Properties of Inconsistency Measures for Databases

Ester Livshits
esterliv@cs.technion.ac.il
Technion
Haifa, Israel

Rina Kochirgan
rina.k@campus.technion.ac.il
Technion
Haifa, Israel

Segev Tsur
segevtsur@campus.technion.ac.il
Technion
Haifa, Israel

Ihab F. Ilyas
ilyas@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

Benny Kimelfeld
bennyk@cs.technion.ac.il
Technion
Haifa, Israel

Sudeepa Roy
sudeepa@cs.duke.edu
Duke University
Durham, NC, USA

ABSTRACT

How should we quantify the inconsistency of a database that violates integrity constraints? Proper measures are important for various tasks, such as progress indication and action prioritization in cleaning systems, and reliability estimation for new datasets. To choose an appropriate inconsistency measure, it is important to identify the desired properties in the application and understand which of these is guaranteed or at least expected in practice. For example, in some use cases, the inconsistency should reduce if constraints are eliminated; in others, it should be stable and avoid jitters and jumps in reaction to small changes in the database. We embark on a systematic investigation of properties for database inconsistency measures. We investigate a collection of basic measures that have been proposed in the past in both the Knowledge Representation and Database communities, analyze their theoretical properties, and empirically observe their behavior in an experimental study. We also demonstrate how the framework can lead to new inconsistency measures by introducing a new measure that, in contrast to the rest, satisfies all of the properties we consider and can be computed in polynomial time.

CCS CONCEPTS

• **Information systems → Data cleaning**; • **Theory of computation → Incomplete, inconsistent, and uncertain databases**.

KEYWORDS

Inconsistent databases, inconsistency measures, database repairs, data cleaning, integrity constraints

ACM Reference Format:
Ester Livshits, Rina Kochirgan, Segev Tsur, Ihab F. Ilyas, Benny Kimelfeld, and Sudeepa Roy. 2021. Properties of Inconsistency Measures for Databases. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3448016.3457310

1 INTRODUCTION

Inconsistency of databases may arise in a variety of situations for a variety of reasons. Database records may be collected from imprecise sources (social encyclopedias/networks, sensors attached to appliances, cameras, etc.) via imprecise procedures (natural-language processing, signal processing, image analysis, etc.), or be integrated from different sources with conflicting information or formats. Common principled approaches to handling inconsistency consider databases that violate *integrity constraints*, but can nevertheless be *repaired* by means of *operations* that revise the database and resolve inconsistency [3].

Instantiations differ in the supported types of integrity constraints, allowed operations, and optimization goals. The constraints may be Functional Dependencies (FDs) or the more general Equality-Generating Dependencies (EGDs) or, more generally, Denial Constraints (DCs), and they may be referential (foreign-key) constraints or the more general inclusion dependencies [2]. A repair operation can be a *deletion* of a tuple, an *insertion* of a tuple, or an *update* of an attribute value. Operations may be associated with different *costs*, representing levels of trust in data items or extent of impact [42]. Optimization goals can be minimization of the costs incurred [33, 40] or producing the most probable clean database from a distribution [48] that fully (hard constraints) or partially (soft constraints) fixes the violated integrity constraints. Various approaches and systems have been proposed for data cleaning and, in particular, data repairing (e.g., [16, 21, 48] to name a few).

We explore the question of *how to measure database inconsistency*. This question may arise in different situations. A measure of inconsistency can be used for estimating the potential usefulness and cost of incorporating databases for downstream analytics [36]. We can also use an inconsistency measure for implementing a progress bar for data repairing. Indeed, the importance of incorporating progress indicators in interactive systems has been well recognized and studied in the Human-Computer Interaction (HCI) community [14, 28, 29, 45, 51]. Note that an inconsistency measure can be used in any cleaning system, where the measure gives one of several indications of progress (namely those that are associated with integrity constraints). Beyond the indication of progress, a measure of inconsistency can be used for prioritizing and recommending actions in data repairing—address the tuples that have the highest *responsibility* to the inconsistency level (e.g., Shapley value for inconsistency [31, 39, 53]) or the ones that might result in the greatest reduction in inconsistency.

Example measures include the number of violations in the database, the number of tuples involved in violations, and the number of operations needed to reach consistency. To choose an appropriate inconsistency measure for a specific use case, it is important to identify the desired properties of the application and understand which of these is guaranteed or expected in practice. For example, to effectively communicate progress indication in repairing, the measure should feature certain characteristics. While it should react to changes in the database and show progress when the database is "cleaner" with respect to the integrity constraints, it should minimize jitters, jumps and sudden changes in reaction to small changes in the database, and in turn, should well correlate with the "expected waiting time"—an important aspect in machine-user interaction. It should also be computationally tractable in interactive systems. Luo et al. [43] enunciate the importance of these properties, referring to them as "acceptable pacing" and "minimal overhead," respectively, in progress indicators for database queries.

In this paper, we explore desirable properties of inconsistency measures and demonstrate the analysis and design of specific inconsistency measures with respect to these properties. As a guiding principle, we adopt the approach of *rationality postulates* of *inconsistency measures* over knowledge bases that have been investigated in depth by the Knowledge Representation (KR) and Logic communities [23, 26, 30–32, 34, 49]. Yet, the studied measures and postulates fall short of capturing our desiderata for various reasons. First, inconsistency is typically measured over a knowledge base of logical sentences (formulas without free variables). In databases, we reason about *tuples* (facts) and fixed integrity constraints, and inconsistency typically refers to the tuples rather than the constraints (which are treated as exogenous information). In particular, while a collection of sentences, and even a single sentence, might form a contradiction, a set of tuples can be inconsistent only in the presence of integrity constraints. Hence, as recently acknowledged [5, 46], it is of importance to seek inconsistency measures that are closer to database applications. More fundamentally, in order to capture the repairing *process* and corresponding changes to the database, the measures should be aware of the underlying *repairing operations* (e.g., tuple insertion, deletion, or revision).

The following example illustrates the concept of a noisy database and will be used throughout the paper to demonstrate the differences among several inconsistency measures.

Example 1. *We have an Airport database*[1] *with the schema*

Airport(Id, Type, Name, Continent, Country, Municipality)

*and the FDs "*Municipality $\rightarrow$ Continent Country*" and "*Country $\rightarrow$ Continent.*" A clean database D_0 where both FDs hold is given in Figure 1a. Figures 1b and 1c depict two noisy versions of the database that violate one or both FDs: D_1 is obtained from D_0 by modifying four values, and D_2 is obtained from D_0 by modifying three values. The changed values are shown in* **bold**. *Of course, in general, we do not know how the noisy database is generated. An inconsistency measure aims to formally quantify the inconsistency level in a noisy database, given a set of constraints. We will revisit this example to illustrate different measures in Section 3 (Table 1).* □

[1]Simplified version of https://ourairports.com/data/.

To illustrate some of the inconsistency measures, let us consider the case where all constraints are anti-monotonic (i.e., consistency cannot be violated by deleting tuples). For example, FDs and the more general DCs are anti-monotonic, whereas referential key constraints are not. One simple measure is the *drastic measure* $\mathcal{I}_{\mathsf{d}}$, which is 1 if the database is inconsistent, and 0 otherwise [49]. Of course, this measure hardly communicates progress of cleaning, as it does not change until the very end. Clearly, we have $\mathcal{I}_{\mathsf{d}}(D_0) = 0$ while $\mathcal{I}_{\mathsf{d}}(D_1) = \mathcal{I}_{\mathsf{d}}(D_2) = 1$ for the databases of Figure 1; hence, this measure does not differentiate the database D_1 from D_2, motivating the need to study more fine-grained measures.

What about the measure $\mathcal{I}_{\mathsf{P}}$ that counts the *problematic tuples* that participate in *(minimal) witnesses of inconsistency* [30, 31]? This measure suffers from a disproportional reaction to repairing operations: the removal of a single tuple (e.g., a misspelling of a country name) may cause a drastic reduction in inconsistency (for example, if a single tuple is involved in most violations). The measure $\mathcal{I}_{\mathsf{MI}}$, that simply counts the constraint violations (i.e., the minimal inconsistent subsets), suffers from the same problem. As another example, consider the measure $\mathcal{I}_{\mathsf{MC}}$ that counts the *maximal consistent subsets*. This measure suffers from various problems: adding constraints can cause a *reduction* in inconsistency, it may fail to reflect repairing operations (i.e., the number of maximal consistent subsets may stay the same after applying an operation), and, again, it may react disproportionally to small changes. Moreover, it is hard to compute (#P-complete) already for simple FDs [38].

A recent effort in the database community proposed measures based on the concept of a *minimal repair*—the minimal number of repairing operations needed to obtain consistency [6]. We refer to this measure as $\mathcal{I}_{\mathcal{R}}$. We show that $\mathcal{I}_{\mathcal{R}}$ satisfies the aforementioned rationality criteria that we formally define later on, and so, we provide a formal justification to its adoption. Yet, it is again intractable (NP-hard) even for simple sets of FDs [33, 40]. Interestingly, we are able to show that a *linear relaxation* of this measure, that we propose in this paper as a new inconsistency measure and refer to as $\mathcal{I}_{\mathcal{R}}^{\mathsf{lin}}$, provides a combination of rationality and tractability.

Contributions. Given the numerous choices of inconsistency measures, we ask and address the question of what properties are desired in such a measure, so that one can make an informed choice. In particular, we make the following contributions.

(1) Formalize and analyze desired properties of inconsistency measures: We define four properties of inconsistency measures in the context of a *repair system* (a space of possibly weighted repairing operations) with the following intuitive meaning:

(i) **Positivity**: the measure is strictly positive if and only if the database is inconsistent,
(ii) **Monotonicity**: inconsistency cannot be reduced if the constraints get stricter,
(iii) **Continuity**: a single operation can have a limited relative impact on inconsistency, and
(iv) **Progression**: we can always find an operation that reduces inconsistency.

Moreover, we use these properties to analyze a collection of inconsistency measures adapted from the KR and Logic literature (Section 3). Specifically, we examine the inconsistency measures against the properties, and show that most of the measures violate

	Id	Type	Name	Continent	Country	Municipality
f_1	00AA	Small airport	Aero B Ranch	NAm	US	Leoti
f_2	7FA0	heliport	Florida Keys Memorial Hospital Heliport	NAm	US	Key West
f_3	7FA1	Small airport	Sugar Loaf Shores Airport	NAm	US	Key West
f_4	KEYW	Medium airport	Key West International Airport	NAm	US	Key West
f_5	KNQX	Medium airport	Naval Air Station Key West/Boca Chica Field	NAm	US	Key West

(a) A clean database D_0.

	⋯	Continent	Country	⋯
f_1		NAm	US	
f_2		**Am**	**USA**	
f_3	⋯	NAm	US	⋯
f_4		NAm	**USA**	
f_5		**Am**	US	

(b) A noisy database D_1.

	⋯	Continent	Country	⋯
f_1		NAm	US	
f_2		**Am**	**USA**	
f_3	⋯	NAm	US	⋯
f_4		NAm	**USA**	
f_5		NAm	US	

(c) A noisy database D_2.

Figure 1: A clean and two noisy Airport databases from Example 1. Missing attributes in D_1 and D_2 are unchanged compared to D_0. Bold values are changed from the clean database. The underlines are explained in Section 3.

at least one of these properties (see Table 2). However, the measure $\mathcal{I}_{\mathcal{R}}$ stands out satisfying all the properties (Section 4).

(2) Computational Complexity: As an additional desideratum for practical purposes, we consider the complexity of computing the measures. While some measures are tractable, the measure $\mathcal{I}_{\mathcal{R}}$, that satisfies all the properties, is known to be intractable already for FDs [40]. We provide a stronger result showing that computing $\mathcal{I}_{\mathcal{R}}$ is hard already for the case of a single EGD (Section 5.1).

(3) Propose a new measure satisfying all desiderata: For tuple deletions as repairing operations, we propose a new measure $\mathcal{I}_{\mathcal{R}}^{\mathsf{lin}}$ relaxing the concept of $\mathcal{I}_{\mathcal{R}}$. We show that $\mathcal{I}_{\mathcal{R}}^{\mathsf{lin}}$ satisfies all four properties and is computable in polynomial time, even for the general case of arbitrary sets of denial constraints; hence, it provides a combination of rationality and tractability (Section 5).

(4) Empirical evaluation of measures: We empirically evaluate the behavior of these measures on multiple datasets with different integrity constraints and inconsistency levels (Section 6). We show that the measures that behave well in theory also exhibit a good empirical behavior under different repair models.

Other related work. As mentioned earlier, inconsistency measures were extensively studied in the KR community [49]. Rationality postulates for inconsistency measures have been studied in the database context, for example by Parisi and Grant [46]. Contrasting with that work, we devise properties that account for operational aspects of repairs and repair systems, and we study the computational complexity of the measures. The properties and complexity analysis lead us to tractable and rational measures that have not been considered in the past (e.g., $\mathcal{I}_{\mathcal{R}}^{\mathsf{lin}}$). Also contrasting with that work, we conduct a thorough empirical evaluation over candidate measures to explore their behavior in practice; to the best of our knowledge, we are the first to conduct such an experimental study.

Some other relevant studies have focused on different aspects of inconsistency measures for databases. Martinez et al. [44] introduced axioms that are inherently based on a numerical interpretation of the database values. Grant and Hunter [24] considered repairing (or *resolution*) operators, and focused on the trade-off between inconsistency reduction and *information loss*: an operation is beneficial if it causes a high reduction in inconsistency alongside a low loss of information. Another complementary problem is that of associating *individual facts* with portions of the database inconsistency (e.g., the Shapley value of the fact) and using these portions to define preferences among repairs [39, 53].

In the bigger context, note that inconsistency measures quantify the extent to which constraints are violated. Data cleaning in general goes beyond constraint resolution to challenges such as outlier detection, record linkage, entity resolution and so on (see, e.g., [1] and references therein). Yet, one can use an inconsistency measure in any cleaning system, adopting any correction mechanism whatsoever, for providing a specific view of the progress: the extent of constraint violation (but not necessarily general dirt). Measuring the status for other kinds of dirt is an important future challenge that goes beyond the scope of this paper.

The remainder of the paper is organized as follows. We present preliminary concepts and terminology in Section 2. We consider inconsistency measures in Section 3 and their properties in Section 4. In Section 5, we discuss complexity aspects and propose a new rational and tractable measure. We describe an experimental evaluation in Section 6. Finally, we make concluding remarks and discuss directions for future work in Section 7.

2 PRELIMINARIES

We first give the basic terminology and concepts.

Relational model. A *relational schema* (or just *schema* for short) $\mathbf{S}$ has a finite set of *relation symbols* R, each associated with a *relation signature* $sig(R)$. In turn, a relation signature α is a sequence $(A_1, \dots, A_k)$ of distinct *attributes* A_i, and k is the *arity* of α. If $sig(R)$ has arity k, then we say that R is k-ary. A *fact* f (over $\mathbf{S}$) is an expression of the form $R(c_1, \dots, c_k)$, where R is a k-ary relation symbol of $\mathbf{S}$, and $c_1, \dots, c_k$ are *values*. If $f = R(c_1, \dots, c_k)$ is a fact and $sig(R) = (A_1, \dots, A_k)$, then we refer to the value c_i as $f.A_i$.

A *database* D over $\mathbf{S}$ is a mapping from a finite set $ids(D)$ of *record identifiers* to facts over $\mathbf{S}$. The set of all databases over $\mathbf{S}$ is denoted by $\mathrm{DB}(\mathbf{S})$. We denote by $D[i]$ the fact that D maps to the identifier i. A database D is a *subset* of a database D', denoted $D \subseteq D'$, if $ids(D) \subseteq ids(D')$ and $D[i] = D'[i]$ for all $i \in ids(D)$.

An *integrity constraint* is a first-order sentence over **S**. A database D satisfies a set Σ of integrity constraints, denoted $D \models \Sigma$, if D satisfies every constraint $\sigma \in \Sigma$. If Σ and Σ' are sets of integrity constraints, then we write $\Sigma \models \Sigma'$ to denote that Σ *entails* Σ'; that is, every database that satisfies Σ also satisfies Σ'. We also write $\Sigma \equiv \Sigma'$ to denote that Σ and Σ' are *equivalent*, that is, $\Sigma \models \Sigma'$ and $\Sigma' \models \Sigma$. By a *constraint system* we refer to a class **C** of integrity constraints (e.g., the class of all functional dependencies).

As a special case, a *Functional Dependency* (FD) $R : X \to Y$, where R is a relation symbol and $X, Y \subseteq sig(R)$, states that every two facts that agree on (i.e., have equal values in) every attribute of X should also agree on Y. The more general *Equality Generating Dependency* (EGD) has the form $\forall \vec{x}\, [\varphi_1(\vec{x}) \wedge \cdots \wedge \varphi_k(\vec{x}) \to (y_1 = y_2)]$ where each $\varphi_j(\vec{x})$ is an atomic formula over the schema and y_1 and y_2 are variables in $\vec{x}$. Finally, a *Denial Constraint* (DC) has the form $\forall \vec{x} \neg\, [\varphi_1(\vec{x}) \wedge \cdots \wedge \varphi_k(\vec{x}) \wedge \psi(\vec{x})]$ where each $\varphi_j(\vec{x})$ is an atomic formula and $\psi(\vec{x})$ is a conjunction of atomic comparisons over $\vec{x}$.

Example 2. *The schema of our running example consists of a single 5-ary relation symbol Airport. All databases of Figure 1 have five facts with $ids(D) = \{f_1, \cdots, f_5\}$. The constraint set Σ consists of two FDs as shown in Example 1.*

Repair systems. Let **S** be a schema. A *repairing operation* (or just *operation*) o transforms a database D over **S** into another database $o(D)$ over **S**, that is, $o : \mathrm{DB}(\mathbf{S}) \to \mathrm{DB}(\mathbf{S})$. An example is *tuple deletion*, denoted $\langle -i \rangle(\cdot)$, parameterized by a tuple identifier i and applicable to D if $i \in ids(D)$; the result $\langle -i \rangle(D)$ is obtained from D by deleting the tuple identifier i (along with the corresponding fact $D[i]$). Another example is *tuple insertion*, denoted $\langle +f \rangle(\cdot)$, parameterized by a fact f; the result $\langle +f \rangle(D)$ is obtained from D by adding f with a new tuple identifier. (For convenience, this is the minimal integer i such that $i \notin ids(D)$.) A third example is *attribute update*, denoted $\langle i.A \leftarrow c \rangle(\cdot)$, parameterized by a tuple identifier i, an attribute A, and a value c, and applicable to D if $i \in ids(D)$ and A is an attribute of the fact $D[i]$; the result $\langle i.A \leftarrow c \rangle(D)$ is obtained from D by setting $D[i].A$ to c. By convention, if o is not applicable to D, then it keeps D intact, that is, $o(D) = D$.

A *repair system* (over a schema **S**) is a collection of repairing operations with an associated *cost* of applying to a given database. For example, a smaller change of value might be less costly than a greater one [20], and some facts might be more costly than others to delete [40, 42] or update [7, 33, 40]; changing a person's zip code might be less costly than changing the person's country, which, in turn, might be less costly than deleting the entire person's record. Formally, a repair system $\mathcal{R}$ is a pair (O, κ) where O is a set of operations and $\kappa : O \times \mathrm{DB}(\mathbf{S}) \to [0, \infty)$ is a cost function that assigns the cost $\kappa(o, D)$ to applying o to D. We require that $\kappa(o, D) = 0$ if and only if $D = o(D)$; that is, the cost is nonzero when, and only when, an actual change occurs.

For a repair system $\mathcal{R}$, we denote by $\mathcal{R}^\star = (O^\star, \kappa^\star)$ the repair system of all *sequences* of operations from $\mathcal{R}$, where the cost of a sequence is the sum of costs of the individual operations thereof. Let **C** be a constraint system and $\mathcal{R}$ a repair system. We say that **C** is *realizable by* $\mathcal{R}$ if it is always possible to make a database satisfy the constraints of **C** by repeatedly applying operations from $\mathcal{R}$. Formally, **C** is realizable by $\mathcal{R}$ if for every database D and a finite set $\Sigma \subseteq \mathbf{C}$ there is a sequence o in $\mathcal{R}^\star$ such that $o(D) \models \Sigma$. An example of **C** is the system $\mathbf{C}_{\mathsf{FD}}$ of all FDs $R : X \to Y$. An example of $\mathcal{R}$ is the *subset* system, denoted $\mathcal{R}_\subseteq$, where O is the set of all tuple deletions (hence, the result is always a subset of the original database), and κ is determined by a special *cost* attribute, $\kappa(\langle -i \rangle(D)) = D[i].\mathtt{cost}$, if a cost attribute exists, and otherwise, $\kappa(\langle -i \rangle(D)) = 1$ (every tuple has unit cost for deletion). Observe that $\mathcal{R}_\subseteq$ realizes **C**, since the latter consists of anti-monotonic constraints.

Example 3. *The database D_1 of Figure 1b may be obtained from D_0 by applying the following sequence of attribute updates (here, we assume that the identifier of a fact f_i is i):*

$$E_1 = \langle 2.\text{Continent} \leftarrow \mathtt{Am} \rangle(D_0) \qquad E_2 = \langle 2.\text{Country} \leftarrow \mathtt{USA} \rangle(E_1)$$
$$E_3 = \langle 4.\text{Country} \leftarrow \mathtt{USA} \rangle(E_2) \qquad D_1 = \langle 5.\text{Continent} \leftarrow \mathtt{Am} \rangle(E_3)$$

If we consider tuple deletions and insertions, we may obtain D_1 from D_0 by applying the following sequence of operations:

$$E_1 = \langle -2 \rangle(D_0) \qquad E_2 = \langle +f_2 \rangle(E_1) \qquad E_3 = \langle -4 \rangle(E_2)$$
$$E_4 = \langle +f_4 \rangle(E_3) \qquad E_5 = \langle -5 \rangle(E_4) \qquad D_1 = \langle +f_5 \rangle(E_5)$$

where f_2, f_4 and f_5 are the facts of D_1. We may also assign a cost to each operation. For example, if both tuple deletions and attribute updates are allowed, we may associate a higher cost with the operation $\langle -4 \rangle(D)$ that deletes an entire fact compared to the operation $\langle 4.\text{Country} \leftarrow \mathtt{USA} \rangle(D)$ that updates a single attribute value.

3 INCONSISTENCY MEASURES

Let **S** be a schema and **C** a constraint system. An *inconsistency measure* is a function $\mathcal{I}$ that maps a finite set $\Sigma \subseteq \mathbf{C}$ of integrity constraints and a database D to a number $\mathcal{I}(\Sigma, D) \in [0, \infty)$. Intuitively, a high $\mathcal{I}(\Sigma, D)$ implies that D is far from satisfying Σ. We make two standard requirements:

- $\mathcal{I}$ is zero on consistent databases; that is, $\mathcal{I}(\Sigma, D) = 0$ whenever $D \models \Sigma$;
- $\mathcal{I}$ is invariant under logical equivalence of constraints; that is, $\mathcal{I}(\Sigma, D) = \mathcal{I}(\Sigma', D)$ whenever $\Sigma \equiv \Sigma'$.

Next, we discuss several examples of inconsistency measures. Some of these measures (namely, $\mathcal{I}_{\mathsf{d}}$, $\mathcal{I}_{\mathsf{MI}}$, $\mathcal{I}_{\mathsf{P}}$ and $\mathcal{I}_{\mathsf{MC}}$) are adapted from the survey of Thimm [49] to the context of relational databases.

The simplest measure is the *drastic inconsistency value*, denoted $\mathcal{I}_{\mathsf{d}}$, which is the indicator function of inconsistency.

$$\mathcal{I}_{\mathsf{d}}(\Sigma, D) := \begin{cases} 0 & \text{if } D \models \Sigma; \\ 1 & \text{otherwise.} \end{cases}$$

In Figure 1, we have that $D_0 \models \Sigma$, while $D_1, D_2 \not\models \Sigma$. Therefore, $\mathcal{I}_{\mathsf{d}}(\Sigma, D_0) = 0$ and $\mathcal{I}_{\mathsf{d}}(\Sigma, D_1) = \mathcal{I}_{\mathsf{d}}(\Sigma, D_2) = 1$

The following measures, $\mathcal{I}_{\mathsf{MI}}$, $\mathcal{I}_{\mathsf{P}}$ and $\mathcal{I}_{\mathsf{MC}}$, apply to systems **C** of *anti-monotonic* constraints. Recall that an integrity constraint σ is anti-monotonic if for all databases D and D', if $D \subseteq D'$ and $D' \models \sigma$, then $D \models \sigma$. Examples of anti-monotonic constraints are the DCs [19], the classic FDs, *conditional FDs* [9], and EGDs [4].

For a set $\Sigma \subseteq \mathbf{C}$ of anti-monotonic constraints and a database D, denote by $\mathsf{MI}_\Sigma(D)$ the set of all *minimal inconsistent* subsets of D; that is, the set of all $E \subseteq D$ such that $E \not\models \Sigma$ and $E' \models \Sigma$ for all proper subsets $E' \subsetneq E$. Since the constraints are anti-monotonic, it holds that $D \models \Sigma$ if and only if $\mathsf{MI}_\Sigma(D)$ is empty. Drawing from

Table 1: The inconsistency measure values on the databases of our running example.

	Noisy database D_1		Noisy database D_2	
Measure	**value**	**explanation**	**value**	**explanation**
$\mathcal{I}_{\mathsf{d}}$	1	inconsistent database	1	inconsistent database
$\mathcal{I}_{\mathcal{R}}$ (deletions)	3	e.g., remove $\{f_2, f_4, f_5\}$ or $\{f_3, f_4, f_5\}$	2	e.g., remove $\{f_2, f_3\}$ or $\{f_2, f_4\}$
$\mathcal{I}_{\mathcal{R}}$ (updates)	4	change the values shown in **bold** or with an underline	3	change the values shown in **bold**
$\mathcal{I}_{\mathsf{MI}}$	7	$\lvert\{\{f_2, f_3\}, \{f_2, f_4\}, \{f_2, f_5\}, \{f_3, f_4\}, \{f_3, f_5\}, \{f_4, f_5\}, \{f_1, f_5\}\}\rvert$	5	$\lvert\{f_2, f_3\}, \{f_2, f_4\}, \{f_2, f_5\}, \{f_3, f_4\}, \{f_4, f_5\}\}\rvert$
$\mathcal{I}_{\mathsf{P}}$	5	all tuples are involved in violations	4	all tuples except f_1 are involved in violations
$\mathcal{I}_{\mathsf{MC}}$	3	$\lvert\{\{f_1, f_2\}, \{f_1, f_3\}, \{f_1, f_4\}, \{f_5\}\}\rvert - 1$	2	$\lvert\{\{f_1, f_2\}, \{f_1, f_4\}, \{f_1, f_3, f_5\}\}\rvert - 1$
$\mathcal{I}_{\mathcal{R}}^{\mathsf{lin}}$	2.5	e.g., assign 0.5 to all $f_1, \cdots, f_5$ (see Section 5.2)	2	e.g., assign 0.5 to all $f_2, \cdots, f_5$ (see Section 5.2)

known inconsistency measures [30, 31, 49], the measure $\mathcal{I}_{\mathsf{MI}}$, also known as *MI Shapley Inconsistency*, is the cardinality of this set.

$$\mathcal{I}_{\mathsf{MI}}(\Sigma, D) := |\mathsf{MI}_\Sigma(D)|$$

A fact f that belongs to a minimal inconsistent subset is called *problematic*, and the measure $\mathcal{I}_{\mathsf{P}}$ counts the problematic facts [24].

$$\mathcal{I}_{\mathsf{P}}(\Sigma, D) := |\cup \mathsf{MI}_\Sigma(D)|$$

Example 4. *The set Σ of constraints of Example 1 consists of FDs, which are violated by pairs of facts; thus, the size of any minimal inconsistent subset w.r.t. Σ is two. Since the database D_0 of Figure 1 satisfies Σ, we have that $\mathcal{I}_{\mathsf{MI}}(\Sigma, D_0) = \mathcal{I}_{\mathsf{P}}(\Sigma, D_0) = 0$. In D_1, all (six) pairs of facts from $\{f_2, f_3, f_4, f_5\}$ as well as the pair $\{f_1, f_5\}$ jointly violate Σ; hence, there are seven violating pairs in total and $\mathcal{I}_{\mathsf{MI}}(\Sigma, D_1) = 7$. As every fact of D_1 occurs in at least one violating pair, we have that $\mathcal{I}_{\mathsf{P}}(\Sigma, D_1) = 5$. Similarly, $\mathcal{I}_{\mathsf{MI}}(\Sigma, D_2) = 5$ as shown in Table 1, and $\mathcal{I}_{\mathsf{P}}(\Sigma, D_2) = 4$ as all facts of D_2 except f_1 are involved in violations.*

For a set $\Sigma \subseteq \mathbf{C}$ of anti-monotonic constraints and a database D, we denote by $\mathsf{MC}_\Sigma(D)$ the set of all *maximal consistent* subsets of D; that is, all $E \subseteq D$ such that $E \models \Sigma$ and, moreover, $E' \not\models \Sigma$ whenever $E \subsetneq E' \subseteq D$. Observe that if $D \models \Sigma$, then $\mathsf{MC}_\Sigma(D)$ is simply the singleton $\{D\}$, and for anti-monotonic constraints, the set $\mathsf{MC}_\Sigma(D)$ is never empty (since, e.g., the empty set is consistent). The measure $\mathcal{I}_{\mathsf{MC}}$ is the cardinality of $\mathsf{MC}_\Sigma(D)$, minus one.

$$\mathcal{I}_{\mathsf{MC}}(\Sigma, D) := |\mathsf{MC}_\Sigma(D)| - 1$$

We note that the definition of $\mathcal{I}_{\mathsf{MC}}$ in the KR setting is slightly different than ours, as it takes into account the number of self-contradictions in the knowledge base [24, 26]. This highlights an important difference between the standard KR setting and ours. As aforementioned, we distinguish between tuples and integrity constraints, and we measure only tuple sets; thus, no self-contradictions exist per se—an individual tuple is always consistent. However, a tuple can be self-inconsistent in the presence of a relevant constraint (e.g., the DC "height > 0"). Parisi and Grant [46] refer to such tuples as *contradictory* tuples. Hence, we also consider here the following variant of $\mathcal{I}_{\mathsf{MC}}$ that counts *self-inconsistencies*.

$$\mathcal{I}'_{\mathsf{MC}}(\Sigma, D) := |\mathsf{MC}_\Sigma(D)| + |\mathsf{SelfInconsistencies}(D)| - 1$$

Example 5. *In Figure 1, it holds that $\mathcal{I}_{\mathsf{MC}}(\Sigma, D_0) = 0$ as D_0 is consistent and $\mathsf{MC}_\Sigma(D_0) = \{D_0\}$. The $\mathcal{I}_{\mathsf{MC}}$ values for D_1 and D_2 are shown in Table 1 along with the maximal consistent subsets. For instance, for D_1, the set $\{f_1, f_2\}$ is a maximal consistent subset as adding f_3 introduces a violation of the FD* Municipality $\rightarrow$ Continent Country, *and the addition of f_4 or f_5 introduces violations of both FD. As we consider FDs, for which there are no self-inconsistencies, for D_0, D_1, D_2, the $\mathcal{I}'_{\mathsf{MC}}$ and $\mathcal{I}_{\mathsf{MC}}$ values coincide.*

All the inconsistency measures considered so far are adapted from the KR literature, and only take the database and set of integrity constraints into account. However, in the context of databases, it is also important to consider the *repair system* that aims to fix an inconsistent database via repairing operations (e.g., value updates or tuple deletions). The next measure assumes an underlying repair system $\mathcal{R}$ and constraint system $\mathbf{C}$ such that $\mathbf{C}$ is realizable by $\mathcal{R}$.

The measure $\mathcal{I}_{\mathcal{R}}$ is the minimal cost of a sequence of operations that repairs the database. It captures the intuition around various notions of repairs known as *cardinality* repairs and *optimal* repairs [2, 33, 40].

$$\mathcal{I}_{\mathcal{R}}(\Sigma, D) := \min\{\kappa^\star(o, D) \mid o \in O^\star \text{ and } o(D) \models \Sigma\}$$

This measure is the same as the *d-hit* inconsistency measure introduced by Grant and Hunter [25]. Moreover, $\mathcal{I}_{\mathcal{R}}$ is the *distance from satisfaction* used in property testing [22] in the special case where the repair system consists of unit-cost insertions and deletions.

While many of the rule-based approaches to error detection and repairing in the database literature have considered anti-monotonic constraints like DCs, EGDs, and FDs [3, 12, 13, 17, 48, 52], the measure $\mathcal{I}_{\mathcal{R}}$ in general can be used with other types of constraints (like referential integrity constraints or other complex global constraints that are not anti-monotonic). This measure could also naturally incorporate weighted (soft) rules [10].

Example 6. *The values of $\mathcal{I}_{\mathcal{R}}$ on the two noisy databases of Figure 1 are given in Table 1 for the cases where the repairing operations are tuple deletions or attribute updates. As mentioned in Example 4, all pairs of facts from $\{f_2, f_3, f_4, f_5\}$ in D_1 jointly violate Σ; thus, only one of these facts may appear in a repair, and the minimal number of facts that we need to remove from the database to satisfy Σ is three (e.g., remove $\{f_2, f_4, f_5\}$). Assuming unit-cost deletions, we have that $\mathcal{I}_{\mathcal{R}}(\Sigma, D_1) = 3$ for the repair system $\mathcal{R}_\subseteq$. If we consider attribute updates, we need to update at least every bold (or underlined) value in D_1 to satisfy the FD* Municipality $\rightarrow$ Continent Country*; hence, in this case, $\mathcal{I}_{\mathcal{R}}(\Sigma, D_1) = 4$ (again, for the case of unit-cost updates).*

4 PROPERTIES OF MEASURES

We now propose and discuss several properties (*postulates*) of database inconsistency measures. We illustrate these properties over the measures of the previous section. In our examples throughout the section, we focus on the case where the constraints are FDs or the

Table 2: Satisfaction of properties for $\mathsf{C}_{\mathsf{FD}}/\mathsf{C}_{\mathsf{DC}}$ and $\mathcal{R}_{\subseteq}$ (e.g., $\mathcal{I}_{\mathsf{MI}}$ satisfies monotonicity for FDs but not DCs). Tractability ("PTime" assuming P ≠ NP) is discussed in Section 5, where we also define the measure $\mathcal{I}_{\mathcal{R}}^{\mathsf{lin}}$ in the last row.

	Pos.	Mono.	B. Cont.	Prog.	PTime
$\mathcal{I}_{\mathsf{d}}$	✓/✓	✓/✓	✗/✗	✗/✗	✓/✓
$\mathcal{I}_{\mathsf{MI}}$	✓/✓	✓/✗	✗/✗	✓/✓	✓/✓
$\mathcal{I}_{\mathsf{P}}$	✓/✓	✓/✗	✗/✗	✓/✓	✓/✓
$\mathcal{I}_{\mathsf{MC}}$	✓/✗	✗/✗	✓/✓	✗/✗	✗/✗
$\mathcal{I}'_{\mathsf{MC}}$	✓/✓	✗/✗	✗/✗	✗/✗	✗/✗
$\mathcal{I}_{\mathcal{R}}$	✓/✓	✓/✓	✓/✓	✓/✓	✗/✗
$\mathcal{I}_{\mathcal{R}}^{\mathsf{lin}}$	✓/✓	✓/✓	✓/✓	✓/✓	✓/✓

more general DCs, and the repair system is the subset system $\mathcal{R}_{\subseteq}$ where only tuple deletions are allowed. We stress, however, that these inconsistency measures can be used under *any* repair system and constraint system (perhaps without the theoretical guarantees), and we illustrate that in our experimental study (Section 6).

Positivity. A basic property is *positivity*, sometimes referred to as *consistency* [26, 46]. This property is also the first axiom suggested by Martinez et al. [44].

Positivity: $\mathcal{I}(\Sigma, D) > 0$ whenever $D \not\models \Sigma$.

Each of $\mathcal{I}_{\mathsf{d}}$, $\mathcal{I}_{\mathsf{MI}}$, $\mathcal{I}_{\mathsf{P}}$, $\mathcal{I}'_{\mathsf{MC}}$, and $\mathcal{I}_{\mathcal{R}}$ satisfies positivity (for any set of anti-monotonic constraints), but not $\mathcal{I}_{\mathsf{MC}}$. For example, let D be a database with of two facts, $R(a)$ and $R(b)$, and Σ consists of the (denial) constraint $\neg R(a)$ (i.e., $R(a)$ is not in the database). Then $\mathcal{I}_{\mathsf{MC}}(\Sigma, D) = 0$ since $\mathsf{MC}_{\Sigma}(D) = \{R(b)\}$. Observe that $R(a)$ is inconsistent by itself; hence, the example does not apply to $\mathcal{I}'_{\mathsf{MC}}$ and $\mathcal{I}'_{\mathsf{MC}}(\Sigma, D) = 1$. Yet, in the case of FDs, every violation involves two facts, and so $|\mathsf{MC}_{\Sigma}(D)| \geq 2$ and positivity holds.

Monotonicity. The next property is *monotonicity*—inconsistency cannot decrease if the constraints get stricter.

Monotonicity: $\mathcal{I}(\Sigma, D) \leq I(\Sigma', D)$ whenever $\Sigma' \models \Sigma$.

For example, $\mathcal{I}_{\mathsf{d}}$ and $\mathcal{I}_{\mathcal{R}}$ satisfy monotonicity for all anti-monotonic constraints, since a repair w.r.t. Σ' is also a repair w.r.t. Σ. The measures $\mathcal{I}_{\mathsf{MI}}$ and $\mathcal{I}_{\mathsf{P}}$ satisfy monotonicity for FDs, since in this case $|\mathsf{MI}_{\Sigma}(D)|$ is the number of fact pairs that jointly violate an FD, which can only increase when adding or strengthening FDs. Yet, they may violate monotonicity for the more general class of DCs.

Proposition 1. *In the case of $\mathcal{I}_{\mathsf{MI}}$ and $\mathcal{I}_{\mathsf{P}}$, monotonicity can be violated already for the class of DCs.*

Proof. We begin with $\mathcal{I}_{\mathsf{MI}}$. Consider a schema with a single relation symbol, and for a natural number $k > 0$, let Σ_k consist of a single DC stating that *there are at most $k-1$ facts in the database.* (The reader can easily verify that, indeed, Σ_k can be expressed as a DC.) Then, $\mathcal{I}_{\mathsf{MI}}(\Sigma_k, D) = \binom{n}{k}$ whenever D has $n \geq k$ facts. In particular, whenever $k' > k$ and D has $n \geq 2k'$ facts, it holds that $\mathcal{I}_{\mathsf{MI}}(\Sigma_{k'}, D) > \mathcal{I}_{\mathsf{MI}}(\Sigma_k, D)$ while $\Sigma_k \models \Sigma_{k'}$.

We now consider the measure $\mathcal{I}_{\mathsf{P}}$. Let $\mathbf{S}$ be a schema that contains two relation symbols $R(A, B)$ and $S(A, B)$. Consider the following two EGDs (which are, of course, special cases of DCs):

$$\sigma_1 = \forall x, y, z, w\big[\big(R(x, y), S(x, z), S(x, w)\big) \Rightarrow z = w\big]$$

$$\sigma_2 = \forall x, z, w\big[\big(S(x, z), S(x, w)\big) \Rightarrow z = w\big]$$

Let $\Sigma_1 = \{\sigma_1\}$ and $\Sigma_2 = \{\sigma_1, \sigma_2\}$. Every set in $\mathsf{MI}_{\Sigma_1}(D)$ is of size three, while the size of the sets in $\mathsf{MI}_{\Sigma_2}(D)$ is two. Hence, in a database where $|\mathsf{MI}_{\Sigma_1}(D)| = |\mathsf{MI}_{\Sigma_2}(D)|$ (i.e., where σ_1 is violated by $\{R(\mathsf{a}, \mathsf{b}), S(\mathsf{a}, \mathsf{c}), S(\mathsf{a}, \mathsf{d})\}$ if and only if σ_2 is violated by $\{S(\mathsf{a}, \mathsf{c}), S(\mathsf{a}, \mathsf{d})\}$), we have $|P_{\Sigma_1}(D)| > |P_{\Sigma_2}(D)|$ while $\Sigma_2 \models \Sigma_1$. □

The measures $\mathcal{I}_{\mathsf{MC}}$ and $\mathcal{I}'_{\mathsf{MC}}$, on the other hand, can violate monotonicity even for FDs (hence, also for DCs).

Proposition 2. *In the case of $\mathcal{I}_{\mathsf{MC}}$ and $\mathcal{I}'_{\mathsf{MC}}$, monotonicity can be violated already for the class of FDs.*

Proof. Let D consist of these facts over $R(A, B, C, D)$:

$$f_1 = R(0,0,0,0) \; f_2 = R(1,0,0,0) f_3 = R(1,1,0,1) \; f_4 = R(0,1,0,1)$$

Let $\Sigma_1 = \{A \to B\}$, $\Sigma_2 = \{A \to B, C \to D\}$. Then $\Sigma_2 \models \Sigma_1$ and:

$$\mathsf{MC}(\Sigma_1, D) = \{\{f_1, f_2\}, \{f_1, f_3\}, \{f_2, f_4\}, \{f_3, f_4\}\}$$

$$\mathsf{MC}(\Sigma_2, D) = \{\{f_1, f_2\}, \{f_3, f_4\}\}$$

We conclude that $\mathcal{I}_{\mathsf{MC}}(\Sigma_1, D) = \mathcal{I}'_{\mathsf{MC}}(\Sigma_1, D) = 3$ and $\mathcal{I}_{\mathsf{MC}}(\Sigma_2, D) = \mathcal{I}'_{\mathsf{MC}}(\Sigma_2, D) = 1$, proving that monotonicity is violated. □

Note that our monotonicity definition is different from that of Parisi and Grant [46], as monotonicity in our case is restricted to the integrity constraints rather than the database. Here, we do not consider a property for monotonicity over the database (i.e., if $D \subseteq D'$ then $\mathcal{I}(D) \leq \mathcal{I}(D')$), as adding a new tuple may, in fact, reduce inconsistency, for example, under foreign-key constraints.

Positivity and monotonicity serve as sanity conditions that the measure indeed quantifies inconsistency—it does not ignore inconsistency, and it does not reward strictness of constraints. Next, we propose two properties that are aware of the underlying repair system $\mathcal{R} = (O, \kappa)$ as a model of operations. They are inspired by what Luo et al. [43] state informally as "acceptable pacing" and "continuously revised estimates." The first, *continuity*, limits the ability of an operation to have a drastic effect, and the second, *progression*, states that the measure is reactive and not indifferent to operations.

Bounded Continuity. *Continuity* means that, intuitively speaking, repairing operations cannot cause disproportional changes to the database. More formally, it is parameterized by a number $\delta \geq 1$ and it states that, for every two databases D_1 and D_2, and for each operation o_1 on D_1 we can find an operation o_2 on D_2 that is (almost) at least as impactful as o_1, as it reduces inconsistency by at least $1/\delta$ of what o_1 does in D_1. This property is important in reliability estimation, for instance, where one wishes to avoid a situation where the database is deemed highly inconsistent and, yet, a small change can make it considerably more consistent. It is also important in progress indication, where it limits unexpected jumps and changes. In what follows, we denote by $\Delta_{\mathcal{I},\Sigma}(o_1, D_1)$ the value $\mathcal{I}(\Sigma, D_1) - \mathcal{I}(\Sigma, o_1(D_1))$.

δ-continuity: For all Σ, D_1, D_2 and $o_1 \in O$, there exists $o_2 \in O$ such that $\Delta_{\mathcal{I},\Sigma}(o_2, D_2) \geq \Delta_{\mathcal{I},\Sigma}(o_1, D_1)/\delta$.

This definition can be extended to the case where the measure is aware of the cost of operations in the repair system $\mathcal{R}$. There, the change is relative to the cost of the operation. That is, we define the weighted version of δ-continuity in the following way.

Weighted δ-continuity: For all Σ, D_1, D_2 and $o_1 \in O$, there exists $o_2 \in O$ such that $\frac{\Delta_{\mathcal{I},\Sigma}(o_2, D_2)}{\kappa(o_2, D_2)} \geq \frac{\Delta_{\mathcal{I},\Sigma}(o_1, D_1)}{\delta \cdot \kappa(o_1, D_1)}$.

We say that a measure $\mathcal{I}$ has *bounded continuity*, if there exists $\delta > 0$ such that $\mathcal{I}$ satisfies δ-continuity. It is an easy observation that $\mathcal{I}_{\mathcal{R}}$ satisfies bounded continuity, and even bounded *weighted* continuity, for any set of anti-monotonic constraints. We will later prove that none of the other measures discussed so far satisfies (unweighted) bounded continuity.

Progression. The last property we discuss is *progression* that states that, within the underlying repair system $\mathcal{R} = (O, \kappa)$, there is always a way to progress towards consistency, as we can find an operation o of $\mathcal{R}$ such that inconsistency reduces after applying o. This property is particularly important for the task of progress indication in data repairing systems, as the combination of bounded continuity and progressions means that it is always possible to progress without significant slowdowns and long pauses, a behavior that users strongly averse towards [28].

Progression: whenever $D \not\models \Sigma$, there is $o \in O$ such that
$$\mathcal{I}(\Sigma, o(D)) < I(\Sigma, D).$$

Clearly, the measure $\mathcal{I}_{\mathsf{d}}$ violates progression. The measure $\mathcal{I}_{\mathcal{R}}$ satisfies progression for any set of anti-monotonic constraints, since we can always remove a fact from the minimum repair. The measure $\mathcal{I}_{\mathsf{MI}}$ satisfies progression, since we can always remove a fact f that participates in one of the minimal inconsistent subsets and, by doing so, eliminate all the subsets that include f. When we remove a fact f that appears in a minimal inconsistent subset, the measure $\mathcal{I}_{\mathsf{P}}$ decreases as well; hence, it satisfies progression. On the other hand, $\mathcal{I}_{\mathsf{MC}}$ and $\mathcal{I}'_{\mathsf{MC}}$ may violate progression even for functional dependencies, as illustrated in the following example.

Example 7. *Consider again the database D and the set Σ_2 from the proof of Proposition 2. As explained there, $\mathcal{I}_{\mathsf{MC}}(\Sigma_2, D) = \mathcal{I}'_{\mathsf{MC}}(\Sigma_2, D) = 1$. The reader can easily verify that for every tuple deletion o, it is still the case that $\mathcal{I}_{\mathsf{MC}}(\Sigma_2, o(D)) = \mathcal{I}'_{\mathsf{MC}}(\Sigma_2, o(D)) = 1$.*

Note that there are some dependencies among the properties, as shown in the following easy proposition (proof is in [41]).

Proposition 3. *Suppose that the class $\mathbf{C}$ is realizable by the repair system $\mathcal{R}$, and let $\mathcal{I}$ be an inconsistency measure.*

- *If $\mathcal{I}$ satisfies progression, then $\mathcal{I}$ satisfies positivity.*
- *If $\mathcal{I}$ satisfies positivity and bounded continuity, then $\mathcal{I}$ satisfies progression.*

Using Proposition 3, we can now prove the following.

Proposition 4. *In the case of $\mathcal{I}_{\mathsf{d}}, \mathcal{I}_{\mathsf{MI}}, \mathcal{I}_{\mathsf{P}}, \mathcal{I}_{\mathsf{MC}}$, and $\mathcal{I}'_{\mathsf{MC}}$, bounded (unweighted) continuity can be violated already for the class $\mathbf{C}_{\mathsf{FD}}$ of FDs and the system $\mathcal{R}_{\subseteq}$ of subset repairs.*

Proof. Let $\Sigma = \{A \rightarrow B\}$ and let D be a database that contains the following facts over $R(A, B, C)$:

$$f_0 = R(0,0,0) \quad f_i = R(0,1,i) \quad f_j^k = R(j,k,0)$$

where $i, j \in \{1, n\}$ for some n and $k \in \{1, 2\}$. The fact f_0 violates the FD with every fact f_i, and for each j, the facts f_j^1 and f_j^2 jointly violate the FD. All the facts in the database participate in a violation of the FD; hence, $\mathcal{I}_{\mathsf{P}}(\Sigma, D) = 3n + 1$. In addition, $\mathcal{I}_{\mathsf{MI}}(\Sigma, D) = 2n$.

Let the operation o_1 be the deletion of f_0. Applying o_1, we significantly reduce inconsistency w.r.t. these two measures, since none of the facts f_i now participates in a violation; thus, $\mathcal{I}_{\mathsf{P}}(\Sigma, o_1(D)) = 2n$ and $\mathcal{I}_{\mathsf{MI}}(\Sigma, o_1(D)) = n$. However, every possible operation o_2 on the database $o_1(D)$ only slightly reduces inconsistency (by two in the case of $\mathcal{I}_{\mathsf{P}}$ and by one in the case of $\mathcal{I}_{\mathsf{MI}}$). Therefore, $\Delta_{\mathcal{I}_{\mathsf{MI}},\Sigma}(o_1, D) = n$ and $\Delta_{\mathcal{I}_{\mathsf{MI}},\Sigma}(o_2, o_1(D)) = 1$, and the ratio between these two values depends on $|D|$. Similarly, it holds that $\Delta_{\mathcal{I}_{\mathsf{P}},\Sigma}(o_1, D) = n + 1$ and $\Delta_{\mathcal{I}_{\mathsf{P}},\Sigma}(o_2, o_1(D)) = 2$, and again the ratio between these two values depends on $|D|$.

As for $\mathcal{I}_{\mathsf{d}}$, $\mathcal{I}_{\mathsf{MC}}$, and $\mathcal{I}'_{\mathsf{MC}}$, we use Proposition 3. In the case of FDs, each of the three measures satisfies positivity but not progression (Example 7), and hence, they violate bounded continuity. □

Table 2 summarizes the satisfaction of the properties held by the different inconsistency measures we discussed here, for the case of a system $\mathbf{C}$ of FDs or DCs and the repair system $\mathcal{R}_{\subseteq}$. The last column refers to computational complexity, discussed in Section 5.1, and the last row refers to another measure, $\mathcal{I}_{\mathcal{R}}^{\mathsf{lin}}$, introduced in Section 5.2.

5 RATIONAL AND TRACTABLE MEASURES

In Section 4, we defined several properties of inconsistency measures, and we have shown that each of the measures we consider satisfies some and violates others, with the exception of $\mathcal{I}_{\mathcal{R}}$ that satisfies all. Unfortunately, as we explain in Section 5.1, this measure is often intractable. This, in turn, raises the question whether there is any tractable inconsistency measure that satisfies all of the properties. We answer this question affirmatively, for the case where repairing operations are tuple deletions, by presenting a new measure in Section 5.2. In Section 5.3, we discuss the challenges in designing such a measure for more general repair systems, particularly for the case where repairing operations are attribute updates.

5.1 Computational Complexity

We now discuss the complexity of measuring inconsistency according to the aforementioned measures. We focus on the class of DCs and the special case of FDs. Moreover, we focus on *data complexity*, which means that the set Σ of constraints is fixed, and only the database D is given as input for the computation of $\mathcal{I}(\Sigma, D)$.

The measure $\mathcal{I}_{\mathsf{d}}$ boils down to testing consistency, which is doable in polynomial time (under data complexity). The measures $\mathcal{I}_{\mathsf{MI}}$ and $\mathcal{I}_{\mathsf{P}}$ can be computed by enumerating all the subsets of D of a bounded size, where this size is determined by Σ. Hence, $\mathcal{I}_{\mathsf{MI}}$ and $\mathcal{I}_{\mathsf{P}}$ can also be computed in polynomial time.

The measures $\mathcal{I}_{\mathsf{MC}}$ and $\mathcal{I}'_{\mathsf{MC}}$ can be intractable to compute already for FDs. When Σ is a set of FDs, $\mathcal{I}_{\mathsf{MC}}(\Sigma, D)$ is the number of maximal independent sets (minus one) of the *conflict graph* wherein the tuples of D are the nodes, and there is an edge between every two tuples that violate an FD. Counting maximal independent sets is generally #P-complete, with several tractable classes of graphs such as the *P_4-free* graphs that do not have any induced subgraph that is a path of length four. Under conventional complexity assumptions, the finite sets Σ of FDs for which $\mathcal{I}_{\mathsf{MC}}(\Sigma, D)$ is computable in polynomial time are *precisely* the sets Σ of FDs that entail a P_4-free conflict graph for every database D [38]. Note that $\mathcal{I}'_{\mathsf{MC}}$ and $\mathcal{I}_{\mathsf{MC}}$ are equivalent for FDs; hence, the same applies to $\mathcal{I}'_{\mathsf{MC}}$.

The measure $\mathcal{I}_{\mathcal{R}}$ is also intractable even for FDs. For $\mathbf{C} = \mathbf{C}_{\mathsf{FD}}$ and $\mathcal{R} = \mathcal{R}_{\subseteq}$, this measure is the size of the minimum *vertex cover* of the conflict graph. Again, this is an NP-hard computational problem on general graphs. In a recent work, it has been shown that there is an

$$\text{Minimize}: \sum_{i \in ids(D)} x_i \cdot \kappa(\langle -i \rangle(\cdot), D) \ \text{ subj. to:}$$

$$\forall E \in \mathsf{MI}_\Sigma(D): \quad \sum_{i \in ids(E)} x_i \ \geq 1 \qquad (1)$$

$$\forall i \in ids(D): \quad x_i \in \{0,1\} \qquad (2)$$

Figure 2: ILP for $\mathcal{I}_\mathcal{R}(\Sigma, D)$ under $\mathbf{C}_{\mathrm{DC}}$ and $\mathcal{R}_\subseteq$.

efficient procedure that takes as input an FD set Σ and determines one of two outcomes: *(a)* $\mathcal{I}_\mathcal{R}(\Sigma, D)$ can be computed in polynomial time, *or (b)* $\mathcal{I}_\mathcal{R}(\Sigma, D)$ is NP-hard to compute (and even approximate beyond some constant) [40]. There, they have also studied the case where the repair system allows only attribute updates. In both repair systems it is the case that, if Σ consists of a single FD per relation (which is a commonly studied case, e.g., key constraints [18, 35]) then $\mathcal{I}_\mathcal{R}(\Sigma, D)$ can be computed in polynomial time. Unfortunately, this is no longer true (under conventional complexity assumptions) if we go beyond FDs to simple EGDs.

Example 8. *Consider the following four EGDs.*

$$\sigma_1: \quad \forall x, y, z[R(x,y), R(x,z) \Rightarrow (y = z)]$$
$$\sigma_2: \quad \forall x, y, z[R(x,y), R(y,z) \Rightarrow (x = z)]$$
$$\sigma_3: \quad \forall x, y, z[R(x,y), R(y,z) \Rightarrow (x = y)]$$
$$\sigma_4: \quad \forall x, y, z[R(x,y), S(y,z) \Rightarrow (x = z)]$$

Observe that σ_1 is an FD whereas σ_2, σ_3 and σ_4 are not. The constraint σ_2 states that there are no paths of length two except for two-node cycles, and σ_3 states that there are no paths of length two except for single-node cycles. Computing $\mathcal{I}_\mathcal{R}(\Sigma, D)$ w.r.t. $\Sigma = \{\sigma_1\}$ or $\Sigma = \{\sigma_4\}$ can be done in polynomial time; however, the problem becomes NP-hard for $\Sigma = \{\sigma_2\}$ and $\Sigma = \{\sigma_3\}$. □

The next theorem fully classifies the complexity of computing $\mathcal{I}_\mathcal{R}(\Sigma, D)$ for Σ that consists of a single EGD with two binary atoms.

Theorem 1. *Let $\mathcal{R} = \mathcal{R}_\subseteq$, and let Σ be a set that consists of a single EGD σ with two binary atoms. If σ is of the following form:*

$$\forall x_1, x_2, x_3[R(x_1, x_2), R(x_2, x_3) \Rightarrow (x_i = x_j)]$$

then computing $\mathcal{I}_\mathcal{R}(\Sigma, D)$ is NP-hard. In any other case, $\mathcal{I}_\mathcal{R}(\Sigma, D)$ can be computed in polynomial time.

The proof of the theorem is given in the extended version [41]. Note that the EGDs σ_2 and σ_3 from Example 8 satisfy the condition of Theorem 1; hence, computing $\mathcal{I}_\mathcal{R}(\Sigma, D)$ w.r.t. these EGDs is indeed NP-hard. The EGDs σ_1 and σ_4 do not satisfy the condition of the theorem; thus, computing $\mathcal{I}_\mathcal{R}(\Sigma, D)$ w.r.t. these EGDs can be done in polynomial time.

5.2 The Subset Repair System

The discussion in the previous section shows that among the measures considered so far, the rational ones are intractable. We now propose a new measure that is both rational and tractable in the case where $\mathbf{C}$ is the class $\mathbf{C}_{\mathrm{DC}}$ of DCs and $\mathcal{R} = \mathcal{R}_\subseteq$ (i.e., operations are tuple deletions). Recall that a DC has the form $\forall\vec{x}\neg[\varphi(\vec{x}) \wedge \psi(\vec{x})]$ where $\varphi(\vec{x})$ is a conjunction of atomic formulas, and $\psi(\vec{x})$ is a conjunction of comparisons over $\vec{x}$. Recall that DCs generalize common classes of constraints such as FDs, conditional FDs, and EGDs.

Let D be a database and Σ a finite set of DCs. For $\mathcal{R} = (O, \kappa)$, the measure $\mathcal{I}_\mathcal{R}(\Sigma, D)$ is the result of the Integer Linear Program (ILP) of Figure 2 wherein each x_i, for $i \in ids(D)$, determines whether to delete the ith tuple ($x_i = 1$) or not ($x_i = 0$). Denote by $\mathcal{I}_\mathcal{R}^{\mathsf{lin}}(\Sigma, D)$ the solution of the *linear relaxation* of this ILP, which is the Linear Program (LP) obtained from the ILP by replacing the last constraint (i.e., Equation (2)) with "$\forall i \in ids(D): 0 \leq x_i \leq 1$."

It is easy to see that the relative rankings of the inconsistency measure values of two databases under $\mathcal{I}_\mathcal{R}^{\mathsf{lin}}$ and $\mathcal{I}_\mathcal{R}$ are consistent with each other if they have sufficient separation under the first one. More formally, for two databases D_1, D_2 we have that $\mathcal{I}_\mathcal{R}^{\mathsf{lin}}(\Sigma, D_1) \geq \mu \cdot \mathcal{I}_\mathcal{R}^{\mathsf{lin}}(\Sigma, D_2)$ implies that $\mathcal{I}_\mathcal{R}(\Sigma, D_1) \geq \mathcal{I}_\mathcal{R}(\Sigma, D_2)$, where μ is the integrality gap of the LP relaxation. The maximum number of tuples involved in a violation of a constraint in Σ gives an upper bound on this integrality gap. In particular, for FDs (as well for the EGDs in Example 8), this number is 2; hence, $\mathcal{I}_\mathcal{R}^{\mathsf{lin}}(\Sigma, D_1) \geq 2 \cdot \mathcal{I}_\mathcal{R}^{\mathsf{lin}}(\Sigma, D_2)$ implies that $\mathcal{I}_\mathcal{R}(\Sigma, D_1) \geq \mathcal{I}_\mathcal{R}(\Sigma, D_2)$.

Example 9. *Consider again the databases of Figure 1 and the FDs of Example 1. In the LP of Figure 2, we define the variables $x_1, \dots, x_5$ corresponding to the facts $f_1, \dots, f_5$. Since D_0 is consistent, $MI_\Sigma(D_0)$ is empty, and we obtain a solution to the ILP by assigning $x_i = 0$ for all $i \in \{1, \dots, 5\}$. For D_1, we have that $\mathsf{MI}_\Sigma(D_1) = \{\{t_2, t_3\}, \{t_2, t_4\}, \{t_2, t_5\}, \{t_3, t_4\}, \{t_3, t_5\}, \{t_4, t_5\}, \{t_1, t_5\}\}$. For every pair $\{t_i, t_j\} \in \mathsf{MI}_\Sigma(D_1)$, the ILP contains a constraint $x_i + x_j \geq 1$. If we assign 0.5 to all x_i, all the constraints are satisfied. Assuming unit cost for deletion, the total cost is $\mathcal{I}_\mathcal{R}^{\mathsf{lin}}(\Sigma, D_1) = 2.5$. Another possible assignment is $x_1 = 0$, $x_2 = x_3 = x_4 = 0.5$, and $x_5 = 1$. Note that $\mathcal{I}_\mathcal{R}^{\mathsf{lin}}(\Sigma, D_1) < \mathcal{I}_\mathcal{R}(\Sigma, D_1) = 3$. However, for D_2, the optimal cost is $\mathcal{I}_\mathcal{R}^{\mathsf{lin}}(\Sigma, D_2) = \mathcal{I}_\mathcal{R}(\Sigma, D_2) = 2$, which can be obtained by assigning 0.5 to x_2, x_3, x_4, x_5 or by $x_2 = x_4 = 1, x_1 = x_3 = x_5 = 0$.*

The following theorem (proof is in the full version [41]) shows that $\mathcal{I}_\mathcal{R}^{\mathsf{lin}}$ satisfies all four properties and can be efficiently computed for the class $\mathbf{C}_{\mathrm{DC}}$ of denial constraints and the repair system $\mathcal{R}_\subseteq$.

Theorem 2. *The following hold for $\mathbf{C} = \mathbf{C}_{\mathrm{DC}}$ and $\mathcal{R} = \mathcal{R}_\subseteq$.*

(1) $\mathcal{I}_\mathcal{R}^{\mathsf{lin}}$ satisfies positivity, monotonicity, progression and constant weighted continuity.
(2) $\mathcal{I}_\mathcal{R}^{\mathsf{lin}}$ can be computed in polynomial time (in data complexity).

It thus appears from Theorem 2 that, for tuple deletions and DCs, $\mathcal{I}_\mathcal{R}^{\mathsf{lin}}$ is a desirable inconsistency measure—it satisfies the discussed properties and avoids the inherent hardness of $\mathcal{I}_\mathcal{R}$ (e.g., Theorem 1).

5.3 More General Repair Systems

In our analysis of the satisfaction of the properties and computational complexity of different inconsistency measures, we focused mostly on tuple deletions (i.e., where $\mathcal{R} = \mathcal{R}_\subseteq$) under DCs. While we have a good understanding of both aspects of inconsistency measures in this setting, the picture for other types of constraints and repairing operations is quite preliminary. As an example, consider the case of *update repairs* [33, 40], where the allowed repairing operations are attribute updates. We again assume that the constraints are DCs. Recall that we denote an attribute update by $\langle i.A \leftarrow c\rangle(\cdot)$, where i is a tuple identifier in D, A is an attribute, and c is the new value assigned to the fact $D[i]$ in attribute A. Here, we assume a countably infinite domain Val of possible attribute values.

We observe that none of the measures considered so far is a rational and tractable measure for update repairs. The satisfaction of positivity and monotonicity does not depend on the repair system at hand; hence, $\mathcal{I}_{\mathsf{MI}}$, $\mathcal{I}_{\mathsf{P}}$, $\mathcal{I}_{\mathsf{MC}}$, and $\mathcal{I}'_{\mathsf{MC}}$ violate at least one of these properties also for update repairs (see Table 2). When considering progression, the measures no longer behave the same for subset and update repairs. Clearly, $\mathcal{I}_{\mathsf{d}}$ still violates progression. As discussed in Section 4, the measures $\mathcal{I}_{\mathsf{MI}}$ and $\mathcal{I}_{\mathsf{P}}$ satisfy progression in the case of tuple deletions; however, both measures violate progression when considering update repairs, even for the more restricted case of FDs, as updating a single value does not necessarily completely resolve a conflict between two facts in the database.

Example 10. *Consider a database D over $R(A,B,C,D)$ consisting of the facts $R(0,0,0,0)$ and $R(0,1,0,1)$. Let $\Sigma = \{A \rightarrow B, C \rightarrow D\}$. The facts violate both FDs, and it is impossible to resolve both conflicts by updating a single value. Hence, $\mathcal{I}_{\mathsf{MI}}(\Sigma, D) = \mathcal{I}_{\mathsf{MI}}(\Sigma, o(D)) = 2$ and $\mathcal{I}_{\mathsf{P}}(\Sigma, D) = \mathcal{I}_{\mathsf{P}}(\Sigma, o(D)) = 2$ for any attribute update o.* □

The above example shows that for update repairs, considering violations at the fact level is insufficient, as a deletion of a single fact resolves every conflict it is involved in, but this is not the case when updating a single value in the fact. Hence, one may suggest to consider a measure that counts the minimal violations instead of the minimal inconsistent subsets. Here, a minimal violation is a pair (F, σ), where $F \subseteq D$ and σ is a constraint in Σ, such that $F \not\models \sigma$, but $F' \models \sigma$ for all $F' \subseteq F$. Note that Example 10 is not a counterexample for progression in this case, as we can always decrease the number of violations by updating a single attribute value. Nonetheless, the new measure still does not satisfy progression, as illustrated next.

Example 11. Let $\Sigma = \{A \rightarrow B, B \rightarrow C, D \rightarrow A\}$. Let D be a database that contains the following four facts over $R(A,B,C,D,E)$:

$$f_0 = R(0,0,0,0,1) \quad f_1 = R(0,0,0,0,2)$$
$$f_2 = R(0,1,1,0,3) \quad f_3 = R(0,1,1,0,4)$$

The minimal violations in D are: $(\{f_0, f_2\}, A \rightarrow B)$, $(\{f_0, f_3\}, A \rightarrow B)$, $(\{f_1, f_2\}, A \rightarrow B)$, and $(\{f_1, f_3\}, A \rightarrow B)$. Since the attributes C, D, and E are not involved in violations, updating values in these attributes will not eliminate violations. Moreover, updating a value in attribute B to a value outside the domain of B can only increase the number of violations. Next, we show that the two remaining operations: *(1)* updating a value in attribute A to a new value, and *(2)* updating a value in attribute B to another value from its domain (i.e., either 0 or 1), also increase the number of violations.

Suppose that we change the value of attribute A in f_0 to 1. This operation resolves the violations $(\{f_0, f_2\}, A \rightarrow B)$ and $(\{f_0, f_3\}, A \rightarrow B)$, but introduces new violations: $(\{f_0, f_1\}, D \rightarrow A)$, $(\{f_0, f_2\}, D \rightarrow A)$, and $(\{f_0, f_3\}, D \rightarrow A)$. Hence, the total number of violations increases. Clearly, updating the value of attribute A in one of f_1, f_2, or f_3 will similarly increase the number of violations.

Next, suppose that we change the value of attribute B in f_0 to 1. This operation again resolves the violations $(\{f_0, f_2\}, A \rightarrow B)$ and $(\{f_0, f_3\}, A \rightarrow B)$. However, it introduces the violations $(\{f_0, f_1\}, A \rightarrow B)$, $(\{f_0, f_2\}, B \rightarrow C)$ and $(\{f_0, f_3\}, B \rightarrow C)$, and the total number of violations increases. A similar argument applies to the case when the value of attribute B is changed in f_1, f_2, or f_3. Therefore, while there exists a sequence of operations that decreases the number of violations, no individual operation does. □

Example 11 can be used to show that $\mathcal{I}_{\mathsf{MC}}$ and $\mathcal{I}'_{\mathsf{MC}}$ violate progression as well. The measure $\mathcal{I}_{\mathcal{R}}$ satisfies progression, as we can always update an attribute value from the minimum repair.

Proposition 3 implies that none of the measures considered so far, except for $\mathcal{I}_{\mathcal{R}}$, satisfies bounded continuity. The measure $\mathcal{I}_{\mathcal{R}}$ satisfies bounded (weighted) continuity, since for any operation o we have that $\frac{\Delta_{\mathcal{I},\Sigma}(o,D_1)}{\delta \cdot \kappa(o,D_1)}$ is either 1 (if o belongs to a minimum repair) or 0 (if no minimum repair contains o). We conclude that $\mathcal{I}_{\mathcal{R}}$ again stands out among the measures as it satisfies every property. Hence, in cases where $\mathcal{I}_{\mathcal{R}}$ is tractable (e.g., when no two constraints share an attribute), this measure provides both rationality and tractability; however, computing $\mathcal{I}_{\mathcal{R}}$ is often hard, even for simple FD sets [40].

Recall that in the case of $\mathcal{R}_{\subseteq}$, the linear relaxation of $\mathcal{I}_{\mathcal{R}}$ can be used to obtain a desired measure that is also efficient. This, however, is no longer the case for updates. We do not have any natural linear relaxation for attribute updates, and finding such a measure remains a challenging open problem for future research.

Remark. We again stress that a chosen measure of inconsistency does not restrict the repair operations that a system allows or applies. Any repair system, supporting any type of operations whatsoever, could adopt measures such as $\mathcal{I}_{\mathcal{R}}$ (i.e., the maximum size of a consistent subset) and $\mathcal{I}^{\mathsf{lin}}_{\mathcal{R}}$. It is just that continuity and progression are guaranteed only for deletion operations, and could be violated otherwise. Indeed, our empirical experience, which we report in the next section, draws an optimistic picture: these measures work well in practical scenarios even for attribute updates.

6 EXPERIMENTAL EVALUATION

In this section, we present an experimental evaluation of the considered measures under different error (and repair) models.

6.1 Setup

Datasets. We evaluate the measures on datasets that were previously used for the problem of mining constraints [8, 11, 37, 47]: the real-world datasets **Stock**, **Hospital**, **Food**, **Airport**, **Adult**, **Flight**, and **Voter**, and the synthetic **Tax** dataset. We use a DC mining algorithm [37] to obtain a set of DCs for each dataset. All DCs are of the form $\forall t, t' \neg(P_1, \dots, P_m)$, where t, t' are database tuples, each P_i is a predicate $t[A]\,\rho\,t'[B]$, such that A and B are attributes of the schema, and ρ is a comparison operator from $\{=, \neq, >, <, \geq, \leq\}$. (Note that it may be the case the $t = t'$.) More details about the datasets and constraints are given in Figure 3.

Measure implementations. We implemented all measures in Python 3 using Pandas. Using SQL, we materialize all conflicting pairs of tuples. For example, the following DC over the Tax dataset:

$$\forall t, t' \neg(t[\mathrm{St}] = t'[\mathrm{St}], t[\mathrm{Salary}] > t'[\mathrm{Salary}], t[\mathrm{Tax}] < t'[\mathrm{Tax}])$$

(where St stands for "State") will give rise to the query

```
SELECT DISTINCT $R_1$.ID, $R_2$.ID
FROM $R$ AS $R_1$, $R$ AS $R_2$
WHERE $R_1$.St = $R_2$.St, $R_1$.Salary > $R_2$.Salary, $R_1$.Tax < $R_2$.Tax .
```

For the measure $\mathcal{I}_{\mathsf{d}}$, we simply return 1 if the query result is nonempty and 0 otherwise. The measure $\mathcal{I}_{\mathsf{MI}}$ counts the tuples in the query result, and $\mathcal{I}_{\mathsf{P}}$ counts the database facts that occur in these tuples. To compute $\mathcal{I}_{\mathsf{MC}}$, we use a C++ implementation [50]

Dataset	#Tuples	#Atts.	#DCs	Example constraint
Stock	123K	7	6	$\forall t \neg (t[\text{High}] < t[\text{Low}])$
Hospital	115K	15	7	$\forall t, t' \neg (t[\text{State}] = t'[\text{State}], t[\text{Measure}] = t'[\text{Measure}], t[\text{StateAvg}] \neq t'[\text{StateAvg}])$
Food	200K	17	6	$\forall t, t' \neg (t[\text{Location}] = t'[\text{Location}], t[\text{City}] \neq t'[\text{City}])$
Airport	55K	9	6	$\forall t, t' \neg (t[\text{Country}] = t'[\text{Country}], t[\text{Continent}] \neq t'[\text{Continent}])$
Adult	32K	15	3	$\forall t, t' \neg (t[\text{Gain}] < t'[\text{Gain}], t[\text{Loss}] < t'[\text{Loss}])$
Flight	500K	20	13	$\forall t, t' \neg (t[\text{Origin}] = t'[\text{Origin}], t[\text{Dest}] = t'[\text{Dest}], t[\text{Distance}] \neq t'[\text{Distance}])$
Voter	950K	22	5	$\forall t, t' \neg (t[\text{BirthYear}] < t'[\text{BirthYear}], t[\text{Age}] > t'[\text{Age}])$
Tax	1M	15	9	$\forall t, t' \neg (t[\text{State}] = t'[\text{State}], t[\text{Salary}] > t'[\text{Salary}], t[\text{Rate}] < t'[\text{Rate}])$

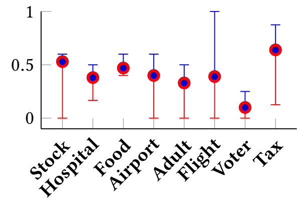


Figure 3: The datasets used in our experiments. On the right: the level of attribute overlapping of the constraints.

of an algorithm for enumerating the maximal cliques [15] over the complement of the conflict graph. The conflict graph is also built from the result of the above SQL query: we add a vertex for each fact and an edge for each fact pair in the result. We use the Gurobi Optimizer [27] to compute $\mathcal{I}_{\mathcal{R}}$ and $\mathcal{I}_{\mathcal{R}}^{\text{lin}}$ using the LP of Figure 2. We dynamically construct the LP from the result of the SQL query (i.e., we add a corresponding constraint for each fact pair in the result).

Noise Generation. Initially, all datasets are consistent w.r.t. the given set of DCs. We use two algorithms to add noise to these datasets. In the first, which we refer to as CONoise **(for Constraint-Oriented Noise)**, we introduce random violations of the constraints, by running several iterations of the following procedure:

(1) Randomly select a constraint φ from the set of constraints.
(2) Randomly select two tuples t and t' from the database.
(3) For every predicate $P = (t[A] \; \rho \; t'[B])$ of φ:
 - If t and t' jointly satisfy P, continue to the next predicate.
 - If $\rho \in \{=, \leq, \geq\}$, change either $t[A]$ to $t[B]$ or vice versa (the choice is random).
 - If $\rho \in \{<, >, \neq\}$, change either $t[A]$ or $t[B]$ (the choice is again random) to another value from the active domain of the attribute such that P is satisfied, if such a value exists, or a random value in the appropriate range otherwise.

The second algorithm, RNoise **(for Random Noise)**, has two parameters: α is used to control the level of noise (we modify α of the values in the dataset), and β, controls the data skewness, as we now explain. At each iteration of RNoise, we randomly select a database cell corresponding to an attribute that occurs in at least one constraint. Then, we either change its value to another value from the active domain of the corresponding attribute (with probability 0.5) or to a typo. For the first case, we use the Zipfian distribution, where the probability of selecting the ith value in the active domain is proportional to $i^{-\beta}$; hence, larger β means larger skew.

General setup. All experiments were executed on a server with two Intel(R) Xeon(R) Gold 6130 CPUs (2.10GHz, 16 cores) with 512GB of RAM running Ubuntu 20.04. Each experiment was repeated five times and the average times are reported. The graphs of Figures 4 and 5 were obtained in one execution and are representative of all five executions, where we observed a similar behavior.

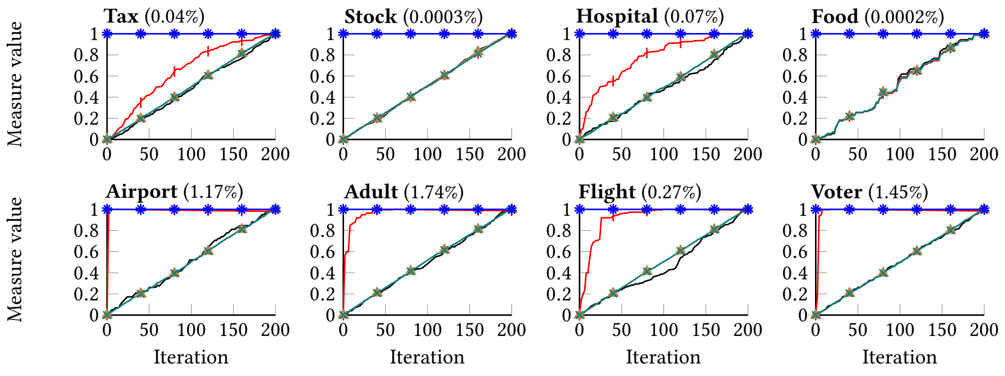
6.2 Results

6.2.1 Measure Behavior. We evaluate the behavior of the measures on samples of 10K tuples from each dataset. First, we run 200 iterations of CONoise on each dataset and compute the measure values after each iteration; the results are in Figure 4a. Then, we run RNoise with $\beta = 0$, $\beta = 1$, and $\beta = 2$ until we modify 1% of the values in the dataset (hence, $\alpha = 0.01$). As the number of iterations may be high, we compute the measure values every ten iterations. The results for $\beta = 0$ are depicted in Figure 4b, and the (similar) results for $\beta = 1, 2$ are in the extended version [41], where we also test different probabilities for typos. We report the final violation ratio (i.e., percentage of violating tuple pairs out of all pairs) obtained in the experiments above each diagram (in parentheses). Note that when we modify values in an iteration of CONoise or RNoise, we may introduce several violations at once, and resolve other violations at the same time. In our experiments, we observed that the number of newly introduced violations is usually significantly higher than the number of resolved ones, as evidenced by the behavior of $\mathcal{I}_{\text{MI}}$ (that counts violations) in the charts of Figure 4: its value generally increases with the number of iterations.

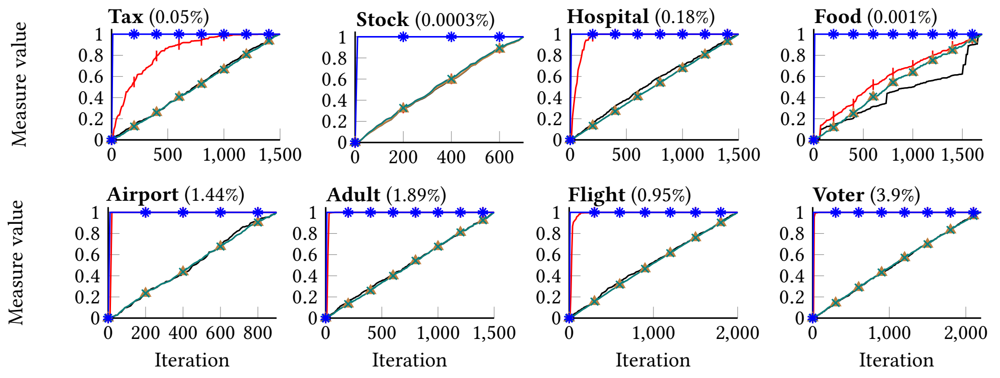
Variations with noise. In general, we see that the measures may behave very differently on the same dataset. As expected, the drastic measure $\mathcal{I}_{\text{d}}$ jumps from zero to one when we introduce the first violation, and stays at that point until the end of the execution. We see that the measure $\mathcal{I}_{\text{P}}$ often behaves in a similar way. For example, on the Airport dataset, it jumps from zero to its maximal value already in the first iteration. This behavior can be explained by observing the constraint set used for this dataset. For example, one of the DCs in this set is $\forall t, t' \neg (t[\text{Country}] = t'[\text{Country}], t[\text{Continent}] \neq t'[\text{Continent}])$, and all the tuples in the dataset initially agree on the value of the country and continent attributes. Hence, whenever we change the value of the continent attribute for a single tuple, all the other tuples are immediately involved in a violation with it, and the value of $\mathcal{I}_{\text{P}}$ jumps to #tuples.

Contrarily, the measure $\mathcal{I}_{\mathcal{R}}$ is able to recognize, in this example, that the dataset contains a single erroneous tuple, and react to this small change in a more proportional way. In general, we see that the measures $\mathcal{I}_{\mathcal{R}}$, $\mathcal{I}_{\mathcal{R}}^{\text{lin}}$, and $\mathcal{I}_{\text{MI}}$ behave similarly in most cases and the corresponding graphs are generally monotonically increasing and close to being linear. While this behavior seems to be very consistent for $\mathcal{I}_{\mathcal{R}}$ and $\mathcal{I}_{\mathcal{R}}^{\text{lin}}$, the measure $\mathcal{I}_{\text{MI}}$ is slightly less stable, as can be seen, for example, on the Food dataset in Figure 4b.

Due to the high computational cost of $\mathcal{I}_{\text{MC}}$ (as we report later on), we evaluate its behavior on a small sample of 100 tuples from each dataset. The results are in Figure 5; missing graphs are due to timeout. The left chart is for CONoise and the right is for RNoise with $\beta = 0$. We run both algorithms for 100 iterations. Observe that this measure is the least stable of all, as we get very different graphs on the different datasets. In particular, on the Stock dataset it resembles a step function and fails to indicate progress for long



(a) Noise added with CONoise.



(b) Noise added with RNOise ($\alpha = 0.01$ and $\beta = 0$).

Figure 4: The normalized values of $\mathcal{I}_{\mathsf{d}}$ (—✱—), $\mathcal{I}_{\mathsf{MI}}$ (——), $\mathcal{I}_{\mathsf{P}}$ (—+—), $\mathcal{I}_{\mathcal{R}}$ (—▲—), and $\mathcal{I}_{\mathcal{R}}^{\mathsf{lin}}$ (—×—). Violation ratio in parentheses.

periods of time. On the Airport dataset, we see a lot of jumps and jitters on these graphs. This behavior may be affected, to some extent, by the small size of the datasets. However, the charts that we obtain for the other measures in this case (given in [41]), while also less stable, follow a similar trend as the ones in Figure 4.

Error rate, data skew, and overlapping constraints. The experiments indicate that the behavior of the measures is largely stable across several properties of the data and constraints. We can see that the error rate, which increases with the number of iterations, affects the value of the measures, but has no evident impact on their behavior (i.e., the trend of the graph). Moreover, we obtain very similar charts in the experiments with $\beta = 1, 2$ and distinct typo probabilities (see [41]); hence, *data skew* and different *distributions of error types* also do not seem to affect the results. Finally, we examine how *overlap of dependencies* affects the results. For each dataset, and for each DC in its constraint set, we compute the ratio of DCs that overlap with it (i.e., the DCs share at least one attribute). Figure 3 (right) shows the minimum, maximum, and average values for each dataset. We again see no clear correlation between the behavior of the measures and level of overlap.

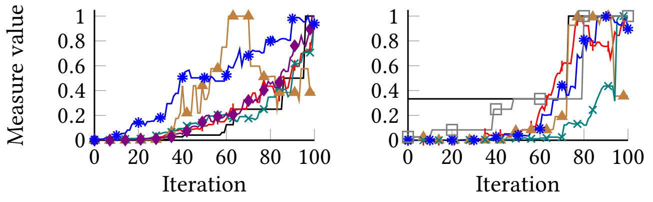


Figure 5: $\mathcal{I}_{\mathsf{MC}}$ (normal.): Stock (——), Hospital (—+—), Food (—□—), Airport (—▲—), Adult (—×—), Flight (—◆—), Voter (—✱—), 100 iterations. Left: CONoise, **Right:** RNoise ($\beta = 0$).

6.2.2 *Case study: the HoloClean repair system.*

Up to now, we described experiments with our synthetic noise generation models. We have shown that the behavior of the measures is robust to the operations and is not sensitive to various parameters of the input such as data skew, error rate, and overlap of constraints. We now further strengthen this finding by showing similar results on a cleaning system that we treat as a black box, namely the HoloClean system [48], that uses soft rules and a statistical approach for automatic data cleaning. To accurately analyze the behavior of our measures, we need a dataset where the behavior of HoloClean is predictable; hence, we run the system on the (dirty) Hospital dataset provided in the HoloClean repository (https://github.com/HoloClean) with a set of 15 DCs. It has been shown that the accuracy of HoloClean on this dataset is very high [48]; thus, the system should significantly decrease the level of inconsistency in this dataset.

Since HoloClean features one-shot automatic cleaning, we simulate a cleaning pipeline by providing it with *a single DC at a time.*

Table 3: Running Times in sec. (The $\mathcal{I}_{\mathsf{MC}}$ measure and the Voter dataset are excluded due to timeout.)

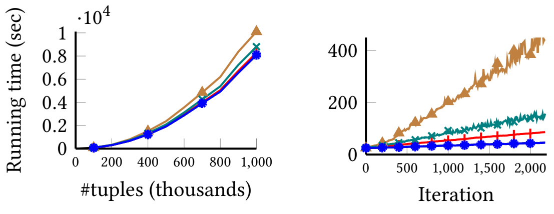
	$\mathcal{I}_{\mathsf{d}}$	$\mathcal{I}_{\mathcal{R}}$	$\mathcal{I}_{\mathsf{MI}}$	$\mathcal{I}_{\mathsf{P}}$	$\mathcal{I}_{\mathcal{R}}^{\mathsf{lin}}$
Tax	8092.894	10102.15	8092.894	8275.692	8804.30
Stock	1.16	1.16	1.16	1.28	1.16
Hospital	199.08	212.59	199.08	200.19	207.91
Food	89.38	89.92	89.38	91.25	89.81
Airport	61.64	78.96	61.64	63.33	73.77
Adult	119.19	240.96	119.19	132.30	179.31
Flight	8084.05	8222.35	8084.05	8138.40	8157.30

That is, we first run HoloClean on the original dataset with a single DC; then on the resulting dataset after adding one more DC to the constraint set, and so on. Note that HoloClean uses soft constraints; hence, it does not necessarily eliminate all violations. We compute the measures after every step; the (normalized) values are in Figure 7. We tried several random permutations of the DCs and obtained similar results. Again, we see that $\mathcal{I}_{\mathsf{d}}$ and $\mathcal{I}_{\mathsf{P}}$ fall short of effectively indicating progress. Contrarily, the other measures, particularly $\mathcal{I}_{\mathcal{R}}$ and $\mathcal{I}_{\mathcal{R}}^{\mathsf{lin}}$, are able to capture the reduction in the inconsistency level, and show an almost linear decay as desired.

6.2.3 Running Times. We now study the running times of the measures we discussed in the paper. We stress that our focus is not on optimizing these measures, but rather on understanding the execution cost obtained in reasonable implementations. Table 3 shows the average running times of the measures on all datasets after running #tuples/1000 iterations of CONoise (with the number of tuples reported in Figure 3). The computation of $\mathcal{I}_{\mathsf{MC}}$ exceeded our 24-hour limit on all datasets (and, in some cases, even on datasets with only one hundred tuples). An immediate conclusion is that $\mathcal{I}_{\mathsf{MC}}$ is not only behaving oddly, but is prohibitively infeasible. The Voter dataset (that we discuss next) is also excluded due to timeout.

We can see that the running times of the measures are usually close to each other. This is because the dominant part of the computation of $\mathcal{I}_{\mathcal{R}}$ and $\mathcal{I}_{\mathcal{R}}^{\mathsf{lin}}$ for large datasets is the evaluation of the SQL query that finds all violations of the constraints. In the case of the Voter dataset, the SQL engine reached the 24-hour limit. The domination of the SQL computation can also be seen in Figure 6a that depicts the running times of all the measures, except for $\mathcal{I}_{\mathsf{MC}}$ (again, due to timeout), on samples of the Tax dataset, consisting of 100K to 1M tuples. The figure shows a quadratic trend, which is consistent with the complexity of the dominating SQL part.

This is not the case, however, for smaller datasets, as can be seen in Figure 6b. This chart depicts the running times of the measures on



(a) (b)

Figure 6: Scalability in $|D|$ on Tax (a) and error rate on Voter (b): $\mathcal{I}_{\mathsf{d}}$ (—✱—), $\mathcal{I}_{\mathsf{MI}}$ (——), $\mathcal{I}_{\mathsf{P}}$ (—+—), $\mathcal{I}_{\mathcal{R}}$ (—▲—), and $\mathcal{I}_{\mathcal{R}}^{\mathsf{lin}}$ (—×—).

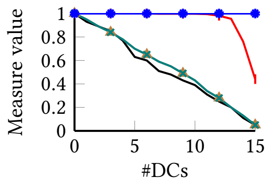


Figure 7: HoloClean case study—normalized measures on Hospital: $\mathcal{I}_{\mathsf{d}}$ (—✱—), $\mathcal{I}_{\mathsf{MI}}$ (——), $\mathcal{I}_{\mathsf{P}}$ (—+—), $\mathcal{I}_{\mathcal{R}}$ (—▲—), $\mathcal{I}_{\mathcal{R}}^{\mathsf{lin}}$ (—×—).

a sample of 10K tuples from the Voter dataset. There, we add noise using RNoise with $\alpha = 0.01$ and $\beta = 0$, and compute the running times every ten iterations. The evaluation of the SQL query is quite fast on these smaller datasets, and the computation of $\mathcal{I}_{\mathcal{R}}$ and $\mathcal{I}_{\mathcal{R}}^{\mathsf{lin}}$ is now dominated by the LP solver. We also see a more significant difference in running times between these measures. While the computation of $\mathcal{I}_{\mathsf{d}}$, $\mathcal{I}_{\mathsf{MI}}$, and $\mathcal{I}_{\mathsf{P}}$ is only slightly affected by the change in error rate (that increases with the number of iterations), the computations of $\mathcal{I}_{\mathcal{R}}$ significantly increases with the increased error rate. We provide similar charts for the other datasets in [41].

7 CONCLUDING REMARKS

We explored inconsistency measures for databases, and investigated the properties that should be accounted for in the choice of a measure for a specific use case. We discussed four properties where two, continuity and progression, are defined in the context of the underlying repair system. We also used the properties to reason about various specific instances of inconsistency measures. The combination of the properties and the computational complexity shed a positive light on the linear relaxation of minimal repairing when considering DCs and tuple deletions. In fact, the design of this measure is driven by that combination, and is not as interpretable as the others. Our experimental study shows that the measures that well behave theoretically for tuple deletions also exhibit a good empirical behavior, even when our error models capture attribute updates rather than tuple insertions (to be remedied by tuple deletions). These measures were also able to effectively capture progress in our HoloClean case study.

This work opens the way to an important angle of inconsistency measurement that has not been treated before, and many fundamental problems remain open. We plan to explore other properties as well as *completeness* criteria for sets of properties to determine sufficiency for certain use cases. Another important direction is to explore more general repair systems (allowing different types of constraints and repairing operations). It is also interesting to investigate the adaption of Grant and Hunter's concept of information loss [24], and explore the trade-off between inconsistency reduction and information loss, in the context of database repairing. The final goal is to devise actual measures that are practically useful, efficient to compute, and justified by a clear theoretical ground.

Acknowledgements. This work was supported by the German Research Foundation (DFG) Project 412400621 (DIP program) and the Israel Science Foundation (ISF), Grant 768/19. The work of Sudeepa Roy was supported by NSF awards IIS-1552538, IIS-1703431, IIS-2008107, and NIH award R01EB025021. We also thank the reviewers of this paper for very helpful comments and suggestions.

REFERENCES

[1] Ziawasch Abedjan, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. 2016. Detecting Data Errors: Where are we and what needs to be done? *Proc. VLDB Endow.* 9, 12 (2016), 993–1004.

[2] Foto N. Afrati and Phokion G. Kolaitis. 2009. Repair checking in inconsistent databases: algorithms and complexity. In *ICDT*, Vol. 361. ACM, 31–41.

[3] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. 1999. Consistent Query Answers in Inconsistent Databases. In *PODS*. ACM Press, 68–79.

[4] Catriel Beeri and Moshe Y. Vardi. 1981. The Implication Problem for Data Dependencies. In *ICALP*, Vol. 115. Springer, 73–85.

[5] Leopoldo E. Bertossi. 2018. Measuring and Computing Database Inconsistency via Repairs. *CoRR* abs/1804.08834 (2018).

[6] Leopoldo E. Bertossi. 2018. Measuring and Computing Database Inconsistency via Repairs. In *SUM (Lecture Notes in Computer Science, Vol. 11142)*. Springer, 368–372.

[7] Leopoldo E. Bertossi, Loreto Bravo, Enrico Franconi, and Andrei Lopatenko. 2008. The complexity and approximation of fixing numerical attributes in databases under integrity constraints. *Inf. Syst.* 33, 4-5 (2008), 407–434.

[8] Tobias Bleifuß, Sebastian Kruse, and Felix Naumann. 2017. Efficient Denial Constraint Discovery with Hydra. *PVLDB* 11, 3 (2017), 311–323.

[9] Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2007. Conditional Functional Dependencies for Data Cleaning. In *ICDE*. IEEE, 746–755.

[10] Nofar Carmeli, Martin Grohe, Benny Kimelfeld, Ester Livshits, and Muhammad Tibi. 2020. Database Repairing with Soft Functional Dependencies. *CoRR* abs/2009.13821 (2020).

[11] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Discovering Denial Constraints. *PVLDB* 6, 13 (2013), 1498–1509.

[12] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Holistic data cleaning: Putting violations into context. In *ICDE*. IEEE Computer Society, 458–469.

[13] Gao Cong, Wenfei Fan, Floris Geerts, Xibei Jia, and Shuai Ma. 2007. Improving Data Quality: Consistency and Accuracy. In *VLDB*. ACM, 315–326.

[14] Alex Paul Conn. 1995. Time Affordances: The Time Factor in Diagnostic Usability Heuristics. In *SIGCHI* (Denver, Colorado, USA) *(CHI '95)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 186–193. https://doi.org/10.1145/223904.223928

[15] Alessio Conte, Roberto Grossi, Andrea Marino, and Luca Versari. 2016. Sublinear-Space Bounded-Delay Enumeration for Massive Network Analytics: Maximal Cliques. In *ICALP (LIPIcs, Vol. 55)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 148:1–148:15.

[16] Amr Ebaid, Ahmed K. Elmagarmid, Ihab F. Ilyas, Mourad Ouzzani, Jorge-Arnulfo Quiané-Ruiz, Nan Tang, and Si Yin. 2013. NADEEF: A Generalized Data Cleaning System. *PVLDB* 6, 12 (2013), 1218–1221.

[17] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional functional dependencies for capturing data inconsistencies. *ACM Trans. Database Syst.* 33, 2 (2008), 6:1–6:48.

[18] Ariel Fuxman and Renée J. Miller. 2007. First-order query rewriting for inconsistent databases. *J. Comput. Syst. Sci.* 73, 4 (2007), 610–635. https://doi.org/10.1016/j.jcss.2006.10.013

[19] Terry Gaasterland, Parke Godfrey, and Jack Minker. 1992. An Overview of Cooperative Answering. *J. Intell. Inf. Syst.* 1, 2 (1992), 123–157.

[20] Jaffer Gardezi, Leopoldo E. Bertossi, and Iluju Kiringa. 2011. Matching dependencies with arbitrary attribute values: semantics, query answering and integrity constraints. In *LID*. 23–30.

[21] Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. 2013. The LLUNATIC Data-Cleaning Framework. *PVLDB* 6, 9 (2013), 625–636.

[22] Oded Goldreich, Shafi Goldwasser, and Dana Ron. 1998. Property Testing and its Connection to Learning and Approximation. *J. ACM* 45, 4 (1998), 653–750. https://doi.org/10.1145/285055.285060

[23] John Grant and Anthony Hunter. 2006. Measuring inconsistency in knowledgebases. *J. Intell. Inf. Syst.* 27, 2 (2006), 159–184.

[24] John Grant and Anthony Hunter. 2011. Measuring Consistency Gain and Information Loss in Stepwise Inconsistency Resolution. In *ECSQARU*, Vol. 6717. Springer, 362–373.

[25] John Grant and Anthony Hunter. 2013. Distance-Based Measures of Inconsistency. In *ECSQARU (Lecture Notes in Computer Science, Vol. 7958)*. Springer, 230–241.

[26] John Grant and Anthony Hunter. 2017. Analysing inconsistent information using distance-based measures. *Int. J. Approx. Reasoning* 89 (2017), 3–26. https://doi.org/10.1016/j.ijar.2016.04.004

[27] LLC Gurobi Optimization. 2020. Gurobi Optimizer Reference Manual. http://www.gurobi.com

[28] Chris Harrison, Brian Amento, Stacey Kuznetsov, and Robert Bell. 2007. Rethinking the Progress Bar. In *UIST* (Newport, Rhode Island, USA) *(UIST '07)*. ACM, New York, NY, USA, 115–118.

[29] Chris Harrison, Zhiquan Yeo, and Scott E. Hudson. 2010. Faster Progress Bars: Manipulating Perceived Duration with Visual Augmentations. In *SIGCHI* (Atlanta, Georgia, USA) *(CHI '10)*. ACM, New York, NY, USA, 1545–1548.

[30] Anthony Hunter and Sébastien Konieczny. 2008. Measuring Inconsistency through Minimal Inconsistent Sets. In *KR*. AAAI Press, 358–366.

[31] Anthony Hunter and Sébastien Konieczny. 2010. On the measure of conflicts: Shapley Inconsistency Values. *Artif. Intell.* 174, 14 (2010), 1007–1026.

[32] Kevin M. Knight. 2003. Two Information Measures for Inconsistent Sets. *Journal of Logic, Language and Information* 12, 2 (2003), 227–248.

[33] Solmaz Kolahi and Laks V. S. Lakshmanan. 2009. On approximating optimum repairs for functional dependency violations. In *ICDT*, Vol. 361. ACM, 53–62.

[34] Sébastien Konieczny, Jérôme Lang, and Pierre Marquis. 2003. Quantifying information and contradiction in propositional logic through test actions. In *IJCAI*. Morgan Kaufmann, 106–111.

[35] Paraschos Koutris and Jef Wijsen. 2017. Consistent Query Answering for Self-Join-Free Conjunctive Queries Under Primary Key Constraints. *ACM Trans. Database Syst.* 42, 2 (2017), 9:1–9:45.

[36] Sebastian Kruse, Paolo Papotti, and Felix Naumann. 2015. Estimating Data Integration and Cleaning Effort. In *EDBT*. OpenProceedings.org, 61–72.

[37] Ester Livshits, Alireza Heidari, Ihab F. Ilyas, and Benny Kimelfeld. 2020. Approximate Denial Constraints. *Proc. VLDB Endow.* 13, 10 (2020), 1682–1695.

[38] Ester Livshits and Benny Kimelfeld. 2017. Counting and Enumerating (Preferred) Database Repairs. In *PODS*. ACM, 289–301.

[39] Ester Livshits and Benny Kimelfeld. 2020. The Shapley Value of Inconsistency Measures for Functional Dependencies. *CoRR* abs/2009.13819 (2020).

[40] Ester Livshits, Benny Kimelfeld, and Sudeepa Roy. 2020. Computing Optimal Repairs for Functional Dependencies. *ACM Trans. Database Syst.* 45, 1 (2020), 4:1–4:46.

[41] Ester Livshits, Rina Kochirgan, Segev Tsur, Ihab F. Ilyas, Benny Kimelfeld, and Sudeepa Roy. 2021. Properties of Inconsistency Measures for Databases. *CoRR* abs/1904.06492v3 (2021).

[42] Andrei Lopatenko and Leopoldo E. Bertossi. 2007. Complexity of Consistent Query Answering in Databases Under Cardinality-Based and Incremental Repair Semantics. In *ICDT*. 179–193.

[43] Gang Luo, Jeffrey F. Naughton, Curt J. Ellmann, and Michael Watzke. 2004. Toward a Progress Indicator for Database Queries. In *SIGMOD*. 791–802.

[44] Maria Vanina Martinez, Andrea Pugliese, Gerardo I. Simari, V. S. Subrahmanian, and Henri Prade. 2007. How Dirty Is Your Relational Database? An Axiomatic Approach. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, Khaled Mellouli (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 103–114.

[45] Brad A. Myers. 1985. The Importance of Percent-done Progress Indicators for Computer-human Interfaces. In *SIGCHI* (San Francisco, California, USA) *(CHI '85)*. ACM, New York, NY, USA, 11–17.

[46] Francesco Parisi and John Grant. 2019. Inconsistency Measures for Relational Databases. *CoRR* abs/1904.03403 (2019).

[47] Eduardo H. M. Pena, Eduardo C. de Almeida, and Felix Naumann. 2019. Discovery of Approximate (and Exact) Denial Constraints. *PVLDB* 13, 3 (2019).

[48] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *PVLDB* 10, 11 (2017), 1190–1201.

[49] Matthias Thimm. 2017. On the Compliance of Rationality Postulates for Inconsistency Measures: A More or Less Complete Picture. *KI* 31, 1 (2017), 31–39.

[50] Luca Versari, Daniele De Sensi, Alessio Conte, and Tiziano De Matteis. 2019. parallel enum. https://github.com/veluca93/parallel_enum.

[51] Ana Villar, Mario Callegaro, and Yongwei Yang. 2013. Where Am I? A Meta-Analysis of Experiments on the Effects of Progress Indicators for Web Surveys. *Soc. Sci. Comput. Rev.* 31, 6 (Dec. 2013), 744–762.

[52] Jef Wijsen. 2005. Database repairing using updates. *ACM Trans. Database Syst.* 30, 3 (2005), 722–768.

[53] Bruno Yun, Srdjan Vesic, Madalina Croitoru, and Pierre Bisquert. 2018. Inconsistency Measures for Repair Semantics in OBDA. In *IJCAI*. ijcai.org, 1977–1983.