The Seesaw Algorithm: Function Optimization Using Implicit Hitting Sets

Mikoláš Janota
Czech Technical University in Prague, Czech Republic

António Morgado
INESC-ID Lisbon, Portugal

José Fragoso Santos
INESC-ID/IST, University of Lisbon, Portugal

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

Abstract

The paper introduces the Seesaw algorithm, which explores the Pareto frontier of two given functions. The algorithm is complete and generalizes the well-known implicit hitting set paradigm. The first given function determines a cost of a hitting set and is optimized by an exact solver. The second, called the oracle function, is treated as a black-box. This approach is particularly useful in the optimization of functions that are impossible to encode into an exact solver. We show the effectiveness of the algorithm in the context of static solver portfolio selection.

The existing implicit hitting set paradigm is applied to cost function and an oracle predicate. Hence, the Seesaw algorithm generalizes this by enabling the oracle to be a function. The paper identifies two independent preconditions that guarantee the correctness of the algorithm. This opens a number of avenues for future research into the possible instantiations of the algorithm, depending on the cost and oracle functions used.

2012 ACM Subject Classification Computing methodologies → Optimization algorithms

Keywords and phrases implicit hitting sets, minimal hitting set, MaxSAT, optimization

Digital Object Identifier 10.4230/LIPIcs.CP.2021.31

Funding The results were supported by the Ministry of Education, Youth and Sports within the dedicated program ERC CZ under the project POSTMAN no. LL1902. This scientific article is part of the RICAIP project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857306. This work was supported by national funds through FCT, Fundação para a Ciência e a Tecnologia, under project UIDB/50021/2020, the project INFOCOS with reference PTDC/CCI-COM/32378/2017 and the project DOME with reference PTDC/CCI-COM/31198/2017.

1 Introduction

Given a set of constraints, solving MaxSAT means finding a subset of given constraints under two different criteria: 1) the set must be the smallest possible 2) removing these constraints makes the whole set satisfiable. These two criteria go against each other because the fewer constraints we remove, the less likely we are to obtain satisfiability. In their seminal work, Davies and Bacchus [4] observe that MaxSAT can be solved by gradually enumerating the sets that any solution must intersect with, i.e., the solution is a hitting set of the enumerated sets. To guarantee that the smallest possible set is found, the algorithm only considers the smallest hitting sets. This style of solving is called the *implicit hitting set* algorithm. Implicit Hitting Set solving approaches have their roots in the 1980s in the theory of diagnosis [32, 33]. Since then, implicit hitting set solving and hitting set duality

have been successfully applied to many different problems including problems that are not necessarily in NP [6, 8, 15–17, 26, 27, 31, 35, 38]. Implicit hitting set approaches follow the common pattern: find a set of the minimal cost that satisfies a certain predicate. In the case of MaxSAT, the cost is the cardinality of the removed set and the predicate is the satisfiability of its complement.

This paper makes a crucial observation: *It is possible to extend the implicit hitting set algorithm beyond predicates.*

We show that it is possible to generalize the algorithm to minimize the cost under an objective function. We call this function the *oracle function*. This means that we are facing a *multi-objective optimization problem* with two objectives: cost and oracle. Since the objectives typically go against each other, it is generally impossible to point to a single best solution. However, we focus on *Pareto-optimal* solutions, which are solutions where improving either of the objectives requires worsening the other. The existing framework defined over predicates [16, 17, 27, 35] is subsumed by our approach because a predicate can be cast as a function that only returns either 0 or 1. Note that in the predicate-setting there are at most two Pareto-optimal solutions.

Why is this generalization useful? Consider the problem of selecting a good set of solvers, henceforth a *solver-portfolio*, for a given set of benchmarks. The selection of this set is guided by two criteria:

1. The solver-portfolio must be as small as possible, i.e., we want to use the minimum possible set of solvers.
2. The solver-portfolio's runtime in the benchmarks must be the best possible, i.e., the inclusion of more solvers means better runtime of the portfolio.

Again we have two optimization criteria going against each other and that is why our framework becomes useful in this context. In practice, we are mainly interested in the best portfolio of fixed size k; this can be tackled by the Seesaw algorithm. The fact that the oracle function in the algorithm is treated in a black-box fashion is also important in this context. This is because the possible ways of measuring the runtime of a solver-portfolio can typically be complex and inconvenient, or even impossible, to encode in traditional exact solvers within available resources. While this alone is already an important example, similar problems often appear in practice, e.g. selecting a good set of tests for a particular software system.

The paper has the following main contributions.

- The **Seesaw Algorithm** is introduced, which extends the implicit hitting set paradigm to calculate Pareto-optimal solutions over given cost and oracle functions.
- Two independent preconditions for the algorithm's correctness are identified.
- The algorithm is implemented and evaluated on optimization of solver portfolios and strategies of real-world portfolio systems.

2 Preliminaries

Standard notions and notation for propositional logic are assumed [37]. A *literal* is a Boolean variable (x) or its negation (denoted $\neg x$); a *clause* is a disjunction of literals. A formula is in *conjunctive normal form (CNF)* if it is a conjunction of clauses.

For a CNF ϕ a subset of its clauses $\psi \subseteq \phi$ is called a *maximal satisfiable set (MSS)* of ϕ if ψ is satisfiable and there is no ψ' such that $\psi \subsetneq \psi'$ and ψ' is satisfiable. Conversely, $\psi \subseteq \phi$ is called a *minimal correction set (MCS)* of ϕ if $\phi \setminus \psi$ is satisfiable and there is no ψ' such

that $\psi' \subsetneq \psi$ and $\phi \setminus \psi'$ is satisfiable. For a CNF ϕ a subset of its clauses $\psi \subseteq \phi$ is called a *minimal unsatisfiable set (MUS)* of ϕ if ψ is unsatisfiable and there is no ψ' such that $\psi' \subsetneq \psi$ and ψ' is unsatisfiable.

The Maximum Satisfiability (MaxSAT) problem is the task of finding the smallest possible correction set, or, equivalently finding the largest maximum satisfiable set, of a given ϕ.

2.1 Functions and Predicates

▶ **Definition 1** (monotone). *A function $f : 2^{\mathcal{U}} \to \mathbb{R}$ is* monotone *if and only if for any $S \subseteq S' \subseteq \mathcal{U}$ it holds that $f(S) \leq f(S')$.*

▶ **Definition 2** (anti-monotone). *A function $f : 2^{\mathcal{U}} \to \mathbb{R}$ is* anti-monotone *if and only if for any $S \subseteq S' \subseteq \mathcal{U}$ it holds that $f(S) \geq f(S')$.*

▶ **Definition 3** (strictly (anti-)monotone). *A function $f : 2^{\mathcal{U}} \to \mathbb{R}$ is strictly* (anti-)monotone *if and only if for any $S \subsetneq S' \subseteq \mathcal{U}$ it holds that $f(S) < f(S')$. (respectively, $f(S) > f(S')$).*

For the purpose of this paper, we treat a *predicate* as a special case of a function that always returns either 0 or 1 representing false and true, respectively. In the context of propositional satisfiability, two predicates have special importance. Given a set of clauses S, the predicate $\mathsf{SAT}(S)$ is true, if and only if S is satisfiable. Conversely, the predicate $\mathsf{UNSAT}(S)$ is true, if and only if S is unsatisfiable. The predicate SAT is monotone and the predicate UNSAT is anti-monotone, but the predicates are not strictly monotone/anti-monotone. The cardinality function, $|S|$ is strictly monotone.

▶ **Definition 4** (Hitting sets). *Let $\Gamma \subseteq 2^{\mathcal{U}}$ be a set of sets over some universe $\mathcal{U}$. A set $H \subseteq \mathcal{U}$ is called a hitting set of Γ if H has a nonempty intersection with every set of Γ. We write $\mathsf{HS}(\Gamma)$ for the set of all hitting sets of Γ.*

For an objective function $f : 2^{\mathcal{U}} \to \mathbb{R}$, a hitting set is f-minimal if it minimizes f over the set of all hitting sets.

2.2 Multi-objective Optimization

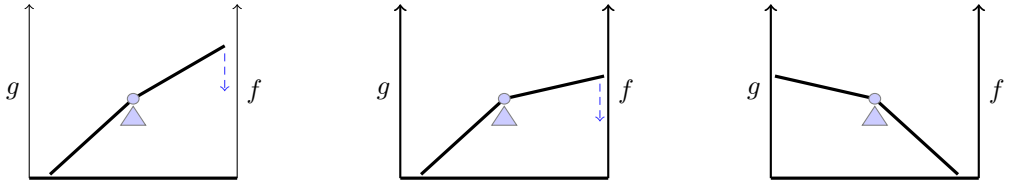
Throughout the paper we assume that any optimization function should be minimized. Under multiple optimization criteria, Pareto optimal solutions are such that improving any criterion means worsening some other. This idea is formalized by the following definitions.

▶ **Definition 5** (dominates). *Let $O = \{f_1, \ldots, f_l\}$ be a set of functions with domain D and range $\mathbb{R}$. We say that $x \in D$* dominates *$x' \in D$, if and only if for all $f \in O$ it holds that $f(x) \leq f(x')$, and, there exists an $f \in O$ such that $f(x) < f(x')$. If x dominates x', we write $x \prec x'$.*

▶ **Definition 6** (Pareto-optimal). *Let $O = \{f_1, \ldots, f_l\}$ be a set of functions with domain D and range $\mathbb{R}$. We say that $x \in D$ is* Pareto optimal *if and only if there does not exist any other $x' \in D$ dominating x.*

▶ **Definition 7** (Pareto frontier). *For some set of optimization functions O, the* Pareto frontier *is the set of all Pareto-optimal individuals.*

Note that even though this section refers to multi-objective optimization on a set of l functions, in this paper we consider optimizing 2 objective functions.



(a) cost g at minimum.

(b) oracle f at minimum relative to minimum cost g.

(c) minimum cost g relative to minimum oracle f.

Figure 1 Phases of the seesaw movement.

3 The Seesaw Algorithm

We are given two objective functions *cost* and *oracle*, denoted as g and f, respectively. Both functions are defined over sets of sets of some universe $\mathcal{U}$. We assume that we are able to minimize the cost function g using a dedicated solver, such as MaxSAT, or Integer Linear Programming (ILP) solver. The oracle function f is used as a black box.

The overall objective is to find the Pareto frontier of $\{g, f\}$. For this purpose we introduce the *Seesaw Algorithm*. The algorithm enables exploring the Pareto frontier using the implicit hitting set paradigm, as long as the functions fulfill certain properties, which are investigated later on. The algorithm is *any-time* in the sense that it may be stopped before the whole Pareto frontier is explored. The stopping criterion depends on the concrete problem at hand.

As a visual aid consider a seesaw where the middle is not completely rigid (Figure 1). This means that it is sometimes possible to push down one of the ends without the other one going immediately up. The height of each of the two ends represents the value of the two respective objective functions. Our algorithm traces the movement of this seesaw and we are at the liberty of stopping whenever we like.

Since the preference is to minimize both functions, we can imagine that there is a child sitting on either of the ends being pulled down towards the optimum by gravity. The movement is such that first there is only a child on the cost-end (g), and then someone places a heavier child on the oracle-end (f).

Figure 1 illustrates some notable phases of the movement. In the beginning, the cost function g is all the way down at its absolute minimum (Figure 1a). After that, the oracle function f starts being pushed down, which eventually causes f to reach its minimum point provided that g is maintained at its minimum (Figure 1b). This is the first Pareto-optimal point that the algorithm visits. After this phase, the cost function g leaves the ground and starts increasing, while the oracle function f continues to decrease. The movement terminates once f reaches its absolute minimum, meaning the seesaw hits the ground on the right-hand side (Figure 1c). This is the last Pareto-optimal point that the algorithm visits.

Let us look at a concrete example. Consider a MaxSAT problem defined by some unsatisfiable CNF ϕ. The objective is to find a smallest $S \subseteq \phi$ such that $\phi \setminus S$ becomes satisfiable. On the left-hand side of the seesaw we have the cost function $g = |S|$, and on the right-hand side we have the oracle function as the unsatisfiability predicate over $\phi \setminus S$ (i.e., $f = \mathsf{UNSAT}(\phi \setminus S)$). This means that if the cost g is all the way down, S is necessarily the empty set, which sends the f side to its maximum value, value 1, because $\mathsf{UNSAT}(\phi \setminus \emptyset)$ is true. Since the oracle f can only give two possible values (0 and 1), the initial phase coincides with the middle phase. The last phase corresponds to the solution of the MaxSAT problem: The unsatisfiability predicate became false and the size of S is the smallest possible under this condition, i.e. for this problem, the solution is obtained once Phase 3 is reached.

Algorithm 1 The Seesaw Algorithm.

```
1 H_best ← ⊥                                              // best candidate so far
2 Γ ← ∅                                                   // set of collected cores
3 while true do
4     H ← argmin_{H∈HS(Γ)} g(H)                           // find g-minimal hitting set
5     if H = ⊥ or stopping criterion then
6         return H_best
7     if f(H) < f(H_best) then                            // f upper bound bound improvement
8         H_best ← H
9     Γ ← Γ ∪ {extractCore(H, f(H_best), f)}              // calculate new core
```

Let us highlight several important properties of this concrete example. The cost-side g of the seesaw is easy to optimize (push down), because we have good solvers to optimize for cardinality. However, the oracle-side f represents some complex problem (satisfiability) over which we have lesser control. We follow this pattern for the rest of the paper, the cost function g may be optimized by a dedicated solver, e.g. MaxSAT, whereas the oracle function f is only queried for its value in a black-box fashion. In practice, however, for concrete applications, it of course makes sense to take advantage of whatever we know about the problem and try to steer the algorithm towards a faster improvement of the value of the oracle.

3.1 Formalizing the Algorithm

Algorithm 1 shows the pseudocode for the algorithm. The algorithm goes through a sequence of sets called *candidates*, which determine the current values of the objectives g and f. The essence of the algorithm is to improve the value of the oracle f while at the same time maintaining the smallest possible cost g. This is done by adding *necessary conditions* for the value of the oracle f to improve. Each of these conditions is recorded in the form of a set called *core*.

Throughout the course of the algorithm, all the cores are being accumulated in the variable Γ and the candidate is always chosen to be a hitting set of Γ. Hence, a core is defined as a set that any candidate improving on the value of f must necessarily intersect with, i.e., we say that the candidate "hits" all the accumulated cores.

▶ **Definition 8** (core). *Given an upper bound $v \in \mathbb{R}$, a set $\kappa \subseteq \mathcal{U}$ is called a* core *if all $H' \subseteq \mathcal{U}$ with $f(H') < v$ intersect with κ.*

In each iteration of the algorithm, first a new candidate (a hitting set of Γ) is calculated and a new core is added by invoking a dedicated function extractCore. New candidates are calculated by taking some g-minimal hitting set of Γ. Since this alone is NP-hard, a dedicated solver for the task is applied.

The concrete implementation of the function extractCore depends on the task at hand. However, two important properties need to be guaranteed by the function: 1) The returned set must prevent the current candidate solution from being found again. 2) The returned set must be a core. The first property guarantees termination and the second property guarantees that the algorithm does not exclude from search any candidates that might improve on the current value of the oracle function f.

Let us introduce definitions formalizing these properties. For the current candidate be excluded from further search, the set returned by extractCore must be a complement of the current candidate; otherwise, the current candidate continues to hit all cores of Γ.

▶ **Definition 9** (H-blocking set). *Given $H \subseteq \mathcal{U}$ a set $\kappa \subseteq \mathcal{U}$ is called* H-blocking *if $\kappa \subseteq (\mathcal{U} \setminus H)$.*

Whenever the current candidate is being blocked, we have to take some care as not to also block future candidates that improve the value of the oracle f. How can it happen that a future candidate is excluded from future search even if it improves on the value of f? Each candidate is a hitting set of the current set of cores Γ and once a candidate *stops* being a hitting set of Γ, immediately, any subset of the candidate also stops being a hitting set of Γ. This is an inherent property of hitting sets and it fundamentally influences the properties and requirements of the algorithm and therefore we note this in the following observation.

▶ **Observation 10.** *If H is* not *a hitting set of some set Γ, then any subset $H' \subseteq H$ is also not a hitting set of Γ.*

Due to this property of hitting sets, we may only block a candidate if we are sure that none of its subsets improve on the value of the oracle function f. We refer to this as careful blocking, anchored in the following definition.

▶ **Definition 11** (careful blocking). *Given a candidate $H \subseteq \mathcal{U}$, with $v = f(H)$, a set κ* carefully blocks *H, if and only if, $\kappa \subseteq (\mathcal{U} \setminus H)$ and for all $H' \subseteq \mathcal{U} \setminus \kappa$ it holds that $f(H') \geq v$.*

▶ **Observation 12.** *Given a candidate $H \subseteq \mathcal{U}$, any set κ that* carefully blocks *H is a core.*

To summarize the discussion so far, in each iteration of Algorithm 1 the new set added to Γ must be chosen as a complement of the current candidate but at the same time, this can only be done if it is guaranteed that no subset of the current candidate improves on the value of the oracle function. In general, such a core may not exist. The following section investigates two separate sufficient conditions for this existence.

3.2 Conditions for Careful Blocking

We identify two independent and sufficient conditions.

1. g is strictly monotone, or
2. f is anti-monotone

This means that the algorithm behaves correctly if the cost function g strictly prefers smaller sets, or, if the oracle function f prefers larger sets (non-strictly). We remark that Saikko et al. [35] only identifies the first of the two conditions in the context of oracles being predicates; since our framework is more general, both conditions apply also to predicates.

While either of the condition is sufficient, both may hold in some instantiations of the framework. For instance, in the case of MaxSAT, the function $|H|$ is strictly monotone, and $\mathsf{UNSAT}(\phi \setminus H)$ is anti-monotone.

In contrast, let us consider the task of finding an unsatisfiable set of some fixed cardinality $k \in \mathbb{N}$. This implies minimizing the cost function $g(H) \triangleq (k \neq |H|)$ (effectively maximizing $k = |H|$). The oracle function is $\mathsf{SAT}(H)$ (effectively maximizing $\mathsf{UNSAT}(H)$). In this situation, the cost function (predicate) is neither monotone nor anti-monotone, whereas the oracle function is anti-monotone.

Under either of the conditions, the weakest possible way of extracting cores is to calculate the complement of the current candidate hitting set, i.e.,

$$\mathsf{extractCore}_{\mathsf{w}}(H, v, f) \triangleq \mathcal{U} \setminus H.$$

Such a set is blocking the candidate H and we will also see that it is blocking the candidate carefully (Definition 11) whenever either of the two above conditions is satisfied. However, this default version of core extraction is way too weak since it effectively enumerates all possible sets. In the case of anti-monotone oracles, we show that we can significantly improve on that. Let us now look at these two conditions separately.

3.2.1 Strictly monotone cost g

▶ **Proposition 13.** *If g is strictly monotone and H is some candidate calculated throughout the course of the algorithm as a hitting set of Γ. Then any $H' \subsetneq H$ is* not *a hitting set of Γ.*

Proof. By contradiction assume that H' is a hitting set of Γ. Since g is strictly monotone, $g(H') < g(H)$, which is a contradiction because H is a g-minimal hitting set of Γ. ◀

▶ **Corollary 14.** *If g is strictly monotone, then any H can be carefully blocked.*

We observe that the requirement of *strict* monotonicity is tight. The requirement of g being monotone does not constitute a sufficient condition as shown by the following example.

▶ **Example 15.** Let $\mathcal{U} = \{a\}$ and $f(\emptyset) = 1$, $f(\{a\}) = 3$. Let g be constantly 0. Hence, both g and f are monotone. Since the algorithm may choose the first candidate arbitrarily, let us assume that it is $\{a\}$, the complement of which is the empty set and therefore the algorithm terminates even though the optimum of g has not been reached.

3.2.2 Anti-Monotone oracle f

For an anti-monotone f core extraction $\mathsf{extractCore}_{\mathsf{am}}$ is calculated as follows:

1. Non-deterministically choose H' a subset-maximal such that $H \subseteq H'$ and $f(H') \geq v$
2. **return** $\kappa \triangleq \mathcal{U} \setminus H'$.

To calculate subset maximal H' we refer to the *monotone predicates* framework [34].

▶ **Proposition 16.** *Let f be anti-monotone and $\kappa \triangleq \mathsf{extractCore}_{\mathsf{am}}(H, v, f)$. Then κ carefully blocks H.*

Proof. Since f is anti-monotone, any subset of $D \subseteq H'$ will either have the same value of f or worse. Hence, it cannot possibly improve f. This means that any solution strictly improving f is *not* be a subset of H', i.e. it must have an intersection with $\mathcal{U} \setminus H'$. ◀

▶ **Corollary 17.** *If f is anti-monotone, then any H can be carefully blocked for any given upper bound v.*

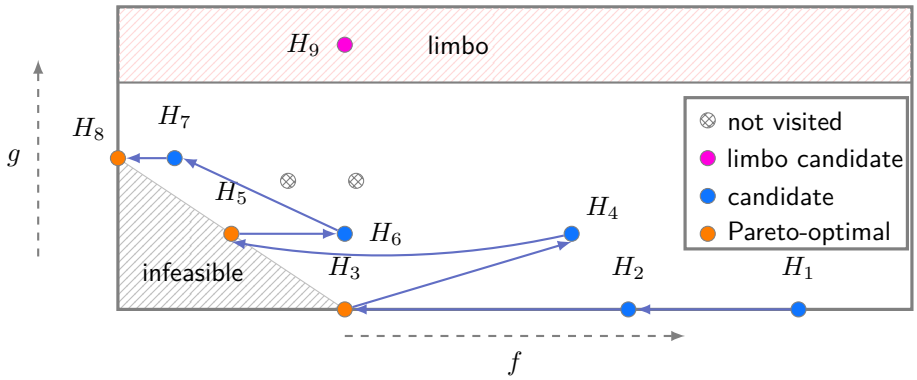


Figure 2 Behavior of the algorithm with respect to the Pareto frontier.

3.3 Properties

Figure 2 illustrates how the algorithm behaves in relation to the Pareto frontier of the two functions g and f. Most notably, the algorithm visits all the Pareto-optimal points in the order of the g function. However, some non-Pareto points may be visited in between. After the search has reached the optimal value of the oracle function, there are no guarantees on the obtained values. All candidates obtained after this stage are said to be in *limbo*. Ideally, we stop the algorithm before this stage, i.e., ideally, the limbo is empty.

Let us highlight some important properties of Figure 2. It starts with g being at the absolute minimum and f at some arbitrary value. After two iterations the first Pareto-optimal point is reached, where f is optimal under the condition that the cost is still minimal. Once the cost worsens, the process starts again, by looking for the next Pareto-optimal point with the smallest cost.

A special situation arises when the absolute minimum of f has been reached in the eighth iteration. In terms of the seesaw paradigm, this means that the oracle-end of the seesaw has hit the ground. After this happens, all bets are off because there are no more candidates that could possibly improve on the value of the oracle function. This means that in fact, any core is valid at this point. In particular, the algorithm may add the empty core and immediately terminate. However, for general implementation of the function $\mathsf{extractCore}$, we do not know if such property is guaranteed. Hence, the algorithm may hopelessly try to improve on the best possible value until it runs out of possible candidates (candidates must necessarily run out because the space is finite). We show that for core extraction $\mathsf{extractCore}_{\mathsf{am}}$, for anti-monotone oracle, the limbo is empty.

In the remainder of the section we focus on proving these properties. The first observation that we make is that the value of g cannot possibly improve over time because the set of possible hitting sets gradually diminishes. This means that the value of g criterion behaves anti-monotonically even if the function itself is not.

▶ **Proposition 18** (worsening of g). *Let H, H' be two candidates so that H was found in an earlier iteration than H'. Then, $g(H') \geq g(H)$.*

Proof. Let Γ, Γ' be the sets of cores used to calculate H, H', respectively. Since Γ only grows over time, the set of possible hitting sets of Γ only diminishes. More precisely, since $\Gamma \subsetneq \Gamma'$, it must necessarily hold that $\mathsf{HS}(\Gamma') \subseteq \mathsf{HS}(\Gamma)$. From which, necessarily, $\min_g(\mathsf{HS}(\Gamma')) \geq \min_g(\mathsf{HS}(\Gamma))$. ◀

▶ **Proposition 19.** *The Seesaw algorithm visits all the Pareto-optimal points of* $\{g, f\}$. *Further, the points are visited in the increasing value of the function* g.

Proof sketch. Let $\mathcal{P}$ be the set of all Pareto-optimal points that have not yet been visited and let H_P be the most recent Pareto-optimal point found. By induction we show that all elements of $\mathcal{P}$ are hitting sets of the current Γ and we have $g(H'_P) > g(H_P)$ for all $H'_P \in \mathcal{P}$. The hypothesis is trivially true at the beginning because Γ is empty.

From the induction hypothesis and Proposition 18, all the points in $\mathcal{P}$ have a larger or equal value of g than $g(H_P)$. Consequently, they also have a lower value of f as otherwise they would be dominated by H_P, i.e., we have $g(H'_P) > g(H_P), f(H'_P) < f(H_P)$ for $H'_P \in \mathcal{P}$. Since the algorithm only carefully blocks candidates (Definition 11), none of the hitting sets from $\mathcal{P}$ will be blocked from future search unless visited. Since candidates are always chosen to be g-minimal, the point from $\mathcal{P}$ to be first visited must be the one with the lowest value of g. ◀

▶ **Proposition 20.** *For anti-monotone* f *and* $\mathsf{extractCore}_{\mathsf{am}}$ *defined as above, there are no candidates in limbo.*

Proof. Let H be such that the value of f reached its maximum, meaning that $f(H) \leq f(H')$ for any $f(H')$. The procedure $\mathsf{extractCore}_{\mathsf{am}}$ goes on adding elements to H while the value of f does not improve but that never happens and therefore the procedure results in calculating H' as $\mathcal{U}$, whose complement is the empty core. Once the empty core is added to Γ, the algorithm terminates because there are no more hitting sets. ◀

4 Experimental Evaluation

The presented Seesaw algorithm is particularly suitable for problems where the optimization function (oracle) is difficult to encode into an exact optimization solver. The problem we choose here is the selection of a portfolio of solvers, or configurations of solvers, out of a large set. There exist approaches that apply machine learning to predict the best solver, or a collection of solvers, per instance, e.g., SATzilla [42] or CPHydra [29], cf. [24]. In practice, however, solvers often employ a *static* portfolio and this is also the setting we consider for our experimental evaluation.

The problem is specified as a set of solvers $\mathcal{U} = S_1, \ldots, S_n$ and a set of instances $\mathcal{I}$ on which the solvers were run. For an instance $i \in \mathcal{I}$, we write $S(i)$ for the runtime of the solver S on this instance. If the solver does not solve the instance, $S(i)$ is some fixed penalty. In competitions, the penalty is typically chosen according to the PAR2 definition, which gives a constant penalty equal to the double of the time limit [1]. We refer to the sum of the values $S(i)$ across all instances as the *PAR2 score* of the solver. The aim is to calculate a subset of the solvers so that their PAR2 score is the smallest, when the solvers are run in parallel. More specifically, for a set of solvers $H \subseteq \mathcal{U}$ the oracle value is defined as follows.

$$\sum_{i \in \mathcal{I}} \min_{S \in H} S(i) \tag{1}$$

The oracle function is anti-monotone. Indeed, since solvers are run in parallel, when a new solver is added, the total score can only decrease or remain the same. The problem is a generalization of the classical set-cover problem [9]. For the cost function, there are two natural choices.

1. Define cost as the cardinality of the candidate, i.e., $g(H) \triangleq |H|$.
2. Define cost as the predicate $|H| = k$ for some fixed integer $k \in 1..|\mathcal{U}|$, i.e., $g(H) \triangleq (1 \text{ if } |H| = k \text{ else } 0)$.

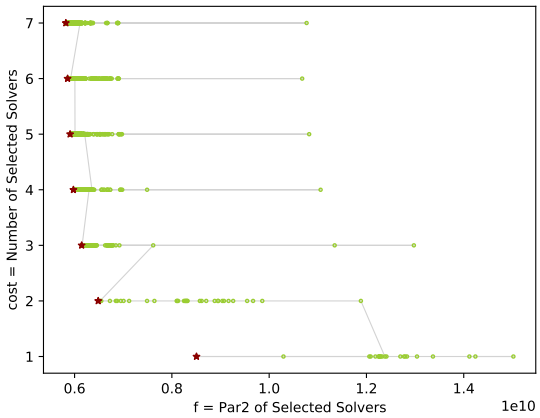


Figure 3 Pairs of values of g, f, for the SMT data set outlining a portion of the Pareto front.

In the case of cost function 1, the algorithm will explore all the Pareto front, given enough resources (see Figure 2). In the case of cost function 2, the algorithm will search for the optimal portfolio of cardinality k with respect to the PAR2 score as defined by equation (1).

Experimental Data

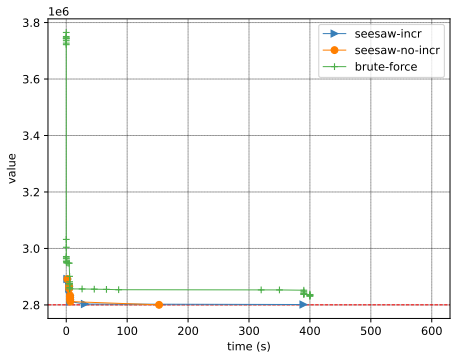
We use two data sets kindly provided by researchers in the corresponding field. The first is collected when exploring strategies for quantifier handling in the satisfiability modulo theories (SMT) solver CVC4 [2]. This exploration is motivated by the search for a set of strategies that the solver is to use in the SMT competition. This data set counts 50 different solvers and 75815 instances. We call this the *SMT dataset*.

The second data set is from the research on automated theorem provers (ATP), where a large body of strategies was considered. These were collected as follows. The various E Prover [36] configurations were invented specifically for first-order translation of Mizar Mathematical Library [41] by various methods. Specifically, they come: (1) from the E's auto-schedule mode, (2) from the system BliStrTune [19] for targeted invention of theorem proving strategies, and finally (3) from various experiments with clause selection guidance system ENIGMA [18, 20] which is based on machine learning. This data set counts 156 different solvers and 57880 instances. We call this the *ATP dataset*.

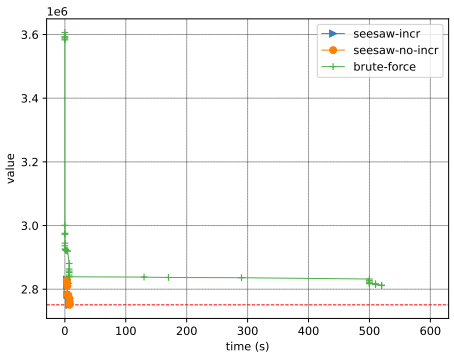
Observe that both data sets have a large number of instances, which incurs a large encoding of the oracle function if encoded explicitly. However, in the Seesaw framework, the function is only calculated on demand programmatically.

Evaluation

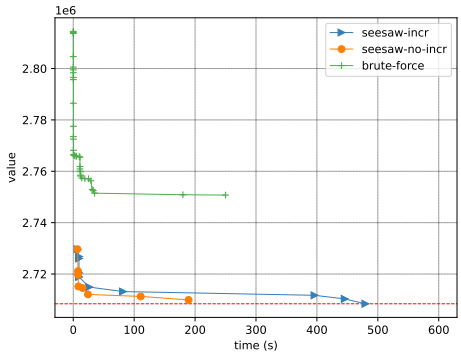
We have implemented Seesaw (Algorithm 1) in C++, using the Gurobi [11] solver to solve the integer linear programming subproblems. We considered two versions of the algorithm, in one of the versions Gurobi is used incrementally and in the other non-incrementally, which means all constraints need to be reloaded each time the solver is called. We also implemented a *brute-force solution* of the problem, which simply enumerates all possible subsets of cardinality k. Further, we implemented a *direct encoding* into Gurobi. However, the direct encoding into Gurobi reached memory limit in all the benchmarks and therefore was not able to solve any of the considered problems.



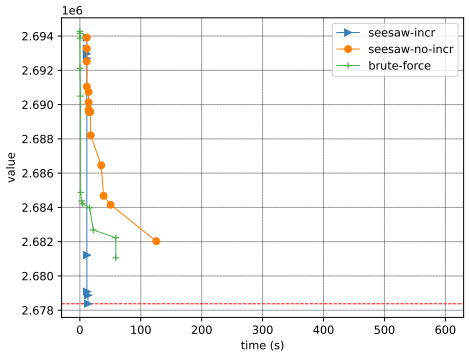
(a) Portfolio size 7, i.e. $k = 7$.



(b) Portfolio size 16, i.e. $k = 16$.



(c) Portfolio size 39, i.e. $k = 39$.



(d) Portfolio size 78, i.e. $k = 78$.

Figure 4 Example runs of the Seesaw algorithm for fixed portfolio size for the ATP data set.

As an additional optimization, when minimizing cores, we shuffle the elements randomly as to increase diversity in cores and therefore increase the size of the minimal hitting set. All the experiments were performed on servers with Intel(R) Xeon(R) CPU at 2.60GHz, 24 cores, 64GB RAM.

We first consider the setting where the cost function is simply the cardinality of the selected portfolio, for the first data set (SMT). The algorithm was run for 5 hours, and the worst cost reached was 7, i.e., the algorithm found all Pareto optimal points for cardinality 1..7. The run of Seesaw is plotted in Figure 3. Each pair cost (g), oracle (f) is represented by a point. Points marked with a star in dark red, correspond to Pareto points; Pareto sub-optimal points are rendered as green circles. The faded grey line, connects the points by the order in which the points are discovered (starting with a point in bottom, and stops in a point in the upper side of the plot). Observe that this figure is analogous to Figure 2.

We consider the ATP data set for fixed cardinality of the portfolio. The time limit for the algorithm was set to 600 seconds.

Since we were only able to prove optimal values for cardinalities 2 and 3, we focus here on the best value obtained by the different approaches. Figures 4a–4d show the evolution of the objective value in time for 4 different cardinalities of the portfolio ($7 \approx 5\%|\mathcal{U}|$, $16 \approx 10\%|\mathcal{U}|$, $39 \approx 25\%|\mathcal{U}|$, $78 \approx 50\%|\mathcal{U}|$). The red dashed line delimits the best value achieved amongst the solvers.

Table 1 Best values achieved on samples of ATP for different cardinalities.

sample	cardinality (k)	seesaw-no-incr	seesaw-incr	brute-force
1	$k = 5$	2814287	**2814191**	2822009
1	$k = 13$	2764882	2764196	**2760229**
1	$k = 25$	**2735018**	2740535	2738386
2	$k = 5$	**2821306**	2823459	2824569
2	$k = 13$	2765332	**2763383**	2771172
2	$k = 25$	2730132	**2729146**	2732681
3	$k = 5$	2818798	**2818174**	2819918
3	$k = 13$	2765479	**2763187**	2767774
3	$k = 25$	2730116	2732724	**2728360**
4	$k = 5$	2827634	2830806	**2825306**
4	$k = 13$	2773371	**2768368**	2774381
4	$k = 25$	**2737996**	2738524	2745392
5	$k = 5$	2827598	**2823750**	2833257
5	$k = 13$	**2756893**	2763900	2775798
5	$k = 25$	2732700	**2732124**	2738979

First thing to observe is that there is a tendency of finding a good solution at the beginning and only improve it a little bit in the long run. From an user perspective, it means that running the algorithm longer has drastically diminishing returns.

This is quite noticeable in $k = 78$, where none of the approaches is able to improve on the value found in the first 2 minutes. Even though the non-incremental version spends much more time on reloading the constraints into Gurobi, it is not necessarily worse (see for instance $k = 39$). This suggests that the right choice of cores is the crucial ingredient in the algorithm. Indeed, in the case of $k = 78$, the incremental version found a better solution in the first 50 iterations then the non-incremental version after 5000 iterations.

In $k = 7$ and $k = 16$ all the approaches are gravitating to the same value and therefore it is possible that we are getting close to the optimal value. However, proving that the value is optimal appears to be extremely hard. Note that already $\binom{156}{16} \approx 3 \times 10^{21}$ and $\binom{156}{78} \approx 6 \times 10^{45}$.

To complement these results, we also sample the ATP dataset into 5 different subsets of size 50. The obtained results are presented in Table 1. For each considered sample and cardinality of the portfolio, the table shows the best value found by the different approaches. In the majority of cases, the Seesaw algorithm finds the best value. However, in some cases it is outperformed by brute-force. This can be explained by the fact that brute-force is able to explore much more many candidates and in some cases it may hit a good one by chance. This suggests that it would be beneficial to design a hybrid approach in order to preserve the intelligence of Seesaw but also cover more ground.

5 Related Work

5.1 Implicit Hitting sets

The implicit hitting set (IHS) approach has been successfully used in the highly competitive MaxSAT solver MaxHS [4,5]. The idea here stems from the fact that any minimal correction set is a hitting set of all MUSes. Hence, if we enumerated all MUSes, the smallest corrections set would be obtained by calculating the minimum hitting set of those. However, the number

of MUSes may be exponential and therefore rather than enumerating all of them, MaxHS enumerates them one by one and it tests whether a correction set is obtained by picking one of the minimum hitting sets of the MUSes enumerated so far.

From the point of view of complexity theory, solving the minimum hitting set problem (MHSP) is as difficult as solving MaxSAT. However, in practice, it has been observed that state-of-the-art integer linear programming solvers perform well on MHSP. This is supported by theoretical results that show that MHSP is intractable for propositional resolution [23].

Moreno-Centeno and Karp introduce a general framework for solving NP-complete problems by the implicit hitting set paradigm and apply it to a number of problems [27]. Saikko et al. extend this framework further and observe that it is not limited to problems in NP [35]. In both aforementioned frameworks, the search for the minimal hitting set is guided by an *oracle predicate*, which effectively determines if the search should stop. The framework presented here generalizes all of the above by considering an *oracle function* rather than just a predicate. Predicates are seen as functions that either return 0 or 1. We further generalize the precondition of the algorithm by demonstrating that the algorithm does not require the cost function to be strictly monotone, as required by Saikko et al., as long as the oracle predicate/function is anti-monotone.

There is a large body of research on the use of oracles and problem decomposition. We highlight the most relevant works. The IHS-based approaches bear similarity with the *Bender's decomposition* [3] in the sense that the problem is decomposed in two different functions. Moreover, Bender's decomposition is not restricted to linear programming and it has been generalized to other optimization problems [10, 12]. Monotone predicates play a special role in our framework, which have been studied extensively in the context of SAT [34]. Our generalization of the IHS paradigm is analogous to the generalization of decision problems to function problems, such as NP to FNP [30]. Nadel employs SAT solver as an oracle in approximate optimization, similar to Walk-SAT but using SAT in each step [28].

5.2 Quantified Boolean Formulas (QBF)

In parallel to IHS, similar approaches were developed in QBF based on the AReQS paradigm [21, 22], which was further extended to QBF under optimization [14, 17]. It can be shown that this approach in fact reduces to IHS for certain QBF problems.

For illustration, consider the *smallest MUS problem*. The problem is specified by a set of clauses $C_1, \ldots, C_n$ on variables from some set X. The QBF formulation introduces *selector variables* $s_1, \ldots, s_n$ and the formula $\exists s_1, \ldots, s_n \forall X \bigvee_{i \in [n]} (s_i \wedge \neg C_i)$. The formula expresses that there is a selection of clauses, determined by the selector variables, so that for any assignment to the X variables there is at least one clause that is falsified.

The task is to find a satisfying assignment for this QBF that sets to true the fewest selector variables. In the optimization AReQS paradigm, the solver collects assignments to X, each of these assignments is substituted into the formula, which results in the disjunction of selector variables of falsified clauses. Minimization is then performed gradually on the set of these disjunctions, which in fact correspond to cores in IHS. This was observed by Ignatiev et al., who carefully implement a specialized algorithm for the smallest MUS problem [16].

5.3 Multi-Objective Optimization

A large body of work exists on *Multi-Objective Combinatorial Optimization (MOCO)* and *Multi-Objective Mathematical Programming (MOMP)* [13, 40]. Multi-objective optimization is typically concerned with different strategies of navigating the Pareto frontier based on user preferences. In our setting, the way the Pareto frontier is navigated is implicit since we treat the oracle function in a black-box fashion.

Multi-objective optimization has also been studied in the context of SAT. Dedicated algorithms exist for solving MaxSAT under lexicographic preferences [25]. Pareto frontier has been explored by calculating minimal correction sets (MCSes) [39]. These approaches, however, are not immediately applicable if the functions are not encodable as propositional constraints.

A large body of work exists on the invention of good solver portfolios. These techniques rely on the combination of machine learning and constraints solving [24].

6 Conclusions and Future Work

The paper introduces and studies the Seesaw algorithm, which enables exploring the Pareto frontier using the implicit hitting set paradigm. The main strength of the algorithm is that it enables combining exact and black-box optimization. The algorithm receives as input two functions (cost and oracle), where cost is optimized exactly by a dedicated solver (e.g. ILP, MaxSAT), whereas the oracle is used in a black-box fashion.

The algorithm generalizes the existing implicit hitting set paradigm on predicates to functions. Implicit hitting sets on predicates were extensively studied in the last decade [4, 14, 16, 27, 35]. Interestingly, the framework is known to be applicable in problems that go beyond NP, which immediately implies that Seesaw algorithm also goes beyond NP; it is an open question of how to precisely characterize the complexity of the algorithm depending on the oracle used.

In our implementation we have used the commercial Gurobi solver [11]; in the future we aim to evaluate a larger set of solvers, such as MaxSAT or modern pseudo-Boolean solvers [7].

The novel Seesaw framework opens a number of opportunities for exact specialized optimization in various domains. We have demonstrated that the framework is readily usable in the context of optimizing solver portfolios and strategies. Furthermore, we aim to apply the algorithm to systematically explore the efficiency of test-quality measures.

References

1 Adrian Balint, Anton Belov, Matti Järvisalo, and Carsten Sinz. Overview and analysis of the SAT challenge 2012 solver competition. *Artif. Intell.*, 223, 2015. doi:10.1016/j.artint.2015.01.002.

2 Clark W. Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In *Computer Aided Verification – 23rd International Conference, CAV*. Springer, 2011. doi:10.1007/978-3-642-22110-1_14.

3 Jacques F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Comput. Manag. Sci.*, 2(1):3–19, 2005. doi:10.1007/s10287-004-0020-y.

4 Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In *Principles and Practice of Constraint Programming - CP*. Springer, 2011. doi:10.1007/978-3-642-23786-7_19.

5 Jessica Davies and Fahiem Bacchus. Exploiting the power of MIP solvers in MaxSAT. In *Theory and Applications of Satisfiability Testing (SAT)*, volume 7962. Springer, 2013. doi:10.1007/978-3-642-39071-5_13.

6 Erin Delisle and Fahiem Bacchus. Solving weighted CSPs by successive relaxations. In *International Conference on Principles and Practice of Constraint Programming*. Springer, 2013.

7 Jan Elffers and Jakob Nordström. A cardinal improvement to pseudo-boolean solving. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence*. AAAI Press, 2020. URL: https://aaai.org/ojs/index.php/AAAI/article/view/5508.

8 Katalin Fazekas, Fahiem Bacchus, and Armin Biere. Implicit hitting set algorithms for maximum satisfiability modulo theories. In *International Joint Conference on Automated Reasoning*. Springer, 2018.

9 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

10 Arthur M Geoffrion. Generalized Benders decomposition. *Journal of optimization theory and applications*, 10(4), 1972.

11 LLC Gurobi Optimization. Gurobi optimizer reference manual, 2021. URL: `http://www.gurobi.com`.

12 John N. Hooker and Greger Ottosson. Logic-based benders decomposition. *Mathematical Programming*, 96(1), 2003.

13 Ching-Lai Hwang and Abu Syed Md. Masud. *Multiple Objective Decision Making — Methods and Applications*. Springer Berlin Heidelberg, 1979. `doi:10.1007/978-3-642-45511-7`.

14 Alexey Ignatiev, Mikoláš Janota, and João Marques-Silva. Quantified maximum satisfiability: A core-guided approach. In *Theory and Applications of Satisfiability Testing (SAT)*, volume 7962. Springer, 2013. `doi:10.1007/978-3-642-39071-5_19`.

15 Alexey Ignatiev, António Morgado, and João Marques-Silva. Propositional abduction with implicit hitting sets. In *ECAI 2016 - 22nd European Conference on Artificial Intelligence*. IOS Press, 2016. `doi:10.3233/978-1-61499-672-9-1327`.

16 Alexey Ignatiev, Alessandro Previti, Mark H. Liffiton, and João Marques-Silva. Smallest MUS extraction with minimal hitting set dualization. In *Principles and Practice of Constraint Programming*. Springer, 2015. `doi:10.1007/978-3-319-23219-5_13`.

17 Alexey Ignatiev, Mikoláš Janota, and João Marques-Silva. Quantified maximum satisfiability. *Constraints An Int. J.*, 21(2), 2016. `doi:10.1007/s10601-015-9195-9`.

18 Jan Jakubův, Karel Chvalovský, Miroslav Olšák, Bartosz Piotrowski, Martin Suda, and Josef Urban. ENIGMA anonymous: Symbol-independent inference guiding machine (system description). In *IJCAR (2)*, volume 12167 of *Lecture Notes in Computer Science*. Springer, 2020. `doi:10.1007/978-3-030-51054-1_29`.

19 Jan Jakubův and Josef Urban. BliStrTune: hierarchical invention of theorem proving strategies. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP*. ACM, 2017. `doi:10.1145/3018610.3018619`.

20 Jan Jakubuv and Josef Urban. Hammering Mizar by learning clause guidance. In *ITP*, volume 141 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

21 Mikoláš Janota, William Klieber, Joao Marques-Silva, and Edmund Clarke. Solving QBF with counterexample guided refinement. *Artificial Intelligence*, 234, 2016. `doi:10.1016/j.artint.2016.01.004`.

22 Mikoláš Janota and Joao Marques-Silva. Abstraction-based algorithm for 2QBF. In Karem A. Sakallah and Laurent Simon, editors, *Theory and Applications of Satisfiability Testing - SAT*. Springer, 2011. `doi:10.1007/978-3-642-21581-0_19`.

23 Stasys Jukna. Exponential lower bounds for semantic resolution. In *Proof Complexity and Feasible Arithmetics, Proceedings of a DIMACS*, volume 39. DIMACS/AMS, 1996. `doi:10.1090/dimacs/039/10`.

24 Marius Lindauer, Holger H. Hoos, Frank Hutter, and Kevin Leyton-Brown. Selection and configuration of parallel portfolios. In *Handbook of Parallel Constraint Reasoning*. Springer, 2018. `doi:10.1007/978-3-319-63516-3_15`.

25 João Marques-Silva, Josep Argelich, Ana Graça, and Inês Lynce. Boolean lexicographic optimization: algorithms & applications. *Ann. Math. Artif. Intell.*, 62(3-4), 2011. `doi:10.1007/s10472-011-9233-2`.

26 Joao Marques-Silva, Alexey Ignatiev, and Antonio Morgado. Horn maximum satisfiability: Reductions, algorithms and applications. In *EPIA Conference on Artificial Intelligence*. Springer, 2017.

27 Erick Moreno-Centeno and Richard M. Karp. The implicit hitting set approach to solve combinatorial optimization problems with an application to multigenome alignment. *Oper. Res.*, 61(2), 2013. doi:10.1287/opre.1120.1139.

28 Alexander Nadel. On optimizing a generic function in SAT. In *2020 Formal Methods in Computer Aided Design, FMCAD 2020*. IEEE, 2020. doi:10.34727/2020/isbn.978-3-85448-042-6_28.

29 Eoin O'Mahony, Emmanuel Hebrard, Alan Holland, Conor Nugent, and Barry O'Sullivan. Using Case-based Reasoning in an Algorithm Portfolio for Constraint Solving. In *Irish Conference on Artificial Intelligence and Cognitive Science*, 2008.

30 Christos H. Papadimitriou. *Computational complexity*. Academic Internet Publ., 2007.

31 Mark Parker and Jennifer Ryan. Finding the minimum weight IIS cover of an infeasible system of linear inequalities. *Annals of Mathematics and Artificial Intelligence*, 17(1), 1996.

32 James A Reggia, Dana S Nau, and Pearl Y Wang. Diagnostic expert systems based on a set covering model. *International journal of man-machine studies*, 19(5), 1983.

33 Raymond Reiter. A theory of diagnosis from first principles. *Artificial intelligence*, 32(1), 1987.

34 Mikoláš Janota and João Marques-Silva. On the query complexity of selecting minimal sets for monotone predicates. *Artif. Intell.*, 233, 2016. doi:10.1016/j.artint.2016.01.002.

35 Paul Saikko, Johannes Peter Wallner, and Matti Järvisalo. Implicit hitting set algorithms for reasoning beyond NP. In *Principles of Knowledge Representation and Reasoning (KR)*. AAAI Press, 2016. URL: http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12812.

36 Stephan Schulz. E - a brainiac theorem prover. *AI Commun.*, 15(2-3), 2002.

37 João P. Marques Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009. doi:10.3233/978-1-58603-929-5-131.

38 Roni Stern, Meir Kalech, Alexander Feldman, and Gregory Provan. Exploring the duality in conflict-directed model-based diagnosis. In *AAAI Conference on Artificial Intelligence*, volume 26, 2012.

39 Miguel Terra-Neves, Inês Lynce, and Vasco M. Manquinho. Introducing Pareto minimal correction subsets. In *Theory and Applications of Satisfiability Testing - SAT*. Springer, 2017. doi:10.1007/978-3-319-66263-3_13.

40 E. L. Ulungu and J. Teghem. Multi-objective combinatorial optimization problems: A survey. *Journal of Multi-Criteria Decision Analysis*, 3(2), 1994. doi:10.1002/mcda.4020030204.

41 Josef Urban. MPTP 0.2: Design, implementation, and initial experiments. *J. Autom. Reason.*, 37(1-2), 2006.

42 Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. SATzilla: Portfolio-based algorithm selection for SAT. *J. Artif. Intell. Res.*, 32, 2008. doi:10.1613/jair.2490.