On Decomposition of Maximal Satisfiable Subsets

Jaroslav Bendík
Max Planck Institute for Software Systems
Kaiserslautern, Germany
xbendik@mpi-sws.org

***Abstract*—In many areas of computer science, we are given an unsatisfiable formula F in CNF, i.e., a set of clauses, with the goal to analyze the unsatisfiability. A kind of such analysis is to identify Minimal Correction Subsets (MCSes) of F, i.e., minimal subsets of clauses that need to be removed from F to make it satisfiable. Equivalently, one might identify the complements of MCSes, i.e., Maximal Satisfiable Subsets (MSSes) of F. The more MSSes (MCSes) of F are identified, the better insight into the unsatisfiability can be obtained. Hence, there were proposed many algorithms for complete MSS (MCS) enumeration. Unfortunately, the number of MSSes can be exponential w.r.t. $|F|$, which often makes the complete enumeration practically intractable.**

In this work, we attempt to cope with the intractability of complete MSS enumeration by initiating the study on *MSS decomposition*. In particular, we propose several techniques that often allows for decomposing the input formula F into several subformulas. Subsequently, we explicitly enumerate all MSSes of the subformulas, and then combine those MSSes to form MSSes of the original formula F. An extensive empirical study demonstrates that due to the MSS decomposition, the number of MSSes that need to be explicitly identified is often exponentially smaller than the total number of MSSes. Consequently, we are able to improve upon a scalability of contemporary MSS enumeration approaches by many orders of magnitude.

I. Introduction

Boolean formulas in the Conjunctive Normal Form (CNF), wherein we are given a set $F = \{c_1, \ldots, c_n\}$ of Boolean clauses, have been widely adopted as a suitable representation language to model the behaviour of systems and properties. In case we are given an unsatisfiable CNF formula F, the goal is usually to analyze the unsatisfiability. To perform such an analysis, two concepts are often used: a *Minimal Unsatisfiable Subset* (MUS) of F, and a *Minimal Correction Subset* (MCS) of F. Intuitively, an MUS represents a minimal reason for the unsatisfiability, whereas an MCS is a minimal subset of clauses that need to be removed from F to make it satisfiable. A dual notion to an MCS is that of a Maximal Satisfiable Subset (MSS), i.e., a satisfiable subset M of F such that for every clause $c \in F \backslash M$ the set $M \cup \{c\}$ is unsatisfiable. It holds that every MSS is a complement of an MCS of F and vice versa, i.e., MSSes and MCSes represent the same information.

MCSes (MSSes) find many practical applications in various areas of computer science. For instance, in the context of belief update and argumentation, MCSes are used during an update of the belief in the presence of an incoming contradictory belief [16], [21]. Similarly, in the field of diagnosis of constraint systems [5], [37], [49], MCSes represent the constraints that need to be relaxed for the system to be conflict-free. Another application of MSSes arises in the context of the maximum satisfiability problem (MaxSAT), since MSSes with the maximum cardinality correspond to the solutions of MaxSAT. Yet other applications of MCSes can be found, e.g., during model based diagnosis [7], ontology debugging, or axiom pinpointing [1].

Often, it is the case that finding just a single MCS is sufficient. However, in many applications, the task of enumerating several or even all MCSes (MSSes) is crucial for properly understanding the underlying sources of the unsatisfiability. For example, enumeration of minimal correction subsets is essential in software fault localization [30]. In the context of MaxSAT solving, a restricted MSS enumeration is effective in approximately solving the problem if finding the exact solution is intractable [41]. In the domain of diagnosis, there have been proposed many diagnosis metrics that are based on complete enumeration and counting of MSSes and MCSes (see, e.g., [26], [52]). Moreover, there are several computational problems, such as enumeration of minimal unsatisfiable subsets [37], prime implicants [28], and maximal and minimal models [39], that can be reduced to MSS enumeration.

In the past decades, there have been proposed many approaches for enumeration of MSSes (see e.g., [5], [9], [11], [22], [35], [39], [44], [51]). However, the complete MSS enumeration is still often practically intractable [11]. One of the reasons is that the identification of the individual MSSes naturally subsumes checking several subsets of F for satisfiability, and these checks are very expensive (NP-complete). Another issue is that there can be in general exponentially many MSSes of F w.r.t. the number $|F|$ of clauses of F.

In spirit, the intractability of complete MSS enumeration is very similar to the intractability that was dealt with in the context of the Boolean model counting problem. That is, given a Boolean formula H, count all models (satisfying assignments) of H. The earliest approaches for model counting were based on a complete model enumeration, however, since the number of models can be exponential w.r.t. the number of variables of H, the complete model enumeration is often practically intractable. Fortunately, due to an extensive research in the past decades (e.g., [6], [43], [50], [53]), the model counting problem is often practically feasible even for formulas with exponentially many models. A substantial ingredient of contemporary model counters is *decomposition*; in particular, the counters are often able to decompose the input formula H into several independent sub-formulas, then count models of the sub-formulas, and multiply the sub-counts to get the model count for the whole H. At this point, one

https://doi.org/10.34727/2021/isbn.978-3-85448-046-4_30

might wonder *whether it is possibly to perform some kind of a* decomposition *in the context of MSS enumeration?*

In this paper, we initiate the study on the problem of MSS decomposition, and provide an affirmative answer to the above question. In particular, we propose two decomposition techniques that are applicable to some kinds of formulas. The first technique attempts to *directly* decompose the input formula F into several independent components (i.e., disjoint subsets of clauses) based on literals in the individual clauses. Due to the decomposition, we can first identify all MSSes of the individual components (using any existing MSS enumerator), and then form the MSSes of F by just cheaply composing the MSSes of the components. Note that the sum of the MSSes in the individual components can be exponentially smaller than the total number of MSSes of F that we obtain from the composition. The second technique is applicable when the input formula F is not *directly decomposable*. In such a case, we first attempt to identify a suitable *cut* K for F, i.e., a subset K of F such that the formula $F\backslash K$ can be directly decomposed. In this case, we can divide the MSSes of F into two groups: 1) MSSes that are subsets of $F\backslash K$, and 2) the remaining MSSes of F. The former group can be decomposed and solved via the first decomposition technique, whereas the latter group can be identified via any existing MSS enumerator.

Based on the two decomposition techniques, we build a novel MSS enumeration algorithm and experimentally compare it with other contemporary MSS enumeration tools. Out of 1491 benchmarks, the best contemporary approach can solve only 415 benchmarks, whereas our approach solves 788 benchmarks. Moreover, whereas contemporary approaches scale only to instances with at most 10^8 MSSes, our approach can handle even benchmarks with 10^{22} MSSes.

Outline. The rest of the paper is organized as follows. Section II introduces preliminaries and Section III discusses related work. The two decomposition techniques are introduced in Section IV, and our MSS enumeration algorithm is presented in Section V. Section VI provides results of our experimental evaluation. Finally, Section VII discusses practical limitations of our approach, and Section VIII concludes.

II. Preliminaries

Standard definitions for propositional (Boolean) logic are assumed. A Boolean formula F is built over a set $Vars(F)$ of Boolean variables. A *literal* l is either a variable $x \in Vars(F)$ or its negation $\neg x$, and $Lits(F)$ denotes the set of all literals used in F. A *clause* $c = \{l_1, \ldots, l_k\}$ is a set of literals. A Boolean formula in conjunctive normal form $F = \{c_1, \ldots, c_n\}$, shortly *a CNF formula*, is a set of clauses.

Given a CNF formula F, a *valuation* π of $Vars(F)$ is a mapping $\pi : Vars(F) \to \{1, 0\}$. The valuation π *satisfies* a clause $c \in F$ iff there exists a variable x such that $x \in c$ and $\pi(x) = 1$ or $\neg x \in c$ and $\pi(x) = 0$. Moreover, π satisfies F if it satisfies every clause $c \in F$; such a valuation π is called a *model* of F. Finally, F is *satisfiable* if it has a model, and otherwise, F is *unsatisfiable*.

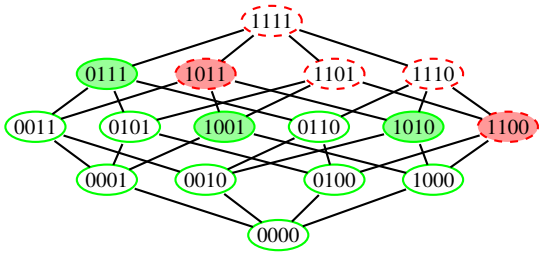

Fig. 1: Illustration of $\mathcal{P}(F)$ from the Example 1. We denote individual subsets of F as bit-vectors, e.g., $\{c_1, c_3\}$ is written as 1010. The subsets with a dashed border are the unsatisfiable subsets, and the others are satisfiable subsets. The MUSes and MSSes are filled with a background color.

Throughout the whole paper, we use $F = \{c_1, \ldots, c_n\}$ to denote the input unsatisfiable CNF formula of interest. Moreover, we write just *formula* instead of *CNF formula*. Finally, given a set X, we write $\mathcal{P}(X)$ to denote the power-set of X, and $|X|$ to denote the cardinality of X.

Definition 1 (MSS)**.** *A set N, $N \subseteq F$, is a* maximal satisfiable subset (MSS) *of F iff N is satisfiable and for every $c \in F\backslash N$ the set $N \cup \{c\}$ is unsatisfiable.*

Definition 2 (MCS)**.** *A set N, $N \subseteq F$, is a* minimal correction subset (MCS) *of F iff $F\backslash N$ is satisfiable and for every $c \in N$ the set $F\backslash(N\backslash\{c\})$ is unsatisfiable. Equivalently, N is an MCS of F iff $F\backslash N$ is an MSS of F.*

Definition 3 (MUS)**.** *A set N, $N \subseteq F$, is a* minimal unsatisfiable subset (MUS) *of F iff N is unsatisfiable and for every $c \in N$ the set $N\backslash\{c\}$ is satisfiable.*

Note that the maximality (minimality) concept used here is a *set maximality (minimality)*, and not a *maximum (minimum) cardinality* as, e.g., in the MaxSAT problem. Consequently, there can be MSSes (MUSes) with different cardinalities, and in general, there can be up to $\mathcal{O}(2^{|F|})$ MSSes (MUSes) of F (intuitively, there are exponentially many pair-wise incomparable subsets of F (w.r.t. the subset inclusion) and all of them can be MSSes (MUSes)). Given a formula N, we write $\texttt{MSS}_N$, $\texttt{MCS}_N$, and $\texttt{MUS}_N$, to denote the set of all MSSes, MCSes, and MUSes of N, respectively. Moreover, given a subset K of N, we write $\texttt{MSS}_N^K$ to denote the set of all MSSes of N that contain at least a single clause from K, i.e., $\texttt{MSS}_N^K = \{M \in \texttt{MSS}_N \mid M \cap K \neq \varnothing\}$.

Example 1. *We illustrate the concepts on a simple example, depicted in Figure 1. Assume that $F = \{c_1 = \{x_1\}, c_2 = \{\neg x_1\}, c_3 = \{x_2\}, c_4 = \{\neg x_1, \neg x_2\}\}$. There are two MUSes: $\texttt{MUS}_F = \{\{c_1, c_2\}, \{c_1, c_3, c_4\}\}$, three MSSes: $\texttt{MSS}_F = \{\{c_1, c_4\}, \{c_1, c_3\}, \{c_2, c_3, c_4\}\}$, and three MCSes: $\texttt{MCS}_F = \{\{c_2, c_3\}, \{c_2, c_4\}, \{c_1\}\}$.*

By the definition, MCSes are exactly the complements of MSSes, and hence finding MSSes is the same as finding MCSes. Both these concepts are used in the literature, since in some situations, it is more suitable to talk about *corrections*,

and in other situations about *maximal satisfiability*. In the rest of the paper, we will stick just to the notion of MSSes and focus on the following problem:

Problem 1. *Given an unsatisfiable CNF formula F, identify the set $\texttt{MSS}_F$ of all MSSes of F.*

When searching for MSSes of a given formula N, it is often possible to *reduce the search-space* via the concepts of *autark variables* and *lean kernel*. A set $A \subseteq Vars(N)$ is an *autark set* for N iff there exists a valuation of A such that every clause of N that uses a variable from A is satisfied by the valuation [42]. Note that a union of two autark sets is also an autark set, and hence there exists a unique maximum autark set of N [31], [32]. The *lean kernel* of N is the set of all clauses of N that do not contain any variable from the maximum autark set. Let L be the lean kernel of N. It is well-known that the set $N \backslash L$ is a subset of every MSS of N (see, e.g., [14], [31], [32]). Furthermore, the following observation holds[1]:

Observation 1. *Let N be a formula and L its lean kernel. Then $\texttt{MSS}_N = \{(N \backslash L) \cup M \mid M \in \texttt{MSS}_L\}$.*

Proof. Let A be the autarky set that corresponds to L, and let π be a valuation of A that satisfies $N \backslash L$.

$\supseteq$: Given $M \in \texttt{MSS}_L$, we show that $(N \backslash L) \cup M \in \texttt{MSS}_N$. First, note that $(N \backslash L) \cup M$ is satisfiable: since $A \cap Vars(M) = \varnothing$, we can combine π with a model π' of M to get a model of $(N \backslash L) \cup M$. Second, by contradiction, assume that there is a clause $c \in L \backslash M$ such that $(N \backslash L) \cup M \cup \{c\}$ has a model ϕ (i.e., $(N \backslash L) \cup M \notin \texttt{MSS}_N$). However, such ϕ is necessarily also a model of $M \cup \{c\}$ which contradicts that $M \in \texttt{MSS}_L$.

$\subseteq$: Given $M' \in \texttt{MSS}_N$, we show that $M = M' \backslash (N \backslash L) \in \texttt{MSS}_L$. Since $M' \supseteq M$ and M' is satisfiable, then M is also satisfiable. Now, by contradiction, assume that $M \notin \texttt{MSS}_L$, i.e., there exists $c \in L \backslash M$ such that $M \cup \{c\}$ is satisfiable with a model ϕ. However, since $Vars(M \cup \{c\}) \cap A = \varnothing$, we can combine ϕ with π to get a model of $M' \cup \{c\}$ which contradicts that $M' \in \texttt{MSS}_N$.

□

In other words, instead of searching for MSSes of the whole N, we can just search for MSSes of the lean kernel of N. If the lean kernel is relatively small, then working just with the kernel can bring a significant runtime and memory improvement.[2] There have been proposed several efficient algorithms for finding maximum autarky sets and the corresponding lean kernels (see, e.g., [33], [40]).

III. Related Work

The problem of MSS (MCS) enumeration was extensively studied in the past decades and many various techniques for the complete enumeration were proposed, e.g., [5], [11], [22], [35], [36], [39], [44], [46]–[48], [51]. Below, we just briefly describe the work-flow of contemporary approaches (for a more detailed overview, please refer to [8]).

Contemporary MSS enumeration approaches gradually explore the power-set of F; *explored subsets* are those whose satisfiability is already determined by the algorithm, and *unexplored* are the other ones. When finding each subsequent MSS M, an MSS enumeration algorithm needs to ensure two things: 1) that M is so far unexplored, and 2) that M is indeed an MSS. Both these tasks are usually carried out via several calls to a SAT solver, and these SAT solver queries are the most time-consuming part of the computation. Despite the fact that extracting just a single MSS is in $\text{FP}^{\text{NP}}[\log]$ [29] (i.e., requiring $\log |F|$ calls to a SAT solver), contemporary MSS enumerators usually need to perform *just* around 1-5 SAT solver calls per MSS (see [11]). Yet, in cases where the number of MSSes is relatively large (or even exponential), the overall number of SAT solver calls is still too high, which makes the complete enumeration practically intractable.

Alternatively, one can identify all MCSes (MSSes) by exploiting the so-called *minimal hitting set duality* [17], [49] between MCSes and MUSes. The duality states that every $M' \in \texttt{MCS}_F$ is a minimal hitting set of $\texttt{MUS}_F$. Hence, one can first identify the set $\texttt{MUS}_F$ via an MUS enumeration approach (e.g., [3]–[5], [9], [10], [12], [18], [24], [25], [35], [37], [44], [46], [51]), and then compute the minimal hitting sets of $\texttt{MUS}_F$ to get all MCSes of F. However, due to potentially exponentially many MUSes w.r.t. $|F|$, the complete MUS enumeration is also often practically intractable.

Recently, we have initiated a study [14] on the problem of counting the number $|\texttt{MSS}_F|$ of MSSes of a given formula F. In particular, we proposed the first MSS counting technique that does not rely on a complete explicit MSS enumeration. Briefly, given a formula F, we defined two Boolean formulas $\texttt{W}$ and $\texttt{R}$ such that $|\texttt{MSS}_F| = M_{\texttt{W}} - M_{\texttt{R}}$, where $M_{\texttt{W}}$ and $M_{\texttt{R}}$ are the number of models of the two formulas, respectively. Therefore, we were able to determine the MSS count via two calls to a model counting tool. Crucially, contemporary model counters often need to explicitly identify just a fraction of the models, i.e., the model-counter somehow *decomposes* the task of identifying/counting MSSes. However, this decomposition is performed on the level of the model counting, whereas in this work, we propose a decomposition scheme that works natively on the structure of MSSes.

Finally, let us note that there were proposed several single MSS extractors, e.g. [2], [20], [23], [41], that are often used as subroutines of contemporary MSS enumerators. Also, there have been proposed several caching techniques, e.g. [47], [48], that can be used to speed up MSS enumerators.

IV. Decomposition of MSSes

In this section, we provide several observations and propose several techniques that can be used to decompose the MSS enumeration problem into multiple easier sub-problems. Subsequently, in Section V, we utilize these techniques to build an efficient MSS enumeration algorithm.

[1] We believe that this observation is also well-known in the community, however, we did not find any work that explicitly formulates and proves it.

[2] Note that we have seen many industrial benchmarks where the lean kernel is indeed relatively small. However, there are also many industrial benchmarks where the lean kernel is the whole formula; in such cases, the extraction of the lean kernel is not useful.

Definition 4 (Decomposition Graph)**.** *Given a formula N, the* decomposition graph *of N, denoted $\mathcal{G}(N)$, is an undirected graph with:*

- *vertices N (a vertex per clause),*
- *and edges $E \subseteq \{\{c_1, c_2\} | c_1, c_2 \in N\}$ such that $\{c_1, c_2\} \in E$ iff there exists $l \in c_1$ with $\neg l \in c_2$.*

Definition 5 (Decomposition)**.** *Given a formula N, the* decomposition *of N, denoted $\mathcal{D}(N)$, is the set of connected components of $\mathcal{G}(N)$ (i.e., $c_1, c_2 \in N$ belong to the same component iff there exists a path between c_1 and c_2 in $\mathcal{G}(N)$).*

Our crucial observation here is that if $|\mathcal{D}(N)| > 1$, then the problem of finding MSSes of N can be solved as follows. First, we identify the MSSes of the individual components in $\mathcal{D}(N)$. Second, we compose the MSSes of the individual components via a *compositional operator* $\sqcup$ into MSSes of the whole N. The compositional operator and our compositional observation is formalized as follows.

Definition 6 ($\sqcup$)**.** *Let $\Omega = \{\mathcal{M}_1, \ldots, \mathcal{M}_p\}$ be a collection of sets of formulas. By $\sqcup(\Omega)$, we denote the set of formulas $\sqcup(\Omega) = \{M_1 \cup \cdots \cup M_p \,|\, M_1 \in \mathcal{M}_1 \wedge \cdots \wedge M_p \in \mathcal{M}_p\}$.*

Proposition 1. *Given a formula N, it holds that $\mathtt{MSS}_N = \sqcup(\{\mathtt{MSS}_C \,|\, C \in \mathcal{D}(N)\})$.*

Proof. Let $\mathcal{D}(N) = \{C_1, \ldots, C_p\}$ and assume a set $M = M_1 \cup \cdots \cup M_p$ such that $M_1 \in \mathtt{MSS}_{C_1} \wedge \cdots \wedge M_p \in \mathtt{MSS}_{C_p}$.

$\supseteq$: Assuming $M \in \sqcup(\{\mathtt{MSS}_C \,|\, C \in \mathcal{D}(N)\})$, we show $M \in \mathtt{MSS}_N$. Let $\pi_1, \ldots, \pi_p$ be models of $M_1, \ldots, M_p$, respectively. W.l.o.g, assume that for every $1 \leqslant k \leqslant p$ and every literal $l \in \mathit{Lits}(M_k)$ such that $\neg l \notin \mathit{Lits}(M_k)$, it holds that π_k satisfies l. By Definition 4, there are no two distinct M_i, M_j with clauses $c_i \in M_i$, $c_j \in M_j$ such that there exists a literal $l \in c_i$ with $\neg l \in c_j$. Consequently, for every two π_i and π_j it holds that they agree on common variables. Hence, we can compose $\pi_1, \ldots, \pi_p$ to form a model of M. To see that M is an MSS of N, assume by contradiction a clause $c \in N \backslash M$ such that $M \cup \{c\}$ is satisfiable. However, this means that there exists $1 \leqslant k \leqslant p$ such that $c \in C_k$ and $M_k \cup \{c\}$ is satisfiable, which contradicts that M_k is an MSS of C_k.

$\subseteq$: Assuming $M \in \mathtt{MSS}_N$, we show $M \in \sqcup(\{\mathtt{MSS}_C \,|\, C \in \mathcal{D}(N)\})$. Since M is satisfiable, then all individual $M_1, \ldots, M_p$ are also satisfiable. Now, by contradiction, assume an M_i that is not an MSS of C_i, i.e., there exists a clause $c \in C_i \backslash M_i$ such that $M_i \cup \{c\}$ has a model π_i. Furthermore, let $\pi_1, \ldots, \pi_{i-1}, \pi_{i+1}, \ldots \pi_p$ be models of $M_1, \ldots, M_{i-1}, M_{i+1}, \ldots M_p$. W.l.o.g, assume that for every $1 \leqslant k \leqslant p$ and every literal $l \in \mathit{Lits}(C_k)$ such that $\neg l \notin \mathit{Lits}(C_k)$, it holds that π_k satisfies l. Same as in $\supseteq$: above, we can compose $\pi_1, \ldots, \pi_p$ to form a model of $M \cup \{c\}$ which contradicts that M is an MSS of N.

□

Example 2. *Let $N = \{c_1 = \{x_1\}, c_2 = \{\neg x_1\}, c_3 = \{x_2\}, c_4 = \{\neg x_2\}, c_5 = \{\neg x_1, \neg x_2\}, c_6 = \{y_1\}, c_7 = \{\neg y_1\}, c_8 = \{y_2\}, c_9 = \{\neg y_1, \neg y_2\}\}$. Here, $\mathcal{D}(N) = \{C_1, C_2\}$, where $C_1 = \{c_1, c_2, c_3, c_4, c_5\}$ and $C_2 = \{c_6, c_7, c_8, c_9\}$. $\mathtt{MSS}_{C_1} = \{\{c_2, c_3, c_5\}, \{c_2, c_4, c_5\}, \{c_1, c_4, c_5\}, \{c_1, c_3\}\}$ and $\mathtt{MSS}_{C_2} = \{\{c_7, c_8, c_9\}, \{c_6, c_8\}, \{c_6, c_9\}\}$. Thus, the whole N has 12 MSSes.*

As witnessed in Example 2, due to Proposition 1, we can substantially reduce the number of MSSes that need to be *explicitly* identified to obtain the whole set $\mathtt{MSS}_N$. Theoretically, it might be even the case that we need to explicitly identify just logarithmically many MSSes w.r.t. $|\mathtt{MSS}_N|$ (assume that N contains $\log_2 |\mathtt{MSS}_N|$ components with 2 MSSes per component). However, from the practical point of view, how often is it the case that we can actually achieve such a reduction? And, moreover, what if $|\mathcal{D}(N)| = 1$, i.e., when Proposition 1 cannot be applied? *Can we still do some decomposition when $|\mathcal{D}(N)| = 1$?* We provide an affirmative answer to this question by finding *decomposition cuts* for N.

Definition 7 (decomposition cut)**.** *Given a formula N such that $|\mathcal{D}(N)| = 1$, a set $K \subsetneq N$ is a* decomposition cut *for N iff $|\mathcal{D}(N \backslash K)| \geqslant 2$.*

Note that decomposition cuts for a formula N correspond to *graph cuts* in the decomposition graph $\mathcal{G}(N)$. Our crucial observation about decomposition cuts is stated in Proposition 2 and Corollary 1.

Proposition 2. *Let N be a formula and K its subset. Then $\mathtt{MSS}_N = \mathtt{MSS}_N^K \cup \{M \in \mathtt{MSS}_{N \backslash K} \,|\, \forall M' \in \mathtt{MSS}_N^K . M \not\subseteq M'\}$.*

Proof. Let us by $\mathtt{MSS}_N^{\overline{K}}$ denote the set of all MSSes of N that do not contain any clause from K. Clearly, $\mathtt{MSS}_M = \mathtt{MSS}_N^K \cup \mathtt{MSS}_N^{\overline{K}}$. To prove Proposition 2, we show that $\mathtt{MSS}_N^{\overline{K}} = \{M \in \mathtt{MSS}_{N \backslash K} \,|\, \forall M' \in \mathtt{MSS}_N^K . M \not\subseteq M'\}$.

$\subseteq$: Assume $M \in \mathtt{MSS}_N^{\overline{K}}$, hence for all $c \in (N \backslash M)$ the set $M \cup \{c\}$ is unsatisfiable, and hence $M \in \mathtt{MSS}_{(N \backslash K)}$. Furthermore, since M is an MSS of N, there cannot exist any $M' \in \mathtt{MSS}_N^K$ with $M \subsetneq M'$.

$\supseteq$: Given $M \in \mathtt{MSS}_{N \backslash K}$ such that $\forall M' \in \mathtt{MSS}_N^K . M \not\subseteq M'$, we show $M \in \mathtt{MSS}_N^{\overline{K}}$. By contradiction, assume that $M \notin \mathtt{MSS}_N^{\overline{K}}$, i.e., there exists $c \in N \backslash M$ such that $M \cup \{c\}$ is satisfiable. Since $M \in \mathtt{MSS}_{N \backslash K}$, then $c \in K$, however, that means that there exists $M' \in \mathtt{MSS}_N^K$ such that $M' \supseteq M \cup \{c\}$.

□

Corollary 1. *Let N be a formula and $K \subsetneq N$ a decomposition cut for N. Then $\mathtt{MSS}_N = \mathtt{MSS}_N^K \cup \{M \in \sqcup(\{\mathtt{MSS}_C \,|\, C \in \mathcal{D}(N \backslash K)\}) \,|\, \forall M' \in \mathtt{MSS}_N^K . M \not\subseteq M'\}$.*

Proof. A direct consequence of Propositions 1 and 2. □

Finally, let us note that graph structures similar to the decomposition graph have been already used in several MUS and MSS related studies (see e.g. the work on *model rotation* [54] or *MUS counting* [13], [15]).

V. Decomposition-Based MSS Enumeration

In this section, we present a novel MSS enumeration algorithm that is based on the *MSS decomposition* observations introduced in the previous section. Moreover, we exploit the concept of the lean kernel which was introduced in Section II.

A. Main Procedure

The main procedure of our algorithm is shown in Algorithm 1. The input is a formula F and the output is the set $\mathtt{MSS}_F$ of all MSSes F. The computation starts by calling a procedure $\mathsf{getKernel}(F)$ that identifies the lean kernel L of F. Based on Observation 1, we can now restrict ourselves just to searching for MSSes of L and then *enlarge* the MSSes of L to MSSes of the whole F. To find MSSes of L, we first use a procedure $\mathsf{getComponents}(L)$ that determines the decomposition $\mathcal{D}(L)$ of L. Subsequently, we iteratively identify all MSSes of the individual components. In particular, each component $N \in \mathcal{D}(L)$ is first checked for satisfiability via a SAT solver (denoted $\mathsf{isSAT}(N)$). If N is satisfiable, then N is the only MSS of N. Otherwise, we use the procedure $\mathsf{processComponent}(N)$ to identify all MSSes of N. We store the sets of MSSes of individual components into an auxiliary set $LMSSparts$. After processing all the components, we exploit Proposition 1 and build the MSSes $\mathtt{MSS}_L$ of L by composing the MSSes of the individual components (stored in $LMSSparts$). Finally, based on Observation 1, we form the set $\mathtt{MSS}_F$ of all MSSes of F by adding the complement $F\backslash L$ of the lean kernel L to the individual MSSes of L.

To implement the procedure $\mathsf{getKernel}(F)$ that identifies a lean kernel of a given formula F, we employ an approach proposed in [40]. To implement the procedure $\mathsf{getComponents}(L)$ that finds the decomposition $\mathcal{D}(L)$ of L, we build the decomposition graph $\mathcal{G}(L)$ and identify its connected components (any graph algorithm for finding connected components can be used). Finally, the procedure $\mathsf{processComponent}(N)$ is more involved and it is described in the following subsection.

B. Processing a Component

The procedure $\mathsf{processComponent}(N)$ (Algorithm 2) starts by computing the lean kernel I of N. Then, we identify a decomposition cut K for I via a procedure $\mathsf{findCut}(I)$. Subsequently, following Corollary 1, we identify all MSSes of I.

In particular, first, we employ an existing MSS enumeration algorithm, denoted $\mathsf{getMSSes}(I, K)$, to identify the set $\mathtt{MSS}_I^K$ of all MSSes of I that contain at least a single clause from K. Subsequently, we use the procedure $\mathsf{getComponents}(I\backslash K)$ to obtain the decomposition $\mathcal{D}(I\backslash K)$ of $I\backslash K$. Then, we iteratively identify all MSSes of individual components $P \in \mathcal{D}(I\backslash K)$ and store the sets of the MSSes into an auxiliary set $IKMSSparts$. Once we process all the components, we can form the MSSes of $I\backslash K$ as $\sqcup(IKMSSparts)$ (Proposition 1). Consequently, following Corollary 1, we can obtain $\mathtt{MSS}_I$ by combining $\mathtt{MSS}_I^K$ and $\sqcup(IKMSSparts)$ (line 8). Finally, to obtain the MSSes of the input set N, we enlarge individual MSSes from $\mathtt{MSS}_I$ by the set $N\backslash I$ (Observation 1).

The procedure $\mathsf{findCut}(I)$ is described in the following subsection. To conclude this subsection, we explain how to implement the procedure $\mathsf{getMSSes}(A, B)$ that identifies all MSS of a formula A that contain at least a single clause from a set B. When $A = B$ (i.e., we look for all MSSes of A (line 7)), we can implement $\mathsf{getMSSes}(A, B)$ by an arbitrary existing MSS enumeration algorithm. In the other case, when $B \subsetneq A$, the situation is more complicated. We are not aware of any existing MSS enumeration tool that would directly allow the user to specify sets A and B and then identify the MSSes of A that contain at least a single clause from B. However, there exist several MSS enumeration algorithms, e.g., [11], [39], that allow the user to specify a subset $B' \subsetneq A$ of *hard clauses* and then identify all MSSes of A that contain *all* clauses in B'. We observe that we can reduce the former task to the latter:

Proposition 3. *Let A and B be formulas such that $B \subsetneq A$. Furthermore, let $A' = A \cup \{c_B\}$ where $c_B = \bigcup_{b \in B} b$. Then $\mathtt{MSS}_A^B = \{M\backslash\{c_B\} \mid M \in \mathtt{MSS}_{A'}^{\{c_B\}}\}$.*

Proof. $\subseteq$: If $M\backslash\{c_B\} \in \mathtt{MSS}_A^B$, then there exists a clause $c \in M \cap B$, and since $M\backslash\{c_B\}$ is satisfiable and $c \subseteq c_B$, then also M is satisfiable. Now, by contradiction, assume that M is not an MSS of $\mathtt{MSS}_A$, i.e., there exists $d \in A\backslash M$ such that $M \cup \{d\}$ is satisfiable, hence $(M \cup \{d\})\backslash\{c_B\}$ is satisfiable (which contradicts that $M\backslash\{c_B\} \in \mathtt{MSS}_A^B$).

$\supseteq$: If $M \in \mathtt{MSS}_{A'}^{\{c_B\}}$, then there necessarily exists a clause $c \subseteq c_B$ such that $c \in B \cap M$. Furthermore, since M is satisfiable, then $M\backslash\{c_B\}$ is also satisfiable. Now, by contradiction, assume that $M\backslash\{c_B\} \notin \mathtt{MSS}_A^B$, i.e., there exists a clause $d \in A\backslash(M\backslash\{c_B\})$ such that $(M\backslash\{c_B\}) \cup \{d\}$ has a model π. Since $c \subseteq c_B$, then π also satisfies $M \cup \{d\}$ which contradicts that $M \in \mathtt{MSS}_{A'}^{\{c_B\}}$. □

Informally, the task of finding MSSes of A that contain at least a single clause from B can be reduced to the task of finding MSSes of A' that contain the hard clause c_B. Namely, in our implementation, we employ the contemporary MSS enumeration tool RIME [11] to carry out $\mathsf{getMSSes}(A, B)$.

Finally, let us note that instead of using an external MSS enumerator to implement $\mathsf{getMSSes}(A, B)$, we could possibly make a recursive call of $\mathsf{processComponent}(\ldots)$ (with some minor modifications) to get the MSSes. That is, we could recursively decompose the input formula into smaller and smaller parts. The reason why we do not do that is explained later in Observation 2. Briefly, every *usable* cut requires existence of two disjoint MUSes in the formula, and based on our empirical experience, industrial benchmarks usually do not contain many disjoint MUSes.

C. Finding a Suitable Decomposition Cut

Recall that finding a decomposition cut K for I with $|\mathcal{D}(I)| = 1$ equals to finding a *graph cut* in the decomposition graph $\mathcal{G}(I)$. Hence, we could use any existing algorithm for finding *cuts in a graph* to find K. However, here we need to find a *suitable* decomposition cut. In the following, we will first describe three properties of a suitable decomposition cut: *Minimality*, *Balance*, and *Necessity*. Subsequently, we describe how to find a decomposition cut with such properties.

For the ease of the presentation, assume that we identify a decomposition cut K for I such that $|\mathcal{D}(I\backslash K)| = 2$, and let us

Algorithm 1: DecExact(F)

```
1  L ← getKernel(F)
2  D(L) ← getComponents(L)
3  LMSSparts ← ∅
4  for N ∈ D(L) do
5      if isSAT(N) then
6          LMSSparts ← LMSSparts ∪ {{N}}
7      else
8          LMSSparts ←
             LMSSparts ∪ {processComponent(N)}
9  MSS_L ← ⊔(LMSSparts)
10 return {(F\L) ∪ M | M ∈ MSS_L}
```

Algorithm 2: processComponent(N)

```
1  I ← getKernel(N)
2  K ← findCut(I)
3  MSS_I^K ← getMSSes(I, K)
4  D(I\K) ← getComponents(I\K)
5  IKMSSparts ← ∅
6  for P ∈ D(I\K) do
7      IKMSSparts ← IKMSSparts ∪ {getMSSes(P, P)}
8  MSS_I ← MSS_I^K ∪ {M ∈ ⊔(IKMSSparts) | ∀M' ∈
     MSS_I^K . M ⊄ M'}
9  return {(N\I) ∪ M | M ∈ MSS_I}
```

by C_1 and C_2 denote the two components of $\mathcal{D}(I\backslash K)$. Hence, in Algorithm 2, it holds that $IKMSSparts = \{\mathtt{MSS}_{C_1}, \mathtt{MSS}_{C_2}\}$.

Minimality Recall that in Algorithm 2, line 8, we build the set $\mathtt{MSS}_I$ as $\mathtt{MSS}_I^K \cup \mathtt{MSS}_I^{\overline{K}}$, where $\mathtt{MSS}_I^{\overline{K}} = \{M \in \sqcup(\{\mathtt{MSS}_{C_1}, \mathtt{MSS}_{C_2}\}) \,|\, \forall M' \in \mathtt{MSS}_I^K . M \not\subseteq M'\}$. Note that whereas the set $\mathtt{MSS}_I^K$ is computed via an external explicit MSS enumerator, i.e., relatively expensively, the set $\mathtt{MSS}_I^{\overline{K}}$ is computed via the decomposition, i.e., relatively cheaply. Consequently, we should attempt to find a decomposition cut K such that $|\mathtt{MSS}_I^K|$ is relatively small (compared to $|\mathtt{MSS}_I^{\overline{K}}|$). Now, observe that since $\mathtt{MSS}_I^K$ contains the MSSes of I that include at least a single clause from K, it holds that the smaller $|K|$ is, the smaller is the maximum possible cardinality of $\mathtt{MSS}_I^K$. Consequently, we should minimize $|K|$.

Balance By Proposition 1, $|\sqcup(\{\mathtt{MSS}_{C_1}, \mathtt{MSS}_{C_2}\})| = |\mathtt{MSS}_{C_1}| \times |\mathtt{MSS}_{C_2}|$. Observe that to maximize $|\sqcup(\{\mathtt{MSS}_{C_1}, \mathtt{MSS}_{C_2}\})|$ while minimizing the number $|\mathtt{MSS}_{C_1}| + |\mathtt{MSS}_{C_2}|$ of MSSes that are needed to build $\sqcup(\{\mathtt{MSS}_{C_1}, \mathtt{MSS}_{C_2}\})$, we should ideally find a decomposition cut K such that $|\mathtt{MSS}_{C_1}|$ and $|\mathtt{MSS}_{C_2}|$ are roughly equal. However, since we do not know in advance what are the MSSes of I, we cannot (cheaply) find a decomposition cut that balances $|\mathtt{MSS}_{C_1}|$ and $|\mathtt{MSS}_{C_2}|$. Instead, we will just try to find a decomposition cut such that $|C_1|$ and $|C_2|$ are roughly equal (and thus the maximal possible number of MSSes in C_1 and C_2 is roughly equal).

Necessity Note in order to ensure that $|\sqcup(\{\mathtt{MSS}_{C_1}, \mathtt{MSS}_{C_2}\})| > |\mathtt{MSS}_{C_1}| + |\mathtt{MSS}_{C_2}|$, it has to hold that $|\mathtt{MSS}_{C_1}| > 1$ and $|\mathtt{MSS}_{C_2}| > 1$. Furthermore, observe that:

Observation 2. *Given a formula X, it holds that $|\mathtt{MSS}_X| > 1$ iff X is unsatisfiable.*

Therefore, for a suitable decomposition cut K, it should hold that both the components C_1 and C_2 are unsatisfiable. All the above three conditions can be straightforwardly generalized for a cut K that yields more than two components.

To find a decomposition cut K with the above three properties, we build a *weighted partial MaxSAT (WPM)* [34] instance and solve it with a MaxSAT solver. In WPM, we are given a tuple $(\mathtt{H}, \mathtt{S}, w : \mathtt{S} \to \mathbb{N}_+)$, where $\mathtt{H}$ is a set of *hard clauses*, $\mathtt{S}$ is a set of *soft clauses*, and w is a weight function that assigns to every soft clause a positive weight. A *solution* of the WPM is a valuation π of $Vars(\mathtt{H} \cup \mathtt{S})$ such that π satisfies all hard clauses and maximizes the sum of the weights of satisfied soft clauses.

In our case, we build $\mathtt{H} \cup \mathtt{S}$ using two sets of Boolean variables: $P = \{p_1, \ldots, p_{|I|}\}$ and $Q = \{q_1, \ldots, q_{|I|}\}$. Note that every valuation π of $P \cup Q$ corresponds to the subsets $\pi_{P,I}$ and $\pi_{Q,I}$ of I defined as $\pi_{P,I} = \{c_i \in I \,|\, \pi(p_i) = 1\}$ and $\pi_{Q,I} = \{c_i \in I \,|\, \pi(q_i) = 1\}$. Furthermore, we write π_K to denote the set $I\backslash(\pi_{P,I} \cup \pi_{Q,I})$. We define a WPM instance $(\mathtt{H}, \mathtt{S}, w : \mathtt{S} \to \mathbb{N}_+)$ in such a way that for every one of its solutions π it holds that: 1) π_K is a decomposition cut for I, and 2) the clauses in $\pi_{P,I}$ and $\pi_{Q,I}$ are disconnected in $\mathcal{G}(I\backslash\pi_K)$, i.e., they *witness* that π_K is a decomposition cut for I. To ease the presentation, we express $\mathtt{H}$ and $\mathtt{S}$ below as plain propositional formulas using the standard Boolean connectives of conjunction ($\wedge$), disjunction ($\vee$) and implication ($\rightarrow$). One can use the Tseitin transformation to convert the formulas to sets of clauses.

The formula (hard clauses) $\mathtt{H}$ is divided into three sub-formulas, $\mathtt{H} = \mathtt{cut} \wedge \mathtt{unsat} \wedge \mathtt{minimal}$. The formula $\mathtt{cut}$ (Equation 1) expresses that π_K is a decomposition cut, and encodes this property via two sub-formulas: $\mathtt{disj}$ and $\mathtt{discn}$. The formula $\mathtt{disj}$ expresses that $\pi_{P,I} \cap \pi_{Q,I} = \emptyset$, whereas $\mathtt{discn}$ encodes that there are no two clauses $c_i \in \pi_{P,I}$ and $c_j \in \pi_{Q,I}$ such that there exists a literal $l \in c_i$ with $\neg l \in c_j$ (i.e. that c_i and c_j are connected in $\mathcal{G}(\pi_K)$). Consequently, the clauses from $\pi_{P,I}$ and $\pi_{Q,I}$ do not belong to a same component of $\mathcal{G}(I\backslash\pi_K)$, and hence, by Definition 7, π_K is a decomposition cut for I. Note that $\mathtt{cut}$ does not enforce that $|\mathcal{D}(I\backslash\pi_K)| = 2$, i.e., $\pi_{Q,I}$ and/or $\pi_{P,I}$ can be fragmented into multiple components in $\mathcal{D}(I\backslash\pi_K)$.

$$
\begin{aligned}
\mathtt{cut} &= \mathtt{disj} \wedge \mathtt{discn}, \text{ where} \\
\mathtt{disj} &= \Big(\bigwedge_{c_i \in I} \neg p_i \vee \neg q_i\Big), \text{ and} \\
\mathtt{discn} &= \bigwedge_{c_i \in I} \Big(\bigwedge_{l \in c_i} \Big(\bigwedge_{c_j \in \{c_j \in I \,|\, \neg l \in c_j\}} \neg p_i \vee \neg q_j \Big)\Big)
\end{aligned}
\tag{1}
$$

The formula $\mathtt{unsat}$ (Equation 2) attempts to encode that both $\pi_{P,I}$ and $\pi_{Q,I}$ are unsatisfiable, i.e., to fulfil the **Necessity**

condition. To ensure this property, we first attempt to identify a pair of disjoint MUSes of I, denoted by M_1 and M_2. Equation 2 expresses that $\pi_{P,I} \supseteq M_1$ and $\pi_{Q,I} \supseteq M_2$, and hence $\pi_{P,I}$ and $\pi_{Q,I}$ are unsatisfiable. To find M_1 and M_2, we enumerate a sequence $X_1, X_2, ...$ of MUSes of I using an MUS enumerator, and for each MUS X_z we check whether $I \backslash X_z$ is unsatisfiable. If there is such an MUS X_z, we use X_z as M_1, and we *shrink* $I \backslash X_z$ to the MUS M_2 via a single MUS extractor. We enumerate only a subset of MUSes of I (limited via a user-definable time limit), and hence, we might fail to identify disjoint MUSes even if there are some. Also, it might be the case that I does not contain disjoint MUSes. In such cases, we set `unsat` to 1 (*True*), i.e, we do not ensure satisfaction of the Necessity condition.

$$\texttt{unsat} = (\bigwedge_{c_i \in M_1} p_i) \wedge (\bigwedge_{c_i \in M_2} q_i) \tag{2}$$

The formula `minimal` (Equation 3) targets the **Minimality** condition. We express that for every $c \in \pi_K$ the set $\pi_K \backslash \{c\}$ is not a decomposition cut for I. Note that the minimality is the minimality in the subset inclusion sense, and not in the cardinality sense. The formula states that every clause $c \in \pi_K$ is connected (in $\mathcal{G}(I \backslash \pi_K)$) to a clause in $\pi_{P,I}$ and to a clause in $\pi_{Q,I}$. Consequently, adding c to $\pi_{P,I}$ ($\pi_{Q,I}$), i.e., flipping the assignment $\pi(p_i)$ ($\pi(q_i)$) to 1, would violate the formula `discn`.

$$\texttt{minimal} = \bigwedge_{c_i \in I} (\neg p_i \wedge \neg q_i) \rightarrow \big((\bigvee_{l \in c_i} (\bigvee_{c_j \in \{c_j \in I \,|\, \neg l\}} p_i)) \wedge (\bigvee_{l \in c_i} (\bigvee_{c_j \in \{c_j \in I \,|\, \neg l\}} q_i))\big) \tag{3}$$

Finally, the *soft formula* (clauses) $\mathtt{S} = \mathtt{S}_1 \wedge \mathtt{S}_2$ is divided into two sub-formulas. $\mathtt{S}_1$ (Equation 4) expresses that every $c \in I$ belongs either to $\pi_{P,I}$ or to $\pi_{Q,I}$, i.e., that π_K is empty. The weight assigned to the clauses of $\mathtt{S}_1$ is $3 \cdot |I|$, which ensures that every solution π of the WPM minimizes $|\pi_K|$. Hence, $\mathtt{S}_1$ further strengthens the **Minimality** condition. $\mathtt{S}_2$ (Equation 5) attempts to fulfil the **Balance** condition. In particular, for every $c_i \in I$, we add two soft clauses, p_i and q_i, and with an equal probability (0.5) we randomly set the weights $w(p_i) = 1$ and $w(q_i) = 2$ or vice versa. Intuitively, the formula `disj` enforces that at most one of p_i and q_i holds, and the weights for $\mathtt{S}_2$ attempt to randomly *push* c_i either towards $\pi_{P,I}$ or $\pi_{Q,I}$.

$$\mathtt{S}_1 = \bigwedge_{c_i \in I} (p_i \vee q_i) \tag{4}$$

$$\mathtt{S}_2 = (\bigwedge_{c_i \in I} p_i) \wedge (\bigwedge_{c_i \in I} q_i) \tag{5}$$

Finally, let us note even if by solving the WPM we obtain a decomposition cut K such that $|\sqcup(\{\mathtt{MSS}_C \,|\, C \in \mathcal{D}(I \backslash K)\})|$ is very large, there is no guarantee that $|\{M \in \sqcup(\{\mathtt{MSS}_C \,|\, C \in \mathcal{D}(I \backslash K)\}) \,|\, \forall M' \in \mathtt{MSS}_I^K . M \not\subset M'\}| > 0$, i.e., the decomposition might not be helpful. Therefore, the three conditions on finding a suitable decomposition cut should be seen as heuristics.

D. Towards Partial MSS Enumeration

Few words are in order concerning the practical tractability of running Algorithm 2. As discussed above, the lean kernel I of the input formula N can possibly contain exponentially many MSSes. Hence the MSS enumeration might be beyond the reach of contemporary MSS enumerators (which usually perform around 1-5 SAT solver calls per MSS [8]). To cope with this intractability, we decompose I into several components, and we hope that the MSSes count for the individual components will be relatively small and thus tractable for a contemporary MSS enumerator. However, note that if there is a component which is still intractable for a contemporary enumerator (calls of getMSSes$(\ldots)$, lines 3 and 7), then Algorithm 2 does not terminate in a reasonable time.

Here, we propose a slight modification of Algorithm 2 that deals with such an intractability. When running getMSSes(A, B), we instruct the underlying MSS enumerator to return at most k MSSes of A, where k can be specified by the user of our algorithm. Consequently, if k is reasonably small, the calls of getMSSes(A, B) become tractable and Algorithm 2 terminates. After such a modification, the sets $\mathtt{MSS}_I^K$ and *IKMSSparts* might be incomplete, and thus the set $\mathtt{MSS}_I$ formed on line 8 can be also incomplete (and hence also the overall set of MSSes returned by Algorithm 1). However, besides the incompleteness, the set $\mathtt{MSS}_I$ might not be sound, i.e., it can contain elements that are not MSSes of I.

In particular, we add to $\mathtt{MSS}_I$ every $M \in \sqcup(IKMSSparts)$ such that $\forall M' \in \mathtt{MSS}_I^K . M \not\subset M'$. Provided that $\mathtt{MSS}_I^K$ is complete, passing the check $\forall M' \in \mathtt{MSS}_I^K . M \not\subset M'$ ensures that M is an MSS of I (Proposition 2). However, if $\mathtt{MSS}_I^K$ is incomplete, then 1) every M that *does not pass* the check *is not* an MSS of I, and 2) every M that *does pass* the check *can be* an MSS of I. Thus, in the case when $\mathtt{MSS}_I^K$ is incomplete, we first check for every M whether it satisfies $\forall M' \in \mathtt{MSS}_I^K . M \not\subset M'$, and if yes, then we also verify that M is an MSS of I using a SAT solver. Such a verification can be performed using a single call of a SAT solver [14] (we check whether $M \wedge (\bigvee_{c \in I \backslash M} c)$ is satisfiable).

VI. Experimental Evaluation

We have implemented our novel approach for MSS/MCS enumeration in a python-based tool using the MSS enumerator RIME [11] to implement the procedure getMSSes, the library PySAT [27] for maintaining CNF formulas, Minisat [19] (accessed via PySAT) as a SAT solver, and UWrMaxSat [45] as a MaxSAT solver. The tool is available at:

https://github.com/jar-ben/MSSDecomposition

Here we provide results of our experimental evaluation. We write DecExact to denote the *complete* MSS enumeration approach as described in Algorithms 1 and 2, and DecApprox to denote the *partial* MSS enumeration version as described in Section V-D. For DecApprox, we set the parameter k to 100000, i.e., every call of getMSSes identifies at most 100000

MSSes. Moreover, we evaluate three contemporary MSS/MCS enumeration algorithms: MARCO[3] [36], FLINT[4] [44], and RIME[5] [11]. In all cases, we used the original implementations of the algorithms with their best (default) settings.

As benchmarks, we used a collection of 1491 Boolean CNF formulas that were used in several recent MSS or MUS related studies. Out of the 1491 formulas, 1200 instances[6] are randomly generated formulas that were first used in [38], and the remaining 291 benchmarks were taken from the MUS track of the SAT Competition 2021[7]. The former benchmarks contain from 100 to 1000 clauses, use from 50 to 996 variables, and have from 2 to at least 10^{22} MSSes (the highest MSS count revealed in our evaluation). The latter benchmarks contain from 70 to 16 million clauses, use from 26 to 4.4 million variables, and have from 2 to at least 10^8 MSSes. We run all experiments on an AMD EPYC 7371 16-Core Processor, 1 TB memory machine running Debian Linux. We used 20 GB memory limit and 3600 seconds (1 hour) time limit per benchmark.

A. Research Questions

We focus on answering the following research questions.

RQ1: Our first research question simply asks: *Can our novel MSS enumeration technique complete the enumeration for more benchmarks than the contemporary approaches?*

RQ2: As discussed above, the proposed MSS decomposition technique can, in a theory, exponentially reduce the number of MSSes that need to be *explicitly* identified. Hence, our novel approach might be able to handle benchmarks with a very large number of MSSes. Our second RQ is thus: *what is the scalability of the evaluated algorithms w.r.t. the number of MSSes in the individual benchmarks?*

RQ3: Finally, we also examine the manifestation of the MSS decomposition in our approach. Our third RQ is: *what is the ratio between the number of* explicitly *identified MSSes and the total number of identified MSSes for the individual benchmarks.*

B. RQ1: Number of Solved Benchmarks

In Figure 2, we show the number of benchmarks for which individual algorithms finished their computation (within the time limit). In particular, a point with coordinate $[x, y]$ means that there are x benchmarks that were finished by the algorithm in at most y seconds. FLINT, RIME, and MARCO were able to identify all MSSes *only* for 364, 376, and 415 benchmarks, respectively. On the other hand, DecExact identified all MSSes for 788 benchmarks, i.e., solving two times as many benchmarks as its competitors. Finally, DecApprox finished the computation for 1240 benchmarks, however, in many cases, it identified only a portion of all MSSes (due to the limit of 100000 MSS per getMSSes call). In particular, DecApprox identified all MSSes for 742 benchmarks, and at least some MSSes for 498 benchmarks.

[3]https://sun.iwu.edu/~mliffito/marco/

[4]The implementation of FLINT was kindly provided to us by its author, Nina Narodytska.

[5]https://github.com/jar-ben/rime

[6]https://github.com/luojie-sklsde/MUS_Random_Benchmarks

[7]http://www.satcompetition.org/

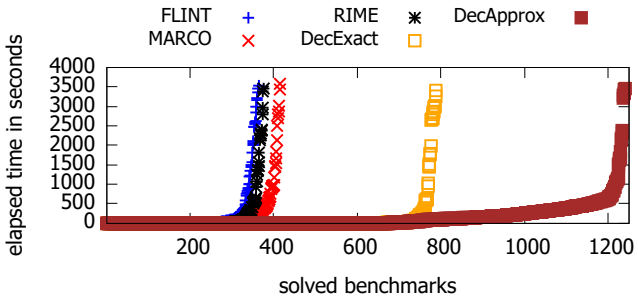

Fig. 2: Number of solved benchmarks.

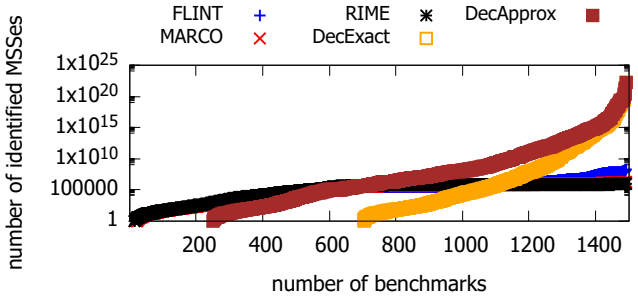

Fig. 3: Scalability w.r.t. the MSS Count

We observed that the tractability of the benchmarks highly correlates with their size (number of clauses). In particular, there are only 16 benchmarks that contain more than 10000 clauses and were solved by at least one of the tools (excluding the incomplete tool DecApprox). Moreover, FLINT, RIME, and MARCO scale better w.r.t. this criterion than DecExact since there are 10 benchmarks that contain more than 500000 clauses (but only up to 20000 MSSes) and were solved by these tools. On the other hand, the largest benchmark solved by DecExact contains only 13236 clauses. We further discuss this bottleneck of our approach in Section VII.

C. RQ2: Scalability W.R.T. the MSS Count

In Figure 3, we compare the scalability of the evaluated algorithms w.r.t. the number of MSSes in the input formulas. In particular, a point with coordinates $[x, y]$ denotes that there are x benchmarks where the corresponding algorithm identified fewer than y MSSes. You can see that MARCO and RIME were able to identify at most only around 10^6 MSSes. FLINT performed slightly better w.r.t. this criterion since for some benchmarks, it identified around 10^8 MSSes. On contrary, both DecExact and DecApprox were able to identify up to 10^{22} MSSes in a benchmark. This witnesses that the use our MSS decomposition techniques allow us to substantially improve the scalability of existing approaches.

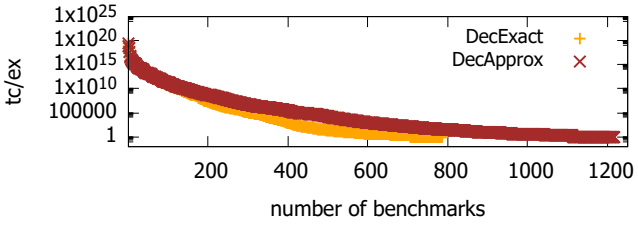

Fig. 4: The ratio between the total number of MSSes and the number of explicitly identified MSSes.

D. RQ3: Number of Explicitly Identified MSSes

Finally, the third research question concerns just our two algorithms, DecExact and DecApprox. Given a formula F, we examine the ratio $\frac{tc}{ex}$, where tc is the total number of identified MSSes of F (i.e., $|\texttt{MSS}_F|$ and an under-approximation of $|\texttt{MSS}_F|$ for DecExact and DecApprox, respectively) and ex is the number of MSSes identified via the calls of getMSSes. A point with coordinates $[x, y]$ in Figure 4 denotes that for the corresponding algorithm, there are x benchmarks where the ratio was at least y. Note that we show the ratio only for the 788 and 1240 benchmarks where DecExact and DecApprox finished the computation.

Recall that getMSSes is implemented via an *explicit MSS enumerator*, i.e., it identifies individual MSSes one by one using sequence of SAT solver calls, i.e., identification of these MSSes is the most expensive part of our algorithm(s). On the other hand, the tc MSSes are identified extremely cheaply since they are built by just composing the MSSes identified via getMSSes. Therefore, the ratio $\frac{tc}{ex}$ actually represents the (maximum possible) speed-up of the MSS enumeration when using DecExact and DecApprox compared to using the *explicit enumerators* FLINT, MARCO, and RIME.

VII. Limitations and Practical Applicability

Even though our novel approaches, DecExact and DecApprox, solved in our evaluation substantially more benchmarks than contemporary MSS enumerators, the practical efficiency of our approaches remains to be unclear. Here, we discuss two main bottlenecks of our approaches and propose ways how to deal with them.

The first bottleneck of our MSS decomposition technique is its reliance on a MaxSAT solver (which is used to find a suitable cut). The size of the formula `cut` (Equation 1) depends on the number $|F|$ of clauses in the input formula F. Hence, for larger input formulas F, solving the MaxSAT problem for `cut` easily becomes practically intractable. A possible way how to deal with this limitation is to use just an *approximate* MaxSAT solver. In particular, recall that our approach for finding a suitable cut via the formula `cut` is just a heuristic, i.e., there is no guarantee that it will indeed find a suitable cut. Using an approximate MaxSAT solver instead of an exact one might increase the scalability of our approach w.r.t. $|F|$.

The second bottleneck of our MSS decomposition technique was stated in Observation 2. In particular, recall there exists a usable cut for a given formula F only if F contains a disjoint pair of MUSes. Based on our empirical experience, there are many applications where the input formula does not contain a disjoint pair of MUSes and hence our approach cannot be applied. Yet, we have also witnessed many industrial benchmarks where disjoint MUSes naturally appear (for instance, there is a SAT encoding of the graph coloring problem where disjoint MUSes correspond to disjoint non-colorable subgraphs). Hence, one might initially check whether the input formula F contains disjoint MUSes and employ our approach only if it is the case.

VIII. Conclusion and Future Work

In this paper, we focused on the problem of enumeration of Maximal Satisfiable Subsets of a given CNF formula F. Despite the fact that the enumeration problem was extensively studied in the past decades, contemporary enumerators are still often unable to finish the computation within a reasonable time limit. The problem is that there can be up to exponentially many MSSes w.r.t. $|F|$ and contemporary approaches usually need to perform a sequence of SAT solver queries to obtain individual MSSes. To combat the combinatorial explosion, we proposed a novel MSS enumeration approach that decomposes F into several smaller sub-formulas, identifies their MSSes, and then compose the MSSes of the sub-formulas to form MSSes of the whole F. Our experimental evaluation witnessed that the decomposition in some cases allows us to identify exponentially more MSSes than other contemporary approaches. Yet, as described in Section VII, the class of benchmarks where our approach can be applied is limited.

We see several directions for future work. A crucial ingredient of our algorithm is the ability to identify a suitable decomposition cut K. The approach for finding K we proposed seems to be quite good, i.e., indeed allowing for a decomposition. However, we believe that there might be even better approaches how to find a suitable decomposition cut. Another direction for future work would be to improve upon the partial MSS enumeration approach (DecApprox). In particular, instead of limiting the number of MSSes returned by getMSSes, one might try to either interleave or parallelize the computation of MSSes of individual components and compose the MSSes on-the-fly. Finally, since our approach is applicable only to a specific class of benchmarks, it might be worth building a portfolio approach.

Acknowledgement

This research was funded in part by the Deutsche Forschungsgemeinschaft project 389792660-TRR 248 and by the European Research Council under the Grant Agreement 610150 (ERC Synergy Grant ImPACT).

References

[1] M. Fareed Arif, Carlos Mencía, and João Marques-Silva. Efficient axiom pinpointing with EL2MCS. In *KI*, volume 9324 of *LNCS*, pages 225–233. Springer, 2015.

[2] Fahiem Bacchus, Jessica Davies, Maria Tsimpoukelli, and George Katsirelos. Relaxation search: A simple way of managing optional clauses. In *AAAI*, pages 835–841. AAAI Press, 2014.

[3] Fahiem Bacchus and George Katsirelos. Using minimal correction sets to more efficiently compute minimal unsatisfiable sets. In *CAV (2)*, volume 9207 of *LNCS*, pages 70–86. Springer, 2015.

[4] Fahiem Bacchus and George Katsirelos. Finding a collection of MUSes incrementally. In *CPAIOR*, volume 9676 of *LNCS*, pages 35–44. Springer, 2016.

[5] James Bailey and Peter J. Stuckey. Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In *PADL*, pages 174–186. Springer, 2005.

[6] Roberto J Bayardo Jr and Joseph Daniel Pehoushek. Counting models using connected components. In *AAAI/IAAI*, pages 157–162, 2000.

[7] Rachel Ben-Eliyahu and Rina Dechter. On computing minimal models. In *AAAI*, pages 2–8. AAAI Press / The MIT Press, 1993.

[8] Jaroslav Bendík. *Minimal Sets over a Monotone Predicate: Enumeration and Counting*. PhD thesis, Masaryk University, 2021.

[9] Jaroslav Bendík, Nikola Beneš, Ivana Černá, and Jiří Barnat. Tunable online MUS/MSS enumeration. In *FSTTCS*, volume 65 of *LIPIcs*, pages 50:1–50:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[10] Jaroslav Bendík and Ivana Černá. Replication-guided enumeration of minimal unsatisfiable subsets. In *CP*, volume 12333 of *LNCS*, pages 37–54. Springer, 2020.

[11] Jaroslav Bendík and Ivana Černá. Rotation based MSS/MCS enumeration. In *LPAR*, volume 73 of *EPiC Series in Computing*, pages 120–137. EasyChair, 2020.

[12] Jaroslav Bendík, Ivana Černá, and Nikola Beneš. Recursive online enumeration of all minimal unsatisfiable subsets. In *ATVA*, volume 11138 of *LNCS*, pages 143–159. Springer, 2018.

[13] Jaroslav Bendík and Kuldeep S. Meel. Approximate counting of minimal unsatisfiable subsets. In *CAV (1)*, volume 12224 of *LNCS*, pages 439–462. Springer, 2020.

[14] Jaroslav Bendík and Kuldeep S. Meel. Counting maximal satisfiable subsets. In *AAAI*, pages 3651–3660. AAAI Press, 2021.

[15] Jaroslav Bendík and Kuldeep S Meel. Counting minimal unsatisfiable subsets. In *CAV*, pages 313–336. Springer, 2021.

[16] Philippe Besnard, Éric Grégoire, and Jean-Marie JM Lagniez. On computing maximal subsets of clauses that must be satisfiable with possibly mutually-contradictory assumptive contexts. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[17] Johan de Kleer and Brian C. Williams. Diagnosing multiple faults. *Artif. Intell.*, 32(1):97–130, 1987.

[18] Maria J. García de la Banda, Peter J. Stuckey, and Jeremy Wazny. Finding all minimal unsatisfiable subsets. In *PPDP*, pages 32–43. ACM, 2003.

[19] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *SAT*, volume 2919 of *LNCS*, pages 502–518. Springer, 2003.

[20] Alexander Felfernig, Monika Schubert, and Christoph Zehentner. An efficient diagnosis algorithm for inconsistent constraint sets. *AI EDAM*, 26(1):53–62, 2012.

[21] Eduardo L. Fermé and Sven Ove Hansson. AGM 25 years - twenty-five years of research in belief change. *J. Philos. Log.*, 40(2):295–331, 2011.

[22] Éric Grégoire, Yacine Izza, and Jean-Marie Lagniez. Boosting mcses enumeration. In *IJCAI*, pages 1309–1315. ijcai.org, 2018.

[23] Éric Grégoire, Jean-Marie Lagniez, and Bertrand Mazure. An experimentally efficient method for (MSS, CoMSS) partitioning. In *AAAI*, pages 2666–2673. AAAI Press, 2014.

[24] Benjamin Han and Shie-Jue Lee. 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] Aimin Hou. A theory of measurement in diagnosis from first principles. *AI*, 65(2):281–328, 1994.

[26] Anthony Hunter and Sébastien Konieczny. Measuring inconsistency through minimal inconsistent sets. In *KR*, pages 358–366. AAAI Press, 2008.

[27] Alexey Ignatiev, António Morgado, and João Marques-Silva. Pysat: A python toolkit for prototyping with SAT oracles. In *SAT*, volume 10929 of *LNCS*, pages 428–437. Springer, 2018.

[28] Saïd Jabbour, João Marques-Silva, Lakhdar Sais, and Yakoub Salhi. Enumerating prime implicants of propositional formulae in conjunctive normal form. In *JELIA*, volume 8761 of *LNCS*, pages 152–165. Springer, 2014.

[29] Mikoláš Janota and Joao Marques-Silva. On the query complexity of selecting minimal sets for monotone predicates. *Artificial Intelligence*, 233:73–83, 2016.

[30] Manu Jose and Rupak Majumdar. Cause clue clauses: error localization using maximum satisfiability. In *PLDI*, pages 437–446. ACM, 2011.

[31] Hans Kleine Büning and Oliver Kullmann. Minimal unsatisfiability and autarkies. In *Handbook of Satisfiability*, volume 185 of *FAIA*, pages 339–401. IOS Press, 2009.

[32] Oliver Kullmann. Investigations on autark assignments. *Discrete Applied Mathematics*, 107(1-3):99–137, 2000.

[33] Oliver Kullmann and João Marques-Silva. Computing maximal autarkies with few and simple oracle queries. In *SAT*, volume 9340 of *LNCS*, pages 138–155. Springer, 2015.

[34] Chu Min Li and Felip Manyà. Maxsat, hard and soft constraints. In *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 613–631. IOS Press, 2009.

[35] Mark H. Liffiton and Ammar Malik. Enumerating infeasibility: Finding multiple MUSes quickly. In *CPAIOR*, volume 7874 of *LNCS*, pages 160–175. Springer, 2013.

[36] Mark H. Liffiton, Alessandro Previti, Ammar Malik, and João Marques-Silva. Fast, flexible MUS enumeration. *Constraints*, 21(2):223–250, 2016.

[37] Mark H. Liffiton and Karem A. Sakallah. Algorithms for computing minimal unsatisfiable subsets of constraints. *JAR*, 40(1):1–33, 2008.

[38] Shaofan Liu and Jie Luo. FMUS2: An efficient algorithm to compute minimal unsatisfiable subsets. In *AISC*, volume 11110 of *LNCS*, pages 104–118. Springer, 2018.

[39] João Marques-Silva, Federico Heras, Mikolás Janota, Alessandro Previti, and Anton Belov. On computing minimal correction subsets. In *IJCAI*, pages 615–622. IJCAI/AAAI, 2013.

[40] João Marques-Silva, Alexey Ignatiev, António Morgado, Vasco M. Manquinho, and Inês Lynce. Efficient autarkies. In *ECAI*, volume 263 of *FAIA*, pages 603–608. IOS Press, 2014.

[41] Carlos Mencía, Alessandro Previti, and João Marques-Silva. Literal-based MCS extraction. In *IJCAI*, pages 1973–1979. AAAI Press, 2015.

[42] Burkhard Monien and Ewald Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discrete Applied Mathematics*, 10(3):287–295, 1985.

[43] Christian Muise, Sheila A McIlraith, J Christopher Beck, and Eric I Hsu. D sharp: fast d-dnnf compilation with sharpsat. In *Canadian Conference on Artificial Intelligence*, pages 356–361. Springer, 2012.

[44] Nina Narodytska, Nikolaj Bjørner, Maria-Cristina Marinescu, and Mooly Sagiv. Core-guided minimal correction set and core enumeration. In *IJCAI*, pages 1353–1361. ijcai.org, 2018.

[45] Marek Piotrów. Uwrmaxsat: Efficient solver for maxsat and pseudo-boolean problems. In *ICTAI*, pages 132–136. IEEE, 2020.

[46] Alessandro Previti and João Marques-Silva. Partial MUS enumeration. In *AAAI*. AAAI Press, 2013.

[47] Alessandro Previti, Carlos Mencía, Matti Järvisalo, and João Marques-Silva. Improving MCS enumeration via caching. In *SAT*, volume 10491 of *LNCS*, pages 184–194. Springer, 2017.

[48] Alessandro Previti, Carlos Mencía, Matti Järvisalo, and João Marques-Silva. Premise set caching for enumerating minimal correction subsets. In *AAAI*, pages 6633–6640. AAAI Press, 2018.

[49] Raymond Reiter. A theory of diagnosis from first principles. *Artif. Intell.*, 32(1):57–95, 1987.

[50] Shubham Sharma, Subhajit Roy, Mate Soos, and Kuldeep S. Meel. GANAK: A scalable probabilistic exact model counter. In *IJCAI*, pages 1169–1176. ijcai.org, 2019.

[51] Roni Tzvi Stern, Meir Kalech, Alexander Feldman, and Gregory M. Provan. Exploring the duality in conflict-directed model-based diagnosis. In *AAAI*. AAAI Press, 2012.

[52] Matthias Thimm. On the evaluation of inconsistency measures. *Measuring Inconsistency in Information*, 73, 2018.

[53] Marc Thurley. sharpsat–counting models with advanced component caching and implicit bcp. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 424–429. Springer, 2006.

[54] Siert Wieringa. Understanding, improving and parallelizing MUS finding using model rotation. In *CP*, volume 7514 of *Lecture Notes in Computer Science*, pages 672–687. Springer, 2012.