Learning MAX-SAT Models from Examples using Genetic Algorithms and Knowledge Compilation

Senne Berden
Declarative Languages and Artificial Intelligence, KU Leuven, Leuven, Belgium

Mohit Kumar
Declarative Languages and Artificial Intelligence, KU Leuven, Leuven, Belgium

Samuel Kolb
Declarative Languages and Artificial Intelligence, KU Leuven, Leuven, Belgium

Tias Guns
Declarative Languages and Artificial Intelligence, KU Leuven, Leuven, Belgium

Abstract

Many real-world problems can be effectively solved by means of combinatorial optimization. However, appropriate models to give to a solver are not always available, and sometimes must be learned from historical data. Although some research has been done in this area, the task of learning (weighted partial) MAX-SAT models has not received much attention thus far, even though such models can be used in many real-world applications. Furthermore, most existing work is limited to learning models from non-contextual data, where instances are labeled as solutions and non-solutions, but without any specification of the contexts in which those labels apply. A recent approach named HASSLE-SLS has addressed these limitations: it can jointly learn hard constraints and weighted soft constraints from labeled contextual examples. However, it is hindered by long runtimes, as evaluating even a single candidate MAX-SAT model requires solving as many models as there are contexts in the training data, which quickly becomes highly expensive when the size of the model increases. In this work, we address these runtime issues. To this end, we make two contributions. First, we propose a faster model evaluation procedure that makes use of knowledge compilation. Second, we propose a genetic algorithm named HASSLE-GEN that decreases the number of evaluations needed to find good models. We experimentally show that both contributions improve on the state of the art by speeding up learning, which in turn allows higher-quality MAX-SAT models to be found within a given learning time budget.

2012 ACM Subject Classification Computing methodologies → Machine learning

Keywords and phrases Machine learning, constraint learning, MAX-SAT

Digital Object Identifier 10.4230/LIPIcs.CP.2022.24

Supplementary Material *Software (Source Code)*: `https://github.com/ML-KULeuven/HASSLE-GEN`

Funding This project was partially funded by the Research Foundation - Flanders (FWO) projects G0G3220N and FWO-S007318N, as well as the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant Nos. 101002802, CHAT-Opt and 694980, SYNTH) and the "Agentschap Innoveren & Ondernemen" (VLAIO) as part of the innovation mandate HBC.2021.0246. (S. Kolb).

Acknowledgements We would like to thank Luc De Raedt for his helpful comments.

1 Introduction

Many real-world problems can be effectively solved by means of combinatorial optimization. With appropriate mathematical models, problems like planning [25], routing [20] and scheduling [9] can be delegated to a highly-optimized solver that can quickly and automatically yield one or more high-quality solutions. However, automatic solvers do not remove the need for human effort altogether; they merely move it from the solving phase to the modeling phase. Unfortunately, the manual construction of an appropriate model can be difficult for several reasons. First, it requires both modeling and expert domain knowledge. Finding people that possess both, or setting up a collaboration to marry both, is not a trivial issue. Second, some optimization criteria and constraints might be hard to express explicitly for a human modeler. Third, even with the criteria and constraints successfully modeled, the potentially difficult task of weighting them by importance relative to each other remains, at least for problems with multiple optimization criteria and problems that contain weighted soft constraints. Finally, manual modeling can be a laborious process. It might require many iterations of gradual refinement of the model, as well as a survey of the various stakeholders to discover what is considered important and what constraints might apply. This arduous model construction phase motivates another approach altogether: *learning* the model from data, as opposed to manually constructing it.

This is the machine learning task we tackle in this work. Specifically, we focus on learning *MAX-SAT models*, a type of mathematical model that can be used to represent many interesting real-world problems [1, 9, 25, 26, 27]. The task involves jointly learning hard and soft constraints—as well as appropriate weights for the latter—that best explain a set of labeled contextual examples [16]. These examples are known solutions and non-solutions to the considered problem. For instance, when learning a model for rostering nurses in a hospital, historical rosters can be used as examples. The rosters would be labeled as positive or negative depending on whether they were deemed suitable.

As alluded to above, the type of examples we consider are *contextual*; they come with associated *contexts*. These contexts represent the states of affairs that the examples occurred in. Let us examine their relevance. In the nurse rostering scenario, a roster in which some nurses have to work several consecutive long shifts is likely not optimal in normal circumstances, and thus not a solution. However, in the context of a pandemic taking place, or many nurses being unavailable due to illness or other circumstances, such a roster might be optimal and an appropriate solution to the rostering problem. Contexts are a useful way of including this type of situational information in the example set, with the goal of learning a model that can generalize across contexts. For any given context, such a model can then be used to construct new solutions that are appropriate in that context. As will be discussed in Section 2, learning from examples that are annotated with contexts is one of the things that sets this work apart from most existing related work.

The problem of learning MAX-SAT models from labeled contextual examples has recently been addressed in [16]. This work laid the theoretical groundwork, showing that, given enough training data and time, high-quality models can be learned. Subsequent work introduced HASSLE-SLS, a more efficient stochastic local search approach [17]. However, HASSLE-SLS still suffers from long runtimes, which stem from the particularly expensive candidate model evaluation involved in learning. Faster learning is desirable, as it would allow for better models to be learned in a given training time budget. To this end, we make two contributions:

1. We develop a novel knowledge-compilation-based evaluation procedure that significantly speeds up MAX-SAT model evaluation when learning small to medium-sized models, or when learning from a large set of examples. Because evaluation is the major bottleneck in the search for good models, improving its efficiency speeds up learning considerably.
2. We develop a novel genetic algorithm named HASSLE-GEN, whose crossover operator takes much larger steps in the search space than HASSLE-SLS, reducing the number of model evaluations needed to find a good model. Because of this, HASSLE-GEN learns high-quality MAX-SAT models significantly faster than the state of the art.

2 Related Work

We focus specifically on learning weighted partial MAX-SAT models. This work is positioned in the research field of constraint learning, because the unweighted and weighted clauses in these models can respectively be seen as hard and weighted soft constraints. Here, we provide an overview of some of the most relevant works in constraint learning. For a more thorough overview, we refer the reader to [8].

Learning hard constraints. We start by discussing a few approaches that are aimed at learning constraint satisfaction problems (CSPs). The first of these, named *ConAcq.1*, is a version space algorithm that learns CSPs from examples labeled as solutions and non-solutions [5]. The version space is represented implicitly as a propositional CNF formula over variables that denote binary constraints. Complete variable assignments that satisfy the formula then correspond to CSPs in the version space, i.e., CSPs that are consistent with respect to the given labeled examples. This approach was followed by *ConAcq.2* [6] and *QuAcq* [4], two active learning approaches that learn more efficiently by respectively asking the user complete and partial membership queries.

Some other approaches are instead focused on learning global constraints. For example, [23] and [24] focus on learning the parameters of global constraints from a small pool of positive examples. Similarly, *Model Seeker* is effective at learning global constraints from positive examples provided in matrix-form [3]. In *Model Seeker*, fitting constraints are taken from a catalogue of constraints and related metadata (e.g., information about implication relations between constraints), and then subjected to several types of redundancy checks and simplifications. This method has proven to be very effective, even in the presence of many variables or only few examples.

Finally, *GenetiCS* is a method that learns constraints for mathematical programming models from known feasible and infeasible examples [22]. The approach is related to the one developed in this work in that it involves an evolutionary algorithm, but differs in that it learns linear and polynomial (in)equalities represented as abstract syntax trees (ASTs), rather than weighted partial MAX-SAT models. Additionally, *GenetiCS* does not learn from contextual examples.

Learning soft constraints. Several methods that focus on learning an objective to be optimized, in the form of weighted soft constraints, also exist. They can be categorized into parameter learning and structure learning approaches. The former is merely aimed at learning the weights of a given set of constraints. The latter also learns the constraints themselves.

One relevant example is *CLEO*, an interactive method whose purpose is to learn the weights of clauses occurring in MAX-SAT or MAX-SMT models [7]. Although the focus lies on learning weights, the algorithm is asked to weight many more constraints than are desired in the final model. However, a sparsity assumption is applied by including the minimization of the weight vector's 1-norm as one of the method's objectives, leading to a lot of constraints getting assigned a weight of zero. In this sense, the approach can be seen as a form of structure learning.

Another approach with the purpose of learning soft constraints and their weights is presented in [13]. In this work, a type of weighted MAX-SAT model is learned. These MAX-SAT models are an extension to the ones we consider, in that the constraints are function-free disjunctive first-order logic clauses, and thus are not limited to propositional logic. Another difference with the work presented here is the input given: the method requires examples of possible worlds as well as preference relations between these worlds to be provided. It then uses inductive logic programming to learn a set of appropriate constraints, which are subsequently weighted using preference learning techniques.

HASSLE. The approaches discussed above are focused on learning either hard constraints or soft constraints. They differ significantly with our approach, in which both types of constraints are being learned *jointly*. This joint learning is desirable in the setting we consider, as will be discussed in Section 4. Another large difference is that none of the above works are aimed at learning from *contextual* examples. However, it is a realistic assumption that what one considers optimal depends on the context one is positioned in.

Although none of the methods discussed above is applicable to the presently considered problem setting, two approaches that do try to solve this task already exist. The first approach, called HASSLE-MILP, was introduced in [16]. This work also proved that MAX-SAT models are PAC-learnable, justifying empirical risk minimization. HASSLE-MILP formulates the learning task as a mixed-integer linear program (MILP), which can be solved by an off-the-shelf solver. The main drawbacks of this method are that it is not optimized for efficiency and that, unless a solver that offers anytime functionality is used, it either produces a solution that has zero empirical error, or no solution at all.

The second approach, called HASSLE-SLS, is a stochastic local search algorithm that improves on HASSLE-MILP by offering anytime functionality and by increasing efficiency [17]. It keeps track of a current model and iteratively constructs and evaluates a neighborhood of minimally altered models, of which the best neighbor then replaces the current model. To make the approach tractable, the neighborhood is constructed heuristically, keeping only those models that show some promise of improving on the current model according to the employed heuristic.

One large drawback of HASSLE-SLS is its long runtime, which is exacerbated when the size of the learning problem increases. The algorithm slows down significantly when the model one aims to learn becomes larger, or when the number of distinct contexts occurring in the training data increases. At the heart of this issue lies the high cost of model evaluation. HASSLE-SLS is particularly affected by this, because in every iteration, it evaluates an entire neighborhood of MAX-SAT models before making only a minimal alteration to the current model. In this paper, we alleviate the runtime issues in two ways. First, we make model evaluation less expensive by making use of knowledge compilation. Second, we reduce the number of evaluations needed to find a good model, by developing a novel search strategy called HASSLE-GEN.

3 Preliminaries

Weighted partial MAX-SAT. Let $X = \{X_1, \ldots, X_n\}$ be a set of Boolean variables. An *assignment* x—also referred to as an *instance*—is a mapping of each variable $X_i \in X$ to either *true* or *false*.

Let $L = \{X_i, \overline{X_i} | X_i \in X\}$ be the set of literals defined over X, where $\overline{X_i}$ is the negation of X_i. A *disjunctive clause* C is a disjunction of literals from L. An assignment satisfies C when it satisfies at least one of the literals occurring in C. A formula in *conjunctive normal form* (CNF) F is a conjunction of clauses. An assignment *satisfies* F when it satisfies all of the clauses occurring in F. A *weighted clause* (C, w) consists of a clause C and a weight w, which we assume lies in $[0, 1]$ without loss of generality. A weighted CNF formula is a conjunction of weighted clauses.

A *MAX-SAT problem* is defined by a CNF formula F. An assignment is a solution to the MAX-SAT problem when it satisfies the maximum number of clauses that can simultaneously be satisfied in F. A *partial* MAX-SAT problem is defined by two CNF formulas, F_h and F_s. An assignment is a solution when it satisfies F_h and satisfies the maximum number of clauses that can simultaneously be satisfied in F_s while satisfying F_h. Finally, a *weighted* partial MAX-SAT problem is defined by a CNF formula F_h and a weighted CNF formula F_s. An assignment is a solution when it satisfies F_h and accumulates the maximum total weight in satisfied weighted clauses that can be accumulated in F_s while satisfying F_h. The clauses in F_h and the weighted clauses in F_s can respectively be seen hard and weighted soft constraints. When an assignment is not a solution because it does not satisfy all the clauses in F_h, it is called *infeasible*. When it *does* satisfy all clauses in F_h, but does not accumulate the maximum attainable weight in satisfied soft constraints of F_s, it is called *suboptimal*.

For the sake of brevity, we from now on refer to weighted partial MAX-SAT simply as MAX-SAT. Because this work aims to *learn* MAX-SAT problems from known solutions and non-solutions, we will generally refer to a MAX-SAT problem as a MAX-SAT *model*. When we speak of the MAX-SAT learning problem, we refer to the problem of learning MAX-SAT models.

Contexts. As explained in Section 1, there is a strong motivation to think of a historical labeled example as an instance that is known to be a solution or non-solution *in a specific context*. As a context might represent a state of affairs, it often makes sense for it to be a conjunction. Most generally, however, a context ϕ is simply a propositional formula over Boolean variables.

An assignment is a solution to a MAX-SAT model consisting of hard constraints F_h and weighted soft constraints F_s in a context ϕ if it satisfies F_h and ϕ, and accumulates the maximum total weight in satisfied weighted clauses that can be accumulated in F_s while satisfying F_h and ϕ. Thus, it is possible for an instance to be optimal in a particular context, but suboptimal outside of that context.

4 Problem Statement

In this work, the goal is not to solve MAX-SAT problems, but to *learn* them from a set of labeled, contextual examples. Specifically, such an example consists of:

1. A context ϕ
2. An assignment x that satisfies context ϕ
3. A Boolean label l, denoting whether assignment x is an optimal solution to the target model in context ϕ or not

Note that the label 'non-solution' does not specify whether the example is a non-solution because it is infeasible or because it is suboptimal. We focus on this type of supervision, because in real-world settings, the reason for the negative label is typically not available. While this assumption on the input makes supervision easier to provide, it also gives rise to a credit assignment problem: when a candidate model wrongly labels an example as a solution, it is unclear whether the hard constraints or the soft constraints should be altered. For this reason, both types of constraints should be learned jointly, rather than separately. The learning task becomes:

▶ **Definition 1** (MAX-SAT learning). *Given Boolean variables $X = \{X_1, \ldots, X_n\}$ and a set of labeled contextual examples $S = \{(\phi_i, x_i, l_i) | i = 1, \ldots, m\}$, find hard constraints F_h and* *weighted soft constraints F_s that define a MAX-SAT model which can be used to obtain high-quality instances in any context ϕ.*

Something not yet specified is what constitutes a high-quality instance. Intuitively, an instance is good when it is feasible and close to optimal with respect to the ground-truth model. The learned model's ability to generate high-quality instances is reflected in its *infeasibility* and *average regret*, which will be the primary performance measures in the experimental evaluation. The infeasibility expresses what proportion of solutions to the learned model are actually infeasible with respect to the ground-truth model. The regret captures how good the learned model's solutions are with respect to the ground-truth model's soft constraints. In section 7, we will discuss how exactly these measures are computed.

As the ground-truth model is not available, infeasibility and regret cannot be used during learning. Instead, we aim to maximize the model's training set accuracy, which is the proportion of examples whose label correctly denotes how the example relates to the model.

5 Knowledge Compilation

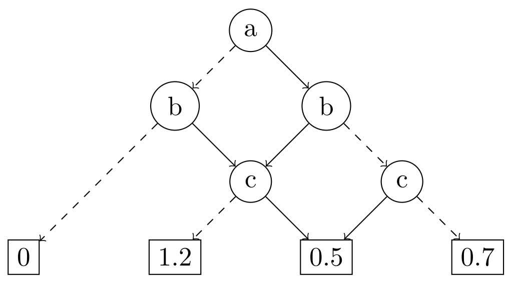
When learning MAX-SAT models, the vast majority of the runtime is spent evaluating candidate models. Let us consider in more detail why this is the case.

To fully evaluate a model's training set accuracy, it has to be evaluated on all contextual examples included in the training data. Checking whether an instance is feasible or infeasible is relatively straightforward: one merely has to loop over at most all hard constraints of the model, and check for each whether it is satisfied by the instance. Checking optimality, however, requires knowing the maximum total weight that can be accumulated while satisfying the hard constraints. Attaining this information requires *solving* the model, which is NP-hard [18]. An additional complication is the fact that examples are accompanied by contexts, which might affect the optimal total weight that can be accumulated. So, in order to correctly evaluate a single candidate model, HASSLE-SLS solves a separate MAX-SAT problem from scratch for every distinct context occurring in the data (which can then be cached and reused for all examples sharing that context). This causes model evaluation to be particularly slow, which in turn significantly slows down learning.

The solution we present is based on a novel representation of a MAX-SAT model as an algebraic decision diagram (ADD). An ADD is an extension of a binary decision diagram (BDD), in which the terminal nodes can be assigned real values, rather than just *true* or *false* [2]. It is a rooted, directed, acyclic graph which consists of two types of nodes: decision nodes and terminal nodes. A decision node is associated with a Boolean variable and branches into two child nodes; one for each truth assignment to the variable. A terminal node is associated with a real value.

Our ADD representation of a MAX-SAT model maps every infeasible instance onto a terminal node with value 0 and every feasible instance onto a terminal node with as value the sum of the weights of all satisfied soft constraints. A useful aspect of this representation is that the optimal value (in the absence of a context) can very easily be discovered: one simply has to loop through all terminals in order to find the highest one. This information is required to determine whether an instance is a solution. An example of a MAX-SAT model and its corresponding ADD representation is shown in Example 2.

▶ **Example 2.**
Consider the MAX-SAT model with hard constraints $F_h = (a \lor b)$ and soft constraints $F_s = (0.5 : b \lor c) \land (0.7 : \neg c)$. This problem corresponds to the following ADD, in which negative edges are represented as dashed lines and positive edges as solid lines:



Although the construction of the ADD is necessarily an expensive operation—as it practically solves the NP-hard MAX-SAT problem—there are potential time benefits in reusing the resulting ADD for context-specific inference. The main idea is that each candidate model can be converted to an ADD once, after which the diagram can be reused to compute the optimal attainable value for all separate contexts occurring in the training data. Contrast this with the original approach, in which a brand new MAX-SAT problem has to be solved for every model-context combination.

The evaluation procedure. The full knowledge-compilation-based evaluation procedure is shown at a high level in Algorithm 1. It takes as input a MAX-SAT model and a set of labeled contextual examples, and it produces as output the model's accuracy on the set of examples. To do this, it performs three steps. First, it constructs an ADD that represents the model in the absence of any context. Next, it runs through all the contexts present in the example set and, for each, uses the ADD constructed in step 1 to compute the optimal value in that context. Finally, it runs through all examples and, for each, uses the ADD constructed in step 1 to compute the value the example achieves. Each example's value, along with its associated context's optimal value computed in step 2, can then be used to label the example. Finally, the assigned label is compared with the example's label in the training data.

This description naturally raises the following two questions, which we answer in turn:

1. How to convert a MAX-SAT model to an ADD?
2. How to incorporate contexts in the inference done on the ADD?

Constructing the ADD. To transform a MAX-SAT model defined by hard constraints F_h and soft constraints F_s to an ADD, we make use of the basic operators defined on BDDs and ADDs. We do not go over the details of these operations here. For more information, we refer the reader to chapter 6 of the book *Logic in Computer Science* [12].

The transformation procedure starts by constructing a separate BDD for each disjunctive clause, irrespective of whether it is weighted or not. In each BDD that corresponds to a soft constraint from F_s, the terminal that denotes *true* is then given the constraint's associated weight as value. This effectively casts the BDD to an ADD representing the weighted soft constraint.

All the BDDs that represent hard constraints are then combined using the multiplication operator. This results in a single BDD that represents the conjunction of all hard constraints. Similarly, the ADDs representing soft constraints are combined using the addition operator. This results in a single ADD in which every instance leads to a terminal that denotes the total weight in satisfied soft constraints accumulated by that instance. However, this ADD does not yet consider whether the instances are feasible or not.

Algorithm 1 The evaluation procedure using knowledge compilation.

```
 1: procedure EVALUATE($model$: a MAX-SAT model, $examples$: a set of labeled contextual
    examples)
 2:     $score \leftarrow 0$
 3:     $ADD_{model} \leftarrow$ convert $model$ to ADD
 4:     $contexts \leftarrow$ set of all separate contexts occuring in $examples$
 5:     for each $context$ in $contexts$ do
 6:         Compute best value of $ADD_{model}$ in $context$
 7:         Cache this best value
 8:     for each $example$ in $examples$ do
 9:         $instance, context, label \leftarrow$ the instance, context and label of $example$
10:         $best\text{-}value \leftarrow$ retrieve optimal value in $context$ from cache
11:         $value \leftarrow$ compute value of $instance$ in $ADD_{model}$
12:         if $value = best\text{-}value$ then
13:             $assigned\text{-}label \leftarrow$ solution
14:         else
15:             $assigned\text{-}label \leftarrow$ non-solution
16:         if $label = assigned\text{-}label$ then
17:             $score \leftarrow score + 1$
18:     $score \leftarrow \frac{score}{\text{LENGTH}(examples)}$
19:     return $score$
```

Finally, the BDD representing the hard constraints and the ADD representing the soft constraints are multiplied. The effect of this is that the terminals of all feasible paths of the ADD representing the soft constraints are multiplied by 1, while the terminals of infeasible paths are multiplied by 0. In turn, all feasible paths still end up in the same terminal node as before, while infeasible paths are redirected to a terminal with value 0. For compactness, the resulting diagram is then reduced to a reduced ordered BDD [12]. The resulting diagram represents the entire MAX-SAT model.

Context-specific inference. Once a candidate model has been transformed into an ADD, we want to be able to quickly infer the optimal value attainable in a specific context ϕ.

A straightforward way of doing this involves multiplying the ADD that represents the MAX-SAT model with a BDD that represents ϕ, and doing inference on the resulting diagram. However, we have found that this generally takes too long and does not shorten overall model evaluation time. Similarly, performing restrict operations on the model's ADD representation in accordance with ϕ is not fast enough.

Instead, we go through the entire ADD, starting from the root node, and ignoring branches that violate ϕ. The maximum terminal value reached this way is the best attainable value achievable in ϕ.

This process is straightforward when ϕ is a conjunction of literals, as one simply has to ignore branches that violate one of the literals in ϕ. Luckily, this is arguably the most common scenario, as a context typically represents a state of affairs, which is most naturally represented using a conjunction.

When ϕ is a disjunction, or more generally a DNF formula, one can repeat the process above for each conjunctive clause in ϕ. The maximum terminal value reached for any of the conjunctive clauses is then the maximum attainable value in ϕ as a whole.

As an optimization, we perform a precomputation before any example or context is considered, which involves finding and storing all paths leading to the terminal node with the maximum value in the ADD representing the MAX-SAT model. A quick computation can then be made for every context occurring in the training data to determine whether it violates all of the paths leading to said terminal node. If any of these paths is not violated, one knows immediately that the optimal value in the context is the same as the one in the absence of a context, which has been precomputed. In the other case, the procedure detailed above is used.

6 The Genetic Algorithm

As discussed above, the bottleneck in HASSLE-SLS is the evaluation of MAX-SAT models, because it involves *solving* MAX-SAT problems, which is NP-hard. In every iteration of HASSLE-SLS, an entire neighborhood—which frequently consists of several dozen models—is evaluated, before the best neighbor is identified and a *minimal* alteration to the current model is made. This is the crux of the problem: many expensive candidate model evaluations lead only to a minimal step in the search space.

For this reason, we develop an alternative search strategy in the form of a genetic algorithm, which we call HASSLE-GEN. Genetic algorithms form a class of population-based metaheuristic optimization approaches that are loosely inspired by biological evolution [10, 11, 21]. They aim to solve optimization problems by evolving a *population* of candidate solutions, also referred to as *individuals*. They generally consist primarily of genetic selection, mutation and crossover operators. An overview of HASSLE-GEN is given in Algorithm 2. In what follows, we discuss its components in detail.

Algorithm 2 HASSLE-GEN, a genetic algorithm for learning MAX-SAT models from examples.

```
1: procedure HASSLE-GEN($examples$: a set of labeled contextual examples, $k$: the
   total number of constraints to learn, $q$: the population size, $p_c$: the crossover probability,
   $[p_{m1}, p_{m2}, p_{m3}]$: the mutation probabilities, $g$: the maximum number of generations, $t$:
   the cutoff time)
2:     Randomly initialize a $population$ of models containing $k$ constraints
3:     Evaluate $population$
4:     while generation $< g \wedge$ runtime $< t$ do
5:         $new\text{-}population \leftarrow$ empty population
6:         while $new\text{-}population$ not of size $q$ do
7:             $parent_1$, $parent_2 \leftarrow$ CROWDING-PARENT-SELECTION($population$)
8:             if RANDOM() $< p_c$ then
9:                 $ind \leftarrow$ CLAUSE-CROSSOVER($parent_1$, $parent_2$, $examples$)
10:            else
11:                $ind \leftarrow$ either $parent_1$ or $parent_2$, selected at random
12:            $ind \leftarrow$ HARDNESS-MUTATION($ind$, $p_{m1}$)
13:            $ind \leftarrow$ WEIGHT-MUTATION($ind$, $p_{m2}$)
14:            $ind \leftarrow$ LITERAL-MUTATION($ind$, $p_{m3}$, $examples$)
15:            $surv_1$, $surv_2 \leftarrow$ CROWDING-SURVIVOR-SELECTION($ind$, $parent_1$, $parent2$)
                   ▷ Includes evaluation of $ind$
16:            Add $surv_1$ and $surv_2$ to $new\text{-}population$
17:        $population \leftarrow new\text{-}population$
18:    return best individual in $population$
```

Selection. Selection consists of two components: parent selection and survivor selection. The former is concerned with determining which individuals of the current population to subject to the mutation and crossover operations. The latter is concerned with determining which individuals to keep in the next generation's population and which to discard. Good selection strikes an appropriate balance between exploitation of useful information present in the current population and exploration of new regions of the search space.

Both HASSLE-GEN's parent and survivor selection are determined by its use of a variation of the deterministic crowding scheme [19], which we employ because it is effective at maintaining population diversity, which in turn benefits the search.

Parent selection is straightforward in deterministic crowding: in every generation, each individual is selected to be a parent exactly once. Because HASSLE-GEN's crossover operator only produces a single offspring, this means that in every generation, every individual gives rise to exactly one offspring, together with another parent individual.

Deterministic crowding's survivor selection requires a distance metric d between individuals to be defined. It uses this metric in the following way. Say parents p_1 and p_2 gave rise to offspring o. Then, by the time survivors have to be selected, a matching is made. Parent p_1 is matched to o only if $d(p_1, o) < d(p_2, o)$; otherwise, parent p_2 is matched to o. The offspring and matched closest parent then compete, wherein only the individual with the highest fitness makes it to the next generation. The other parent automatically survives. Here, the fitness of a MAX-SAT model is simply its training set accuracy.

A first consequence of this scheme is that the average fitness of the population never decreases, because a parent never gets replaced by a worse individual. Second, the best individual is automatically kept in the next generation, unless this individual has produced an even better offspring. Finally, and most importantly, there is a strong emphasis on maintaining the initial population's diversity. This is true because an offspring is typically quite similar to its parents, with one of which it has to compete for survival, and because the matching procedure sets up offspring-parent competitions in such a way that the distance between the competitors is minimized. Having new individuals compete for survival with similar old individuals is how diversity is maintained, because this prevents any specific genetic information from quickly taking over the entire population.

What remains to be answered is how the distance metric d is instantiated. We opt for a metric that captures the semantic distance between MAX-SAT models, rather than a syntactic distance. The metric makes use of the notion of an accuracy bit vector.

▶ **Definition 3** (Accuracy bit vector). *Given a list of examples S, a MAX-SAT model M has an associated accuracy bit vector v, which has as many entries as S has examples. For each index i, the entry at index i in v is 1 iff M accurately labels the example at index i in S, and 0 otherwise.*

▶ **Definition 4** (Semantic distance). *Let S be a list of m examples and let M_1 and M_2 be two MAX-SAT models with respective accuracy bit vectors v_1 and v_2. Let their number of correctly labeled examples be s_1 and s_2, respectively. Finally, let d_H be the Hamming distance between v_1 and v_2. Then, the semantic distance d_{Sem} between M_1 and M_2 is*

$$d_{Sem} = \frac{d_H - L}{U - L}, \text{ where } L = |s_1 - s_2| \text{ and } U = m - |s_1 + s_2 - m|$$

This semantic distance metric considers the Hamming distance of the accuracy bit vectors, *relative to a lower and upper bound.* If we were to simply use the Hamming distance itself, we would give inherent preference to matching highly accurate or highly inaccurate models.

Mutation. A mutation operator takes a single individual as input. Usually, it only makes a small modification to the individual, in order to slightly redirect the search or to introduce new genetic information that can then be recombined by crossover operators.

HASSLE-GEN contains three mutation operators, which are all applied to any selected parent. The first operator, named *hardness mutation*, loops over all of the individual's constraints and, independently for each constraint, with probability p_{m1}, changes the constraint from a hard to a soft constraint or vice versa.

The second operator, named *weight mutation*, loops over all of the individual's weighted soft constraints, and independently for each constraint, with probability p_{m2}, replaces the constraint's weight by a weight sampled uniformly at random from $(0, 1]$.

Finally, *literal mutation* is responsible for altering the occurrences of variables in clauses. This operator differs in nature from the first two in that it is more informed; it considers the training data. It takes effect with probability p_{m3}. It starts by randomly selecting one constraint of the individual being mutated. This constraint is the only one to be affected by the mutation. The operator then constructs a neighborhood around this constraint, consisting of all other constraints that differ in only a single variable occurrence (i.e., a single literal appears, disappears or changes its sign), excluding all constraints that already occur in the model. Finally, the operator uses a heuristic way of evaluating all neighboring constraints, and mutates the selected constraint into the best neighboring one. The heuristic evaluation of constraints is based on the notion of *coverage*, where constraints with higher coverage are better according to the heuristic.

▶ **Definition 5** (Covering). *A constraint c is said to cover an example e consisting of instance i and label l if and only if:*
- *l is 'solution' and i satisfies c*
- *l is 'non-solution' and i does not satisfy c*

▶ **Definition 6** (Coverage bit vector). *Given a list of examples S, a constraint c has an associated coverage bit vector v, which has as many entries as S has examples. For each index i, the entry at index i in v is 1 iff c covers the example at index i in S, and 0 otherwise. The sum of all of v's entries is called c's coverage.*

Note that a constraint's coverage is only a heuristic measure of its usefulness, as it disregards the weights of soft constraints and the existence of contexts. Still, in preliminary experiments, we found that the use of this heuristic leads to a much more effective operator than one that simply alters literals at random.

Crossover. A crossover operator recombines information from multiple individuals, generally two. The motivation is that by combining individuals that are fit for different reasons, new individuals can be obtained that combine the strengths of both parents.

HASSLE-GEN contains one crossover operator, which we call *constraint crossover*. It takes two parents as input and produces a single offspring. Unlike the mutation operators, constraint crossover keeps the constraints themselves intact, but rearranges which constraints co-occur. It is through this mechanism that HASSLE-GEN is able to take larger steps in the search space than HASSLE-SLS. Like literal mutation, constraint crossover is an informed operator and considers the training data. This allows it to bias its steps towards directions that are likely to improve the model's training set accuracy.

For any pair of parents, constraint crossover takes effect with probability p_c. Given two parents, each containing k constraints, it selects k constraints that are combined in the single offspring. It does not do so blindly, but considers combinations of the constraints' coverage bit vectors.

▶ **Definition 7** (Combining coverage bit vectors). *Let S be a list of examples, and let c_1 and c_2 be constraints with respective associated coverage bit vectors v_1 and v_2. The coverage bit vector v of conjunction $c_1 \wedge c_2$ is attained by performing:*

- *a pairwise AND operation on v_1 and v_2 for all examples in S labeled 'solution'*
- *a pairwise OR operation on v_1 and v_2 for all examples in S labeled 'non-solution'*

The sum of all of v's entries is called the coverage of $c_1 \wedge c_2$.

One option would be to choose the k constraints taken from the two parents which lead to the largest coverage when combined. However, identifying these constraints involves computing the coverage of $\binom{2k}{k}$ combinations.

Instead, we opt for a sequential selection of constraints that works as follows. First, the constraint with the highest coverage is selected and copied into the offspring. Then follows a repeated selection of the constraint that leads to the highest coverage when combined with the already selected constraints, until k constraints have been selected.

7 Experimental Evaluation

In this section, we thoroughly investigate the following research questions.

Q1 Does the knowledge-compilation-based model evaluation procedure speed up the evaluation of candidate MAX-SAT models?

Q2 When given the same amount of time, is HASSLE-GEN able to learn higher-quality MAX-SAT models than HASSLE-SLS?

Datasets. Each synthetic ground-truth model with k_h hard constraints and k_s soft constraints over n variables is generated such that none of its clauses is entailed by any combination of the other clauses in the model. For each disjunctive clause, the number of literals is chosen uniformly at random from $[1, 5]$. The literals themselves are also selected randomly. For k_s of the generated clauses, a weight is sampled uniformly from $(0, 1]$.

For each generated ground-truth model, a dataset is constructed in two phases. First, a set of conjunctive contexts of $n/2$ literals is generated such that each included context actually affects the maximum attainable value in the ground-truth model. Then, for each context, a specified number of infeasible, suboptimal and solution instances are generated. Infeasible and suboptimal instances are acquired simply by generating random instances that satisfy the context until the desired number of infeasible and suboptimal ones are found. To generate solution instances, a solver is used. To prevent overly similar solution instances, 10 times as many solutions as required are generated for every context, after which the desired amount are sampled at random.

Performance measures. To answer **Q1**, we consider the relative increase in the number of evaluations HASSLE-SLS makes per second when using the novel evaluation procedure compared to when using the original procedure. For example, a speed-up factor of 3.4 would mean that HASSLE-SLS was able to perform 3.4 times as many evaluations with the new evaluation procedure than with the original procedure in the given cutoff time.

To answer **Q2**, we compute the learned model's score, infeasibility and average regret. A model's score is simply its training set accuracy, which was used as training objective.

The other two metrics assess the quality of the learned model's solutions with respect to the ground-truth model that was used to generate the training data.

A learned model M's *infeasibility* expresses what proportion of solutions of M are actually infeasible with respect to the ground-truth model M^*. It can be measured exactly by use of *model counting* (MC), i.e., counting the number of solutions to a propositional formula. However, the MAX-SAT models we consider are not just propositional formulas; they also contain weighted soft constraints. For this reason, we require a propositional formula expressing the solutions to the MAX-SAT model. Let M be a MAX-SAT model with hard constraints F_h. Say that $\hat{x}$ is a solution to the model which realizes a value of $\hat{v}$ in satisfied soft constraints. We can then find each subset of soft constraints S_i for which the associated weights sum up to $\hat{v}$. Using this, a propositional formula θ_M expressing the solutions of M can be attained as:

$$\theta_M = F_h \wedge (\bigvee_{S_i} \bigwedge_{\theta_s \in S_i} \theta_s)$$

This formula expresses that an instance is a solution to the model if it satisfies the hard constraints and satisfies one of the subsets of soft constraints that realizes the optimal value. With F_h^* denoting the hard constraints of ground-truth model M^*, the infeasibility of M can be computed as:

$$\text{inf}_{M^*}(M) = \frac{\text{MC}(\theta_M \wedge \neg F_h^*)}{\text{MC}(\theta_M)}$$

The *average regret* of a learned model M with respect to a ground-truth model M^* expresses how good the solutions of M are with respect to the soft constraints of M^*. We compute average regret using only solutions to M that are feasible in M^*. Hence, infeasibility and regret should be considered together to get a complete picture of a model's quality. We generate up to 1000 such feasible instances. For each such instance x, let v^* be the value it realizes with respect to M^*'s soft constraints, and let $\hat{v}^*$ be M^*'s optimal value. The regret of x is then simply $(\hat{v}^* - v^*)/\hat{v}^*$. The average regret of M with respect to M^* is then the average regret over the considered instances. The regret is computed in the *global context*, i.e., outside of any particular context. Thus, achieving low regret requires good generalization across contexts, as the model is evaluated in the global context, but is trained using only contextual data in which the contexts actually matter (i.e., affect the optimal attainable value of the ground-truth model).

Results. To answer **Q1**, we run HASSLE-SLS with a cutoff time of 60 seconds on learning problems of various sizes. We consider sizes similar to the ones considered in [16] and [17]. By default we use learning problems with 10 variables, 8 hard constraints and 8 soft constraints in the ground-truth model and 100 contexts with 1 infeasible, 1 suboptimal and 2 solution examples per context in the dataset. We change each of these properties in turn, while keeping the others at their default values. When increasing the number of total constraints of the ground-truth model, half are hard constraints and half are soft constraints. When increasing the number of examples per context, half are solutions, a quarter are infeasible and a quarter are suboptimal. For each learning problem size, we vary the randomization seed to create 15 different learning problems, over which we average the results.

As shown in Figure 1, the novel ADD-based model evaluation procedure gives rise to a significant speed-up for all learning problem sizes considered. It should be noted that, as the number of variables or the number of constraints increases, the relative speed-up decreases.

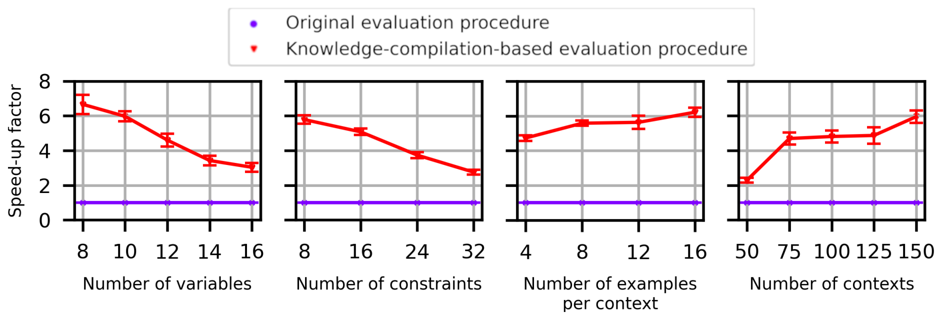


Figure 1 The speed-up in model evaluation realized by using the knowledge-compilation-based model evaluation procedure decreases as the number of variables or the number of constraints in the model increases. On the other hand, the speed-up increases as the example set grows larger.

This is caused by the ADD representation growing in size as the size of the MAX-SAT model increases. On the other hand, when larger training sets are considered, the ADD representation can be reused to a higher degree, increasing the relative speed-up. We can conclude that the knowledge-compilation-based evaluation procedure is most useful when learning relatively small MAX-SAT models, or when learning from large training datasets. This answers **Q1**.

To answer **Q2**, we again vary several aspects of the learning problem in turn, but consider slightly larger learning problems. This time, by default we use 16 variables, 16 hard constraints and 16 soft constraints in the ground-truth model and 100 contexts with 1 infeasible, 1 suboptimal and 2 solution instances per context in the dataset. Again, 15 different randomization seeds are used for each learning problem size. We run HASSLE-SLS and HASSLE-GEN—both using the knowledge-compilation-based evaluation procedure—on each learning problem using a cutoff time of 150 seconds. HASSLE-GEN was run with a population size of 20, $p_c = 0.5$, $p_{m1} = 0.05$, $p_{m2} = 0.05$ and $p_{m3} = 1$, which were determined in a coarse grid search on a separate set of learning problems.

As the first row of Figure 2 shows, HASSLE-GEN consistently achieves a higher score than HASSLE-SLS across many learning problem instances with varying properties. Furthermore, when the size of the considered models increases, the difference between the scores achieved by the search methods grows larger. This can be explained by the fact that HASSLE-SLS only makes a minimal alteration to the current model in every iteration: it changes at most a single literal, the hardness or the weight of a single constraint. These alterations become increasingly inconsequential when considering larger models, resulting in exponentially larger search spaces. By contrast, HASSLE-GEN's constraint crossover allows for much larger steps in the search space, and because it considers the training data, these steps tend to go into an improving direction.

The second and third rows of Figure 2 show how the higher score achieved by HASSLE-GEN in turn translates into lower infeasibility and regret. This shows that the models learned by HASSLE-GEN do not merely achieve a higher training set accuracy than those learned by HASSLE-SLS, but that the solutions they generate are also of a higher quality with respect to the ground-truth model. This answers **Q2**.

8 Conclusion

We have made two contributions aimed at alleviating the runtime issues of the state-of-the-art technique for learning MAX-SAT models from labeled contextual examples. First, to speed

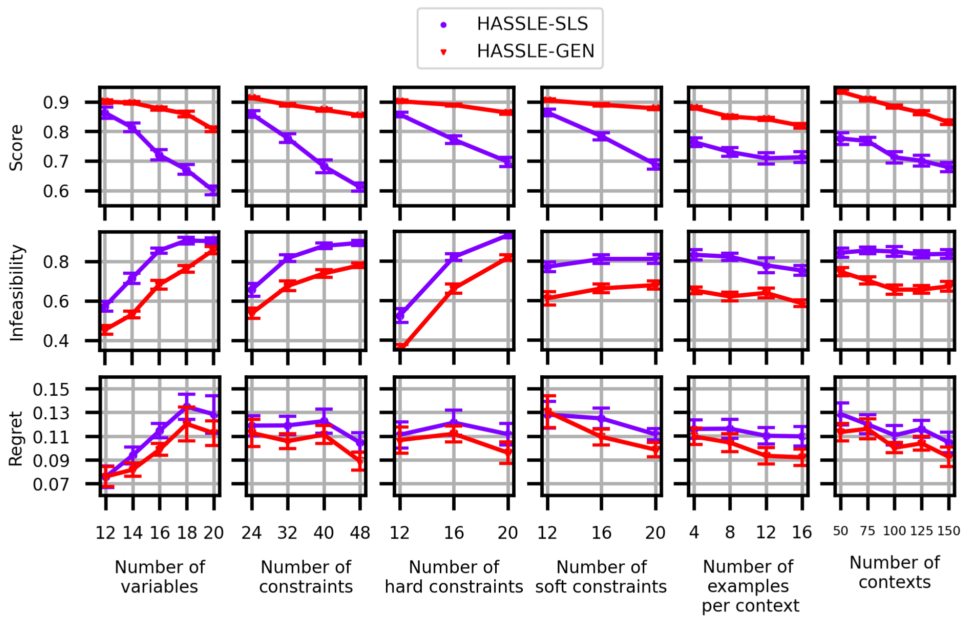


Figure 2 When given the same cutoff time, HASSLE-GEN consistently learns models with a higher score (i.e., training set accuracy) than HASSLE-SLS. This in turn translates into lower infeasibility and lower regret, which means that the models learned by HASSLE-GEN can be used to generate higher-quality solutions than those learned by HASSLE-SLS.

up model evaluation, we proposed a knowledge-compilation-based evaluation procedure. Our experiments showed this procedure to be most useful when learning relatively small MAX-SAT models or when learning from a large set of training data. Second, to reduce the amount of evaluations required to find a good model, we proposed a genetic algorithm named HASSLE-GEN. In the experiments, HASSLE-GEN consistently beat the state of the art on learning problems of various sizes: when given the same amount of training time, it learned models that can be used to generate higher-quality solutions.

One possible direction for future work is to try to speed up model evaluation even further. In our proposed evaluation procedure, an ADD representing the MAX-SAT model is computed in its entirety, to then be used for context-specific inference. However, some parts of the ADD might not be relevant in any context occurring in the training data, and thus do not strictly have to be computed. A 'lazy' decision diagram construction procedure could exploit this fact and lead to even faster evaluation.

Another possible direction is to focus on learning different types of models from examples. One possible extension is to learn maximum satisfiability modulo theories (MAX-SMT) models, where constraints are not limited to disjunctive clauses over binary variables, but can be first-order logic formulas with respect to one or more background theories. Some work has already been done on learning SMT models from examples [14], but learning MAX-SMT models from contextual examples has thus far not been explored. Another possible extension is to learn mixed-integer linear programs (MILP) from contextual data. To this end, recent work [15] has proposed a search strategy that can be seen as a hybrid between stochastic local search and stochastic gradient descent. However, the approach suffers from runtime issues, suggesting that the ideas we proposed here might be of use.

References

1 Roberto Asín Achá and Robert Nieuwenhuis. Curriculum-based course timetabling with SAT and MaxSAT. *Annals of Operations Research*, 218(1):71–91, 2014.

2 R Iris Bahar, Erica A Frohm, Charles M Gaona, Gary D Hachtel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi. Algebraic decision diagrams and their applications. *Formal Methods in System Design*, 10(2):171–206, 1997.

3 Nicolas Beldiceanu and Helmut Simonis. A model seeker: Extracting global constraint models from positive examples. In *International Conference on Principles and Practice of Constraint Programming*, pages 141–157. Springer, 2012.

4 Christian Bessiere, Remi Coletta, Emmanuel Hebrard, George Katsirelos, Nadjib Lazaar, Nina Narodytska, Claude-Guy Quimper, and Toby Walsh. Constraint acquisition via partial queries. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

5 Christian Bessiere, Remi Coletta, Frédéric Koriche, and Barry O'Sullivan. A SAT-based version space algorithm for acquiring constraint satisfaction problems. In *European Conference on Machine Learning*, pages 23–34. Springer, 2005.

6 Christian Bessiere, Remi Coletta, Barry O'Sullivan, Mathias Paulin, et al. Query-driven constraint acquisition. In *IJCAI*, volume 7, pages 50–55, 2007.

7 Paolo Campigotto, Roberto Battiti, and Andrea Passerini. Learning modulo theories for preference elicitation in hybrid domains. *CoRR*, abs/1508.04261, 2015. URL: `http://arxiv.org/abs/1508.04261`, `arXiv:1508.04261`.

8 Luc De Raedt, Andrea Passerini, and Stefano Teso. Learning constraints from examples. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

9 Emir Demirović, Nysret Musliu, and Felix Winter. Modeling and solving staff scheduling with partial weighted maxSAT. *Annals of Operations Research*, 275(1):79–99, 2019.

10 Agoston E Eiben and James E Smith. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

11 David Edward Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Addison-Wesley Pub. Co., 1989.

12 Michael Huth and Mark Ryan. *Logic in computer science: Modelling and reasoning about systems.* Cambridge University Press, 2004.

13 Samuel Kolb. Learning constraints and optimization criteria. In *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

14 Samuel Kolb, Stefano Teso, Andrea Passerini, and Luc De Raedt. Learning SMT(LRA) constraints using SMT solvers. In *IJCAI*, volume 18, pages 2333–2340, 2018.

15 Mohit Kumar, Samuel Kolb, Luc De Raedt, and Stefano Teso. Learning mixed-integer linear programs from contextual examples, 2021. `arXiv:2107.07136`.

16 Mohit Kumar, Samuel Kolb, Stefano Teso, and Luc De Raedt. Learning MAX-SAT from contextual examples for combinatorial optimisation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):4493–4500, Apr. 2020. URL: `https://ojs.aaai.org/index.php/AAAI/article/view/5877`, `doi:10.1609/aaai.v34i04.5877`.

17 Mohit Kumar, Samuel Kolb, Stefano Teso, and Luc De Raedt. Learning MAX-SAT from contextual examples for combinatorial optimisation, 2022. `arXiv:2202.03888`.

18 Chu Min Li and Felip Manya. MaxSAT, hard and soft constraints. In *Handbook of satisfiability*, pages 613–631. IOS Press, 2009.

19 Samir W Mahfoud et al. Crowding and preselection revisited. In *PPSN*, volume 2, pages 27–36. Citeseer, 1992.

20 Patrick Mills and Edward Tsang. Guided local search for solving sat and weighted max-sat problems. *Journal of Automated Reasoning*, 24(1):205–223, 2000.

21 Melanie Mitchell. *An introduction to genetic algorithms.* MIT Press, 1998.

22 Tomasz P Pawlak and Krzysztof Krawiec. Synthesis of mathematical programming constraints with genetic programming. In *European Conference on Genetic Programming*, pages 178–193. Springer, 2017.

23 Émilie Picard-Cantin, Mathieu Bouchard, Claude-Guy Quimper, and Jason Sweeney. Learning parameters for the sequence constraint from solutions. In *International Conference on Principles and Practice of Constraint Programming*, pages 405–420. Springer, 2016.

24 Émilie Picard-Cantin, Mathieu Bouchard, Claude-Guy Quimper, and Jason Sweeney. Learning the parameters of global constraints using branch-and-bound. In *International Conference on Principles and Practice of Constraint Programming*, pages 512–528. Springer, 2017.

25 Nathan Robinson, Charles Gretton, Duc Nghia Pham, and Abdul Sattar. Partial weighted MaxSAT for optimal planning. In *Pacific Rim International Conference on Artificial Intelligence*, pages 231–243. Springer, 2010.

26 Sean Safarpour, Hratch Mangassarian, Andreas Veneris, Mark H Liffiton, and Karem A Sakallah. Improved design debugging using maximum satisfiability. In *Formal Methods in Computer Aided Design (FMCAD'07)*, pages 13–19. IEEE, 2007.

27 Qiang Yang, Kangheng Wu, and Yunfei Jiang. Learning action models from plan examples using weighted MAX-SAT. *Artificial Intelligence*, 171(2-3):107–143, 2007.