On Finding Minimum Satisfying Assignments

Alexey Ignatiev[1,2(✉)], Alessandro Previti[1], and Joao Marques-Silva[1]

[1] LaSIGE, Faculty of Science, University of Lisbon, Lisbon, Portugal
{aignatiev,jpms}@ciencias.ulisboa.pt, apreviti.research@gmail.com
[2] ISDCT SB RAS, Irkutsk, Russia

Abstract. Given a Satisfiability Modulo Theories (SMT) formula, a minimum satisfying assignment (MSA) is a partial assignment of minimum size that ensures the formula is satisfied. Minimum satisfying assignments find a number of practical applications that include software and hardware verification, among others. Recent work proposes the use of branch-and-bound search for computing MSAs. This paper proposes a novel counterexample-guided implicit hitting set approach for computing one MSA. Experimental results show significant performance gains over existing approaches.

1 Introduction

For a propositional formula, represented in conjunctive normal form (CNF), a minimum size prime implicant can be computed with a logarithmic number of calls to a SAT oracle [18], e.g. by solving unweighted partial maximum satisfiability. For arbitrary propositional formulae it is unclear how to solve the problem. Nevertheless, for formulas expressed in decidable fragments of first order logic (FOL), the same problem has been investigated as computing a *minimum satisfying assignment* (MSA), which represents a partial assignment of minimum cost to the formula's variables such that, for any assignment to the remaining variables, the formula is true [8]. The MSA problem finds application in software verification, hardware verification, but also in abductive inference [7]. Recent work identified other uses of MSAs [15,24].

Earlier work proposed a branch-and-bound algorithm for computing MSAs of logic formulas expressed in decidable fragments of FOL, being applied in relatively small-scale test cases. Given the potential for wider use in software verification, but also in abductive inference, and with the goal of targeting more challenging problem instances, this paper investigates alternative approaches for computing MSAs. Concretely, the paper builds on the recent work on implicit hitting sets [5,12–14,16,19,23], and develops an implicit hitting set approach for computing MSAs, where implicit hitting sets are obtained by refining identified counterexamples to the general goal of computing one MSA. The experimental results, obtained on existing but also novel suites of problem instances, show performance gains when compared with the existing branch-and-bound approach.

The paper is organized as follows. Section 2 introduces the notation used throughout the paper. The use of implicit hitting sets for computing MSAs is

M. Rueher (Ed.): CP 2016, LNCS 9892, pp. 287–297, 2016.
DOI: 10.1007/978-3-319-44953-1_19

detailed in Sect. 3, resulting in a new tool referred to as MINT. Section 4 compares the MINT solver with the existing tool MISTRAL [8]. Section 5 concludes the paper and identifies research directions.

2 Preliminaries

This section introduces the concepts used throughout the paper. Standard definitions commonly used in first-order logic apply. We follow [3,8], since we build on this earlier work. The paper assumes basic knowledge of first-order logic, but some basic definitions are presented to make the paper more self-contained. A first-order theory $\mathcal{T}$ is a set of first-order sentences over a signature $\mathcal{S}$, where the signature $\mathcal{S}$ specifies a set of predicates and function constants. A first-order model $\mathcal{M}$ is a pair $\langle \mathcal{U}, \mathcal{I} \rangle$, where the set $\mathcal{U}$ represents a *universe*, and $\mathcal{I}$ represents an *interpretation* that assigns a semantics to every symbol in $\mathcal{S}$. $\mathcal{V}$ denotes the set of variables (which are distinct from $\mathcal{S}$). Given a model $\mathcal{M}$, a *valuation* ω is a partial map from $\mathcal{V}$ to $\mathcal{U}$. For simplicity, we assume $\mathcal{V}$ to be the set of variables that occur free in the formula. In what follows, $\mathcal{F}$ denotes a first-order formula modulo a theory defined over variables $\mathsf{var}(\mathcal{F})$. If ω is a valuation in $\mathcal{V} \rightarrow \mathcal{U}$, we write $\mathcal{M}, \omega \vDash \mathcal{F}$ to indicate that the formula $\mathcal{F}$ is true, according to the usual semantic of first-order logic, in model $\mathcal{M}$, with ω giving the valuation of the free variables in $\mathcal{F}$. We say that $\mathcal{M}$ is a model of $\mathcal{F}$ when every sentence of $\mathcal{F}$ is true in $\mathcal{M}$.

Definition 1. *Formula $\mathcal{F}$ is satisfiable modulo $\mathcal{T}$ when there exists a model $\mathcal{M} = \langle \mathcal{U}, \mathcal{I} \rangle$ of $\mathcal{T}$ and an assignment $\omega \in \mathcal{V} \rightarrow \mathcal{U}$ such that $\mathcal{M}, \omega \vDash \mathcal{F}$. We say that the pair $\langle \mathcal{M}, \omega \rangle$ is a* satisfying assignment (SA) *for $\mathcal{F}$.*

A concept used throughout the paper is that of *partial satisfying assignment*:

Definition 2. *A* partial satisfying assignment *for a formula $\mathcal{F}$ is a pair $\langle \mathcal{M}, \omega \rangle$, where $\mathcal{M}$ is a model and ω is a valuation over $\mathcal{M}$ such that $dom(\omega) \subseteq \mathcal{V}$ and such that for any valuation of $\alpha \in \mathcal{V} \setminus dom(\omega) \rightarrow \mathcal{U}$, we have that $\langle \mathcal{M}, (\omega \cup \alpha) \rangle$ is a satisfying assignment for $\mathcal{F}$.*

Definition 3. *A partial satisfying assignment $\langle \mathcal{M}, \omega \rangle$ for $\mathcal{F}$ is said to be minimal (mSA) if for any valuation $\alpha \in \mathcal{V} \rightarrow \mathcal{U}$ s.t. $dom(\alpha) \subset dom(\omega)$, the pair $\langle \mathcal{M}, \alpha \rangle$ is not a satisfying assignment for $\mathcal{F}$. A Minimum Satisfying Assignment (MSA) is an mSA of smallest size.*

Note that an MSA can be defined more generally in terms of a cost function. In this paper we will refer to MSAs in terms of their size (i.e. the cost associated with each variable is 1). We conclude this section with the definition of a hitting set:

Definition 4. *Given a collection Γ of sets from a universe $\mathbb{U}$, a hitting set h for Γ is a set such that $\forall S \in \Gamma, h \cap S \neq \emptyset$.*

A hitting set is *minimal* when none of its subsets is a hitting set. Let us denote with HS(Γ) the set of all hitting sets of Γ. Then a hitting set $h \in HS(\Gamma)$ is said to be a *minimum* hitting set if $\forall h' \in HS(\Gamma)$ we have that $|h| \leq |h'|$.

3 Computing One MSA with Implicit Hitting Sets

This section proposes an algorithm for computing one MSA by exploiting an implicit hitting set approach, following the ideas of [21] and similar in spirit to related recent work in different areas [5,12–14,16,19,23]. However, in contrast to earlier work related with propositional formulas, our approach exploits implicit hitting sets for selecting sets of variables and not sets of clauses.

The goal of computing one MSA is to find a minimum size set $X \in \mathsf{var}(\mathcal{F})$ such that $\exists_X \forall_Y.\mathcal{F}$ holds, with $Y = \mathsf{var}(\mathcal{F}) \setminus X$. Let us consider the set of sets $\mathcal{J}$, where each set $I \in \mathcal{J}$ represents the complement of one counterexample to our goal, i.e. each set $I \in \mathcal{J}$ is the complement of X_{cex} such that $\exists_{X_{\text{cex}}} \forall_Y.\mathcal{F}$ is false. If a solution is to exist, it must use at least one variable from each set I; otherwise, we would be repeating the complement of set I, namely X_{cex}, and we know there is no solution in that case. Thus, any solution set $X \in \mathsf{var}(\mathcal{F})$ for which $\exists_X \forall_Y.\mathcal{F}$ is true, must *hit* any set $I \in \mathcal{J}$; otherwise, set X would not be a solution. Moreover, we can make each set to hit stronger, by finding a *maximal* counterexample, i.e. by *growing* X_{cex} [14] (and, hence, *reducing* its complement $I \in \mathcal{J}$). Thus, the minimal hitting set duality relation between goal sets X and reduced complements $I \in \mathcal{J}$ of counterexamples X_{cex} can be devised, following the ideas of [21]. In practice, it is unrealistic in many cases to explicitly represent the set of sets $\mathcal{J}$. Thus, the sets to hit are generated on demand, and this explains why the approach is referred to as an *implicit hitting set* approach. The proposed algorithm is depicted in Algorithm 1. The remainder of this section formalizes the approach outlined above.

Definition 5. *Given a formula $\mathcal{F}$ modulo theory $\mathcal{T}$ s.t. formula $\mathcal{F}$ is defined over set of variables $\mathsf{var}(\mathcal{F}) = X \cup Y$ and $\exists_X \forall_Y.\mathcal{F}$ is true, set Y is called a* universal subset (US) *for formula $\mathcal{F}$ and set X is called an* existential subset (ES) *for $\mathcal{F}$.*

Algorithm 1. MSA algorithm

```
input  : Formula $\mathcal{F}$
output: One MSA of $\mathcal{F}$
1  $\mathcal{H} \leftarrow \emptyset$
2  while true do
3      $X \leftarrow \texttt{MinHS}(\mathcal{H})$
4      $Y \leftarrow \mathsf{var}(\mathcal{F}) \setminus X$
5      $(\mathsf{st}, \mu_X) \leftarrow \texttt{Solve}(\exists_X \forall_Y.\mathcal{F})$
6      if st then
7          break
8      else
9          $I \leftarrow \mathsf{var}(\mathcal{F}) \setminus \texttt{Grow}(X, \mathcal{F})$
10         $\mathcal{H} \leftarrow \mathcal{H} \cup I$
11 return MSA $\leftarrow \mu_X$
```

An existential subset X (universal subset Y, resp.) is said to be minimal ES or MinES (maximal US or maxUS, resp.) if $\exists_{X\setminus\{z\}}\forall_{Y\cup\{z\}}.\mathcal{F}$ is false for any $z \in X$. An ES $\mathcal{E}$ of minimum size (US $\mathcal{U}$ of maximum size, resp.) is called a minimum existential subset or MinES (maximum universal subset or MaxUS, resp.). Observe that given an existential subset for formula $\mathcal{F}$ of the smallest size (i.e. a MinES), one can easily extract an MSA for $\mathcal{F}$. Indeed, it assigns variables of the existential subset satisfying $\mathcal{F}$, which can be done by calling a decision procedure for $\mathcal{F}$, i.e. an SMT oracle.

Definition 6. *A falsifying subset (FS) for a formula $\mathcal{F}$ modulo theory $\mathcal{T}$ is a set of variables $Y \subseteq \mathsf{var}(\mathcal{F})$ such that $\exists_X\forall_Y.\mathcal{F}$ is false.*

As usually, we identify with minFS and MinFS the minimal and minimum falsifying set, respectively. We can now introduce an important proposition highlighting the existing duality between the minESes and the minFSes of a formula:

Proposition 1. *Given a formula $\mathcal{F}$, let $minES(\mathcal{F})$ and $minFS(\mathcal{F})$ be the set of all minESes and minFSes of F. Then the following holds:*

1. *A subset $X \subseteq \mathsf{var}(\mathcal{F})$ is a minES for $\mathcal{F}$ iff X is a minimal hitting set of $minFS(\mathcal{F})$.*
2. *A subset $Y \subseteq \mathsf{var}(\mathcal{F})$ is a minFS for $\mathcal{F}$ iff Y is a minimal hitting set of $minES(\mathcal{F})$.*

The intuition[1] for the first statement is the following (a dual argument can be used for the second one). We know that since X is a minES for $\mathcal{F}$ then $\exists_X\forall_Y.\mathcal{F}$ is true. This means that for any minFS $Y' \in minFS(\mathcal{F})$ we have that $Y' \nsubseteq Y$, which implies that for any $Y' \in minFS(\mathcal{F})$ at least one variable of Y' is in X. If X is a minimal hitting set of $minFS(\mathcal{F})$ then at least one variable of every $Y' \in minFS(\mathcal{F})$ is in X. This means that $\exists_X\forall_Y.\mathcal{F}$ is true since $Y' \nsubseteq Y$ for any $Y' \in minFS(\mathcal{F})$.

Proposition 2. *A subset $X \subseteq \mathsf{var}(\mathcal{F})$ is a MinES for $\mathcal{F}$ iff X is a minimum hitting set of $minFS(\mathcal{F})$.*

Proposition 2 enables us to compute an MSA once we have the set $minFS(\mathcal{F})$. However, computing $minFS(\mathcal{F})$ is not always feasible due to the size of the set. Thus, the idea is to compute a subset of $minFS(\mathcal{F})$, from which an MSA can be extracted. However, a minimum hitting set X on a subset of $S \subseteq minFS(\mathcal{F})$ does not necessarily correspond to a MinES for $\mathcal{F}$. For X to be a MinES for $\mathcal{F}$ we need two conditions:

Proposition 3. *Let $Y = \mathsf{var}(\mathcal{F}) \setminus X$. If (1) X is a minimum hitting set of $S \subseteq minFS(\mathcal{F})$ and (2) $\exists_X\forall_Y.\mathcal{F}$ is true, then X is a MinES of $\mathcal{F}$.*

[1] Proofs of Propositions 1 and 2 are omitted here due to lack of space. Note that they can be constructed following the ideas of [12,22] where similar proofs are presented.

Proof. Since $\exists_X \forall_Y . \mathcal{F}$ is true we have that, by definition, X is an ES. We know that an ES is an hitting set of $minFS(\mathcal{F})$ (see Proposition 1). Since X is a *minimum* hitting set of $S \subseteq minFS(\mathcal{F})$ we have that X is also a minimum hitting set of $minFS(\mathcal{F})$. By Proposition 2, it follows that X is a MinES for $\mathcal{F}$. $\square$

Condition (2) of Proposition 3 is checked at line 5 of Algorithm 1 when the SMT oracle is invoked. If this oracle call returns true (line 6) then X is a MinES for $\mathcal{F}$ and, thus, an MSA can be extracted. Otherwise, X is extended[2] (line 9) and its complement (minFS) is added to the set $\mathcal{H}$, which contains the set of minFSes computed so far.

Table 1. MSA computation for $\mathcal{F} = ((a+b \geq 0) \vee (c \leq 0)) \wedge ((a+b \geq 0) \vee (b-a \leq 0))$

`MinHS`($\mathcal{H}$)	`Solve`($\exists_X \forall_Y . \mathcal{F}$)	$I \leftarrow \mathtt{var}(\mathcal{F}) \setminus \mathtt{Grow}(X, \mathcal{F})$	$\mathcal{H} = \mathcal{H} \cup I$
$X \leftarrow \{\emptyset\}$	`false`	$I \leftarrow \{b, c\}$	$\{\{b, c\}\}$
$X \leftarrow \{b\}$	`false`	$I \leftarrow \{a\}$	$\{\{b, c\}, \{a\}\}$
$X \leftarrow \{a, c\}$	`true`	$\{a = 1, c = 0\}$ is an MSA of $\mathcal{F}$	

An example of a run of Algorithm 1 is shown in Table 1. Column 1 contains the set X of variables identified by the minimum hitting set of $\mathcal{H}$. Whenever `Solve`($\exists_X \forall_Y . \mathcal{F}$) returns true, the set X is extended by the `Grow` procedure (column 3) and its complement (minFS) is assigned to I. The last column shows the current $\mathcal{H}$, representing the set of all minFSes computed so far. In the last row, `Solve`($\exists_{\{a,c\}} \forall_{\{b\}} . \mathcal{F}$) returns true, with $a = 1, c = 0$. This means that $\{a, c\}$ is a MinES, and $a = 1, c = 0$ is an MSA.

4 Preliminary Experimental Results

This section evaluates the proposed approach to computing a minimum satisfying assignment. The experiments were performed in Ubuntu Linux on an Intel Xeon E5-2630 2.60 GHz processor with 64 GByte of memory. The time limit was set to 3600 s and the memory limit to 10 GByte.

The proposed approach was implemented in a prototype called MINT (**Mini**mum satisfy**IN**g assignmen**T** extractor). The MINT MSA extractor is written as a Python script, which instruments the interaction between a minimum hitting set enumerator and an SMT solver, as described in Algorithm 1. The minimum hitting set enumerator (see line 3 of Algorithm 1) was implemented as an *incremental* MaxSAT solver[3] following the ideas of [17]. The MaxSAT solver was

[2] The `Grow` procedure is implemented as a sequence of SMT oracle calls, each increasing set X.

[3] Indeed, one can observe that Algorithm 1 requires the minimum hitting set solver to report new hitting sets on demand, i.e. when a new counterexample is detected. This can be done in an incremental fashion [10], e.g. by adding new clauses when necessary and computing new solutions on demand while keeping all the information found during the previous calls.

implemented on top of the well-known SAT solver Glucose 3.3 [1]. The MINT extractor was implemented in the PySMT framework [11], which enables it to use any SMT solver capable of dealing with the theories of input SMT formulas, e.g. Z3 [6], CVC4 [2], Yices2 [9], etc. In contrast, the state-of-the-art MSA solver MISTRAL [8] can work only with formulas in the theory of linear arithmetic over integers (LIA formulas). In the experimental evaluation, CVC4 was used in MINT as a backend SMT solver because CVC4[4] performed reasonably well in the SMT competition SMT-COMP15[5] winning a few benchmark subcategories in the QF_LIA category, and it also supports LIA formulas with quantifiers.

Additionally, an improved version of MINT was implemented, which is referred to as MINT+. The only difference between MINT and MINT+ is that MINT+ tries to bootstrap the main algorithm with sets of size 1 that need to be hit in order to get an MSA. More precisely, the procedure traverses all variables of $y \in \mathsf{var}(\mathcal{F})$ and decides satisfiability of the formula $\forall y.\ \mathcal{F}$. If the formula is unsatisfiable than set $\{y\}$ is a MinFS of $\mathcal{F}$ of size 1, i.e. each MSA of $\mathcal{F}$ necessarily hits it. The bootstrapping procedure is called before running Algorithm 1. To assess the efficiency of MINT and MINT+, they were compared to the state-of-the-art MSA extractor MISTRAL.[6]

4.1 Original Benchmark Instances

To evaluate the performance of MINT and MINT+, we used the benchmark set referred to as *CAV12*, which was proposed and also considered in [8]. According to [8], the benchmark constraints were generated by the program analysis tool Compass (e.g. see [7]). In this setting, MSAs are important for reducing the number of queries, which help users diagnose error reports as real bugs or false alarms. Thus, the size of an MSA greatly affects the quality of queries presented to users (and, hence, also the time spent on debugging users' programs). The total number of variables for these instances varies from 1 to 53 and the total number of instances in the benchmark set is 373.[7] The relative size of MSAs for the CAV12 instances varies from 0 % (i.e. an instance is a tautology) to 100 % (i.e. an MSA necessarily contains all variables) with the average size ≈ 58 %.

The performance of the chosen competitors is shown in Fig. 1. Note that the Y-axis of Fig. 1 is scaled logarithmically. As one can observe, the CAV12 instances are trivial to solve for all the competitors. All competitors spend about 10 s to solve the hardest instances in the benchmark suite. The averate running

[4] https://github.com/CVC4/CVC4.

[5] http://smtcomp.sourceforge.net/2015.

[6] Note that the original distribution of MISTRAL does not have a command-line interface. But one can easily create one since the source code of the tool is available online at https://www.cs.utexas.edu/~tdillig/mistral.

[7] We also tested the proposed approach on the standard SMTLIB benchmarks. However, minimum satisfying assignments for the majority of benchmarks in the QF_LIA category of the SMTLIB benchmarks have trivial minimum satisfying assignments, which contain all variables of the original formula. Therefore, considering these instances makes no sense.

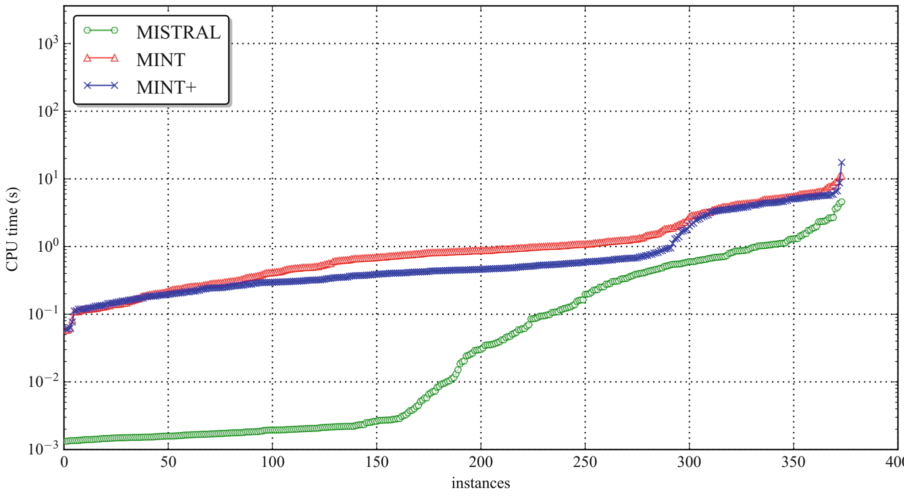


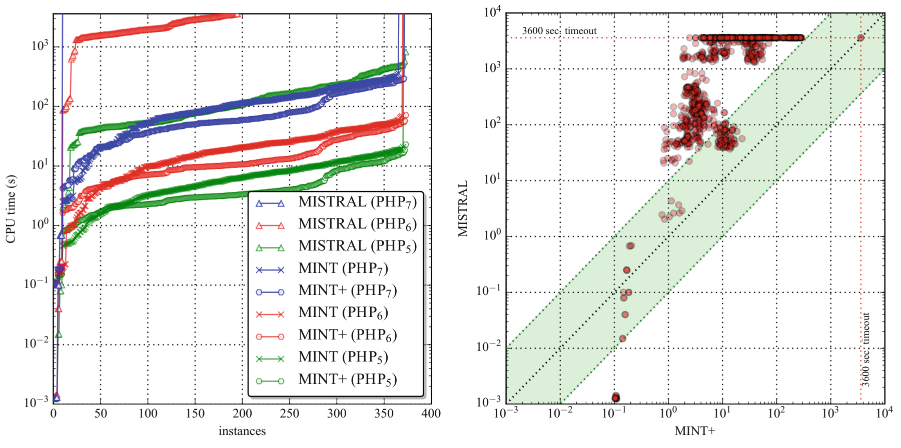
Fig. 1. Performance of MINT, MINT+, and MISTRAL on the CAV12 benchmark instances.

time for MINT, MINT+, and MISTRAL is 1.59 s, 1.23 s and 0.33 s, respectively. A possible explanation of why the new approach is about 0.1 second slower for most of the instances than MISTRAL (it takes 0.1 s vs 0.001 s spent on these instances by MISTRAL) is that it is implemented as a Python script, which requires some time to initialize the Python interpreter environment, while MISTRAL is run as a binary executable written in C++.

4.2 Hardened Benchmarks

The results for the CAV12 benchmark set suggest to consider harder instances in order to conduct a reasonable performance evaluation of the proposed approach. One way to create harder instances is to *combine* the existing CAV12 benchmarks with unsatisfiable formulas that are hard to solve, i.e. for each SMT formula $\mathcal{F}$ (from the CAV12 benchmark set) defined over variables X one needs to consider $\mathcal{F} \vee \mathcal{U}$, where $\mathcal{U}$ is an unsatisfiable formula over variables Y s.t. $X \cap Y = \emptyset$. Notice that, by construction, an MSA of formula $\mathcal{F}$ is also an MSA of $\mathcal{F} \vee \mathcal{U}$, and vice versa. As unsatisfiable components $\mathcal{U}$, we considered well-known families of unsatisfiable formulas, which are proved to be hard to refute by resolution-based reasoning, namely *pigeon-hole principle* formulas PHP_n [20] and formulas GT_n, which are based on the ordering principle that any partial order on a finite set must have a maximal element [4]. Given 373 CAV12 instances $\mathcal{F}$, the following experiments considered n ranging from 5 to 7 for both PHP_n and GT_n resulting in 3 combined benchmark sets $\mathcal{F} \vee PHP_n$ and 3 benchmark sets $\mathcal{F} \vee GT_n$, each also having 373 instances. The translation of CNF formulas PHP_n and GT_n was done in the standard way of encoding CNF formulas into integer linear programming sets of constraints.[8] The number of Boolean variables in formulas PHP_n and GT_n is $n \times (n-1)$, which results in $2 \times n \times (n-1)$ integer variables appearing in the integer linear constraints encoding the original CNF formulas. Thus, the largest number of additional variables in the combination $\mathcal{F} \vee \mathcal{U}$ is 84 (for $n = 7$) and, hence, the largest total number of variables among the constructed benchmarks is 137.

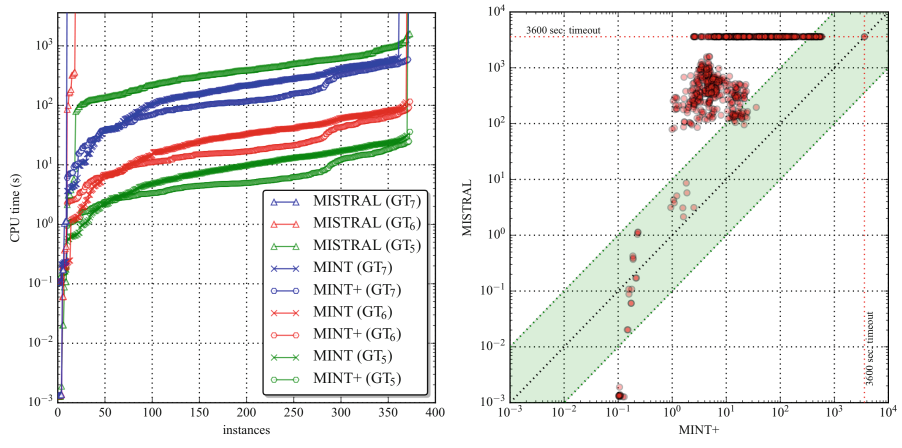
The performance of the considered solvers shown for the $\mathcal{F} \vee PHP_n$ formulas is detailed in Fig. 2. The cactus plot shown in Fig. 2a illustrates how the performance of the competitors changes with the growth of n from 5 to 7. (Again, the Y-axis is scaled logarithmically in the cactus plot shown in Fig. 2a.) Observe that MINT+ and MINT are almost not affected by the unsatisfiable part of the formulas. However, the performance of MISTRAL drops significantly even for $n = 5$ (recall that for the CAV12 benchmarks the average running time of MISTRAL was 0.33 s while for $\mathcal{F} \vee PHP_5$ it is 149.4 s). This tendency towards increasing the running time dramatically for MISTRAL is persistent for $n \in \{6, 7\}$. The number of instances of $\mathcal{F} \vee PHP_6$ and $\mathcal{F} \vee PHP_7$ solved by MISTRAL is 195 and 9, respectively, whereas MINT+ can solve 373 and 371 and MINT solves 369 and 365 instances, respectively. The advantage of MINT+ over MISTRAL is confirmed by the scatter plot shown in Fig. 2b aggregating all instances $\mathcal{F} \vee PHP_n$, $n \in \{5, 6, 7\}$. As detailed in Fig. 3, similar results are shown by the new approach for the $\mathcal{F} \vee GT_n$ formulas while MISTRAL performs even worse (compared to



(a) Cactus plot (b) MINT+ vs MISTRAL

Fig. 2. Performance of MINT, MINT+, and MISTRAL on the $\mathcal{F} \vee PHP_n$ instances.

[8] For each variable x of the original CNF, two integer variables x_+ and x_- are introduced s.t. $0 \le x_+ \le 1$ and $0 \le x_- \le 1$. Variables x_+ and x_- cannot take value 0 or 1 at the same, which is forced by adding constraints of the form $x_+ + x_- = 1$. Each clause $l_1 \vee \ldots \vee l_m$ is translated into constraint $x_{1*} + \ldots + x_{m*} \ge 1$, where each x_{i*} represents either x_{i+} or x_{i-} depending on the polarity of literal l_i.



(a) Cactus plot (b) MINT+ vs MISTRAL

Fig. 3. Performance of MINT, MINT+, and MISTRAL on the $\mathcal{F} \vee GT_n$ instances.

195 $\mathcal{F} \vee PHP_6$ instances solved it solves only 18 instances of $\mathcal{F} \vee GT_6$). This confirms that for harder formulas the new approach significantly overperforms MISTRAL. This clear advantage of the new algorithm over MISTRAL is caused not only by the hardness of the PHP_n and GT_n formulas (since it does not affect the new approach that much) but also by a larger number of variables compared to the original CAV12 benchmark instances, which means that the proposed approach scales better in practice being able to solve harder problem instances.

5 Conclusions

MSAs [8] are generalizations of prime implicants for first-order logic formulas that find applications in a range of practical settings [7,15,24]. Recent work proposed a branch-and-bound approach for computing MSAs. In contrast, this paper proposes the use of an implicit hitting set solution [5,12–14,16,19,23] for computing MSAs. Experimental results, collected on challenging problem instances obtained from those used in earlier work [8], indicate significant performance gains compared to the earlier work [8].

The work described in this paper can be extended in different ways. First, more efficient algorithms for MSA will motivate additional applications and revisiting existing ones. Second, the growing range of uses of implicit hitting sets motivates devising more efficient algorithms.

References

1. Audemard, G., Lagniez, J.-M., Simon, L.: Improving glucose for incremental SAT solving with assumptions: application to MUS extraction. In: Järvisalo, M., Van Gelder, A. (eds.) SAT 2013. LNCS, vol. 7962, pp. 309–317. Springer, Heidelberg (2013)
2. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 171–177. Springer, Heidelberg (2011)
3. Barrett, C.W., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability modulo theories. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability, pp. 825–885. IOS Press (2009)
4. Beame, P., Kautz, H.A., Sabharwal, A.: Towards understanding and harnessing the potential of clause learning. J. Artif. Intell. Res. (JAIR) **22**, 319–351 (2004)
5. Davies, J., Bacchus, F.: Solving MAXSAT by solving a sequence of simpler SAT instances. In: Lee, J. (ed.) CP 2011. LNCS, vol. 6876, pp. 225–239. Springer, Heidelberg (2011)
6. de Moura, L., Bjørner, N.S.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008)
7. Dillig, I., Dillig, T., Aiken, A.: Automated error diagnosis using abductive inference. In: PLDI, pp. 181–192 (2012)
8. Dillig, I., Dillig, T., McMillan, K.L., Aiken, A.: Minimum satisfying assignments for SMT. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 394–409. Springer, Heidelberg (2012)
9. Dutertre, B.: Yices 2.2. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 737–744. Springer, Heidelberg (2014)
10. Eén, N., Sörensson, N.: Temporal induction by incremental SAT solving. Electr. Notes Theor. Comput. Sci. **89**(4), 543–560 (2003)
11. Gario, M., Micheli, A.: PySMT: a solver-agnostic library for fast prototyping of SMT-based algorithms. In: SMT Workshop (2015)
12. Ignatiev, A., Previti, A., Liffiton, M., Marques-Silva, J.: Smallest MUS extraction with minimal hitting set dualization. In: Pesant, G. (ed.) CP 2015. LNCS, vol. 9255, pp. 173–182. Springer, Heidelberg (2015)
13. Janota, M., Marques-Silva, J.: Solving QBF by clause selection. In: IJCAI, pp. 325–331 (2015)
14. Liffiton, M.H., Previti, A., Malik, A., Marques-Silva, J.: Fast, flexible MUS enumeration. Constraints **21**(2), 223–250 (2016)
15. Maity, S., Ghosh, S.K.: Conflict resolution in heterogeneous co-allied MANET: a formal approach. In: Chatterjee, M., Cao, J., Kothapalli, K., Rajsbaum, S. (eds.) ICDCN 2014. LNCS, vol. 8314, pp. 332–346. Springer, Heidelberg (2014)
16. Moreno-Centeno, E., Karp, R.M.: The implicit hitting set approach to solve combinatorial optimization problems with an application to multigenome alignment. OR **61**(2), 453–468 (2013)
17. Morgado, A., Ignatiev, A., Marques-Silva, J.: MSCG: robust core-guided MaxSAT solving. JSAT **9**, 129–134 (2015)
18. Palopoli, L., Pirri, F., Pizzuti, C.: Algorithms for selective enumeration of prime implicants. Artif. Intell. **111**(1–2), 41–72 (1999)
19. Previti, A., Ignatiev, A., Morgado, A., Marques-Silva, J.: Prime compilation of non-clausal formulae. In: IJCAI, pp. 1980–1987 (2015)

20. Razborov, A.A.: Proof complexity of pigeonhole principles. In: Kuich, W., Rozenberg, G., Salomaa, A. (eds.) DLT 2001. LNCS, vol. 2295, pp. 100–116. Springer, Heidelberg (2002)
21. Reiter, R.: A theory of diagnosis from first principles. Artif. Intell. **32**(1), 57–95 (1987)
22. Rymon, R.: An SE-tree-based prime implicant generation algorithm. Ann. Math. Artif. Intell. **11**(1–4), 351–366 (1994)
23. Saikko, P., Wallner, J.P., Järvisalo, M.: Implicit hitting set algorithms for reasoning beyond NP. In: KR (2016)
24. Vigo, R., Nielson, F., Nielson, H.R.: Uniform protection for multi-exposed targets. In: Ábrahám, E., Palamidessi, C. (eds.) FORTE 2014. LNCS, vol. 8461, pp. 182–198. Springer, Heidelberg (2014)