Counting Minimal Unsatisfiable Subsets

Jaroslav Bendík[1,2(✉)] and Kuldeep S. Meel[2]

[1] Faculty of Informatics, Masaryk University, Brno, Czech Republic
xbendik@fi.muni.cz
[2] National University of Singapore, Singapore, Singapore

Abstract. Given an unsatisfiable Boolean formula F in CNF, an unsatisfiable subset of clauses U of F is called Minimal Unsatisfiable Subset (MUS) if every proper subset of U is satisfiable. Since MUSes serve as explanations for the unsatisfiability of F, MUSes find applications in a wide variety of domains. The availability of efficient SAT solvers has aided the development of scalable techniques for finding and enumerating MUSes in the past two decades. Building on the recent developments in the design of scalable model counting techniques for SAT, Bendík and Meel initiated the study of MUS counting techniques. They succeeded in designing the first approximate MUS counter, AMUSIC, that does not rely on exhaustive MUS enumeration. AMUSIC, however, suffers from two shortcomings: the lack of exact estimates and limited scalability due to its reliance on 3-QBF solvers.
In this work, we address the two shortcomings of AMUSIC by designing the first exact MUS counter, CountMUST, that does not rely on exhaustive enumeration. CountMUST circumvents the need for 3-QBF solvers by reducing the problem of MUS counting to projected model counting. While projected model counting is #NP-hard, the past few years have witnessed the development of scalable projected model counters. An extensive empirical evaluation demonstrates that CountMUST successfully returns MUS count for 1500 instances while AMUSIC and enumeration-based techniques could only handle up to 833 instances.

1 Introduction

Boolean formulas serve as a primary representation language to model the behaviour of systems and properties. Given an unsatisfiable Boolean formula F in Conjunctive Normal Form (CNF), i.e. a set of clauses $F = \{f_1, f_2, \ldots, f_n\}$, a subset $U \subseteq F$ is called Minimal Unsatisfiable Subset (MUS) of F iff U is unsatisfiable and for every $f \in U$, $U \setminus \{f\}$ is satisfiable.

MUSes serve as *explanations* or *reasons* for unsatisfiability of F, and have, consequently, found applications in a wide variety of domains such as diagnosis [56,24], constrained sampling and counting [28], equivalence checking [20], and the like [1,64,47,2,30,25]. While the early applications relied on identifying a single [53,51,6,7,3] or enumerating multiple [39,52,4,12,41,10] MUSes, the rapid adoption of MUSes lead researchers to investigate problem formulations and their corresponding applications that do not rely on explicit MUS identification.

These include, e.g., computing the union of all MUSes [45], deciding whether a given clause belongs to an MUS [31], or counting the number of MUSes. Especially, the counting of MUSes found many applications in the domain of diagnosis where the MUS count can be used to compute various inconsistency metrics [25,48,65,49,50,29] for general propositional knowledge bases.

A straightforward, and for many years the only available, approach for counting MUSes is to simply enumerate them. However, there can be up to exponentially many MUSes w.r.t. $|F|$ and hence the complete enumeration is often practically intractable [39,69,9,10]. Inspired by the development of model counting techniques in the context of SAT, which in its nascent stages also depended on complete model enumeration while contemporary techniques often need to explicitly identify just a fraction of models, Bendík and Meel [13] recently initiated an investigation of counting MUSes without their explicit enumeration. In this context, they succeeded by developing a hashing-based approximate counter, AMUSIC [13], that provides the so-called PAC guarantees, also known as (ε, δ)-guarantees, wherein the computed answer is within the $(1 + \varepsilon)$-factor of the exact count with confidence at least $1 - \delta$. AMUSIC reduces the problem of MUS counting to logarithmically many calls to a Σ_3^P oracle (3-QBF solver, in practice) wherein every Σ_3^P query is constructed over a CNF formula conjuncted with XORs.

While AMUSIC achieved its stated goal of avoiding explicit enumeration, its scalability is significantly hampered by its reliance on a 3-QBF solver that can efficiently handle formulas conjuncted with XOR constraints. It is worth highlighting that the scalability of model counting techniques [17,60] in the context of SAT crucially relies on the availability of CryptoMiniSAT [61], a SAT solver with native support for CNF-XOR constraints. Despite significant advances in QBF solving over the years, the scalability remains a formidable challenge for 3-QBF solvers, and even more when XOR constraints are involved. As such, AMUSIC could scale to formulas involving few hundreds of variables and clauses.

In this work, we focus on addressing the scalability of MUS counting techniques. We begin our investigation by focusing on the observation of Bendík and Meel that their technique relied on a Σ_3^P oracle even though the problem of finding an MUS is in FP^{NP} [19,44]. Therefore, a natural direction is to investigate the design of an algorithmic framework that can circumvent reliance on oracles with high complexity. In this context, we rely on the observation of Durand, Hermann, and Koliatis [21] that the complexity of counting problems whose search problems have FP^{NP} complexity tend to be #NP (which contains #P class). Such an observation is timely given the recent surge of interest in designing efficient techniques for projected model counting, which is #NP-hard. Therefore, one wonders: *whether it is possible to design a MUS counting technique that can take advantage of projected model counters?*

The primary contribution of this paper is an affirmative answer to the above question. We design a new algorithmic framework, CountMUST, that reduces the problem of MUS counting to two projected model counting queries. In particular, CountMUST constructs a wrapper $\mathcal{W}$ and its remainder $\mathcal{R}$ such that the

number of MUSes of F is $|\mathcal{W}| - |\mathcal{R}|$, i.e., the wrapper $\mathcal{W}$ over-approximates the set of MUSes while the remainder contains the spurious, non-MUS, subsets of F that emerge due to the over-approximation. We encode the wrapper $\mathcal{W}$ and the remainder $\mathcal{R}$ with Boolean formulas $\mathbb{W}$ and $\mathbb{R}$ such that the projected model counts for $\mathbb{W}$ and $\mathbb{R}$ (for a suitable projection set) equal to $|\mathcal{W}|$ and $|\mathcal{R}|$, respectively. An interesting (and perhaps surprising) aspect of our CountMUST is that we do not enumerate a single MUS in our process, which is in stark contrast to the design of AMUSIC that relies on the enumeration of a *small* number of MUSes.

We discuss several strategies to construct wrappers (and their corresponding remainders) that are efficient to compute and are tight over-approximations of the set of MUSes. We conduct a detailed empirical analysis over 2553 instances and observe that CountMUST successfully returns MUS count for 1500 instances while AMUSIC and enumeration-based techniques could only handle up to 833 instances. We observe interesting complementary nature of the exact and approximate MUS counting approaches: the scalability of AMUSIC is often impacted by the number of clauses and appears to be less impacted by the number of MUSes while, on the other hand, the scalability of CountMUST is less impacted by the number of clauses and appears to depend on the number of MUSes.

Finally, our empirical analysis showcases that our wrappers $\mathcal{W}$ approximate the set of MUSes very tightly. Motivated by the tightness of our wrappers, we discuss several interesting applications of our framework: approximate MUS counting [13], MUS enumeration [5,40], MUS Sampling, estimation of minimum and maximum MUS cardinality [38,27], and MUS membership testing [31].

The rest of the paper is organized as follows. We introduce preliminaries in Section 2 and discuss related work in Section 3. We then present the primary technical contribution of our work in Section 4. We present the empirical evaluation in Section 5 and then discuss the implications of the tightness of our wrappers in Section 6. We finally conclude in Section 7.

2 Preliminaries and Problem Definition

A Boolean formula F is built over Boolean values $\{1, 0\}$ and over a set $Vars(F)$ of Boolean variables connected via standard logical operators: $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, $\neg$. A literal is either a variable $x \in Vars(F)$ or its negation $\neg x$; $Lits(F)$ denotes the set of all literals used in F. Given a set A of variables, a valuation $\pi : A \rightarrow \{1, 0\}$ assigns to each variable its Boolean value. $F[\pi]$ denotes the formula that emerges from F by substituting every variable x of F that is in the domain of π by $\pi(x)$; furthermore, trivial simplifications, e.g., $G \vee 0 = G$, $G \wedge 0 = 0$, $\neg 1 = 0$, $\neg 0 = 1$, are applied. Note that if $A \supseteq Vars(F)$, then $F[\pi]$ is simplified either to 1 or to 0. In the case when $A \supseteq Vars(F)$ and $F[\pi] = 1$, we call π a *model* of F and write $\pi \models F$; otherwise, when $F[\pi] = 0$, we write $\pi \not\models F$. A formula F is *satisfiable* if it has a model; otherwise, F is *unsatisfiable*. We write M_F to denote the set of all models of F. Moreover, given a set $A \subseteq Vars(F)$ of variables, we write $M_{F\downarrow A}$ to denote the projection of M_F on A, and for every $\pi \in M_F$, we write $\pi_{\downarrow A}$ to denote

the projection of π on A. Finally, given two variable sets, $A = \{a_1, \ldots, a_k\}$ and $B = \{b_1, \ldots, b_k\}$, such that $A \subseteq \mathit{Vars}(F)$, we write $F_{[A/B]}$ to denote the formula that originates from F by substituting each variable $a_i \in A$ by $b_i \in B$.

A formula in conjunctive normal form, shortly a *CNF formula*, is a conjunction of *clauses* where a clause is a disjunction of literals. When suitable, a CNF formula can also be viewed as a multiset of clauses where a clause is a set of literals; we use the two representations interchangeably based on the context. Throughout the whole text, let us by $F = \{f_1, \ldots, f_n\}$ denote the input CNF formula of interest. Furthermore, capital letters, e.g., S, K, N, or blackboard bold letters, e.g., $\mathbb{W}$, $\mathbb{R}$, are used to denote other formulas, small letters, e.g., f, f_1, f_i, are used to denote clauses, and small letters, e.g., x, x', y, are used to denote variables. Finally, given a set X, $\mathcal{P}(X)$ denotes the power-set of X, and $|X|$ denotes the cardinality of X.

Definition 1 (MUS). *A subset N of F is a* minimal unsatisfiable subset *(MUS) of F iff N is unsatisfiable and for every $f \in N$ it holds that $N \setminus \{f\}$ is satisfiable.*

Definition 2 (MSS). *A subset N of F is a* maximal satisfiable subset *(MSS) of F iff N is satisfiable and for every $f \in F \setminus N$ it holds that $N \cup \{f\}$ is unsatisfiable.*

Definition 3 (MCS). *A subset N of F is a* minimal correction subset *(MCS) of F iff $F \setminus N$ is satisfiable and for every $f \in N$ it holds that $F \setminus (N \setminus \{f\})$ is unsatisfiable. Equivalently, N is an MCS iff $F \setminus N$ is an MSS.*

Note that the Boolean satisfiability is monotone w.r.t. the (clause) subset inclusion, i.e., all subsets of a satisfiable set of clauses are satisfiable. Consequently, all proper subsets of an MUS are in fact satisfiable, and, dually, all proper supersets of an MSS are unsatisfiable. Also, note that the minimality/maximality concept used here is a *set minimality/maximality* and not a *minimum/maximum cardinality*. Consequently, there can be up to $\binom{|F|}{|F|/2}$ MUSes/MCSes/MSSes of F (intuitively, this is the number of pair-wise incomparable subsets of F; see the Sperner's theorem [62]). We write *maximum* and *minimum* MUS to denote an MUS with the maximum and the minimum cardinality, respectively. Note that there can also be exponentially many maximum and minimum MUSes. We write $\texttt{MUS}_F$ to denote the set of all MUSes of F, and $\texttt{SS}_F$ to denote the set of all satisfiable subsets of F.

Example 1. Let us demonstrate the concepts of MUSes, MSSes and MCses on an example. Assume that $F = \{f_1 = \{x_1\}, f_2 = \{\neg x_1\}, f_3 = \{x_2\}, f_4 = \{\neg x_1, \neg x_2\}\}$. There are 2 MUSes: $\texttt{MUS}_F = \{\{f_1, f_3, f_4\}, \{f_1, f_2\}\}$, 3 MSSes: $\{\{f_2, f_3, f_4\}, \{f_1, f_4\}, \{f_1, f_3\}\}$, and thus also 3 MCSes: $\{\{f_1\}, \{f_2, f_3\}, \{f_2, f_4\}\}$. For illustration, see Fig. 1.

In this paper, we are concerned with the following two problems.

Name: `#MUS`
Input: A CNF formula F.
Output: The number $|\texttt{MUS}_F|$ of MUSes of F.

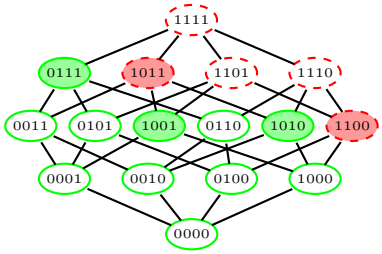

Fig. 1. Illustration of $\mathcal{P}(F)$ from the Example 1. Individual subsets are represented as bit-vectors, e.g., $\{f_1, f_2\}$ is written as 1100. The subsets with a dashed border are the unsatisfiable subsets, and the others are satisfiable subsets. MUSes and MSSes are filled with a background colour.

Name: `proj-#SAT`
Input: A formula G and a set of variables $S \subseteq \mathit{Vars}(G)$.
Output: The number $|M_{G\downarrow S}|$ of models of G projected on S.

Our goal is to solve the `#MUS` problem, and to do that, we propose a *strong subtractive reduction* to the `proj-#SAT` problem.

Definition 4 (Strong Subtractive Reductions). *[21] Let Σ be an alphabet and let Q_1 and Q_2 be two binary relations over Σ. Let $\#\cdot Q_1$ and $\#\cdot Q_2$ represent the corresponding counting problems. Then, $\#\cdot Q_1$ reduces to $\#\cdot Q_2$ via a strong subtractive reduction, if there exist polynomial-time computable functions f and g such that for every string $z \in \Sigma^*$:*

1. $Q_2(f(z)) \subseteq Q_2(g(z))$
2. $|Q_1(z)| = |Q_2(g(z))| - |Q_2(f(z))|$.

3 Related Work

MUS Counting A straight-forward approach to count the MUSes is to simply enumerate them via an MUS enumeration algorithm, e.g. [5,39,52,41,4,8,12,10]. However, since there can be up to exponentially many MUSes w.r.t. $|F|$, the complete enumeration is often practically intractable. An alternative approach to identify the MUS count is based on a so-called *minimal hitting set duality* between MUSes and MCSes that states that every MUS is a *minimal hitting set* of the set of all MCSes [56,32]. Consequently, one can determine the MUS count by first identifying all MCSes and then counting their minimal hitting sets [40]. However, there can be in general up to exponentially many MCSes, which makes this approach also often practically intractable [52,11].

The study of MUS counting without relying on exhaustive enumeration was initiated just recently by Bendík and Meel [13], who proposed an (ε, δ)-approximation scheme called AMUSIC. AMUSIC extends a prior hashing-based model counting framework [63,15,18] to MUS counting. Briefly, AMUSIC divides

the power-set $\mathcal{P}(F)$ into *nCells* small *cells*, then pick one of the cells and count the number *inCell* of MUSes in the cell, and estimate the overall MUS count as $nCells \times inCell$. The approach requires to perform logarithmically many calls to a Σ_3^P oracle (3-QBF solver) wherein each query consists of a CNF formula conjuncted with XOR constraints. The lack of solvers with native support for such constraints presents the major hindrance to the scalability of AMUSIC.

It is worth remarking on a recent work by Bendík and Meel [14] that focuses on exact counting of maximal satisfiable subsets (MSSes). While MUSes and MSSes are closely related concepts, to the best of our knowledge, there does not exist any efficient reduction from MUS counting to MSS counting, or vice versa. Note that the best known upper-bound on the problem of finding an MUS is $\mathrm{FP}^{\mathrm{NP}}$ [19], whereas for findind an MSS a tighter upper-bound $\mathrm{FP}^{\mathrm{NP}}[wit, log]$ is known [44], which suggests that counting MUSes is practically harder than counting MSSes. It would be an interesting question for future work if the counter developed in this work can be employed to perform MSS counting.

Model Counting The complexity-theoretic study of model counting was initiated by Valiant [67] who showed that `proj-#SAT` is #P-complete when $S = Vars(G)$. Subsequently, Durand, Hermann, and Koliatis [21] showed that the general problem of `proj-#SAT` is #NP-hard. A significant conceptual contribution of Durand et al. was to show the importance of subtractive reductions for problems in #NP; this idea has been applied for reductions to projecting counting [14].

Our work relies on the recent progress in the development of efficient projected model counters; in particular, we employ GANAK [59], a state-of-the-art *search-based* exact model counter; the entry based on GANAK won the projected model counting track in 2020 Model Counting Competition [23]. Search-based model counters build on three core ideas: (1) for a formula G and $x \in S$, we have $|M_{G\downarrow S}| = |M_{G(x\mapsto 0)\downarrow S}| + |M_{G(x\mapsto 1)\downarrow S}|$, (2) if G can be partitioned into subset of clauses $\{C_1, C_2, \ldots C_k\}$ such that $\forall i, j.\ Vars(C_i) \cap Vars(C_j) = \emptyset$, then we have $|M_{G\downarrow S}| = \prod_{i=1}^{k} |M_{C_i\downarrow S}|$, and (3) finally, component caching is employed to cache the components. Consequently, the model count can be often determined by explicitly identifying just a fraction of all models. GANAK is built on top of earlier search-based model counters, sharpSAT [66] and Cachet [58,57].

4 MUS Counting via a Projected Model Counter

We now gradually introduce several subtractive reductions of the MUS counting problem to the projected model counting, starting with the base idea in Section 4.1, and following with the particular reductions in Sections 4.2-4.11.

4.1 Basic MUS Counting Idea

Definition 5 (wrapper and remainder). *A set $\mathcal{W}$ of subsets of F is a* wrapper *iff $\mathtt{MUS}_F \subseteq \mathcal{W} \subseteq \mathtt{MUS}_F \cup \mathtt{SS}_F$. Furthermore, the* remainder *of $\mathcal{W}$ is the set $\mathcal{R} = \mathcal{W} \cap \mathtt{SS}_F$.*

Proposition 1. *Let $\mathcal{W}$ be a wrapper and $\mathcal{R}$ its corresponding remainder. Then* $|\mathtt{MUS}_F| = |\mathcal{W}| - |\mathcal{R}|$.

Proof. Since $\mathcal{R} = \mathcal{W} \cap \mathtt{SS}_F$, then $\mathtt{MUS}_F \cap \mathcal{R} = \emptyset$, and hence $|\mathcal{W}| = |\mathtt{MUS}_F| + |\mathcal{R}|$.

Our approach to determine the MUS count $|\mathtt{MUS}_F|$ consists of the following steps. First, we define a wrapper $\mathcal{W}$ and its corresponding remainder $\mathcal{R}$. Subsequently, we encode the wrapper $\mathcal{W}$ with a Boolean formula $\mathbb{W}$ such that each projected model of $\mathbb{W}$ (for a suitable projection set) corresponds to an element of $\mathcal{W}$. Similarly, we construct a Boolean formula $\mathbb{R}$ such that each projected model of $\mathbb{R}$ corresponds to an element of the remainder $\mathcal{R}$. Finally, we employ a projected model counter to determine the projected model counts of $\mathbb{W}$ and $\mathbb{R}$, i.e., $|\mathcal{W}|$ and $|\mathcal{R}|$, and hence we obtain the MUS count $|\mathtt{MUS}_F| = |\mathcal{W}| - |\mathcal{R}|$.

In the following, we first describe in Section 4.2 how to build a simple wrapper $\mathcal{W}_1$ and its remainder $\mathcal{R}_1$ and how to encode them via Boolean formulas $\mathbb{W}_1$ and $\mathbb{R}_1$, respectively. Subsequently, in Sections 4.3-4.11, we propose several additional wrappers (and their remainders) that improve upon the base wrapper $\mathcal{W}_1$ by exploiting various observations about MUSes. Finally, in Section 4.12, we show how to combine the individual wrappers.

4.2 $\mathcal{W}_1$ - The Base Wrapper and Its Reminder

Our base wrapper, $\mathcal{W}_1$, is simply the set of all satisfiable subsets and all MUSes of F, i.e., $\mathcal{W}_1 = \mathtt{SS}_F \cup \mathtt{MUS}_F$. The corresponding remainder $\mathcal{R}_1$ is thus the set $\mathtt{SS}_F$ of all satisfiable subsets of F. In the following, we describe how to encode the wrapper $\mathcal{W}_1$ and the remainder $\mathcal{R}_1$ via Boolean formulas $\mathbb{W}_1$ and $\mathbb{R}_1$ whose projected models correspond to elements of $\mathcal{W}_1$ and $\mathcal{R}_1$, respectively.

Let us start with encoding the remainder $\mathcal{R}_1 = \mathtt{SS}_F$. Given the unsatisfiable formula $F = \{f_1, \ldots, f_n\}$, we introduce a set $\mathcal{A} = \{a_1, \ldots, a_n\}$ of *activation variables*. Note that every valuation π of $\mathcal{A}$ one-to-one maps to an *activated* subset $\pi_{\mathcal{A},F}$ of F defined as $\pi_{\mathcal{A},F} = \{f_i \in F \,|\, \pi(a_i) = 1\}$. Using the activation variables, we build the formula $\mathbb{R}_1$ as follows:

$$\mathbb{R}_1 = \bigwedge_{f_i \in F} a_i \rightarrow f_i \tag{1}$$

Intuitively, if we set a_i to 0 then the formula $a_i \rightarrow f_i$ is trivially satisfied, and if we set a_i to 1 then f_i has to be satisfied to satisfy $a_i \rightarrow f_i$. Hence, the models of $\mathbb{R}_1$ projected on $\mathcal{A}$ map to satisfiable subsets of F; formally:

Proposition 2. *For every valuation π of $\mathcal{A}$, $\pi \in M_{\mathbb{R}_1 \downarrow \mathcal{A}}$ iff $\pi_{\mathcal{A},F} \in \mathcal{R}_1 = \mathtt{SS}_F$. Consequently, $|M_{\mathbb{R}_1 \downarrow \mathcal{A}}| = |\mathcal{R}_1|$.*

Let us note that the concept of activation variables (or alternatively *relaxation variables*) and the idea behind the formula $\mathbb{R}_1$ is not novel and it appeared also in several MUS/MSS/MCS related studies such as [31,42,14]. However, we are the first who apply it in the context of MUS counting.

To build a formula $\mathbb{W}_1$ that represents the wrapper $\mathcal{W}_1 = \mathtt{SS}_F \cup \mathtt{MUS}_F$, we will proceed similarly, i.e., we build $\mathbb{W}_1$ using the activation variables $\mathcal{A}$ in such a way that a valuation π of $\mathcal{A}$ is a projected model of $\mathbb{W}_1$ iff $\pi_{\mathcal{A},F} \in \mathcal{W}_1$. A straightforward approach to encode $\mathcal{W}_1$ is to directly express that we are interested either in satisfiable subsets or MUSes of F. Such an encoding might look as $\mathbb{R}_1(\mathcal{A}) \vee \mathsf{isMUS}(\mathcal{A})$ where $\mathbb{R}_1(\mathcal{A})$ is the formula from Eq. 1 encoding that $\pi_{\mathcal{A},F}$ is satisfiable and $\mathsf{isMUS}(\mathcal{A})$ is a formula encoding that $\pi_{\mathcal{A},F}$ is an MUS. However, encoding that a set S is an MUS is quite expensive since one has to express that all subsets of S are satisfiable and that S is unsatisfiable (Definition 1). Especially, encoding that a set S is unsatisfiable requires to assume all the exponentially many valuations of $\mathit{Vars}(S)$. Several MUS related studies used various QBF encodings for the property of being an MUS, e.g., [31,13]. In particular, to express that a set S is an MUS, one can use the following, intuitively described, $\forall\exists$-QBF encoding: "**for every** valuation τ of $\mathit{Vars}(S)$ the valuation τ models $\neg S$ (i.e., S is unsatisfiable) **and for every** subset S' of S there **exists** a valuation τ' of $\mathit{Vars}(S')$ that satisfies S'". One could convert the $\forall\exists$-QBF encoding into a plain Boolean formula by explicitly enumerating all the possible valuations of $\mathit{Vars}(S)$ and all the subsets of S, however, this yields an exponentially large, and thus intractable, formula. Hence, instead of directly expressing that every element of the wrapper $\mathcal{W}_1$ is either a satisfiable subset or an MUS of F, we propose another approach based on a novel concept of an *evidence*.

Definition 6 (evidence). *Let A be a subset of $F = \{f_1, \ldots, f_n\}$. An* evidence *for A is a tuple $(\rho_1, \ldots, \rho_n)$ such that for every $1 \leq i \leq n$ it holds that:*

1. $\rho_i : \mathit{Vars}(F) \to \{1, 0\}$ *is truth assignment, and*
2. $\rho_i \models A \setminus \{f_i\}$.

Crucially, we observe the following:

Proposition 3. *For every subset A of F it holds that $A \in \mathtt{SS}_F \cup \mathtt{MUS}_F = \mathcal{W}_1$ iff there exists an evidence for A.*

Our formula $\mathbb{W}_1$ (Eq. 2) that encodes the wrapper $\mathcal{W}_1$ captures every set $A \subseteq F$ for which there exists an evidence $(\rho_1, \ldots, \rho_n)$. To represent the set A, we use the activation variables $\mathcal{A} = \{a_1, \ldots, a_n\}$. To represent the truth assignments $\rho_1, \ldots, \rho_n$, we introduce variable sets $\mathcal{I}_1, \ldots, \mathcal{I}_n$ where $\mathcal{I}_i$ is a fresh copy of $\mathit{Vars}(F)$ for every $i \in \{1, \ldots, n\}$.

$$\mathbb{W}_1 = \bigwedge_{a_i \in A} a_i \to \Big(\bigwedge_{j \in \{1,\ldots,n\} \setminus \{i\}} (a_j \to f_{j[\mathit{Vars}(F)/\mathcal{I}_i]}) \Big) \tag{2}$$

Intuitively, let π' be a valuation of $\mathit{Vars}(\mathbb{W}_1)$ and $\pi'_{\mathcal{A},F} = \{f_i \in F \mid \pi'(a_i) = 1\}$ the subset of F activated by $\mathcal{A}$. For every activated clause $f_i \in \pi'_{\mathcal{A},F}$, the formula expresses that $\pi'_{\downarrow \mathcal{I}_i}$ is a model of $\pi'_{\mathcal{A},F} \setminus \{f_i\}$ where the variable set $\mathit{Vars}(F)$ is substituted by $\mathcal{I}_i$.

Proposition 4. *For every valuation π of $\mathcal{A}$, $\pi \in M_{\mathbb{W}_1 \downarrow \mathcal{A}}$ iff $\pi_{\mathcal{A},F} \in \mathcal{W}_1 = \mathtt{SS}_F \cup \mathtt{MUS}_F$. Consequently, $|M_{\mathbb{W}_1 \downarrow \mathcal{A}}| = |\mathcal{W}_1|$.*

Based on Propositions 2 and 4, we can now employ a projected model counter to obtain the model counts $|M_{\mathbb{W}_1\downarrow\mathcal{A}}|$ and $|M_{\mathbb{R}_1\downarrow\mathcal{A}}|$, which yields $|\mathcal{W}_1|$ and $|\mathcal{R}_1|$, and hence also $|\texttt{MUS}_F|$ (Proposition 1). However, the concern here is the tractability of obtaining the model counts. There are mainly two criteria that affect the practical tractability of projected model counting. One criterion is the number of projected models, i.e. the cardinality of the wrapper (and the remainder), and the other criterion is the cardinality of the projection set, i.e., $|\mathcal{A}|$. The wrapper $\mathcal{W}_1$ is not very efficient w.r.t. these two criteria. Especially, $\mathcal{W}_1$ contains all satisfiable subsets of F, and there are often exponentially many satisfiable subsets of F w.r.t. $|F|$. Therefore, in the following, we will present nine additional wrappers, $\mathcal{W}_2, \ldots, \mathcal{W}_{10}$, and their corresponding remainders. Each of the wrappers captures a property of MUSes that allows us to provide a better description of MUSes, and hence reduce the cardinality of the wrapper and/or the cardinality of the projection set. Similarly as in the case of $\mathcal{W}_1$, we will use the activation variables $\mathcal{A}$ to represent the elements of the wrappers/remainders. Moreover, every of the following wrappers $\mathcal{W}_i$ will be encoded by a Boolean formula $\mathbb{W}_i$ such that for every valuation π of $\mathcal{A}$, $\pi \in M_{\mathbb{W}_i\downarrow\mathcal{A}}$ iff $\pi_{\mathcal{A},F} \in \mathcal{W}_i$ (and similarly for the remainders).

4.3 $\mathcal{W}_2$ - The Intersection of MUSes

Our second wrapper $\mathcal{W}_2$ is based on a simple observation: every MUS of F has to contain the intersection $\texttt{IMUS}_F$ of all MUSes of F. Hence, we define the wrapper as $\mathcal{W}_2 = \{N \in \mathcal{W}_1 \mid N \supseteq \texttt{IMUS}_F\}$ and encode it via $\mathbb{W}_2$ as follows:

$$\mathbb{W}_2 = \mathbb{W}_1 \wedge \bigwedge_{f_i \in \texttt{IMUS}_F} a_i \tag{3}$$

Proposition 5. *For every valuation π of $\mathcal{A}$, $\pi \in M_{\mathbb{W}_2\downarrow\mathcal{A}}$ iff $\pi_{\mathcal{A},F} \in \mathcal{W}_2$. Consequently, $|M_{\mathbb{W}_2\downarrow\mathcal{A}}| = |\mathcal{W}_2|$.*

The remainder $\mathcal{R}_2$ of $\mathcal{W}_2$ is by Definition 5 the set $\mathcal{W}_2 \cap \texttt{SS}_F$. To build the formula $\mathbb{R}_2$ that encodes $\mathcal{R}_2$, observe that we already have an encoding for the set $\mathcal{W}_2$ (Eq. 3), and we also have an encoding for the set $\texttt{SS}_F$ since $\texttt{SS}_F = \mathcal{R}_1$. Hence, we can build $\mathbb{R}_2$ as a conjunction of the two encodings: $\mathbb{R}_2 = \mathbb{W}_2 \wedge \mathbb{R}_1$. Note that this construction of the remainder and the formula that encodes it is purely mechanical and does not involve any specific property of the particular wrapper. Therefore, for every wrapper $\mathcal{W}_i$ and its encoding $\mathbb{W}_i$ that are presented in the following sections, we define the reminder as $\mathcal{R}_i = \mathcal{W}_i \cap \mathcal{R}_1$ and encode it as $\mathbb{R}_i = \mathbb{W}_i \wedge \mathbb{R}_1$. Proposition 6 witnesses the soundness of this construction:

Proposition 6. *For every valuation π of $\mathcal{A}$, $\pi \in M_{\mathbb{R}_i\downarrow\mathcal{A}}$ iff $\pi_{\mathcal{A},F} \in \mathcal{R}_i$. Consequently, $|M_{\mathbb{R}_i\downarrow\mathcal{A}}| = |\mathcal{R}_i|$.*

This section's final question is how to compute the intersection $\texttt{IMUS}_F$. It is well-known that a clause $f \in F$ belongs to $\texttt{IMUS}_F$ iff $F \setminus \{f\}$ is satisfiable (see,

e.g., [56,32,40]). Hence, a straightforward way would be to perform such satisfiability check for each $f \in F$, however, that might be very expensive. Fortunately, there has been recently proposed [13] a quite efficient algorithm to compute $\texttt{IMUS}_F$ which usually requires only few satisfiability checks, so we implemented that algorithm and use it while building the wrapper.

4.4 $\mathcal{W}_3$ - The Union of MUSes

Our next wrapper, $\mathcal{W}_3$, is very similar to the previous wrapper. Observe that every MUS of F is necessarily a subset of the union $\texttt{UMUS}_F$ of all MUSes of F. Consequently, also a weaker observation holds: every MUS of F is a subset of every over-approximation of $\texttt{UMUS}_F$. We define the wrapper as $\mathcal{W}_3 = \{N \in \mathcal{W}_1 \,|\, N \subseteq U\}$ where U is either the exact union $\texttt{UMUS}_F$ or its over-approximation ($U \supseteq \texttt{UMUS}_F$). Details on obtaining U are provided below. The encoding $\mathbb{W}_3$ of $\mathcal{W}_3$ is analogical to $\mathbb{W}_2$:

$$\mathbb{W}_3 = \mathbb{W}_1 \wedge \bigwedge_{f_i \notin U} \neg a_i \tag{4}$$

Proposition 7. *For every valuation π of $\mathcal{A}$, $\pi \in M_{\mathbb{W}_3 \downarrow \mathcal{A}}$ iff $\pi_{\mathcal{A},F} \in \mathcal{W}_3$. Consequently, $|M_{\mathbb{W}_3 \downarrow \mathcal{A}}| = |\mathcal{W}_3|$.*

The computation of the union $\texttt{UMUS}_F$ has been examined in two recent studies [45,13] that provided two different approaches for that task. Unfortunately, due to the problem's hardness, both the studies showed that the proposed approaches can usually handle only relatively small input formulas. Namely, the approach from [13] requires $\mathcal{O}(|F|)$ calls of a Σ_2^P oracle. Fortunately, it is often possible to cheaply compute a good over-approximation of $\texttt{UMUS}_F$ via the concepts of *autark variables* and a *lean kernel*. Briefly, a subset V of $Vars(F)$ is an *autark* [46] of F iff there exists a valuation χ of V such that for every clause $f \in F$ that contains a variable from V it holds that $\chi \models f$. Since a union of two autark sets is also an autark set, there exists a unique maximum autark set [34,33]. The *lean kernel* K of F is the set of clauses that do not use any variable from the maximum autark set. It has been shown (e.g. [34,33]), that the lean kernel is an over-approximation of $\texttt{UMUS}_F$. Hence, when building the wrapper $\mathcal{W}_3$, we use the lean kernel K as the over-approximation U of $\texttt{UMUS}_F$, i.e., $\mathcal{W}_3 = \{N \in \mathcal{W}_1 \,|\, N \subseteq K\}$. There have been proposed several algorithms to compute the lean kernel, e.g. [43,36]; we have implemented the algorithm by Marques-Silva et al. [43] using a MaxSAT solver UWrMaxSat [54] as a back-end.

Few words are in order to the effect of the two wrappers, $\mathcal{W}_2$ and $\mathcal{W}_3$, on the tractability of the projected model counting. Observe that in both cases ($\mathbb{W}_2$ and $\mathbb{W}_3$), we fix values of some variables from the projection set $\mathcal{A}$. Hence, before passing the formulas to the projected model counter, we first propagate the fixed values of $\mathcal{A}$ to simplify the formulas. By doing so, we effectively reduce the size of the projection set $\mathcal{A}$ by $|\texttt{IMUS}_F|$ and $|U| = |K|$, respectively.

Finally, let us note that the fact that an MUS has to be a subset of the union of all MUSes and a superset of the intersection of all MUSes is well-known and it has been already exploited in various ways in several MUS related studies (see, e.g., [45,11,10]). Especially, the approximate MUS counting algorithm AMUSIC [13] utilizes $\texttt{UMUS}_F$ in its preprocessing phase, and $\texttt{IMUS}_F$ to simplify 3-QBF queries while searching for MUSes.

4.5 $\mathcal{W}_4$ - Minimum MUS Cardinality

Assume we can somehow compute the cardinality of a minimum MUS or at least its lower-bound `minMUS`. Knowing this number, we define our next wrapper as $\mathcal{W}_4 = \{N \in \mathcal{W}_1 \mid |N| \geq \texttt{minMUS}\}$. To encode this wrapper via a formula $\mathbb{W}_4$, we employ a Boolean cardinality constraint $\texttt{atLeast}(\mathcal{A}, \texttt{minMUS})$ expressing that at least `minMUS` variables from $\mathcal{A}$ are set to 1:

$$\mathbb{W}_4 = \mathbb{W}_1 \wedge \texttt{atLeast}(\mathcal{A}, \texttt{minMUS}) \tag{5}$$

Proposition 8. *For every valuation π of $\mathcal{A}$, $\pi \in M_{\mathbb{W}_4 \downarrow \mathcal{A}}$ iff $\pi_{\mathcal{A},F} \in \mathcal{W}_4$. Consequently, $|M_{\mathbb{W}_4 \downarrow \mathcal{A}}| = |\mathcal{W}_4|$.*

There have been proposed several algorithms for computing an MUS with the minimum cardinality, e.g. [27,38,26]. However, since the task of computing a minimum MUS is in $\text{FP}^{\Sigma_2^P}$ [37,27], computing exactly a minimum MUS is too expensive for our scenario (empirically experienced). Instead, we propose an approach for cheaply computing a lower-bound on the minimum MUS cardinality.

Our method is based on a well-known relationship between MUSes and MCSes called *minimal hitting set duality* [56,32]. Given a collection $\mathcal{C}$ of sets, a set X is a *hitting set* of $\mathcal{C}$ iff $C \cap X \neq \emptyset$ for every $C \in \mathcal{C}$. Furthermore, a hitting set X of $\mathcal{C}$ is *minimal* if none of its proper subsets is a hitting set. The duality relation states that a set N is an MUS of F iff N is a minimal hitting set of the set $\texttt{MCS}_F$ of all MCSes of F. Dually, a set M is an MCS of F iff M is a minimal hitting set of the set $\texttt{MUS}_F$. Consequently, one can identify all the MCSes and then compute their *minimum minimal* hitting set to get an MUS with the minimum cardinality. However, there can be up to exponentially many MCSes of F, and thus their complete enumeration is often practically intractable. Our approach to obtain a lower-bound on the minimum MUS cardinality is the following. First, we employ a recent MCS enumeration algorithm RIME [11] to generate a subset $\mathcal{M}$ of $\texttt{MCS}_F$. Subsequently, we compute a minimum minimal hitting set of $\mathcal{M}$ and use it as the lower-bound `minMUS` on the minimum MUS cardinality while building the wrapper $\mathcal{W}_4$. Note that since $\mathcal{M} \subseteq \texttt{MCS}_F$, it holds that every hitting set of $\texttt{MCS}_F$ is also a hitting set of $\mathcal{M}$, and hence `minMUS` is indeed a sound lower-bound on the cardinality of a minimum hitting set of $\texttt{MCS}_F$.

Let us also briefly describe an algorithm for computing the minimum MUS by Ignatiev et al. [27], since it works on a similar principle as our approach. Their algorithm iteratively maintains a set *kMCSes* of known MCSes; initially $kMCes = \emptyset$. In each iteration, the algorithm computes a minimum minimal

hitting set X of *kMCSes* and checks X for satisfiability. If X is unsatisfiable, then it is guaranteed to be a minimum MUS. Otherwise, X is enlarged to an MSS using a single MSS extraction subroutine, the complement of the MSS (i.e., an MCS) is added to *kMCSes*, and the algorithm proceeds with a next iteration. Observe that one can also terminate their approach after a given time limit and use the last computed X as a lower-bound on the minimum MUS cardinality. The main difference between our and their approach is that we employ a dedicated MCS enumerator in the first step and then compute just a single minimum minimal hitting set, whereas they alternate single MCS extraction with minimum minimal hitting set computation.

4.6 $\mathcal{W}_5$ - Maximum MUS Cardinality

Assuming that we can somehow compute an upper-bound `maxMUS` on the maximum cardinality of an MUS of F, we define our next wrapper as $\mathcal{W}_5 = \{N \in \mathcal{W}_1 \mid |N| \leq \texttt{maxMUS}\}$. Similarly as in the case of $\mathbb{W}_4$, to build the formula $\mathbb{W}_5$ that encodes $\mathcal{W}_5$, we introduce a Boolean cardinality constraint $\texttt{atMost}(\mathcal{A}, \texttt{maxMUS})$ expressing that at most `maxMUS` variables from $\mathcal{A}$ are set to 1:

$$\mathbb{W}_5 = \mathbb{W}_1 \wedge \texttt{atMost}(\mathcal{A}, \texttt{maxMUS}) \tag{6}$$

Proposition 9. *For every valuation π of $\mathcal{A}$, $\pi \in M_{\mathbb{W}_5\downarrow\mathcal{A}}$ iff $\pi_{\mathcal{A},F} \in \mathcal{W}_5$. Consequently, $|M_{\mathbb{W}_5\downarrow\mathcal{A}}| = |\mathcal{W}_5|$.*

We are not aware of any prior work on computing the cardinality of the maximum MUS nor of a reasonable approach for computing at least its upper-bound. Hence, we propose a custom approach to compute such an upper-bound `maxMUS`. The base idea is to exploit our concept of wrappers:

Proposition 10. *Let $\mathcal{W}$ be a wrapper, i.e. $\mathcal{W} \subseteq \texttt{MUS}_F \cup \texttt{SS}_F$, $\mathcal{A}$ the set of activation variables, and $\mathbb{W}$ a formula such that for every valuation π of $\mathcal{A}$, $\pi \in M_{\mathbb{W}\downarrow\mathcal{A}}$ iff $\pi_{\mathcal{A},F} \in \mathcal{W}$. Furthermore, let $\texttt{maxOnes} = \texttt{max}(\{\textit{ones}(\pi) \mid \pi \in M_{\mathbb{W}\downarrow\mathcal{A}}\})$ where $\textit{ones}(\pi) = |\{a_i \in \mathcal{A} \mid \pi(a_i) = 1\}|$. Then `maxOnes` is an upper-bound on the maximum MUS cardinality.*

We use `maxOnes` as the value `maxMUS` while constructing wrapper $\mathcal{W}_5$. Any of the wrappers and its encoding presented in this paper can be used as $\mathcal{W}$ and $\mathbb{W}$, respectively. To determine the value `maxOnes`, we define a partial MaxSAT problem using the formula $\mathbb{W} \wedge \bigwedge_{a_i \in \mathcal{A}} a_i$, where $\mathbb{W}$ are the hard clauses and $\bigwedge_{a_i \in \mathcal{A}} a_i$ are the soft clauses. To solve the problem, we employ the MaxSAT solver UWrMaxSat [54].

4.7 $\mathcal{W}_6$ - Component Partitioning

It is often the case that the clauses of F can be partitioned into several *components*, i.e. disjoint subsets of clauses, such that every MUS of F consists only of clauses from a single component. In particular:

Definition 7 (components). *Given a clause $f_i \in F$, the component $\mathcal{C}(f_i)$ of f_i is the minimal subset of F satisfying:*

1. *$f_i \in \mathcal{C}(f)$, and*
2. *for every $l \in f_i$ and every $f_j \in F$ with $\neg l \in f_j$, $\mathcal{C}(f_i) = \mathcal{C}(f_j)$.*

Example 2. Assume that $F = \{\{x_1\}, \{\neg x_1\}, \{x_2\}, \{\neg x_1, \neg x_2\}, \{x_3\}, \{\neg x_3\}, \{x_4\}, \{x_4, x_5\}\}$. There are four components: $C_1 = \{\{x_1\}, \{\neg x_1\}, \{x_2\}, \{\neg x_1, \neg x_2\}\}$, $C_2 = \{\{x_3\}, \{\neg x_3\}\}$, $C_3 = \{\{x_4\}\}$, and $C_4 = \{\{x_4, x_5\}\}$. C_1 has two MUSes: $\{\{x_1\}, \{\neg x_1\}\}$ and $\{\{x_1\}, \{x_2\}, \{\neg x_1, \neg x_2\}\}$, C_2 has one MUS: $\{\{x_3\}, \{\neg x_3\}\}$, and C_3 and C_4 have no MUSes.

Proposition 11. *Let N be an MUS. Then for every two clauses $f_i, f_j \in N$, it holds that $\mathcal{C}(f_i) = \mathcal{C}(f_j)$.*

The wrapper $\mathcal{W}_6$ captures the partition of MUSes into components, and it is defined as $\mathcal{W}_6 = \{N \in \mathcal{W}_1 \mid \forall_{f_i, f_j \in N}. \mathcal{C}(f_i) = \mathcal{C}(f_j)\}$ and encoded via $\mathbb{W}_6$:

$$\mathbb{W}_6 = \mathbb{W}_1 \wedge \bigwedge_{a_i \in \mathcal{A}} (a_i \rightarrow \bigwedge_{f_j \in F \setminus \mathcal{C}(f_i)} \neg a_j) \tag{7}$$

Proposition 12. *For every valuation π of $\mathcal{A}$, $\pi \in M_{\mathbb{W}_6 \downarrow \mathcal{A}}$ iff $\pi_{\mathcal{A},F} \in \mathcal{W}_6$. Consequently, $|M_{\mathbb{W}_6 \downarrow \mathcal{A}}| = |\mathcal{W}_6|$.*

To partition the input formula F into individual components, we construct an undirected graph whose vertices are the clauses of F and every two vertices, f_i and f_j, are connected via an edge iff there exists $l \in f_i$ such that $\neg l \in f_j$. The components of F then correspond to connected components of the graph (which can be identified in linear time w.r.t. the size of F by traversing the graph). Note that a similar *flip graph* has been used in a study [68] on *model rotation* and its usage during single MUS extraction.

4.8 $\mathcal{W}_7$ - Minimal Hitting Set Duality

We again exploit the minimal hitting set duality between MUSes and MCSes (Section 4.5). Recall that if a set M is an MCS of F then $M \cap N \neq \emptyset$ for every $N \in \texttt{MUS}_F$. We define the wrapper $\mathcal{W}_7$ as $\{N \in \mathcal{W}_1 \mid \forall_{M \in \mathcal{M}} M \cap N \neq \emptyset\}$ where $\mathcal{M}$ is a set of MCSes. To obtain $\mathcal{M}$, we run an MCS enumeration algorithm RIME [11] constrained by a user-defined time limit. The encoding $\mathbb{W}_7$ of $\mathcal{W}_7$ is:

$$\mathbb{W}_7 = \mathbb{W}_1 \wedge \bigwedge_{M \in \mathcal{M}} \bigvee_{f_i \in M} a_i \tag{8}$$

Proposition 13. *For every valuation π of $\mathcal{A}$, $\pi \in M_{\mathbb{W}_7 \downarrow \mathcal{A}}$ iff $\pi_{\mathcal{A},F} \in \mathcal{W}_7$. Consequently, $|M_{\mathbb{W}_7 \downarrow \mathcal{A}}| = |\mathcal{W}_7|$.*

4.9 $\mathcal{W}_8$ - Literal Negation Cover

Our next wrapper captures the following observation about MUSes.

Proposition 14. *Let N be an MUS of F, $f_i \in N$ a clause of N, and $l \in f_i$ a literal of f_i. Then there exists a clause $f_j \in N$ such that $\neg l \in f_j$.*

Based on the above proposition, we define the wrapper $\mathcal{W}_8$ as $\mathcal{W}_8 = \{N \in \mathcal{W}_1 \,|\, \forall_{f_i \in N}. \forall_{l \in f_i}. \exists_{f_j \in N}. \neg l \in f_j\}$, and encode it as follows:

$$\mathbb{W}_8 = \mathbb{W}_1 \wedge \bigwedge_{a_i \in \mathcal{A}} a_i \rightarrow \Big(\bigwedge_{l \in f_i} \Big(\bigvee_{f_j \in \{f_j \in F \,|\, \neg l \in f_j\}} a_j \Big)\Big) \tag{9}$$

Proposition 15. *For every valuation π of $\mathcal{A}$, $\pi \in M_{\mathbb{W}_8 \downarrow \mathcal{A}}$ iff $\pi_{\mathcal{A},F} \in \mathcal{W}_8$. Consequently, $|M_{\mathbb{W}_8 \downarrow \mathcal{A}}| = |\mathcal{W}_8|$.*

4.10 $\mathcal{W}_9$ - Non-Extendable Evidence Models

Assume that N is an MUS and $(\rho_1, \ldots, \rho_n)$ is its evidence. By Definition 6, it holds that $\rho_i \models N \backslash \{f_i\}$ for every $1 \leq i \leq n$. Observe that since N is unsatisfiable, then it is also necessarily the case that $\rho_i \models \neg f_i$ for every $1 \leq i \leq n$. Hence, we define our next wrapper, $\mathcal{W}_9$, as $\mathcal{W}_9 = \{N \in \mathcal{W}_1 \,|\, \exists \rho_1, \ldots, \rho_n. \forall_{1 \leq i \leq n}. \rho_i \models N \setminus \{f_i\} \text{ and } \rho_i \models \neg f_i\}$. Note that the above-stated property applies *universally* to every evidence of an MUS, and yet we require in the definition of the wrapper only an *existence* of one such evidence. The reason is that there can be up to exponentially many evidences for an MUS w.r.t. $|Vars(F)|$ and hence it is intractable to reason about all of them in the Boolean encoding of the wrapper.

$$\mathbb{W}_9 = \mathbb{W}_1 \wedge \bigwedge_{a_i \in \mathcal{A}} a_i \rightarrow \neg f_{i[Vars(F)/\mathcal{I}_i]} \tag{10}$$

Proposition 16. *For every valuation π of $\mathcal{A}$, $\pi \in M_{\mathbb{W}_9 \downarrow \mathcal{A}}$ iff $\pi_{\mathcal{A},F} \in \mathcal{W}_9$. Consequently, $|M_{\mathbb{W}_9 \downarrow \mathcal{A}}| = |\mathcal{W}_9|$.*

4.11 $\mathcal{W}_{10}$ - Enforced Evidence Models

Our final wrapper, $\mathcal{W}_{10}$, again builds on the variable valuations $\rho_1, \ldots, \rho_n$ that form an evidence of an MUS N of F. In the previous wrapper, $\mathcal{W}_9$, we have exploited that none of the variable valuations can be a model of N. Here, we express that none of the valuations can be *easily modified* to be a model of N. In particular, if $f_i \in N$, then by the definition of an evidence, $\rho_i \models N \setminus \{f_i\}$. Assume that we pick a literal $l \in f_i$ and turn ρ_i into a valuation ρ_i' by flipping the assignment to l so that $\rho_i' \models f_i$. Since N is an MUS (i.e., unsatisfiable), there necessarily exists a clause $f_j \in N$ such that $\rho_i' \not\models f_j$, i.e., f_j *forces* ρ_i to satisfy $\neg l$ and hence prevents from flipping ρ_i to a model ρ_i' of the whole N. Formally:

Proposition 17. *Let N be an MUS, $f_i \in N$ a clause of N, and ρ_i a model of $N \setminus \{f_i\}$. Then for every literal $l \in f_i$, there exists a clause $f_j \in N$ such that $\neg l \in f_j$ and $\rho_i \not\models f_j \setminus \{\neg l\}$.*

Similarly as in the case of $\mathcal{W}_9$, observe that Proposition 17 applies *universally* to every evidence of an MUS, however, since there can be exponentially many such evidences, it is expensive to reason about all of them. Hence, in the wrapper, we capture just an *existence* of such an evidence: $\mathcal{W}_{10} = \{N \in \mathcal{W}_1 \mid \exists \rho_1, \ldots, \rho_n . \forall_{1 \leq i \leq n} . \rho_i \models N \setminus \{f_i\}$ and if $f_i \in N$ then $\forall_{l \in f_i} . \exists_{f_j \in N} . \neg l \in f_j$ and $\rho_i \not\models f_j \setminus \{\neg l\}\}$. Eq. 11 shows the corresponding encoding via $\mathbb{W}_{10}$:

$$\mathbb{W}_{10} = \mathbb{W}_1 \wedge \bigwedge_{a_i \in \mathcal{A}} a_i \rightarrow \bigwedge_{l \in f_i} \Big(\bigvee_{f_j \in \{f_j \in F \mid \neg l \in f_j\}} a_j \wedge \neg (f_j \setminus \{\neg l\})_{[Vars(F)/\mathcal{I}_i]} \Big) \tag{11}$$

Proposition 18. *For every valuation π of $\mathcal{A}$, $\pi \in M_{\mathbb{W}_{10}\downarrow\mathcal{A}}$ iff $\pi_{\mathcal{A},F} \in \mathcal{W}_{10}$. Consequently, $|M_{\mathbb{W}_{10}\downarrow\mathcal{A}}| = |\mathcal{W}_{10}|$.*

4.12 Combining Wrappers and Their Remainders

In the previous sections, we have presented multiple wrappers, each of which captures a different property of MUSes. In this section, we show that the individual wrappers can be easily combined and, hence, form wrappers that provide a more accurate description of the set $\texttt{MUS}_F$.

Proposition 19. *Let $\mathcal{A}$ be the set of activation variables, $\mathcal{W}^k$ and $\mathcal{W}^l$ wrappers, and $\mathcal{R}^k$ and $\mathcal{R}^l$ the remainders of $\mathcal{W}^k$ and $\mathcal{W}^l$. Furthermore, for every $m \in \{k, l\}$, let $\mathbb{W}^m$ and $\mathbb{R}^m$ be formulas such that:*

- *for every valuation π of $\mathcal{A}$, $\pi \in M_{\mathbb{W}^m\downarrow\mathcal{A}}$ iff $\pi_{\mathcal{A},F} \in \mathcal{W}^m$, and*
- *for every valuation π of $\mathcal{A}$, $\pi \in M_{\mathbb{R}^m\downarrow\mathcal{A}}$ iff $\pi_{\mathcal{A},F} \in \mathcal{R}^m$.*

Then all the following hold:

1. *$\mathcal{W}^k \cap \mathcal{W}^l$ is a wrapper and $\mathcal{R}^k \cap \mathcal{R}^l$ is its reminder.*
2. *For every valuation π of $\mathcal{A}$, $\pi \in M_{(\mathbb{W}^k \wedge \mathbb{W}^l)\downarrow\mathcal{A}}$ iff $\pi_{\mathcal{A},F} \in \mathcal{W}^k \cap \mathcal{W}^l$. Consequently, $|M_{(\mathbb{W}^k \wedge \mathbb{W}^l)\downarrow\mathcal{A}}| = |\mathcal{W}^k \cap \mathcal{W}^l|$.*
3. *For every valuation π of $\mathcal{A}$, $\pi \in M_{(\mathbb{R}^k \wedge \mathbb{R}^l)\downarrow\mathcal{A}}$ iff $\pi_{\mathcal{A},F} \in \mathcal{R}^k \cap \mathcal{R}^l$. Consequently, $|M_{(\mathbb{R}^k \wedge \mathbb{R}^l)\downarrow\mathcal{A}}| = |\mathcal{R}^k \cap \mathcal{R}^l|$.*

Note that although Proposition 19 discusses only a combination of two wrappers, it can be applied repeatedly on already combined wrappers. Hence, we can combine any subset of the wrappers $\mathcal{W}_1, \ldots, \mathcal{W}_{10}$ we proposed. Also, note that all the formulas $\mathbb{W}_2, \ldots, \mathbb{W}_{10}$ subsume the formula $\mathbb{W}_1$, and hence if we combine multiple wrappers, we duplicate some clauses. In our implementation, we first remove all the duplicates and apply other straightforward model preserving simplifications before we pass the encoding to a projected model counter.

5 Experimental Evaluation

We have implemented our approach for counting MUSes in a python-based tool[3], using the projected model counter GANAK [59] to count the models of wrappers and remainders, and also using several auxiliary tools as described above.

We presented 10 *base* wrappers $\mathcal{W}_1, \ldots, \mathcal{W}_{10}$ and shown how to combine them. Since $\mathcal{W}_1$ is subsumed by all the wrappers $\mathcal{W}_2, \ldots, \mathcal{W}_{10}$, there are 2^9 combined wrappers. Due to the large number of the combinations, we were able to evaluate only some of them. In particular, we evaluated the combination $\mathcal{W}_1 \cap \cdots \cap \mathcal{W}_{10}$, denoted as *Wall*, of all wrappers since it provides the most precise description of MUSes. We also evaluated 6 wrappers that emerge from Wall by excluding individual base wrappers or combinations of similar base wrappers, and also the most basic wrapper $\mathcal{W}_1$. The table below shows the names and definitions of the evaluated combinations:

name	definition
W1	$\mathcal{W}_1$
Wno23	$\bigcap_{i\in\{4,5,6,7,8,9,10\}} \mathcal{W}_i$
Wno4	$\bigcap_{i\in\{2,3,5,6,7,8,9,10\}} \mathcal{W}_i$
Wno5	$\bigcap_{i\in\{2,3,4,6,7,8,9,10\}} \mathcal{W}_i$

name	definition
Wno6	$\bigcap_{i\in\{2,3,4,5,7,8,9,10\}} \mathcal{W}_i$
Wno7	$\bigcap_{i\in\{2,3,4,5,6,8,9,10\}} \mathcal{W}_i$
Wno8910	$\bigcap_{i\in\{2,3,4,5,6,7\}} \mathcal{W}_i$
Wall	$\bigcap_{i\in\{2,\ldots,10\}} \mathcal{W}_i$

We also evaluated two contemporary MUS enumerators, MARCO[4] [39] and UNIMUS[5] [10]. Moreover, we evaluated the approximate MUS counter AMUSIC[6] [13] using its default guarantees, i.e., the provided MUS count estimates are within 1.8 multiplicative factor of the true count with 80% confidence.

Our benchmark suite consists of the 2553 instances previously employed in the prior MUS and MSS literature, including those released by authors of AMUSIC [13]. The formulas contain from 78 to 1000 clauses and from 40 to 996 variables. The MUS count varies from 1 to 1.7×10^9 MUSes.

We focus on three comparison criteria: 1) the number of benchmarks solved by the evaluated tools (i.e. benchmarks where the tools provided the MUS count), 2) the scalability of the tools w.r.t. the number of MUSes in the benchmarks, and 3) we examine the *accuracy* of our wrappers.

All experiments were run using a time limit of 3600 seconds per benchmark on a Linux machine with AMD 16-Core Processor and 20GB memory limit. When using wrappers $\mathcal{W}_4$ and $\mathcal{W}_7$, we used a combined limit of 300 seconds (included in the 3600 seconds) and 100000 MCSes for the MCS enumeration while building the wrappers; if both wrappers were used, we run the MCS enumeration just once. Finally, while constructing a combined wrapper of the form $\mathcal{W}_* \cap \mathcal{W}_5$, we used $\mathcal{W}_*$ to compute the value `maxMUS` for creating $\mathcal{W}_5$.

[3] https://github.com/jar-ben/exactMUSCounter
[4] https://sun.iwu.edu/~mliffito/marco/
[5] https://github.com/jar-ben/unimus
[6] https://github.com/jar-ben/amusic

			Our Wrapper-Remainder Based Tools							
AMUSIC	UNIMUS	MARCO	W1	Wno23	Wno4	Wno6	Wno7	Wno8910	Wall	Wno5
623	833	799	403	1475	1498	1486	1445	1058	1486	**1500**

Table 1. Number of solved benchmarks by individual tools.

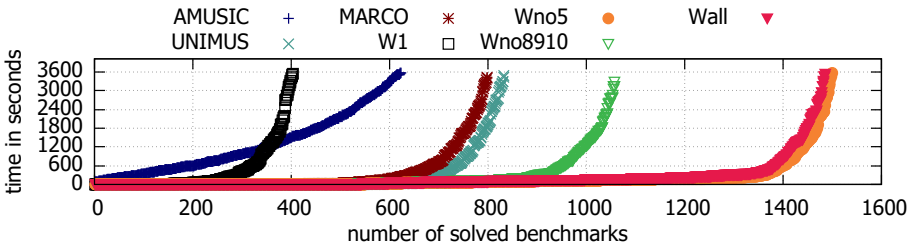

Fig. 2. The number of solved benchmarks in time.

5.1 Solved Benchmarks

In Table 1, we show the number of benchmarks that were solved by the individual evaluated tools. The worst performance was achieved by the basic wrapper W1 ($\mathcal{W}_1$), which is not surprising since it does not provide a good description of MUSes. AMUSIC solved 623 benchmarks, whereas UNIMUS and MARCO solved 833 and 799 benchmarks, respectively. Except for Wno8910 (and W1), which solved *only* 1058 benchmarks, all the remaining combined wrappers solved around 1450-1500 benchmarks and hence significantly dominated both AMUSIC and the two MUS enumerators. Maybe surprisingly, Wall that combines all the base wrappers ended up at the third position; the highest number (1500) of solved benchmarks was achieved by Wno5, and the second-highest (1498) by Wno4. Note that Wno5 and Wno4 exclude encoding of the minimum and maximum MUS cardinality via Boolean cardinality constraints. In general, solving Boolean cardinality constraints is often quite hard, and hence even though a presence of the two wrappers might provide a better description of MUSes, the constraints increase the hardness of the generated instances.

Fig. 2 compares the time needed to solve the benchmarks by a subset (for a better clarity) of the evaluated tools. A point with coordinates $[x, y]$ means that x benchmarks were solved (by the corresponding tool) within the first y seconds.

5.2 Scalability w.r.t the MUS Count

In Fig. 3, we compare the scalability of the evaluated tools w.r.t. the number of MUSes in the benchmarks. In particular, a point with coordinates $[x, y]$ denotes that the corresponding tool solved y benchmarks that contained at most x MUSes. For a better clarity, we compare only our best wrapper, Wno5, with AMUSIC, MARCO, and UNIMUS. Note that whereas AMUSIC scales to instances with 10^8 MUSes, the remaining three tools scale only to instances with at most

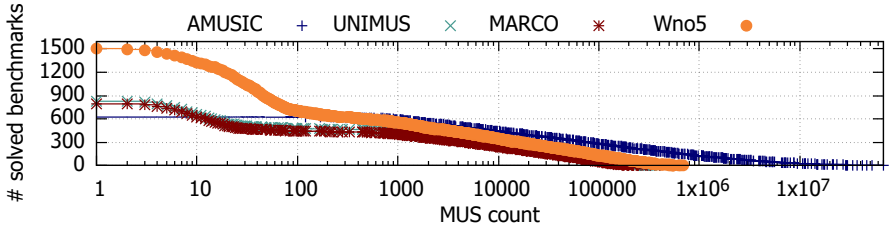

Fig. 3. The number of solved w.r.t. the MUS count.

a million of MUSes. In fact, note that even though AMUSIC solved in overall *just* 623 benchmarks, there are 319 benchmarks that were solved only by AMUSIC. Based on a closer examination of the results, we identified that AMUSIC scales much better than the other tools w.r.t. the MUS count, however, it does not scale so well w.r.t. the number of clauses in the input formula F. This is not surprising since AMUSIC is *just* an approximate counter and as such, it needs to explicitly identify only logarithmically many MUSes w.r.t. $|F|$ even though there can be up to $\mathcal{O}(2^{|F|})$ many MUSes. On the other hand, AMUSIC relies on repeated calls to a 3-QBF solver whose efficiency highly depends on $|F|$.

5.3 Accuracy of Wrappers

Recall that a wrapper $\mathcal{W}$ *over-approximates* the set $\texttt{MUS}_F$ of all MUSes of F, i.e., $\mathcal{W} \supseteq \texttt{MUS}_F$ (Definition 5), and hence we are interested in measuring the *accuracy* of the over-approximations. In particular, given a wrapper $\mathcal{W}$ and its remainder $\mathcal{R}$ constructed over a formula F, we measure the ratio $\frac{|\mathcal{R}|}{|\mathcal{W}|}$. The range of the ratio is $[0, 1)$; the closer to 0 the more accurate the wrapper is, and especially when $\frac{|\mathcal{R}|}{|\mathcal{W}|} = 0$, the wrapper *exactly* captures the set $\texttt{MUS}_F$ (i.e., $\mathcal{W} = \texttt{MUS}_F$).

We illustrate the ratio $\frac{|\mathcal{R}|}{|\mathcal{W}|}$ achieved by individual wrappers in Fig. 4. A point with coordinates $[x, y]$ expresses that for x percent of benchmarks completed by the corresponding tool, the ratio $\frac{|\mathcal{R}|}{|\mathcal{W}|}$ was at most y. As expected, the ratio achieved by the most basic wrapper W1 ($\mathcal{W}_1$) is very high for all the benchmarks, i.e., the wrapper captures $\texttt{MUS}_F$ very inaccurately. On the other hand, the other wrappers achieved for a vast majority of benchmarks a very low ratio, i.e., they over-approximate $\texttt{MUS}_F$ very tightly. In fact, for 87 percent of benchmarks, the wrappers Wno23, Wno4, Wno5, Wno6, and Wall, achieved the ratio 0, i.e., the wrappers exactly captured the set $\texttt{MUS}_F$. In contrast, the wrappers Wno7 and Wno8910 achieved the ratio 0 for *only* 68 and 80 percent of benchmarks, which suggest that the use of the corresponding wrappers, $\mathcal{W}_7$, $\mathcal{W}_8$, $\mathcal{W}_9$, and $\mathcal{W}_{10}$, is vital for an accurate description of $\texttt{MUS}_F$. Moreover, note that the accuracy of the wrappers highly correlate with the number of solved benchmarks (Table 1), since Wno7 and Wno8910 (and W1) were the least efficient wrappers.

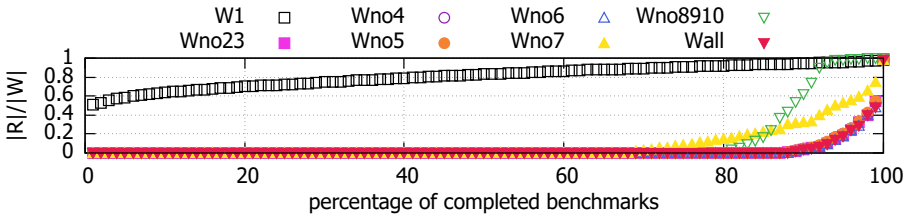

Fig. 4. The ratio $\frac{|\mathcal{R}|}{|\mathcal{W}|}$ expressing the inaccuracy of wrappers.

6 Future Possible Applications of Wrappers and Remainders

Recall that a wrapper $\mathcal{W}$ *over-approximates* the set $\texttt{MUS}_F$ of all MUSes of F, i.e., $\mathcal{W} \supseteq \texttt{MUS}_F$ (Definition 5). Moreover, in Section 5, we empirically witnessed that the best of our wrappers usually over-approximate $\texttt{MUS}_F$ very tightly or they even capture it exactly. Consequently, the propositional encodings $\mathbb{W}$ and $\mathbb{R}$ of a wrapper $\mathcal{W}$ and its remainder $\mathcal{R}$, respectively, can very precisely capture the set $\texttt{MUS}_F$. We strongly believe that such an accurate propositional description of $\texttt{MUS}_F$ paves the way (and will be thoroughly examined in our future work) to efficiently solve many other MUS related problems including, e.g., the following:

Approximate MUS Counting Recall that $|\texttt{MUS}_F| = |\mathcal{W}| - |\mathcal{R}|$. Assuming that $|\mathcal{R}|$ is much smaller than $|\mathcal{W}|$ and observing that $\mathcal{R} \subseteq \mathcal{W}$, computing $|M_{\mathbb{R}\downarrow\mathcal{A}}| = |\mathcal{R}|$ should be much faster than computing $|M_{\mathbb{W}\downarrow\mathcal{A}}| = |\mathcal{W}|$. Hence, one could first relatively quickly *exactly* compute the value $|M_{\mathbb{R}\downarrow\mathcal{A}}|$, and then use an *approximate* model counter to find an *estimate* w' of $|M_{\mathbb{W}\downarrow\mathcal{A}}|$. The MUS count $|\texttt{MUS}_F|$ can be then approximated as $w' - |\mathcal{R}|$. The *accuracy* of the approximation depends on the approximation guarantees of the model counter (e.g. using ApproxMC4 [18,60], we get the (ϵ, δ)-guarantees provided by AMUSIC).

MUS Enumeration Assume a valuation π of the activation variables $\mathcal{A}$ and the corresponding *activated* subset $\pi_{\mathcal{A},F} = \{f_i \in F \mid \pi(a_i) = 1\}$ of F. As shown in Section 4, $\pi_{\mathcal{A},F}$ is an MUS iff $\pi \in M_{\mathbb{W}\downarrow\mathcal{A}}$ and $\pi \notin M_{\mathbb{R}\downarrow\mathcal{A}}$. Hence, one can enumerate MUSes by enumerating projected models of $\mathbb{W}$ and discarding those that are also projected models of $\mathbb{R}$.

MUS Sampling To sample an MUS of F, one can iteratively sample an element π of $M_{\mathbb{W}\downarrow\mathcal{A}}$ until it identifies π such that $\pi \notin M_{\mathbb{R}\downarrow\mathcal{A}}$, i.e., $\pi_{\mathcal{A},F}$ is an MUS. Note that while the past decade has witnessed significant progress in the development of projected model sampling approaches [16,22,55] (with various distribution guarantees), we are not aware of any existing MUS sampling technique (with reasonable distribution guarantees).

Minimum and Maximum MUS Cardinality As discussed in Section 4.6 ($\mathcal{W}_5$), one can over-approximate the maximum MUS cardinality by finding a model $\pi \in M_{\mathbb{W}\downarrow\mathcal{A}}$ that maximizes the number of variables assigned 1. Similarly, one can under-approximate the minimum MUS cardinality by finding a model

$\pi \in M_{\mathbb{W}\downarrow\mathcal{A}}$ that minimizes the number of variables assigned 1. Intuitively, the smaller $|\mathcal{R}|$ is, the more precise approximations can be expected. Moreover, by checking if $\pi \in M_{\mathbb{R}\downarrow\mathcal{A}}$, one can actually verify if $\pi_{\mathcal{A},F}$ is an MUS.

MUS Membership The MUS membership problem is to decide if a clause $f_i \in F$ belongs to an MUS of F and it is known to be Σ_2^P-complete [35,37,31]. Contemporary techniques for deciding the problem are mainly based on solving 2-QBF or 3-QBF encodings [31,13]. Our wrapper-based framework allows for an alternative approach: to decide if a clause f_i belongs to an MUS of F, one can check if there exists a valuation π of $\mathcal{A}$ such that $\pi(a_i) = 1$, $\pi \in M_{\mathbb{W}\downarrow\mathcal{A}}$, and $\pi \notin M_{\mathbb{R}\downarrow\mathcal{A}}$. Note that when $|\mathcal{R}| = 0$ or when $|\mathcal{R}|$ can be bounded by a constant, this check boils down to a single call of a SAT solver.

7 Conclusion and Future Work

In this paper, we focused on the problem of MUS counting and proposed the first exact MUS counter, called CountMUST, that does not rely on explicit MUS enumeration. The base idea is to reduce the problem of MUS counting to (two queries of) projected model counting via the framework of wrappers and remainders. The availability of scalable projected model counter, GANAK, allowed CountMUST to scale much better and solve significantly more instances than other existing approaches. Moreover, as discussed in Section 6, the tightness of wrappers and remainders opens up new potential applications ranging from approximating counting, enumeration, membership, and the like.

We also revisit the complementary nature of CountMUST and AMUSIC with respect to the size of instances and the MUS count. The complementary performance opens up opportunities for a portfolio approach that can achieve the best of both of the worlds. Finally, let us note that we are fighting here the *chicken and egg* nature of the existence of practical applications and scalable algorithmic techniques for problems in automated reasoning. Often the lack of scalable techniques leads to a lack of incentives for end-users to design reductions to practical applications, and vice versa. Even though MUS counting has already many applications in the diagnosis domain [25,48,65,49,50,29], we hope that the availability of CountMUST will break this chicken and egg loop in other areas and enable further investigations into MUS counting applications.

Acknowledgements

This work was supported in part by the National Research Foundation Singapore under its NRF Fellowship Programme [NRF-NRFFAI1-2019-0004] and the AI Singapore Programme [AISG-RP-2018-005], NUS ODPRT Grant [R-252-000-685-13].

References

1. Andraus, Z.S., Liffiton, M.H., Sakallah, K.A.: CEGAR-based formal hardware verification: A case study. Tech. rep., University of Michigan, CSE-TR-531-07 (2007)

2. Arif, M.F., Mencía, C., Ignatiev, A., Manthey, N., Peñaloza, R., Marques-Silva, J.: BEACON: an efficient sat-based tool for debugging *EL+* ontologies. In: SAT. LNCS, vol. 9710, pp. 521–530. Springer (2016)
3. Bacchus, F., Katsirelos, G.: Using minimal correction sets to more efficiently compute minimal unsatisfiable sets. In: CAV (2). LNCS, vol. 9207, pp. 70–86. Springer (2015)
4. Bacchus, F., Katsirelos, G.: Finding a collection of MUSes incrementally. In: CPAIOR. LNCS, vol. 9676, pp. 35–44. Springer (2016)
5. Bailey, J., Stuckey, P.J.: Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In: PADL, pp. 174–186. Springer (2005)
6. Belov, A., Heule, M., Marques-Silva, J.: MUS extraction using clausal proofs. In: SAT. LNCS, vol. 8561, pp. 48–57. Springer (2014)
7. Belov, A., Marques-Silva, J.: MUSer2: An efficient MUS extractor. JSAT **8**, 123–128 (2012)
8. Bendík, J., Beneš, N., Černá, I., Barnat, J.: Tunable online MUS/MSS enumeration. In: FSTTCS. LIPIcs, vol. 65, pp. 50:1–50:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2016)
9. Bendík, J., Černá, I.: MUST: minimal unsatisfiable subsets enumeration tool. In: TACAS (1). LNCS, vol. 12078, pp. 135–152. Springer (2020)
10. Bendík, J., Černá, I.: Replication-guided enumeration of minimal unsatisfiable subsets. In: CP. LNCS, vol. 12333, pp. 37–54. Springer (2020)
11. Bendík, J., Černá, I.: Rotation based MSS/MCS enumeration. In: LPAR. EPiC Series in Computing, vol. 73, pp. 120–137. EasyChair (2020)
12. Bendík, J., Černá, I., Beneš, N.: Recursive online enumeration of all minimal unsatisfiable subsets. In: ATVA. LNCS, vol. 11138, pp. 143–159. Springer (2018)
13. Bendík, J., Meel, K.S.: Approximate counting of minimal unsatisfiable subsets. In: CAV (1). LNCS, vol. 12224, pp. 439–462. Springer (2020)
14. Bendík, J., Meel, K.S.: Counting maximal satisfiable subsets. In: AAAI (2021 (to appear))
15. Chakraborty, S., Meel, K.S., Vardi, M.Y.: A scalable approximate model counter. In: CP. pp. 200–216 (2013)
16. Chakraborty, S., Meel, K.S., Vardi, M.Y.: Balancing scalability and uniformity in SAT witness generator. In: DAC. pp. 60:1–60:6. ACM (2014)
17. Chakraborty, S., Meel, K.S., Vardi, M.Y.: Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic SAT calls. In: IJCAI. pp. 3569–3576. IJCAI/AAAI Press (2016)
18. Chakraborty, S., Meel, K.S., Vardi, M.Y.: Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic sat calls. In: IJCAI (2016)
19. Chen, Z., Toda, S.: The complexity of selecting maximal solutions. Inf. Comput. **119**(2), 231–239 (1995)
20. Cohen, O., Gordon, M., Lifshits, M., Nadel, A., Ryvchin, V.: Designers work less with quality formal equivalence checking. In: Design and Verification Conference (DVCon). Citeseer (2010)
21. Durand, A., Hermann, M., Kolaitis, P.G.: Subtractive reductions and complete problems for counting complexity classes. Theor. Comput. Sci. **340**(3), 496–513 (2005)
22. Dutra, R., Laeufer, K., Bachrach, J., Sen, K.: Efficient sampling of SAT solutions for testing. In: ICSE. pp. 549–559. ACM (2018)
23. Fichte, J.K., Hecher, M., Hamiti, F.: The model counting competition 2020. arXiv preprint arXiv:2012.01323 (2020)

24. Han, B., Lee, S.: Deriving minimal conflict sets by cs-trees with mark set in diagnosis from first principles. IEEE Trans. Systems, Man, and Cybernetics, Part B **29**(2), 281–286 (1999)
25. Hunter, A., Konieczny, S.: Measuring inconsistency through minimal inconsistent sets. In: KR. pp. 358–366. AAAI Press (2008)
26. Ignatiev, A., Janota, M., Marques-Silva, J.: Quantified maximum satisfiability. Constraints An Int. J. **21**(2), 277–302 (2016)
27. Ignatiev, A., Previti, A., Liffiton, M.H., Marques-Silva, J.: Smallest MUS extraction with minimal hitting set dualization. In: CP. LNCS, vol. 9255, pp. 173–182. Springer (2015)
28. Ivrii, A., Malik, S., Meel, K.S., Vardi, M.Y.: On computing minimal independent support and its applications to sampling and counting. Constraints **21**(1) (2016)
29. Jabbour, S., Raddaoui, B., Sais, L.: Inconsistency-based ranking of knowledge bases. In: ICAART (2). pp. 414–419. SciTePress (2015)
30. Jannach, D., Schmitz, T.: Model-based diagnosis of spreadsheet programs: a constraint-based debugging approach. Autom. Softw. Eng. **23**(1), 105–144 (2016)
31. Janota, M., Marques-Silva, J.: On deciding MUS membership with QBF. In: CP. LNCS, vol. 6876, pp. 414–428. Springer (2011)
32. de Kleer, J., Williams, B.C.: Diagnosing multiple faults. Artif. Intell. **32**(1), 97–130 (1987)
33. Kleine Büning, H., Kullmann, O.: Minimal unsatisfiability and autarkies. In: Handbook of Satisfiability, FAIA, vol. 185, pp. 339–401. IOS Press (2009)
34. Kullmann, O.: Investigations on autark assignments. Discrete Applied Mathematics **107**(1-3), 99–137 (2000)
35. Kullmann, O.: Constraint satisfaction problems in clausal form: Autarkies and minimal unsatisfiability. Electronic Colloquium on Computational Complexity (ECCC) **14**(055) (2007)
36. Kullmann, O., Marques-Silva, J.: Computing maximal autarkies with few and simple oracle queries. In: SAT. LNCS, vol. 9340, pp. 138–155. Springer (2015)
37. Liberatore, P.: Redundancy in logic I: CNF propositional formulae. Artif. Intell. **163**(2), 203–232 (2005)
38. Liffiton, M.H., Mneimneh, M.N., Lynce, I., Andraus, Z.S., Marques-Silva, J., Sakallah, K.A.: A branch and bound algorithm for extracting smallest minimal unsatisfiable subformulas. Constraints An Int. J. **14**(4), 415–442 (2009)
39. Liffiton, M.H., Previti, A., Malik, A., Marques-Silva, J.: Fast, flexible MUS enumeration. Constraints **21**(2), 223–250 (2016)
40. Liffiton, M.H., Sakallah, K.A.: Algorithms for computing minimal unsatisfiable subsets of constraints. JAR **40**(1), 1–33 (2008)
41. Luo, J., Liu, S.: Accelerating MUS enumeration by inconsistency graph partitioning. Science China Information Sciences **62**(11), 212104 (2019)
42. Marques-Silva, J., Heras, F., Janota, M., Previti, A., Belov, A.: On computing minimal correction subsets. In: IJCAI. pp. 615–622. IJCAI/AAAI (2013)
43. Marques-Silva, J., Ignatiev, A., Morgado, A., Manquinho, V.M., Lynce, I.: Efficient autarkies. In: ECAI. FAIA, vol. 263, pp. 603–608. IOS Press (2014)
44. Marques-Silva, J., Janota, M.: On the query complexity of selecting few minimal sets. Electronic Colloquium on Computational Complexity (ECCC) **21**, 31 (2014)
45. Mencía, C., Kullmann, O., Ignatiev, A., Marques-Silva, J.: On computing the union of MUSes. In: SAT. LNCS, vol. 11628, pp. 211–221. Springer (2019)
46. Monien, B., Speckenmeyer, E.: Solving satisfiability in less than 2^n steps. Discrete Applied Mathematics **10**(3), 287–295 (1985)

47. Mu, K.: Formulas free from inconsistency: An atom-centric characterization in priest's minimally inconsistent LP. J. Artif. Intell. Res. **66**, 279–296 (2019)
48. Mu, K.: Formulas free from inconsistency: An atom-centric characterization in priest's minimally inconsistent LP (extended abstract). In: IJCAI. pp. 5090–5094. ijcai.org (2020)
49. Mu, K., Liu, W., Jin, Z.: A general framework for measuring inconsistency through minimal inconsistent sets. Knowl. Inf. Syst. **27**(1), 85–114 (2011)
50. Mu, K., Liu, W., Jin, Z.: Measuring the blame of each formula for inconsistent prioritized knowledge bases. J. Log. Comput. **22**(3), 481–516 (2012)
51. Nadel, A., Ryvchin, V., Strichman, O.: Accelerated deletion-based extraction of minimal unsatisfiable cores. JSAT **9**, 27–51 (2014)
52. Narodytska, N., Bjørner, N., Marinescu, M., Sagiv, M.: Core-guided minimal correction set and core enumeration. In: IJCAI. pp. 1353–1361. ijcai.org (2018)
53. Oh, Y., Mneimneh, M.N., Andraus, Z.S., Sakallah, K.A., Markov, I.L.: AMUSE: A minimally-unsatisfiable subformula extractor. In: DAC. pp. 518–523. ACM (2004)
54. Piotrow, M.: Uwrmaxsat: Efficient solver for maxsat and pseudo-boolean problems. In: 2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI). pp. 132–136. IEEE Computer Society, Los Alamitos, CA, USA (nov 2020)
55. Plazar, Q., Acher, M., Perrouin, G., Devroey, X., Cordy, M.: Uniform sampling of SAT solutions for configurable systems: Are we there yet? In: ICST. pp. 240–251. IEEE (2019)
56. Reiter, R.: A theory of diagnosis from first principles. Artif. Intell. **32**(1), 57–95 (1987)
57. Sang, T., Bacchus, F., Beame, P., Kautz, H.A., Pitassi, T.: Combining component caching and clause learning for effective model counting. In: SAT (2004)
58. Sang, T., Beame, P., Kautz, H.A.: Performing bayesian inference by weighted model counting. In: AAAI. pp. 475–482. AAAI Press / The MIT Press (2005)
59. Sharma, S., Roy, S., Soos, M., Meel, K.S.: GANAK: A scalable probabilistic exact model counter. In: IJCAI. pp. 1169–1176. ijcai.org (2019)
60. Soos, M., Meel, K.S.: BIRD: engineering an efficient CNF-XOR SAT solver and its applications to approximate model counting. In: AAAI. pp. 1592–1599. AAAI Press (2019)
61. Soos, M., Nohl, K., Castelluccia, C.: Extending SAT solvers to cryptographic problems. In: SAT. LNCS, vol. 5584, pp. 244–257. Springer (2009)
62. Sperner, E.: Ein satz über untermengen einer endlichen menge. Mathematische Zeitschrift **27**(1), 544–548 (1928)
63. Stockmeyer, L.J.: The complexity of approximate counting (preliminary version). In: STOC. pp. 118–126. ACM (1983)
64. Stuckey, P.J., Sulzmann, M., Wazny, J.: Interactive type debugging in haskell. In: Haskell. pp. 72–83. ACM (2003)
65. Thimm, M.: On the evaluation of inconsistency measures. Measuring Inconsistency in Information **73** (2018)
66. Thurley, M.: sharpsat - counting models with advanced component caching and implicit BCP. In: SAT. LNCS, vol. 4121, pp. 424–429. Springer (2006)
67. Valiant, L.G.: The complexity of enumeration and reliability problems. SIAM J. Comput. **8**(3), 410–421 (1979)
68. Wieringa, S.: Understanding, improving and parallelizing MUS finding using model rotation. In: CP. LNCS, vol. 7514, pp. 672–687. Springer (2012)
69. Xiao, G., Ma, Y.: Inconsistency measurement based on variables in minimal unsatisfiable subsets. In: ECAI. Frontiers in Artificial Intelligence and Applications, vol. 242, pp. 864–869. IOS Press (2012)