Springer, 2005. ISBN: 978-3-540-21398-7. DOI: 10.1007/3-540-27659-9_2.

[27] M. D. Santis, G. Eichfelder, J. Niebling and S. Rocktäschel. Solving multiobjective mixed integer convex optimization problems. *SIAM Journal on Optimization*, 30(4):3122–3145, 2020. DOI: 10.1137/19M1264709.

[28] T. Soh, M. Banbara, N. Tamura and D. L. Berre. Solving multiobjective discrete optimization problems with propositional minimal model generation. In J. C. 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.

[29] M. Janota, A. Morgado, J. F. Santos and V. M. Manquinho. The Seesaw algorithm: Function optimization using implicit hitting sets. In L. 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*, 31:1–31:16. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2021. DOI: 10.4230/LIPIcs.CP.2021.31.

[30] M. Terra-Neves, I. Lynce and V. M. Manquinho. Multi-objective optimization through pareto minimal correction subsets. In J. 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.

[31] A. Ignatiev, J. Marques-Silva, N. Narodytska and P. J. Stuckey. Reasoning-based learning of interpretable ML models. In Z. 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.

[32] J. Yu, A. Ignatiev, P. L. Bodic and P. J. Stuckey. Optimal decision lists using SAT. *Computing Research Repository*, abs/2010.09919, 2020. arXiv: 2010.09919.

[33] A. Ignatiev, E. Lam, P. J. Stuckey and J. Marques-Silva. A scalable two stage approach to computing optimal decision sets. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2–9, 2021*, pages 3806–3814. AAAI Press, 2021. URL: https://ojs.aaai.org/index.php/AAAI/article/view/16498.

[34] A. J. Silvério, R. S. Couto, M. E. M. Campista and L. H. M. K. Costa. A bi-objective optimization model for segment routing traffic engineering. en. *Ann. Telecommun.*, February 2022. DOI: 10.1007/s12243-022-00907-w.

[35] T. Pinto-Varela, A. P. F. D. Barbosa-Póvoa and A. Q. Novais. Bi-objective optimization approach to the design and planning of supply chains: Economic versus environmental performances. *Computer & Chemical Engineering*, 35(8):1454–1468, 2011. DOI: 10.1016/j.compchemeng.2011.03.009.

[36] C. P. S. Tautenhain, A. P. F. D. Barbosa-Póvoa and M. C. V. Nascimento. A multi-objective matheuristic for designing and planning sustainable supply chains. *Computers & Industrial Engineering*, 135:1203–1223, 2019. DOI: 10.1016/j.cie.2018.12.062.

[37] R. Marler and J. 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.

[38] J. Argelich, I. Lynce and J. P. M. Silva. On solving boolean multilevel optimization problems. In C. 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.

[39] J. Marques-Silva, J. Argelich, A. Graça and I. 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.

[40] D. L. Berre and A. Parrain. The Sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(2–3):59–6, 2010. DOI: 10.3233/sat190075.

[41] N. Eén and N. Sörensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1–4):1–26, 2006. DOI: 10.3233/sat190014.

[42] J. Marques-Silva and J. Planes. On using unsatisfiability for solving maximum satisfiability. *Computing Research Repository*, abs/0712.1097, 2007. arXiv: 0712.1097.

[43] C. Ansótegui, M. L. Bonet and J. Levy. Solving (weighted) partial MaxSAT through satisfiability testing. In O. 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.

[44] A. Morgado, C. Dodaro and J. Marques-Silva. Core-guided MaxSAT with soft cardinality constraints. In B. 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.

[45] A. Ignatiev, A. Morgado and J. Marques-Silva. RC2: an efficient MaxSAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 11(1):53–64, 2019. DOI: 10.3233/SAT190116.

[46] Z. Fu and S. Malik. On solving the partial MAX-SAT problem. In A. Biere and C. 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.

[47] C. Jabs, J. Berg, A. Niskanen and M. Järvisalo. MaxSAT-based bi-objective boolean optimization. In K. S. Meel and O. Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing (SAT 2022), Haifa, Israel, August 2–5, 2022*, volume 236 of *LIPIcs*. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2022.

[48] S. Prestwich. Cnf encodings. In *Handbook of Satisfiability*. Volume 336, FAIA, chapter 2, pages 75–100. IOS Press, 2021. DOI: 10.3233/FAIA200985.

[49] G. S. Tseitin. On the complexity of derivation in propositional calculus. In *Automation of Reasoning*, pages 466–483. Springer Berlin Heidelberg, 1983. DOI: 10.1007/978-3-642-81955-1_28.

[50] S. A. Cook. The complexity of theorem-proving procedures. In M. A. Harrison, R. B. Banerji and J. D. Ullman, editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3–5, 1971, Shaker Heights, Ohio, USA*, pages 151–158. ACM, 1971. DOI: 10.1145/800157.805047.

[51] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. DOI: 10.1007/978-1-4684-2001-2_9.

[52] N. Eén and N. 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.

[53] O. Bailleux and Y. Boufkhad. Efficient CNF encoding of boolean cardinality constraints. In F. 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.

[54] C. Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In P. van Beek, editor, *Principles and Practice of Constraint Programming—CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1–5, 2005, Proceedings*, volume 3709 of *Lecture Notes in Computer Science*, pages 827–831. Springer, 2005. DOI: 10.1007/11564751_73.

[55] R. Martins, S. Joshi, V. M. Manquinho and I. Lynce. Incremental cardinality constraints for MaxSAT. In B. 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.

[56] J. Davies and F. Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In J. H. 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.

[57] C. Li, Z. Xu, J. Coll, F. Manyà, D. Habet and K. He. Combining clause learning and branch and bound for MaxSAT. In L. 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*, 38:1–38:18. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2021. DOI: 10.4230/LIPIcs.CP.2021.38.

[58] C. M. Le and F. Manyà. MaxSAT, hard and soft constraints. In *Handbook of Satisfiability*. Volume 336, FAIA, chapter 23, pages 903–927. IOS Press, 2021. DOI: 10.3233/FAIA201007.

[59] T. Alsinet, F. Manyà and J. Planes. Improved exact solvers for weighted Max-SAT. In F. Bacchus and T. Walsh, editors, *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19–23, 2005, Proceedings*, volume 3569 of *Lecture Notes in Computer Science*, pages 371–377. Springer, 2005. DOI: 10.1007/11499107_27.

[60] J. Planes. Improved branch and bound algorithms for Max-2-SAT and weighted Max-2-SAT. In F. 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*, page 991. Springer, 2003. DOI: 10.1007/978-3-540-45193-8_115.

[61] T. Alsinet, F. Manyà and J. Planes. An efficient solver for weighted Max-SAT. *Journal of Global Optimization*, 41(1):61–73, 2008. DOI: 10.1007/s10898-007-9166-9.

[62] F. Heras, J. Larrosa and A. Oliveras. MiniMaxSAT: An efficient weighted Max-SAT solver. *Journal of Artificial Intelligence Research*, 31:1–32, 2008. DOI: 10.1613/jair.2347.

[63] C.-M. Li, Z. Xu, J. Coll, F. Manyà, D. Habet and K. He. Boosting branch-and-bound MaxSAT solvers with clause learning. *AI Communications*, Preprint(Preprint):1–21, 2021. ISSN: 1875-8452. DOI: 10.3233/AIC-210178.

[64] J. Davies and F. Bacchus. Postponing optimization to speed up MAXSAT solving. In C. 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.

[65] J. Davies and F. Bacchus. Exploiting the power of MIP solvers in MAXSAT. In M. Järvisalo and A. V. 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.

[66] J. Berg, F. Bacchus and A. Poole. Abstract cores in implicit hitting set MaxSat solving. In L. Pulina and M. Seidl, editors, *Theory and Applications of Satisfiability Testing—SAT 2020—23rd International Conference, Alghero, Italy, July 3–10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 277–294. Springer, 2020. DOI: 10.1007/978-3-030-51825-7_20.

[67] H. Xu, R. A. Rutenbar and K. A. Sakallah. sub-SAT: A formulation for relaxed boolean satisfiability with applications in routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(6):814–820, 2003. DOI: 10.1109/TCAD.2003.811450.

[68] B. Andres, B. Kaufmann, O. Matheis and T. Schaub. Unsatisfiability-based optimization in clasp. In A. Dovier and V. S. 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.

[69] M. Ehrgott. Introduction. In *Multicriteria Optimization (2. ed.)* Chapter 1, pages 1–21. Springer, 2005. ISBN: 978-3-540-21398-7. DOI: 10.1007/3-540-27659-9_1.

[70] M. Ehrgott. The weighted sum method and related topics. In *Multicriteria Optimization (2. ed.)* Chapter 3, pages 65–95. Springer, 2005. ISBN: 978-3-540-21398-7. DOI: 10.1007/3-540-27659-9_3.

[71] H. Isermann. The enumeration of all efficient solutions for a linear multiple-objective transportation problem. *Naval Research Logistics Quarterly*, 26(1):123–139, March 1979. DOI: 10.1002/nav.3800260112.

[72] R. Hartert and P. Schaus. A support-based algorithm for the bi-objective pareto constraint. In C. E. Brodley and P. 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.

[73] M. Ehrgott. Other definitions of optimality—nonscalarizing methods. In *Multicriteria Optimization (2. ed.)* Chapter 5, pages 127–149. Springer, 2005. ISBN: 978-3-540-21398-7. DOI: 10.1007/3-540-27659-9_5.

[74] M. Koshimura, H. Nabeshima, H. Fujita and R. Hasegawa. Minimal model generation with respect to an atom set. In N. Peltier and V. 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.

[75] F. Rossi, P. van Beek and T. Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006. ISBN: 978-0-444-52726-4. URL: https://www.sciencedirect.com/science/bookseries/15746526/2.

[76] N. Tamura, M. Banbara and T. Soh. Compiling pseudo-boolean constraints to SAT with order encoding. In *25th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2013, Herndon, VA, USA, November 4–6, 2013*, pages 1020–1027. IEEE Computer Society, 2013. DOI: 10.1109/ICTAI.2013.153.

[77] M. Terra-Neves, I. Lynce and V. M. Manquinho. Enhancing constraint-based multi-objective combinatorial optimization. In S. A. McIlraith and K. 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.

[78] M. Terra-Neves, I. Lynce and V. M. Manquinho. Stratification for constraint-based multi-objective combinatorial optimization. In J. 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.

[79] J. Bendík and I. Cerna. Rotation based MSS/MCS enumeration. In E. Albert and L. 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.

[80] A. Morgado, M. H. Liffiton and J. Marques-Silva. Maxsat-based MCS enumeration. In A. Biere, A. Nahir and T. 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.

[81] A. Previti, C. Mencía, M. Järvisalo and J. Marques-Silva. Improving MCS enumeration via caching. In S. Gaspers and T. 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.

[82] K. Chandrasekaran, R. M. Karp, E. Moreno-Centeno and S. S. Vempala. Algorithms for implicit hitting set problems. In D. Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23–25, 2011*, pages 614–629. SIAM, 2011. DOI: 10.1137/1.9781611973082.48.

[83] E. Moreno-Centeno and R. M. Karp. The implicit hitting set approach to solve combinatorial optimization problems with an application to multigenome alignment. *Operations Research*, 61(2):453–468, 2013. DOI: 10.1287/opre.1120.1139.

[84] M. Parker and J. Ryan. Finding the minimum weight IIS cover of an infeasible system of linear inequalities. *Annals of Mathematics and Artificial Intelligence*, 17(1-2):107–126, 1996. DOI: 10.1007/BF02284626.

[85] J. A. Reggia, D. S. Nau and P. Y. Wang. Diagnostic expert systems based on a set covering model. *International Journal of Man-Machine Studies*, 19(5):437–460, 1983. DOI: 10.1016/S0020-7373(83)80065-0.

[86] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987. DOI: 10.1016/0004-3702(87)90062-2.

[87] A. Ignatiev, A. Morgado and J. Marques-Silva. Propositional abduction with implicit hitting sets. In G. A. Kaminka, M. Fox, P. Bouquet, E. Hüllermeier, V. Dignum, F. Dignum and F. van Harmelen, editors, *ECAI 2016—22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands—Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 1327–1335. IOS Press, 2016. DOI: 10.3233/978-1-61499-672-9-1327.

[88] A. Ignatiev, A. Previti, M. H. Liffiton and J. Marques-Silva. Smallest MUS extraction with minimal hitting set dualization. In G. 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.

[89] P. Saikko, J. P. Wallner and M. Järvisalo. Implicit hitting set algorithms for reasoning beyond NP. In C. Baral, J. P. Delgrande and F. 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.

[90] M. Ehrgott and X. Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spectrum*, 22(4):425–460, 2000. DOI: 10.1007/s002910000046.

[91] T. Paxian, P. Raiola and B. Becker. On preprocessing for weighted MaxSAT. In F. Henglein, S. Shoham and Y. 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.

[92] C. Ansótegui, M. L. Bonet, J. Gabàs and J. Levy. Improving sat-based weighted MaxSAT solvers. In M. 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.

[93] L. N. V. Wassenhove and L. F. Gelders. Solving a bicriterion scheduling problem. *European Journal of Operational Research*, 4(1):42–48, January 1980. DOI: 10.1016/0377-2217(80)90038-7.

[94] M. Gavanelli. An algorithm for multi-criteria optimization in csps. In F. van Harmelen, editor, *Proceedings of the 15th European Conference on Artificial Intelligence, ECAI'2002, Lyon, France, July 2002*, pages 136–140. IOS Press, 2002.

[95] P. Schaus and R. Hartert. Multi-objective large neighborhood search. In C. 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 611–627. Springer, 2013. DOI: 10.1007/978-3-642-40627-0_46.

[96] S. Bouveret and M. Lemaître. Computing leximin-optimal solutions in constraint networks. *Artificial Intelligence*, 173(2):343–364, 2009. DOI: 10.1016/j.artint.2008.10.010.

[97] M. Ehrgott. Introduction to multicriteria linear programming. In *Multicriteria Optimization (2. ed.)* Chapter 6, pages 151–170. Springer, 2005. ISBN: 978-3-540-21398-7. DOI: 10.1007/3-540-27659-9_6.

[98] L. M. Rasmussen. Zero—one programming with multiple criteria. *European Journal of Operational Research*, 26(1):83–95, 1986. ISSN: 0377-2217. URL: https://www.sciencedirect.com/science/article/pii/037722178690161X.

[99] M. J. Alves and J. 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.

[100] M. Ehrgott. A multiobjective simplex method. In *Multicriteria Optimization (2. ed.)* Chapter 7, pages 171–196. Springer, 2005. ISBN: 978-3-540-21398-7. DOI: 10.1007/3-540-27659-9_7.

[101] J. P. Evans and R. E. Steuer. A revised simplex method for linear multiple objective programs. *Mathematical Programming*, 5(1):54–72, 1973. DOI: 10.1007/BF01580111.

[102] N. Adelgren and A. Gupte. Branch-and-bound for biobjective mixed-integer linear programming. *INFORMS Journal on Computing*, October 2021. DOI: 10.1287/ijoc.2021.1092.

[103] E. Sun. On optimization over the efficient set of a multiple objective linear programming problem. *Journal of Optimization Theory and Applications*, 172(1):236–246, 2017. DOI: 10.1007/s10957-016-1030-y.

[104] K. Lu, S. Mizuno and J. 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.

[105] R. M. Soland. Multicriteria optimization: A general characterization of efficient solutions. *Decision Sciences*, 10(1):26–38, January 1979. DOI: 10.1111/j.1540-5915.1979.tb00004.x.

[106] N. Saini and S. Saha. Multi-objective optimization techniques: A survey of the state-of-the-art and applications. *The European Physical Journal Special Topics*, 230(10):2319–2335, September 2021. DOI: 10.1140/epjs/s11734-021-00206-w.

[107] J. Dubois-Lacoste, M. López-Ibáñez and T. Stützle. Pareto local search algorithms for anytime bi-objective optimization. In J. Hao and M. Middendorf, editors, *Evolutionary Computation in Combinatorial Optimization—12th European Conference, EvoCOP 2012, Málaga, Spain, April 11–13, 2012. Proceedings*, volume 7245 of *Lecture Notes in Computer Science*, pages 206–217. Springer, 2012. DOI: 10.1007/978-3-642-29124-1_18.

[108] J. Dubois-Lacoste, M. López-Ibáñez and T. Stützle. Anytime pareto local search. *European Journal of Operational Research*, 243(2):369–385, 2015. DOI: 10.1016/j.ejor.2014.10.062.

[109] A. Jaszkiewicz. Many-objective pareto local search. *European Journal of Opererational Research*, 271(3):1001–1013, 2018. DOI: 10.1016/j.ejor.2018.06.009.

[110] E. Zitzler and L. Thiele. Multiobjective optimization using evolutionary algorithms—A comparative case study. In A. E. Eiben, T. Bäck, M. Schoenauer and H. Schwefel, editors, *Parallel Problem Solving from Nature—PPSN V, 5th International Conference, Amsterdam, The Netherlands, September 27–30, 1998, Proceedings*, volume 1498 of *Lecture Notes in Computer Science*, pages 292–304. Springer, 1998. DOI: 10.1007/BFb0056872.

[111] S. Kirkpatrick, D. G. Jr. and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. DOI: 10.1126/science.220.4598.671.

[112] S. Bandyopadhyay, S. Saha, U. Maulik and K. Deb. A simulated annealing-based multiobjective optimization algorithm: AMOSA. *IEEE Transactions on Evolutionary Computation*, 12(3):269–283, 2008. DOI: 10.1109/TEVC.2007.900837.

[113] R. Sengupta and S. Saha. Reference point based archived many objective simulated annealing. *Information Sciences*, 467:725–749, 2018. DOI: 10.1016/j.ins.2018.05.013.

[114] K. Deb, S. Agrawal, A. Pratap and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002. DOI: 10.1109/4235.996017.

[115] J. Berg, E. Demirovic and P. J. Stuckey. Core-boosted linear search for incomplete MaxSAT. In L. Rousseau and K. Stergiou, editors, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research—16th International Conference, CPAIOR 2019, Thessaloniki, Greece, June 4–7, 2019, Proceedings*, volume 11494 of *Lecture Notes in Computer Science*, pages 39–56. Springer, 2019. DOI: 10.1007/978-3-030-19212-9_3.

[116] C. Ansótegui, M. L. Bonet, J. Gabàs and J. Levy. Improving WPM2 for (weighted) partial MaxSAT. In C. 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.

[117] R. Martins, V. M. Manquinho and I. Lynce. Open-WBO: A modular maxsat solver. In C. Sinz and U. 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.

[118] N. Froleyks, M. Heule, M. Iser, M. Järvisalo and M. Suda. SAT competition 2020. *Artificial Intelligence*, 301:103572, 2021. DOI: 10.1016/j.artint.2021.103572.

[119] A. Biere, K. Fazekas, M. Fleury and M. Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In T. Balyo, N. Froleyks, M. Heule, M. Iser, M. Järvisalo and M. 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.

[120] D. Dua and C. Graff. UCI machine learning repository, 2021. URL: http://archive.ics.uci.edu/ml.

Appendix A Datasets Used for Decision Rule Learning

Table A.1 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 Malioutov and Meel [11]. The original files were downloaded from the UCI Machine Learning Repository [120] and from Kaggle (https://www.kaggle.com). 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 [11]: 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 about the formulas generated with the encoding [11] 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 samples of the Connect 4 dataset. The formula of this dataset has 678 variables and 4152 clauses. The largest instance in terms of the number of samples 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 samples and 1511 discretized features.

Table A.1: The datasets used in the decision rule experiments and some summary statistics about them and the encoded formulas created 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