

# On Database Inconsistency Measures

(Discussion Paper)

Francesco Parisi<sup>1</sup>, John Grant<sup>2</sup>

<sup>1</sup>*DIMES Department, University of Calabria, Italy*

<sup>2</sup>*University of Maryland at College Park, USA*

## Abstract

Although there has been an extensive body of work on handling inconsistency in databases, little work has been done on measuring inconsistency. In this paper, we present inconsistency measures (IMs) for databases with denial constraints. Our database IMs are inspired by well-established methods to quantify inconsistency in propositional knowledge bases, but are tailored to the relational database context where data are generally the reason for inconsistency, not the integrity constraints. We investigate the complexity of some problems related to measuring inconsistency and briefly discuss other ideas concerning IMs.

## Keywords

Relational database, Inconsistency measure, Denial constraints

## 1. Introduction

There is a growing number of applications where inconsistent information arises, often because data are obtained from multiple sources. This has led to an extensive body of work on handling inconsistent data, and in particular inconsistent databases. Approaches for dealing with inconsistent databases include consistent query answering frameworks [1, 2], inconsistency management policies [3, 4, 5], interactive data repairing and cleaning systems [6, 7, 8, 9], as well as interactive data exploration tools [10, 11]. However, little work has been done on measuring inconsistency in databases, a problem that in contrast has been extensively investigated for propositional knowledge bases [12, 13, 14, 15, 16].

Measuring the amount of inconsistency in a database can help in understanding the primary sources of conflicts as well as devising ways to deal with them. For instance, inconsistency measurement has recently been explored in [17] to quantify and understand inconsistency in bank holding company data w.r.t. four consistency rules (i.e., integrity constraints) specifically asserted for the considered domain. In general, quantifying and monitoring the amount of inconsistency helps to get information on the health status of data, whose quality is more and more important nowadays. Furthermore, measuring inconsistency makes it possible to compare the amount of inconsistency between various chunks of information. Indeed, inconsistency measures (IMs for

---


*SEBD 2021: The 29th Italian Symposium on Advanced Database Systems, September 5-9, 2021, Pizzo Calabro (VV), Italy*

✉ fparisi@dimes.unical.it (F. Parisi); grant@cs.umd.edu (J. Grant)

🆔 0000-0001-9977-1355 (F. Parisi)



© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

short) are important for inconsistency-tolerant integrity checking [18], where one wants to accept an update only if the measure of inconsistency of the old state does not increase in the new state.

Although the idea of measuring inconsistency was introduced more than 40 years ago, in [19], at that time it did not seem to be an important issue. The problem became more noticeable in the 1990s when it became possible to store large amounts of information. It was only in the early 2000s when several AI researchers started to investigate this issue systematically [16]. The bulk of this work since then has been for propositional knowledge bases, that is, where the information is presented as a set of formulas in propositional logic. In the last couple of years the work has been extended to other frameworks. [14] surveys what has been done so far and gives some extensions.

There are few works addressing the problem of measuring inconsistency in relational databases. [20] first developed single-dependency axioms for dirtiness functions quantifying inconsistency w.r.t. one functional dependency (FD)—a simple type of denial constraint—considered in isolation, and proposed an IM that satisfies these axioms. Then a single axiom for dirtiness functions that can handle multiple FDs was proposed, although such functions are supposed to be built on top of dirtiness functions for single FDs. The approach in [21, 22, 23] shows how database IMs can be applied to integrity checking [24, 18], relaxing repairs, and repair checking, which are applications that perfectly fit within our framework. However, the degrees of inconsistency defined in [23] form a partially ordered set; hence it is not always possible to compare the inconsistency of different databases. Finally, [25] proposes an IM based on an abstract repair semantics, where the degree of inconsistency depends on the distance between the database instance and the set of repairs, and an instantiation for cardinality-repairs that can be computed via answer-set programs [26].

Previous work has not investigated the problem of tailoring propositional IMs to relational data, as well as analysing postulates compliance and the complexity of the resulting IMs, which is the focus of our work [27]. We introduce new IMs for relational databases with *denial constraints*. These measures are inspired by IMs for propositional logic. However, we do not apply well-known methods for propositional logic directly, as done for instance in [28, 29]; rather, we use these methods as inspiration to define (by analogy) new IMs for databases. In particular, we introduce the database counterpart, namely  $\mathcal{I}_x$  with  $x \in \{B, M, \#, P, A, H, C, \eta\}$ , of several propositional IMs  $I_x$ . Every *database IM*  $\mathcal{I}_x$  measures the inconsistency by blaming database tuples only. This is different from measuring the inconsistency of a set of formulas, all of which have the same status, as is the case for propositional knowledge bases. This leads to some substantial differences between the two cases: for instance, some measures that are distinct for propositional knowledge bases become identical in our setting (namely  $\mathcal{I}_C$  and  $\mathcal{I}_H$ ). We investigate the complexity of the problems of deciding whether a given value is lower than (**LV**), greater than (**UV**), or equal to (**EV**) the inconsistency measured using a given IM  $\mathcal{I}_x$ . A summary of the complexity results obtained for these problems, as well as for the problem of computing the actual value of an inconsistency measure (**IM** problem), is reported in Table 1 (Section 2.4). We conclude the paper by discussing rationality postulates for IMs and outlining some directions for future work.

We assume the reader is familiar with the standard concepts of relational database and integrity constraint, and with the complexity classes used in the paper (see e.g. [30] for a comprehensive overview).

## 2. Database Inconsistency Measures

We use  $\mathbf{D}$  to denote the set of all database instances over a fixed but arbitrary database scheme  $\mathcal{D}\mathcal{S}$ . We say that a database instance  $D$  is consistent, meaning that  $D$  is consistent w.r.t. a fixed but arbitrary set  $\mathcal{C}$  of denial constraints. A *denial constraint* is a first-order sentence of the form:  $\forall \vec{x}_1, \dots, \vec{x}_k [\neg R_1(\vec{x}_1) \vee \dots \vee \neg R_k(\vec{x}_k) \vee \varphi(\vec{x}_1, \dots, \vec{x}_k)]$  where: (i)  $\forall i \in [1..k]$ ,  $\vec{x}_i$  are tuples of variables and  $R_i(\vec{x}_i)$  are atoms over  $\mathcal{D}\mathcal{S}$ ; and (ii)  $\varphi$  is a disjunction of built-in predicates of the form  $\tau_i \circ \tau_j$  where  $\tau_i$  and  $\tau_j$  are variables in  $\vec{x}_1, \dots, \vec{x}_k$  or constants, and  $\circ \in \{=, \neq, >, <, \geq, \leq\}$ . We will write  $[\neg R_1(\vec{x}_1) \vee \dots \vee \neg R_k(\vec{x}_k) \vee \varphi(\vec{x}_1, \dots, \vec{x}_k)]$  for a constraint (of arity  $k$ ). For the sake of presentation, we will often omit the database scheme and the set of constraints in the terminology.

The idea of an inconsistency measure is to assign a nonnegative number to a knowledge base that measures its inconsistency [12, 13, 14, 15, 16]. We now define it for a database.

**Definition 1 (Inconsistency Measure).** A function  $\mathcal{I} : \mathbf{D} \rightarrow \mathbb{R}_{\infty}^{\geq 0}$  is an **inconsistency measure** if the following two conditions hold for all  $D, D' \in \mathbf{D}$ :

- 1) **Consistency:**  $\mathcal{I}(D) = 0$  iff  $D$  is consistent;
- 2) **Monotony:** If  $D \subseteq D'$ , then  $\mathcal{I}(D) \leq \mathcal{I}(D')$ .

Consistency and Monotony are called (rationality) postulates. Postulates are desirable properties for IMs and we will mention additional ones in Section 3. However, we require that a function on databases must at least satisfy these two postulates in order to be called an IM. Consistency means that all and only consistent databases get measure 0. Monotony means that the enlargement of a database cannot decrease its measure.

A **minimal inconsistent subset** of  $D$  is a set of tuples  $X \subseteq D$  such that  $X$  is inconsistent (with respect to  $\mathcal{C}$ ) and no proper subset of  $X$  is inconsistent. We denote by  $MI(D)$  the set of minimal inconsistent subsets of  $D$ . Similarly, a **maximal consistent subset** is a set of tuples  $Y$  that is consistent and no proper superset of  $Y$  is consistent. We write  $MC(D)$  for the set of maximal consistent subsets (of  $D$ ). Any tuple that occurs in a minimal inconsistent subset is **problematic**; otherwise it is **free**. We use  $Problematic(D)$  and  $Free(D)$  to denote the sets of problematic and free tuples of  $D$ . A tuple  $t$  is **contradictory** if  $\{t\}$  is inconsistent w.r.t.  $\mathcal{C}$ . We write  $Contradictory(D)$  for the set of contradictory tuples.

**Example 1.** Consider the database scheme  $\mathcal{D}\mathcal{S}_{ex}$  consisting of the relation *MealTicket*(*Number*, *Value*, *Holder*, *Date*) whose instance contains the number, the value, the holder, and the issue date of meal tickets (one for each tuple) provided by a company to the employees.  $\mathcal{C}_{ex}$  consists of the following denial constraints:

- $c_1 = [\neg MealTicket(x_1, x_2, x_3, x_4) \vee x_2 > 0]$ , stating that the value (i.e., the amount of the ticket) of every tuple of *MealTicket* must be a positive number.
- $c_2 = [\neg MealTicket(x_1, x_2, x_3, x_4) \vee \neg MealTicket(x_1, x_5, x_6, x_7) \vee x_2 = x_5]$ , i.e., the functional dependency *Number*  $\rightarrow$  *Value*, stating that there cannot be two distinct tickets with the same number and different values.
- $c_3 = [\neg MealTicket(x_1, x_2, x_3, x_4) \vee \neg MealTicket(x_1, x_5, x_6, x_7) \vee x_3 = x_6]$ , i.e., the functional dependency *Number*  $\rightarrow$  *Holder*.

- $c_4 = [\neg \text{MealTicket}(x_1, x_2, x_3, x_4) \vee \neg \text{MealTicket}(x_5, x_6, x_3, x_4) \vee \neg \text{MealTicket}(x_7, x_8, x_3, x_4) \vee x_1 = x_5 \vee x_1 = x_7 \vee x_5 = x_7]$ , stating the numerical dependency (see [31]) *Holder, Date*  $\rightarrow^2$  *Number* whose meaning is that for every holder and date there can be at most 2 meal ticket numbers.

An instance  $D_{ex}$  of  $\mathcal{DS}_{ex}$  consisting of 7 tuples  $\vec{t}_1, \dots, \vec{t}_7$  is shown below.

Number	Value	Holder	Date	Tuple
1001	15	Matthew	2018-12-13	$\vec{t}_1$
1001	15	Matthew	2018-12-18	$\vec{t}_2$
1001	15	Sophia	2018-12-17	$\vec{t}_3$
1004	20	Sophia	2018-12-17	$\vec{t}_4$
1005	0	Alex	2018-12-18	$\vec{t}_5$
1006	10	Alex	2018-12-18	$\vec{t}_6$
1007	20	Alex	2018-12-18	$\vec{t}_7$

Going through the integrity constraints we find that: 1)  $\vec{t}_5$  is inconsistent with  $c_1$ : it is a contradictory tuple; 2) no pair of tuples violates  $c_2$ ; 3) the two pairs of tuples,  $\vec{t}_1, \vec{t}_3$ , and  $\vec{t}_2, \vec{t}_3$ , are inconsistent with  $c_3$ ; 4) the three tuples  $\vec{t}_5, \vec{t}_6$ , and  $\vec{t}_7$  together are inconsistent with  $c_4$ . Hence,  $\text{MI}(D_{ex}) = \{\{\vec{t}_5\}, \{\vec{t}_1, \vec{t}_3\}, \{\vec{t}_2, \vec{t}_3\}\}$ . Note that  $\{\vec{t}_5, \vec{t}_6, \vec{t}_7\}$  is not included because it contains  $\{\vec{t}_5\}$  (and thus it is not minimal). Thus there are 4 problematic tuples in  $D_{ex}$ , and 3 free-tuples. Also,  $\text{MC}(D) = \{\{\vec{t}_1, \vec{t}_2, \vec{t}_4, \vec{t}_6, \vec{t}_7\}, \{\vec{t}_3, \vec{t}_4, \vec{t}_6, \vec{t}_7\}\}$ .

## 2.1. Measures using Minimal Inconsistent Subsets

We start with the measures that rely on minimal inconsistent subsets and the related concepts defined above. We give the rationale for each measure below.

**Definition 2 (Database Inconsistency Measures).** For a database  $D$ , the IMs  $\mathcal{I}_B$ ,  $\mathcal{I}_M$ ,  $\mathcal{I}_\#$ ,  $\mathcal{I}_P$ ,  $\mathcal{I}_A$ , and  $\mathcal{I}_H$  are such that:

- $\mathcal{I}_B(D) = 1$  if  $D$  is inconsistent and  $\mathcal{I}_B(D) = 0$  if  $D$  is consistent.
- $\mathcal{I}_M(D) = |\text{MI}(D)|$ .
- $\mathcal{I}_\#(D) = \begin{cases} 0 & \text{if } D \text{ is consistent,} \\ \sum_{X \in \text{MI}(D)} \frac{1}{|X|} & \text{otherwise.} \end{cases}$
- $\mathcal{I}_P(D) = |\text{Problematic}(D)|$ .
- $\mathcal{I}_A(D) = (|\text{MC}(D)| + |\text{Contradictory}(D)|) - 1$ .
- $\mathcal{I}_H(D) = \min\{|X| \text{ s.t. } X \subseteq D \text{ and } \forall M \in \text{MI}(D), X \cap M \neq \emptyset\}$ .

$\mathcal{I}_B$  is also called the drastic measure [15]: 0 means consistent; 1 means inconsistent. So this measure simply distinguishes between consistent and inconsistent databases.  $\mathcal{I}_M$  counts the number of minimal inconsistent subsets [15]. The rationale is that a minimal inconsistent subset represents a minimal inconsistency for a set of database tuples; hence this measure counts the

number of such inconsistencies.  $\mathcal{I}_\#$  also counts the number of minimal inconsistent subsets, but it gives larger sets a smaller weight. The reason for  $\mathcal{I}_\#$  in the propositional logic setting is that when a minimal inconsistent set contains more formulas than another minimal inconsistent set, the former is intuitively less inconsistent than the latter [15]. For instance, we can say that intuitively  $\{a, \neg a\}$  is more inconsistent than  $\{a, a \rightarrow b, b \rightarrow c, c \rightarrow d, \neg d\}$ . In the database setting, this measure gives a higher value to the violations of constraints having smaller arity, that is, to violations due to a smaller sets of tuples. It is a way to differentiate between constraints. For instance, in Example 1 this measure gives a violation of  $c_1$  the highest value, a violation of  $c_4$  the lowest value, and a violation of  $c_2$  or  $c_3$  a middle value.  $\mathcal{I}_P$  counts the number of tuples that are in one or more minimal inconsistencies [12].  $\mathcal{I}_A$  uses the cardinality of the set of maximal consistent subsets [12], i.e., of the set of database repairs [1]. Intuitively, the larger this set, the larger is the space of different ways to get consistency, the higher the degree of inconsistency. Contradictory tuples are added as they do not appear in any way in a maximal consistent set; then 1 must be subtracted to obtain  $\mathcal{I}_A(D) = 0$  for a consistent  $D$  because every consistent database has a maximal consistent subset, namely  $D$  itself.  $\mathcal{I}_H$  counts the minimal number of tuples whose deletion makes the database consistent [13]. Hence  $\mathcal{I}_H$  can be written as  $\mathcal{I}_H = \min\{|X| \text{ s.t. } D \setminus X \text{ is consistent}\}$ . In fact, both  $\mathcal{I}_A$  and  $\mathcal{I}_H$  link the inconsistency measurement to the ways of restoring consistency, an idea recently explored in [25, 26] where the degree of inconsistency depends on the distance between the database instance and the set of possible repairs under a given repair semantics.

**Example 2.** *The values of IMs for the database of our running example are as follows.*

- $\mathcal{I}_B(D_{ex}) = 1$  as the database is inconsistent.
- $\mathcal{I}_M(D_{ex}) = 3$  as there are 3 minimal inconsistent subsets as given above.
- $\mathcal{I}_\#(D_{ex}) = 1 + \frac{1}{2} + \frac{1}{2} = 2$  as there is one minimal inconsistent subset of size 1 and two minimal inconsistent subsets of size 2.
- $\mathcal{I}_P(D_{ex}) = 4$  as there are 4 distinct tuples in  $MI(D_{ex})$ .
- $\mathcal{I}_A(D_{ex}) = 2$  because there are 2 maximal consistent subsets.
- $\mathcal{I}_H(D_{ex}) = 2$  as the set  $\{\vec{t}_3, \vec{t}_5\}$  intersects with each minimal inconsistent subset.

## 2.2. A Measure using Three-valued Logic

The contension measure  $\mathcal{I}_C$  [12] uses a three-valued (3VL) logic. A *3VL-interpretation* is a function  $i$  that assigns to each atom  $R(\vec{t})$  in  $D$  one of the three truth values:  $T$  (true),  $F$  (false), or  $B$  (both). A 3VL-interpretation uses an ordering on the truth values where  $F < B < T$  and  $\wedge$  computes the minimum value while  $\vee$  computes the maximum value; also  $\neg(B) = B$ . So, for example,  $B \wedge F = F$  and  $B \vee F = B$ . Then, an interpretation  $i$  satisfies a formula iff the truth-value of the formula for  $i$  is  $T$  or  $B$ .

Given a database  $D$  with a set  $\mathcal{C}$  of integrity constraints, a 3VL interpretation is a 3VL model iff all the constraints are satisfied and no atom  $R(\vec{t}) \in D$  is assigned  $F$ . We use  $\text{Models}(D)$  to denote the set of 3VL models for a database  $D$  (with the integrity constraints in the background). Also, for a 3VL interpretation  $i$  we define  $\text{Conflictbase}(i) = \{R(\vec{t}) \mid i(R(\vec{t})) = B\}$ , the tuples that have truth value  $B$ .

**Definition 3 (Contension measure  $\mathcal{I}_C$ ).** For database  $D$ , measure  $\mathcal{I}_C$  is such that  $\mathcal{I}_C(D) = \min\{|\text{Conflictbase}(i)| \mid i \in \text{Models}(D)\}$ .

Thus  $\mathcal{I}_C$  counts the minimal number of tuples that if we could consider them both true and false would resolve all inconsistencies. For the database of our running example, we have that  $\mathcal{I}_C(D_{ex}) = 2$  as the minimal number of  $B$  values for an interpretation occurs when  $i(\vec{t}_1) = i(\vec{t}_2) = i(\vec{t}_4) = i(\vec{t}_6) = i(\vec{t}_7) = T$  and  $i(\vec{t}_3) = i(\vec{t}_5) = B$ .

In the propositional case, of the measures considered no 2 measures give the same result for all knowledge bases. But because of the special structure of databases, for database IMs there is an equality that does hold:  $\mathcal{I}_C(D) = \mathcal{I}_H(D)$ . In fact,  $\mathcal{I}_C(D)$  counts the minimum number of tuples that need to be assigned  $B$  in order to get a 3VL model in  $\text{Models}(D)$ . This is the same as the cardinality of the set  $X \subseteq D$  having a non-empty intersection with every minimal inconsistent subset of  $D$ . It can be shown that no other pair of IMs considered in this paper is identical [27].

### 2.3. A Probabilistic Measure

Finally, we define the database counterpart of the probabilistic measure  $I_\eta$  that uses the **PSAT (probabilistic satisfiability)** concept [16]. A PSAT instance is a set,  $\Gamma = \{\text{Pr}(\phi_i) \geq p_i \mid 1 \leq i \leq m\}$ , that assigns probability lower bounds to a set  $\{\phi_1, \dots, \phi_m\}$  of formulas; therefore  $0 \leq p_i \leq 1$  for  $1 \leq i \leq m$ . A *probability function* over a set  $X$  is a function  $\pi : X \rightarrow [0, 1]$  such that  $\sum_{x \in X} \pi(x) = 1$ . Let  $\text{Int}$  be the set of all (classical) interpretations of the set of formulas and  $\pi$  a probability function over  $\text{Int}$ . The probability of a formula  $\phi$  according to  $\pi$  is the sum of the probabilities assigned to the interpretations assigning  $T$  (true) to  $\phi$ , that is,  $\text{Pr}_\pi(\phi) = \sum_{i \in \text{Int}, i(\phi)=T} \pi(i)$  for every formula  $\phi \in K$ . A PSAT instance is *satisfiable* if there is a probability function  $\pi$  over  $\text{Int}$  such that  $\text{Pr}_\pi(\phi_i) \geq p_i$  for all  $1 \leq i \leq m$ .

Given a database  $D$ , a set of integrity constraints  $\mathcal{C}$ , and a probability threshold  $\eta \in [0, 1]$ , we define the PSAT instance  $\Gamma_{\mathcal{C}, \eta}(D) = \{\text{Pr}(R(\vec{t})) \geq \eta \mid R(\vec{t}) \in D\} \cup \{\text{Pr}(g(c)) = 1 \mid g(c) \text{ is a ground constraint for } c \in \mathcal{C}\}$ . Here, integrity constraints are assigned a probability equal to one, analogously to the case of probabilistic databases with integrity constraints (see e.g. [32]).

**Definition 4 (Probabilistic measure  $\mathcal{I}_\eta$ ).** Given a database  $D$  and a set of constraints  $\mathcal{C}$ , the *inconsistency measure*  $\mathcal{I}_\eta$  is such that  $\mathcal{I}_\eta(D) = 1 - \max\{\eta \in [0, 1] \mid \Gamma_{\mathcal{C}, \eta}(D) \text{ is satisfiable}\}$ .

Thus,  $\mathcal{I}_\eta(D)$  is one minus the maximum probability lower bound  $\eta$  that one can consistently assign to all tuples in  $D$ . For our running example, we have that  $\mathcal{I}_\eta(D_{ex}) = 1$  because the maximum probability that can be assigned to (contradictory) tuple  $\vec{t}_5$  is 0.

### 2.4. Complexity of the Database Inconsistency Measures

We investigate the data-complexity of the three decision problems, which intuitively ask if a given rational value  $v$  is, respectively, lower than, greater than, or equal to the value returned by a given IM when applied to a given database. Observe that every IM returns a rational number, including  $\mathcal{I}_\eta$  (as it can be shown by reasoning as in [16]).

**Table 1**  
Complexity of **LV**, **UV**, **EV**, and **IM** problems.

Measure(s)	$\mathbf{LV}_{\mathcal{I}}(D, v)$	$\mathbf{UV}_{\mathcal{I}}(D, v)$	$\mathbf{EV}_{\mathcal{I}}(D, v)$	$\mathbf{IM}_{\mathcal{I}}(D)$
$\mathcal{I}_B, \mathcal{I}_M, \mathcal{I}_\#, \mathcal{I}_P$	$P$	$P$	$P$	$FP$
$\mathcal{I}_A$	$PP$	$PP$	$PP$	$\#P$ -complete
$\mathcal{I}_H, \mathcal{I}_C$	$coNP$ -complete	$NP$ -complete	$D^P$ -complete	$FP^{NP[\log n]}$ -complete
$\mathcal{I}_\eta$	$coNP$ -complete	$NP$ -complete	$D^P$	$FP^{NP}$

**Definition 5 (Lower (LV), Upper (UV), and Exact Value (EV) problems).** Let  $\mathcal{I}$  be an IM. Given a database  $D$  over a fixed database scheme with a fixed set of denial constraints, and a value  $v \in \mathbb{Q}^{>0}$ ,  $\mathbf{LV}_{\mathcal{I}}(D, v)$  is the problem of deciding whether  $\mathcal{I}(D) \geq v$ . Given  $D$  and  $v' \in \mathbb{Q}^{\geq 0}$ ,  $\mathbf{UV}_{\mathcal{I}}(D, v')$  is the problem of deciding whether  $\mathcal{I}(D) \leq v'$ , and  $\mathbf{EV}_{\mathcal{I}}(D, v')$  is the problem of deciding whether  $\mathcal{I}(D) = v'$ .

We also consider the problem of determining the IM value.

**Definition 6 (Inconsistency Measurement (IM) problem).** Let  $\mathcal{I}$  be an IM. Given a database  $D$  over a fixed database scheme with a fixed set of denial constraints,  $\mathbf{IM}_{\mathcal{I}}(D)$  is the problem of computing the value of  $\mathcal{I}(D)$ .

The complexity of  $\mathbf{LV}_{\mathcal{I}}(D, v)$ ,  $\mathbf{UV}_{\mathcal{I}}(D, v)$ ,  $\mathbf{EV}_{\mathcal{I}}(D, v)$ , and  $\mathbf{IM}_{\mathcal{I}}(D)$  is as given in Table 1 [27]. Interestingly, while the measures  $\mathcal{I}_B, \mathcal{I}_M, \mathcal{I}_\#, \mathcal{I}_P$  become tractable in the database setting and the complexity of  $\mathcal{I}_A$  decreases, the measures  $\mathcal{I}_H$  and  $\mathcal{I}_\eta$  remain as hard as in the propositional case [33] even under data complexity.

### 3. Discussion and Future Work

We have introduced a framework for measuring inconsistency in databases. We believe that the definition and investigation of IMs for databases benefit from our approach to the problem, which stems from what has been done in the past by the AI community.

In [27] we also introduced the database counterparts of six rationality *postulates* defined for propositional knowledge bases (in addition to Consistency and Monotony, cf. Definition 1). For instance, *Free-Tuple Independence* states that deleting/adding a free tuple does not change the IM value, while *Penalty* states that deleting a problematic tuple decreases the measure. Thus, IMs satisfying these postulates allow us to check if deleting a given tuple makes a database less inconsistent or not. *Super-Additivity* deals with the union of two disjoint databases, and states that the measure of the union is at least as great as the sum of the measures of the two databases. Thus, an IM satisfying Super-Additivity is able to separately count the inconsistency arising from different sources. Other postulates considered in [27] help in describing the behavior of IMs.

Some postulates violated in the propositional case become satisfied in the database case. Specifically,  $\mathcal{I}_M$  satisfies all the postulates considered in [27]. One step down, we find  $\mathcal{I}_\#$  which satisfies all the postulates but one. Next,  $\mathcal{I}_P$  satisfies all the postulates but two.  $\mathcal{I}_H$  which as said above coincides with  $\mathcal{I}_C$  in our setting also satisfies all the postulates but two, though it

turns out to be computationally hard. Notably  $\mathcal{I}_M$ ,  $\mathcal{I}_\#$ , and  $\mathcal{I}_P$  can be evaluated by standard SQL, meaning that measuring inconsistency using these measures scales as far as the database can answer queries in reasonable time.

Many interesting issues concerning IMs in databases remain unexplored. We have dealt with denial constraints, a common type of integrity constraint which can express for instance equality generating dependencies (e.g., functional dependencies). We plan to extend our work to other types of integrity constraints, and in particular to inclusion dependencies. Also, we plan to identify tractable cases for the hard measures, possibly exploiting connections with work done on inconsistent databases (e.g., results in [34] help in characterizing the complexity of  $\mathcal{I}_A$ ), and devise efficient algorithms and index structures for evaluating IMs. The IMs we have considered work at the tuple-level, without distinguishing inconsistency arising from different attributes, which is another issue we want to address in the future. Moreover, we plan to investigate methods for measuring the degree of inconsistency of query answers by leveraging on our IMs (an recent proposal in this direction is [35]). Finally, another interesting direction for future work is considering incomplete information (e.g., databases with null values).

## References

- [1] M. Arenas, L. E. Bertossi, J. Chomicki, Consistent query answers in inconsistent databases, in: Proc. of Symposium on Principles of Database Systems (PODS), 1999, pp. 68–79.
- [2] M. Calautti, L. Libkin, A. Pieris, An operational approach to consistent query answering, in: Proc. of Symposium on Principles of Database Systems (PODS), 2018, pp. 239–251.
- [3] M. V. Martinez, F. Parisi, A. Pugliese, G. I. Simari, V. S. Subrahmanian, Inconsistency management policies, in: Proc. of KR, 2008, pp. 367–377.
- [4] M. V. Martinez, F. Parisi, A. Pugliese, G. I. Simari, V. S. Subrahmanian, Efficient policy-based inconsistency management in relational knowledge bases, in: Proc. of Int. Conference on Scalable Uncertainty Management (SUM), 2010, pp. 264–277.
- [5] M. V. Martinez, F. Parisi, A. Pugliese, G. I. Simari, V. S. Subrahmanian, Policy-based inconsistency management in relational databases, *Int. J. Approx. Reas.* 55 (2014) 501–528.
- [6] B. Fazzinga, S. Flesca, F. Furfaro, F. Parisi, DART: A data acquisition and repairing tool, in: EDBT Workshop on Inconsistency and Incompleteness in Databases, 2006, pp. 297–317.
- [7] S. Flesca, F. Furfaro, F. Parisi, Preferred database repairs under aggregate constraints, in: Proc. of Int. Conference on Scalable Uncertainty Management (SUM), 2007, pp. 215–229.
- [8] S. Hao, N. Tang, G. Li, J. He, N. Ta, J. Feng, A novel cost-based model for data repairing, *IEEE Trans. Knowl. Data Eng.* 29 (2017) 727–742.
- [9] J. He, E. Veltri, D. Santoro, G. Li, G. Mecca, P. Papotti, N. Tang, Interactive and deterministic data cleaning, in: Proc. of SIGMOD, 2016, pp. 893–907.
- [10] T. Bleifuß, L. Bornemann, D. V. Kalashnikov, F. Naumann, D. Srivastava, DBChEx: Interactive exploration of data and schema change, in: Proc. of CIDR, 2019.
- [11] A. Giuzio, G. Mecca, E. Quintarelli, M. Roveri, D. Santoro, L. Tanca, INDIANA: an interactive system for assisting database exploration, *Inf. Syst.* 83 (2019) 40–56.
- [12] J. Grant, A. Hunter, Measuring consistency gain and information loss in stepwise inconsistency resolution, in: Proc. of ECSQARU, 2011, pp. 362–373.



- [13] J. Grant, A. Hunter, Distance-based measures of inconsistency, in: Proc. of ECSQARU, 2013, pp. 230–241.
- [14] J. Grant, M. V. Martinez, Measuring Inconsistency in Information, College Pub., 2018.
- [15] A. Hunter, S. Konieczny, Measuring inconsistency through minimal inconsistent sets, in: Proc. of Principles of Knowledge Representation and Reasoning (KR), 2008, pp. 358–366.
- [16] K. Knight, Measuring inconsistency, *J. Philosophical Logic* 31 (2002) 77–98.
- [17] L. Fan, M. D. Flood, J. Grant, Measuring inconsistency in bank holding company data, in: Proc. of SIGMOD Workshop on Data Science for Macro-modeling with Financial and Economic Datasets (DSMM), 2019, pp. 5:1–5:5.
- [18] H. Decker, D. Martinenghi, Inconsistency-tolerant integrity checking, *IEEE Trans. Knowl. Data Eng.* 23 (2011) 218–234.
- [19] J. Grant, Classifications for inconsistent theories, *Notre Dame J. Formal Log.* 19 (1978) 435–444.
- [20] M. V. Martinez, A. Pugliese, G. I. Simari, V. S. Subrahmanian, H. Prade, How dirty is your relational database? an axiomatic approach, in: Proc. of ECSQARU, 2007, pp. 103–114.
- [21] H. Decker, Inconsistency-tolerant database repairs and simplified repair checking by measure-based integrity checking, *Trans. L. S. Data Knowl. Cent. Syst.* 34 (2017) 153–183.
- [22] H. Decker, S. Misra, Database inconsistency measures and their applications, in: Proc. of Int. Conf. on Information and Software Technologies (ICIST), 2017, pp. 254–265.
- [23] H. Decker, Measuring database inconsistency, in: J. Grant, M. V. Martinez (Eds.), *Measuring Inconsistency in Information*, College Publications, 2018, pp. 271–311.
- [24] H. Decker, D. Martinenghi, Classifying integrity checking methods with regard to inconsistency tolerance, in: Proc. of Prin. and Pract. of Decl. Prog. (PPDP), 2008, pp. 195–204.
- [25] L. E. Bertossi, Measuring and computing database inconsistency via repairs, in: Proc. of International Conference on Scalable Uncertainty Management (SUM), 2018, pp. 368–372.
- [26] L. E. Bertossi, Repair-based degrees of database inconsistency, in: Proc. of Logic Programming and Nonmonotonic Reasoning (LPNMR), 2019, pp. 195–209.
- [27] F. Parisi, J. Grant, On measuring inconsistency in relational databases with denial constraints, in: Proc. of European Conf. on Artificial Intelligence (ECAI), 2020, pp. 857–864.
- [28] J. Grant, F. Parisi, Measuring inconsistency in a general information space, in: Proc. of Int. Symp. on Foundations of Information and Knowledge Systems (FoIKS), 2020, pp. 140–156.
- [29] J. Grant, F. Parisi, General information spaces: measuring inconsistency, rationality postulates, and complexity, *Ann. Math. Artif. Intell.* (2021).
- [30] C. H. Papadimitriou, *Computational complexity*, Addison-Wesley, 1994.
- [31] J. Grant, J. Minker, Inferences for numerical dependencies, *TCS* 41 (1985) 271–287.
- [32] S. Flesca, F. Furfaro, F. Parisi, Consistency checking and querying in probabilistic databases under integrity constraints, *J. Comput. Syst. Sci.* 80 (2014) 1448–1489.
- [33] M. Thimm, J. P. Wallner, Some complexity results on inconsistency measurement, in: Proc. of Principles of Knowledge Representation and Reasoning (KR), 2016, pp. 114–124.
- [34] E. Livshits, B. Kimelfeld, Counting and enumerating (preferred) database repairs, in: Proc. of Symposium on Principles of Database Systems (PODS), 2017, pp. 289–301.
- [35] O. Issa, A. Bonifati, F. Toumani, Evaluating top-k queries with inconsistency degrees, *Proc. VLDB Endow.* 13 (2020) 2146–2158.