Minimal Sets over Monotone Predicates in Boolean Formulae

Joao Marques-Silva[1,2], Mikoláš Janota[2], and Anton Belov[1]

[1] CASL, University College Dublin, Ireland
[2] IST/INESC-ID, Technical University of Lisbon, Portugal

Abstract. The importance and impact of the Boolean satisfiability (SAT) problem in many practical settings is well-known. Besides SAT, a number of computational problems related with Boolean formulas find a wide range of practical applications. Concrete examples for CNF formulas include computing prime implicates (PIs), minimal models (MMs), minimal unsatisfiable subsets (MUSes), minimal equivalent subsets (MESes) and minimal correction subsets (MCSes), among several others. This paper builds on earlier work by Bradley and Manna and shows that all these computational problems can be viewed as computing a minimal set subject to a monotone predicate, i.e. the MSMP problem. Thus, if cast as instances of the MSMP problem, these computational problems can be solved with the same algorithms. More importantly, the insights provided by this result allow developing a new algorithm for the general MSMP problem, that is asymptotically optimal. Moreover, in contrast with other asymptotically optimal algorithms, the new algorithm performs competitively in practice. The paper carries out a comprehensive experimental evaluation of the new algorithm on the MUS problem, and demonstrates that it outperforms state of the art MUS extraction algorithms.

1 Introduction

The theoretical and practical significance of Boolean Satisfiability (SAT) cannot be overstated. This is illustrated by the ever increasing number of practical applications of SAT solvers (see [23,16] for recent overviews). Besides SAT, other computational problems related with Boolean formulas are of interest, both from theoretical and practical perspectives. These include computing prime implicates (PIs) (given an original implicate), minimal models (MMs), minimal unsatisfiable subsets (MUSes), minimal equivalent (or irredundant) subsets (MESes), and minimal correction subsets (MCSes), among several others. Some of these problems find applications in verification. For example, prime implicates have

* This work is partially supported by SFI grant BEACON (09/IN.1/I2618), by FCT grants ATTEST (CMU-PT/ELE/0009/2009) and POLARIS (PTDC/EIA-CCO/123051/2010), and by INESC-ID multiannual funding from the PIDDAC program funds.

been used in inductive strengthening [8,9], whereas MUSes and MCSes find application in proof-based abstraction [26] and counterexample-guided abstraction refinement [1]. Besides verification, the range of applications include knowledge representation [12], non-monotonic reasoning [25], and description logics [30].

Over the years, many different algorithms have been proposed for the above computational problems. Thus, there are dedicated algorithms for computing PIs [8,9,24], MUSes [5,17], MCSes [2,21,15,27], among others. The approach for computing PIs proposed in [8,9] is described using a general framework for computing a minimal set subject to a monotone predicate. We refer to this problem as the *minimal set over a monotone predicate* (MSMP) problem. This paper exploits this insight [8,9], and shows that all of the above computational problems (i.e. PIs, MMs, MUSes, MCSes, etc.) can be represented as instantiations of the MSMP problem. This result immediately implies that algorithms for MSMP can be used for solving *any* of the computational problems listed above, including PIs, MMs, MUSes, MCSes, among others. In addition, existing algorithms for any of these computational problems can be abstracted to the general framework of the MSMP problem. More importantly, the insights provided by these observations allow developing a new algorithm for the MSMP, that is in some sense optimal (i.e. it is asymptotically as efficient as the most efficient algorithms), but that performs well in practice (in contrast with other optimal algorithms developed for specific computational problems, e.g. PIs and MUSes [19,8,9]). This new algorithm is specialized for the case of MUS extraction, thus illustrating how existing pruning techniques (and new ones) can be integrated into the general MSMP algorithm. Experimental results, obtained on well-known practically-relevant problem instances, demonstrate that the new algorithm outperforms the current state of the art MUS extraction algorithm [5,6].

The paper is organized as follows. The next section introduces the definitions used throughout the paper. Section 4 presents the MSMP problem, and shows that the computational problems listed above can be formulated as instantiations of the MSMP problem. Afterwards, Section 5 develops a new algorithm for the MSMP problem (and so it is applicable to PIs, MMs, MUSes, MCSes, etc.). Section 6 presents and analyzes the experimental results for the case of MUS extraction. Finally, Section 7 concludes the paper.

2 Preliminaries

This section briefly introduces the definitions used throughout. Additional standard definitions can be found elsewhere (e.g. [16,23,7]). Boolean formulas are represented in calligraphic font, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{U}$, $\mathcal{T}$, etc. A Boolean formula in conjunctive normal form (CNF) is defined as a finite set of finite sets of literals. Where appropriate, a CNF formula will also be understood as a conjunction of disjunctions of literals. The variables of formula $\mathcal{F}$ are denoted by $\mathsf{var}(\mathcal{F})$. An assignment is a map $\mu : \mathsf{var}(\mathcal{F}) \to \{0, 1\}$. A clause is satisfied by an assignment if one of its literals is assigned value 1. A model of $\mathcal{F}$ is an assignment that satisfies all clauses in $\mathcal{F}$.

When manipulating CNF formulas, it will often be necessary to consider the clauses in a given range. Given CNF formula $\mathcal{F}$ and range $i..j$, where $\mathcal{F}$ is viewed as a sequence $\langle c_1, c_2, \ldots, c_{|\mathcal{F}|} \rangle$, the notation $\mathcal{F}_{i..j}$ represents the clauses in the range $i..j$, i.e. $\{c_i, c_{i+1}, \ldots, c_j\}$ for $j \geq i$, or $\emptyset$ if $j < i$. This same notation will be used for other sets. The following definitions will be used throughout [10,8,21,5].

Definition 1 (Prime Implicate given Implicate). *A prime implicate π of $\mathcal{F}$ given an implicate c is a minimal subset of the literals in c such that $\mathcal{F} \models \pi$.*

Definition 2 (Minimal Model). *A minimal model is a model μ of $\mathcal{F}$ such that the set of true variables is minimal with respect to set containment.*

Definition 3 (MU). *$\mathcal{F}$ is* Minimally Unsatisfiable *(MU) iff $\mathcal{F}$ is unsatisfiable and $\forall_{c \in \mathcal{F}}, \mathcal{F} \setminus \{c\}$ is satisfiable.*

Definition 4 (MUS). *$\mathcal{M}$ is a* Minimally Unsatisfiable Subformula *(MUS) of $\mathcal{F}$ iff $\mathcal{M} \subseteq \mathcal{F}$ and $\mathcal{M}$ is minimally unsatisfiable.*

Definition 5 (MCS). *$\mathcal{C} \subseteq \mathcal{F}$ is a* Minimal Correction Subset *(MCS) iff $\mathcal{F} \setminus \mathcal{C}$ is satisfiable and $\forall_{c \in \mathcal{C}}, \mathcal{F} \setminus (\mathcal{C} \setminus \{c\})$ is unsatisfiable.*

The definition of other computational problems, mentioned in the paper but not explicitly addressed, can be found in the references [21,5,4]. Finally, although the paper focuses on CNF formulas, the results can be extended to disjunctive normal form (DNF) formulas, provided the computational problems of interest are modified accordingly.

3 Related Work

This section overviews work on PIs (subject to an implicate), MMs, MUSes, and MCSes. There is a large body of work on computing prime implicates, e.g. see [24] for an overview. However, the problem this paper addresses focuses on computing a prime implicate starting from an implicate. This problem is studied in [8,9] (also see references therein). A key insight of Bradley&Manna's work is in representing the problem of computing a prime implicate in terms of computing a minimal set subject to a monotone predicate. This insight is extensively used in our work. Another contribution of [8,9] is an optimal (when there exists a single unique minimal set) algorithm for computing a prime implicate. As highlighted later, this optimal algorithm for computing a prime implicate corresponds to the QuickXPlain algorithm for MUS extraction [19]. Minimal models find a wide range of applications in Artificial Intelligence. A concrete example is non-monotonic reasoning [25]. A recent example of applying minimal models is [31].

Recent years have seen a large body of work on computing MUSes (see [5,17] and references therein). A wealth of algorithms have been proposed, of which the most efficient in practice is the so-called *hybrid* algorithm [22,5]. Essential to modern algorithms are techniques to reduce the number of calls to a SAT oracle. The most effective are clause set refinement [14,22] and model rotation [22,5].

A theoretically optimal algorithm for MUS extraction is the QuickXPlain algorithm [19], which in practice performs poorly on CNF formulas. MCSes find a large number of applications, an example of which is in using hitting set duality for enumerating MUSes [28,21]. Recent algorithms for computing MCSes include the use of Maximum Satisfiability [21], iterative clause analysis [27], and a modified QuickXPlain algorithm [15].

To our best knowledge, there is no work relating these computational problems (and the ones mentioned in Section 1), and showing that all can be solved with the same algorithms. This is the focus of the next section.

4 Minimal Sets over Monotone Predicates

This section introduces the minimal model subject to a monotone predicate (MSMP) problem, using the framework developed in [8,9], and shows that several computational problems on CNF formulas can be mapped to the MSMP problem. This section also illustrates how some of the existing MUS extraction algorithms can be adapted to the MSMP problem. Finally, the section analyzes how well-known pruning techniques used in MUS extraction can be used in algorithms for the MSMP problem.

4.1 The General Framework

This section revisits the approach presented in [8,9]. Let $\mathcal{F}$ be a CNF formula. Moreover, let $\mathcal{R}$ be a set of elements (in some way related with $\mathcal{F}$), i.e. the *reference set*. A predicate $p : 2^{\mathcal{R}} \rightarrow \{0, 1\}$, defined on $\mathcal{R}$, is said to be *monotone* if it has the following properties:

1. $p(\mathcal{R})$ holds.
2. If $p(\mathcal{R}_0)$ holds, and $\mathcal{R}_0 \subseteq \mathcal{R}_1 \subseteq \mathcal{R}$, then $p(\mathcal{R}_1)$ also holds.

As shown below, the set of elements $\mathcal{R}$ can represent different objects related with $\mathcal{F}$, e.g. set of clauses or literals.

Definition 6. *(MSMP) The* Minimal Set over a Monotone Predicate *(MSMP) problem consists in finding a subset $\mathcal{M}$ of $\mathcal{R}$ such that $p(\mathcal{M})$ holds, and for any $\mathcal{M}' \subset \mathcal{M}$, $p(\mathcal{M}')$ does not hold, i.e. $\mathcal{M}$ is minimal.*

In [8,9], Bradley&Manna show that the problem of computing a prime implicate (from an existing implicate) can be represented as an instantiation of the MSMP problem. In addition, an algorithm for the MSMP problem is proposed, which is argued to be optimal (at least for the case when there exists one minimal set).

In this section we show that several other computational problems can be cast as instances of the MSMP problem.

Theorem 1. *Given a CNF formula $\mathcal{F}$, there exists an instantiation of the MSMP problem for each of the following problems:*

Table 1. Mappings to the MSMP Problem

	$\mathcal{R}$	$p(\mathcal{W})$, $\mathcal{W} \subseteq \mathcal{R}$
PI given c	$\{l \,\vert\, l \in c\}$	$\neg\mathsf{SAT}(\mathcal{F} \land \land_{l\in\mathcal{W}}\neg l)$
MM	$\mathsf{var}(\mathcal{F})$	$\mathsf{SAT}(\mathcal{F} \land \land_{x\in\mathsf{var}(\mathcal{F})\setminus\mathcal{W}}(\neg x))$
MUS	$\mathcal{F}$	$\neg\mathsf{SAT}(\mathcal{W})$
MCS	$\mathcal{F}$	$\mathsf{SAT}(\mathcal{F} \setminus \mathcal{W})$

1. *Computing a prime implicate (PI) of $\mathcal{F}$ given an implicate c of $\mathcal{F}$.*
2. *Computing a minimal model (MM) of $\mathcal{F}$.*
3. *Computing a minimal unsatisfiable subset (MUS) of $\mathcal{F}$.*
4. *Computing a minimal correction subset (MCS) of $\mathcal{F}$.*

Proof. Let $\mathcal{F}$ be a CNF formula. We consider each case separately.
1. For the case of a prime implicate of $\mathcal{F}$ given implicate c, the reference set is $\mathcal{R} = \{l \,\vert\, l \in c\}$. For $\mathcal{W} \subseteq \mathcal{R}$, define $p(\mathcal{W}) \triangleq \neg\mathsf{SAT}(\mathcal{F} \land \land_{l\in\mathcal{W}}\neg l)$. Clearly, $p(\mathcal{R})$ holds, because c is an implicate of $\mathcal{F}$. A minimal set $\mathcal{M} \subseteq \mathcal{R}$ such that $p(\mathcal{M})$ holds is a prime implicate of $\mathcal{F}$.
2. For the case of a minimal model, the reference set is $\mathcal{R} = \mathsf{var}(\mathcal{F})$. For $\mathcal{W} \subseteq \mathcal{R}$, define $p(\mathcal{W}) \triangleq \mathsf{SAT}(\mathcal{F} \land \land_{x\in\mathsf{var}(\mathcal{F})\setminus\mathcal{W}}(\neg x))$. Clearly, $p(\mathcal{R})$ holds if $\mathcal{F}$ is satisfiable, because no negated literals will be added to the formula. A minimal set $\mathcal{M} \subseteq \mathcal{R}$ such that $p(\mathcal{M})$ holds is a minimal model of $\mathcal{F}$. Observe that $\mathsf{var}(\mathcal{F}) \setminus \mathcal{M}$ represents a maximal set of literals that can be assigned value 0 while still satisfying the formula, and so the remaining literals represent a minimal model.
3. For the case of an MUS, the reference set is $\mathcal{R} = \mathcal{F}$ (or a subset known to be unsatisfiable). For $\mathcal{W} \subseteq \mathcal{R}$, define $p(\mathcal{W}) \triangleq \neg\mathsf{SAT}(\mathcal{W})$. Clearly, $p(\mathcal{R})$ holds if $\mathcal{F}$ is unsatisfiable. A minimal set $\mathcal{M} \subseteq \mathcal{R}$ such that $p(\mathcal{M})$ holds is an MUS of $\mathcal{F}$, since any subset will be satisfiable.
4. For the case of an MCS, the reference set is $\mathcal{R} = \mathcal{F}$. For $\mathcal{W} \subseteq \mathcal{R}$, define $p(\mathcal{W}) \triangleq \mathsf{SAT}(\mathcal{F} \setminus \mathcal{W})$. Clearly, $p(\mathcal{R})$ holds, because removing all clauses from a clause makes the (empty) formula satisfiable. A minimal set $\mathcal{M} \subseteq \mathcal{R}$ such that $p(\mathcal{M})$ holds is an MCS of $\mathcal{F}$, because then $\mathcal{F} \setminus \mathcal{M}$ is satisfiable, and for any $\mathcal{M}' \subset \mathcal{M}$, $\mathcal{F} \setminus \mathcal{M}'$ is unsatisfiable, particularly when $\mathcal{M}' = \mathcal{M} \setminus \{c\}$ for $c \in \mathcal{M}$.
To conclude the proof we note that the monotonicity of the predicate p in all cases above follows from the basic properties of CNF formulas and their models. □

Table 1 summarizes the mappings of the above computational problems to the MSMP problem. Theorem 1 can easily be extended to other computational problems. Concrete examples are minimal equivalent (or irredundant) subsets (MESes) and minimal distinguishing subsets (MDSes) [4]. Other immediate extensions are the problems studied above but for the case where clauses are handled as groups [21,26], e.g. group MUS, group MES, etc.

Algorithm 1: QUICKXPLAIN algorithm for the MSMP problem

Input: $\mathcal{B}$; $\mathcal{T}$, *has_set*
Output: Elements in the minimal set

if *has_set* $\land\, p(\mathcal{B})$ **then return** $\emptyset$
if $|\mathcal{T}| = 1$ **then return** $\mathcal{T}$
$m \leftarrow \lfloor \frac{|\mathcal{T}|}{2} \rfloor$
$(\mathcal{T}_1, \mathcal{T}_2) \leftarrow (\mathcal{T}_{1..m}, \mathcal{T}_{m+1..|\mathcal{T}|})$
$\mathcal{M}_2 \leftarrow \text{QUICKXPLAIN}(\mathcal{B} \cup \mathcal{T}_1, \mathcal{T}_2, |\mathcal{T}_1| > 0)$
$\mathcal{M}_1 \leftarrow \text{QUICKXPLAIN}(\mathcal{B} \cup \mathcal{M}_2, \mathcal{T}_1, |\mathcal{M}_2| > 0)$
return $\mathcal{M}_1 \cup \mathcal{M}_2$

4.2 Algorithms for MSMP

Given the results of the previous section, and the algorithms proposed over the years for each of the above problems, one can conclude that many algorithms can be developed for the MSMP problem by adapting any existing algorithm to the framework of computing a minimal set over a monotone predicate. Clearly, some algorithms proposed for different problems correspond to the same algorithm in this framework. For example, the optimal algorithm proposed for Bradley&Manna for computing a prime implicate [8,9] corresponds to the well-known QUICKXPLAIN algorithm [19] for MUS extraction.

Given the wealth of algorithms proposed for MUS extraction [5,17], one can develop the following types of algorithms for the MSMP problem: (i) Insertion-based [13,32,22]; (ii) Deletion-based [11,3,22]; (iii) Dichotomic [18,17]; and (iv) QUICKXPLAIN [19]. Let m is the number of elements in the initial set of elements, and k is the size of the largest minimal set. In terms predicate tests, insertion-based algorithms require a number of tests that ranges from $\mathcal{O}(m)$ [22] to $\mathcal{O}(m\,k)$ [13,32]. Deletion-based algorithms require $O(m)$ predicate tests. Dichotomic algorithms require $\mathcal{O}(k \log m)$ predicate tests. In the context of computing a prime implicate, [8,9] develop a QUICKXPLAIN-like algorithm. Algorithm 1 presents QUICKXPLAIN [19] adapted to the MSMP problem. $\mathcal{B}$ and $\mathcal{T}$ denote sets of elements, and the algorithm minimizes $\mathcal{T}$ with respect to a base $\mathcal{B}$. The initial values for $\mathcal{B}$ and $\mathcal{T}$ are, respectively, the $\emptyset$ and the original set of elements. The QUICKXPLAIN algorithm for MSMP iteratively splits the target set of elements ($\mathcal{T}$), and recursively calls itself. The predicate is tested when there exists another set of elements besides the current target. If the predicate is true, then the current target set is irrelevant and can be discarded. The base case corresponds to target sets of size 1, in which case the set is returned. As shown independently in [19] and in [8,9], the asymptotic number of predicate tests is $\mathcal{O}(k + k \log \frac{m}{k})$.

Algorithm 2: QuickXplain_CR MSMP with certificate refinement

Input: $\mathcal{B}$; $\mathcal{T}$, *has_set*
Output: $\mathcal{M}$; $\mathcal{C}$

```
if has_set then
    (st, C) = p(B)
    if st then return (∅, C)
if |T| = 1 then return (T, B)
m ← ⌊|T|/2⌋
(T1, T2) ← (T1..m, Tm+1..|T|)
(M2, C1) ← QuickXplain_CR(B ∪ T1, T2, |T1| > 0)
(M1, C2) ← QuickXplain_CR(B ∪ M2, T1 ∩ C1, |M2| > 0)
return (M1 ∪ M2, (C1 ∪ C2) ∩ B)
```

4.3 Pruning Predicate Tests

Recent work on MUS extraction is characterized by the development of several techniques to reduce the number of calls to a SAT oracle. Examples include clause set refinement [14], redundancy removal [32] and model rotation [22,5,33]. Regarding these techniques, we state a few results for clause set refinement and model rotation, without proof; due to space restrictions this is beyond the scope of this paper. Clause set refinement for the MSMP problem can be used for PIs (given implicate c) and MUSes. For the case of MMs and MCSes, the equivalent notion is to keep only the clauses falsified by models. We refer to the generalized concept as *certificate refinement*. Certificate refinement requires the predicate test to return a certificate for the tested property being true. More precisely, $p(S)$ returns a pair $(\mathtt{st}, \mathcal{C})$ where $\mathtt{st}$ is true iff the tested property holds, and, $\mathcal{C} \subseteq S$ is such that that the property holds for $\mathcal{C}$ whenever $\mathtt{st}$ is true. If the property consists in CNF *unsatisfiability* (e.g. PIs and MUSes), then the certificate is an unsatisfiable core. In contrast, for CNF *satisfiability* (e.g. MMs and MCSes) the certificate is a set of falsified clauses.

Model rotation can be used for the PIs and MUSes. There is no equivalent concept for MMs and MCSes. The use of certificate refinement is illustrated for the case of QuickXplain in Algorithm 2. Although the algorithm may seem rather asymmetric in how it handles certificate refinement, recursion guarantees that certificates are used in most of the partitions made.

For the concrete cases of MUS extraction and PI computation, model rotation can also be integrated into the QuickXplain algorithm. Whenever the predicate does not hold (i.e. formula $\mathcal{B}$ is satisfiable), if the number of clauses in $\mathcal{T}$ is 1, or if the number of falsified clauses is 1, then a transition clause has been identified, and so model rotation can be applied. As shown in Section 6 for the case of MUSes, clause set refinement and model rotation serve to improve the basic algorithm. However, for the case of MUS extraction, QuickXplain performs significantly worse than most of the other algorithms considered.

5 Progression-Based Algorithms

This section describes a new algorithm for the MSMP problem, which is also specialized for the case of MUS extraction. Recent work on MUS extraction showed that the most efficient algorithms are based on iteratively deciding with a SAT oracle whether each clause is in an MUS. This approach is referred to as deletion-based [11] MUS extraction and (more recently) as the hybrid approach [22]. In addition, a number of techniques are used to reduce the number of calls to a SAT oracle, namely clause set refinement [14], redundancy removal [32] and model rotation [22,5,33]. However, algorithms that in the worst case analyze all clauses are *not* optimal. As illustrated by the work of Junker [19] with the QuickXPlain algorithm for MUSes and Bradley&Manna [8,9] on PIs, algorithms can be developed that guarantee better worst-case asymptotic performance in terms of the number of SAT solver calls. Unfortunately, these algorithms perform poorly in practice. As shown by recent results [5,6], the algorithms with good theoretical properties tend to perform poorly when compared with recent high-performance algorithms [5]. (The experimental results in Section 6 confirm this observation.)

For MUS extraction, techniques such as clause set refinement allow dropping many clauses that are not included in an MUS. Similarly, model rotation often allows finding many clauses that must be in an MUS. Thus, in practice, the theoretical advantages of QuickXPlain are not observed. Moreover, in many settings, most of the clauses MUS algorithms end up analyzing are in the MUS. When the size of the MUS is close to the number of clauses that need to be analyzed, then QuickXPlain performs worse than approaches like the hybrid algorithm. Another drawback of an algorithm like QuickXPlain is that only a restricted version of model rotation can be integrated (see Section 4.3).

This section develops an algorithm for the MSMP problem that addresses all the drawbacks of the Bradley&Manna and QuickXPlain algorithms. The algorithm is shown to be correct, and the worst case number predicate tests is shown to be asymptotically equivalent to that of Bradley&Manna's and QuickXPlain algorithms. Afterwards, this section specializes the algorithm for the concrete case of MUS extraction, integrating techniques known to be essential for good performance [5].

5.1 A Progression Algorithm for the MSMP Problem

Algorithm 3 shows the organization of the new *progression*-based approach for the MSMP problem. The algorithm uses a geometric progression to define a subset of elements to drop from the set of elements on which the predicate is tested. If the predicate holds for the reduced set of elements, then the dropped elements are discarded, and the value of the progression is increased. Once the predicate does not hold, the algorithm invokes a binary search function (see Algorithm 4) to identify a *transition element* to include in the set of elements representing the minimal set. Similarly to the MUS case, a transition element is an element that, if dropped, the predicate does not hold. After identifying a transition element, the geometric progression is reset to 1, and the process

Algorithm 3: Progression-based computation of a minimal set

Input: Working set $\mathcal{W}$
Output: Minimal set $\mathcal{M}$

$(\mathcal{M}, i) \leftarrow (\emptyset, 0)$
while $\mathcal{W} \neq \emptyset$ **do**
  $\nu \leftarrow \min(2^i, |\mathcal{W}|)$
  if $p(\mathcal{M} \cup \mathcal{W} \setminus \mathcal{W}_{1..\nu})$ **then**
    $\mathcal{W} \leftarrow \mathcal{W} \setminus \mathcal{W}_{1..\nu}$
    $i \leftarrow i + 1$
  else
    $j \leftarrow$ `BinSearchTransElem`$(\mathcal{M}, \mathcal{W}, \nu)$
    $\mathcal{W} \leftarrow \mathcal{W} \setminus \mathcal{W}_{1..j}$
    $\mathcal{M} \leftarrow \mathcal{M} \cup \mathcal{W}_{j..j}$
    $i \leftarrow 0$
return $\mathcal{M}$

Algorithm 4: Binary search for a transition element

function `BinSearchTransElem`$(\mathcal{M}, \mathcal{W}, \nu)$
  Input: $\mathcal{M}$; $\mathcal{W}$; ν
  Output: Index of transition element r
  $(l, r) \leftarrow (0, \nu)$
  while $l < r - 1$ **do** // *Inv:* $p(\mathcal{M} \cup \mathcal{W} \setminus \mathcal{W}_{1..l}) \wedge \neg p(\mathcal{M} \cup \mathcal{W} \setminus \mathcal{W}_{1..r})$
    $m \leftarrow \lfloor \frac{l+r}{2} \rfloor$
    if $p(\mathcal{M} \cup \mathcal{W} \setminus \mathcal{W}_{1..m})$ **then**
      $l \leftarrow m$
    else
      $r \leftarrow m$
  return r
end

continues. As will be shown after specializing Algorithm 3, standard pruning techniques are easily integrated in the progression-based algorithm. Some of the insights of the new algorithm include the following. First, dropping more than one element is achieved by the geometric progression. Second, identification of the transition element is achieved using binary search. The progression is reset to 1, to simplify the complexity analysis; heuristics could be used to decide how to reset the progression after a transition element is identified.

5.2 Analysis of the Progression-Based Algorithm

This section analyzes Algorithm 3. The purpose is to show that the algorithm terminates, that it is correct (i.e. it computes a minimal set), and to prove the asymptotic complexity in terms of the number of times the predicate is

tested. (Observe that, as illustrated in the previous section, the predicate is tested through a SAT solver call.) Before that, we need to analyze Algorithm 4.

Lemma 1. *Algorithm 4 terminates.*

Proof. If $l \geq r-1$, then the while loop is not executed, and the algorithm terminates. If $l < r-1$, then at each iteration of the while loop, either l or r are updated — the update would either increase l, or decrease r. □

Analysis of the pseudo-code allows concluding that the invariant $p(\mathcal{M} \cup \mathcal{W} \setminus \mathcal{W}_{1..l}) \land \neg p(\mathcal{M} \cup \mathcal{W} \setminus \mathcal{W}_{1..r})$ holds while executing *BinSearchTransElem*.

Lemma 2. *The value r returned by Algorithm 4 is the smallest index in the range $1..\nu$ such that $p(\mathcal{M} \cup (\mathcal{W} \setminus \mathcal{W}_{1..r-1}))$ holds and $p(\mathcal{M} \cup (\mathcal{W} \setminus \mathcal{W}_{1..r}))$ does not hold.*

Proposition 1. *Algorithm 3 terminates.*

Proof. By precondition (see Section 2), the sets considered are finite, and so is $\mathcal{W}$. At each execution of the while loop, either the predicate p holds or is does not. If the predicate holds, the set $\mathcal{W}$ is reduced by $\nu \geq 1$ elements. If the predicate does not hold, then the call to *BinSearchTransElem* (line 8) returns a value $1 \leq j \leq \nu$. Thus, set $\mathcal{W}$ is reduced by $j \geq 1$ elements. Therefore, the size of set $\mathcal{W}$ is reduced in each iteration of the while loop and so the algorithm terminates. □

Correctness requires conditions on set $\mathcal{W}$, besides being finite. Concretely, we require $p(\mathcal{W})$ to hold.

Proposition 2. *Algorithm 3 is correct, i.e. if $p(\mathcal{W})$ holds, then $p(\mathcal{M})$ holds and $\mathcal{M}$ is a minimal such subset of $\mathcal{W}$.*

Proof. Let $\mathcal{W}_o$ denote the original set given to the algorithm. We will show the invariant that $\mathcal{M}$ is a minimal subset of $\mathcal{W}_o \setminus \mathcal{W}$ s.t. $p(\mathcal{M} \cup \mathcal{W})$ holds. This invariant is sufficient to show the functional correctness since the algorithm terminates iff $\mathcal{W} = \emptyset$ and thus $\mathcal{M}$ is a minimal subset of $\mathcal{W}_o$ s.t. $p(\mathcal{M})$. The invariant holds upon initialization as $\mathcal{M} = \emptyset$ and thus $p(\mathcal{M} \cup \mathcal{W})$ holds by precondition and $\mathcal{M}$ is minimal. If $p(\mathcal{M} \cup \mathcal{W} \setminus \mathcal{W}_{1..\nu})$, then $\mathcal{W}_{1..\nu}$ is removed from $\mathcal{W}$ and thus at the end of the iteration $p(\mathcal{M} \cup \mathcal{W})$ still holds and $\mathcal{M}$ remains minimal since it has not been updated and at the same time $\mathcal{W}$ was shrunk. In the **else** branch of the **if** statement, j is computed s.t. $p(\mathcal{M} \cup (\mathcal{W} \setminus \mathcal{W}_{1..j-1}))$ holds but $p(\mathcal{M} \cup (\mathcal{W} \setminus \mathcal{W}_{1..j}))$ does *not* hold (by Lemma 2). Thus, removing $\mathcal{W}_{1..j}$ from $\mathcal{W}$ and adding $W_{j..j}$ to $\mathcal{M}$ preserves $p(\mathcal{M} \cup \mathcal{W})$. Moreover, it preserves the minimality of $\mathcal{M}$ since if $W_{j..j}$ were not inserted into $\mathcal{M}$, $p(\mathcal{M} \cup \mathcal{W})$ would not hold after $\mathcal{W}_{1..j}$ is removed from $\mathcal{W}$; and any other element cannot be removed from $\mathcal{M}$ since $\mathcal{M}$ was minimal by induction hypothesis, $W_{j..j}$ was part of the original W, and p is monotone. □

The complexity of the algorithm is measured in terms of the number of predicate tests (which correspond to calls to an NP, in our case SAT, oracle).

Proposition 3. *Let the size of $\mathcal{W}$ be $m = |\mathcal{W}|$ and the size of the largest minimal subset $\mathcal{M}$ be $k = |\mathcal{M}|$. If $k = 0$, Algorithm 3 requires $\mathcal{O}(\log(1+m))$ tests of predicate p, and if $k > 0$, it requires $\mathcal{O}(k \log(1+\frac{m}{k}))$ tests of predicate p.*

Proof. If $k = 0$, then the algorithm executes a simple geometric progression, and this gives the result. If $k > 0$, let $\mathcal{M} = \{\tau_1, \tau_2, \ldots, \tau_k\}$. Since Algorithm 3 analyzes the elements of $\mathcal{W}$ in order, the elements of $\mathcal{M}$ will also be discovered in order, first τ_1, then τ_2, and so on. For $i = 2, \ldots, k$, let α_i denote the number of elements of $\mathcal{W}$ between τ_{i-1} and τ_i, plus 1 due to τ_i. For $i = 1$, α_1 denotes the number of elements of $\mathcal{W}$ not in $\mathcal{M}$ before τ_1, plus 1 due to τ_1. We consider there are no elements above τ_k, since this gives the worst case (i.e. more elements located to the left of τ_k). Next consider the workings of Algorithm 3. While the predicate holds, the algorithm considers increasingly large subsets of elements of $\mathcal{W}$, starting with size 1 and progressing by powers of 2. For each i, the number of tests until the predicate does not hold (because the removed set contains τ_i) is $\log(1+\alpha_i)$, and the number of elements that need to be considered after τ_i is at most α_i. (We should use $\lceil \log(1+\alpha_i) \rceil$, but this does not change the asymptotic result.) Binary search will then require $\mathcal{O}(\log(1+\alpha_i))$ predicate tests to locate τ_i. (Again, we should use $\mathcal{O}(\lceil \log(1+\alpha_i) \rceil)$, but this does not change the asymptotic result.) Thus, for each τ_i, the number of predicate tests is $\log(1+\alpha_i)$, $i = 1, \ldots, k$. (Observe that for $k = i$ we are counting predicate tests that have been discarded, but this is correct in terms of computing an upper bound.) Summing over all i, we get $\sum_{i=1}^{k} \log(1+\alpha_i)^2 \equiv 2 \log \Pi_{i=1}^{k}(1+\alpha_i)$. Taking into consideration that $\sum_{i=1}^{k} \alpha_i = m$, the worst case corresponds to having sets of equal size, i.e. $\alpha_i = \frac{m}{k}$. Thus the worst case number of predicate tests is $2\,k \log(1+\frac{m}{k}) = \mathcal{O}(k \log(1+\frac{m}{k}))$. □

The asymptotic number of predicate tests can be shown to correspond to those of QuickXplain [19] and the optimal algorithm of Bradley&Manna [8,9], for both of which the number of predicate tests is $\mathcal{O}(k + k \log(\frac{m}{k}))$. If m is much larger than k, then the asymptotic number of tests is $O(k \log(\frac{m}{k}))$. If m is of the order of k, then the asymptotic number of tests is $O(k)$. Moreover, the progression-based algorithm incurs smaller constants for the case when $m = k$, i.e. when the reference set is a minimal set. In this situation, it is a factor of two better than either QuickXplain [19] or the optimal algorithm in [8,9].

5.3 Specialization for MUSes

This section shows how to specialize the progression-based MSMP algorithm to the case of MUS extraction. Observe that Algorithm 3 could be used, but it is possible to improve its performance in practice. The objective is to illustrate how two important pruning techniques can be integrated, namely clause-set refinement [14,22] and model rotation [22,5]. Other optimizations are also highlighted. Algorithm 5 shows the progression-based MUS extraction algorithm, whereas Algorithm 6 shows the binary search step. Several techniques to improve performance can be considered. Clause set refinement is applied each time

Algorithm 5: Progression-based computation of an MUS

Input: Unsatisfiable set $\mathcal{U} \subseteq \mathcal{F}$, viewed as sequence $\mathcal{U} = \langle c_1, \ldots, c_{|\mathcal{U}|} \rangle$
Output: MUS $\mathcal{M}$

1 $(\mathcal{M}, i) \leftarrow (\emptyset, 0)$
2 **while** $\mathcal{U} \neq \emptyset$ **do**
3 $\quad \nu \leftarrow \min(2^i, |\mathcal{U}|)$
4 $\quad (\mathit{st}, \mu, \mathcal{C}) \leftarrow \texttt{SAT}(\mathcal{M} \cup \mathcal{U} \setminus \mathcal{U}_{1..\nu})$
5 $\quad$ **if not** st **then**
6 $\quad\quad \mathcal{U} \leftarrow \mathcal{U} \cap \mathcal{C}$ `// Refine` $\mathcal{U}$
7 $\quad\quad i \leftarrow i + 1$
8 $\quad$ **else**
9 $\quad\quad \mathcal{T} \leftarrow \texttt{FalsifiedClauses}(\mu, \mathcal{U}_{1..\nu}\})$
10 $\quad\quad (c_j, \mu, \mathcal{U}) \leftarrow \texttt{BinSearchTransCl}(\mathcal{M}, \mathcal{U} \setminus \mathcal{T}, \mathcal{T}, \mu)$
11 $\quad\quad \mathcal{M} \leftarrow \mathcal{M} \cup \{c_j\}$
12 $\quad\quad \mathcal{U} \leftarrow \texttt{ModelRotate}(\mathcal{M}, \mathcal{U}, \mu)$
13 $\quad\quad i \leftarrow 0$
14 **return** $\mathcal{M}$

the SAT solver call returns false. This is shown in line 6 in Algorithm 5 and lines 7 and 8 in Algorithm 6. Model rotation is applied each time a clause is added to the MUS set $\mathcal{M}$. This is shown in line 12 in Algorithm 5. Another improvement is to exploit each model computed by the SAT solver to reduce the number of target clauses. For each computed model, the algorithms just need to subsequently search the transition clause over the clauses falsified by the model. This is achieved with function *FalsifiedClauses*. Observe that in Algorithm 6 the indices need to be corrected when the sets of clauses are changed. The additional functions ensure the correct indices are computed.

6 Experimental Results

This section presents the results of an experimental evaluation of a number of MUS extraction algorithms, including the progression-based algorithm presented in Section 5 [3]. Recent experimental results [5,6] have established the so-called *hybrid* algorithm, with clause set refinement and model rotation, to be the most efficient MUS extraction algorithm for practically-relevant benchmark sets. This section compares an implementation of the hybrid algorithm, with implementations of the progression-based algorithm (cf. Section 5) and the QuickXPlain algorithm [19] with various optimizations, as well as a number of additional state-of-the-art MUS extractors. The algorithms were evaluated on the benchmark instances from the MUS track of SAT Competition 2011 [4]. The

[3] Given the results in this paper, we could have also considered PIs, MMs, MCSes, MESes, etc. but opted to focus on MUS extraction.
[4] `http://www.satcompetition.org/`.

Algorithm 6: Binary search for transition clause

```
   function BinSearchTransCl(M, U, T, μ)
       Input: M; U; T ⊆ U; μ
       Output: Index of transition clause r; μ_R; U
1      μ_R ← μ
2      (l, r) ← (0, |T|)
3      while l < r − 1 do
4          m ← ⌊(l+r)/2⌋
5          (st, μ, C) ← SAT(M ∪ U ∪ T \ T_{1..m})
6          if not st then
7              U ← U ∩ C                                   // Refine U
8              (l, r, T) ← DropNonCoreClauses(C, T, l, r)
9          else
10             (l, r, T) ← FalsifiedClauses(μ, T, l, r)
11             μ_R ← μ              // Save model for model rotation later
12     return (c_r, μ_R, T ∪ U)
   end
```

experiments were performed on an HPC cluster, where each node is dual quad-core Intel Xeon E5450 3 GHz with 32 GB of RAM. Each algorithm was run with a timeout of 3600 seconds and a memory limit of 4 GB per input instance.

Figure 1 presents a cactus plot comparing the performance of the following MUS extractors: (*i*) our implementation of the QuickXPlain algorithm (Algorithm 1, denoted `QuickXPlain` in the plot), additionally with the certificate refinement for MUSes (Algorithm 2, denoted `QuickXPlain+UC`), and also with specialized version of model rotation (denoted `QuickXPlain+UC+ROT`); (*ii*) the top three MUS extractors from SAT Competition 2011, namely `MoUsSaka` [20] and `Haifa-MUC` [29] with and without preprocessing; (*iii*) the implementation of the hybrid and the dichotomic algorithms in the state-of-the-art MUS extractor `MUSer2` [6], denoted as `HYB` and `DICH`, respectively; (*iv*) the implementation of the progression-based algorithm (Algorithm 3, denoted `PROG`), also in `MUSer2`.

A number of conclusions can be drawn from the plot in Figure 1. First we note that the progression-based algorithm outperforms the hybrid algorithm, which, to our knowledge, is the best performing MUS extraction algorithm to date [5,6]. This latter claim is additionally supported by the fact that the top three MUS extractors from the SAT Competition 2011 trail significantly behind both `HYB` and `PROG` in Figure 1. This result is significant, since it implies that the progression-based algorithm is the first algorithm that is both asymptotically optimal, and that also performs well in practice. The experimental results are also very clear about the significant performance difference between the new algorithm and QuickXplain, even with the addition of the certificate refinement and model rotation. As a side note, we point out that these optimizations, which have not been proposed in previous work, have a notable positive impact on the performance of QuickXplain.

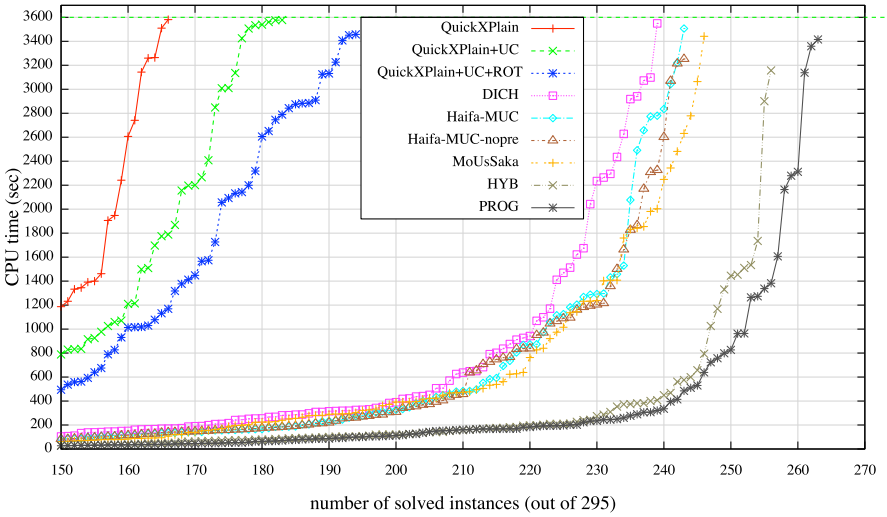

Fig. 1. Cactus plot: CPU runtimes of selected extractors on the benchmarks from the MUS track of SAT Competition 2011. Time limit 3600 sec, memory limit 4 GB.

The scatter plots in Figure 2 provide additional insights into performance profile of the progression-based algorithm. Comparing the algorithm with QUICKXPLAIN (left plot), we conclude that the new algorithm is a clear win, even when QUICKXPLAIN is augmented with the proposed optimizations. This suggests that the new algorithm captures the optimal behavior of the recursive partitioning scheme employed by QUICKXPLAIN, while at the same time avoiding the unfavorable for QUICKXPLAIN cases of instances with few non-MUS clauses. The comparison with the hybrid algorithm (right plot) confirms the overall positive trend towards the progression-based algorithm, but also shows that the performance of the two algorithms might be complementary, suggesting a possible integration of the algorithms into a portfolio.

7 Conclusions & Research Directions

This paper shows that several computational problems on Boolean formulas can be formulated as computing a minimal set subject to a monotone predicate, i.e. the MSMP problem. Examples include prime implicates, minimal models, minimal unsatisfiable subsets, minimal correction subsets, among others. This result allows using the same algorithms to solve all of these problems. In addition, the paper summarizes how standard pruning techniques can be adapted to each concrete computational problem. The insights provided by this result motivate the development of a new optimal algorithm for the MSMP problem, which is asymptotically as efficient as the asymptotically best algorithms. More importantly, the new algorithm has smaller constants than other algorithms,

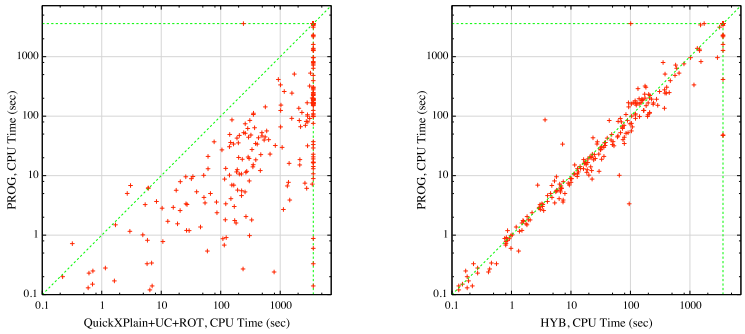

Fig. 2. CPU runtime of MUS extraction. Left: progression-based vs QUICKXPLAIN with optimizations. Right: progression-based algorithm vs hybrid.

and is amenable to the integration of well-known pruning techniques. The paper also shows how the new algorithm for the MSMP problem can be specialized for the case of MUS extraction. Experimental results, obtained on representative MUS problem instances, demonstrate that the new progression-based algorithm outperforms another optimal algorithm, namely an optimized version of QUICKXPLAIN [19], that includes clause set refinement and model rotation. The experimental results also demonstrate that the new algorithm outperforms the current state of the art hybrid algorithm [5,6].

The results in this paper open several research directions. For example, how does the new algorithm for the MSMP problem perform on other problems, e.g. prime implicates, minimal models, minimal correction subsets among others? How to apply the results in this paper to the case of group MUS, etc.?

References

1. Z. S. Andraus, M. H. Liffiton, and K. A. Sakallah. Reveal: A formal verification tool for verilog designs. In *LPAR*, pages 343–352, 2008.
2. J. Bailey and P. J. Stuckey. Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In *PADL*, pages 174–186, 2005.
3. R. R. Bakker, F. Dikker, F. Tempelman, and P. M. Wognum. Diagnosing and solving over-determined constraint satisfaction problems. In *IJCAI*, pages 276–281, 1993.
4. A. Belov, M. Janota, I. Lynce, and J. Marques-Silva. On computing minimal equivalent subformulas. In *CP*, pages 158–174, 2012.
5. A. Belov, I. Lynce, and J. Marques-Silva. Towards efficient MUS extraction. *AI Commun.*, 25(2):97–116, 2012.
6. A. Belov and J. Marques-Silva. MUSer2: An efficient MUS extractor, system description. *JSAT*, 8:123–128, 2012.
7. A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*. IOS Press, 2009.
8. A. R. Bradley and Z. Manna. Checking safety by inductive generalization of counterexamples to induction. In *FMCAD*, pages 173–180, 2007.

9. A. R. Bradley and Z. Manna. Property-directed incremental invariant generation. *Formal Asp. Comput.*, 20(4-5):379–405, 2008.
10. M. Cadoli and F. M. Donini. A survey on knowledge compilation. *AI Commun.*, 10(3-4):137–150, 1997.
11. J. W. Chinneck and E. W. Dravnieks. Locating minimal infeasible constraint sets in linear programs. *INFORMS Journal on Computing*, 3(2):157–168, 1991.
12. A. Darwiche and P. Marquis. A knowledge compilation map. *J. Artif. Intell. Res. (JAIR)*, 17:229–264, 2002.
13. J. L. de Siqueira N. and J.-F. Puget. Explanation-based generalisation of failures. In *ECAI*, pages 339–344, 1988.
14. N. Dershowitz, Z. Hanna, and A. Nadel. A scalable algorithm for minimal unsatisfiable core extraction. In *SAT*, pages 36–41, 2006.
15. A. Felfernig, M. Schubert, and C. Zehentner. An efficient diagnosis algorithm for inconsistent constraint sets. *AI EDAM*, 26(1):53–62, 2012.
16. C. P. Gomes, H. Kautz, A. Sabharwal, and B. Selman. Satisfiability solvers. In V. L. Frank van Harmelen and B. Porter, editors, *Handbook of Knowledge Representation*, volume 3, pages 89–134. 2008.
17. É. Grégoire, B. Mazure, and C. Piette. On approaches to explaining infeasibility of sets of Boolean clauses. In *ICTAI*, pages 74–83, November 2008.
18. F. Hemery, C. Lecoutre, L. Sais, and F. Boussemart. Extracting MUCs from constraint networks. In *ECAI*, pages 113–117, 2006.
19. U. Junker. QUICKXPLAIN: Preferred explanations and relaxations for over-constrained problems. In *AAAI*, pages 167–172, 2004.
20. S. Kottler. Description of the SApperloT, SArTagnan and MoUsSaka solvers for the SAT-Competition 2011. http://www.satcompetition.org/2011/, 2011.
21. M. H. Liffiton and K. A. Sakallah. Algorithms for computing minimal unsatisfiable subsets of constraints. *J. Autom. Reasoning*, 40(1):1–33, 2008.
22. J. Marques-Silva and I. Lynce. On improving MUS extraction algorithms. In *SAT*, pages 159–173, 2011.
23. J. Marques-Silva, I. Lynce, and S. Malik. Conflict-driven clause learning SAT solvers. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, pages 131–153. IOS Press, 2009.
24. P. Marquis. Consequence finding algorithms. In *Algorithms for Defeasible and Uncertain Reasoning*. Kluwer Academic Publishers, 2000.
25. J. McCarthy. Circumscription - a form of non-monotonic reasoning. *Artif. Intell.*, 13(1-2):27–39, 1980.
26. A. Nadel. Boosting minimal unsatisfiable core extraction. In *FMCAD*, pages 121–128, October 2010.
27. A. Nöhrer, A. Biere, and A. Egyed. Managing SAT inconsistencies with HUMUS. In *VaMoS*, pages 83–91, 2012.
28. R. Reiter. A theory of diagnosis from first principles. *Artif. Intell.*, 32(1):57–95, 1987.
29. V. Ryvchin and O. Strichman. Faster extraction of high-level minimal unsatisfiable cores. In *SAT*, pages 174–187, 2011.
30. S. Schlobach, Z. Huang, R. Cornet, and F. van Harmelen. Debugging incoherent terminologies. *J. Autom. Reasoning*, 39(3):317–349, 2007.
31. T. Soh and K. Inoue. Identifying necessary reactions in metabolic pathways by minimal model generation. In *ECAI*, pages 277–282, 2010.
32. H. van Maaren and S. Wieringa. Finding guaranteed MUSes fast. In *SAT*, pages 291–304, 2008.

33. S. Wieringa. Understanding, improving and parallelizing MUS finding using model rotation. In *CP*, pages 672–687, 2012.