Algorithms for Inconsistency Measurement using Answer Set Programming

Isabelle Kuhlmann and Matthias Thimm
University of Koblenz-Landau
Universitätsstraße 1
56072 Koblenz, Germany
{iskuhlmann, thimm}@uni-koblenz.de

Abstract

We present algorithms based on answer set programming (ASP) encodings for solving the problem of determining inconsistency degrees in propositional knowledge bases. For that, we consider the contension inconsistency measure, the forgetting-based inconsistency measure, and the hitting set inconsistency measure. Our experimental evaluation shows that all three algorithms significantly surpass the state of the art.

1 Introduction

A major challenge in symbolic approaches to AI is the handling of *inconsistent* information. The field of *Inconsistency Measurement*—see the seminal work (Grant 1978) and the book (Grant and Martinez 2018)—provides an *analytical* perspective on the issue of inconsistency in formal knowledge representation formalisms. Its aim is to quantitatively assess the *severity* of inconsistency in order to both guide automatic reasoning mechanisms and to help human modellers in identifying issues and compare different alternative formalizations. For example, inconsistency measures have been used to estimate reliability of agents in multi-agent systems (Cholvy, Perrussel, and Thevenin 2017), to analyze inconsistencies in news reports (Hunter 2006), to support collaborative software requirements specifications (Martinez, Arias, and Vilas 2004), to allow for inconsistency-tolerant reasoning in probabilistic logic (Potyka and Thimm 2017), and to monitor and maintain quality in database settings (Bertossi 2018).

Previous research on the computational complexity of inconsistency measures (Thimm and Wallner 2019) showed that evaluating them is computationally hard in general. However, as the list of application areas above shows, there is a need to practical working solutions. In this paper, we address this need by leveraging existing problem solving paradigms to develop effective algorithmic solutions to some prominent inconsistency measures. More precisely, we consider the contension inconsistency measure (Grant and Hunter 2011), the hitting set inconsistency measure (Thimm 2016), and the forgetting-based inconsistency measure (Besnard 2016) (we will give their formal definitions in Section 2). Natural decision problems pertaining to those measures are hard for the first level of the polynomial hierarchy, but still easier compared to many other measures (Thimm and Wallner 2019). We therefore believe that these measures are most likely suitable for real-world applications, due to the existence of general problem solving paradigms able to solve problems of this complexity in comparably short time. Here, we use Answer Set Programming (ASP) (Gelfond and Lifschitz 1991; Gelfond and Leone 2002; Gebser et al. 2012) for this purpose, a non-monotonic logic programming language that has been proven successful to solve problems in many other areas such as formal argumentation (Dvorák et al. 2020) and automated planning (Erdem et al. 2013), see also (Erdem, Gelfond, and Leone 2016). We selected the contension, forgetting-based, and hitting set inconsistency measure, as they are conceptually more similar to each other than to the other measures which Thimm and Wallner identified to be on the first level of the polynomial hierarchy (Thimm and Wallner 2019). Part of our ongoing research is, however, to investigate the other measures on this level, for instance the distance-based inconsistency measures proposed by Grant and Hunter (2017).

In summary, the contributions of this paper are as follows:

1. We introduce algorithms based on answer set encodings for determining the inconsistency value wrt. the contension inconsistency measure, the forgetting-based inconsistency measure, and the hitting set inconsistency measure (Section 3).
2. We present our findings of an experimental evaluation of these algorithms, where we compare their runtime with the runtime of existing baseline implementations (Section 4).

In Section 2 we give an overview on the necessary preliminaries, in particular about inconsistency measurement and answer set programming. We conclude with a discussion of our findings and possible future work in Section 5.

A short paper introducing a preliminary version of the encoding for the contension inconsistency measure has been published before (Kuhlmann and Thimm 2020). In this paper, we present an improved encoding of that measure and novel encodings for the other two measures.

2 Preliminaries

Let At be a fixed set of propositional atoms and let $\mathcal{L}(\mathsf{At})$ be the corresponding propositional language constructed with the usual connectives $\wedge$ (*conjunction*), $\vee$ (*disjunction*), and $\neg$ (*negation*). A *knowledge base* $\mathcal{K}$ is a finite set of formulas $\mathcal{K} \subseteq \mathcal{L}(\mathsf{At})$. Moreover, we define $\mathbb{K}$ as the set of all knowledge bases. Further, $\mathsf{At}(X)$ denotes the set of atoms appearing in a formula (or set of formulas) X.

Interpretations give semantics to propositional languages. An interpretation i on At is a function $i : \mathsf{At} \to \{\textit{true}, \textit{false}\}$. We define $\mathsf{Int}(\mathsf{At})$ as the set of all interpretations wrt. At. An interpretation i *satisfies* an atom $x \in \mathsf{At}$, denoted $i \models x$, iff $i(x) = \textit{true}$. This concept is extended to formulas in the usual manner. If an interpretation i satisfies a formula ϕ, it is called a *model* of ϕ, respectively.

Let $\Phi \subseteq \mathcal{L}(\mathsf{At})$ be a set of formulas. We define $i \models \Phi$ iff $i \models \phi$ for all $\phi \in \Phi$. A formula (or set of formulas) X_1 *entails* another formula (or set of formulas) X_2, indicated as $X_1 \models X_2$ if $i \models X_1$ implies $i \models X_2$ for every interpretation i. If there exists no interpretation i with $i \models X$, we denote this as $X \models \perp$, and X is called *inconsistent*.

2.1 Inconsistency Measurement

In general, an *inconsistency measure* $\mathcal{I}$ is a function $\mathcal{I} : \mathbb{K} \to \mathbb{R}^{\infty}_{\geq 0}$ (Thimm 2019). The intuition behind such inconsistency measures is that a higher value indicates a more severe inconsistency than a lower one. The minimal value 0 is supposed to model the absence of inconsistency, i.e., consistency.

The Contension Inconsistency Measure The *contension inconsistency measure* (Grant and Hunter 2011) is based on Priest's three-valued logic (Priest 1979). In addition to *true* and *false*, this logic introduces a third truth value denoted *both (true and false)* or *paradoxical*. In the remainder of this work we will also refer to these truth values as T, F, and B, respectively. The truth tables of this logic are presented in Table 1. A corresponding three-valued interpretation i^3 is a function that assigns one of the three truth values to each atom in a knowledge base $\mathcal{K}$:

$$i^3 : \mathsf{At}(\mathcal{K}) \mapsto \{\textit{true}, \textit{both}, \textit{false}\}$$

Such an interpretation is called a *model* if each formula $\phi \in \mathcal{K}$ evaluates to either *true* or *both*. The set of all models wrt. $\mathcal{K}$ is defined as

$$\mathsf{Models}(\mathcal{K}) = \{i^3 \mid \forall \phi \in \mathcal{K}, i^3(\phi) = T \text{ or } i^3(\phi) = B\}$$

Further, we can divide the domain of an interpretation i^3 into two sets. One contains those atoms that are assigned a classical truth value (T, F), the other one contains those that are assigned truth value B, i. e., those which are involved in a conflict. The latter is defined as

$$\mathsf{Conflictbase}(i^3) = \{x \in \mathsf{At}(\mathcal{K}) \mid i^3(x) = B\}.$$

Finally, we can define the contension inconsistency measure $\mathcal{I}_c$ wrt. a knowledge base $\mathcal{K}$ as follows:

$$\mathcal{I}_c(\mathcal{K}) = \min\{|\mathsf{Conflictbase}(i^3)| \mid i^3 \in \mathsf{Models}(\mathcal{K})\}.$$

Hence, $\mathcal{I}_c$ describes the minimum number of atoms that are assigned truth value B wrt. a knowledge base $\mathcal{K}$.

x	y	$x \wedge y$	$x \vee y$
T	T	T	T
T	B	B	T
T	F	F	T
B	T	B	T
B	B	B	B
B	F	F	B
F	T	F	T
F	B	F	B
F	F	F	F

x	$\neg x$
T	F
B	B
F	T

Table 1: Truth tables for Priest's propositional three-valued logic.

Example 1 *Consider the following inconsistent knowledge base:*

$$\mathcal{K}_1 = \{a \wedge b, \neg a \wedge c, a, \neg a, \neg b\}$$

Let i_1^3 be the interpretation that assigns T to c, and B to a and b. Thus, $\mathsf{Conflictbase}(i_1^3) = \{a, b\}$. *Because each formula in $\mathcal{K}_1$ evaluates to either T or B given i_1^3, then i_1^3 is also a model of $\mathcal{K}_1$. It is easy to see that a and b must be assigned B in order to make the knowledge base satisfiable, and that no lower number of atoms being assigned B could make $\mathcal{K}_1$ satisfiable. Hence, we get $\mathcal{I}_c(\mathcal{K}_1) = 2$.*

The Forgetting-Based Inconsistency Measures The intuition behind the *forgetting-based inconsistency measure* (Besnard 2016) is to count how many atom occurrences in a knowledge base $\mathcal{K}$ have to be "forgotten" in order to recover consistency in $\mathcal{K}$, where "forgetting" is interpreted as replacing the atom occurrence with either $\top$ or $\perp$. To illustrate this, we first label each atom occurrence according to its position in $\mathcal{K}$. For instance, we can give label "1" to the first occurrence of an atom a, label "2" to the second occurrence, and so forth.

Example 2 *Recall knowledge base $\mathcal{K}_1$ given in Example 1. Assigning labels as described above yields the knowledge base*

$$\mathcal{K}_1^l = \{a^1 \wedge b^1, \neg a^2 \wedge c^1, a^3, \neg a^4, \neg b^2\}.$$

For a formula ϕ, let $\phi[x_1^{n_1} \to \psi_1, \ldots, x_p^{n_k} \to \psi_k]$ denote the formula ϕ' where the atoms $x_1, \ldots, x_p$ with labels $n_1, \ldots, n_k$ are replaced by $\psi_1, \ldots, \psi_k$.

Example 3 *Let* $\phi_1 := (a^1 \wedge b^1) \vee (\neg a^2 \wedge b^2)$.

$$\phi_1[a^2 \to \top, b^1 \to \perp] = (a \wedge \perp) \vee (\neg \top \wedge b)$$

Consequently, we can define the forgetting-based inconsistency measure as

$$\mathcal{I}_f(\mathcal{K}) = \min\{k \mid (\bigwedge \mathcal{K})[x_1^{n_1} \to \psi_1, \ldots, x_p^{n_k} \to \psi_k] \not\models \perp, \psi_1, \ldots, \psi_k \in \{\top, \perp\}\}$$

for all $\mathcal{K} \in \mathbb{K}$ with $\{x_1, \ldots, x_p\} \in \mathsf{At}(\mathcal{K})$ and $n_1, \ldots, n_k$ being the corresponding labels.

Example 4 *With regard to the labeled knowledge base $\mathcal{K}_1^l$, given in Example 2, we could replace a^1, a^3, and b^1 by $\top$,* *i.e., we could forget a^1, a^3, and b^1, in order to restore consistency. Although there are other options to recover consistency, e. g., by forgetting a^2, a^4, and b^2, one can clearly see that it is not possible to obtain consistency by forgetting fewer than 3 atom occurrences. Hence, $\mathcal{I}_f(\mathcal{K}_1) = 3$.*

The Hitting Set Inconsistency Measure A subset $H \subseteq \mathsf{Int}(\mathsf{At})$ is a *hitting set* of a knowledge base $\mathcal{K}$ if for every formula $\phi \in \mathcal{K}$ there is an interpretation $\omega \in H$ with $\omega \models \phi$. Thus, there only exists a hitting set for $\mathcal{K}$ iff there is no $\phi \in \mathcal{K}$ with $\phi \models \perp$, i.e., no formula $\phi \in \mathcal{K}$ is contradictory. Moreover, if there exists a hitting set H wrt. $\mathcal{K}$, and $|H| = 1$, the only element in H is a model of $\mathcal{K}$. Thus, in this case, $\mathcal{K}$ is consistent. Based on the preceding definitions, the hitting set inconsistency measure $\mathcal{I}_h(\mathcal{K})$ (Thimm 2016) is defined as the minimum number of elements in the hitting set, subtracted by 1. If there exists no hitting set wrt. $\mathcal{K}$, then $\mathcal{I}_h(\mathcal{K}) = \infty$. Formally,

$$\mathcal{I}_h(\mathcal{K}) = \min\{|H| \mid H \text{ is a hitting set of } \mathcal{K}\} - 1,$$

with $\min \emptyset = \infty$ for all $\mathcal{K} \in \mathbb{K} \backslash \{\emptyset\}$. Further, $\mathcal{I}_h(\emptyset) = 0$.

Example 5 *Consider again $\mathcal{K}_1$ as defined in Example 1. As none of the formulas in $\mathcal{K}_1$ is contradictory, there must exist a hitting set. Also, as the knowledge base is obviously inconsistent, we need at least two interpretations to compile a hitting set. Let interpretation i_1 assign T to the formulas a, b, and c, and let interpretation i_2 assign F to a and b, and T to c. Each formula $\phi \in \mathcal{K}_1$ is satisfied by one of these two interpretations. Hence, $\mathcal{I}_h(\mathcal{K}_1) = 2 - 1 = 1$.*

2.2 Answer Set Programming

Answer set programming (ASP) (Gebser et al. 2012; Lifschitz 2008; Brewka, Eiter, and Truszczynski 2011) is a declarative problem solving approach targeted at difficult search problems. ASP incorporates ideas of logic programming and Reiter's default logic (Reiter 1980). A problem is modeled as an *extended logic program* which consists of a set of *rules*. An ASP rule is of the form

$$r = H \leftarrow A_1, \ldots, A_n, \texttt{not}\, B_1, \ldots, \texttt{not}\, B_m. \qquad (1)$$

where H, A_j with $j \in \{1, \ldots, n\}$, and B_k with $k \in \{1, \ldots, m\}$ are classical literals. ASP rules consist of a *head* and a *body*, both of which can be empty. We denote the sets of literals contained in the head and body of a rule r as $\mathsf{head}(r)$, and $\mathsf{body}(r)$, respectively. A rule with an empty body is called a *fact*, a rule with an empty head is referred to as a *constraint*. In (1), $\mathsf{head}(r) = \{H\}$, and $\mathsf{body}(r) = \{A_1, \ldots, A_n, B_1, \ldots, B_m\}$. An extended logic program is *positive* if it does not contain any instance of `not`. Moreover, a set of literals L is called *closed* under a positive program P if and only if for any rule $r \in P$, $\mathsf{head}(r) \in L$ whenever $\mathsf{body}(r) \subseteq L$. The set L is consistent if it does not contain both A and $\neg A$ for some literal A. The smallest of such sets wrt. a positive program P, which is always uniquely defined, is referred to as $\mathsf{Cn}(P)$. With regard to an arbitrary program P, a set L is an *answer set* of P if $L = \mathsf{Cn}(P^L)$ and L is consistent, with

$$\begin{aligned} P^L = \{H \leftarrow A_1, \ldots, A_n \mid \\ & H \leftarrow A_1, \ldots, A_n, \texttt{not}\, B_1, \ldots, \texttt{not}\, B_m. \in P, \\ & \{B_1, \ldots, B_m\} \cap L = \emptyset\} \end{aligned}$$

The head of an ASP rule is not necessarily comprised of only one literal. Some ASP dialects allow for more complex structures, such as *cardinality constraints*, which can be used as both body elements and heads. A cardinality constraint with lower bound l and upper bound u is defined as

$$l\{A_1, \ldots, A_n, \texttt{not}\, B_1, \ldots, \texttt{not}\, B_m\}u.$$

This can be interpreted as follows: if at least l and at most u of the literals $A_1, \ldots, A_n, B_1, \ldots, B_m$ are included in an answer set, a cardinality rule is satisfied by this answer set.

ASP additionally offers the option to express cost functions involving minimization and/or maximization in order to solve optimization problems (Gebser et al. 2012). Here, we only need optimisation statements of the form

$$minimize\{\ell_1, \ldots, \ell_n\}$$

which instruct the ASP solver to include only a minimal number of the literals $\ell_1, \ldots, \ell_n$ in any answer set.

3 Measuring Inconsistency Using ASP

The proposed algorithms involve the development of ASP encodings for each one of the three previously described inconsistency measures. Although the specifics of each inconsistency measure have to be considered individually, there are some aspects that all ASP-based algorithms we propose have in common. To begin with, each atom is supposed to be assigned a unique truth value.

Example 6 *In classical propositional logic, we could model that an atom x is supposed to be assigned either T or F by introducing two corresponding ASP atoms e_{x_T} and e_{x_F}. An atom is true, if it is not false, and vice versa. The respective ASP rules can be defined as follows:*

$$e_{x_T} \leftarrow \texttt{not}\, e_{x_F}.$$
$$e_{x_F} \leftarrow \texttt{not}\, e_{x_T}.$$

In the remainder of this paper, we refer to this part as *unique atom evaluation*. Note that ASP atoms e_{ϕ_T}, e_{ϕ_F} representing the evaluation of a formula ϕ can be created in the same manner as shown above wrt. propositional atoms.

Moreover, within the encoding, each formula ϕ must be satisfied, i.e., no formula should evaluate to F. This is achieved through *integrity constraints* of the form

$$\leftarrow e_{\phi_F}.$$

for every $\phi \in \mathcal{K}$. Another element that is common in all three encodings is that the relations between elements within a formula have to be encoded. More precisely, encodings for the connectors $\wedge$, $\vee$, and $\neg$ have to be created.

Example 7 *The evaluation of a conjunction of two propositional formulas ϕ, ψ can be modeled in ASP by encoding that it is only true if both ϕ and ψ are true, and false otherwise:*

$$e_{(\phi\wedge\psi)_T} \leftarrow e_{\phi_T}, e_{\psi_T}.$$
$$e_{(\phi\wedge\psi)_F} \leftarrow \texttt{not}\, e_{(\phi\wedge\psi)_T}.$$

In the following, we will refer to this part as *connector encodings*.

Since the possible inconsistency values regarding all three considered measures are natural numbers from a well-defined interval (Thimm and Wallner 2019), we can make use of a minimization statement to compute the desired inconsistency value.

Example 8 *Let our aim be to minimize the number of atoms $x \in \mathsf{At}(\mathcal{K})$ that are assigned truth value B in three-valued logic, wrt. a knowledge base $\mathcal{K}$. Let the ASP atom e_{x_B} encode the assignment of B to atom x. The corresponding minimize statement can be expressed as*

$$minimize\{e_{x^1_B}, \dots, e_{x^n_B}\}.$$

with $\{x^1, \dots, x^n\} = \mathsf{At}(\mathcal{K})$

The subsequent sections describe the specific ASP encodings for $\mathcal{I}_c$, $\mathcal{I}_h$, and $\mathcal{I}_f$.

3.1 The Contension Inconsistency Measure

With regard to the contension inconsistency measure $\mathcal{I}_c$, we can construct an extended logic program $P_c(\mathcal{K})$ to compute $\mathcal{I}_c(\mathcal{K})$ wrt. a knowledge base $\mathcal{K}$ as follows.

1. We first include rules that guess a three-valued interpretation. For that, we need to ensure unique atom evaluation for each $x \in \mathsf{At}(\mathcal{K})$ wrt. Priest's three-valued logic. Thus, an atom is *true* if it is neither *both* nor *false*. The other two cases follow analogously:
$$e_{x_T} \leftarrow \texttt{not}\, e_{x_B}, \texttt{not}\, e_{x_F}.$$
$$e_{x_B} \leftarrow \texttt{not}\, e_{x_T}, \texttt{not}\, e_{x_F}.$$
$$e_{x_F} \leftarrow \texttt{not}\, e_{x_B}, \texttt{not}\, e_{x_T}.$$
2. The connector encodings for each (sub)formula in $\mathcal{K}$ follow from the truth tables given in Table 1. For instance, a conjunction is only *true* if both conjuncts are *true*. It is *false*, if at least one of its conjuncts is *false*, and it is *both* if it is neither *true* nor *false*. The rules for disjunction and negation are created in the same fashion.
$$\begin{aligned} \phi \wedge \psi \mapsto \quad & e_{(\phi\wedge\psi)_T} \leftarrow e_{\phi_T}, e_{\psi_T}. \\ & e_{(\phi\wedge\psi)_F} \leftarrow e_{\phi_F}. \\ & e_{(\phi\wedge\psi)_F} \leftarrow e_{\psi_F}. \\ & e_{(\phi\wedge\psi)_B} \leftarrow \texttt{not}\, e_{(\phi\wedge\psi)_F}, \texttt{not}\, e_{(\phi\wedge\psi)_T}. \end{aligned}$$
$$\begin{aligned} \phi \vee \psi \mapsto \quad & e_{(\phi\vee\psi)_F} \leftarrow e_{\phi_F}, e_{\psi_F}. \\ & e_{(\phi\vee\psi)_T} \leftarrow e_{\phi_T}. \\ & e_{(\phi\vee\psi)_T} \leftarrow e_{\psi_T}. \\ & e_{(\phi\vee\psi)_B} \leftarrow \texttt{not}\, e_{(\phi\vee\psi)_F}, \texttt{not}\, e_{(\phi\vee\psi)_T}. \end{aligned}$$
$$\begin{aligned} \neg\phi \mapsto \quad & e_{(\neg\phi)_B} \leftarrow e_{\phi_B}. \\ & e_{(\neg\phi)_T} \leftarrow e_{\phi_F}. \\ & e_{(\neg\phi)_F} \leftarrow e_{\phi_T}. \end{aligned}$$
3. Every formula $\phi \in \mathcal{K}$ must be evaluated to *true* or *both* in three-valued logic, i.e., it must not be evaluated to *false*. We therefore add an integrity constraint for each formula:
$$\leftarrow e_{\phi_F}.$$
4. Finally, we want to minimize the number of atoms in $\mathcal{K}$ that are assigned the truth value B. Hence, we add the following minimize statement:
$$minimize\{e_{x^1_B}, \dots, e_{x^n_B}\}.$$

Now $P_c(\mathcal{K})$ is the union of all rules defined in 1–4. Further, let i^3_M be the three-valued interpretation represented by an answer set M of $P_c(\mathcal{K})$.

Theorem 1 *Let M be an optimal answer set of $P_c(\mathcal{K})$. Then $|i^3_M(B)^{-1}| = \mathcal{I}_c(\mathcal{K})$*[1].

The proof of the above theorem as well as further technical results are omitted due to space restrictions, but can be found in the appendix[2].

3.2 The Forgetting-Based Inconsistency Measure

The forgetting-based inconsistency measure $\mathcal{I}_f(\mathcal{K})$ is determined by the number of atom occurrences that need to be "forgotten" in order to make the knowledge base $\mathcal{K}$ consistent. An extended logic program $P_f(\mathcal{K})$ which computes $\mathcal{I}_f(\mathcal{K})$ wrt. a knowledge base $\mathcal{K}$ can be constructed as described below.

1. We first include rules that guess a model for the knowledge base after forgetting operations took place, in order to ensure that the knowledge base is consistent. Although individual atom *occurrences* may be replaced by $\top$ or $\bot$, an atom must be either *true* or *false* in that interpretation. Thus, for every $x \in \mathsf{At}(\mathcal{K})$:
$$e_{x_T} \leftarrow \texttt{not}\, e_{x_F}.$$
$$e_{x_F} \leftarrow \texttt{not}\, e_{x_T}.$$
2. We need to ensure that each atom occurrence is evaluated uniquely. This means that an atom occurrence x^n can either be *true*, *false*, or forgotten, i.e., replaced by either $\top$ or $\bot$. If an atom occurrence x^n is supposed to be replaced by $\top$ or $\bot$, we represent this using the ASP atoms $e_{x^n_{\text{forget}_\top}}$ or $e_{x^n_{\text{forget}_\bot}}$, respectively:
$$e_{x^n_{\text{forget}_\top}} \leftarrow \texttt{not}\, e_{x^n_T}, \texttt{not}\, e_{x^n_F}, \texttt{not}\, e_{x^n_{\text{forget}_\bot}}.$$
$$e_{x^n_{\text{forget}_\bot}} \leftarrow \texttt{not}\, e_{x^n_T}, \texttt{not}\, e_{x^n_F}, \texttt{not}\, e_{x^n_{\text{forget}_\top}}.$$
We also need to ensure that an individual atom occurrence is only set to *true* or *false* if the atom as a whole is evaluated to *true* or *false*, respectively.
$$e_{x^n_T} \leftarrow e_{x_T}, \texttt{not}\, e_{x^n_F}, \texttt{not}\, e_{x^n_{\text{forget}_\top}}, \texttt{not}\, e_{x^n_{\text{forget}_\bot}}.$$
$$e_{x^n_F} \leftarrow e_{x_F}, \texttt{not}\, e_{x^n_T}, \texttt{not}\, e_{x^n_{\text{forget}_\top}}, \texttt{not}\, e_{x^n_{\text{forget}_\bot}}.$$
3. The connector encodings for all (sub)formulas ϕ, ψ in $\mathcal{K}$

[1]For any function $f : X \mapsto Y$ and $y \in Y$ we define $f^{-1}(y) = \{x \in X \mid f(x) = y\}$

[2]http://mthimm.de/misc/nmr21_ikmt.pdf

simply model propositional entailment:

$$\phi \wedge \psi \mapsto \quad e_{(\phi\wedge\psi)_T} \leftarrow e_{\phi_T}, e_{\psi_T}.$$
$$e_{(\phi\wedge\psi)_F} \leftarrow \texttt{not}\, e_{(\phi\wedge\psi)_T}.$$

$$\phi \vee \psi \mapsto \quad e_{(\phi\vee\psi)_F} \leftarrow e_{\phi_F}, e_{\psi_F}.$$
$$e_{(\phi\vee\psi)_T} \leftarrow \texttt{not}\, e_{(\phi\vee\psi)_F}.$$

$$\neg\phi \mapsto \quad e_{(\neg\phi)_T} \leftarrow e_{\phi_F}.$$
$$e_{(\neg\phi)_F} \leftarrow \texttt{not}\, e_{(\neg\phi)_T}.$$

4. If a (sub)formula ϕ is actually an atom occurrence x^n, it can either be set to *true* or *false*, or forgotten:

$$e_{\phi_T} \leftarrow e_{x^n_T}.$$
$$e_{\phi_T} \leftarrow e_{x^n_{\text{forget}_\top}}.$$
$$e_{\phi_F} \leftarrow e_{x^n_F}.$$
$$e_{\phi_F} \leftarrow e_{x^n_{\text{forget}_\bot}}.$$

5. All formulas $\phi \in \mathcal{K}$ must evaluate to *true* after the forgetting operation is applied. Hence, we add the following integrity constraint for each $\phi \in \mathcal{K}$:

$$\leftarrow e_{\phi_F}.$$

6. Lastly, we minimize the number of atom occurrences which are forgotten:

$$minimize\{e_{x^n_{\text{forget}_\top}}, e_{x^n_{\text{forget}_\bot}}, \ldots,$$
$$e_{y^n_{\text{forget}_\top}}, e_{y^n_{\text{forget}_\bot}}, \ldots\}.$$

Note that the rules described in 2. ensure that no atom occurrence is simultaneously replaced by $\top$ and $\bot$.

The union of all rules defined above (in 1–6) constitute the extended logic program $P_f(\mathcal{K})$. We denote the set of atom occurrences that are replaced by $\top$ wrt. a knowledge base $\mathcal{K}$ as T_M, and the set of atom occurrences that are replaced by $\bot$ as F_M. With M being an answer set of $P_f(\mathcal{K})$ we define

$$T_M = \{x^n \mid x \in \mathsf{At}(\mathcal{K}), e_{x^n_{\text{forget}_\top}} \in M\},$$
$$F_M = \{x^n \mid x \in \mathsf{At}(\mathcal{K}), e_{x^n_{\text{forget}_\bot}} \in M\}.$$

Theorem 2 *Let M be an optimal answer set of $P_f(\mathcal{K})$. Then $|T_M| + |F_M| = \mathcal{I}_f(\mathcal{K})$.*

3.3 The Hitting Set Inconsistency Measure

The hitting set inconsistency measure $\mathcal{I}_h(\mathcal{K})$ is defined by the size of the minimal hitting set wrt. a knowledge base $\mathcal{K}$, subtracted by 1. The maximal size of such a hitting set is determined by the number of formulas in $\mathcal{K}$. In the following, we denote the number of formulas in $\mathcal{K}$ as N. Further, i_n refers to the n-th interpretation out of the N possible interpretations we need to consider, assuming that the interpretations have an arbitrary, but fixed order. An interpretation i_n is represented as ω_n in our ASP encoding. Note that the notation ω_n does not only appear as an ASP atom on its own, but also serves the purpose of a label linking the representations of formulas and atoms to specific interpretations. For example, the ASP atom e_{ϕ_T,ω_n} represents the formula ϕ being evaluated to T under the interpretation i_n. We construct an extended logic program $P_h(\mathcal{K})$ which computes $\mathcal{I}_h(\mathcal{K})$ as follows.

1. We first include rules that guess the N interpretations, some of those may be used in the final hitting set. We need to ensure unique atom evaluation wrt. each atom $x \in \mathsf{At}(\mathcal{K})$, as we did with the previous two encodings. However, this time we need to take each possible interpretation into account as well, because an atom may be assigned the truth value T wrt. one interpretation in the hitting set, but F wrt. another one. Thus, for each atom $x \in \mathsf{At}(\mathcal{K})$ and each interpretation i_n, $n \in \{1, \ldots, N\}$, we define:

$$e_{x_T,\omega_n} \leftarrow \texttt{not}\, e_{x_F,\omega_n}.$$
$$e_{x_F,\omega_n} \leftarrow \texttt{not}\, e_{x_T,\omega_n}.$$

2. We ensure that at least one ASP atom ω_n representing an interpretation is contained in the answer set by constructing the following cardinality constraint:

$$1\{\omega_1, \ldots, \omega_N\}N.$$

3. The connector encodings for each (sub)formula in $\mathcal{K}$ follow classical propositional entailment. Again, each rule has to be created with regard to each possible interpretation:

$$\phi \wedge \psi \mapsto \quad e_{(\phi\wedge\psi)_T,\omega_n} \leftarrow e_{\phi_T,\omega_n}, e_{\psi_T,\omega_n}.$$
$$e_{(\phi\wedge\psi)_F,\omega_n} \leftarrow \texttt{not}\, e_{(\phi\wedge\psi)_T,\omega_n}.$$

$$\phi \vee \psi \mapsto \quad e_{(\phi\vee\psi)_F,\omega_n} \leftarrow e_{\phi_F,\omega_n}, e_{\psi_F,\omega_n}.$$
$$e_{(\phi\vee\psi)_T,\omega_n} \leftarrow \texttt{not}\, e_{(\phi\vee\psi)_F,\omega_n}.$$

$$\neg\phi \mapsto \quad e_{(\neg\phi)_T,\omega_n} \leftarrow e_{\phi_F,\omega_n}.$$
$$e_{(\neg\phi)_F,\omega_n} \leftarrow \texttt{not}\, e_{(\neg\phi)_T,\omega_n}.$$

4. In order to meet the definition of a hitting set, we need to ensure that each formula $\phi \in \mathcal{K}$ is satisfied wrt. at least one interpretation:

$$e_{\phi_T} \leftarrow e_{\phi_T,\omega_n}, \omega_n.$$
$$e_{\phi_F} \leftarrow \texttt{not}\, e_{\phi_T}.$$

5. Again, we add an integrity constraint for each formula $\phi \in \mathcal{K}$:

$$\leftarrow e_{\phi_F}.$$

6. We minimize the number of interpretations that are required to satisfy each formula in the given knowledge base using the following minimize statement:

$$minimize\{\omega_1, \ldots, \omega_N\}.$$

As opposed to the minimize statements of the other two encodings, the minimal value is not 0, but 1. This is because we minimize the number of interpretations required

Data-set	Signature size	Formulas per knowl. base	Atoms per formula (mean)	Atoms per formula (max)	Timeouts $\mathcal{I}_c$ naive	Timeouts $\mathcal{I}_c$ ASP	Timeouts $\mathcal{I}_h$ naive	Timeouts $\mathcal{I}_h$ ASP	Timeouts $\mathcal{I}_f$ naive	Timeouts $\mathcal{I}_f$ ASP
A	3	5–15	2.22	6	0	0	0	0	116	0
A	5	15–25	3.10	11	0	0	107	0	200	0
A	10	15–25	3.14	10	0	0	74	0	200	0
A	15	15–25	3.11	11	25	0	108	0	200	0
A	15	25–50	3.11	11	195	0	179	0	200	0
A	20	25–50	3.12	11	199	0	198	0	200	0
B	25	25–50	3.08	13	198	0	199	0	200	0
B	25	50–100	3.11	11	200	0	200	24	200	116
B	30	50–100	3.10	13	200	0	200	9	200	140

Table 2: Overview of the sets of knowledge bases making up dataset A and dataset B.

to make a knowledge base consistent. Hence, if we only need one interpretation, the respective knowledge base is consistent. Consequently, we need to subtract 1 from the computed minimum in order to get the correct value of $\mathcal{I}_h(\mathcal{K})$.

It should be noted that there are knowledge bases which contain one or more contradictory formulas, such as $a \wedge \neg a$. In such a case, there exists no interpretation (in classical propositional logic) which could satisfy the respective formula. If a formula $\phi \in \mathcal{K}$ is contradictory, we cannot include e_{ϕ_T} in any answer set of $P_h(\mathcal{K})$. We therefore needed to include e_{ϕ_F} in the answer set—which is not allowed due to the integrity constraint. Thus, no answer set of $P_h(\mathcal{K})$ exists, and $\mathcal{I}_h(\mathcal{K})$ has the value ∞.

We define $P_h(\mathcal{K})$ to be the extended logic program specified by the union of all rules defined in 1–6. For each $\omega_n \in M$, with M being an answer set of $P_h(\mathcal{K})$, we define i_{M,ω_n} via

$$i_{M,\omega_n}(x) = \begin{cases} \top & e_{a_T,\omega_n} \in M \\ \bot & e_{a_F,\omega_n} \in M \end{cases}$$

for all $x \in \mathsf{At}(\mathcal{K})$. Further, we define

$$\Omega(M) = \{i_{M,\omega_n} \mid \omega_n \in M\},$$

which corresponds to the minimal hitting set of $\mathcal{K}$.

Theorem 3 *Let M be an optimal answer set of $P_h(\mathcal{K})$. Then $|\Omega(M)| - 1 = \mathcal{I}_h(\mathcal{K})$. If no answer set of $P_h(\mathcal{K})$ exists, $\mathcal{I}_h(\mathcal{K}) = \infty$.*

4 Experiments

The goal of our experimental evaluation is to compare the empirical runtime of our ASP-based implementations with existing baseline implementations of the individual measures.

The three introduced ASP encodings for inconsistency measurement were constructed by means of the Java libraries provided by *TweetyProject*[3]. The actual calculation of the answer sets is performed by the Clingo solver, version 5.4.0 (Gebser et al. 2016). TweetyProject also includes naive (brute-force) implementations of all three inconsistency measures. More precisely, $\mathcal{I}_c$ is implemented by iterating through all subsets of atoms (with increasing cardinality), forgetting all occurrences of the atoms of the current set in the knowledge base (thus effectively setting their three-valued truth value to B), and then checking whether the resulting knowledge base is consistent by means of a SAT solver (here, Sat4j v2.3.5[4]). Once a consistent knowledge base is found, the cardinality of the current set of atoms is returned. The measure $\mathcal{I}_f$ is implemented by iterating through all possible forgetting operations (with increasing number) and checking whether the resulting knowledge base is consistent (again using Sat4j v2.3.5). The measure $\mathcal{I}_h$ is implemented by considering every set of interpretations (with increasing cardinality) and checking whether each formula of the knowledge base is satisfied by at least one interpretation. To the best of our knowledge, no further implementations of the three inconsistency measures exist.

All experiments were run on a computer with 16 GB RAM and a quad core Intel Core i7-8550U CPU which has a maximum clock speed of 4000 MHz.

4.1 Datasets

For evaluation, we consider both some existing benchmarks as well as a set of newly compiled knowledge bases which were created using *TweetyProject*. It should be noted that, to the best of our knowledge, in the field of inconsistency measurement no dedicated dataset exists that could be utilized to evaluate different implementations against each other. Hence, we compile our own dataset. More specifically, we generated four different sets of knowledge bases (datasets A–D)[5] tailored for different purposes as elaborated in the following.

To get a fundamental overview of the behavior of our implementations, we compiled dataset A, which consists of overall rather small random knowledge bases of varying complexity. To be precise, the dataset is comprised of six subsets, each containing 200 knowledge bases. The simplest subset contains between 5 and 15 formulas per knowledge

[3]http://tweetyproject.org/

[4]https://www.sat4j.org

[5]All datasets will be made publicly available.

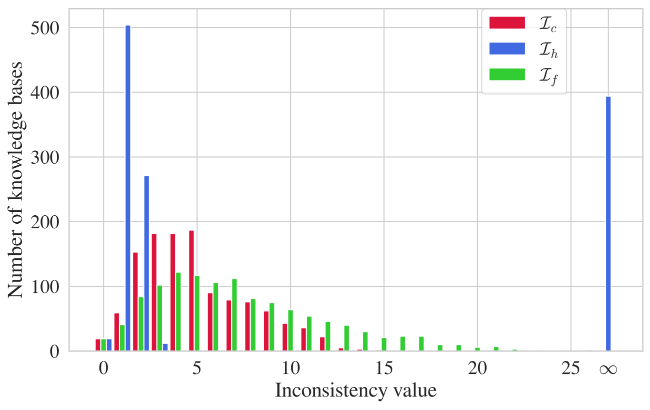

Figure 1: Overview of the inconsistency values of the knowledge bases in dataset A.

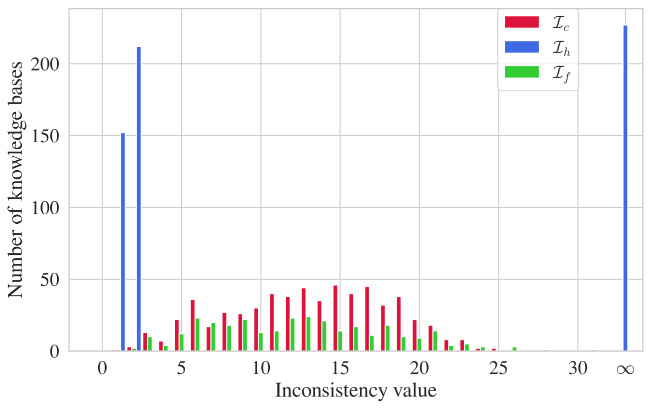

Figure 2: Overview of the inconsistency values of the knowledge bases in dataset B. Note that the set of values regarding $\mathcal{I}_f$ is incomplete due to both implementations timing out wrt. 256 out of the 600 instances.

base wrt. a signature size of 3, the most complex one between 25 and 50 wrt. a signature size of 20. More details can be obtained from the upper left part of Table 2. To generate these knowledge bases, the *SyntacticRandomSampler*[6] provided by *TweetyProject* was utilized. A few knowledge bases in dataset A are consistent, and some knowledge bases contain contradictory formulas (i. e., wrt. the latter knowledge bases, $\mathcal{I}_h = \infty$). More details regarding the inconsistency values of the knowledge bases of dataset A are provided in Figure 1.

As dataset A already reveals the limits of some of the implementations, dataset B is designed to be a bit more challenging for the remaining ones. Again, the *SyntacticRandomSampler* was used for the generation process, and again, the dataset consists of subsets of 200 formulas each. Overall, the dataset is comprised of 600 knowledge bases which contain between 25 and 150 formulas with signature sizes of 25 or 30. More details are given in the lower left part of Table 2. Besides, an overview of the inconsistency values of dataset B is provided in Figure 2.

In addition to the sampled knowledge bases, we considered benchmark data from different SAT competitions in dataset C. Because the subject of this work is to measure inconsistency, only inconsistent instances were considered. In total, we gathered 105 instances from four different sources:

1. 5 instances referring to the Pigeon Hole problem[7]. They consist of between 42 and 110 variables as well as between 133 and 561 clauses.
2. 8 knowledge bases encoding the two-coloring of a graph consisting of 60 to 160 variables and 160 to 400 clauses.
3. 8 knowledge bases from circuit fault analysis which comprise between 435 and 10,410 variables, and between 1027 and 34,238 clauses.
4. 84 DaimlerChrysler benchmarks[8] with between 1608 and 2038 variables and between 4496 and 11,352 clauses.

The inconsistency values of the knowledge bases in dataset C are 1 in all cases wrt. $\mathcal{I}_c$ and $\mathcal{I}_f$. Regarding $\mathcal{I}_h$, the inconsistency values are either 1 or ∞. Note that we refer only to those knowledge bases which did not cause a timeout for both implementations of $\mathcal{I}_f$ and $\mathcal{I}_h$.

Dataset D consists of knowledge bases extracted from benchmark data of the International Competition on Computational Models of Argumentation 2019 (ICCMA'19)[9]. An abstract argumentation framework (Dung 1995) is a directed graph $F = (A, R)$ where A is a set of arguments and R models a conflict relation between arguments. A computational task here is to find a *stable extension*, i. e., a set $E \subseteq A$ with $(a, b) \notin R$ for all $a, b \in E$ and $(a, c) \in R$ for all $c \in A \setminus E$ and some $a \in E$. For each instance from ICCMA'19, we encode the instance and the problem of finding such a stable extension via the approach from (Besnard, Doutre, and Herzig 2014) and, additionally, add constraints to ensure that 20% of randomly selected arguments have to be contained in E. Note that the latter constraints usually make the knowledge base inconsistent.

4.2 Results

To begin with, we measure the runtime of both the naive (brute-force) and the ASP-based versions of all three inconsistency measures $\mathcal{I}_c$, $\mathcal{I}_h$, and $\mathcal{I}_f$ on dataset A. A timeout is set to 120 seconds. The results clearly demonstrate the limitations of all three brute-force algorithms. Figure 3 shows a cactus plot which illustrates a direct comparison between the naive versions of all measures and their respective ASP-based counterparts. The measured execution times were sorted from low to high wrt. each algorithm. None of the ASP-based algorithms produced a timeout, while all

[6]http://tweetyproject.org/api/1.17/net/sf/tweety/logics/pl/util/SyntacticRandomSampler.html

[7]Those instances referring to the two-coloring of graphs, to circuit fault theory, and to the Pigeon Hole problem are available at https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html

[8]Available at https://web.archive.org/web/20080820084020/http://www-sr.informatik.uni-tuebingen.de/~sinz/DC/

[9]http://argumentationcompetition.org/2019/

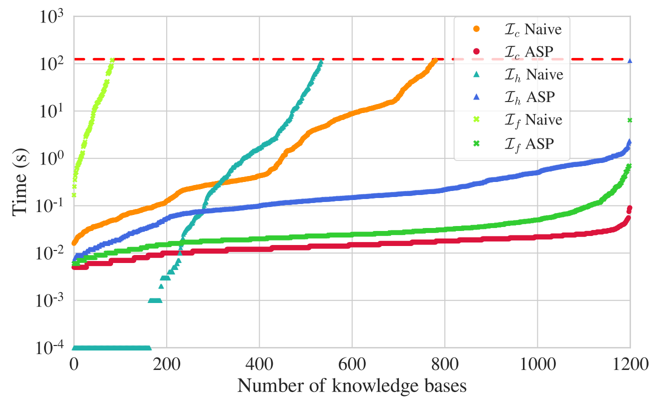

Figure 3: Runtimes of $\mathcal{I}_c$, $\mathcal{I}_h$, and $\mathcal{I}_f$ regarding both the naive and the ASP-based algorithms wrt. dataset A. The red dashed line indicates the timeout of 120 seconds.

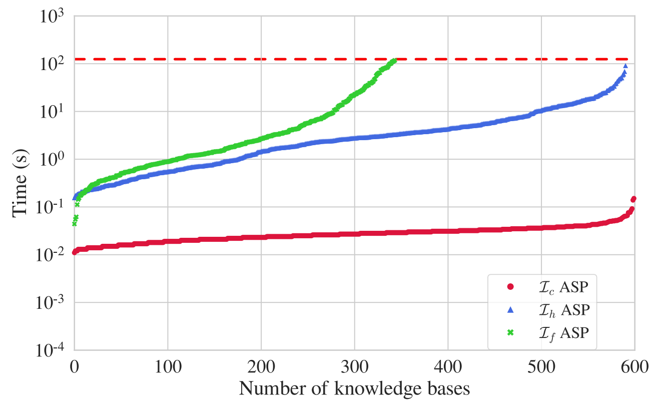

Figure 4: Runtimes of the ASP-based algorithms wrt. dataset B. Because the naive algorithms produced timeouts for all instances (except $\mathcal{I}_c$ in two cases and $\mathcal{I}_h$ in one case), they are not visualized in the plot. The red dashed line indicates the timeout of 120 seconds.

three naive versions did so in several hundred cases. In particular, the naive algorithm for $\mathcal{I}_f$ performs very poorly. The right part of Table 2 reveals that it could only handle some instances of the simplest subset of dataset A at all—for all other instances, it produced a timeout. Another noteworthy point is that both implementations for $\mathcal{I}_c$ performed comparatively well. The reason for this presumably lies in the nature of the inconsistency measure itself. For example, the number of possible values is, in most cases, smaller than that of $\mathcal{I}_h$ or $\mathcal{I}_f$.

Next, we applied all algorithms on dataset B. However, it turned out that all three naive algorithms produce timeouts in almost all instances. The only exceptions are two instances that could be solved by the naive variant of $\mathcal{I}_c$ and one instance that could be solved by the naive variant of $\mathcal{I}_h$. Hence, when considering dataset B, we focus on the ASP-based algorithms. Although the knowledge bases in dataset B are not much more complex than those of dataset A (see Table 2 for details), the ASP-based algorithms for $\mathcal{I}_h$ and $\mathcal{I}_f$ exhibit some difficulties, as Figure 4 visualizes. More precisely, the ASP-based implementation of $\mathcal{I}_h$ produces a timeout in 33 out of 600 cases, and the implementation for $\mathcal{I}_f$ even in 256 cases. More details regarding the number of timeouts for each implementation wrt. each subset of dataset B are provided in the lower right section of Table 2. Nevertheless, it should be noted that the ASP-based implementation of the contension inconsistency measure behaved differently: not only did it not produce any timeouts, it took only < 1 second for each knowledge base in dataset B.

We also applied all algorithms on dataset C. Because of the large size of some of the knowledge bases (up to 2038 variables and 11,352 clauses), we increased the timeout from 2 to 5 minutes. As Figure 5 shows, both implementations of $\mathcal{I}_c$ as well as both implementations of $\mathcal{I}_f$ were able to compute inconsistency values for most knowledge bases. However, regarding both measures, the ASP version produced fewer timeouts than its respective naive counterpart. The naive implementation of $\mathcal{I}_h$ could not solve a single instance of dataset C. The corresponding ASP-based implementation could at least compute the inconsistency values of 11 knowledge bases.

The overall rather poor performance of both implementations of $\mathcal{I}_h$ compared to the implementations of the other two measures is presumably due to the nature of dataset C: all knowledge bases are given in the DIMACS[10] file format. This means that all knowledge bases are in conjunctive normal form and each clause is considered an individual formula. Moreover, the inconsistency value is always 1, except for some instances regarding $\mathcal{I}_h$, where the inconsistency value is ∞. Further, most knowledge bases contain a large number of clauses. The size of the ASP encodings regarding $\mathcal{I}_c$ and $\mathcal{I}_f$ largely depends on the number of atoms, or atom occurrences, respectively, as well as the size and complexity of the individual formulas. The size of the ASP encoding of $\mathcal{I}_h$, on the other hand, highly depends on the number of possible interpretations, i. e., the number of formulas, because every formula, subformula and atom needs to be encoded wrt. each of these interpretations. Consequently, with a large number of formulas in a knowledge base, we also get an answer set program containing a vast number of rules. This makes $\mathcal{I}_h$ slower and less practically applicable as the other two measures in a dataset which possesses properties like datset C.

Another aspect that strikes out with regard to dataset C is that the naive implementations of $\mathcal{I}_c$ and $\mathcal{I}_f$ perform relatively well in comparison to dataset A and B. The reason for this lies most probably in the inconsistency values of the knowledge bases in dataset C, which is always 1 for $\mathcal{I}_c$ and $\mathcal{I}_f$. The inconsistency values of most instances in both dataset A and B are significantly higher (see Figures 1 and 2). Since the brute-force implementations of $\mathcal{I}_c$ and $\mathcal{I}_f$ check the lowest possible values first, they can compute lower inconsistency values faster than higher ones, given the size of the respective knowledge bases is the same.

Finally, we run all implementations on dataset D. As with

[10]http://www.satcompetition.org/2011/format-benchmarks2011.html

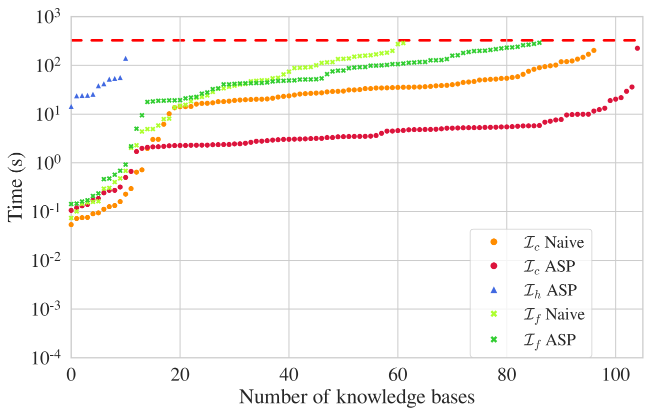

Figure 5: Runtimes of $\mathcal{I}_c$, $\mathcal{I}_f$, and $\mathcal{I}_h$ regarding dataset C. The naive implementation of $\mathcal{I}_h$ produced a timeout for all instances, so it is not shown in the plot. The red dashed line indicates the timeout of 300 seconds.

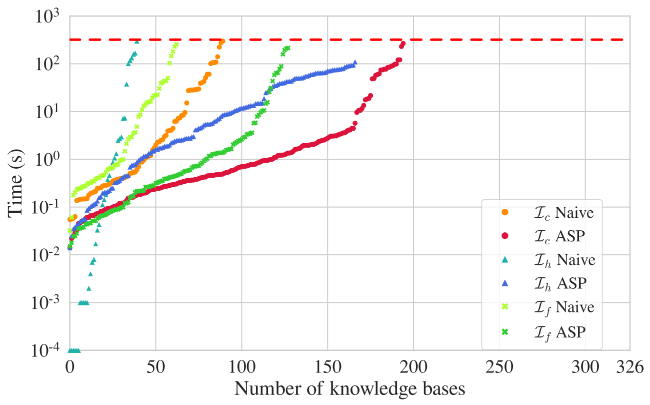

Figure 6: Runtimes of $\mathcal{I}_c$, $\mathcal{I}_h$, and $\mathcal{I}_f$ regarding both the naive and the ASP-based algorithms wrt. dataset D. The red dashed line indicates the timeout of 300 seconds.

dataset C, we set a timeout to 5 minutes. As Figure 6 visualizes, all six implementations could solve a non-empty subset of the knowledge bases in the dataset. Nonetheless, it is noticeable that none of the implementations could compute inconsistency values for the entire dataset. All implementations produced a timeout for at least 100 instances. Again, each ASP-based implementation had fewer timeouts than its respective naive counterpart.

5 Conclusion

In the course of this paper, we presented algorithms based on reductions to ASP for the contension inconsistency measure, the forgetting-based inconsistency measure, and the hitting set inconsistency measure. Moreover, we experimentally evaluated them against corresponding brute-force algorithms wrt. execution time. The evaluation showed that the novel ASP-based implementations perform clearly superior. We also learned that the naive implementations perform relatively worse when inconsistency values are large. The ASP encodings, on the other hand, are not dependent on the level of inconsistency.

With regard to future work, one aim is to utilize answer set programming to encode other inconsistency measures as well. For example, measures with higher computational complexity than those considered in this paper may be examined. Furthermore, it is of interest to investigate how our ASP-based algorithms perform in real-world applications. For instance, Nagel et al. presented a study about inconsistencies in business rules, which takes a quantitative perspective (Nagel, Corea, and Delfmann 2019), and thus could benefit from practically applicable algorithms. Other possible areas of application are mentioned in Section 1.

One of the insights gained throughout the course of this work is that large-sized knowledge bases are still problematic. With regard to datasets B and C, the ASP-based implementations for both $\mathcal{I}_f$ and $\mathcal{I}_h$ produced some timeouts, and with regard to dataset D, none of the three implementations could compute inconsistency values for a number of knowledge bases within the time limit. Therefore, another area of research that may be relevant with respect to the algorithmic perspective on inconsistency measures is that of approximate algorithms.

Acknowledgements This research is supported by the German Research Association (DFG), project number 424710479.

References

Bertossi, L. 2018. Measuring and Computing Database Inconsistency via Repairs. In *Proceedings of the 12th International Conference on Scalable Uncertainty Management (SUM'18)*.

Besnard, P. 2016. Forgetting-based inconsistency measure. In *International Conference on Scalable Uncertainty Management*, 331–337. Springer.

Besnard, P.; Doutre, S.; and Herzig, A. 2014. Encoding argument graphs in logic. In *International Conference on Information Processing and Management of Uncertainty in Knowledge-based Systems - IPMU 2014*.

Brewka, G.; Eiter, T.; and Truszczynski, M. 2011. Answer set programming at a glance. *Communications of the ACM* 54(12): 92–103.

Cholvy, L.; Perrussel, L.; and Thevenin, J.-M. 2017. Using inconsistency measures for estimating reliability. *International Journal of Approximate Reasoning* 89: 41–57.

Dung, P. M. 1995. On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games. *Artificial Intelligence* 77(2): 321–358.

Dvorák, W.; Gaggl, S. A.; Rapberger, A.; Wallner, J. P.; and Woltran, S. 2020. The ASPARTIX System Suite. In Prakken, H.; Bistarelli, S.; Santini, F.; and Taticchi, C., eds., *Computational Models of Argument - Proceedings of COMMA 2020, Perugia, Italy, September 4-11, 2020*, volume 326 of *Frontiers in Artificial Intelligence and Applications*, 461–462. IOS Press.

Erdem, E.; Gelfond, M.; and Leone, N. 2016. Applications of answer set programming. *AI Magazine* 37(3): 53–68.

Erdem, E.; Patoglu, V.; Saribatur, Z. G.; Schüller, P.; and Uras, T. 2013. Finding optimal plans for multiple teams of robots through a mediator: A logic-based approach. *Theory and Practice of Logic Programming* 13(4-5): 831–846.

Gebser, M.; Kaminski, R.; Kaufmann, B.; Ostrowski, M.; Schaub, T.; and Wanko, P. 2016. Theory solving made easy with clingo 5. In *Technical Communications of the 32nd International Conference on Logic Programming (ICLP 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2012. Answer set solving in practice. *Synthesis lectures on artificial intelligence and machine learning* 6(3): 1–238.

Gelfond, M.; and Leone, N. 2002. Logic programming and knowledge representation—the A-Prolog perspective. *Artificial Intelligence* 138(1-2): 3–38.

Gelfond, M.; and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New generation computing* 9(3-4): 365–385.

Grant, J. 1978. Classifications for Inconsistent Theories. *Notre Dame Journal of Formal Logic* 19(3): 435–444.

Grant, J.; and Hunter, A. 2011. Measuring consistency gain and information loss in stepwise inconsistency resolution. In *Proceedings ECSQARU'11*, 362–373. Springer.

Grant, J.; and Hunter, A. 2017. Analysing Inconsistent Information Using Distance-Based Measures. *Int. J. Approx. Reasoning* 89(C): 3–26.

Grant, J.; and Martinez, M. V., eds. 2018. *Measuring Inconsistency in Information*, volume 73 of *Studies in Logic*. College Publications.

Hunter, A. 2006. How to act on inconsistent news: Ignore, resolve, or reject. *Data & Knowledge Engineering* 57(3): 221–239.

Kuhlmann, I.; and Thimm, M. 2020. An Algorithm for the Contension Inconsistency Measure using Reductions to Answer Set Programming. In *International Conference on Scalable Uncertainty Management*, 289–296. Springer.

Lifschitz, V. 2008. What is answer set programming? In *Proceedings AAAI'08*, 1594–1597.

Martinez, A. B. B.; Arias, J. J. P.; and Vilas, A. F. 2004. On Measuring Levels of Inconsistency in Multi-Perspective Requirements Specifications. In *Proceedings of the 1st Conference on the Principles of Software Engineering (PRISE'04)*.

Nagel, S.; Corea, C.; and Delfmann, P. 2019. Effects of quantitative measures on understanding inconsistencies in business rules. In *Proceedings of the 52nd Hawaii International Conference on System Sciences*, 146–155.

Potyka, N.; and Thimm, M. 2017. Inconsistency-tolerant reasoning over linear probabilistic knowledge bases. *International Journal of Approximate Reasoning* 88: 209–236.

Priest, G. 1979. Logic of Paradox. *Journal of Philosophical Logic* 8: 219–241.

Reiter, R. 1980. A logic for default reasoning. *Artificial intelligence* 13(1-2): 81–132.

Thimm, M. 2016. Stream-based inconsistency measurement. *International Journal of Approximate Reasoning* 68: 68–87.

Thimm, M. 2019. Inconsistency Measurement. In Amor, N. B.; Quost, B.; and Theobald, M., eds., *Proceedings of the 13th International Conference on Scalable Uncertainty Management (SUM'19)*, 9–23. Springer International Publishing.

Thimm, M.; and Wallner, J. 2019. On the Complexity of Inconsistency Measurement. *Artificial Intelligence* 275.