Minimum Satisfying Assignments for SMT*

Isil Dillig[1], Thomas Dillig[1], Kenneth L. McMillan[2], and Alex Aiken[3]

[1] College of William & Mary
[2] Microsoft Research
[3] Stanford University

Abstract. A *minimum satisfying assignment* of a formula is a minimum-cost partial assignment of values to the variables in the formula that guarantees the formula is true. Minimum satisfying assignments have applications in software and hardware verification, electronic design automation, and diagnostic and abductive reasoning. While the problem of computing minimum satisfying assignments has been widely studied in propositional logic, there has been no work on computing minimum satisfying assignments for richer theories. We present the first algorithm for computing minimum satisfying assignments for satisfiability modulo theories. Our algorithm can be used to compute minimum satisfying assignments in theories that admit quantifier elimination, such as linear arithmetic over reals and integers, bitvectors, and difference logic. Since these richer theories are commonly used in software verification, we believe our algorithm can be gainfully used in many verification approaches.

1 Introduction

A *minimum satisfying assignment (MSA)* σ of a formula ϕ, relative to a cost function C that maps variables to costs, is a partial variable assignment that entails ϕ while minimizing C. For example, consider the following formula in linear integer arithmetic:

$$\varphi \ : \ x + y + w > 0 \ \vee \ x + y + z + w < 5 \qquad (*)$$

and suppose that the cost function C assigns cost 1 to each variable in the formula. A partial satisfying assignment to this formula is $x = 1, y = 0, w = 0$. This partial assignment has cost 3, since it uses variables x, y, w, each with cost 1. Another satisfying partial assignment to this formula is $z = 0$; this assignment has cost 1 since it only uses variable z. Furthermore, $z = 0$ is a minimum satisfying assignment of the formula since $z = 0 \models \varphi$ and no other satisfying partial assignment assignment of φ has lower cost than the assignment $z = 0$.

Minimum satisfying assignments have important applications in software and hardware verification, electronic design automation, and diagnostic and abductive reasoning [1–3, 6, 4, 5]. For example, in software verification, minimum

* This work was supported by NSF grants CCF-0915766 and CCF-0702681 and DARPA contract 11635025.

P. Madhusudan and S.A. Seshia (Eds.): CAV 2012, LNCS 7358, pp. 394–409, 2012.

satisfying assignments are useful for classifying and diagnosing error reports [6], for finding reduced counterexample traces in bounded model checking [1], and for minimizing the number of predicates required in predicate abstraction [3].

Some applications, such as [1] have used *minimal* rather than *minimum* satisfying assignments (in the context of propositional logic). A minimal satisfying assignment is one from which no variable can be removed while still guaranteeing satisfaction of the formula. Minimal assignments have the advantage that they can be computed greedily by simply removing variables as long as the formula is implied by the remaining assignment. However, a minimal satisfying assignment can be arbitrarily far from optimal, as the following example shows:

Example 1. Consider again the formula φ from $(*)$. The assignment $x = -1, y = -1, w = -1, z = 5$ is *minimal*, with cost 4. That is, removing the assignment to any one variable allows the formula to be false. However, as observed above, the *minimum* cost is 1.

In this paper, we consider the more difficult problem of computing true minimum-cost assignments. This problem has been studied in propositional logic, where it is commonly known as the *minimum prime implicants* problem [4, 5]. However, there has been no work on computing minimum satisfying assignments in the context of satisfiability modulo theories (SMT).

In this paper, we present the first algorithm for computing minimum satisfying assignments for first-order formulas modulo a theory. The algorithm applies to any theory for which full first-order logic, including quantifiers, is decidable. This includes all theories that admit effective quantifier elimination, such as linear arithmetic over reals, linear integer arithmetic, bitvectors, and difference logic. Since these theories and their combinations are commonly used in software verification, we believe an algorithm for computing minimum satisfying assignments in these theories can be gainfully used in many verification approaches.

This paper makes the following key contributions:

- We define minimum satisfying assignments modulo a theory and discuss some useful properties of minimum satisfying assignments.
- We present a branch-and-bound style algorithm for computing minimum satisfying assignments of SMT formulas.
- We consider improvements over the basic branch-and-bound approach that effectively prune parts of the search space.
- We show how to use and compute theory-satisfiable propositional implicants to obtain a good variable order and initial cost bound.
- We describe how to obtain and use a special class of implicates of the original formula to further prune the search space.
- We present an experimental evaluation of the performance of our algorithm.

2 Minimum Satisfying Assignments and Their Properties

To begin with, let us precisely define the notion of MSA for a formula in first-order logic, modulo a theory. This definition is a bit subtle because, to speak of

an assignment of values to variables in first-order logic, we must specify a *model* of the theory.

For a given theory $\mathcal{T}$, we have a fixed signature $\mathcal{S}$ of predicate and function constants of specified arity. The *theory* $\mathcal{T}$ is a set of first-order sentences over signature $\mathcal{S}$. A first-order model M is a pair $(\mathcal{U},\mathcal{I})$ where the set $\mathcal{U}$ is the *universe*, and $\mathcal{I}$ is the *interpretation* that gives a semantics to every symbol in $\mathcal{S}$. We assume a countable set of *variables* $\mathcal{V}$, distinct from $\mathcal{S}$. Given a model M, a *valuation* σ is a partial map from $\mathcal{V}$ to $\mathcal{U}$. We write $\text{free}(\phi)$ for the set of variables occurring free in formula ϕ. If σ is a valuation in $\text{free}(\phi) \to \mathcal{U}$, we write $M, \sigma \models \phi$ to indicate that formula ϕ is true, according to the usual semantics of first-order logic, in model M, with σ giving the valuation of the free variables in ϕ. We say M is *model of* theory $\mathcal{T}$ when every sentence of $\mathcal{T}$ is true in M.

Definition 1. (Satisfying assignment) *Formula ϕ is* satisfiable modulo $\mathcal{T}$ *when there* exists *a model $M = (\mathcal{U},\mathcal{I})$ of $\mathcal{T}$ and an assignment $\sigma \in \text{free}(\phi) \to \mathcal{U}$ such that $M, \sigma \models \phi$. We say the pair (M, σ) is a* satisfying assignment *for ϕ.*

Our intuition behind an MSA for ϕ is that it gives a valuation for a minimum-cost *subset* of the free variables of ϕ, such that ϕ is true for all valuations of the remaining variables. We capture this idea with *satisfying partial assignments*:

Definition 2. (Satisfying partial assignment) *A* satisfying partial assignment *for formula ϕ is a pair (M, σ), where $M = (\mathcal{U},\mathcal{I})$ is a model, σ is a valuation over M such that* $\text{dom}(\sigma) \subseteq \text{free}(\phi)$*, and such that for every valuation $\rho \in (\text{free}(\phi) \setminus \text{dom}(\sigma)) \to \mathcal{U}$, $(M, \sigma \cup \rho)$ is a satisfying assignment for ϕ.*

The following is an alternate statement of this definition:

Proposition 1. *A* satisfying partial assignment *for formula ϕ is a satisfying assignment for the formula $\forall X.\ \phi$, for some $X \subseteq \text{free}(\phi)$.*

Now, in order to define a *minimum* partial assignment, we introduce a cost function over partial assignments. For a given function $C \in \mathcal{V} \to \mathbb{N}$, the cost of a set of variables X is $C(X) = \Sigma_{v \in X} C(v)$ and the cost of a valuation σ is $C(\sigma) = C(\text{dom}(\sigma))$. Minimum satisfying assignments are now defined as follows:

Definition 3. (Minimum satisfying assignment) *Given a cost function $C \in \mathcal{V} \to \mathbb{N}$, a* minimum satisfying assignment *(MSA) for formula ϕ is a partial satisfying assignment (M, σ) for ϕ minimizing $C(\sigma)$.*

As observed in Proposition 1, a partial satisfying assignment (M, σ) for ϕ is just a satisfying assignment for $\forall X.\phi$. We observe that the free variables in this formula are $\text{free}(\phi) \setminus X$, therefore $\text{dom}(\sigma) = \text{free}(\phi) \setminus X$. Minimizing the cost of σ is thus equivalent to *maximizing* the cost of X such that $\forall X.\ \phi$ is satisfiable. We can formalize this idea as follows:

Definition 4. (Maximum universal subset) *A* universal set *for formula ϕ modulo theory $\mathcal{T}$ is a set of variables X such that $\forall X.\phi$ is satisfiable. For a given cost function $C \in \mathcal{V} \to \mathbb{N}$, a* maximum universal subset *(MUS) is a universal set $X \subseteq \text{free}(\phi)$ maximizing $C(X)$.*

MUS's and MSA's are related by the following theorem:

Theorem 1. *An MSA of formula ϕ for a given cost function C is precisely a satisfying assignment of $\forall X.\ \phi$ for some MUS X.*

Proof. By Proposition 1, a set $X \subseteq \text{free}(\phi)$ is a universal set exactly when there is a partial satisfying assignment (M, σ) for ϕ such $\text{dom}(\sigma) = \text{free}(\phi) \setminus X$, and therefore $C(\sigma) = C(\text{free}(\phi)) - C(X)$. It follows that X is maximum-cost exactly when σ is minimum-cost. □

The following corollary follows immediately from Theorem 1:

Corollary 1. *Let σ be an MSA for formula ϕ, and let X be an MUS of ϕ for cost function C. Then,*

$$C(\sigma) = \left(\sum_{v \in \text{free}(\phi)} C(v) \right) - C(X)$$

Universal sets have some useful properties, derived from the properties of universal quantifiers that will aid us in maximizing their cost. First, as stated by Proposition 2, universal sets are downward-closed. Second, as stated by Proposition 3, universal sets are closed under implications:

Proposition 2. *Given a universal set X for formula ϕ, every $X' \subseteq X$ is also a universal set of ϕ.*

Proposition 3. *If X is a universal set for ϕ and ϕ implies ψ, then X is a universal set for ψ.*

3 A Branch-and-Bound Algorithm for Computing MSAs

To compute minimum satisfying assignments, we first focus on the problem of finding maximum universal subsets. Since we cannot compute MUSs using a greedy approach, we apply a recursive branch-and-bound algorithm for finding maximum universal sets, shown in Figure 1. This algorithm relies on the downward closure property of universal sets (Proposition 2) to bound the search.

The algorithm `find_mus` of Figure 1 takes as input a formula ϕ, a cost function C, a set of candidate variables V, and a lower bound L, and computes a maximum-cost universal set for ϕ that is a subset of V, with cost greater than L. If there is no such subset, it returns the empty set. The lower bound allows us to cut off the search in cases where the best result thus far cannot be improved.

At each recursive call, the algorithm evaluates at line 1 whether the given lower bound can be improved using the available candidate variables. If not, it gives up and returns the empty set. Otherwise, if there are remaining candidates, it chooses a variable x from the candidate set V (line 3) and decides whether the cost of the universal subset containing x is higher than the cost of the universal subset not containing x (lines 4-10).

At lines $4-7$, the algorithm determines the cost of the universal subset containing x. Before adding x to the universally quantified subset, we test whether

```
Requires: φ is satisfiable
find_mus(φ, C, V, L) {
1.  If V = ∅ or C(V) ≤ L  return ∅   /* cannot improve bound */
2.  Set best = ∅
3.  choose x ∈ V

4.  if(SAT(∀x.φ)) {
5.   Set Y = find_mus(∀x.φ, C, V \ {x}, L − C(x));
6.   Int cost = C(Y) + C(x)
7.   If (cost > L) { best = Y ∪ {x}; L = cost }
     }
8.  Set Y = find_mus(φ,C,V \ {x},L);
9.  If (C(Y) > L) { best = Y }

10. return best;
}
```

Fig. 1. Algorithm to compute a maximum universal subset (MUS)

the result is still satisfiable. If not, we give up, since adding more universal quantifiers cannot make the formula satisfiable (the downward closure property of Proposition 2). The recursive call at line 5 computes the maximum universal subset of $\forall x.\phi$, adjusting the cost bound and candidate variables as necessary. Finally, we compute the cost of the universal subset involving x, and if it is higher than the previous bound L, we set the new lower bound to `cost`.

Lines $8-9$ consider the cost of the universal subset not containing x. The recursive call at line 8 computes the maximum universal subset of ϕ, but the current variable x is removed from the candidate variable set. The algorithm compares the costs of the universal subsets with and without x, and returns the subset with the higher cost.

Finally, the algorithm in Figure 2 computes an MSA of ϕ by using `find_mus`. Here, we first test whether ϕ is satisfiable. If so, we compute a maximum universal subset X for ϕ, and return a satisfying assignment for $\forall X.\ \phi$, as described in Theorem 1. This algorithm potentially needs to explore an exponential number of possible universal subsets. However, in practice, the recursion can often be cut off at a shallow depth, either because a previous solution cannot be improved, or because the formula $\forall x.\ \phi$ becomes unsatisfiable, allowing us to avoid branching. Of course, the effectiveness of these pruning strategies depend strongly on the choice of the candidate variable at line 4 as well as the initial bound L on the cost of the MUS. In the following sections, we will describe some heuristics for this purpose that dramatically improve the performance of the algorithm in practice.

```
find_msa(φ, C) {
1.  If φ is unsatisfiable, return ''UNSAT''
2.  Set X = find_mus(φ, C, free(φ), 0)
3.  return a satisfying assignment for ∀X. φ.
}
```

Fig. 2. Algorithm to compute minimum satisfying partial assignment

4 Variable Order and Initial Cost Estimate

The performance of the algorithm described in Section 3 can be greatly improved by computing a good initial lower bound L on the cost of the MUS, and by choosing a good variable order. A good initial cost bound L should be as close as possible to the actual MUS cost in order to maximize pruning opportunities at line 1 of the algorithm from Figure 1. Furthermore, a good variable order should first choose variables x for which $\forall x.\phi$ is unsatisfiable at line 4 of the `find_mus` algorithm, as this choice avoids branching early on and immediately excludes large parts of the search space.

Thus, to improve the algorithm of Section 3, we need to compute a good initial MUS cost estimate as well as a set of variables for which the test at line 4 is likely to be unsatisfiable. Observe that computing an initial MUS cost estimate is equivalent to computing an MSA cost estimate, since these values are related as stated by Corollary 1. Furthermore, observe that if x is not part of an MSA, $\forall x.\phi$ is guaranteed to be satisfiable and the check at line 4 of the algorithm will never avoid branching. Thus, if we choose variables likely to be part of an MSA first, there is a much greater chance we can avoid branching early on.

Therefore, our goal is to compute a partial satisfying assignment σ that is a reasonable approximation for an MSA of the formula. That is, σ should have cost close to the minimum cost, and the variables that are part of σ should largely overlap with variables part of an MSA. If we can compute such a partial assignment σ in a reasonably cheap way, we can use it to both compute the initial lower bound L on the cost of the MUS, and choose a good variable order by considering variables part of σ first.

4.1 Using Implicants to Approximate MSAs

One very simple heuristic to approximate MSAs is to greedily compute a *minimal* satisfying assignment σ for ϕ, and use σ to approximate both the cost and the variables of an MSA. Unfortunately, as discussed in Section 1, minimal satisfying assignments can be arbitrarily far from an MSA and, in practice, do not yield good cost estimates or good variable orders (see Section 7).

In this section, we show how to exploit Proposition 3 to find partial satisfying assignments that are good approximations of an MSA. Recall from Proposition 3 that, if ϕ' implies ϕ (is an *implicant* of ϕ), then a universal set of ϕ' is also a universal set of ϕ. In other words, if ϕ' is an implicant of ϕ, then a partial satisfying assignment of ϕ' is also a partial satisfying assignment of ϕ. Thus, if we can compute an implicant of ϕ with a low-cost partial satisfying assignment, we can use it to approximate both the cost as well as the variables of an MSA.

The question then is, how can we cheaply find implicants of ϕ with high-cost universal sets (correspondingly, low-cost partial satisfying assignments)? To do this, we adapt methods for computing "minimum prime implicants" of propositional formulas [4, 5], and consider implicants that are conjunctions of literals. We define a $\mathcal{T}$*-satisfiable implicant* as a conjunction of literals that

propositionally implies ϕ and is itself satisfiable modulo $\mathcal{T}$. We say a *cube* is a conjunction of literals which does not contain any atom and its negation.

Definition 5. ($\mathcal{T}$-satisfiable implicant) *Let $\mathcal{B}_\phi$ be a bijective function from each atom in $\mathcal{T}$-formula ϕ to a fresh propositional variable. We say that a cube π is a $\mathcal{T}$-satisfiable implicant of ϕ if (i) $\mathcal{B}_\phi(\pi)$ is a propositional implicant of $\mathcal{B}_\phi(\phi)$ and (ii) π is $\mathcal{T}$-satisfiable.*

Of course, for an implicant to be useful for improving our algorithm, it not only needs to be satisfiable modulo theory $\mathcal{T}$, but also needs to have a low-cost satisfying assignment. It would defeat our purpose, however, to optimize this cost. Instead, we will simply use the cost of the free variables in the implicant as a trivial upper bound on its MSA. Thus, we will search for implicants whose free variables have low cost.

Definition 6. (Minimum $\mathcal{T}$-satisfiable implicant) *Given a cost function $C \in \mathcal{V} \to \mathbb{N}$, a minimum $\mathcal{T}$-satisfiable implicant of formula ϕ is a $\mathcal{T}$-satisfiable implicant π of ϕ minimizing $C(\text{free}(\pi))$.*

Example 2. Consider the formula

$$(a + b \geq 0 \vee 2c + d \leq 10) \wedge (a - b \leq 5)$$

For this formula, $a + b \geq 0 \wedge a - b \leq 5$ and $2c + d \leq 10 \wedge a - b \leq 5$ are both $\mathcal{T}$-satisfiable implicants. However, only $a + b \geq 0 \wedge a - b \leq 5$ is a minimum $\mathcal{T}$-satisfiable implicant (with cost 2).

To improve the algorithm from Section 3, what we would like to do is to compute a minimum $\mathcal{T}$-satisfiable implicant for formula ϕ, and use the cost and variables in this implicant as an approximation for those of an MSA of ϕ. Unfortunately, the problem of finding true minimum $\mathcal{T}$-satisfiable implicants subsumes the problem of finding minimum propositional prime implicants, which is already Σ_2^p-complete. For this reason, we will consider a subclass of $\mathcal{T}$-satisfiable implicants, called *monotone implicants*, whose variable cost can be optimized using SMT techniques.

To define monotone implicants, we consider only quantifier-free formulas in negation-normal form (NNF) meaning negation is applied only to atoms. If ϕ is not originally in this form, we assume that quantifier elimination is applied and the result is converted to NNF.

Definition 7. (Minimum $\mathcal{T}$-satisfiable monotone implicant) *Given a bijective map $\mathcal{B}_\phi$ from* literals *of ϕ to fresh propositional variables, let ϕ^+ denote ϕ with every literal l in ϕ replaced by $\mathcal{B}_\phi(l)$. We say a cube π is a* monotone implicant *of ϕ if π^+ implies ϕ^+. A minimum $\mathcal{T}$-satisfiable monotone implicant of ϕ is a monotone implicant that is $\mathcal{T}$-satisfiable and minimizes $C(\text{free}(\pi))$ with respect to a cost function C.*

To see how monotone implicants differ from implicants, consider the formula $\phi = p \vee \neg p$. Clearly TRUE is an implicant of ϕ. However, it is not a monotone

implicant. That is, suppose that $\mathcal{B}_\phi$ maps literals p and $\neg p$ to fresh propositional variables q and r respectively, thus $\phi^+ = q \vee r$. This formula is *not* implied by TRUE. In fact, the only monotone implicants are p and $\neg p$. In general, every monotone implicant is an implicant, but not conversely.

4.2 Computing Minimum $\mathcal{T}$-satisfiable Monotone Implicants

Our goal is to use minimum $\mathcal{T}$-satisfiable monotone implicants to compute a conservative upper bound on MSA cost and guide variable selection order. In this section, we describe a practical technique for computing minimum $\mathcal{T}$-satisfiable monotone implicants. Our algorithm is inspired by the technique of [4] and formulates this problem as an optimization problem.

The first step in our algorithm is to construct a boolean abstraction ϕ^+ of ϕ as described in Definition 7. Observe that this boolean abstraction is different from the standard boolean skeleton of ϕ in that two atoms A and $\neg A$ are replaced with different boolean variables. We note that a satisfying assignment for ϕ^+ corresponds to a monotone implicant of ϕ, provided it is consistent, meaning that it does not assign true to both A and $\neg A$ for some atom A.

After we construct the boolean abstraction ϕ^+, we add additional constraints to ensure that any propositional assignment to ϕ^+ is $\mathcal{T}$-satisfiable. Let $\mathcal{L}$ be the set of literals occurring in ϕ. We add a constraint Ψ encoding theory-consistency of the implicant as follows:

$$\Psi = \bigwedge_{l \in \mathcal{L}} (\mathcal{B}_\phi(l) \Rightarrow l)$$

Note that in particular, this constraint guarantees that any satisfying assignment is consistent. Moreover, it guarantees that the satisfying assignments modulo $\mathcal{T}$ correspond to precisely the $\mathcal{T}$-satisfiable monotone implicants.

Finally, we construct a constraint Ω to encode the cost of the monotone implicant. To do this, we first introduce a fresh cost variable c_x for each variable x in the original formula ϕ. Intuitively, c_x will be set to the cost of x if any literal containing x is assigned to true, and to 0 otherwise. We construct Ω as follows:

$$\Omega = \bigwedge_{l \in \mathcal{L}} \left(\mathcal{B}_\phi(l) \Rightarrow \left(\bigwedge_{x \in \text{free}(l)} c_x = C(x) \right) \right) \wedge \bigwedge_{x \in \text{free}(\phi)} (c_x \geq 0)$$

The first conjunct of this formula states that if the boolean variable representing literal l is assigned to true, then the cost variable c_x for each variable in l is assigned to the actual cost of x. The second conjunct states that all cost variables must have a non-negative value.

Finally, to compute a minimum $\mathcal{T}$-satisfiable monotone implicant, we solve the following optimization problem:

$$\textit{Minimize:} \quad \textstyle\sum_k c_k \quad \textit{subject to} \quad (\phi^+ \wedge \Psi \wedge \Omega)$$

This optimization problem can be solved, for example, using the binary search technique of [7], and the minimum value of the cost function yields the cost

of the minimum $\mathcal{T}$-satisfiable monotone implicant. Similarly, the minimum $\mathcal{T}$-satisfiable monotone implicant can be obtained from an assignment to $(\phi^+ \wedge \Psi \wedge \Omega)$ minimizing the value of the cost function.

5 Using Implicates to Identify Non-universal Sets

Another useful optimization to the algorithm of Section 3 can be obtained by applying the contrapositive of Proposition 3. That is, suppose that we can find a formula ψ that is implied by ϕ (that is, an *implicate* of ϕ). If Y is not a universal set for ψ then it cannot be a universal set for ϕ. This fact can allow us to avoid the satisfiability test in line 4 of Algorithm `find_mus`, since as soon as our proposed universal subset X contains Y, we know that $\forall X.\ \phi$ must be unsatisfiable. To use this idea, we need a cheap way to find implicates of ϕ that have small *non*-universal sets.

To make this problem easier, we will consider only theories $\mathcal{T}$ that are *complete*. This means that all the models of $\mathcal{T}$ are *elementarily equivalent*, that is, they satisfy the same set of first-order sentences. Another way to say this is that $\mathcal{T}$ entails every sentence or its negation. An example of a complete theory is Presburger arithmetic, with signature $\{0, 1, +, \leq\}$. Given completeness, we have the following proposition:

Proposition 4. *Given a complete theory $\mathcal{T}$, and formula ψ, if $\neg\psi$ is satisfiable modulo $\mathcal{T}$, then $\forall \text{free}(\psi).\ \psi$ is unsatisfiable modulo $\mathcal{T}$.*

Proof. Let V be the set of free variables in ψ. Since $\neg\psi$ is satisfiable modulo $\mathcal{T}$, there is a model M of $\mathcal{T}$ such that $M \models \exists V.\neg\psi$. Since $\mathcal{T}$ is complete, it follows that $\exists V.\neg\psi$ is true in all models of $\mathcal{T}$, hence $\forall V.\ \phi$ is unsatisfiable modulo $\mathcal{T}$. □

This means that if we can find an implicate ψ of ϕ, such that $\neg\psi$ is satisfiable, then we can rule out any candidate universal subset for ϕ that contains the free variables of ψ. To find such non-trivial implicates, we will search for formulas of the form $\psi = \psi_1 \Rightarrow \psi_2$, where ψ_1 and ψ_2 are built from sub-formulas of ϕ. The advantage of considering this special class of implicates is that they can be easily derived from the boolean structure of the formula.

Specifically, to derive these implicates, we first convert ϕ to NNF and compute a so-called *trigger* Π for each subformula of ϕ. Triggers of each subformula are defined recursively as follows:

1. For the top-level formula ϕ, $\Pi(\phi) = \mathit{true}$.
2. For a subformula $\phi' = \phi_1 \wedge \phi_2$, $\Pi(\phi_1) = \Pi(\phi')$, and $\Pi(\phi_2) = \Pi(\phi')$.
3. For a subformula $\phi' = \phi_1 \vee \phi_2$, $\Pi(\phi_1) = \Pi(\phi') \wedge \neg\phi_2$, and $\Pi(\phi_2) = \Pi(\phi') \wedge \neg\phi_1$.

Example 3. Consider the formula $x \neq 0 \vee (x + y < 5 \wedge z = 3)$. Here, the triggers for each literal are as follows:

$$\begin{aligned} \Pi(x + y < 5) &= \neg(x \neq 0) \\ \Pi(z = 3) &= \neg(x \neq 0) \\ \Pi(x \neq 0) &= \neg(x + y < 5 \wedge z = 3) \end{aligned}$$

It is easy to see that if l is a literal in formula ϕ with trigger $\Pi(l)$, then ϕ implies $\Pi(l) \Rightarrow l$. Thus, $\Pi(l) \Rightarrow l$ is always a valid implicate of ϕ. However, it is not the necessarily the case that $\neg(\Pi(l) \Rightarrow l)$ is satisfiable. To make sure we only obtain implicates where $\neg(\Pi(l) \Rightarrow l)$ is satisfiable, we first convert ϕ to a simplified form defined in [8]. This representation guarantees that for any trigger $\Pi(l)$ of l, $\neg(\Pi(l) \Rightarrow l)$ is satisfiable. Thus, once a formula ϕ has been converted to simplified form, implicates with satisfiable negations can be read off directly from the boolean structure of the formula without requiring satisfiability checks.

If $\psi = \Pi(l) \Rightarrow l$ is an implicate obtained as described above, we know that no universal subset for ϕ contains free(ψ). Thus, when the last variable in free(ψ) is universally quantified, we can backtrack without checking satisfiability.

6 Implementation

We have implemented the techniques described in this paper in our Mistral SMT solver available at `www.cs.wm.edu/~tdillig/mistral.tar.gz`. Mistral solves constraints in the combined theories of linear integer arithmetic, theory of equality with uninterpreted functions, and propositional logic. Mistral solves linear inequalities over integers using the Cuts-from-Proofs algorithm described in [9], and uses the MiniSAT solver as its SAT solving engine [10].

While the algorithm described in this paper applies to all theories that admit quantifier elimination, our implementation focuses on computing minimum satisfying assignments in Presburger arithmetic (linear arithmetic over integers). To decide satisfiability of quantified formulas in linear integer arithmetic, we use Cooper's technique for quantifier elimination [11]. However, since we expect a significant portion of the universally quantified formulas constructed by the algorithm to be unsatisfiable, we perform a simple optimization designed to detect unsatisfiable formulas: In particular, before we apply Cooper's method, we first instantiate universally quantified variables with a few concrete values. If any of these instantiated formulas are unsatisfiable, we know that the universally quantified formula must be unsatisfiable.

The algorithm presented in this paper performs satisfiability checks on many similar formulas. Since many of these formulas are comprised of the same set of atoms, the SMT solver typically relearns the same theory conflict clauses many times. Thus, to take advantage of the similarity of satisfiability queries, we reuse theory conflict clauses across different satisfiability checks whenever possible.

For computing minimum $\mathcal{T}$-satisfiable implicants, we have implemented the technique described in Section 4, and used the binary search technique described in [7] for optimizing the cost function. However, since finding the actual minimum monotone implicant can be expensive (see Section 7.1), our implementation allows terminating the search for an optimal value after a fixed number of steps. In practice, this results in implicants that are not in fact minimum, but "close enough" to the minimum. This approach is sound because the underlying optimization procedure hill climbs from an initial solution towards an optimal one, and the solution at any step of the optimization procedure can be used as a bound on the cost of a minimum $\mathcal{T}$-satisfiable implicant.

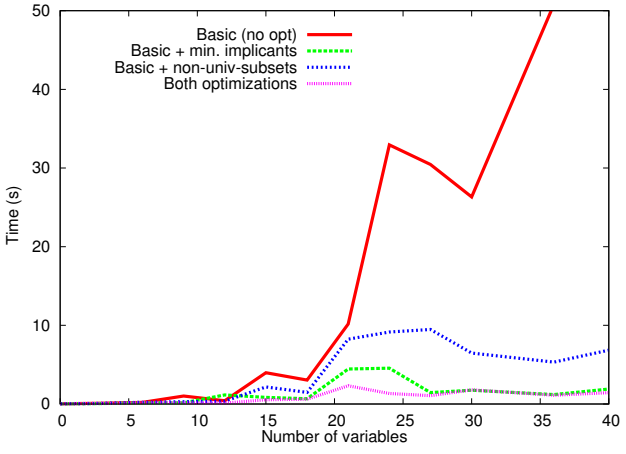


Fig. 3. (# variables in the original formula vs. time to compute MSA in seconds

7 Experimental Results

To evaluate the performance of the algorithm proposed in this paper, we computed minimum satisfying assignments for approximately 400 constraints generated by the program analysis tool Compass [6, 12]. In this application, minimum satisfying assignments are used to compute small, relevant queries that help users diagnose error reports as real bugs or false alarms [6]. In this setting, the number of variables in the satisfying partial assignment greatly affects the quality of queries presented to users. As a result, the time programmers take to diagnose potential errors depends greatly on the number of variables used in the satisfying assignment; thus, computing true minimum-cost assignments is crucial in this setting. The benchmarks we used for our evaluation are available from `www.cs.wm.edu/~tdillig/msa-benchmarks.tar.gz`.

We chose to evaluate the proposed algorithm on the constraints generated by Compass rather than the standard SMTLIB benchmarks for two reasons: First, unlike the constraints we used, SMTLIB benchmarks are not taken from applications that require computing minimum satisfying assignments. Second, the large of majority of benchmarks in the QF_LIA category of the SMTLIB benchmarks contain uninteresting MSAs (containing all or almost all variables in the original formula), making them inappropriate for evaluating an algorithm for computing MSAs.

The constraints we used in our experimental evaluation range in size from a few to several hundred boolean connectives, with up to approximately 40 variables. In our evaluation, we measured the performance of all four versions of the algorithm. The first version, indicated with red in Figures 3 and 4, corresponds to the basic branch-and-bound algorithm described in Section 3. The second version, indicated with green on the graphs, uses the minimum implicant optimization of Section 4. However, as mentioned earlier, since computing

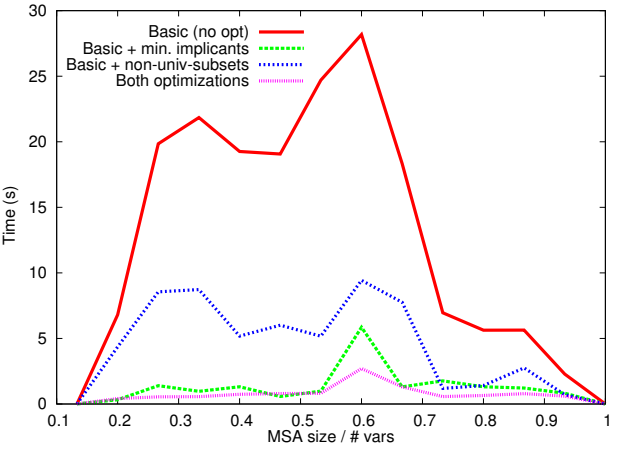


Fig. 4. (# variables in MSA/# of variables in formula) vs. time in seconds

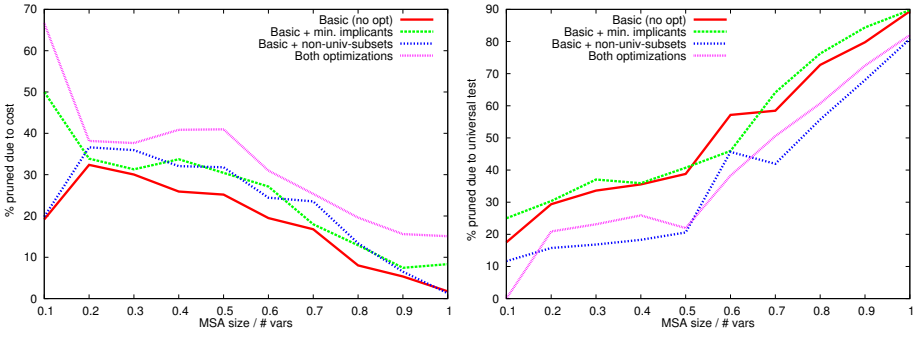
the true minimum implicant can be expensive (see Section 7.1), we only use an approximate solution to the resulting optimization problem. The third version of the algorithm is indicated with blue lines and corresponds to the basic algorithm from Section 3 augmented with the technique of Section 5 for identifying non-universal sets. Finally, the last version of the algorithm using both of the optimizations of Sections 4 and 5 is indicated in the graphs with pink lines.

Figure 3 plots the number of variables in the original formula against running time in seconds for all four versions of the algorithm. As this figure shows, the performance of the basic algorithm is highly sensitive to the number of variables in the original formula and does not seem to be practical for formulas containing more than ~ 18 variables. Fortunately, the improvements described in Sections 4 and 5 have a dramatic positive impact on performance. As is evident from a comparison of the blue, green, and pink lines, the two optimizations of Section 4 and Section 5 complement each other, and we obtain the most performant version of the algorithm by combining both of these optimizations. In fact, the cost of the algorithm using both optimizations seems to grow slowly in the number of variables, indicating that the algorithm should perform well in many settings. However, even using both optimizations, computing MSAs is still much more computationally expensive than deciding satisfiability. On average, computing MSAs is about 25 times as expensive as computing satisfiability on our benchmarks.

Figure 4 plots the fraction

$$\chi = \frac{\#\text{of variables in MSA}}{\#\text{of variables in formula}}$$

against running time in seconds. As this figure shows, if χ is very small (i.e., the MSA is small compared to the number of variables in the formula), the problem of computing minimum satisfying assignments is easy, particularly for



(a) χ vs. % paths pruned due to cost (b) χ vs. % paths pruned due to Prop. 2

Fig. 5. Effectiveness of pruning strategies

the versions of the algorithm using the minimum implicant optimization. Dually, as χ gets close to 1 (i.e., MSA contains almost all variables in the formula, thus few variables can be universally quantified), the problem of computing minimum satisfying assignments again becomes easier. As is evident from the shape of all four graphs in Figure 4, the problem seems to be the hardest for those constraints where χ is approximately 0.6. Furthermore, observe that for constraints with $\chi < 0.6$, the minimum implicant optimization is much more important than the optimization of Section 5. In contrast, the non-universal sets optimization seems to become more important as χ exceeds the value 0.7. Finally, observe that the fully optimized version of the algorithm often performs at least an order of magnitude better than the basic algorithm; at $\chi = 0.6$, the optimized algorithm takes an average of 2.7 seconds, while the basic algorithm takes 28.2 seconds.

Figure 5 explores why we observe a bell-shaped curve in Figure 4. Figure 5(a) plots the value χ against the percentage of all search paths pruned because the current best cost estimate cannot be improved. As this figure shows, the smaller χ is, the more paths can be pruned due to the bound and the more important it is to have a good initial estimate. This observation explains why the minimum implicant optimization is especially important for small values of χ.

In contrast, Figure 5(b) plots the value of χ against the percentage of paths pruned due to the formula ϕ becoming unsatisfiable (i.e., due to Proposition 2). This graph shows that, as the value of χ increases, and thus the MUS's become smaller, more paths are pruned in this way. This observation explains why all versions of the algorithm from Figure 4 perform much better as χ increases.

7.1 Other Strategies to Obtain Bound and Variable Order

In earlier sections, we made the following claims:

1. Computing true minimum-cost implicants is too expensive, but we can obtain a very good approximation to the minimum implicant by terminating the optimizer after a small number of steps
2. *Minimal* satisfying assignments are not useful for obtaining a good cost estimate and variable order

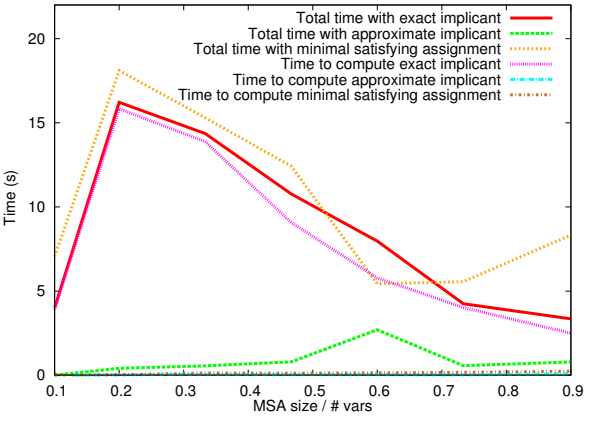


Fig. 6. Comparison of strategies to obtain cost estimate and variable order

In this section, we give empirical data to justify both of these claims.

Figure 6 compares the performance of the algorithm using different strategies to obtain a cost estimate and variable order. As before, the x-axis plots the value of χ and the y-axis is running time in seconds. The red line in this figure shows the total running time of the MSA algorithm using the true minimum-cost monotone implicant. In contrast, the green line shows the total running time of the algorithm using an approximation of the minimum-cost monotone implicant, obtained by terminating the search for the optimum value after a fixed small number of steps. As is evident from this figure, the performance of the algorithm using the true-cost minimum implicant is much worse than the approximately-minimum implicant. This observation is explained by considering the pink line in Figure 6, which plots the time for computing the true minimum-cost monotone implicant. As can be seen by comparing the red and pink lines, the time to compute the true minimum implicant completely dominates the time for the total MSA computation. In contrast, the time to compute the approximate minimum implicant (shown in blue) is negligible, but it is nearly as effective for improving the running time of the MSA algorithm.

We now consider the performance of the algorithm (shown in orange in Figure 6) when we use a *minimal* satisfying assignment to bound the initial cost estimate and choose a variable order. The brown line in the figure shows the time to compute a minimal satisfying assignment. As is clear from Figure 6, the overhead of computing a minimal satisfying assignment is very low, but the performance of the MSA computation algorithm using minimal satisfying assignments is very poor. One explanation for this is that *minimal* satisfying assignments do not seem to be very good approximations for true MSAs. For instance, on average, the cost of a minimal satisfying assignment is 30.6% greater than the cost of an MSA, while the cost of the approximately minimum monotone implicant is only 7.7% greater than the MSA cost. Thus, using minimal satisfying assignments to bound cost and choose a variable order does not seem to be a very good heuristic.

8 Related Work

The problem of computing minimum satisfying assignments in propositional logic is addressed in [4, 13, 5]. All of these approaches formulate the problem of computing implicants as integer linear programming and solve an optimization problem to find a satisfying assignment. Our technique for computing minimum $\mathcal{T}$-satisfiable monotone implicants as described in Section 4 is similar to these approaches. However, we are not aware of any algorithms for computing minimum satisfying assignments in theories richer than propositional logic.

Minimum satisfying assignments have important applications in program analysis and verification. One application of minimum satisfying assignments is finding concise explanations for potential program errors. For instance, recent work [6] uses minimum satisfying assignments for automating error classification and diagnosis using abductive inference. In this context, minimum satisfying assignments are used for computing small, intuitive queries that are sufficient for validating or discharging potential errors. Similarly, the work described in [1] uses minimal satisfying assignments to make model checking tools more understandable to users. In this context, minimal satisfying assignments are used to derive small, reduced counterexample traces that are easily understandable.

Another important application of minimum satisfying assignments in verification is abstraction refinement. One can think of minimum satisfying assignments in this context as an application of Occam's razor: the simplest explanation of satisfiability is the best; thus, minimum satisfying assignments can be used as a guide to choose the most relevant refinements. For instance, the work of Amla and McMillan [14] uses an approximation of minimal satisfying assignments, referred to as *justifications*, for abstraction refinement in SAT-based model checking. Similarly, the work presented in [3] uses minimal satisfying assignments for obtaining a small set of predicates used in the abstraction. However, the results presented in [3] indicate that using minimum rather than minimal satisfying assignments might be more beneficial in this context. In fact, the authors themselves remark on the following: "Another major drawback of the greedy approach is its unpredictability ... Clearly, the order in which this strategy tries to eliminate predicates in each iteration is very critical to its success."

9 Conclusion

In this paper, we have considered the problem of computing minimum satisfying assignments for SMT formulas, which has important applications in software verification. We have shown that MSAs can be computed with reasonable cost in practice using a branch-and-bound approach, at least for a set of benchmarks obtained from software verification problems. We have shown that the search can be usefully bounded by computing implicants with upper-bounded MSAs and implicates with upper-bounded MUS's, provided the cost of obtaining these is low. While our optimizations seem effective, we anticipate that significant improvements are possible, both in the basic algorithm and the optimizations.

Expanding the approach to richer theories is also an interesting research direction. The problem of finding MSA modulo $\mathcal{T}$ is decidable when the satisfiability

modulo $\mathcal{T}$ is decidable in the universally quantified fragment of the logic. This is true for a number of useful theories, including Presburger and bitvector arithmetic. While our approach does not apply to theories that include uninterpreted functions, arrays or lists, this problem may be solved or approximated in practice. In this case, it could be that the notion of partial assignment must be refined, so that the cost metric can take into account the complexity of valuations of structured objects such as arrays and lists.

In summary, we believe that the problem of finding MSAs modulo theories will have numerous applications and is a promising avenue for future research.

References

1. Ravi, K., Somenzi, F.: Minimal Assignments for Bounded Model Checking. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 31–45. Springer, Heidelberg (2004)
2. Marquis, P.: Extending Abduction from Propositional to First-Order Logic. In: Jorrand, P., Kelemen, J. (eds.) FAIR 1991. LNCS, vol. 535, pp. 141–155. Springer, Heidelberg (1991)
3. Chaki, S., Clarke, E., Groce, A., Strichman, O.: Predicate Abstraction with Minimum Predicates. In: Geist, D., Tronci, E. (eds.) CHARME 2003. LNCS, vol. 2860, pp. 19–34. Springer, Heidelberg (2003)
4. Silva, J.: On computing minimum size prime implicants. In: International Workshop on Logic Synthesis, Citeseer (1997)
5. Pizzuti, C.: Computing prime implicants by integer programming. In: IEEE International Conference on Tools with Artificial Intelligence, pp. 332–336. IEEE (1996)
6. Dillig, I., Dillig, T., Aiken, A.: Automated error diagnosis using abductive inference. In: PLDI (2012)
7. Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R., Stenico, C.: Satisfiability Modulo the Theory of Costs: Foundations and Applications. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 99–113. Springer, Heidelberg (2010)
8. Dillig, I., Dillig, T., Aiken, A.: Small Formulas for Large Programs: On-Line Constraint Simplification in Scalable Static Analysis. In: Cousot, R., Martel, M. (eds.) SAS 2010. LNCS, vol. 6337, pp. 236–252. Springer, Heidelberg (2010)
9. Dillig, I., Dillig, T., Aiken, A.: Cuts from Proofs: A Complete and Practical Technique for Solving Linear Inequalities over Integers. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 233–247. Springer, Heidelberg (2009)
10. Sörensson, N., Een, N.: Minisat v1. 13-a sat solver with conflict-clause minimization. In: SAT 2005, p. 53 (2005)
11. Cooper, D.: Theorem proving in arithmetic without multiplication. Machine Intelligence 7(91-99), 300 (1972)
12. Dillig, I., Dillig, T., Aiken, A.: Precise reasoning for programs using containers. POPL 46(1), 187–200 (2011)
13. Manquinho, V., Flores, P., Silva, J., Oliveira, A.: Prime implicant computation using satisfiability algorithms. In: ICTAI, pp. 232–239 (1997)
14. Amla, N., McMillan, K.L.: Combining Abstraction Refinement and SAT-Based Model Checking. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 405–419. Springer, Heidelberg (2007)