International Journal of Approximate Reasoning 55 (2014) 1659–1693

Contents lists available at ScienceDirect

International Journal of Approximate Reasoning

www.elsevier.com/locate/ijar

Computational approaches to finding and measuring inconsistency in arbitrary knowledge bases ☆

Kevin McAreavey [a,*], Weiru Liu [b], Paul Miller [a]

[a] Centre for Secure Information Technologies, Institute of Electronics, Communications and Information Technology (ECIT), Queen's University Belfast, BT3 9DT, Northern Ireland, United Kingdom
[b] School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, BT7 1NN, Northern Ireland, United Kingdom

ARTICLE INFO

Article history:
Received 9 October 2013
Received in revised form 26 May 2014
Accepted 19 June 2014
Available online 27 June 2014

Keywords:
Inconsistency measures
Minimal inconsistent subsets
Minimal unsatisfiable subformulae
SAT
Random SAT

ABSTRACT

There is extensive theoretical work on measures of inconsistency for arbitrary formulae in knowledge bases. Many of these are defined in terms of the set of minimal inconsistent subsets (MISes) of the base. However, few have been implemented or experimentally evaluated to support their viability, since computing all MISes is intractable in the worst case. Fortunately, recent work on a related problem of minimal unsatisfiable sets of clauses (MUSes) offers a viable solution in many cases. In this paper, we begin by drawing connections between MISes and MUSes through algorithms based on a MUS generalization approach and a new optimized MUS transformation approach to finding MISes. We implement these algorithms, along with a selection of existing measures for flat and stratified knowledge bases, in a tool called MIMUS. We then carry out an extensive experimental evaluation of MIMUS using randomly generated arbitrary knowledge bases. We conclude that these measures are viable for many large and complex random instances. Moreover, they represent a practical and intuitive tool for inconsistency handling.

1. Introduction

A fundamental law in classical logic says that if a knowledge base is inconsistent (contains a contradiction), then any statement from its language can be proved through classical reasoning (*ex falso quodlibet*). While this is an uncompromising view, it does highlight the negative implications of inconsistency as well as a potential need to address the underlying causes. Inconsistency has been studied extensively in a range of applications. For example, in network security, a logic-based analysis of inconsistency in an Intrusion Detection System (IDS) was carried out in [49]. In this case the industry standard IDS [64], which was the focus of this work, had a false alarm rate of between 69% [67] and 96% [66] and it was suggested that this was, at least in part, the result of inconsistency in the rule set. Moreover, the size of the knowledge base in this example (involving around 8500 unique rules) emphasizes the need for a formal computational approach to inconsistency handling. A variety of techniques have been proposed for this purpose. In fault-based diagnostics [15], as well as in coherence-based approaches for defeasible reasoning (such as default logic) [7,11] and argumentation systems [1], it is generally the case that reasoning from an inconsistent knowledge base is tolerated through the notion of (preferred) consistent subsets. In the related approach of inconsistency resolution, it is argued that it may be more appropriate to resolve inconsistency. For example, in Requirements Engineering (RE), an inconsistency resolution approach for software requirements

☆ This is a significantly revised and extended version of [50].

* Corresponding author.
E-mail addresses: kmcareavey01@qub.ac.uk (K. McAreavey), w.liu@qub.ac.uk (W. Liu), p.miller@qub.ac.uk (P. Miller).

http://dx.doi.org/10.1016/j.ijar.2014.06.003
0888-613X/

specifications was proposed in [46]. In this work, resolving inconsistency was seen as a useful means to make trade-off decisions between stakeholders. In particular, it was demonstrated that taking into account the *stratification* of different requirements allowed a consistent knowledge base to be recovered while minimizing the loss of important requirements. Inconsistency handling techniques have attracted attention from a range of other applications including knowledge merging [62] and ontology management [51] as well as belief revision and negotiation [33].

In the Artificial Intelligence (AI) community, various methods have been proposed to address the issue of formal inconsistency handling. In general, understanding the nature of inconsistency in a knowledge base is an important requirement. For example, it is well known that simply characterizing a knowledge base as either consistent or inconsistent is of little practical value when dealing with inconsistency. However, it is often the case that an inconsistent knowledge base is the result of more than one instance of inconsistent information. As a consequence of this, the field of quantitative inconsistency measuring has developed a range of measures for characterizing inconsistent knowledge bases in a more meaningful way. In an overview of the area, it is argued in [31] that, given a knowledge base represented as a set of formulae, inconsistency measures can generally be divided into two classes: *atom-centric* measures, which focus on the atoms involved in inconsistency; and *formula-centric* measures, which focus on the formulae involved in inconsistency. In addition to this, it is argued in [31] that these measures can be further classified in terms of the level (in the knowledge base) at which inconsistency is measured, namely:

base-level measures, which assign a single inconsistency value to the knowledge base; and
formula-level measures, which assign an inconsistency value to each formula in the knowledge base w.r.t. the inconsistency in the base as a whole.

More specifically, base-level measures allow us to determine a *degree of inconsistency* for the knowledge base but do not provide any description of the composition of inconsistency in the base. In other words, base-level measures can be more useful than a simple binary truth value that the knowledge base is either consistent or inconsistent but are not discriminative in terms of the syntactic composition of the base. Formula-level measures, on the other hand, allow us to identify a degree of inconsistency for each formula in the base w.r.t. the overall inconsistency of the base. This provides a means to analyze inconsistency in a more precise manner based on the syntactic composition and to define a total order over formulae w.r.t. the inconsistency of the base. Therefore we can say that some formulae have a greater share in the inconsistency of the knowledge base than others. Incorporating additional information, such as stratification of formulae, has also proved beneficial when measuring formula-level inconsistency [48,49].

In a review of the field in [48], it is suggested that formula-level inconsistency measures in the literature have generally been defined in terms of:

minimal inconsistent subsets, i.e., an inconsistent subset of formulae where every strict subset is consistent (removing any one formula will make the set consistent); or
coalitional game models, where each formula is considered a player in a coalitional game (the knowledge base) and a proportional inconsistency value, based on the Shapley value, is distributed between formulae from a base-level measure for the whole knowledge base.

While both approaches for defining formula-level inconsistency measures can also be used to define base-level measures, their real interest lies in analyzing inconsistency syntactically in terms of formulae. This syntax-sensitivity is also necessary for a wide range of applications including the previously mentioned work on network security systems [49] and requirements specifications [46]. Also, it has been said that minimal inconsistent subsets represent the *purest* form of formula-centric inconsistency [63,33,48], so it is intuitive to define formula-level measures using this approach. In fact, an inconsistency measure based on the coalitional game model has also been defined w.r.t. minimal inconsistent subsets [33].

Essentially a minimal inconsistent subset represents a single *cause* of inconsistency (from a formula-centric perspective) in an inconsistent knowledge base and is the basis for the majority of formula-level inconsistency measures presented in the literature. For this reason, this work will focus solely on those measures (whether for flat or stratified knowledge bases) which are defined in terms of minimal inconsistent subsets. Finding the complete set of minimal inconsistent subsets is therefore a fundamental challenge in computing many of these existing formula-level inconsistency measures. However, even though it is assumed that these minimal inconsistent subsets are derivable, work on formula-level inconsistency measures does not generally discuss how they can be derived. Fortunately, in recent years, there has been much work from the boolean satisfiability (SAT) community on a related problem. Specifically, given a boolean formula in conjunctive normal form (CNF) and represented as a set of clauses, this problem deals with finding one, some or all minimal unsatisfiable subformulae (subsets) of this CNF formula. From a computational perspective, there are a number of fundamental issues. For example, the underlying task of determining whether a CNF formula is satisfiable (resp. unsatisfiable) is a classic NP-complete (resp. coNP-complete) problem. Also, determining whether a CNF formula is minimally unsatisfiable is DP-complete [60] while computing a single minimal unsatisfiable subformula is in Σ_2^P [18]. Moreover, the number of minimal unsatisfiable subformulae can be exponential on the number of clauses and thus enumerating them is intractable in the worst case.

Given these hard restrictions however, there have been a number of algorithms proposed which can find minimal unsatisfiable subformulae in many practical instances (including existing difficult SAT benchmarks). In general, the most efficient algorithms which can compute the exhaustive set of solutions involve some form of heuristics [43,28,14] or optimization [29,24,55]. So, while formula-level inconsistency measures based on minimal inconsistent subsets are intractable in the worst case, they may be viable for many practical problems (assuming the number of minimal inconsistent subsets remains tractable). In this work we aim to address possible methods to compute minimal inconsistent subsets from existing work on minimal unsatisfiable subformulae. We also aim to experimentally evaluate the viability of these methods, along with a representative sample of some existing inconsistency measures, using a suitable and sufficiently large dataset.

To address these issues, in this paper we:

1. establish a relationship between minimal inconsistent subsets (from the inconsistency research community) and related concepts from the SAT community, which allows us to explore the use of existing SAT algorithms in order to find minimal inconsistent subsets;
2. propose an alternative sound and complete algorithm for computing minimal inconsistent subsets using existing SAT algorithms;
3. evaluate implementations of these algorithms using a variety of randomly generated (flat and stratified) arbitrary knowledge bases in order to illustrate their viability; and
4. evaluate an implementation of four representative formula-level inconsistency measures based on minimal inconsistent subsets (including stratified measures), using these randomly generated arbitrary knowledge bases.

From this work we conclude that, in many cases, it is practically viable to compute minimal inconsistent subsets and to calculate various formula-level inconsistency measures. Our evaluation demonstrates this with experimental results for random arbitrary knowledge bases obtained from the GENBAL random non-CNF SAT generator [57] (which uses a fixed-shape model for representing arbitrary formulae). In particular, these random arbitrary knowledge bases include those with large amounts of formulae (both simple and complex) and those which are generated in the hard region for these random instances (a well known phenomenon in random SAT). We find that, in some random knowledge bases, it is possible to compute upwards of 60,000 minimal inconsistent subsets within 10 seconds (s). Moreover, we identify a suitable method for finding minimal inconsistent subsets (given certain types of random inconsistent knowledge bases) through a full comparison of two possible approaches. We also find that the actual calculation of inconsistency measures for all formulae in a knowledge base is trivial in comparison to the computation of minimal inconsistent subsets. For example, given the sum of the cardinality of all MISes in a random knowledge base and a sample of random knowledge bases where the mean of this value is under 25,000 for the sample, then the mean time for calculating two selected inconsistency measures for flat knowledge bases was under 10 milliseconds (ms), in all cases. Similarly, where the mean of this value is under 30,000 for the sample, then the mean time for calculating two selected inconsistency measures for stratified knowledge bases was under 1 second (s) in all cases (although the time increases w.r.t. to the number of strata). For this reason, the central focus of this paper is on the issue of computing all MISes, rather than the actual calculation of inconsistency values.

The paper is organized as follows: in Section 2 we introduce notations and formally define minimal inconsistent subsets; in Section 3 we discuss the relationship between existing SAT research and the problem of finding minimal inconsistent subsets; in Section 4 we describe algorithms for computing minimal inconsistent subsets using existing SAT algorithms; in Section 5 we describe a process for generating random arbitrary knowledge bases which we then use to thoroughly evaluate implementations of these algorithms; in Section 6 we discuss the calculation of four representative formula-level inconsistency measures for flat and stratified knowledge bases and evaluate implementations of these measures using our randomly generated knowledge bases; in Section 7 we compare related work; and in Section 8 we conclude the paper.

2. Preliminaries

Set inclusion (resp. strict set inclusion) is denoted by $\subseteq$ (resp. $\subset$). Let $\bigcup S$ (resp. $\bigcap S$) denote the union (resp. intersection) of elements in a set of sets S. Let $|S|$ denote the cardinality of a set S. A *total order* over a set S is a binary relation for all $a, b, \ldots \in S$, denoted $a \succeq b$, which satisfies the following: if $a \succeq b$ and $b \succeq a$ then $a \simeq b$ (**antisymmetry**); if $a \succeq b$ and $b \succeq c$ then $a \succeq c$ (**transitivity**); and $a \succeq b \vee b \succeq a$ (**totality**). Let S be a set where $S_1, \ldots, S_n \subseteq S$, then $[S] = \{S_1, \ldots, S_n\}$ denotes a set of sets representing a partition of S where: (i) $\emptyset \notin [S]$; (ii) $\bigcup [S] = S$; and (iii) $\forall S', S'' \in [S]$ s.t. $S' \neq S''$, $S' \cap S'' = \emptyset$. We call $S_i \in [S]$ a cell of a partitioned set $[S]$.

Let $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$ be vectors where $\boldsymbol{u} = (u_1, u_2, \ldots, u_n)$ and $\boldsymbol{v} = (v_1, v_2, \ldots, v_n)$. If $u_i = v_i$ for each $i \leq n$, then $\boldsymbol{u} = \boldsymbol{v}$. The lexicographical ordering relation between vectors, denoted $\succeq$, is defined as $\boldsymbol{u} \succeq \boldsymbol{v}$ iff

1. $\boldsymbol{u} = \boldsymbol{v}$; or
2. there exists $k \leq n$ s.t. $u_k > v_k$ and $u_i = v_i$ for each $i < k$.

Also, $\boldsymbol{u} \succ \boldsymbol{v}$ iff $\boldsymbol{u} \succeq \boldsymbol{v}$ and $\boldsymbol{u} \neq \boldsymbol{v}$. Let $\mathbf{0} \in \mathbb{R}^n$ denote the n-size vector $(0, \ldots, 0)$.

Let $\mathcal{L}$ denote the propositional language built from a finite set of variables $\mathcal{P}$ using logical connectives $\{\wedge, \vee, \neg, \rightarrow\}$ and logical constants $\{\top, \bot\}$. Let $\phi, \psi, \rho, \ldots$ denote formulae from $\mathcal{L}$ and $p, q, r, \ldots$ denote variables from $\mathcal{P}$. A knowledge base

$K \in 2^{\mathcal{L}}$ is a finite set of arbitrary formulae, interpreted as the conjunction of its elements. Every variable $p \in \mathcal{P}$ is called an atomic formula (atom). A literal ρ is an atom or its negation. A clause ψ is a formula restricted to a disjunction of literals. Let $\mathsf{Lit}(\psi)$ denote the set of literals in a clause ψ. A formula ϕ is in conjunctive normal form (CNF) iff ϕ consists of a conjunction of clauses. Every formula can be translated into an equivalent CNF formula, called normalization, although this can result in an exponential increase in size in the worst case. Let ϕ^* denote the set of clauses in the equivalence-preserving CNF translation of ϕ. A normalized knowledge base is a knowledge base in which every formula is a clause. Let K^* denote a normalized knowledge base from the knowledge base K where $K^* = \{\psi : \psi \in \phi^* \mid \phi \in K\}$.[1] We say that K^* is optimized iff $\forall \psi \in K^*$, $\nexists \psi' \in K^*$ s.t. $\psi \neq \psi'$ and $\mathsf{Lit}(\psi) = \mathsf{Lit}(\psi')$. In other words, K^* is optimized if there are no redundant clauses.

In propositional logic, the syntactic approach in proof theory says that ϕ can be derived from K, denoted $K \vdash \phi$, iff ϕ is provable from K via some inference method. If $K \vdash \phi$ and $K \vdash \neg\phi$ then K is inconsistent. With the semantic approach in model theory, an interpretation or world ω is a function $\omega : \mathcal{P} \mapsto \{0, 1\}$ from $\mathcal{P}$ to the set of boolean values $\{0, 1\}$. Let $2^{\mathcal{P}}$ denote the set of worlds of $\mathcal{L}$. A world ω is a model of K, denoted $\omega \models K$, iff K is true under ω in the classical truth-functional manner. Let $\mathsf{mod}(K)$ denote the set of models of K, i.e., $\mathsf{mod}(K) = \{\omega : \omega \in 2^{\mathcal{P}} \mid \omega \models K\}$. We say that K is satisfiable iff there exists a model of K. Conversely, K is unsatisfiable iff there are no models of K. In propositional logic the syntactic concept of consistency and the semantic concept of satisfiability coincide [22], i.e., a knowledge base K is consistent (resp. inconsistent) iff K is satisfiable (resp. unsatisfiable).

As mentioned previously, work on formula-level inconsistency measures originating from the inconsistency research community is commonly defined in terms of minimal inconsistent subsets of formulae. We formally define this concept, along with the dual concept of maximal consistency, as follows:

Definition 1. Let K be a knowledge base. A minimal inconsistent subset (MIS) Φ of K is a set of formulae s.t.

1. $\Phi \subseteq K$;
2. $\Phi \vdash \bot$; and
3. $\forall \Phi' \subset \Phi$, $\Phi' \nvdash \bot$.

Let $\mathsf{MI}(K)$ denote the set of MISes of K.

Definition 2. Let K be a knowledge base. A maximal consistent subset (MCS) Φ of K is a set of formulae s.t.

1. $\Phi \subseteq K$;
2. $\Phi \nvdash \bot$; and
3. $\forall \Phi' \subseteq K$, if $\Phi \subset \Phi'$ then $\Phi' \vdash \bot$.

Let $\mathsf{MC}(K)$ denote the set of MCSes of K.

Definition 3. Let K be a knowledge base. A formula $\phi \in K$ is called a free formula of K iff $\nexists \Phi \in \mathsf{MI}(K)$ s.t. $\phi \in \Phi$.

Let $\mathsf{FREE}(K)$ denote the set of free formulae of K, i.e., $\mathsf{FREE}(K) = K \setminus \bigcup \mathsf{MI}(K)$. However, the converse is also true, i.e., $\mathsf{FREE}(K) = \bigcap \mathsf{MC}(K)$.

In the SAT community, the equivalent concept for minimal inconsistency (resp. maximal consistency) is minimal unsatisfiability (resp. maximal satisfiability). We formally define these as follows:

Definition 4. Let K^* be a set of clauses. A minimal unsatisfiable subformula (MUS) Ψ of K^* is a set of clauses s.t.

1. $\Psi \subseteq K^*$;
2. $\nexists \omega \in \mathsf{mod}(\Psi)$; and
3. $\forall \Psi' \subset \Psi$ s.t. $\Psi' \neq \emptyset$, $\exists \omega \in \mathsf{mod}(\Psi')$.

Let $\mathsf{MU}(K^*)$ denote the set of MUSes of K^*.

Definition 5. Let K^* be a set of clauses. A maximal satisfiable subformula (MSS) Ψ of K^* is a set of clauses s.t.

1. $\Psi \subseteq K^*$;

[1] Generally, SAT work defines a normalized knowledge base K^* as a set of clauses. In practice however, SAT implementations consider K^* as a multiset of clauses, which means the same clause can appear more than once. For example, $K^* = \{p, p, \neg p\}$ is a valid multiset of clauses. For simplicity, we define a normalized knowledge base K^* as a set of clauses where all elements are treated as unique even if they are identical syntactically. When this is ambiguous, we will always distinguish between syntactically identical clauses by assigning unique identifiers, e.g., $K^* = \{\psi_1, \psi_2, \psi_3\}$ where $\psi_1 = p$, $\psi_2 = p$ and $\psi_3 = \neg p$.

2. $\exists \omega \in \text{mod}(\Psi)$; and
3. $\forall \Psi' \subseteq K^*$, if $\Psi \subset \Psi'$ then $\nexists \omega \in \text{mod}(\Psi')$.

Let $\text{MS}(K^*)$ denote the set of MSSes of K^*.

Given that consistency and satisfiability coincide in propositional logic, MISes and MCSes can also be defined in terms of satisfiability while MUSes and MSSes can be defined in terms of consistency. Clearly MISes (resp. MCSes) and MUSes (resp. MSSes) are compatible, where the former can be viewed as the general case (arbitrary formulae) and the latter can be viewed as a special case (clauses). Note that, in contrast to MISes, singleton MUSes are not possible since a clause cannot be self-contradictory.

Unlike base-level inconsistency measures, there are few properties of formula-level inconsistency measures which are applicable to a variety of measures and which are broadly accepted in the literature. However, we can suggest a general definition of a formula-level inconsistency measure based on MISes as follows:

Definition 6. A formula-level inconsistency measure for a knowledge base is a function $\mathbf{I} : 2^{\mathcal{L}} \times \mathcal{L} \mapsto \mathbb{R}^n$ s.t. $\forall K \in 2^{\mathcal{L}}$ and $\forall \phi, \phi' \in K$:

1. $\mathbf{I}(K, \phi) = \mathbf{0}$ iff $\phi \in \text{FREE}(K)$; and **(consistency)**
2. $\mathbf{I}(K, \phi) = \mathbf{I}(K \setminus \{\phi'\}, \phi)$ if $\phi' \in \text{FREE}(K)$. **(independence)**

Given a knowledge base K, then a formula-level inconsistency measure $\mathbf{I}$ assigns an n-size vector of real numbers to each formula $\phi \in K$, denoted $\mathbf{I}(K, \phi)$, called the inconsistency value of ϕ. When $n = 1$, we denote the formula-level inconsistency measure by I, where I assigns a single real number to ϕ. The term blame (or degree of blame) is a synonym of this inconsistency value [48]. An inconsistency ordering, or blame ordering, is a total order over formulae in a knowledge base w.r.t. a formula-level inconsistency measure. Given a knowledge base K and a formula-level inconsistency measure $\mathbf{I}$, then a formula $\phi \in K$ is *more inconsistent* than another formula $\phi' \in K$ w.r.t. $\mathbf{I}$, denoted $\phi \succ_{\mathbf{I}} \phi'$, iff $\mathbf{I}(K, \phi) > \mathbf{I}(K, \phi')$. In terms of properties, **consistency** says that a formula-level inconsistency measure should only assign the null inconsistency value to free formulae, since these formulae are not involved in the inconsistency of the knowledge base (defined by MISes/MCSes). On the other hand, **independence** says that if a free formula is removed, then this should not effect the formula-level inconsistency value of another formula in the knowledge base. Note that these properties may be too strong to characterize formula-level inconsistency measures which are not based on MISes.

3. Background

The most common problem discussed in relation to MUSes, usually called MUS extraction, refers to computing a single MUS [27,54] (also called an unsatisfiable kernel [53] or unsatisfiable core [8,58,70]), since this problem can often be solved relatively efficiently through a linear traversal of the search space [13]. A related, but more difficult problem is finding a minimum cardinality MUS, called a smallest MUS (SMUS) [40] or a minimum unsatisfiable subset [42]. However, neither problem is directly relevant to the application of formula-level inconsistency measures where the complete set of MISes is required. Fortunately there is some work on the problem of computing *all* MUSes which provides a suitable means to compute the complete set of MISes. Importantly, the underlying approach taken by most of the state-of-the-art solutions to this problem is to avoid expensive unsatisfiability tests (since testing for satisfiability is easier in practice) by computing the dual concept of MSSes instead [6,10,41]. Then, by exploiting a relationship between MSSes and MUSes, the latter set can be computed indirectly. Implementations of state-of-the-art exhaustive MUS algorithms have been shown to perform well for many existing difficult SAT benchmarks involving thousands of clauses [43,29,55,54]. However, since MISes deal with arbitrary formulae and MUSes deal with clauses, the two concepts are not directly equivalent (except when every formula in an arbitrary knowledge base is a clause).

3.1. Associating MUSes and MISes

In the real-world, knowledge bases are usually defined in terms of sets of arbitrary formulae since this is an efficient, intuitive and compact means of representation. For example, IDS rules in [49] and requirements specifications in [46] are always defined in terms of arbitrary formulae. Applying these practical algorithms for computing MUSes to such real-world applications is a useful path for research, but since they require knowledge bases in clausal form, the algorithms cannot be directly applied. Also, when an arbitrary knowledge base is normalized as a set of clauses, the syntactic composition of the knowledge base is lost. So, in the case of measuring inconsistency for example, it is not possible to apply MIS-based formula-level inconsistency measures using MUSes as a substitute for MISes.

Extending MUS research for arbitrary formulae was touched on briefly in [43], where formulae (in CNF already) and clauses were called high-level and low-level constraints, respectively. In this work, two methods for computing MISes (called MUSes of high-level constraints) were suggested:

generalization, whereby sets of clauses are grouped in relation to the formulae from which they originate, then, rather than finding minimal unsatisfiable sets of clauses, minimal unsatisfiable sets of groups of clauses are found instead; or alternatively

transformation, whereby minimal unsatisfiable sets of clauses are found as usual and mapped back to the formulae from which they originate.

Unfortunately there are few technical details of either method and only the generalization approach was actually implemented. However the generalization approach was said to have advantages since the search space could be reduced by eliminating groups of clauses (rather than single clauses only), during the first phase of finding MSSes.

Recently the generalization approach, now called *group MUSes* (see Definition 10 in Section 4.2), has been receiving more attention [13,5,56,28]. Essentially, this approach requires input as a set of clauses and a (non-overlapping) partition of this set representing the group of clauses for each arbitrary formula.

Example 1. Given a knowledge base $K_1 = \{p \wedge q, p \wedge r, \neg p\}$, then $\mathsf{MI}(K_1) = \{\{p \wedge q, \neg p\}, \{p \wedge r, \neg p\}\}$. Also, $K_1^* = \{\psi_1, \psi_2, \ldots, \psi_5\}$ where $\psi_1 = p$, $\psi_2 = q$, $\psi_3 = p$, $\psi_4 = r$ and $\psi_5 = \neg p$. Grouping clauses in K_1^* w.r.t. formulae in K_1 results in a set of clause groups $[K_1^*] = \{\{\psi_1, \psi_2\}, \{\psi_3, \psi_4\}, \{\psi_5\}\}$, representing a partition of K_1^*. Then the group MUSes $\{\{\psi_1, \psi_2\}, \{\psi_5\}\}$ and $\{\{\psi_3, \psi_4\}, \{\psi_5\}\}$ correspond to the MISes $\{p \wedge q, \neg p\}$ and $\{p \wedge r, \neg p\}$, respectively.

In this sense, we can say that the input for this method is not optimized since it may be the case that for an input K^*, there exists a clause $\psi \in K^*$ and another clause $\psi' \in K^*$ s.t. $\mathsf{Lit}(\psi) = \mathsf{Lit}(\psi')$, i.e., where ψ or ψ' is redundant. Given that these approaches were compared in [43] w.r.t. the same input, it is safe to assume that the transformation approach suggested in this work is not optimized either.

Example 2. Given a knowledge base $K_1 = \{p \wedge q, p \wedge r, \neg p\}$, then $K_1^* = \{\psi_1, \psi_2, \ldots, \psi_5\}$ where $\psi_1 = p$, $\psi_2 = q$, $\psi_3 = p$, $\psi_4 = r$ and $\psi_5 = \neg p$. Clearly $\mathsf{Lit}(\psi_1) = \mathsf{Lit}(\psi_3)$, so K_1^* can be optimized as $K_1^* = \{\psi_1, \psi_2, \psi_4, \psi_5\}$.

It is likely that the lack of optimization would be acceptable for the generalization method since an overlapping partition of an optimized set of clauses would only serve to reduce the size of the input, not reduce the number of group MUSes to be found. In contrast, the transformation approach requires the computation of all low-level MUSes, so it is likely that optimization would improve this method because the number of low-level MUSes could be reduced. The reason for this potential improvement is that, given a non-optimized set of clauses K^* with clauses $\psi, \psi' \in K^*$ s.t. $\mathsf{Lit}(\psi) = \mathsf{Lit}(\psi')$, then for every MUS $\Psi \in \mathsf{MU}(K^*)$ s.t. $\psi \in \Psi$, there will be a MUS $\Psi' \in \mathsf{MU}(K^*)$ s.t. $\psi' \in \Psi'$, i.e., Ψ and Ψ' are equivalent. So, given the MUS Ψ, then computing Ψ' can be said to be redundant since Ψ' could be inferred from Ψ and K^*. This type of preprocessing optimization is common in standard SAT problems [17,20] and has also been applied in MUS extraction [5,28].

Example 3. Given a knowledge base $K_1 = \{p \wedge q, p \wedge r, \neg p\}$, then $K_1^* = \{\psi_1, \psi_2, \ldots, \psi_5\}$ where $\psi_1 = p$, $\psi_2 = q$, $\psi_3 = p$, $\psi_4 = r$, and $\psi_5 = \neg p$. So, $\mathsf{MU}(K_1^*) = \{\{\psi_1, \psi_5\}, \{\psi_3, \psi_5\}\}$. If K_1^* is optimized then $K_1^* = \{\psi_1, \psi_2, \psi_4, \psi_5\}$. So, $\mathsf{MU}(K_1^*) = \{\{\psi_1, \psi_5\}\}$. Either way, $\mathsf{MI}(K_1) = \{\{p \wedge q, \neg p\}, \{p \wedge r, \neg p\}\}$.

In cases where a single MUS results in multiple MISes, as is the case in Example 3, an optimized MUS transformation approach may outperform MUS generalization, since extrapolating multiple MISes would likely prove easier than computing multiple MISes directly. However, there are potential issues when moving from traditional MUSes to MISes and these also need to be investigated. For example, an approach to computational argumentation was proposed in [3] whereby support for an argument was found by inserting the negated conclusion of that argument and then computing MUSes. With this method, it was found that one MUS may result in support for an argument which is strictly included in some support generated from a different MUS (Theorem 1 in [3]) and so a post-check for minimality was required. Given these reasons, a formal proposal for a MUS transformation approach is justified since these issues were not addressed by the original discussion in [43]. So, in the following section, we describe the state-of-the-art approach for finding all MUSes in a set of clauses. Then, in order to thoroughly evaluate methods for computing MISes in arbitrary knowledge bases, we present details of the existing MUS generalization approach and propose our own alternative optimized MUS transformation approach.

4. Computing all MISes from MUSes

There are a number of important ways of characterizing inconsistency. We might mention, for example, inconsistent truth assignments in Belnap [2], LP_m [59] or Quasi-classical [4] paraconsistent models as well as the notion of minimal proofs [36]. However, MISes are often seen as the *purest* form of formula-centric inconsistency [63,33,48] since they express inconsistency by means of conflicting formulae. It is because of this intuition that the majority of formula-level inconsistency measures in the literature have been defined in terms of MISes. In fact, it was proved that a simple formula-level measure based on MISes is equivalent to a measure based on the coalitional game model thereby giving MISes the support of

this game theoretic approach [33]. Formula-level inconsistency measures based on MISes are, therefore, a logical and well supported approach to analyzing inconsistency and are important for a wide range of applications [9,46,25,49].

In order to calculate these inconsistency measures for large and complex knowledge bases (assuming the obvious tractable limitations), we establish a link between MUSes and MISes which provides a viable means to compute the complete set of MISes in many cases. So, in this section we present a state-of-the-art method for computing all MUSes in a set of clauses followed by two MUS-based methods for computing all MISes in a knowledge base (including our new optimized MUS transformation proposal).

4.1. Computing all MUSes

The problem of determining whether a set of clauses is satisfiable (SAT) is NP-complete. The converse problem of determining whether a set of clauses is unsatisfiable (UNSAT) is coNP-complete. In practice, solving SAT problems tends to be easier than solving UNSAT problems. This explains why, in practice, finding MSSes directly (via SAT calls) tends to be easier than finding MUSes directly (via UNSAT calls) [43]. For this reason, much of the existing work on computing the set of all MUSes is based on a relationship between MSSes and MUSes which allows the complete set of MUSes to be computed indirectly from the complete set of MSSes. In order to describe this approach however, we must begin by introducing some important concepts.

The first concept is as follows: by definition, a MSS Ψ of a set of clauses K^* is a maximal subset of clauses in K^* which together are satisfiable. This means that given the set complement $\Psi' = K^* \setminus \Psi$, then $\forall \psi \in \Psi'$, $\Psi \cup \{\psi\}$ is unsatisfiable. The set of clauses Ψ' is called the CoMSS[2] of Ψ w.r.t. K^*.

Definition 7. Let K^* be a set of clauses and $\mathsf{MS}(K^*)$ be the set of MSSes of K^*. The set of CoMSSes of K^* is a set of sets of clauses, denoted $\mathsf{MS}^c(K^*)$, defined as:

$$\mathsf{MS}^c\big(K^*\big) = \big\{K^* \setminus \Psi \;\big|\; \Psi \in \mathsf{MS}\big(K^*\big)\big\}.$$

We can demonstrate this with a simple example:

Example 4. Given a set of clauses $K_2^* = \{p, \neg p, \neg q, \neg p \vee q, r\}$, then

$$\mathsf{MS}\big(K_2^*\big) = \big\{\{\neg p, \neg q, \neg p \vee q, r\}, \{p, \neg p \vee q, r\}, \{p, \neg q, r\}\big\},$$
$$\mathsf{MS}^c\big(K_2^*\big) = \big\{\{p\}, \{\neg p, \neg q\}, \{\neg p, \neg p \vee q\}\big\}.$$

The second concept is as follows: informally, a hitting set (or hypergraph traversal) of a set of sets S, is a set containing at least one element from every set in S.

Definition 8. Let S be a set of sets from some finite domain D. A hitting set of S is a set $H \subseteq D$ s.t. $\forall S' \in S$, $H \cap S' \neq \emptyset$.

Let $\mathsf{HIT}(S)$ denote the set of hitting sets of S where $\forall H \in \mathsf{HIT}(S)$, $\nexists H' \in \mathsf{HIT}(S)$ s.t. $H' \subset H$. Therefore $\mathsf{HIT}(S)$ is the set of minimal hitting sets of S. We can demonstrate this with another example:

Example 5. Given a set of sets of integers $S = \{\{1, 2, 3\}, \{2, 4\}, \{3, 4\}\}$, then the hitting sets of S are

$$\{1, 2, 3, 4\}, \{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}, \{2, 3, 4\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}.$$

However, the set of all minimal hitting sets of S is

$$\mathsf{HIT}(S) = \big\{\{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}\big\}.$$

The most efficient algorithms for computing all MUSes are based on a relationship between the set of all CoMSSes and the set of all MUSes. This relationship, called *hitting set dualization*, was discovered separately in [6,10,41]. Specifically, the set of MUSes of a set of clauses K^* and the set of CoMSSes of K^* are *hitting set duals* of each other, i.e., the set of MUSes (resp. CoMSSes) of K^* is equal to the set of all minimal hitting sets of the set of CoMSSes (resp. MUSes) of K^*. This relationship is formalized as Theorem 4.5(c) and (d), along with a proof, in [6]. Consider the following example:

[2] CoMSS was the original name for what was later called a *minimal correction set* in [43], however we use the original terminology to avoid confusion with MCSes.

Example 6. Given a set of clauses $K_3^* = \{p, \neg p, p \vee \neg q, q, q \vee r\}$, then

$$\mathsf{MS}(K_3^*) = \big\{\{\neg p, q, q \vee r\}, \{\neg p, p \vee \neg q, q \vee r\}, \{p, p \vee \neg q, q, q \vee r\}\big\},$$
$$\mathsf{MS}^c(K_3^*) = \big\{\{p, p \vee \neg q\}, \{p, q\}, \{\neg p\}\big\},$$
$$\mathsf{MU}(K_3^*) = \big\{\{p, \neg p\}, \{\neg p, p \vee \neg q, q\}\big\}.$$

Clearly $\mathsf{MS}^c(K_3^*)$ and $\mathsf{MU}(K_3^*)$ are *hitting set duals*, i.e.,

$$\mathsf{HIT}\big(\mathsf{MS}^c(K_3^*)\big) = \mathsf{MU}(K_3^*),$$
$$\mathsf{HIT}\big(\mathsf{MU}(K_3^*)\big) = \mathsf{MS}^c(K_3^*).$$

The process for finding the MUSes of a set of clauses K^* therefore involves two parts: firstly, computing the set of all CoMSSes of K^* (related to computing the set of all MSSes); and secondly, computing the minimal hitting sets of the set of all CoMSSes of K^*. This second step finds the complete set of MUSes of K^*. However, work based on this hitting set dualization approach is primarily concerned with the first step, since there already exist efficient solutions for the hitting set problem. Of the various solutions proposed for computing CoMSSes, one of the best known algorithms is CAMUS [43]. This algorithm is based on an iterative MaxSAT search (a SAT optimization problem which is concerned with finding satisfiable subsets of clauses with maximum cardinality) and was shown to outperform earlier algorithms, e.g., those in [6,10]. While CAMUS remains near state-of-the-art, other algorithms have also been proposed which provide more efficient solutions in some cases. For example, HYCAM [29] adapts CAMUS by adding an inexpensive local search pretreatment while PICOMCS [55] offers some additional optimizations. Also, HYCAM itself was improved with some additional heuristics in [28]. Alternatively, FASTDIAG [24] offers some optimizations of the earlier QUICKXPLAIN [37] algorithm while NOPTSAT [14] incorporates preferences when finding MUSes. More recently, a family of algorithms was proposed in [54].

As an example we can consider the CAMUS approach to finding CoMSSes. Given that a MaxSAT search is concerned with finding a satisfiable subset with maximum cardinality, it is clear that MSSes are a generalization of this problem since every MaxSAT subset is also a MSS (although the converse does not hold). Clearly computing the set of all CoMSSes coincides with the computation of the set of all MSSes. So, in CAMUS, MSSes are found using an iterative MaxSAT search (utilizing an incremental SAT solver) which finds the largest satisfiable subset that has not been found in previous iterations. Then, finding all CoMSSes involves a standard SAT backtracking search where a y_i clause-selector variable is added to each clause ψ_i in order to enable or disable ψ_i (equivalent to setting y_i to true or false, respectively). A CoMSS is found when a minimal set of y_i variables are set to false but the set of clauses remains satisfiable. Then the CoMSS is equivalent to the set of ψ_i clauses relevant to this minimal set of y_i variables.

As for computing minimal hitting sets, we can consider the algorithm from [43] used in conjunction with CAMUS to find the set of MUSes. In this case, let K^* be a set of clauses and let $C = \mathsf{MS}^c(K^*)$ be the set of CoMSSes of K^*. The general idea of finding the set of MUSes from C, denoted $\mathsf{HIT}(C)$, is to begin with an empty set of clauses $\Psi = \emptyset$. Next, loop through each CoMSS $\Psi' \in C$ and add one clause $\psi \in \Psi'$ from the current CoMSS Ψ' to the forming MUS Ψ, i.e., $\Psi = \Psi \cup \{\psi\}$. Then, remove any remaining CoMSSes from C which contain this clause ψ, i.e., $C = C \setminus \{\Psi' : \Psi' \in C \mid \psi \in \Psi'\}$. Removing all other CoMSSes from C which contain the clause ψ forces ψ to be *essential* to the forming MUS Ψ, since removing ψ would leave at least one CoMSS (i.e., the current CoMSS) unrepresented in the final MUS. In other words, Ψ would not be a hitting set of C. The set of all MUSes from C can be found recursively by repeating this process until all CoMSSes have been removed from C, i.e., $C = \emptyset$.

Currently, hitting set dualization represents the most efficient approach to computing the complete set of MUSes. However, we must recall some of the related complexity issues. For example, the fundamental problems of SAT and UNSAT are NP-complete and coNP-complete, respectively. Also, computing one MSS or one MUS belongs to the second level of the polynomial hierarchy. Moreover, the number of MSSes (CoMSSes) and MUSes can be exponential on the number of clauses and thus enumerating them is intractable in the worst case. On the other hand, while computing one minimum (cardinality) hitting set is NP-hard, computing one minimal hitting set is a less strict requirement and can actually be generated in polynomial time [43,29]. Aside from these worst case complexity issues, some of the state-of-the-art algorithms have been shown to be viable for many practical cases.

As discussed previously, the two concepts of MUSes and MISes are compatible since an arbitrary knowledge base can be normalized as a set of clauses. If we convert an arbitrary knowledge base to a set of clauses we can then apply existing algorithms to find MUSes. However, if we wish to apply formula-level inconsistency measures we can only calculate clause-level values for this normalized knowledge rather than formula-level values for the original knowledge base. For this reason we must extend work on MUSes for arbitrary knowledge bases. Clearly this will also introduce an additional complexity issue since translating an arbitrary formulae to CNF while preserving equivalence has complexity $O(2^n)$ in the worst case. In the following sections, we present an existing MUS generalization method, as well as our new optimization MUS transformation method, which allow the complete set of MISes to be found in many practical cases.

4.2. Computing all MISes by MUS generalization

The MUS generalization approach involves grouping clauses in a normalized knowledge base w.r.t. formulae in the original knowledge base.

Definition 9. Let K be a knowledge base. A set of clause groups, denoted $[K^*]$, is a set of sets of clauses[3] defined as:

$$\left[K^*\right] = \left\{\phi^* \mid \phi \in K\right\}.$$

As with a set of clauses (or formulae), a set of clause groups is interpreted as the conjunction of its elements. In other words, given a set of clause groups $[K^*]$, then $\mathsf{mod}([K^*])$ denotes the set of models of a formula equivalent to the conjunction of all clauses in $[K^*]$. Given a knowledge base K, then recall that the non-optimized set of clauses K^* from K is defined as $K^* = \{\psi : \psi \in \phi^* \mid \phi \in K\}$ where $\forall \phi \in K$ and $\forall \psi \in \phi^*$, ψ is assumed to be unique. Clearly, $[K^*]$ represents a partition of K^* w.r.t. K. We can demonstrate this with an example:

Example 7. Given a knowledge base $K_4 = \{p, \neg p, \neg p \wedge q, q \vee r\}$, then $K_4^* = \{\psi_1, \psi_2, \ldots, \psi_5\}$ where $\psi_1 = p$, $\psi_2 = \neg p$, $\psi_3 = \neg p$, $\psi_4 = q$ and $\psi_5 = q \vee r$. So

$$\left[K_4^*\right] = \left\{\{\psi_1\}, \{\psi_2\}, \{\psi_3, \psi_4\}, \{\psi_5\}\right\}.$$

In order to describe the MUS generalization approach to finding MISes, we must generalize the definitions of MUSes, MSSes and CoMSSes w.r.t. a set of clause groups as follows:

Definition 10. Let $[K^*]$ be a set of clause groups. A group MUS (GMUS) $[\Psi]$ of $[K^*]$ is a set of clause groups s.t.

1. $[\Psi] \subseteq [K^*]$;
2. $\nexists \omega \in \mathsf{mod}([\Psi])$; and
3. $\forall [\Psi'] \subset [\Psi]$, $\exists \omega \in \mathsf{mod}([\Psi'])$.

Let $\mathsf{GMU}([K^*])$ denote the set of GMUSes of $[K^*]$.

Definition 11. Let $[K^*]$ be a set of clause groups. A group MSS (GMSS) $[\Psi]$ of $[K^*]$ is a set of clause groups s.t.

1. $[\Psi] \subseteq [K^*]$,
2. $\exists \omega \in \mathsf{mod}([\Psi])$,
3. $\forall [\Psi'] \subseteq [K^*]$, if $[\Psi] \subset [\Psi']$ then $\nexists \omega \in \mathsf{mod}([\Psi'])$.

Let $\mathsf{GMS}([K^*])$ denote the set of GMSSes of $[K^*]$.

Definition 12. Let $[K^*]$ be a set of clause groups and $\mathsf{GMS}([K^*])$ be the set of GMSSes of $[K^*]$. The set of CoGMSS of $[K^*]$ is a set of sets of clause groups, denoted $\mathsf{GMS}^{\mathsf{c}}([K^*])$, defined as:

$$\mathsf{GMS}^{\mathsf{c}}\left(\left[K^*\right]\right) = \left\{\left[K^*\right] \setminus [\Psi] \mid [\Psi] \in \mathsf{GMS}\left(\left[K^*\right]\right)\right\}.$$

Essentially, a GMUS is a minimal unsatisfiable set of clause groups and can be found in the same way as MUSes with the exception that clauses are treated as groups (cells in the partitioned set), rather than individually. In the CAMUS algorithm for example, CoGMSSes are found by adding the same y_i variable to each clause group g_i, i.e., $\forall \psi_j \in g_i$, y_i is added to ψ_j. This allows groups of clauses to be enabled or disabled together (setting y_i to true or false, respectively). Then the set of GMUSes is equal to the minimal hitting sets of the CoGMSSes of a set of clause groups. Converting each formula in a knowledge base to a set of clauses produces each clause group and so each MIS in the original knowledge base is directly equivalent to a GMUS in the set of clause groups. However this approach requires a non-overlapping partition of clauses, since each clause must be assigned one and only one y_i variable, which means that the set of clauses K^* from a knowledge base K cannot be optimized.

A simple method for finding MISes using GMUSes is shown in Algorithm 1. In detail: line 3 converts an arbitrary formula ϕ to a set of clauses and constructs a mapping Ψ_ϕ, from ϕ to this set of clauses; line 4 constructs the set of clause groups Ψ_K for the input knowledge base K; line 6 finds the set of GMUSes for Ψ_K; line 9 constructs the equivalent MIS Φ for each GMUS $\Psi \in \mathsf{GMU}(\Psi_K)$; line 10 constructs the set of MISes in K; and finally, line 11 returns the complete set $\mathsf{MI}(K)$.

[3] As with K^*, we consider all elements in $[K^*]$ to be unique even if they are syntactically equal. When this is ambiguous we will always distinguish between syntactically equal elements by assigning unique identifiers.

Algorithm 1: Finding MISes using MUS generalization.

Input: Knowledge base K
Output: $\mathsf{MI}(K)$

```
Ψ_K ← ∅;
for each φ ∈ K do
    Ψ_φ ← φ*;
    Ψ_K ← Ψ_K ∪ {Ψ_φ};
M ← ∅;
for each Ψ ∈ GMU(Ψ_K) do
    Φ ← ∅;
    for each Ψ_φ ∈ Ψ do
        Φ ← Φ ∪ {φ};
    M ← M ∪ {Φ};
return M;
```

4.3. Computing all MISes by optimized MUS transformation

As an alternative to the MUS generalization approach, we now propose a new optimized transformation approach to computing MISes. This serves to both formalize and improve the transformation approach mentioned in [43].

Definition 13. Let K be a knowledge base and K^* be the optimized set of clauses from K. A mapping $\sigma_K : K^* \mapsto 2^K$ is defined $\forall \psi \in K^*$ as:

$$\sigma_K(\psi) = \left\{\phi : \phi \in K \;\middle|\; \exists \psi' \in \phi^* \text{ where } \mathsf{Lit}(\psi) = \mathsf{Lit}\left(\psi'\right)\right\}.$$

This function accounts for clause optimization by mapping each clause in the optimized set of clauses to a set of formulae from the original knowledge base, such that the clause is equivalent to an element in the set of clauses for each formula in the set. With this mapping, we can determine the formula (or formulae) from which each clause in a MUS originated. So for a knowledge base K and a clause $\psi \in K^*$ s.t. K^* is optimized, $\sigma_K(\psi)$ is the set of formulae in K from which ψ originated.

Example 8. Given a knowledge base $K = \{p \wedge q, p \wedge r, \neg p\}$, then $K^* = \{\psi_1, \psi_2, \ldots, \psi_5\}$ where $\psi_1 = p$, $\psi_2 = q$, $\psi_3 = p$, $\psi_4 = r$ and $\psi_5 = \neg p$. If K^* is optimized then $K^* = \{\psi_1, \psi_2, \psi_4, \psi_5\}$. So, $\sigma_K(\psi_1) = \{p \wedge q, p \wedge r\}$.

Definition 14. Let K be a knowledge base, K^* be the optimized set of clauses from K, $\Psi \in \mathsf{MU}(K^*)$ be a MUS of K^* and σ_K be a mapping from K^* to 2^K. A FoMUS of Ψ w.r.t. σ_K, denoted $\mathsf{MU}^{\sigma_K}(\Psi)$, is a set of sets of formulae defined as:

$$\mathsf{MU}^{\sigma_K}(\Psi) = \left\{\sigma_K(\psi) \;\middle|\; \psi \in \Psi\right\}.$$

The FoMUS for a MUS Ψ contains all the formulae that have *provided* clauses to Ψ. This accounts for clause optimization with the assumption that any formula containing a clause which is equivalent to any clause in Ψ, can be considered to have *provided* a clause to Ψ.

Definition 15. Let K be a knowledge base, K^* be the optimized set of clauses from K and $\Psi \in \mathsf{MU}(K^*)$ be a MUS of K^*. Then the scope of Ψ w.r.t. K, denoted K_Ψ, is defined as:

$$K_\Psi = \bigcup \mathsf{MU}^{\sigma_K}(\Psi).$$

In other words, given a knowledge base K and its optimized set of clauses K^*, then the scope of a MUS $\Psi \in \mathsf{MU}(K^*)$ w.r.t. K is the set of formulae $K_\Psi \subseteq K$ where $\forall \phi \in K_\Psi$, $\exists \psi \in \Psi$ and $\exists \psi' \in \phi^*$ s.t. $\mathsf{Lit}(\psi) = \mathsf{Lit}(\psi')$. Simply put, the scope of Ψ w.r.t. K is the set of all formulae in the FoMUS $\mathsf{MU}^{\sigma_K}(\Psi)$, i.e., the set of formulae in K which are *touched* by Ψ, while accounting for clause optimization.

Definition 16. Let K be a knowledge base, K^* be the optimized set of clauses from K and $\Psi \in \mathsf{MU}(K^*)$ be a MUS of K^*. Then

$$\mathsf{HIT}\left(\mathsf{MU}^{\sigma_K}(\Psi)\right)$$

is called the set of pseudo-MISes of Ψ.

Lemma 1. *Let K be a knowledge base, K^* be the optimized set of clauses from K and $\Psi \in \mathsf{MU}(K^*)$ be a MUS of K^*. Then*

$$\mathsf{HIT}\big(\mathsf{MU}^{\sigma_K}(\Psi)\big) = \mathsf{MI}(K_\Psi),$$

if $\mathsf{MU}((K_\Psi)^*) = \{\Psi\}$.

Proof. Let K be a knowledge base and K^* be the optimized set of clauses from K. By definition, if K is inconsistent then there exists a MUS of K^*. Conversely, if there exists a MUS of K^* then K is inconsistent. Let $\Psi \in \mathsf{MU}(K^*)$ be a MUS of K^*. Recall that the FoMUS of Ψ w.r.t. K, denoted $\mathsf{MU}^{\sigma_K}(\Psi)$, is a set of sets of formulae from K. Specifically, for each clause $\psi \in \Psi$, there is a set of formulae $\Phi \in \mathsf{MU}^{\sigma_K}(\Psi)$ s.t. for each formula $\phi \in \Phi$, there exists a clause $\psi' \in \phi^*$ where $\mathsf{Lit}(\psi) = \mathsf{Lit}(\psi')$. Recall that a hitting set of $\mathsf{MU}^{\sigma_K}(\Psi)$ is a set of formulae containing at least one formula from each element of $\mathsf{MU}^{\sigma_K}(\Psi)$. Let Φ be a hitting set of $\mathsf{MU}^{\sigma_K}(\Psi)$. Then, for each clause $\psi \in \Psi$, by definition, there exists a formula $\phi \in \Phi$ s.t. there exists a clause $\psi' \in \phi^*$ where $\mathsf{Lit}(\psi) = \mathsf{Lit}(\psi')$, i.e., Φ is an inconsistent set of formulae since Ψ is inconsistent and for every clause $\psi \in \Psi$, there exists an equivalent clause $\psi' \in \Phi^*$. Thus, for every MUS $\Psi \in \mathsf{MU}(K^*)$, a hitting set of $\mathsf{MU}^{\sigma_K}(\Psi)$ is inconsistent since Ψ is inconsistent. Recall that the set of minimal hitting sets of $\mathsf{MU}^{\sigma_K}(\Psi)$ is defined as $\mathsf{HIT}(\mathsf{MU}^{\sigma_K}(\Psi))$. This is called the set of pseudo-MISes of Ψ. So, if Φ is a pseudo-MIS of Ψ then, by definition, Φ is inconsistent since Ψ is inconsistent and Φ is minimal w.r.t. the set of pseudo-MISes of Ψ. Finally, recall that the scope of a MUS Ψ w.r.t. K, denoted K_Ψ, is defined as the set of formulae in the FoMUS of Ψ w.r.t. K, i.e., $\bigcup \mathsf{MU}^{\sigma_K}(\Psi)$. Since the lemma is restricted to the case where $\mathsf{MU}((K_\Psi)^*) = \{\Psi\}$, then Ψ is the only MUS of $(K_\Psi)^*$. In other words, there cannot exist a pseudo-MIS of another MUS which is strictly included in a pseudo-MIS of Ψ. Thus, if $\mathsf{MU}((K_\Psi)^*) = \{\Psi\}$, then the set of pseudo-MISes of Ψ is the sound and complete set of MISes of K_Ψ, i.e., $\mathsf{HIT}(\mathsf{MU}^{\sigma_K}(\Psi)) = \mathsf{MI}(K_\Psi)$. □

Therefore, given a knowledge base K, a MIS w.r.t. the scope K_Ψ of a MUS $\Psi \in \mathsf{MU}(K^*)$ is simply a minimal hitting set of the FoMUS $\mathsf{MU}^{\sigma_K}(\Psi)$, as long as Ψ is the only MUS in the set of clauses for K_Ψ. So the set of pseudo-MISes of a MUS can be found using the same method used for finding a set of MUSes from a set of CoMSSes. The set of all pseudo-MISes of K is then defined as

$$\big\{\Phi : \Phi \in \mathsf{HIT}\big(\mathsf{MU}^{\sigma_K}(\Psi)\big) \;\big|\; \Psi \in \mathsf{MU}\big(K^*\big)\big\}.$$

The final issue, however, is that given a knowledge base K, although each pseudo-MIS of a MUS $\Psi \in \mathsf{MU}(K^*)$ is minimal w.r.t. the set of pseudo-MISes of Ψ, it may not be minimal w.r.t. a pseudo-MIS of another MUS $\Psi' \in \mathsf{MU}(K^*)$. So a pseudo-MIS of Ψ is locally minimal w.r.t. the set of pseudo-MISes of Ψ but may not be globally minimal w.r.t. the set of all pseudo-MISes in K. The algorithm presented in [50] for example, was not sound in that it found the set of all pseudo-MISes but this set contains potentially non-minimal sets.

Example 9. Given a knowledge base $K_5 = \{p \to (q \wedge r), p \wedge \neg q, \neg r\}$, then $K_5^* = \{\neg p \vee q, \neg p \vee r, p, \neg q, \neg r\}$. Clearly, K_5^* is optimized. So, $\mathsf{MU}(K_5^*) = \{\{p, \neg p \vee q, \neg q\}, \{p, \neg p \vee r, \neg r\}\}$. This results in the set of pseudo-MISes $\{\{p \to (q \wedge r), p \wedge \neg q\}, K_5\}$. However, given that $\{p \to (q \wedge r), p \wedge \neg q\} \subset K_5$, then $\mathsf{MI}(K_5) = \{\{p \to (q \wedge r), p \wedge \neg q\}\}$.

For this reason, a final post-check for set inclusion is required to remove non-minimal sets from the set of all pseudo-MISes of a knowledge base. As mentioned in Section 3, this issue was also encountered in [3] where it was found that using MUSes to compute arguments may produce an argument from one MUS which is strictly included in an argument from a different MUS. To address this, a post-check for minimality was required. Also, in [45], an equivalent post-check for set inclusion was required to remove result sets which are locally minimal but not globally minimal. A similar requirement was made in [44] to remove result sets which are locally maximal but not globally maximal. Essentially, optimized MUS transformation is a heuristic approach to finding MISes using MUSes as an indication of minimality and inconsistency and can be formalized as follows:

Theorem 1. *Let K be a knowledge base and K^* be the optimized set of clauses from K. Let P be the set of pseudo-MISes from K where*

$$P = \big\{\Phi : \Phi \in \mathsf{HIT}\big(\mathsf{MU}^{\sigma_K}(\Psi)\big) \;\big|\; \Psi \in \mathsf{MU}\big(K^*\big)\big\}.$$

Let M be the set of minimal sets in P where

$$M = \big\{\Phi : \Phi \in P \;\big|\; \nexists \Phi' \in P, \Phi' \subset \Phi\big\}.$$

Then

$$M = \mathsf{MI}(K).$$

Proof. Soundness: Let K be a knowledge base and let K^* be the optimized set of clauses from K. By definition, if K is inconsistent then there exists a MUS of K^*. Conversely, if there exists a MUS of K^* then K is inconsistent. Let $\Psi \in \mathsf{MU}(K^*)$

Algorithm 2: Finding MISes using optimized MUS transformation.

Input: Knowledge base K
Output: $\mathsf{MI}(K)$

1 $\Psi_K \leftarrow \emptyset$;
2 **for each** $\phi \in K$ **do**
3 **for each** $\psi \in \phi^*$ **do**
4 **if** $\exists \psi' \in \Psi_K$ **s.t.** $\mathsf{Lit}(\psi) = \mathsf{Lit}(\psi')$ **then**
5 $\sigma_{\psi'} \leftarrow \sigma_{\psi'} \cup \{\phi\}$;
6 **else**
7 $\Psi_K \leftarrow \Psi_K \cup \{\psi\}$;
8 $\sigma_{\psi} \leftarrow \{\phi\}$;
9 $P \leftarrow \emptyset$;
10 **for each** $\Psi \in \mathsf{MU}(\Psi_K)$ **do**
11 $\Phi \leftarrow \emptyset$;
12 **for each** $\psi \in \Psi$ **do**
13 $\Phi \leftarrow \Phi \cup \{\sigma_\psi\}$
14 $P \leftarrow P \cup \mathsf{HIT}(\Phi)$;
15 $M \leftarrow \mathsf{Minimal}(P)$;
16 **return** M;

be a MUS of K^*. Recall from Lemma 1 that the set of pseudo-MISes of Ψ is defined as $\mathsf{HIT}(\mathsf{MU}^{\sigma_K}(\Psi))$. Recall also that if a set of formulae Φ is a pseudo-MIS of Ψ, then Φ is inconsistent since Ψ is inconsistent and Φ is minimal w.r.t. the set of pseudo-MISes of Ψ.

(i) *Inconsistency*: Given that $P = \{\Phi : \Phi \in \mathsf{HIT}(\mathsf{MU}^{\sigma_K}(\Psi)) \mid \Psi \in \mathsf{MU}(K^*)\}$ is defined as the set of pseudo-MISes for all MUSes of K^*, then by definition, for every set of formulae $\Phi \in P$, Φ is inconsistent. Also, by definition, $M \subseteq P$. Thus, for every set of formulae $\Phi \in M$, then Φ is inconsistent.
(ii) *Minimality*: Given that $P = \{\Phi : \Phi \in \mathsf{HIT}(\mathsf{MU}^{\sigma_K}(\Psi)) \mid \Psi \in \mathsf{MU}(K^*)\}$ is defined as the set of pseudo-MISes for all MUSes of K^*, then by definition, for every MUS $\Psi \in \mathsf{MU}(K^*)$, every pseudo-MIS of Ψ is in P. Also, by definition, $M = \{\Phi : \Phi \in P \mid \nexists \Phi' \in P, \Phi' \subset \Phi\}$ is the set of minimal pseudo-MISes in P. Thus, for every set of formulae $\Phi \in M$, there does not exist another set of formulae $\Phi' \in M$ s.t. $\Phi' \subset \Phi$, i.e., Φ is minimal w.r.t. M.

Completeness: Let K be a knowledge base and K^* be the optimized set of clauses from K. By definition, if K is inconsistent then there exists a MUS of K^*. Conversely, if there exists a MUS of K^* then K is inconsistent. Let $\Phi \in \mathsf{MI}(K)$ be a MIS of K. Then, by definition, there exists a MUS of Φ^* (which is also a MUS of K^*) since Φ is inconsistent. However, for each set of formulae $\Phi' \subset \Phi$, there does not exist a MUS of Φ', since Φ' is consistent. Let $\Psi \in \mathsf{MU}(K^*)$ be a MUS of K^* s.t. $\Psi \in \mathsf{MU}(\Phi^*)$. Recall from Lemma 1 that the set of pseudo-MISes of Ψ is defined as $\mathsf{HIT}(\mathsf{MU}^{\sigma_K}(\Psi))$. Recall also that if a set of formulae Φ' is a pseudo-MIS of Ψ, then Φ' is inconsistent since Ψ is inconsistent and Φ' is minimal w.r.t. the set of pseudo-MISes of Ψ. Clearly, Φ is a pseudo-MIS of Ψ since, by definition, Φ is a MIS and Ψ is a MUS of Φ^* but, given any strict subset $\Phi' \subset \Phi$, then Ψ is not a MUS of $(\Phi')^*$. Also, given that $P = \{\Phi : \Phi \in \mathsf{HIT}(\mathsf{MU}^{\sigma_K}(\Psi)) \mid \Psi \in \mathsf{MU}(K^*)\}$ is defined as the set of pseudo-MISes for all MUSes of K^*, then by definition, $\Phi \in P$. Finally, by **inconsistency**, there does not exist a set of formulae $\Phi' \in P$ s.t. Φ' is consistent. Thus, given that $M = \{\Phi : \Phi \in P \mid \nexists \Phi' \in P, \Phi' \subset \Phi\}$ is defined as the minimal elements from P, then $\Phi \in M$ since for each strict subset $\Phi' \subset \Phi$, Φ' is consistent and so $\Phi' \notin P$. □

A suitable method to compute the MISes of a knowledge base using optimized MUS transformation is described in Algorithm 2. We assume that clauses are represented as sets of literals. In other words, equivalent clauses are equal sets of literals. Also, given a set of sets P, then the function $\mathsf{Minimal}(P)$ returns the minimal elements of P. In detail: line 3 converts each arbitrary formula $\phi \in K$ to a set of clauses ϕ^*; for each clause $\psi \in \phi^*$, line 4 checks if an equivalent clause has already been encountered; lines 5–8 then construct the optimized set of clauses Ψ_K and the σ_K mappings using either ψ or the existing clause ψ'; line 10 finds the set of MUSes of Ψ_K; line 13 constructs the FoMUS Φ for each MUS $\Psi \in \mathsf{MU}(\Psi_K)$ as a set of sets of formulae where each element is the σ_c mapping for each clause $\psi \in \Psi$; line 14 constructs the complete set of pseudo-MISes in K by finding the minimal hitting sets of each FoMUS Φ from the MUS Ψ, i.e., the minimal sets of formulae which (when converted to CNF) contain a set of clauses equivalent to Ψ; line 15 is a post-check for set inclusion to remove non-minimal sets from the set of pseudo-MISes in K; and finally, line 16 returns the complete set $\mathsf{MI}(K)$.

4.4. A practical example

We now demonstrate the process of computing the complete set of MISes in a knowledge base (using MUS generalization and optimized MUS transformation) with an example selected from the literature (the knowledge base Δ_1 from [31]).

Example 10 *(CNF).* Given a knowledge base $K_6 = \{\phi_1, \phi_2, \ldots, \phi_{14}\}$ where

$$
\begin{aligned}
\phi_1 &= red \rightarrow fast, \\
\phi_2 &= fast \rightarrow \neg fuelEfficient, \\
\phi_3 &= offRoad \rightarrow expensive, \\
\phi_4 &= sporty \rightarrow \big(expensive \\
&\qquad \wedge (black \vee red \vee white)\big), \\
\phi_5 &= \neg expensive \rightarrow under\ 20K, \\
\phi_6 &= cabriolet \rightarrow \neg bigCapacity, \\
\phi_7 &= fuelEfficient \rightarrow \neg offRoad,
\end{aligned}
\qquad
\begin{aligned}
\phi_8 &= red, \\
\phi_9 &= offRoad, \\
\phi_{10} &= \neg expensive, \\
\phi_{11} &= fuelEfficient, \\
\phi_{12} &= sporty, \\
\phi_{13} &= cabriolet, \\
\phi_{14} &= bigCapacity,
\end{aligned}
$$

then converting each $\phi \in K_6$ to CNF results in the following clauses:

$$
\begin{aligned}
\phi_1^* &= \{\ \psi_1 = \neg red \vee fast \ \}, \\
\phi_2^* &= \{\ \psi_2 = \neg fast \vee \neg fuelEfficient \ \}, \\
\phi_3^* &= \{\ \psi_3 = \neg offRoad \vee expensive \ \}, \\
\phi_4^* &= \left\{ \begin{array}{l} \psi_4 = \neg sporty \vee expensive, \\ \psi_5 = \neg sporty \vee black \vee red \vee white \end{array} \right\}, \\
\phi_5^* &= \{\ \psi_6 = expensive \vee under\ 20K \ \}, \\
\phi_6^* &= \{\ \psi_7 = \neg cabriolet \vee \neg bigCapacity \ \}, \\
\phi_7^* &= \{\ \psi_8 = \neg fuelEfficient \vee \neg offRoad \ \}, \\
\phi_8^* &= \{\ \psi_9 = red \ \}, \\
\phi_9^* &= \{\ \psi_{10} = offRoad \ \} \\
\phi_{10}^* &= \{\ \psi_{11} = \neg expensive \ \}, \\
\phi_{11}^* &= \{\ \psi_{12} = fuelEfficient \ \}, \\
\phi_{12}^* &= \{\ \psi_{13} = sporty \ \}, \\
\phi_{13}^* &= \{\ \psi_{14} = cabriolet \ \}, \\
\phi_{14}^* &= \{\ \psi_{15} = bigCapacity \ \}.
\end{aligned}
$$

Therefore, K_6^* is the set of clauses from K_6 where $K_6^* = \{\psi_1, \psi_2, \ldots, \psi_{15}\}$. Since there are no equivalent clauses in K_6^*, we can say that K_6^* is optimized. Finally, the set of clause groups $[K_6^*]$ from K_6 is:

$$
[K_6^*] = \left\{ \begin{array}{l} \{\psi_1\}, \{\psi_2\}, \{\psi_3\}, \{\psi_4, \psi_5\}, \{\psi_6\}, \{\psi_7\}, \{\psi_8\}, \\ \{\psi_9\}, \{\psi_{10}\}, \{\psi_{11}\}, \{\psi_{12}\}, \{\psi_{13}\}, \{\psi_{14}\}, \{\psi_{15}\} \end{array} \right\}.
$$

We can now demonstrate finding the MISes in K_6 using MUS generalization.

Example 11 *(MUS generalization).* Given the knowledge base K_6 and the set of clause groups $[K_6^*]$ from Example 10, then the set of GMSSes from $[K_6^*]$ is $\mathsf{GMS}([K_6^*]) = \{[\Psi_1], [\Psi_2], \ldots, [\Psi_{69}]\}$ where

$$
\begin{aligned}
[\Psi_1] &= \{\ \{\psi_1\}, \{\psi_2\}, \{\psi_3\}, \{\psi_4, \psi_5\}, \{\psi_6\}, \{\psi_8\}, \{\psi_9\}, \{\psi_{10}\}, \{\psi_{13}\}, \{\psi_{14}\}, \{\psi_{15}\} \ \}, \\
[\Psi_2] &= \{\ \{\psi_1\}, \{\psi_2\}, \{\psi_3\}, \{\psi_4, \psi_5\}, \{\psi_6\}, \{\psi_7\}, \{\psi_8\}, \{\psi_9\}, \{\psi_{10}\}, \{\psi_{13}\}, \{\psi_{14}\} \ \}, \\
[\Psi_3] &= \{\ \{\psi_1\}, \{\psi_2\}, \{\psi_3\}, \{\psi_4, \psi_5\}, \{\psi_6\}, \{\psi_7\}, \{\psi_8\}, \{\psi_9\}, \{\psi_{10}\}, \{\psi_{13}\}, \{\psi_{15}\} \ \}, \\
[\Psi_4] &= \{\ \{\psi_1\}, \{\psi_3\}, \{\psi_5\}, \{\psi_6\}, \{\psi_8\}, \{\psi_9\}, \{\psi_{11}\}, \{\psi_{12}\}, \{\psi_{13}\}, \{\psi_{14}\}, \{\psi_{15}\} \ \}, \\
[\Psi_5] &= \{\ \{\psi_1\}, \{\psi_3\}, \{\psi_4, \psi_5\}, \{\psi_6\}, \{\psi_7\}, \{\psi_8\}, \{\psi_9\}, \{\psi_{12}\}, \{\psi_{13}\}, \{\psi_{15}\} \ \}, \\
[\Psi_6] &= \{\ \{\psi_2\}, \{\psi_3\}, \{\psi_4, \psi_5\}, \{\psi_6\}, \{\psi_7\}, \{\psi_9\}, \{\psi_{10}\}, \{\psi_{12}\}, \{\psi_{13}\}, \{\psi_{15}\} \ \}, \\
[\Psi_7] &= \{\ \{\psi_1\}, \{\psi_2\}, \{\psi_3\}, \{\psi_4, \psi_5\}, \{\psi_6\}, \{\psi_7\}, \{\psi_8\}, \{\psi_{12}\}, \{\psi_{13}\}, \{\psi_{15}\} \ \}, \\
[\Psi_8] &= \{\ \{\psi_1\}, \{\psi_3\}, \{\psi_4, \psi_5\}, \{\psi_6\}, \{\psi_7\}, \{\psi_9\}, \{\psi_{10}\}, \{\psi_{12}\}, \{\psi_{13}\}, \{\psi_{15}\} \ \}, \\
[\Psi_9] &= \{\ \{\psi_2\}, \{\psi_3\}, \{\psi_4, \psi_5\}, \{\psi_6\}, \{\psi_7\}, \{\psi_8\}, \{\psi_9\}, \{\psi_{12}\}, \{\psi_{13}\}, \{\psi_{15}\} \ \}, \\
&\cdots \\
[\Psi_{69}] &= \{\ \{\psi_1\}, \{\psi_2\}, \{\psi_3\}, \{\psi_4, \psi_5\}, \{\psi_6\}, \{\psi_7\}, \{\psi_{10}\}, \{\psi_{12}\}, \{\psi_{13}\}, \{\psi_{15}\} \ \}.
\end{aligned}
$$

The following CoGMSSes can be found from the set of GMSSes $\mathsf{GMS}([K_6^*])$:

$[\Psi_i]$	$\mathsf{GMS}^c([K_6^*])$
$[\Psi_1]$	$\{\{\psi_7\},\{\psi_{11}\},\{\underline{\psi_{12}}\}\}$
$[\Psi_2]$	$\{\{\psi_{11}\},\{\underline{\psi_{12}}\},\{\psi_{15}\}\}$
$[\Psi_3]$	$\{\{\psi_{11}\},\{\underline{\psi_{12}}\},\{\psi_{14}\}\}$
$[\Psi_4]$	$\{\{\underline{\psi_2}\},\{\psi_4,\psi_5\},\{\psi_7\},\{\psi_{10}\}\}$
$[\Psi_5]$	$\{\{\underline{\psi_2}\},\{\psi_{10}\},\{\psi_{11}\},\{\psi_{14}\}\}$
$[\Psi_6]$	$\{\{\underline{\psi_1}\},\{\psi_8\},\{\psi_{11}\},\{\psi_{14}\}\}$
$[\Psi_7]$	$\{\{\underline{\psi_9}\},\{\psi_{10}\},\{\psi_{11}\},\{\psi_{14}\}\}$
$[\Psi_8]$	$\{\{\underline{\psi_2}\},\{\psi_8\},\{\psi_{11}\},\{\psi_{14}\}\}$
$[\Psi_9]$	$\{\{\underline{\psi_1}\},\{\psi_{10}\},\{\psi_{11}\},\{\psi_{14}\}\}$
	...
$[\Psi_{69}]$	$\{\{\psi_8\},\{\underline{\psi_9}\},\{\psi_{11}\},\{\psi_{14}\}\}$

The set of minimal hitting sets from the set of CoGMSSes $\mathsf{GMS}^c([K_6^*])$ is $\mathsf{HIT}(\mathsf{GMS}^c([K_6^*])) = \{[\Psi_1'],[\Psi_2'],\ldots,[\Psi_5']\}$ where

$$[\Psi_1'] = \{\{\psi_1\},\{\psi_2\},\{\psi_9\},\{\underline{\psi_{12}}\}\},\quad [\Psi_2'] = \{\{\psi_8\},\{\psi_{10}\},\{\psi_{12}\}\},\quad [\Psi_3'] = \{\{\psi_3\},\{\psi_{10}\},\{\psi_{11}\}\},$$
$$[\Psi_4'] = \{\{\psi_4,\psi_5\},\{\psi_{11}\},\{\psi_{13}\}\},\quad [\Psi_5'] = \{\{\psi_7\},\{\psi_{14}\},\{\psi_{15}\}\}.$$

So, with a simple mapping to formulae in K_6, the MISes in K_6 can be found from the set $\mathsf{HIT}(\mathsf{GMS}^c([K_6^*]))$, i.e., $[\Psi_1'] \to \Phi_1, [\Psi_2'] \to \Phi_2, \ldots, [\Psi_5'] \to \Phi_5$ where

$$\Phi_1 = \{\phi_1,\phi_2,\phi_8,\phi_{11}\},\quad \Phi_2 = \{\phi_7,\phi_9,\phi_{11}\},\quad \Phi_3 = \{\phi_3,\phi_9,\phi_{10}\},$$
$$\Phi_4 = \{\phi_4,\phi_{10},\phi_{12}\},\quad \Phi_5 = \{\phi_6,\phi_{13},\phi_{14}\}.$$

Finally:

$$\mathsf{MI}(K_6) = \{\Phi_1,\Phi_2,\ldots,\Phi_5\}.$$

Next we demonstrate finding MISes by optimized MUS transformation.

Example 12 *(Optimized MUS transformation).* Given the knowledge base K_6 and the optimized set of clauses K_6^* from Example 10, then the set of MSSes from K_6 is $\mathsf{MS}(K_6^*) = \{\Psi_1,\Psi_2,\ldots,\Psi_{69}\}$ where

$$\begin{aligned}
\Psi_1 &= \{\psi_1,\psi_2,\psi_3,\psi_4,\psi_5,\psi_6,\psi_8,\psi_9,\psi_{10},\psi_{13},\psi_{14},\psi_{15}\},\\
\Psi_2 &= \{\psi_1,\psi_2,\psi_3,\psi_4,\psi_5,\psi_6,\psi_7,\psi_8,\psi_9,\psi_{10},\psi_{13},\psi_{14}\},\\
\Psi_3 &= \{\psi_1,\psi_2,\psi_3,\psi_4,\psi_5,\psi_6,\psi_7,\psi_8,\psi_9,\psi_{10},\psi_{13},\psi_{15}\},\\
\Psi_4 &= \{\psi_1,\psi_3,\psi_5,\psi_6,\psi_8,\psi_9,\psi_{11},\psi_{12},\psi_{13},\psi_{14},\psi_{15}\},\\
\Psi_5 &= \{\psi_1,\psi_3,\psi_4,\psi_5,\psi_6,\psi_7,\psi_8,\psi_9,\psi_{12},\psi_{13},\psi_{15}\},\\
\Psi_6 &= \{\psi_2,\psi_3,\psi_4,\psi_5,\psi_6,\psi_7,\psi_9,\psi_{10},\psi_{12},\psi_{13},\psi_{15}\},\\
\Psi_7 &= \{\psi_1,\psi_2,\psi_3,\psi_4,\psi_5,\psi_6,\psi_7,\psi_8,\psi_{12},\psi_{13},\psi_{15}\},\\
\Psi_8 &= \{\psi_1,\psi_3,\psi_4,\psi_5,\psi_6,\psi_7,\psi_9,\psi_{10},\psi_{12},\psi_{13},\psi_{15}\},\\
\Psi_9 &= \{\psi_2,\psi_3,\psi_4,\psi_5,\psi_6,\psi_7,\psi_8,\psi_9,\psi_{12},\psi_{13},\psi_{15}\},\\
&\ldots\\
\Psi_{69} &= \{\psi_1,\psi_2,\psi_3,\psi_4,\psi_5,\psi_6,\psi_7,\psi_{10},\psi_{12},\psi_{13},\psi_{15}\}.
\end{aligned}$$

The following CoMSSes can be found from the set of MSSes $\mathsf{MS}(K_6^*)$:

Ψ_i	$\mathsf{MS}^{\mathsf{c}}(K_6^*)$
Ψ_1	$\{ \quad \psi_7, \psi_{11}, \underline{\psi_{12}} \quad \}$
Ψ_2	$\{ \quad \psi_{11}, \underline{\psi_{12}}, \psi_{15} \quad \}$
Ψ_3	$\{ \quad \psi_{11}, \underline{\psi_{12}}, \psi_{14} \quad \}$
Ψ_4	$\{ \quad \underline{\psi_2}, \psi_4, \psi_7, \psi_{10} \quad \}$
Ψ_5	$\{ \quad \underline{\psi_2}, \psi_{10}, \psi_{11}, \psi_{14} \quad \}$
Ψ_6	$\{ \quad \underline{\psi_1}, \psi_8, \psi_{11}, \psi_{14} \quad \}$
Ψ_7	$\{ \quad \underline{\psi_9}, \psi_{10}, \psi_{11}, \psi_{14} \quad \}$
Ψ_8	$\{ \quad \underline{\psi_2}, \psi_8, \psi_{11}, \psi_{14} \quad \}$
Ψ_9	$\{ \quad \underline{\psi_1}, \psi_{10}, \psi_{11}, \psi_{14} \quad \}$
	$\cdots$
Ψ_{69}	$\{ \quad \psi_8, \underline{\psi_9}, \psi_{11}, \psi_{14} \quad \}$

The set of minimal hitting sets of the set of CoMSSes $\mathsf{MS}^{\mathsf{c}}(K_6^*)$ is $\mathsf{HIT}(\mathsf{MS}^{\mathsf{c}}(K_6^*)) = \{\Psi_1', \Psi_2', \ldots, \Psi_5'\}$ where

$$\Psi_1' = \{ \ \underline{\psi_1, \psi_2, \psi_9, \psi_{12}} \ \}, \quad \Psi_4' = \{ \ \psi_4, \psi_{11}, \psi_{13} \ \},$$
$$\Psi_2' = \{ \ \psi_8, \psi_{10}, \psi_{12} \ \}, \quad \Psi_5' = \{ \ \psi_7, \psi_{14}, \psi_{15} \ \}.$$
$$\Psi_3' = \{ \ \psi_3, \psi_{10}, \psi_{11} \ \},$$

In other words $\mathsf{MU}(K_6^*) = \{\Psi_1', \Psi_2', \ldots, \Psi_5'\}$. To transform the MUSes of K_6^* to the MISes of K_6, we must first determine the formulae from which each clause in each MUS originated (FoMUSes), i.e.:

Ψ_i'	$\mathsf{MU}^{\sigma_K}(\Psi_i')$
Ψ_1'	$\{ \{\phi_1\}, \{\phi_2\}, \{\phi_8\}, \{\phi_{11}\} \}$
Ψ_2'	$\{ \{\phi_7\}, \{\phi_9\}, \{\phi_{11}\} \}$
Ψ_3'	$\{ \{\phi_3\}, \{\phi_9\}, \{\phi_{10}\} \}$
Ψ_4'	$\{ \{\phi_4\}, \{\phi_{10}\}, \{\phi_{12}\} \}$
Ψ_5'	$\{ \{\phi_6\}, \{\phi_{13}\}, \{\phi_{14}\} \}$

Then to find the pseudo-MISes from each FoMUS, we find the minimal hitting sets from each FoMUS. This is straightforward for K_6 since no clause originates from more than one formula, i.e.:

$$\mathsf{HIT}(\mathsf{MU}^{\sigma_K}(\Psi_1')) = \{ \ \Phi_1 \ \}, \quad \mathsf{HIT}(\mathsf{MU}^{\sigma_K}(\Psi_4')) = \{ \ \Phi_4 \ \},$$
$$\mathsf{HIT}(\mathsf{MU}^{\sigma_K}(\Psi_2')) = \{ \ \Phi_2 \ \}, \quad \mathsf{HIT}(\mathsf{MU}^{\sigma_K}(\Psi_5')) = \{ \ \Phi_5 \ \}.$$
$$\mathsf{HIT}(\mathsf{MU}^{\sigma_K}(\Psi_3')) = \{ \ \Phi_3 \ \}$$

where

$$\Phi_1 = \{ \ \phi_1, \phi_2, \phi_8, \phi_{11} \ \}, \quad \Phi_4 = \{ \ \phi_4, \phi_{10}, \phi_{12} \ \},$$
$$\Phi_2 = \{ \ \phi_7, \phi_9, \phi_{11} \ \}, \quad \Phi_5 = \{ \ \phi_6, \phi_{13}, \phi_{14} \ \}.$$
$$\Phi_3 = \{ \ \phi_3, \phi_9, \phi_{10} \ \},$$

So, the pseudo-MISes in K_6 are $\Phi_1, \Phi_2, \ldots, \Phi_5$. Finally, since each pseudo-MIS is minimal, this is exactly the set of MISes in K_6, i.e.:

$$\mathsf{MI}(K_6) = \{\Phi_1, \Phi_2, \ldots, \Phi_5\}.$$

5. Evaluation for computing MISes

The algorithms for computing the MISes of a knowledge base (using the existing MUS generalization and our new optimized MUS transformation approaches) were implemented in a tool called MIMUS (**MI**Ses from **MUS**es). Some selected inconsistency measures were also implemented and evaluated and this will be detailed in Section 6. Many real-world knowledge bases (e.g., the previously mentioned QRadar and Snort rule sets) represent interesting problems for evaluating MIMUS. However, the process of formalizing these systems into propositional or first-order logic is rarely a trivial task. Since our motivation is to analyze inconsistency in these systems, it is essential that this process results in an accurate representation of the original knowledge base and does not introduce inconsistencies which were not originally present. For this reason, the formalization process generally requires some form of domain expertise. Moreover, it is unlikely that any single real-world knowledge base would provide a sufficiently large and varied dataset to thoroughly evaluate MIMUS. For example, if we wish to consider the effect of the number of variables in the language, the complexity of formulae, or

the number of MISes, then a range of inconsistent knowledge bases would be required. Finally, it is often the case that these real-world knowledge bases are highly optimized through a manual process involving some informal inconsistency checking. This is certainly true of the default QRadar rule set where we believe the primary interest of MIMUS is to validate user-customized (private and unoptimized) rule sets. Note that a case study of formalizing the QRadar rule set (including some examples of the types of inconsistencies which may occur in user-customized rule sets) can be found in [50]. So, while we hope to evaluate MIMUS with real-world knowledge bases in the future, in this work we opt for generating synthetic sets of knowledge bases which will allow us to evaluate various practical properties of our algorithms.

To fulfill this purpose, we obviously require knowledge bases which are: (i) non-trivial; (ii) constrained enough so that the probability of generating MISes is sufficiently high; and (iii) sufficient to demonstrate the key properties of the algorithms. In the SAT community, the complexity of creating efficient SAT solvers has resulted in the development of a range of SAT benchmarks. Many of these benchmarks have been used in the literature for evaluating MUS implementations. For example, the DC benchmark set from an automotive product configuration domain [65] was used in [43,69], while pigeonhole and xor-chain benchmarks[4] were used in [43,29]. More generally, there has been extensive work on generating random SAT benchmarks for evaluating SAT solvers [21]. Whether random or not, these benchmarks are usually formatted as sets of clauses which means they have little experimental value for the problem of finding MISes. Some of these benchmarks were actually used in [50] as a test set to evaluate a prototype implementation of MIMUS whereby each clause was considered an arbitrary formula. However, as stated in Section 3, when every formula in a knowledge base is a clause, MISes and MUSes are equivalent. For this reason, the early evaluation of MIMUS presented in [50] focused solely on the performance of the implemented inconsistency measures (since the complexity of formulae is irrelevant for most formula-level measures).

Fortunately there has been some work on generating random non-CNF SAT benchmarks in [57,52], which provide a means to generate the type of arbitrary knowledge base for which MIMUS was intended. In particular, this work is based on generating formulae with a *fixed-shape* model where all formulae have an equal number of conjunctions, disjunctions and literals. While this restricts us to knowledge bases where formulae have uniform complexity, this is sufficient to evaluate the performance of MIMUS since these knowledge bases will still exhibit varying amounts of MISes. In fact, the *fixed-shape* model is ideal for this evaluation since it allows greater control over the types of inconsistent knowledge bases which will be generated. In other words, it supports a more principled evaluation of MIMUS in terms of analyzing the impact of certain factors (e.g., formula complexity, the number of MISes, etc.) on the efficiency of MIMUS. Also, this method can be easily extended to generate a random weight for each random formula and thereby generate random stratified knowledge bases. So, in this section, we first describe some general properties of random SAT benchmarks before giving an overview of how random arbitrary knowledge bases can be generated. Finally, we present a thorough evaluation of MIMUS using these random knowledge bases.

5.1. Generating random unsatisfiable SAT instances

In the SAT community, k-SAT is the problem of determining the satisfiability of a SAT instance (a set of clauses) with at most k literals per clause (k-CNF). It is well known that the *clause : variable ratio* (clause density) is the most important aspect in terms of the probability that a given *random* k-SAT instance will be satisfiable. When the ratio is low (i.e., little repetition of variables) random k-SAT instances are more likely to be satisfiable, but as the ratio increases (i.e., more repetition of variables) random k-SAT instances are more likely to be unsatisfiable [21]. This is intuitive since a low ratio means the k-SAT instance is under-constrained and a high ratio means the k-SAT instance is over-constrained.

The hardness of determining satisfiability follows the easy–hard–easy (or more specifically, easy–hard–moderately hard) pattern based on this ratio, where the most difficult problems are in the sharp phase transition (satisfiability threshold) between problems being satisfiable with a high probability and problems being unsatisfiable with a high probability [57]. This sharp phase transition is where the k-SAT instance is just constrained enough to be potentially unsatisfiable [52]. Furthermore, for random k-SAT instances the phase transition is dependent on k. For example, in random 2-SAT instances (which can actually be solved in polynomial time) the threshold is known to be where the *clause : variable ratio* is around 1, while for random 3-SAT instances the best lower bound is 3.52 and the best upper bound is 4.506 [12]. It is known that a threshold also exists when $k > 3$ but there are no accepted interval values [23].

For this evaluation, our main requirement is to generate random knowledge bases which are inconsistent, i.e., exhibit MISes. In order to thoroughly evaluate MIMUS however, an important requirement will be the ability to generate (with some degree of accuracy) knowledge bases exhibiting an increasing number of MISes. It is clear that in terms of generating random unsatisfiable k-SAT instances, the *clause : variable ratio* is the most important factor. This feature suggests that, given some parameter values for a random generator, it is possible to reliably predict whether a randomly generated knowledge base will be inconsistent. By implication, this suggests that increasing the probability that a randomly generated knowledge base will be inconsistent, is also likely to increase the number of MISes in the generated knowledge base (since the knowledge base will become more constrained).

[4] SATLIB. Benchmark Problems. http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html.

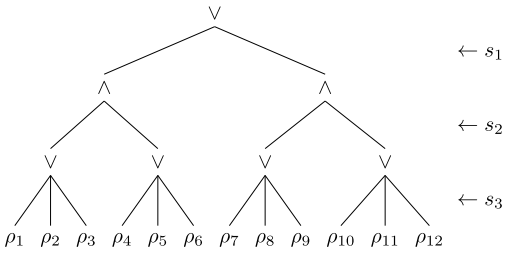


Fig. 1. Balanced tree representing the structure of an arbitrary $\langle 2, 2, 3\rangle$-formula. Given the parameter v (denoting the number of variables), then to generate a random formula, each literal ρ_i in the balanced tree is randomly generated from the set of variables $\mathcal{P} = \{p_1, \ldots, p_v\}$.

5.2. Generating random arbitrary formulae

As mentioned previously, there are some existing random SAT generators which can generate non-CNF formulae. A suitable example is GENBAL from [57] which can generate a single arbitrary formula as a conjunction of fixed-shape arbitrary subformulae in negation normal form (NNF). So, if we treat the formula as a knowledge base and its immediate subformulae as formulae in the knowledge base, GENBAL can generate a knowledge base with a set of random fixed-shape formulae. The limitation is that we are restricted to knowledge bases where all formulae are of equal complexity (in terms sharing an equal number of conjunctions, disjunctions and literals).

Three parameters are required by GENBAL in order to generate a random knowledge base. These are:

- v the number of variables;
- r the *formula : variable ratio*; and
- s the shape of the arbitrary formulae.

The value of v is an integer representing the finite number of variables in the language $\mathcal{P} = \{p_1, \ldots, p_v\}$. However, rather than defining the number of formulae to be generated, GENBAL requires a real number r representing the target *formula : variable ratio*. Note that this value does not necessarily represent the actual *formula : variable ratio* of the generated knowledge base since not all variables in the language may be randomly selected.[5] The remaining parameter, s, is denoted by a collection of integers $\langle s_1, \ldots, s_n\rangle$ representing a fixed-shape balanced tree with n levels, where s_1 is the highest level and s_n is the lowest. We call s_i the i-th level of s. The balanced tree is defined in terms of alternating disjunctions and conjunctions between levels, starting with a disjunction at s_1, and represents the shape of the generated formulae. A formula with a shape s is called an s-formula. Fig. 1 shows a balanced tree representing the structure of a $\langle 2, 2, 3\rangle$-formula.

Example 13. Given a randomly generated knowledge base K_7 from parameters $v = 5$, $r = 1$ and $s = \langle 2\rangle$ where

$$K_7 = \{\neg p_1 \vee \neg p_4, \neg p_4 \vee p_2, \neg p_1 \vee \neg p_3, \neg p_3 \vee \neg p_5, p_5 \vee p_4\},$$

then

$$K_7^* = K_7.$$

If s is restricted to a single integer, then the resulting formulae will always be in CNF since the shape has only one level s_1 (where the first level s_1 is always a disjunction). In Example 13, a knowledge base with 5 formulae ($v \times r$) is generated where all formulae are in 2-CNF. In fact, a set of random $\langle k\rangle$-formulae will always be in k-CNF. This means that the *formula : variable ratio* for a knowledge base K, where $s = \langle k\rangle$, will be equal to the *clause : variable ratio* for a random k-SAT instance. For example, a set of random $\langle 2\rangle$-formulae, where $r > 1$, will be inconsistent with a high probability since the phase transition for random 2-SAT instances is where the *clause : variable ratio* is around 1. Obviously MUSes and MISes are equivalent for $\langle k\rangle$-formulae though.

Example 14. Given a randomly generated knowledge base K_8 from parameters $v = 5$, $r = 0.2$ and $s = \langle 3, 2\rangle$ where

$$K_8 = \{(\neg p_3 \wedge p_1) \vee (p_2 \wedge p_4) \vee (p_5 \wedge \neg p_4)\},$$

then

[5] Random CNF generators take the same approach where r is instead the *clause : variable ratio*, i.e., in the literature, suggested phase transitions for random CNF k-SAT instances are based on the parameter values rather than the final *clause : variable ratio* of generated instances.

$$K_8^* = \left\{ \begin{array}{llll} \neg p_3 \vee \neg p_4 \vee p_2, & \neg p_3 \vee \neg p_4 \vee p_4, \\ \neg p_3 \vee p_2 \vee p_5, & \neg p_3 \vee p_4 \vee p_5, \\ \neg p_4 \vee p_1 \vee p_2, & \neg p_4 \vee p_1 \vee p_4, \\ p_1 \vee p_2 \vee p_5, & p_1 \vee p_4 \vee p_5 \end{array} \right\}.$$

In Example 14, importantly, MUSes and MISes are no longer equivalent since the (single) formula in K_8 is not a clause. So, for a random knowledge base K where $s = \langle 3, 2\rangle$, conversion of $\phi \in K$ to CNF results in 8 clauses with 3 literals each (3-CNF). In other words, given r and a $\langle 3, 2\rangle$ shaped arbitrary knowledge base K, then a *clause : variable ratio* for K^* could be determined by $8r$. So in this example, the *clause : variable ratio* for K_8 w.r.t. r would be 1.6. However, since K_8 is not in CNF originally, a phase transition for K_8 cannot be directly inferred from existing work on k-SAT phase transitions. In fact, results in [57] demonstrate that the probability of satisfiability in a fixed-shape knowledge base K does not reliably match that of K^*. In other words, given that $\langle 3, 2\rangle$-formulae will be in 3-CNF, we cannot simply say that for a random knowledge base K where $s = \langle 3, 2\rangle$, K^* will have a phase transition equal to that of a 3-SAT instance. Fortunately, experiments in [57] have shown that random arbitrary fixed-shape knowledge bases do exhibit sharp phase transitions of their own w.r.t. the *formula : variable ratio* r. This means that, as with standard random k-SAT instances, it is possible to predict the parameter values needed to generate random arbitrary knowledge bases in the phase transition region (where inconsistent knowledge bases may exhibit the fewest MISes) for a given value of s.

Importantly, work in [57] also attempted to find if these random fixed-shape SAT instances also exhibit the easy-hard-easy pattern (around the phase transition region) which is present in random k-SAT instances. This was achieved by: firstly, applying a standard (equivalence-preserving) and an optimized (equisatisfiable) CNF translation of randomly generated non-CNF fixed-shape formulae; and secondly, measuring the number of branches explored by a variety of popular DPLL-based CNF SAT solvers (which generally indicates the hardness of a SAT problem) when solving these translated CNF formulae. It was found that the fixed-shape model for random non-CNF formulae does, in fact, exhibit the same easy–hard–easy pattern around the phase transition of a particular shape.

In summary, given that equivalence-preserving CNF translation results in a potentially exponential increase in the size of the problem, it is clear that the value of s for a random knowledge base K has a dramatic effect on the size of K^*. Also, standard random k-SAT problems exhibit an easy–hard–easy pattern where the hardest problems are in the sharp *clause : variable ratio* region (phase transition) where the probability of generating an unsatisfiable SAT instance goes from almost 0 to almost 1. An equivalent sharp phase transition phenomenon exists in random fixed-shape arbitrary knowledge bases in relation to the *formula : variable ratio*. These random fixed-shape knowledge bases also exhibit an equivalent easy–hard–easy pattern around the phase transition. However, we suggest that the number of MISes in these random fixed-shape arbitrary knowledge bases will increase as the *formula : variable ratio* increases through the phase transition region. Moreover, we suggest that the hardness of computing the complete set of MISes will increase as the *formula : variable ratio* increases. In other words, while using the comparatively efficient and succinct encodings of random fixed-shape knowledge bases, we can theoretically predict the optimal parameter values for GENBAL in order to generate random knowledge bases which exhibit a range of MISes. This will allow a structured evaluation of MIMUS.

5.3. Experiments

The implementation of MIMUS[6] was written in C++ and compiled for x64 with g++. A CNF converter was implemented for parsing an infix propositional expression, building a binary expression tree using the traditional Shunting-yard algorithm and applying the standard equivalence-preserving CNF translation. Internally, the CAMUS 1.0.5 [43] implementation of the CAMUS algorithm was used for computing CoMSSes, CoGMSSes and minimal hitting sets. This version of CAMUS uses the popular and highly optimized MINISAT 1.12b [19] SAT solver for incremental SAT checking. For the optimized MUS transformation algorithm, equivalence between clauses was determined by a simple comparison of ordered sets. Also, a naive set inclusion function was implemented for the post-check to remove non-minimal pseudo-MISes. The test version was MIMUS 1.0.4. All experiments were carried out on a 2.67 GHz Intel Xeon 12 core HPC server with 94 GB of RAM running Red Hat Enterprise GNU/Linux Server 5.7. For each experiment, 10 instances of MIMUS were run concurrently with each instance using a single core. In other words, the experiments were run efficiently but CPU performance for MIMUS was comparable to a standard desktop computer.

5.3.1. Generating suitable knowledge bases exhibiting MISes

Firstly, we define a sample (of knowledge bases) as a set of fixed-shape knowledge bases randomly generated with GENBAL. Unless otherwise stated, we assume all random knowledge bases in a sample are generated from the same parameters values for: the number of variables v; the *formula : variable ratio* r; and the fixed-shape s.

In [57], random s-formulae knowledge bases were shown to exhibit phase transitions w.r.t. their *formula : variable ratio* (analogous to the *clause : variable ratio* in random k-SAT instances). Fig. 2 shows the results of an experiment on the probability of satisfiability for different random s-formulae knowledge bases in relation to r. So for each

[6] MIMUS is available from http://www.cs.qub.ac.uk/~kmcareavey01/mimus.html.

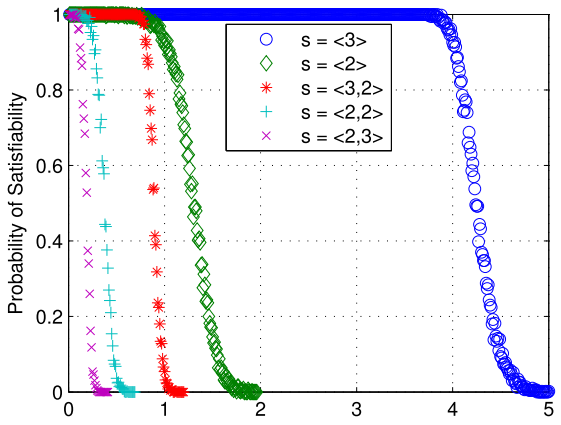


Fig. 2. Probability of satisfiability w.r.t. the *formula : variable ratio* of random 140 variable s-formulae knowledge bases (generated with GENBAL).

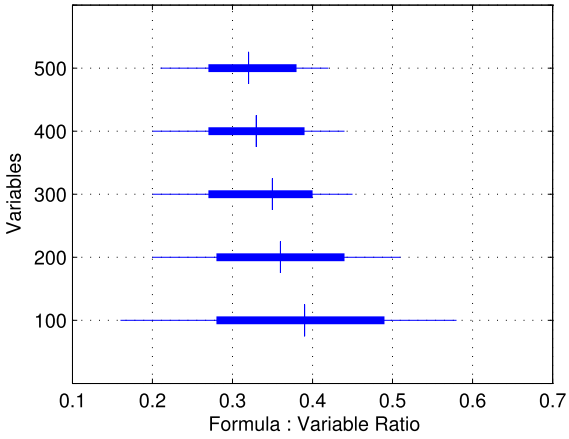


Fig. 3. Phase transitions, represented by 0.01 (thinner line) and 0.1 (thicker line) ϵ-windows, w.r.t. the *formula : variable ratio* of random v variable $\langle 2,2\rangle$-formulae knowledge bases where $v \in \{100, 200, \ldots, 500\}$. The vertical line indicates the crossover point where the *formula : variable ratio* values to the left (resp. right) result in a probability of satisfiability greater than (resp. less than) 0.5.

$s \in \{\langle 2\rangle, \langle 3\rangle, \langle 2,2\rangle, \langle 2,3\rangle, \langle 3,2\rangle\}$ and a constant $v = 140$, the probability of satisfiability for each value of $r \in \{0.01, 0.02, \ldots\}$ was determined from a sample of 500 randomly generated knowledge bases. Then the experiment, for each value of s, ran until the samples for 10 consecutive values of r were found to have a probability of satisfiability equal to 0. The sharp phase transition for each s is the *formula : variable ratio* range where the probability of satisfiability goes from almost 1 to almost 0. These results, as well as those in [57], show that increasing the value of r also increases the probability of randomly generating an inconsistent knowledge base with GENBAL. However, the value of r required to generate inconsistent knowledge bases is dependent on s. For example, a random $\langle 2,2\rangle$-formulae knowledge base where $r = 0.75$ will be inconsistent with a high probability, but a random $\langle 3,2\rangle$-formulae knowledge base where $r = 0.75$ will be consistent with a high probability.

In order to describe phase transitions more formally, we now introduce the concept of an ϵ-window [57].

Definition 17. Let S be a set of ordered pairs of real numbers s.t. $(a,b) \in S$, $a > 0$ and $0 \leq b \leq 1$. Let ϵ be a real number s.t. $0 < \epsilon < 0.5$. Let S_ϵ be a set of ordered pairs of real numbers defined as $S_\epsilon = \{(a,b) \in S \mid b \in [\epsilon, 1-\epsilon]\}$. Let $\min(S_\epsilon, a)$ (resp. $\max(S_\epsilon, a)$) denote the minimum (resp. maximum) value for a where $(a,b) \in S_\epsilon$. Then the ϵ-window of S w.r.t. b is an interval of real numbers defined as $[\min(S_\epsilon, a), \max(S_\epsilon, a)]$.

Let $E = \{e_1, e_2, \ldots, e_n\}$ be a set of samples of randomly generated knowledge bases. Let $S = \{(r_1, p_1), (r_2, p_2), \ldots, (r_n, p_n)\}$ be a set of ordered pairs of real numbers where r_i $(0 \leq i \leq n)$ represents the parameter value of r used to generate each random knowledge base $K \in e_i$ and where p_i represents the probability of satisfiability for the sample e_i. Then the probability of satisfiability ϵ-window for E is an interval of real numbers representing the *formula : variable ratio* range defined by the minimum and maximum values of r required to generate a sample $e_i \in E$ where the probability of satisfiability for e_i is in the interval $[\epsilon, 1-\epsilon]$.

Fig. 3 shows the phase transitions, represented by 0.01 and 0.1 ϵ-windows, for random $\langle 2,2\rangle$-formulae knowledge bases with different values for v. Specifically, given each $v \in \{100, 200, \ldots, 500\}$, then samples of 1000 knowledge bases

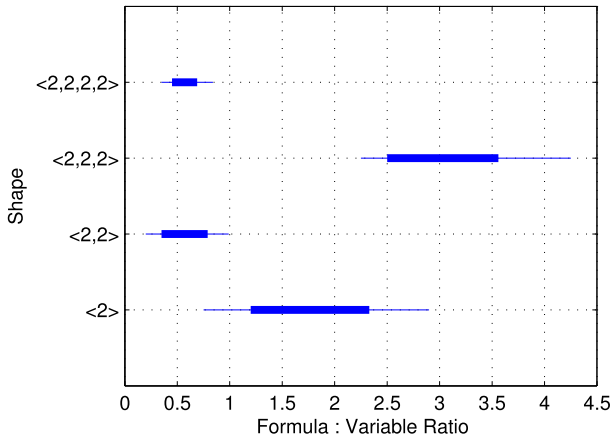


Fig. 4. Phase transitions, represented by 0.01 (thinner line) and 0.1 (thicker line) ϵ-windows, w.r.t. the *formula : variable ratio* of random 20 variable s-formulae knowledge bases where $s \in \{\langle 2\rangle, \langle 2,2\rangle, \ldots, \langle 2,2,2,2\rangle\}$.

were generated for each $r \in \{0.11, 0.12, \ldots, 0.7\}$. A probability of satisfiability was then determined for each sample. These ϵ-windows clearly demonstrate that increasing the number of variables in the language serves to *sharpen* the phase transition region for random fixed-shape knowledge bases but does not change the region's general position. In other words, it is likely that the phase transition for a given value of s will be roughly the same for any value of v (this is supported by the results found in [57]). However, the phase transition region is less precise with lower numbers of variables, e.g., $v = 100$ in Fig. 3.

In a similar experiment, Fig. 4 shows the phase transitions, represented by 0.01 and 0.1 ϵ-windows, for random 20 variable s-formulae knowledge bases. From this group, $s = \langle 2,2,2,2\rangle$ represents the largest and most complex formulae while $s = \langle 2\rangle$ represents the smallest and least complex. Of the four values of s, the phase transition occurs at the highest *formula : variable ratio* range when $s = \langle 2,2,2\rangle$ in contrast to the lower ratios required when $s = \langle 2,2\rangle$ and $s = \langle 2,2,2,2\rangle$. In other words, given a fixed set of variables, a random $\langle 2,2,2\rangle$-formulae knowledge base will need significantly more formulae in order to increase the probability of generating an inconsistent base, since a higher ratio means there is a greater number of formulae in relation to the fixed number of variables.

We now move on to analyzing inconsistency through MISes. Since finding MISes in a knowledge base is a computationally difficult problem, a solution may not always be found within a fixed period of time. For this reason, in order to carry out a suitable evaluation of MIMUS, it is necessary to characterize the feasibility of computing MISes in a given sample of knowledge bases with MIMUS.

Definition 18. Let S be a sample of knowledge bases, let n be a real number and let $\{\mathsf{Gen}, \mathsf{Trans}\}$ denote the MUS generalization and optimized MUS transformation methods for finding MISes with MIMUS, respectively. Then the n second feasible set of knowledge bases from S w.r.t. a method $\Sigma \in \{\mathsf{Gen}, \mathsf{Trans}\}$, denoted $\mathsf{F}_\Sigma(S,n)$, is the set of knowledge bases from S where the complete set of MISes can be found by MIMUS within n seconds using a method Σ.

We can then define a feasibility measure for a sample of knowledge bases.

Definition 19. Let S be a sample of knowledge bases, let n be a real number and let Σ be a method for finding MISes with MIMUS. Then the probability of n second feasibility for S w.r.t. Σ, denoted $\mathsf{PF}_\Sigma(S,n)$, is defined as:

$$\mathsf{PF}_\Sigma(S,n) = \frac{|\mathsf{F}_\Sigma(S,n)|}{|S|}.$$

If we have a sample of random knowledge bases S generated from fixed parameters v, r and s, then we can determine a probability of n second feasibility w.r.t. a method Σ and these parameters. In general, as we increase the difficulty of samples, e.g., by increasing the value of r (resp. v) but maintaining the values of the remaining parameters, then the value of $\mathsf{PF}_\Sigma(S,n)$ should not increase.

Up to this point we have been considering both consistent and inconsistent knowledge bases. However, from this point, will are only interested in inconsistent knowledge bases. So, in order to evaluate samples of equal size, and since a randomly generated knowledge base may be consistent, then for each sample, random knowledge bases are repeatedly generated and a satisfiability check is carried out until a complete sample is obtained.

In Fig. 5, the previous intuition that the number of MISes would increase as the value of r increases through the phase transition region is shown to be correct. In this experiment, samples of 1000 inconsistent knowledge bases were generated for each $r \in \{0.11, 0.12, \ldots, 0.6\}$, given a constant $v = 100$ and $s = \langle 2,2\rangle$. Then, for each knowledge base in each sample,

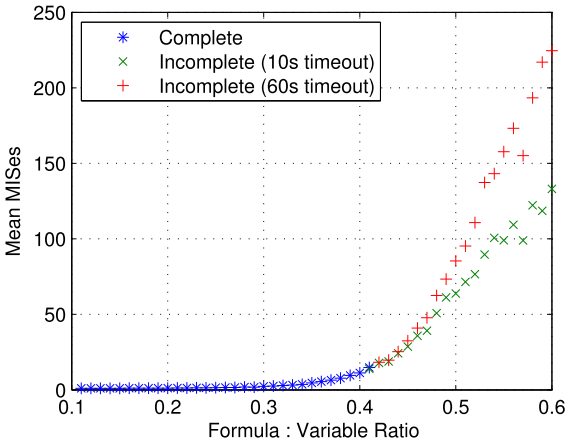


Fig. 5. Mean number of MISes w.r.t. the *formula : variable ratio* of random 100 variable $\langle 2, 2\rangle$-formulae knowledge bases. Each increment of r represents a sample of inconsistent knowledge bases S where $|S| = 1000$. The *complete* points indicate the mean number of MISes for S since $\mathsf{PF}_{\mathsf{Gen}}(S, 10) = 1$. The *incomplete* points indicate the mean number of MISes for the samples $\mathsf{F}_{\mathsf{Gen}}(S, 10)$ and $\mathsf{F}_{\mathsf{Gen}}(S, 60)$ from S, respectively.

MIMUS was run using the generalization method to find MISes, with a timeout set at 10 s. The *complete* points in Fig. 5, e.g., where $r \leq 0.41$, show the mean number of MISes for the complete sample, since MISes were found for all knowledge bases in the sample within this timeout. The *incomplete* points on the other hand, e.g., where $r > 0.41$, only show the mean number of MISes for the subset of the sample where all MISes were found within the 10 s timeout or a longer 60 s timeout. It is likely that the actual mean number of MISes for the samples where timeouts occurred will be higher than is shown. This is supported by the results for both timeouts, where the mean number of MISes rises more sharply as the timeout is increased from 10 s to 60 s. In other words, the most difficult problems (where a timeout occurred) have, on average, a greater number of MISes.

Given that the probability of satisfiability crossover point for 100 variable $\langle 2, 2\rangle$-formulae knowledge bases was found in Fig. 3 to be where $r = 0.39$, it is clear from Fig. 5 that an explosion of MISes occurs past this point. The significance of this result is that we can say, with a high degree of certainty, that increasing the value of r will increase the number of MISes. This is particularly important for the evaluation of any MIS implementation (and equally relevant to the evaluation of MUS implementations) since we can identify the region where the number of MISes will begin to increase from 1, i.e., the phase transition region. Moreover, we know to expect that the difficulty (in terms of finding all MISes) will also increase with r, becoming prohibitively difficult soon after this point.

We have seen in Fig. 3 that increasing the number of variables serves to sharpen the probability of satisfiability phase transition region but does not affect the region's general position. Fig. 6 shows the results of an experiment on the effect of the number of variables in $\langle 2, 2\rangle$-formulae knowledge bases w.r.t. the mean number of MISes and the *formula : variable ratio*. So, for each $r \in \{0.11, 0.12, \ldots, 0.6\}$, 1000 inconsistent knowledge bases were generated for each sample S and the mean number of MISes was found for the set of 10 s feasible knowledge bases $\mathsf{F}_{\mathsf{Gen}}(S, 10)$. The sequence was aborted when MISes were found within 10 s for fewer than 50% of knowledge bases in a sample S, i.e., $\mathsf{PF}_{\mathsf{Gen}}(S, 10) < 0.5$. The results in Fig. 6 show that the explosion of MISes occurs more quickly as the number of variables is increased. The implication in terms of evaluating MIMUS, is that the maximum *feasible formula : variable ratio* (in terms of the maximum value of r where all MISes can be found within a specified time) will decrease as the number of variables increases, since the number of MISes will increase more quickly in relation to r.

5.3.2. *Performance of* MIMUS

In the next experiment, shown in Fig. 7, the total runtime of MIMUS (using the MUS generalization method) was analyzed w.r.t. the number of MCSes (resp. MISes) in random inconsistent knowledge bases. Specifically, Fig. 7 represents the individual results for the set of 60 s feasible random 100 variable $\langle 2, 2\rangle$-formulae knowledge bases from all samples shown in Fig. 5. Clearly, in these knowledge bases, there is more correlation between the number of MCSes and the time it takes MIMUS to find the complete set of MISes (using the generalization method), than with the actual number of MISes. Notably, with this set of knowledge bases, it becomes much more difficult to find all MISes when the number of MCSes reaches the 2000–3000 region, after which the payoff in terms of finding the complete set of MISes within 60 s, begins to decline. However this result is unsurprising given the hitting set dualization approach taken by CAMUS and a more direct approach to finding MISes would likely produce a greater degree of correlation between runtime and the number of MISes.

So far we have made several findings w.r.t. random $\langle 2, 2\rangle$-formulae knowledge bases and the performance of MIMUS, including: an explosion of MISes that occurs in the phase transition region; a correlation between the number of MCSes and the time taken by MIMUS to find the complete set of MISes (using the MUS generalization method); and the effect of increasing the number of variables in the language. Importantly, these findings are equally applicable to other types of random s-formulae knowledge bases. For example, in each case we have found that an explosion of MISes occurs where

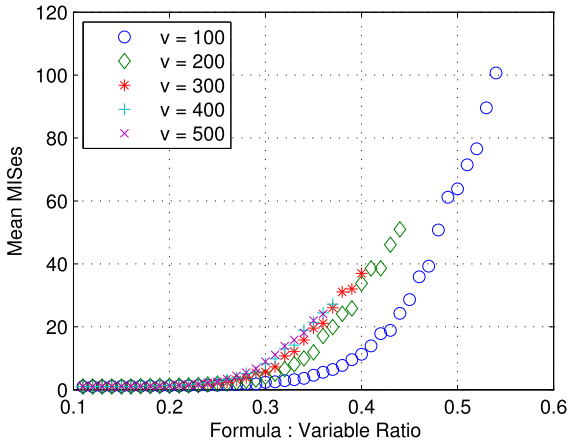


Fig. 6. Mean number of MISes in random v variable $\langle 2,2\rangle$-formulae knowledge bases. For each $v \in \{100, 200, \ldots, 500\}$, each increment of r represents a sample of knowledge bases S where $|S| = 1000$. The points indicate the mean number of MISes for the sample $\mathsf{F}_{\mathsf{Gen}}(S, 10)$ from S where $\mathsf{PF}_{\mathsf{Gen}}(S, 10) \geq 0.5$.

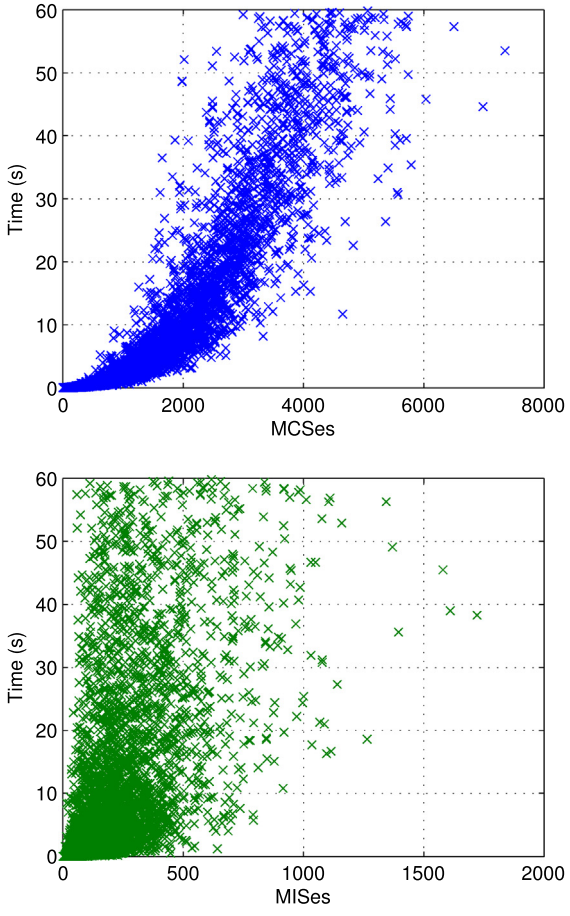


Fig. 7. Total runtime of MIMUS using MUS generalization for random 100 variable $\langle 2,2\rangle$-formulae knowledge bases where $r \in \{0.11, 0.12, \ldots, 0.6\}$ in a sample S. Each point indicates the total runtime for a knowledge base $K \in \mathsf{F}_{\mathsf{Gen}}(S, 60)$ w.r.t. the number of MCSes (resp. MISes) in K.

the interval values of r represents the unique phase transition region for a particular s (as was shown in Fig. 5 for random 100 variable $\langle 2,2\rangle$-formulae knowledge bases). However, the value of s (as with r and v) will also have an impact on the performance of MIMUS. Before going any further, we now compare the two approaches to finding MISes: MUS generalization; and optimized MUS transformation.

In Fig. 8, samples of 1000 random inconsistent 20 variable $\langle 2,2\rangle$-formulae knowledge bases were generated for each $r \in \{0.11, 0.12, \ldots, 2\}$. Then, for each sample, MIMUS was run (with a timeout set at 10 s) using the MUS generalization and

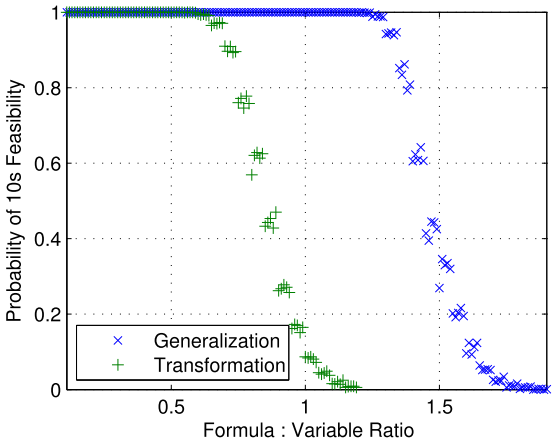


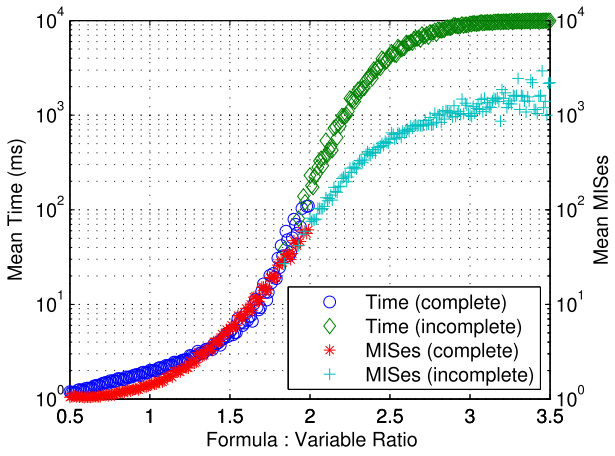
Fig. 8. Probability of 10 s feasibility for random 20 variable $\langle 2,2\rangle$-formulae knowledge bases. Each increment of r represents a sample S where $|S| = 1000$. The *generalization* and *transformation* points indicate the probability of feasibility values $\mathsf{PF}_{\mathsf{Gen}}(S, 10)$ and $\mathsf{PF}_{\mathsf{Trans}}(S, 10)$ for S, respectively.

then the optimized MUS transformation methods for finding MISes. A probability of 10 s feasibility for both methods was then determined for each sample. So, the results in Fig. 8 confirm the findings in [43] that MUS generalization has a clear advantage over the optimized MUS transformation approach. The main reason for the poor performance of the optimized MUS transformation method is that, in random fixed-shape knowledge bases, MUSes will be evenly distributed between MISes and so, as the size of formulae increases, so too does the number of MUSes that must be computed for each MIS. On the other hand, if a knowledge base were to have a MUS : MIS ratio below 1, then it is possible that the optimized MUS transformation method would be preferable since extrapolating multiple MISes from one MUS would be easier than computing multiple GMUSes. However, since the state-of-the-art algorithms for finding MUSes are based on hitting set dualization, the number of CoMSSes may prove problematic in this case. Fortunately, the optimized MUS transformation approach is independent of the algorithm used to compute MUSes so future algorithms may improve its viability. In terms of this evaluation, these factors justify the selection of the generalization approach for all remaining experiments.

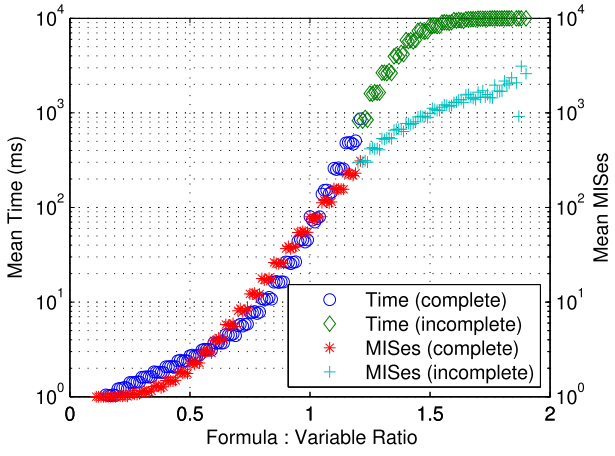
Moving on to Fig. 9. In this experiment, the total runtime and the number of MISes for different types of random 20 variable s-formulae knowledge bases are compared w.r.t. the *formula : variable ratio* for each sample. Given that lower values of r have a high probability of satisfiability, generating inconsistent knowledge bases is more difficult with these values. For this reason, we first found the lowest value of r for which a sample of 1000 inconsistent knowledge bases could be feasibly generated for each $s \in \{\langle 2\rangle, \langle 2,2\rangle, \langle 2,2,2\rangle, \langle 2,2,2,2\rangle\}$. From these lower bounds, samples were obtained for each s as usual by incrementing r by 0.01, generating a sample S and attempting to compute the set of MISes for each knowledge base in S within a 10 s timeout. As such, mean values for runtime and the number of MISes were found from the set of 10 s feasible knowledge bases $\mathsf{F}_{\mathsf{Gen}}(S, 10)$. This process was aborted for each s when 10 samples were found to have a probability of 10 s feasibility equal to 0. Thus, the range of r for each s represents a lower bound of r at which MISes could be feasibly generated, to an upper bound of r at which MISes could be feasibly computed within 10 s. So, for each s and for each sample in this range, Fig. 9 shows the mean runtime on the left y-axis and the mean number of MISes on the right y-axis.

It is worth noting that while we increment r by a fixed 0.01 for each s, in practice this increment may be too small to have an immediate effect. For example, if we generate a random knowledge base where $v = 100$ and $r = 0.611$ and another where $v = 100$ and $r = 0.612$, then both will have 100 variables and $100 \times r \simeq 61$ formulae either way. This effect is apparent in the 'step' pattern which is most visible in Fig. 9d. Seeing as we focus on different regions of r for each s, a 0.01 increment was chosen as a simple compromise for providing comparable parameter values but precise results. From this figure we can see that mean runtime values level off towards 10 s. This is a result of the timeout, since the results only include runtime values for 10 s feasible knowledge bases. In other words, the mean runtime for the incomplete points represents a baseline for these samples since the results reflect easier problems. Likewise, referring to the results from Fig. 5 (which shows an increase in the mean number of MISes when the timeout is increased), we can infer that the mean number of MISes for the incomplete points is a probable baseline for these samples. In addition, the incomplete curves become less uniform for each s as r is increased because the 10 s feasible sample sizes decrease as r is increased. However, if we follow the curves w.r.t. the mean number of MISes for the complete points, it is easy to see why computing MISes quickly becomes infeasible as r is increased – even conservative estimates will show that the number of MISes becomes huge.

Further to this finding, we can also see that the number of MISes does not predictably increase from 1 for the lowest values of r. When $s = \langle 2,2,2\rangle$ for example, the lowest value of r in which samples of inconsistent knowledge bases can be feasibly generated was found to be around $r = 2.1$. However, when inconsistent knowledge bases are occasionally generated at this ratio, they generally contain at least several hundred MISes. Thus, it is very difficult in practice to generate $\langle 2,2,2\rangle$-formulae knowledge bases exhibiting small numbers of MISes (e.g., less than 100). This also helps to explain why, for this s, only one sample was found to have a probability of 10 s feasibility equal to 1. In general terms, we can see that MIMUS can compute as many as 60,000 MISes within 10 s for many complex formulae, e.g., where $s = \langle 2,2,2\rangle$, while in



(a) $s = \langle 2 \rangle$



(b) $s = \langle 2, 2 \rangle$

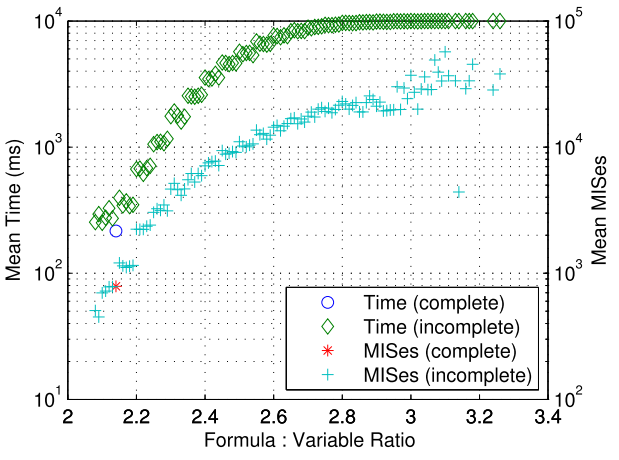
Fig. 9. Mean runtime and number of MISes for random 20 variable s-formulae knowledge bases around the phase transition region. Each increment of r represents a sample S where $|S| = 1000$. The left y-axis indicates the mean runtime while the right y-axis indicates the mean number of MISes. In each case, the *complete* points indicate the results for S since $\mathsf{PF}_{\mathsf{Gen}}(S, 10) = 1$, while the *incomplete* points indicate the results for $\mathsf{F}_{\mathsf{Gen}}(S, 10)$.

others, e.g., where $s = \langle 2 \rangle$, the number of MISes computed within 10 s is much lower. So, given that $\langle 2, 2, 2 \rangle$-formulae are more complex than $\langle 2 \rangle$-formulae, this suggests that there is no obvious correlation between s-formula complexity and the number of MISes in 10 s feasible knowledge bases.

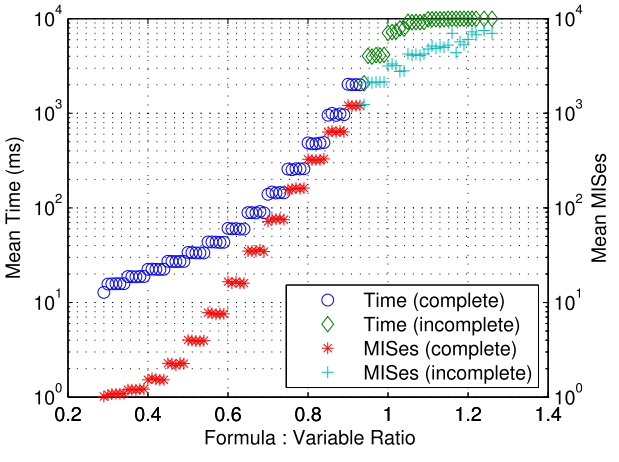
5.3.3. Summary

In summary, the findings of these experiments on properties of random inconsistent knowledge bases and the performance of MIMUS in computing MISes, include:

- the probability of generating an inconsistent random fixed-shape knowledge base increases with the *formula : variable ratio* (r) (see Fig. 2);
- the interval values of r where the probability of generating inconsistent knowledge bases rapidly increases (i.e., the phase transition region) is dependent on the type of fixed-shape formulae (s) (see Figs. 2 and 4);
- increasing the number of variables (v) serves to sharpen this region (see Fig. 3);
- when v is fixed, the number of MISes in a random inconsistent knowledge base explodes in the phase transition region (see Figs. 5 and 6);
- this explosion occurs more quickly as the number of variables is increased (see Fig. 6);
- the number of variables has a major impact on the feasibility of computing MISes (see Fig. 6);
- when v is fixed, the number of MCSes has a greater impact on the performance of MIMUS, using MUS generalization, than the number of MISes (see Fig. 7);
- the generalization approach for finding MISes significantly outperforms optimized MUS transformation (see Fig. 8); and



(c) $s = \langle 2, 2, 2 \rangle$



(d) $s = \langle 2, 2, 2, 2 \rangle$

Fig. 9. (*continued*)

- there is no clear correlation between s-formula complexity and the number of MISes in 10 s feasible knowledge bases (see Fig. 9).

In the following section we will consider how these approaches to finding MISes can be incorporated into practical inconsistency handling techniques using a variety of inconsistency measures.

6. Measuring the inconsistency of formulae

In the previous sections we have shown that computing the MISes of a knowledge base is possible. However, in terms of inconsistency handling, identifying MISes is not a solution in itself. Taking the approach of inconsistency resolution for example, the next step would be to modify the parts of the knowledge base involved in inconsistency. However, when considering large or highly inconsistent knowledge bases, manually resolving these inconsistencies would quickly become impractical. Furthermore, not all parts of the knowledge base involved in inconsistency contribute equally to the inconsistency of the base. For this reason, the area of inconsistency measurements has developed.

Obviously there are too many existing inconsistency measures for us to provide a comprehensive overview. However, Fig. 10 shows the classification of a selection of relevant formula-level (and some important base-level) inconsistency measures from the literature. These include: the measure from [30] (denoted Inc_G in [31]); η-consistency from [38]; the Coherence measure from [34]; the S scoring function from [35]; the I_MI, $\mathsf{I}_{\mathsf{LP}_\mathsf{m}}$ and $\mathsf{M}_{\mathsf{I}_\mathsf{MI}}$ measures from [33]; the $\mathsf{S}_{\mathsf{I}_\mathsf{MI}}$ and $\mathsf{S}_{\mathsf{I}_{\mathsf{LP}_\mathsf{m}}}$ Shapley Inconsistency Values also from [33]; the $\mathbf{I_R}$, $\mathbf{MIV_R}$, I_W and MIV_W measures from [47]; the $\mathbf{Blame_v}$ measure from [48]; the $\mathbf{Blame_L}$ measure from [49]; the ID_MUS measure from [69]; and the $\mathsf{I}_{\mathcal{P}_\mathsf{m}}$ measure from [36]. Each circle indicates a particular category of measure where the intersections indicate measures which fall into multiple categories.

In terms of the level at which inconsistency is measured, Inc_G, η-consistency, Coherence, I_MI, $\mathsf{I}_{\mathsf{LP}_\mathsf{m}}$, $\mathbf{I_R}$, I_W and ID_MUS are all base-level measures, while $\mathsf{M}_{\mathsf{I}_\mathsf{MI}}$, $\mathsf{S}_{\mathsf{I}_\mathsf{MI}}$, $\mathsf{S}_{\mathsf{I}_{\mathsf{LP}_\mathsf{m}}}$, $\mathbf{MIV_R}$, MIV_W, $\mathbf{Blame_v}$ and $\mathbf{Blame_L}$ are all formula-level measures. However,

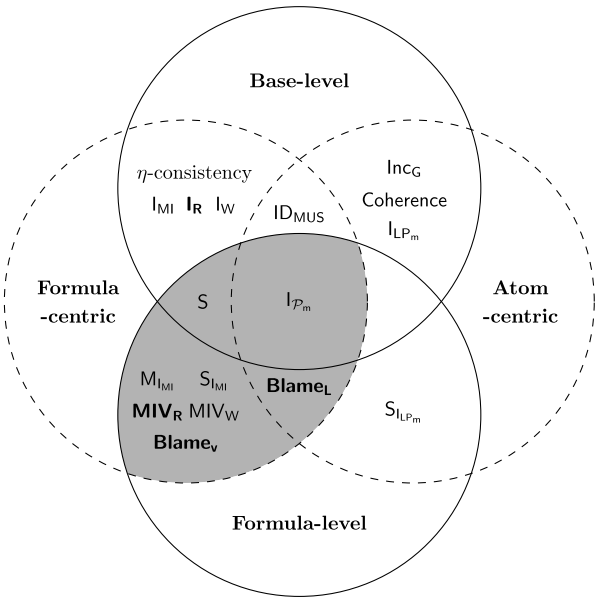


Fig. 10. Euler diagram for the classification of some existing inconsistency measures. In this paper we focus on those in the grey intersection, i.e., formula-centric formula-level measures.

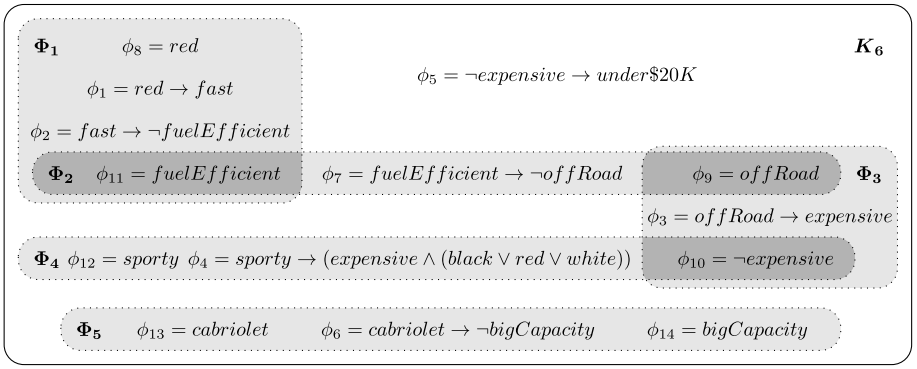


Fig. 11. Euler diagram for the knowledge base K_6 from Example 10 showing $\mathsf{MI}(K_6) = \{\Phi_1, \Phi_2, \ldots, \Phi_5\}$. The intersections indicate formulae which contribute to multiple MISes, i.e., $\phi_{11} = \textit{fuelEfficient}$, $\phi_9 = \textit{offRoad}$ and $\phi_{10} = \neg\textit{expensive}$.

the S scoring function and the $\mathsf{I}_{\mathcal{P}_m}$ measure can be considered as a special case since they represent subset-level measures, i.e., they can be applied both as base-level and as formula-level measures.

The measures can be similarly classified as formula-centric and atom-centric measures. In this case, η-consistency, I_{MI}, S, $\mathsf{MI}_{\mathsf{I}_{\mathsf{MI}}}$, $\mathsf{S}_{\mathsf{I}_{\mathsf{MI}}}$, $\mathbf{I_R}$, I_{W} and $\mathbf{Blame_v}$ are all formula-centric measures while $\mathsf{Inc}_{\mathsf{C}}$, Coherence, $\mathsf{I}_{\mathsf{LP}_m}$, and $\mathsf{S}_{\mathsf{I}_{\mathsf{LP}_m}}$ are all atom-centric measures. Measures that are considered both atom-centric and formula-centric are those which deal with atoms (rather than formulae) involved in some formula-centric characterization of inconsistency such as MISes. The measures $\mathbf{Blame_L}$, $\mathsf{ID}_{\mathsf{MUS}}$ and $\mathsf{I}_{\mathcal{P}_m}$ fall into this category.

As discussed previously, formula-level measures can be more useful than base-level measures from a perspective of inconsistency resolution. The reason for this is that they allow an inconsistency value to be assigned to formulae based on their involvement in the inconsistency of the base, rather than simply assigning an inconsistency value to the knowledge base as a whole. MISes are particularly suitable for defining this type of measure since they represent the *purest* form of formula-centric inconsistency [63,33,48]. Specifically, every formula in a MIS is *essential*, i.e., removing any formula would resolve that particular MIS, so the more MISes in which a particular formula is involved, the more it contributes to the inconsistency of the base. Fig. 11 demonstrates this concept with the knowledge base K_6 from Example 10.

Also, it is often the case that knowledge bases contain additional information which may be useful in characterizing the responsibility of formulae for the inconsistency of the base. In the case of stratified knowledge bases for example, where some formulae are considered more important than others, it may be useful to also consider this when measuring inconsistency. Some measures, such as $\mathbf{Blame_v}$ and $\mathbf{Blame_L}$, can take this additional information into account. So, for each

formula in a knowledge base, the inconsistency value is based on its contribution to inconsistency as well as its degree of importance in the base.

Either way, once an inconsistency ordering of formulae in a knowledge base has been determined, methods such as deletion, splitting or weakening [26] of the most inconsistent formulae, provide a means to systematically and efficiently remove the causes of inconsistency in the base by resolving MISes. The practical benefit of these measures in relation to inconsistency resolution in a network IDS rule set has been discussed in [49].

Although many different inconsistency measures have been proposed theoretically, there is little practical discussion about the implementation of these measures, nor about the feasibility of computing these measures for arbitrary knowledge bases with hundreds or thousands of formulae, such as in QRadar[7] or Snort[8] systems. In most theoretical studies, only toy examples were used. To systematically study the feasibility of computing inconsistency measures for a given knowledge base, in this section we provide a brief overview of some well known inconsistency measures. These include: the $\mathsf{M}_{\mathsf{I}_{\mathsf{MI}}}$ and $\mathsf{S}_{\mathsf{I}_{\mathsf{MI}}}$ measures for flat knowledge bases; and the $\mathbf{Blame}_{\mathbf{V}}$ and $\mathbf{Blame}_{\mathbf{L}}$ measures for stratified knowledge bases. The rational for selecting these measures is that they represent formula-centric formula-level inconsistency measures, founded on MISes, which can be applied to traditional or stratified knowledge bases. As discussed previously, formula-level measures represent a more discriminative inconsistency measure in terms of the syntactic composition of the knowledge base which is useful for a range of applications. Moreover, MISes are often viewed as representing the *purest* form of formula-centric inconsistency [63,33,48] so formula-level measures based on them are intuitive. The S scoring function is omitted from our study since, when applied to a formula, S is equivalent to the $\mathsf{M}_{\mathsf{I}_{\mathsf{MI}}}$ measure. We also omit the $\mathbf{MIV}_{\mathbf{R}}$ and $\mathsf{MIV}_{\mathsf{W}}$ measures since they represent an extension of the $\mathsf{S}_{\mathsf{I}_{\mathsf{MI}}}$ measure which has been incorporated into the $\mathbf{Blame}_{\mathbf{V}}$ measure. We omit other types of formula-level measures which are not based on MISes, e.g., those based on inconsistent truth assignments in paraconsistent models, such as $\mathsf{S}_{\mathsf{I}_{\mathsf{LP}_{\mathsf{m}}}}$. We then carry out a full evaluation of these measures (which have been implemented in MIMUS) using the randomly generated arbitrary knowledge bases discussed in the previous section.

6.1. Measures for flat knowledge bases

A simple base-level measure, denoted $\mathsf{I}_{\mathsf{MI}}(K)$, was defined in [33] as the number of causes of (formula-centric) inconsistency in a knowledge base K, i.e., $\mathsf{I}_{\mathsf{MI}}(K) = |\mathsf{MI}(K)|$. A simple formula-level measure for a formula $\phi \in K$, denoted $\mathsf{M}_{\mathsf{I}}(K, \phi)$, was then defined in terms of a base-level measure I. When I_{MI} is applied as an M_{I} formula-level measure on $\phi \in K$, denoted $\mathsf{M}_{\mathsf{I}_{\mathsf{MI}}}(K, \phi)$, the result is the total number of MISes in which ϕ is involved. In other words, since every formula in a MIS is *essential*, $\mathsf{M}_{\mathsf{I}_{\mathsf{MI}}}(K, \phi)$ is the number of MISes that would be resolved if ϕ were removed from K. So, the more inconsistencies caused by ϕ, the higher the $\mathsf{M}_{\mathsf{I}_{\mathsf{MI}}}$ value of ϕ.

The Shapley Inconsistency Value (SIV), another formula-level inconsistency measure, was also proposed in [33]. This measure takes a base-level inconsistency measure as a payoff function in coalitional form and, using the Shapley value from coalitional game theory, determines a proportional inconsistency value for each formula in a knowledge base. So, given a knowledge base K and $\forall \Phi \subseteq K$, the SIV for $\phi \in K$, denoted $\mathsf{S}_{\mathsf{I}}(K, \phi)$, calculates the sum of the inconsistency of ϕ w.r.t. Φ, normalized with a weighting for Φ w.r.t. K. Unfortunately, given a knowledge base K, a formula $\phi \in K$ and a base-level inconsistency measure I, then the number of subsets is exponential on the size of K where, for each $\Phi \subseteq K$, the value of $\mathsf{I}(\Phi) - \mathsf{I}(\Phi \setminus \{\phi\})$ must be computed in order to calculate the value of $\mathsf{S}_{\mathsf{I}}(K, \phi)$. So, compounded with the complexity of the underlying base-level inconsistency measure I, the SIV is a computationally difficult measure to calculate. However, a logical property of the I_{MI} measure when applied as a SIV allows a feasible computation of this particular SIV [32]. Basically, when the I_{MI} measure is applied as a SIV on a formula $\phi \in K$, denoted $\mathsf{S}_{\mathsf{I}_{\mathsf{MI}}}(K, \phi)$, the result is the sum of the inverse of the cardinality of each MIS in which ϕ is involved, i.e., $\forall \Phi \in \mathsf{MI}(K)$ s.t. $\phi \in \Phi$, $\sum \frac{1}{|\Phi|}$.

6.2. Measures for stratified knowledge bases

If there are two formulae, $\phi \in K$ and $\phi' \in K$, which are equally involved in the MISes of K, but where ϕ is considered more important than ϕ', then it makes sense that they should not be treated equally from a perspective of inconsistency resolution. However, both $\mathsf{M}_{\mathsf{I}_{\mathsf{MI}}}$ and $\mathsf{S}_{\mathsf{I}_{\mathsf{MI}}}$ are based solely on the involvement of ϕ and ϕ' in the MISes of K, which means that $\mathsf{M}_{\mathsf{I}_{\mathsf{MI}}}(K, \phi) = \mathsf{M}_{\mathsf{I}_{\mathsf{MI}}}(K, \phi')$ and $\mathsf{S}_{\mathsf{I}_{\mathsf{MI}}}(K, \phi) = \mathsf{S}_{\mathsf{I}_{\mathsf{MI}}}(K, \phi')$. To address this issue, a number of inconsistency measures have been proposed which can discriminate between the responsibility of ϕ and ϕ' for the inconsistency of K, if K is stratified. Before discussing these however, we first introduce a particular type of stratified knowledge base.

Definition 20. Let K be a knowledge base. Let $[K]$ be a partition of K where $[K] = \{K_1, \ldots, K_n\}$. Let $K_m \succ_p K_{m+1}$ (where $1 \leq m < n$) denote that K_m has a higher priority than K_{m+1}. A prioritized knowledge base[9] $\widehat{K}$ from K is an n-tuple of cells from $[K]$, denoted $\widehat{K} = \langle K_1, \ldots, K_n \rangle$, where $K_1 \succ_p \ldots \succ_p K_n$.

[7] QRadar is an exploit detection system from Q1 Labs/IBM.

[8] Snort is an industry standard network IDS.

[9] For the purposes of this work, we simplify the definition of a Type-II prioritized knowledge base presented in [48].

Let $\widehat{K}^{(i)}$ denote the i-th priority level of $\widehat{K}$. Given a prioritized knowledge base $\widehat{K} = \langle K_1, \ldots, K_n \rangle$, then the *flattening* of $\widehat{K}$, denoted $\overline{\widehat{K}}$, is defined as

$$\overline{\widehat{K}} = \bigcup_{i=1}^{n} \widehat{K}^{(i)}.$$

To illustrate this definition, we introduce a prioritized version of the knowledge base K_6 from Example 10, which will be referred to for the remainder of this paper.

Example 15. Given the knowledge base $K_6 = \{\phi_1, \phi_2, \ldots, \phi_{14}\}$ from Example 10, then the prioritized knowledge base $\widehat{K_6}$ is defined as

$$\widehat{K_6} = \big\langle \{\phi_1, \phi_2, \phi_3, \phi_4, \phi_5, \phi_6, \phi_7\}, \{\phi_{11}, \phi_{14}\}, \{\phi_8, \phi_9, \phi_{10}, \phi_{12}, \phi_{13}\} \big\rangle.$$

The knowledge base K_6 represents a scenario of choosing a new family car. In particular, the formulae $\phi_1, \phi_2, \ldots, \phi_7$ represent domain knowledge while the formulae $\phi_8, \phi_9, \ldots, \phi_{14}$ represent the combined preferences from different family members. In this case, the family's preferences are constrained by domain knowledge and so, in the prioritized knowledge base $\widehat{K_6}$, domain knowledge has a higher priority level than preferences. Moreover, the formulae ϕ_{11} and ϕ_{14} are agreed by the family to be the most important (or desirable) preferences. In other words, $\widehat{K_1}$ has three priority levels representing domain knowledge, more important preferences and less important preferences, in that order.

The $\textbf{Blame}_{\textbf{v}}$ measure was introduced in [48] as a formula-level inconsistency measure for prioritized knowledge bases. Essentially, given a prioritized knowledge base $\widehat{K}$ and a formula $\phi \in \overline{\widehat{K}}$, then $\textbf{Blame}_{\textbf{v}}(\widehat{K}, \phi)$ is a measure of the involvement of ϕ in the MISes of $\widehat{K}$, over each priority level. As such the result of $\textbf{Blame}_{\textbf{v}}(\widehat{K}, \phi)$, where $\widehat{K} = \langle K_1, \ldots, K_n \rangle$, is a vector of n inconsistency values representing the inconsistency of ϕ at each level. So using lexicographic ordering between vectors, the $\textbf{Blame}_{\textbf{v}}$ measure can identify a total order over formulae in a prioritized knowledge base in terms of their involvement in the MISes of the base as well as their relative priority.

The final inconsistency measure we discuss, denoted $\textbf{Blame}_{\textbf{L}}$, was introduced in [49] as an extension to the $\textbf{Blame}_{\textbf{v}}$ measure. Unlike the three previous measures, $\textbf{Blame}_{\textbf{L}}$ also considers the atoms involved in MISes. Essentially the $\textbf{Blame}_{\textbf{v}}$ measure is extended with a new significance function for MISes, defined in terms of the normalized number of atoms at each priority level of a MIS. The rational being that if two formulae are of varying degrees of complexity, e.g., where $\phi_4 : \mathit{sporty} \rightarrow (\mathit{expensive} \wedge (\mathit{black} \vee \mathit{red} \vee \mathit{white}))$ is more complex than $\phi_{12} : \mathit{sporty}$, then they should not be treated as contributing equally to the inconsistency of the base. This is also the approach subsequently applied in the $\mathsf{ID}_{\mathsf{MUS}}$ measure from [69]. In general the $\textbf{Blame}_{\textbf{L}}$ measure provides a deeper inspection of the formulae involved in inconsistency. In this way, blame can be attributed to formulae in terms of their involvement in inconsistency, their knowledge base priority level and their syntactic complexity.

6.3. A practical example

A full comparison, based on the knowledge base K_6 from Example 10 (prioritized as $\widehat{K_1}$ in Example 15), of the four inconsistency measures reviewed in this section is shown in Table 1. The formula ϕ_5 is assigned a null inconsistency value by each measure, since ϕ_5 is a free formula. The simplest measure $\mathsf{M}_{\mathsf{I_{MI}}}$ produces an inconsistency ordering with two strata where ϕ_9, ϕ_{10} and ϕ_{11} are considered (equally) as more inconsistent than the remaining formulae. On the other hand, the more discriminative $\mathsf{S}_{\mathsf{I_{MI}}}$ measure identifies ϕ_9 and ϕ_{10} equally as being more inconsistent than ϕ_{11}. Unsurprisingly, when the prioritization of formulae is taken into consideration (as $\widehat{K_6}$) by the measures $\textbf{Blame}_{\textbf{v}}$ and $\textbf{Blame}_{\textbf{L}}$, a more precise inconsistency ordering is found. Interestingly, $\textbf{Blame}_{\textbf{v}}$ identifies ϕ_9 as most inconsistent, while in contrast, $\textbf{Blame}_{\textbf{L}}$ identifies ϕ_{10} as the most inconsistent. This disagreement can be explained by the number of atoms involves in the MISes for which ϕ_9 and ϕ_{10} are responsible, as well as the distribution of atoms across priority levels.

6.4. Experiments

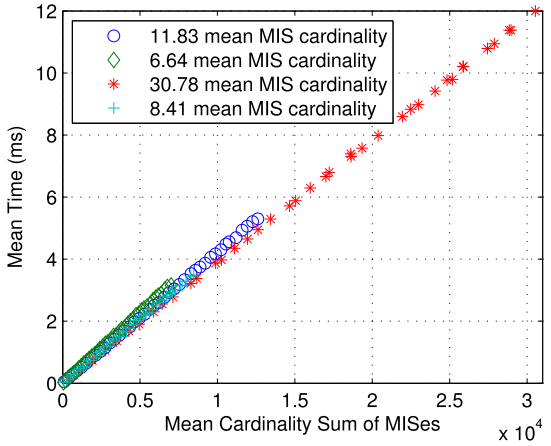
Having analyzed properties of random fixed-shape knowledge bases and the feasibility of finding MISes with MIMUS in the previous section, we can now begin to focus on the main motivation behind this work, i.e., the computation of formula-level inconsistency measures. First of all, given a knowledge base K, then the *cardinality sum of MISes* is defined as:

$$\sum_{\Phi \in \mathsf{MI}(K)} |\Phi|.$$

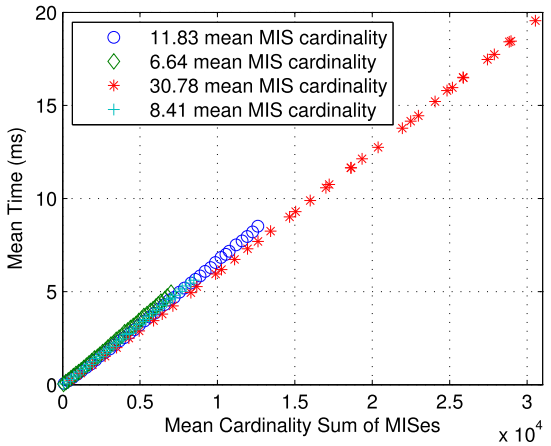
In Fig. 12, the mean runtime required to calculate the $\mathsf{M}_{\mathsf{I_{MI}}}$ and $\mathsf{S}_{\mathsf{I_{MI}}}$ measures for all formulae in a knowledge base is shown w.r.t. cardinality sum of MISes. In this case, the subset of random inconsistent knowledge bases generated in Fig. 9, exhibiting between 1 and 1000 MISes, were divided into 40 (varying) sample groups for each s: those with 1–25 MISes; those with 26–50 MISes; etc. Then, for each s and for each sample group, the mean cardinality sum of MISes and the mean calculation time for each measure were recorded. The purpose of this was to allow the calculation time to be analyzed

ϕ_i	$\mathsf{M}_{\mathsf{I}_{\mathsf{MI}}}(K_6, \phi_i)$	$\mathsf{S}_{\mathsf{I}_{\mathsf{MI}}}(K_6, \phi_i)$	$\mathbf{Blame}_{\mathsf{V}}(\widehat{K_6}, \phi_i)$	$\mathbf{Blame}_{\mathsf{L}}(\widehat{K_6}, \phi_i)$
ϕ_1	1	0.25	(0.02, 0.02, 0.02)	(0.05, 0.17, 0.07)
ϕ_2	1	0.25	(0.02, 0.02, 0.02)	(0.05, 0.17, 0.07)
ϕ_3	1	0.33	(0.00, 0.00, 0.11)	(0.00, 0.00, 0.20)
ϕ_4	1	0.33	(0.00, 0.00, 0.11)	(0.00, 0.00, 0.20)
ϕ_5	0	0.00	(0.00, 0.00, 0.00)	(0.00, 0.00, 0.00)
ϕ_6	1	0.33	(0.00, 0.06, 0.06)	(0.00, 0.25, 0.10)
ϕ_7	1	0.33	(0.00, 0.06, 0.06)	(0.00, 0.25, 0.10)
ϕ_8	1	0.25	(0.04, 0.02, 0.00)	(0.09, 0.17, 0.00)
ϕ_9	2	0.67	(0.11, 0.06, 0.06)	(0.18, 0.25, 0.10)
ϕ_{10}	2	0.67	(0.11, 0.00, 0.11)	(0.32, 0.00, 0.20)
ϕ_{11}	2	0.58	(0.10, 0.00, 0.08)	(0.18, 0.00, 0.17)
ϕ_{12}	1	0.33	(0.06, 0.00, 0.06)	(0.23, 0.00, 0.10)
ϕ_{13}	1	0.33	(0.06, 0.06, 0.00)	(0.09, 0.25, 0.00)
ϕ_{14}	1	0.33	(0.06, 0.00, 0.06)	(0.09, 0.00, 0.10)

Table 1
Comparison of formula-level inconsistency values for the knowledge base K_6 from Example 10 (and the related prioritized knowledge base $\widehat{K_6}$ from Example 15) rounded to 2 decimal places. Each row represents a formula $\phi_i \in K_6$ and each column represents the inconsistency value for ϕ w.r.t. a formula-level inconsistency measure. Darker cells denote formulae which are more inconsistent w.r.t. this measure.



(a) $\mathsf{M}_{\mathsf{I}_{\mathsf{MI}}}$



(b) $\mathsf{S}_{\mathsf{I}_{\mathsf{MI}}}$

Fig. 12. Mean runtime required to calculate the $\mathsf{M}_{\mathsf{I}_{\mathsf{MI}}}$ and $\mathsf{S}_{\mathsf{I}_{\mathsf{MI}}}$ measures w.r.t. the cardinality sum of MISes for samples of random fixed-shaped knowledge bases with up 1000 MISes and different mean MIS cardinalities.

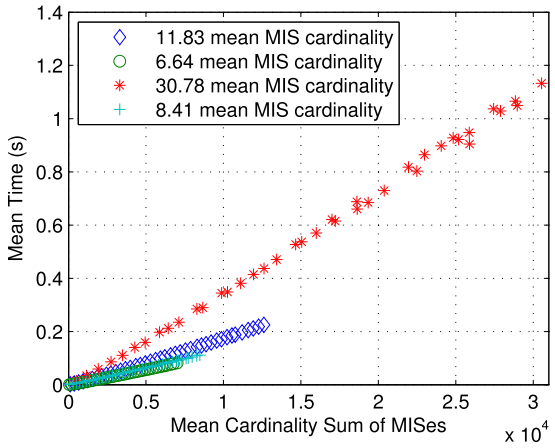


Fig. 13. Mean runtime required to calculate the $\textbf{Blame}_{\mathbf{V}}$ measure w.r.t. the cardinality sum of MISes for samples of random fixed-shaped knowledge bases with up 1000 MISes and different mean MIS cardinalities.

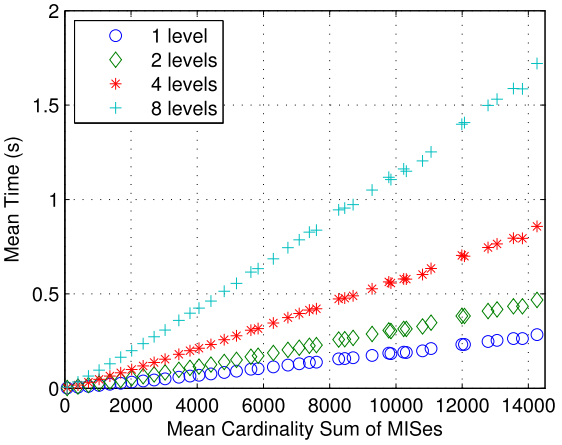


Fig. 14. Mean runtime required to calculate the $\textbf{Blame}_{\mathbf{V}}$ measure w.r.t. the cardinality sum of MISes for samples of random fixed-shaped knowledge bases with up 1000 MISes and different degrees of prioritization.

w.r.t. the cardinality sum of MISes while avoiding the difficulty in randomly generating specific numbers of MISes. However, the limitation was that the number of samples in each group decreased with the increasing number of MISes caused by the explosion of MISes in the phase transition region. Next, in order to evaluate the effect of MIS cardinality, mean MIS cardinality values were determined for each s and these are shown in the figure. Clearly calculating the $\mathsf{M}_{\mathsf{I}_{\mathsf{MI}}}$ and $\mathsf{S}_{\mathsf{I}_{\mathsf{MI}}}$ measures is linear in time w.r.t. the cardinality sum of MISes. Moreover, the cardinality of individual MISes is irrelevant as long as the cardinality sum of MISes remains the same.

By repeating this experiment on the $\textbf{Blame}_{\mathbf{V}}$ measure, differences between this measure and the previous measures become apparent. Fig. 13 shows that while this measure is also linear in time w.r.t. the cardinality sum of MISes, in this case the cardinality of MISes does have an impact on runtime. It is clear that random knowledge bases with larger MISes, i.e., where the mean MIS cardinality is 30.78, represent more difficult problems. The explanation for this overhead is the requirement to compute $\mathsf{Opp}_{\mathsf{V}}$ for each formulae in a MIS (see full definition in [48]). Aside from this, the $\textbf{Blame}_{\mathbf{V}}$ measure is significantly more difficult than the $\mathsf{M}_{\mathsf{I}_{\mathsf{MI}}}$ and $\mathsf{S}_{\mathsf{I}_{\mathsf{MI}}}$ measures (taking around 200 ms for a 10,000 cardinality sum of MISes, given a flat knowledge base with 6.64 mean MIS cardinality, compared to 4 ms and 6 ms for $\mathsf{M}_{\mathsf{I}_{\mathsf{MI}}}$ and $\mathsf{S}_{\mathsf{I}_{\mathsf{MI}}}$, respectively).

Up to this point we have only carried out experiments on flat knowledge bases. However, in order to accurately evaluate the $\textbf{Blame}_{\mathbf{V}}$ and $\textbf{Blame}_{\mathbf{L}}$ prioritized measures, random prioritized knowledge bases must also be generated. The format used by MIMUS to represent these prioritized knowledge bases is to append an integer to each formula representing its relative ordering. So, in order to randomly prioritize a knowledge base K with n levels (where $n \leq |K|$), a tool was implemented using the in-built BASH pseudo-random integer generator `$RANDOM`. Then for each $\phi \in K$, an integer $1 \leq i \leq n$ was randomly generated and assigned to ϕ. After all formulae were assigned a priority level, the knowledge base was analyzed to ensure that the number of unique priority levels was exactly n. If this was not the case then the process was repeated until the knowledge base contained exactly n priority levels.

Fig. 14 shows the results of an experiment on calculating the $\textbf{Blame}_{\mathbf{V}}$ measure for this type of randomly generated prioritized knowledge base. Specifically, the set of $\langle 2, 2\rangle$-formulae knowledge bases from Fig. 13 were randomly prioritized

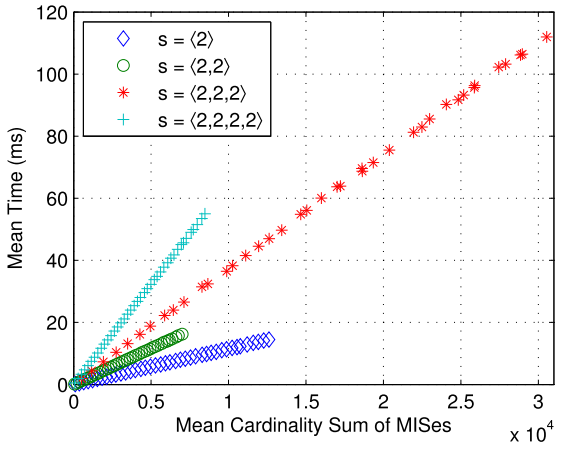


Fig. 15. Mean runtime required to calculate the $\textbf{Blame}_{\textbf{L}}$ measure w.r.t. the cardinality sum of MISes for samples of random fixed-shaped knowledge bases with up 1000 MISes and for each shape $s \in \{\langle 2\rangle, \langle 2,2\rangle, \ldots, \langle 2,2,2,2\rangle\}$.

with $n \in \{1, 2, 4, 8\}$ priority levels. Again the knowledge bases, exhibiting between 1 and 1000 MISes, were split into 40 sample groups and, for each sample group, the mean calculation time and mean cardinality sum of MISes were recorded. In this case we can see that increasing the number of priority levels in a knowledge base results in a linear increase in the runtime required to calculate the $\textbf{Blame}_{\textbf{V}}$ measure (taking an additional 100 ms per level given a 10,000 cardinality sum of MISes). Whether a knowledge base is flat or prioritized has no effect on the difficulty of finding MISes.

The final experiment is shown in Fig. 15. Since $\textbf{Blame}_{\textbf{L}}$ is an extension of the $\textbf{Blame}_{\textbf{V}}$ measure, properties of $\textbf{Blame}_{\textbf{V}}$, such as the effect of MIS cardinality and priority levels, will also be true for $\textbf{Blame}_{\textbf{L}}$. For this reason the mean $\textbf{Blame}_{\textbf{V}}$ time was subtracted from the mean $\textbf{Blame}_{\textbf{L}}$ time and so Fig. 15 only shows the *additional* time required to calculate $\textbf{Blame}_{\textbf{L}}$. This serves to isolate the computation time required to calculate the additional information and, importantly, to accurately evaluate the effect of formula complexity on this measure. So, given the same sample groups of random s-formulae knowledge bases from Fig. 13, Fig. 15 shows the additional runtime required to calculated the $\textbf{Blame}_{\textbf{L}}$ measure w.r.t. the cardinality sum of MISes for each s. The results are unsurprising in that the most difficult knowledge bases w.r.t. $\textbf{Blame}_{\textbf{L}}$ are those with the most complex formulae, e.g., where $s = \langle 2, 2, 2, 2\rangle$.

7. Related work

There are very few implementations of MIS-based formula-level inconsistency measures for knowledge bases. As far as we are aware, the most relevant examples include: an implementation for measuring incoherence in description logic through scoring functions [61]; an implementation of an *abstract and refine* technique for finding MUSes, using the $\mathsf{S}_{\mathsf{I_{MI}}}$ measure as a heuristic on arbitrarily grouped clauses [28]; the INCMEASURER tool for measuring the significance of inconsistency in requirements specifications from [45]; and the PRISM reasoning platform (based on the $\mathsf{S}_{\mathsf{I_{MI}}}$ measure) from [39].

The Scoring function from [35] is defined for a knowledge base K and $\forall \Phi \subseteq K$, denoted $\mathsf{S}(K, \Phi)$, as $\mathsf{S}(K, \Phi) = |\mathsf{MI}(K)| - |\mathsf{MI}(K \setminus \Phi)|$. As mentioned in Section 6, the Scoring function is equivalent to the $\mathsf{M}_{\mathsf{I_{MI}}}$ measure when applied to individual formulae in a knowledge base, i.e., $\forall \phi \in K$ then $\mathsf{S}(K, \{\phi\}) = \mathsf{M}_{\mathsf{I_{MI}}}(K, \Phi)$. In description logic, the notion of incoherence in ontologies corresponds (loosely) to the notion of inconsistency in classical knowledge bases. In particular, the notion of minimal incoherence-preserving sub-TBoxes (MIPSes) corresponds to the notion of MISes. The relationship between MIPSes and MISes was used in [61] to extend the Scoring function for ontologies in description logic. This new scoring function for ontologies was implemented using the KAON2 reasoner[10] and MIPSes were found through a process incorporating an early algorithm for computing MUSes [16]. A preliminary experimental evaluation was then carried out using three existing ontology datasets. While this evaluation is limited, it is interesting because it represents an experimental evaluation of formula-level inconsistency measures involving real-world data. However, since this implementation is based on description logic, it is not strictly comparable to MIMUS.

In [28], a new abstract and refine technique for computing MUSes was proposed to improve the feasibility of the HYCAM algorithm from [29]. This work was primarily concerned with computing a single MUS but also supported the computation of the exhaustive set of MUSes. In particular, this approach involves arbitrarily splitting a set of clauses into m clause groups (called clusters), then finding the set of GMUSes and using $\mathsf{S}_{\mathsf{I_{MI}}}$ to measure the inconsistency of each clause group. Then, based on these inconsistency values, clauses groups are split or pruned in order to refine the search space. A MUS is found when each clause group in a GMUS consists of a single clause. Essentially, the $\mathsf{S}_{\mathsf{I_{MI}}}$ measure is used as a heuristic to indicate where low-level MUSes might be found w.r.t. GMUSes in a set of arbitrarily constructed clause groups. This is an interesting

[10] http://kaon2.semanticweb.org.

application of a formula-level inconsistency measure since it shows that these measures can improve the efficiency of computing low-level MUSes. In fact, this approach could be easily adapted to find GMUSes in a predefined set of clause groups where these clause groups, rather than individual clauses, could be clustered. Thus, formula-level inconsistency measures could be applied during the actual search for MISes.

In terms of the INCMEASURER tool, a new algorithm was proposed for finding MISes as a variation on an existing algorithm for finding MSSes from [44]. Specifically, the algorithm formulates the task of finding MISes as a breadth-first search problem in a binomial tree. However, in contrast to the state-of-the-art work on finding MUSes, the algorithm does not exploit the relationship between the dual concepts of MSSes (resp. MCSes) and MUSes (resp. MISes). Instead the algorithm computes MISes directly and so, while it benefits from root and leaf pruning (as is the case in [44]), it requires expensive unsatisfiability tests. Also, since it is possible to find locally minimal subsets w.r.t. one branch which are not minimal w.r.t. the set of subsets for the tree, the algorithm requires a post-check for set inclusion to remove these pseudo-minimal subsets.

The implementation of this algorithm in INCMEASURER requires arbitrary formulae to be input in CNF and uses the SAT4J[11] library to determine unsatisfiability. In other words, this implementation finds GMUSes, rather than strictly finding MISes. So in order to integrate this into a real-world arbitrary knowledge base system, CNF conversion and a mapping of formulae to clauses would also be required (as described in Algorithm 1). Furthermore, there has been no experimental evaluation of the performance of INCMEASURER. However, since the SAT4J library is designed for flexibility and ease of use rather than performance, and given that their algorithm does not exploit the state-of-the-art hitting set dualization approach, it is unlikely that INCMEASURER would perform as well as other systems (e.g., MIMUS) which are based on implementations such as CAMUS or HYCAM.

As for the PRISM reasoning platform, arbitrary formulae are automatically converted to equisatisfiable CNF using the Tseitin [68] translation. This allows conversion of arbitrary formulae to CNF with (at most) a linear, rather than exponential, increase in size at the cost of introducing additional variables. Then the SAT4J in-built GMUS implementation is used to find GMUSes in the equisatisfiable CNF which are mapped back to the original formulae. Again, no experimental evaluation of this method for finding MISes, nor of the $\mathsf{S}_{\mathsf{I}_{\mathsf{MI}}}$ measure, was carried out in this work. Aside from CNF translation, it is unlikely that PRISM would outperform MIMUS in terms of computing MISes, since (as with INCMEASURER) PRISM is based on the SAT4J library. In fact the GMUS implementation in SAT4J actually proved unreliable in the recent SAT 2011 Competition,[12] where it produced 40.61% *bad answers* (incorrect results). However, the authors of PRISM have identified this limitation and have suggested incorporating CAMUS or HYCAM in future work.

Each system represents an interesting application of formula-level inconsistency measures, namely: incoherence in description logic ontologies; a heuristic MUS search; inconsistency in requirements specifications; and a reasoning platform based on the $\mathsf{S}_{\mathsf{I}_{\mathsf{MI}}}$ measure, respectively. Of the four examples, INCMEASURER and PRISM are most closely related to MIMUS. However, in terms of performance, their reliance on the SAT4J library for finding GMUSes means that other systems which are based on state-of-the-art implementations such as CAMUS or HYCAM (e.g., MIMUS), are likely to outperform them in terms of finding MISes. On the other hand, given that MIMUS currently incorporates exponential CNF conversion, the Tseitin translation (used by PRISM) for CNF conversion is an interesting optimization and could be included in future versions of MIMUS. Importantly, the main limitation of these two systems as we see it, is the lack of an experimental evaluation. We believe that the approach taken in our evaluation, in terms of generating random arbitrary knowledge bases, represents a suitable and effective means of evaluating these systems.

8. Conclusion

In this paper we have argued that there is a need for an automated approach to inconsistency handling in many real-world knowledge bases. To that end we have suggested the use of formula-level inconsistency measures, based on MISes, as a viable means of achieving this. Through the development and implementation of the MIMUS tool, followed by an extensive evaluation with random arbitrary knowledge bases, we have demonstrated the feasibility of this approach. In particular, MUS algorithms such as CAMUS represent a practical means to accomplish the computationally difficult task of finding MISes. Our evaluation has shown that these algorithms are viable for many large and complex inconsistent knowledge bases. In terms of calculating MIS-orientated formula-level inconsistency measures, our experiments have shown that they are comparatively trivial to compute after the set of MISes has been found.

To summarize the main contributions of this paper, we have:

- established connections between MUSes (from the SAT community) and MISes (from the inconsistency measure community);
- presented an existing MUS generalization approach to finding MISes;
- proposed an alternative optimized MUS transformation approach;
- implemented both approaches to finding MISes (using the state-of-the-art CAMUS MUS algorithm) in a tool called MIMUS;
- selected and compared a representative sample of formula-level inconsistency measures from the literature, including those for stratified knowledge bases;

[11] http://www.sat4j.org.
[12] http://www.cril.univ-artois.fr/SAT11/results.

- implemented this selection of inconsistency measures in MIMUS;
- proposed the use of GENBAL to generate random arbitrary knowledge bases (including stratified knowledge bases) and described their characteristics in terms of inconsistency and MISes; and
- completed an extensive experimental evaluation of MIMUS using these random arbitrary knowledge bases.

The approach taken during this evaluation of MIMUS, in terms of generating random arbitrary propositional knowledge bases (both flat and stratified), provides a useful foundation for evaluating a number of similar applications. These include:

- other inconsistency measures for propositional knowledge bases not included in this work, such as base-level and model-based measures;
- belief merging and revision operators which are still largely theoretical; and
- uncertain knowledge bases where random formulae could, for example, be assigned a random probability value.

The nature of real-world systems means that it can be a challenge to obtain suitable knowledge bases for evaluating these applications. This in itself can limit such an evaluation because of the quantity and variety of experimental data available. While not a comprehensive solution, random knowledge bases on the other hand provide a solid and well supported source of experimental data. The properties of random knowledge bases identified and discussed in this evaluation, such as sharp phase transitions and the explosion of MISes, will have implications for these applications also.

In terms of future work, we plan to investigate the use of an equisatisfiable CNF translation of arbitrary formulae during MIS computation. We intend to implement this in MIMUS to support an experimental evaluation. We also aim to extend this type of evaluation (using randomly generated arbitrary knowledge bases) to other application areas. Evaluating implementations of belief merging operators is particularly suitable since obtaining suitable test data represents a similar challenge as faced when evaluating MIMUS. Finally, having demonstrated the viability of MIMUS, as well as the benefit of these measures in terms of analyzing large inconsistent knowledge bases, we hope to incorporate this tool into a real-world knowledge base system.

Acknowledgements

This work was supported by the EPSRC projects EP/G034303/1, EP/H049606/1 and EP/J012149/1. We would like to thank Jianbing Ma for his helpful comments on the proof for Theorem 1. We would also like to thank the anonymous referees for their insightful and constructive feedback.

References

[1] Salem Benferhat, Didier Dubois, Henri Prade, Argumentative inference in uncertain and inconsistent knowledge bases, in: Proceedings of the 9th International Conference on Uncertainty in Artificial Intelligence, 1993, pp. 411–419.
[2] Nuel Belnap, A useful four-valued logic, in: Modern Uses of Multiple-Valued Logic, 1977, pp. 8–37.
[3] Philippe Besnard, Éric Grégoire, Cédric Piette, Badran Raddaoui, MUS-based generation of arguments and counter-arguments, in: Proceedings of the 2010 IEEE International Conference on Information Reuse & Integration, 2010, pp. 239–244.
[4] Philippe Besnard, Anthony Hunter, Quasi-classical logic: non-trivializable classical reasoning from inconsistent information, in: Symbolic and Quantitative Approaches to Uncertainty, in: LNCS, vol. 946, 1995, pp. 44–51.
[5] Anton Belov, Matti Järvisalo, João Marques-Silva, Formula preprocessing in MUS extraction, in: Proceedings of the 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, 2013, pp. 108–123.
[6] Elazar Birnbaum, Eliezer Lozinskii, Consistent subsets of inconsistent systems: structure and behaviour, J. Exp. Theor. Artif. Intell. 15 (1) (2003) 25–46.
[7] Gerhard Brewka, Preferred subtheories: an extended logical framework for default reasoning a framework for nonmonotonic systems, in: Proceedings of the 11th International Joint Conference on Artificial Intelligence, 1989, pp. 1043–1048.
[8] Renato Bruni, Minimal unsatisfiable subformulae by means of adaptive core search, Discrete Appl. Math. 130 (2) (2003) 85–100.
[9] Steve Barker, Peter Stuckey, Flexible access control policy specification with constraint logic programming, ACM Trans. Inf. Syst. Secur. 6 (4) (2003) 501–546.
[10] James Bailey, Peter Stuckey, Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization, in: Proceedings of the 7th International Conference on Practical Aspects of Declarative Languages, 2005, pp. 174–186.
[11] Claudette Cayrol, Marie-Christine Lagasquie-Schiex, Thomas Schiex, Nonmonotonic reasoning: from complexity to algorithms, Ann. Math. Artif. Intell. 22 (3–4) (1998) 207–236.
[12] Amin Coja-Oghlan, Alan Frieze, Random k-SAT: the limiting probability for satisfiability for moderately growing k, Electron. J. Comb. 15 (1) (2008) N2.
[13] Johannes Dellert, Interactive extraction of minimal unsatisfiable cores enhanced by meta learning, Informatik diplom, Eberhard Karls Universität Tübingen, 2013.
[14] Emanuele Di Rosa, Enrico Giunchiglia, Marco Maratea, Solving satisfiability problems with preferences, Constraints 15 (4) (2010) 485–515.
[15] Johan de Kleer, Brian Williams, Diagnosing multiple faults, Artif. Intell. 32 (1) (1987) 97–130.
[16] Maria Garcia de la Banda, Peter Stuckey, Jeremy Wazny, Finding all minimal unsatisfiable subsets, in: Proceedings of the 5th ACM SIGPLAN International Conference on Principles and Practice of Declaritive Programming, 2003, pp. 32–43.
[17] Niklas Eén, Armin Biere, Effective preprocessing in SAT through variable and clause elimination, in: Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing, 2005, pp. 1–15.
[18] Thomas Eiter, Georg Gottlob, On the complexity of propositional knowledge base revision, updates, and counterfactuals, Artif. Intell. 57 (2–3) (1992) 227–270.
[19] Niklas Eén, Niklas Sörensson, An extensible SAT-solver, in: Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing, 2003, pp. 502–518.

[20] Olivier Fourdrinoy, Éric Grégoire, Bertrand Mazure, Lakhdar Saïs, Eliminating redundant clauses in SAT instances, in: Proceedings of the 4th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 2007, pp. 71–83.
[21] Abraham Flaxman, Algorithms for random 3-SAT, in: Encyclopedia of Algorithms, 2008, pp. 742–744.
[22] John Franco, John Martin, A history of satisfiability, in: Armin Biere, Marijn Heule, Hans van Maaren, Toby Walsh (Eds.), Handbook of Satisfiability, in: Frontiers in Artificial Intelligence and Applications, vol. 185, IOS Press, 2009, pp. 3–76 (Chapter 1).
[23] Ehud Friedgut, Sharp thresholds of graph properties, and the k-sat problem, J. Am. Math. Soc. 12 (4) (1999) 1017–1054.
[24] Alexander Felfernig, Monika Schubert, Christoph Zehentner, An efficient diagnosis algorithm for inconsistent constraint sets, Artif. Intell. Eng. Des. Anal. Manuf. 26 (1) (2011) 53–62.
[25] John Govaerts, Arosha Bandara, Kevin Curran, A formal logic approach to firewall packet filtering analysis and generation, Artif. Intell. Rev. 29 (3–4) (2009) 223–248.
[26] John Grant, Anthony Hunter, Measuring consistency gain and information loss in stepwise inconsistency resolution, in: Proceedings of the 11th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 2011, pp. 362–373.
[27] Éric Grégoire, Bertrand Mazure, Cédric Piette, Local-search extraction of MUSes, Constraints 12 (3) (2007) 325–344.
[28] Éric Grégoire, Bertrand Mazure, Cédric Piette, Does this set of clauses overlap with at least one MUS?, in: Automated Deduction – CADE-22, in: LNCS, vol. 5663, 2009, pp. 100–115.
[29] Éric Grégoire, Bertrand Mazure, Cédric Piette, Using local search to find MSSes and MUSes, Eur. J. Oper. Res. 199 (3) (2009) 640–646.
[30] John Grant, Classifications for inconsistent theories, Notre Dame J. Form. Log. 19 (3) (1978) 435–444.
[31] Anthony Hunter, Sébastien Konieczny, Approaches to measuring inconsistent information, in: Inconsistency Tolerance, in: LNCS, vol. 3300, 2005, pp. 189–234.
[32] Anthony Hunter, Sébastien Konieczny, Measuring inconsistency through minimal inconsistent sets, in: Proceedings of the 11th International Conference on Knowledge Representation, 2008, pp. 358–366.
[33] Anthony Hunter, Sébastien Konieczny, On the measure of conflicts: Shapley inconsistency values, Artif. Intell. 174 (14) (2010) 1007–1026.
[34] Anthony Hunter, Measuring inconsistency in knowledge via quasi-classical models, in: Proceedings of the 18th American National Conference on Artificial Intelligence, 2002, pp. 68–73.
[35] Anthony Hunter, Logical comparison of inconsistent perspectives using scoring functions, Knowl. Inf. Syst. 6 (5) (2004) 1–16.
[36] Said Jabbour, Badran Raddaoui, Measuring inconsistency through minimal proofs, in: Proceedings of the 12th European Conference Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 2013, pp. 290–301.
[37] Ulrich Junker, QUICKXPLAIN: preferred explanations and relaxations for over-constrained problems, in: Proceedings of the 19th National Conference on Artificial Intelligence, 2004, pp. 167–172.
[38] Kevin Knight, Measuring inconsistency, J. Philos. Log. 31 (1) (2002) 77–98.
[39] Sébastien Konieczny, Stéphanie Roussel, A reasoning platform based on the MI Shapley inconsistency value, in: Proceedings of the 12th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 2013, pp. 315–327.
[40] Mark Liffiton, Maher Mneimneh, Inês Lynce, Zaher Andraus, João Marques-Silva, Karem Sakallah, A branch and bound algorithm for extracting smallest minimal unsatisfiable subformulas, Constraints 14 (4) (2008) 415–442.
[41] Mark Liffiton, Michael Moffitt, Martha Pollack, Karem Sakallah, Identifying conflicts in overconstrained temporal problems, in: Proceedings of the 19th International Joint Conference on Artificial Intelligence, 2005, pp. 205–211.
[42] Inês Lynce, João Marques-Silva, On computing minimum unsatisfiable cores, in: Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing, 2004, pp. 305–310.
[43] Mark Liffiton, Karem Sakallah, Algorithms for computing minimal unsatisfiable subsets of constraints, J. Autom. Reason. 40 (1) (2007) 1–33.
[44] Robert Malouf, Maximal consistent subsets, Comput. Linguist. 33 (2) (2007) 153–160.
[45] Kedian Mu, Zhi Jin, Weiru Liu, Didar Zowghi, Bo Wei, Measuring the significance of inconsistency in the viewpoints framework, Sci. Comput. Program. 78 (9) (2013) 1572–1599.
[46] Kedian Mu, Zhi Jin, Ruqian Lu, Weiru Liu, Measuring inconsistency in requirements specifications, in: Proceedings of the 8th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 2005, pp. 440–451.
[47] Kedian Mu, Weiru Liu, Zhi Jin, A general framework for measuring inconsistency through minimal inconsistent sets, Knowl. Inf. Syst. 27 (1) (2010) 85–114.
[48] Kedian Mu, Weiru Liu, Zhi Jin, Measuring the blame of each formula for inconsistent prioritized knowledge bases, J. Log. Comput. 22 (3) (2012) 481–516.
[49] Kevin McAreavey, Weiru Liu, Paul Miller, Kedian Mu, Measuring inconsistency in a network intrusion detection rule set based on snort, Int. J. Semant. Comput. 5 (3) (2011) 281–322.
[50] Kevin McAreavey, Weiru Liu, Paul Miller, Chris Meenan, Tools for finding inconsistencies in real-world logic-based systems, in: Proceedings of the 6th Starting AI Researchers' Symposium, 2012, pp. 192–203.
[51] Yue Ma, Guilin Qi, Pascal Hitzler, Zuoquan Lin, Measuring inconsistency for description logics based on paraconsistent semantics, in: Proceedings of the 9th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 2007, pp. 30–41.
[52] Muhammad Rafiq, Peter Stuckey, A stochastic non-CNF SAT solver, in: Proceedings of the 9th Pacific Rim International Conference on Artificial Intelligence, 2006, pp. 120–129.
[53] Bertrand Mazure, Lakhdar Saïs, Éric Grégoire, A powerful heuristic to locate inconsistent kernels in knowledge-based systems, in: Proceedings of the International Conference on Information Processing and Management of Uncertainty in Knowledge-Base Systems, 1996, pp. 1265–1269.
[54] João Marques-Silva, Federico Heras, Mikoláš Janota, Alessandro Previti, Anton Belov, On computing minimal correction subsets, in: Proceedings of the 23rd International Joint Conference on Artificial Intelligence, 2013, pp. 615–622.
[55] Alexander Nöhrer, Armin Biere, Alexander Egyed, Managing SAT inconsistencies with HUMUS, in: Proceedings of the 6th International Workshop on Variability Modeling of Software-Intensive Systems, 2012, pp. 83–91.
[56] Alexander Nadel, Vadim Ryvchin, Ofer Strichman, Efficient MUS extraction with resolution, in: Proceedings of the 13th Conference on Formal Methods in Computer Aided Design, 2013, pp. 197–200.
[57] Juan Navarro, Andrei Voronkov, Generation of hard non-clausal random satisfiability problems, in: Proceedings of the 20th National Conference on Artificial Intelligence, 2005, pp. 436–442.
[58] Yoonna Oh, Maher Mneimneh, Zaher Andraus, Karem Sakallah, Igor Markov, AMUSE: a minimally-unsatisfiable subformula extractor, in: Proceedings of the 41st Annual Conference on Design Automation, 2004, pp. 518–523.
[59] Graham Priest, Minimally inconsistent LP, Stud. Log. 50 (2) (1991) 321–331.
[60] Christos Papadimitriou, David Wolfe, The complexity of facets resolved, J. Comput. Syst. Sci. 37 (1) (1988) 2–13.
[61] Guilin Qi, Anthony Hunter, Measuring incoherence in description logic-based ontologies, in: The Semantic Web, in: LNCS, vol. 4825, 2007, pp. 381–394.
[62] Guilin Qi, Weiru Liu, David Bell, Measuring conflict and agreement between two prioritized belief bases, in: Proceedings of the 19th International Joint Conference on Artificial Intelligence, 2005, pp. 552–557.
[63] Raymond Reiter, A theory of diagnosis from first principles, Artif. Intell. 32 (1) (1987) 57–95.

[64] Martin Roesch, Snort-lightweight intrusion detection for networks, in: Proceedings of the 13th USENIX Systems Administration Conference, 1999, pp. 229–238.
[65] Carsten Sinz, Andreas Kaiser, Wolfgang Küchlin, Formal methods for the validation of automotive product configuration data, Artif. Intell. Eng. Des. Anal. Manuf. 17 (1) (2003) 75–97.
[66] Gina Tjhai, Maria Papadaki, Steven Furnell, Nathan Clarke, Investigating the problem of IDS false alarms: an experimental study using snort, in: Proceedings of the IFIP TC 11 23rd International Information Security Conference, 2008, pp. 253–267.
[67] Gina Tjhai, Maria Papadaki, Steven Furnell, Nathan Clarke, The problem of false alarms: evaluation with snort and DARPA 1999 dataset, in: Proceedings of the 5th International Conference: Trust, Privacy and Security in Digital Business, 2008, pp. 139–150.
[68] Grigorii Samuilovich Tseitin, On the complexity of derivation in propositional calculus, in: Studies in Mathematics and Mathematical Logics, vol. 2, 1968, pp. 115–125.
[69] Guohui Xiao, Yue Ma, Inconsistency measurement based on variables in minimal unsatisfiable subsets, in: Proceedings of the 20th European Conference on Artificial Intelligence, 2012, pp. 864–869.
[70] Lintao Zhang, Sharad Mali, Extracting small unsatisfiable cores from unsatisfiable boolean formula, in: Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing, 2003.