Repair-Based Degrees of Database Inconsistency

Leopoldo Bertossi[1,2(✉)]

[1] RelationalAI Inc., Toronto, Canada
[2] Carleton University, Ottawa, Canada
bertossi@scs.carleton.ca

Abstract. We propose and investigate a concrete numerical measure of the inconsistency of a database with respect to a set of integrity constraints. It is based on a database repair semantics associated to cardinality-repairs. More specifically, it is shown that the computation of this measure can be intractable in data complexity, but answer-set programs are exhibited that can be used to compute it. Furthermore, its is established that there are polynomial-time deterministic and randomized approximations. The behavior of this measure under small updates is analyzed, obtaining fixed-parameter tractability results. We explore abstract extensions of this measure that appeal to generic classes of database repairs. Inconsistency measures and repairs at the attribute level are investigated as a particular, but relevant and natural case.

1 Introduction

Intuitively, a relational database may be more or less consistent than other databases with the same schema, and with respect to the same integrity constraints (ICs). This comparison can be accomplished by assigning a *measure of inconsistency* to databases, which represents a quantitative degree of satisfaction of the intended ICs by the database. In this work we propose such an inconsistency measure, we investigate its computational properties, and we propose a generalization and abstraction that gives rise to a whole family of inconsistency measures that depend on how consistency is restored.

The problem of measuring inconsistency has been investigated mostly by the knowledge representation (KR) community, but scarcely by the data management community. Furthermore, the approaches and results obtained in KR do not immediately apply to databases, or do not address problems that are natural and relevant in databases, such as the computational complexity in terms of the size of the database, i.e. in data complexity. Actually, several (in)consistency measures have been considered in KR [20,21,32], mostly for propositional knowledge bases, or have been applied with grounded first-order representations, obtaining in essence a propositional representation. It becomes interesting to

Member of the "Millenium Institute for Foundational Research on Data" (IMFD, Chile).

M. Balduccini et al. (Eds.): LPNMR 2019, LNAI 11481, pp. 195–209, 2019.
https://doi.org/10.1007/978-3-030-20528-7_15

consider inconsistency measures that are closer to database applications, and whose formulation and computation stay at the relational, first-order level.

In this work we make these ideas concrete by introducing and investigating a particular and natural inconsistency measure. We provide an approach to the computation of the inconsistency measure that is based on *answer-set programming* (ASP) [9], also known as *logic programming with stable model semantics* [19]. This is a natural choice since: (a) an inconsistency measure is non-monotonic in general; (b) the complexity results for its computation show that ASPs provide the exact expressive and computational power needed to compute this measure; (c) database repairs are the basis for the measure, and there are already ASPs that specify them [12] (more on this point below).

The investigation we carry out for the particular inconsistency measure is, independently from possible alternative measures, interesting *per se*: In addition to staying at the relational level, we stress computability and complexity issues in terms of the size of the database. This provides a pattern for the investigation of other possible consistency measures, along similar lines. We are not aware of research that emphasizes computational aspects of inconsistency measures; and here we start filling in this gap. It is likely that other possible consistency measures in the relational setting are also polynomially-reducible to the one we investigate here (or the other way around), and results for one of them can be leveraged for the others. This is a matter of future research.

The particular inconsistency measure we investigate in detail is motivated by one used before to measure the degree of satisfaction of *functional dependencies* (FDs) in a relational database [25]. We extend and reformulate it in terms of database repairs that are based on tuple deletions.[1] As such, it can be applied to the larger class of *denial constraints* [3], and even more, to any class of monotonic ICs (in the sense that, as the database grows, only more violations can be added). However, this measure can also be applied to non-monotonic classes of ICs, such as inclusion- and tuple-generating dependencies, as long as we repair, i.e. restore consistency, only through tuple deletions.[2] Actually, the connection between the inconsistency measure and database repairs motivates our use of ASPs for its computation: We can rely on ASPs that specify and compute the repairs of a database (cf. [3] for a survey and references).

The particular connection of the inconsistency measure and a particular class of database repairs is used here as a basis for proposing more general and abstract inconsistency measures, which have origin in different classes of repairs. From this point of view, we can capture the intuition that the inconsistency degree of a database D with respect to (wrt.) a set of ICs Σ depends on how complex it is to restore consistency (as represented by the admissible class of *repairs* of D wrt. Σ). More technically, our take is that a degree of inconsistency depends

[1] Intuitively, a repair of an inconsistent database D is an alternative instance for the same schema that satisfied the given ICs, and is "maximally close" to D.

[2] The measure can be easily redefined using the symmetric difference between the original database and the repairs when tuple insertions are also allowed as repair actions.

upon a repair semantics, and then, on the admissible repair actions, and on how close we want stay to the instance at hand.

Our main contributions are the following: (a) We introduce an inconsistency measure that is based on cardinality-repairs (Sect. 3). (b) We introduce answer-set programs to compute the inconsistency-measures (Sect. 4); and we show that they provide the required expressive power (Sect. 5). (c) We obtain data complexity results for the inconsistency measure, showing that its computation (as a decision problem) is NP-complete for denial constraints (DCs) and some classes of FDs (Sect. 5). (d) We obtain deterministic and randomized PTIME approximation results for the inconsistency measure, with approximation ratio d (Sect. 5). (e) We establish that the inconsistency measure behaves well under updates, in that small updates keep the inconsistency measure within narrow boundaries. Furthermore, we establish that the computation of the inconsistency measure is fixed-parameter tractable when one starts with a consistent instance, and the parameter is the number of updates (Sect. 6). (f) We introduce a general inconsistency-measure based on an abstract repair-semantics (Sect. 7), and we instantiate it using attribute-based repairs (Sect. 8). (g) We briefly introduce a causality-based notion of contribution of individual tuples to the inconsistency of the database (Sect. 9). All the proofs, additional examples, and an extended discussion can be found in the extended version of this work [1]. All the complexity statements refer to *data complexity*, i.e. in the size of the DB instance at hand.

2 Background on Relational Databases and Repairs

A relational schema $\mathcal{R}$ contains a domain of constants, $\mathcal{C}$, and a set of predicates of finite arities, $\mathcal{P}$. $\mathcal{R}$ gives rise to a language $\mathfrak{L}(\mathcal{R})$ of first-order (FO) predicate logic with built-in equality, $=$. Variables are usually denoted with $x, y, z, ...$, and finite sequences thereof with $\bar{x}, ...$; and constants with $a, b, c, ...$, etc. An *atom* is of the form $P(t_1, \ldots, t_n)$, with n-ary $P \in \mathcal{P}$ and $t_1, \ldots, t_n$ *terms*, i.e. constants, or variables. An atom is *ground* (a.k.a. a tuple) if it contains no variables. A DB instance, D, for $\mathcal{R}$ is a finite set of ground atoms; and it serves as an interpretation structure for $\mathfrak{L}(\mathcal{R})$.

A *conjunctive query* (CQ) is a FOformula, $\mathcal{Q}(\bar{x})$, of the form $\exists \bar{y}\ (P_1(\bar{x}_1 \wedge \cdots \wedge P_m(\bar{x}_m))$, with $P_i \in \mathcal{P}$, and (distinct) free variables $\bar{x} := (\bigcup \bar{x}_i) \smallsetminus \bar{y}$. If $\mathcal{Q}$ has n (free) variables, $\bar{c} \in \mathcal{C}^n$ is an *answer* to $\mathcal{Q}$ from D if $D \models \mathcal{Q}[\bar{c}]$, i.e. $Q[\bar{c}]$ is true in D when the variables in $\bar{x}$ are componentwise replaced by the values in $\bar{c}$. $\mathcal{Q}(D)$ denotes the set of answers to $\mathcal{Q}$ from D. $\mathcal{Q}$ is a *boolean conjunctive query* (BCQ) when $\bar{x}$ is empty; and when *true* in D, $\mathcal{Q}(D) := \{true\}$. Otherwise, it is *false*, and $\mathcal{Q}(D) := \emptyset$. Sometimes CQs are written in Datalog notation as follows: $\mathcal{Q}(\bar{x}) \leftarrow P_1(\bar{x}_1), \ldots, P_m(\bar{x}_m)$.

We consider as integrity constraints, i.e. sentences of $\mathfrak{L}(\mathcal{R})$: (a) *denial constraints* (DCs), i.e. of the form $\kappa : \neg\exists\bar{x}(P_1(\bar{x}_1) \wedge \cdots \wedge P_m(\bar{x}_m))$, where $P_i \in \mathcal{P}$, and $\bar{x} = \bigcup \bar{x}_i$; and (b) *functional dependencies* (FDs), i.e. of the form

$\varphi\colon \neg\exists\bar{x}(P(\bar{v},\bar{y}_1,z_1) \wedge P(\bar{v},\bar{y}_2,z_2) \wedge z_1 \neq z_2)$.[3] Here, $\bar{x} = \bar{y}_1 \cup \bar{y}_2 \cup \bar{v} \cup \{z_1, z_2\}$, and $z_1 \neq z_2$ is an abbreviation for $\neg z_1 = z_2$. A *key constraint* (KC) is a conjunction of FDs: $\bigwedge_{j=1}^{k} \neg\exists\bar{x}(P(\bar{v},\bar{y}_1) \wedge P(\bar{v},\bar{y}_2) \wedge y_1^j \neq y_2^j)$, with $k = |\bar{y}_1| = |\bar{y}_2|$, and generically y^j stands for the jth variable in $\bar{y}$. For example, $\forall x\forall y\forall z(Emp(x,y) \wedge Emp(x,z) \rightarrow y = z)$, is an FD (and also a KC) that could say that an employee (x) can have at most one salary. This FD is usually written as $EmpName \rightarrow EmpSalary$. In the following, we will include FDs and key constraints among the DCs. If an instance D does not satisfy the set Σ of DCs associated to the schema, we say that D is *inconsistent*, which is denoted with $D \not\models \Sigma$.

When a database instance D does not satisfy its intended ICs, it is *repaired*, by deleting or inserting tuples from/into the database. An instance obtained in this way is a *repair* of D if it satisfies the ICs and minimally departs from D [3]. In this work, mainly to fix ideas and simplify the presentation, we consider mostly sets Σ of ICs that are monotone, in the sense that $D \not\models \Sigma$ and $D \subseteq D'$ imply $D' \not\models \Sigma$. This is the case for DCs.[4] For monotone ICs, repairs are obtained by tuple deletions (later on we will also consider value-updates as repair actions). We introduce the most common repairs of databases wrt. DCs by means of an example.

Example 1. The DB $D = \{P(a), P(e), Q(a,b), R(a,c)\}$ is inconsistent wrt. Σ containing the DCs $\kappa_1 : \neg\exists x\exists y(P(x) \wedge Q(x,y))$, and $\kappa_2 : \neg\exists x\exists y(P(x) \wedge R(x,y))$. Here, $D \not\models \{\kappa_1, \kappa_2\}$.

A *subset-repair*, in short *S-repair*, of D wrt. Σ is a $\subseteq$-maximal subset of D that is consistent, i.e. no proper superset is consistent. The following are S-repairs: $D_1 = \{P(e), Q(a,b), R(a,c)\}$ and $D_2 = \{P(e), P(a)\}$. Under this repair semantics, both repairs are equally acceptable. A *cardinality-repair*, in short a *C-repair*, is a maximum-cardinality S-repair. D_1 is the only C-repair. □

For an instance D and a set Σ of DCs, the sets of S-repairs and C-repairs are denoted with $Srep(D,\Sigma)$ and $Crep(D,\Sigma)$, resp. It holds: $Crep(D,\Sigma) \subseteq Srep(D,\Sigma)$. More generally, for a set Σ of ICs, not necessarily DCs, they can be defined by (cf. [3]): (a) $Srep(D,\Sigma) = \{D' : D' \models \Sigma$, and $D \triangle D'$ is minimal under set inclusion$\}$; and (b) $Crep(D,\Sigma) = \{D' : D' \models \Sigma$, and $D \triangle D'$ is minimal in cardinality$\}$. Here, $D \triangle D'$ is the symmetric set-difference $(D \setminus D') \cup (D' \setminus D)$.

3 An Inconsistency Measure

In this section we consider a concrete inconsistency measure. It is natural, and has been consider already in knowledge representation [21], but its investigation

[3] The variables in $\bar{v}$ do not have to go first in the atomic formulas; what matters is keeping the correspondences between the variables in those formulas.

[4] Put in different terms, a DC is associated to (or is the negation of) a conjunctive queries Q, which is monotone in the usual sense: $D \models Q$ and $D \subseteq D' \Rightarrow D' \models Q$.

in a database context has not been undertaken yet. It has also appeared in [25], as measure g_3, among other possible measures and in a restricted form in relation to the satisfaction of FDs, but it was not analyzed much. Its analysis in terms of applicability and properties in the context of DBs, that we here undertake, should serve as a pattern to follow for the analysis of other possible inconsistency measures for DBs. To fix ideas, we consider only DCs. For them, the repair semantics $Srep(D, \Sigma)$ and $Crep(D, \Sigma)$ provide repairs D' that are is maximally contained in the initial instance D. On this basis, we define:

$$inc\text{-}deg^{s,g_3}(D, \Sigma) := \frac{|D| - max\{|D'| \ : \ D' \in Srep(D, \Sigma)\}}{|D|}, \tag{1}$$

$$inc\text{-}deg^{c,g_3}(D, \Sigma) := \frac{|D| - max\{|D'| \ : \ D' \in Crep(D, \Sigma)\}}{|D|}. \tag{2}$$

The first is relative to S-repairs and the second, to C-repairs.

Example 2. (Example 1 cont.) Here, $Srep(D, \Sigma) = \{D_1, D_2\}$, and $Crep(D, \Sigma) = \{D_1\}$. They provide the inconsistency degrees:

$$inc\text{-}deg^{s,g_3}(D, \Sigma) = \frac{4 - |D_1|}{4} = \frac{1}{4}, \text{ and } inc\text{-}deg^{c,g_3}(D, \Sigma) = \frac{4 - |D_1|}{4} = \frac{1}{4}. \qquad \square$$

It holds $Crep(D, \Sigma) \subseteq Srep(D, \Sigma)$, but $max\{|D'| \ : \ D' \in Crep(D, \Sigma)\} = max\{|D'| \ : \ D' \in Srep(D, \Sigma)\}$, so it holds $inc\text{-}deg^{s,g_3}(D, \Sigma) = inc\text{-}deg^{c,g_3}(D, \Sigma)$. This measure always takes a value between 0 and 1. The former when D is consistent (so it itself is its only repair). This measure will be generalized in Sect. 7. Before that, in the next sections we investigate this measure of inconsistency.

4 ASP-Based Computation of the Inconsistency Measure

We concentrate here on the computation of the inconsistency measure $inc\text{-}deg^{c,g_3}(D, \Sigma)$ in (2), which appeals to repairs in $Crep(D, \Sigma)$. This can be done through a compact specification of repairs by means of ASPs.[5] More precisely, given a database instance D and a set of ICs Σ (not necessarily DCs), it is possible to write an ASP whose intended models, i.e. the *stable models* or *answer sets*, are in one-to-one correspondence with the S-repairs of D wrt. Σ (cf. [12] for a general formulation). Here we show only some cases of ICs and examples. In them we use, only to ease the formulation and presentation, global unique tuple identifiers (tids), i.e. every tuple $R(\bar{c})$ in D is represented as $R(t; \bar{c})$ for some integer (or constant) t that is not used by any other tuple in D.

If Σ is a set of DCs containing $\kappa : \neg\exists\bar{x}(P_1(\bar{x}_1) \wedge \cdots \wedge P_m(\bar{x}_m))$, we first introduce for a predicate P_i of the database schema, a nickname predicate P_i' that has, in addition to a first attribute for tids, an extra, final attribute to

[5] This approach was followed in [2] to compute maximum *responsibility degrees* of database tuples as causes for violations of DCs, appealing to a causality-repair connection [7].

hold an annotation from the set $\{\mathsf{d}, \mathsf{s}\}$, for "delete" and "stays", resp. Nickname predicates are used to represent and compute repairs. Next, the *repair-ASP*, $\Pi(D, \Sigma)$, for D and Σ contains all the tuples in D as facts (with tids), plus the following rules for κ:

$$P'_1(t_1; \bar{x}_1, \mathsf{d}) \vee \cdots \vee P'_m(t_n; \bar{x}_m, \mathsf{d}) \leftarrow P_1(t_1; \bar{x}_1), \ldots, P_m(t_m; \bar{x}_m).$$
$$P'_i(t_i; \bar{x}_i, \mathsf{s}) \leftarrow P_i(t_i; \bar{x}_i), \ \textit{not}\ P'_i(t_i; \bar{x}_i, \mathsf{d}). \qquad i = 1, \cdots, m.$$

A stable model M of the program determines a repair D' of D: $D' := \{P(\bar{c}) \mid P'(t; \bar{c}, \mathsf{s}) \in M\}$, and every repair can be obtained in this way [5,12].

For an FD in Σ, say φ: $\neg\exists xyz_1z_2vw(R(x, y, z_1, v) \wedge R(x, y, z_2, w) \wedge z_1 \neq z_2)$, which makes the third attribute functionally depend upon the first two, the repair program contains the rules:

$$R'(t_1; x, y, z_1, v, \mathsf{d}) \vee R'(t_2; x, y, z_2, w, \mathsf{d}) \leftarrow R(t_1; x, y, z_1, v), R(t_2; x, y, z_2, w),$$
$$z_1 \neq z_2.$$
$$R'(t; x, y, z, v, \mathsf{s}) \leftarrow R(t; x, y, z, v), \ \textit{not}\ R'(t; x, y, z, v, \mathsf{d}).$$

For DCs and FDs, the repair programs can be made *normal*, i.e. non-disjunctive, by moving all the disjuncts but one, in turns, in negated form to the body of the rule [12]. For example, the rule $P(a) \vee R(b) \leftarrow \textit{Body}$, can be written as the two rules $P(a) \leftarrow \textit{Body}, \textit{not}\ R(b)$ and $R(b) \leftarrow \textit{Body}, \textit{not}\ P(a)$.[6] Still the resulting program can be *non-stratified* if there is recursion via negation [18], e.g. for FDs and DCs with self-joins.

Example 3. (Example 1 cont.) The initial instance with tids is $D = \{P(1, e), P(2, a), \ Q(3, a, b), \ R(4, a, c), \}$. The repair program contains the following rules, with the first and second for κ_1 and κ_2, resp.:

$$P'(t_1; x, \mathsf{d}) \vee Q'(t_2; x, y, \mathsf{d}) \leftarrow P(t_1; x), Q(t_2; x, y).$$
$$P'(t_1; x, \mathsf{d}) \vee R'(t_2; x, y, \mathsf{d}) \leftarrow P(t_1; x), R(t_2; x, y).$$
$$P'(t; x, \mathsf{s}) \leftarrow P(t; x), \ \textit{not}\ P'(t; x, \mathsf{d}). \qquad \text{etc.}$$

The *repair program* $\Pi(D, \{\kappa_1, \kappa_2\})$ has the stable models: $\mathcal{M}_1 = \{P'(1, e, \mathsf{s}), Q'(3, a, b, \mathsf{s}), \ R'(4, a, c, \mathsf{s}), \ P'(2, a, \mathsf{d})\} \cup D$ and $\mathcal{M}_2 = \{P'(1, e, \mathsf{s}), P'(2, a, \mathsf{s}), Q'(3, a, b, \mathsf{d}), R'(4, a, c, \mathsf{d})\} \cup D$, which correspond to the S-repairs D_1, D_2, resp. □

In order to compute $\textit{inc-deg}^{c,g_3}(D, \Sigma)$ via C-repairs, we need to specify the latter, which can be achieved by adding to Π: (a) rules to collect the *tids* of deleted tuples; (b) a rule with aggregation to compute the number of deleted tuples; and (c) a *weak program-constraint* (WC) [26] that eliminates all the stable models (equivalently, S-repairs) that violate the body of the WC a non-minimum number of times:

$$\text{(a)} \quad \textit{Del}(t) \leftarrow P'_i(t, \bar{x}_i, \mathsf{d}). \qquad i = 1, \ldots, m$$
$$\text{(b)} \quad \textit{NumDel}(n) \leftarrow \#\textit{count}\{t : \textit{Del}(t)\} = n. \qquad \text{(c)} \quad :\sim \textit{Del}(t).$$

[6] This transformation preserves the semantics, because these repair-ASPs turn out to be head-cycle-free [12].

With them, in each model of the program the tids of deleted tuples are collected and counted; only the models where the number of deletions is a minimum are kept.[7] With the WC we keep only cardinality repairs, but not the S-repairs that are maximal, but not maximum subinstances of D.

Example 4. (Example 3 cont.) If we add to Π the rule $Del(t) \leftarrow R'(t, x, y, \mathsf{d})$, similarly for Q' and P'; and the rule counting the deleted tuples, $NumDel(n) \leftarrow \#count\{t : Del(t)\} = n$, the stable model $\mathcal{M}_1$ of the original program would be extended with the atoms $Del(2), NumDel(1)$. Similarly for $\mathcal{M}_2$. If we also add the WC $:\sim Del(t)$, only (the extended) model $\mathcal{M}_1$ remains. It corresponds to the only C-repair. □

The value for *NumDel* in any of the remaining models can be used to compute $inc\text{-}deg^{c,g_3}(D, \Sigma)$. So, there is no need to explicitly compute all stable models, their sizes, and compare them. This value can be obtained by means of the query "$: - NumDel(x)$?", answered by the extended program under the *brave semantics* (returning answers that hold in *some* stable model). An extended example with DLV-Complex [11,26] is shown in the extended version [1].

Brave reasoning under ASPs with weak constraints is $\Delta_2^P(log(n))$-complete in data complexity, i.e. in the size of the database [10]. As we will see in Sect. 5 (cf. Theorem 1), this complexity matches the intrinsic complexity of the computation of the inconsistency measure.

5 Complexity of the Inconsistency Measure Computation

We recall that the *functional complexity class* $FP^{NP(log(n))}$ contains computation problems whose counterparts as decision problems are in the class $P^{NP(log(n))}$, i.e. they are solvable in polynomial time with a logarithmic number of calls to an NP-oracle [30].

Theorem 1. For DCs, computing $inc\text{-}deg^{c,g_3}(D, \Sigma)$ belongs to the functional class $FP^{NP(log(n))}$; and there is a relational schema and a set of DCs Σ for which computing $inc\text{-}deg^{c,g_3}(D, \Sigma)$ is $FP^{NP(log(n))}$-complete (in data complexity). □

This result still holds for a set $\mathcal{F}$ of two FDs of the form: $A \rightarrow B$, $B \rightarrow C$ [1], which deserves a comment: In [27] it is established that if a set of FDs is "simplifiable", then a C-repair can be computed in polynomial time. Clearly if we can build such a repair, we can immediately compute the inconsistency measure in polynomial time (one C-repair suffices). As expected, the set $\mathcal{F}$ is not simplifiable.

[7] If we had a (hard) program-constraint instead, written $\leftarrow Del(t)$, we would be prohibiting the satisfaction of the rule body (in this case, deletions would be prohibited), and we would be keeping only the models where there are no deletions. This would return no model or the original D depending on whether D is inconsistent or not.

Remark 1. In the following we make use several times of the fact that, for a set Σ of DCs and an instance D, one can build a *conflict-hypergraph*, $CG(D, \Sigma)$, whose vertices are the tuples in D and hyperedges are subset-minimal sets of tuples that simultaneously participate in the violation of one of the DCs in Σ [13,28]. More precisely, for a DC $\kappa\colon\ \neg\exists\bar{x}(P_1(\bar{x}_1) \wedge \ldots \wedge P_l(\bar{x}_l))$ in Σ, $S \subseteq D$ forms a hyperedge, if S satisfies the BCQ associated to κ, $\mathcal{Q}^{\kappa} \leftarrow P_1(\bar{x}_1), \ldots, P_l(\bar{x}_l)$, and S is subset-minimal for this property.[8] A C-repair turns out to be the complement of a minimum-size vertex cover for the conflict-hypergraph; equivalently, of a minimum-size hitting-set for the set of hyperedges; or, equivalently, a maximum-size independent set of $CG(D, \Sigma)$. □

The complexity results above show that the normal ASPs introduced in Sect. 4 have the right expressive power to deal with the computational problem at hand. Despite the high-complexity results above, there is a good polynomial-time algorithm, *appID*, that approximates $\mathit{inc\text{-}deg}^{c,g_3}(D, \Sigma)$.

Theorem 2. There is a polynomial-time, deterministic algorithm that returns an approximation $\mathit{appID}(D, \Sigma)$ to $\mathit{inc\text{-}deg}^{c,g_3}(D, \Sigma)$, with the maximum number d of atoms in a DC in Σ as constant factor: $\mathit{appID}(D, \Sigma) \leq d \times \mathit{inc\text{-}deg}^{c,g_3}(D, \Sigma)$. □

Another approach to the approximate computation of the inconsistency measure is based on randomization applied to a relaxed, linear-programming version of the hitting-set (HS) problem for the set of d-bounded hyperedges (or, equivalently, as vertex-covers in hypergraphs with d-bounded hyperedges). In our case, this occurs when each of the DCs in Σ has a number of atoms bounded by d. In this case, we say Σ is d-bounded, and the hyperedges in the conflict-hypergraph have all size at most d. The algorithm in [15] returns a "small", possibly non-minimum HS, which in our case is a set of database tuples whose removal from D restores consistency. The size of this HS approximates the numerator of the inconsistency measure.

Proposition 1. There is a polynomial-time, randomized algorithm that approximates $\mathit{inc\text{-}deg}^{c,g_3}(D, \Sigma)$ within a ratio d, and with probability $\frac{3}{5}$. □

In this result, d is determined by the fixed set of DCs, and does not depend on D. Actually, as shown in [15], the ratio of the algorithm can be improved to $(d - \frac{8}{\Delta})$, where $\Delta \leq \frac{1}{4}|D|^{\frac{1}{4}}$, and d is the maximum degree of a vertex, i.e. in our case the maximum number of tuples that co-violate a DC.[9] For FDs we have conflict-graphs, and $d = 2$.

[8] More technically, each DC $\kappa\colon\ \neg\exists\bar{x}(P_1(\bar{x}_1) \wedge \ldots \wedge P_l(\bar{x}_l) \wedge \ldots)$ gives rise to conjunctive queries $\mathcal{Q}^{\kappa}_{P_l}(\bar{x}_l) \leftarrow P_1(\bar{x}_1), \ldots, P_l(\bar{x}_l), \ldots$. A tuple $P(\bar{a})$ participates in the violation of κ if $\bar{a}$ is an answer to $\mathcal{Q}^{\kappa}_{P}(\bar{x})$.

[9] It is known that there is no polynomial-time approximation with ratio of the form $(d - \epsilon)$ for any constant ϵ [24].

6 Inconsistency Degree Under Updates

Let us assume we have a $\mathit{inc}\text{-}\mathit{deg}^{s,g_3}(D,\Sigma)$ for an instance D and a set of DCs Σ. If, possibly virtually or hypothetically for exploration purposes, we insert m new tuples into D, the resulting instance, D', may suffer from more IC violations than D. The question is how much can the inconsistency measure change. The next results tell us that the inconsistency degree does not experiments unexpected jumps under small updates. They can be seen as a *sensitivity analysis*, and the result as a *continuity property* of the inconsistency measure.

Proposition 2. Given an instance D and a set Σ of DCs, if $\epsilon \times |D|$ new tuples are added to D, with $0 < \epsilon < 1$, obtaining instance D', then $\mathit{inc}\text{-}\mathit{deg}^{c,g_3}(D',\Sigma) \leq \mathit{inc}\text{-}\mathit{deg}^{c,g_3}(D,\Sigma) + \frac{1}{1+\frac{1}{\epsilon}}$; and $\mathit{inc}\text{-}\mathit{deg}^{c,g_3}(D,\Sigma) \leq \frac{1}{1-\epsilon} \times \mathit{inc}\text{-}\mathit{deg}^{c,g_3}(D',\Sigma)$. □

When tuples are deleted, the number of DC violations can only decrease, but also the reference size of the database decreases. However, the inconsistency degree stays within a tight upper bound.

Proposition 3. Given an instance D and a set Σ of DCs, if $\epsilon \times |D|$ tuples are deleted from D, with $0 < \epsilon < 1$, obtaining instance D', then $\mathit{inc}\text{-}\mathit{deg}^{c,g_3}(D',\Sigma) \leq \frac{1}{1-\epsilon} \times \mathit{inc}\text{-}\mathit{deg}^{c,g_3}(D,\Sigma)$; and $\mathit{inc}\text{-}\mathit{deg}^{c,g_3}(D,\Sigma) \leq \frac{1}{1-\epsilon} \times \mathit{inc}\text{-}\mathit{deg}^{c,g_3}(D',\Sigma) + \epsilon$. The last term can be dropped if the tuples deleted form D did not participate in DC violations. □

A natural situation occurs when D is consistent wrt. a set Σ of DCs, and one adds a set U of m tuples (deletions will not affect consistency). It turns out that if Σ is d-bounded, then computing the inconsistency measure is fixed-parameter tractable [16], where the fixed parameter is m.

Theorem 3. For a fixed set of d-bounded DCs Σ, a database D that is consistent wrt. Σ, and U a set of extra tuples, computing $\mathit{inc}\text{-}\mathit{deg}^{c,g_3}(D \cup U,\Sigma)$ is *fixed-parameter tractable* with parameter $m = |U|$. More precisely, there is an algorithm that computes the inconsistency measure in time $O(log(m) \times (C^m + mN))$, where $N = |D|$, $m = |U|$, and C is a constant that depends on d. □

The complexity is exponential in the number of updates, but linear in the size of the initial database. In many situations, m would be relatively small in comparison to $|D|$.

7 Repair Semantics and Inconsistency Degrees

In general terms, a *repair semantics* S for a schema $\mathcal{R}$ that includes a set Σ of ICs assigns to each instance D for $\mathcal{R}$ (which may not satisfy Σ), a class $\mathit{Rep}^{\mathsf{S}}(D,\Sigma)$ of S-*repairs* of D wrt. Σ. These are the instances for $\mathcal{R}$ that satisfy Σ and minimally depart from D according to some minimization criterion. Beside the repairs introduced in Example 1, several repair semantics have been investigated, e.g. *prioritized repairs* [31], *attribute-based repairs* that change attribute values

by other data values [33], or by a null value, NULL, as in SQL databases [2]. The latter will be retaken in Sect. 8.

According to our take on how an inconsistency degree depends on database repairs, we define the *inconsistency degree* of an instance D wrt. a set of ICs Σ in relation to a given repair semantics S. Namely, as the distance from D to the class $Rep^{\mathsf{S}}(D, \Sigma)$:

$$\textit{inc-deg}^{\mathsf{S}}(D, \Sigma) := \textit{dist}(D, Rep^{\mathsf{S}}(D, \Sigma)). \tag{3}$$

This is an abstract measure that depends on S and a numerical function that gives the distance, $\textit{dist}(W, \mathcal{W})$, from a world W to a set $\mathcal{W}$ of possible worlds, which in this case are database instances. Under the assumption that any repair semantics should return D when D is consistent wrt. Σ and $\textit{dist}(D, \{D\}) = 0$, a consistent instance D should have 0 as inconsistency degree.[10]

The class $Rep^{\mathsf{S}}(D, \Sigma)$ might contain instances that are not sub-instances of D, for example, for different forms of *inclusion dependencies* (INDs) we may want to insert tuples;[11] or even under DCs, we may want to appeal to attribute-based repairs. *In the rest of this section, we consider only repairs that are sub-instances of the given instance.* Still this leaves much room open for different kinds of repairs. For example, we may prefer to delete some tuples over others [31]. Or, as in database causality [7,29], the database can be partitioned into *endogenous* and *exogenous* tuples, assuming we have more control on the former, or we trust more the latter; and we prefer *endogenous repairs* that delete only, or preferably, endogenous tuples [2]. The consistency measure we have investigated so far can be defined with an abstract class $Rep^{\mathsf{S}}(D, \Sigma)$:

$$\begin{aligned} \textit{inc-deg}^{\mathsf{S},g_3}(D, \Sigma) := \textit{dist}^{g_3}(D, Rep^{\mathsf{S}}(D, \Sigma)) &:= \frac{|D| - \textit{max}\{|D'| : D' \in Rep^{\mathsf{S}}(D, \Sigma)\}}{|D|} \\ &= \frac{\textit{min}\{|D \smallsetminus D'| \ : \ D' \in Rep^{\mathsf{S}}(D, \Sigma)\}}{|D|}. \end{aligned} \tag{4}$$

This measure can be applied more generally as a "quality measure", not only in relation to inconsistency, but also whenever possibly several intended "quality versions" of a dirty database exist, e.g. as determined by additional contextual information [8]. Particularly prominent is the instantiation of (4) on S-repairs (cf. Sect. 3).

The measure in (4) takes the value 1 only when $Rep^{\mathsf{S}}(D, \Sigma) = \emptyset$ (assuming that $\textit{max}\{\ |D'| \ : \ D' \in \emptyset\} = 0$), i.e. the database is *irreparable*, which is never the case for DCs and S-repairs: there is always an S-repair. However, it could be irreparable with different, but related repair semantics. As mentioned before, in database causality [29] tuples can be endogenous or exogenous, being the former those we can play with, e.g. applying virtual updates on them, producing

[10] Abstract distances between two point-sets are investigated in [14], with their computational properties. Our setting is a particular case.

[11] For INDs repairs based only on tuple deletions can be considered [13].

counterfactual scenarios. One can define *endogenous repairs* as those obtained updating only endogenous tuples [7].

Example 5. (Example 2 cont.) Assume D is partitioned into endogenous and exogenous tuples, say resp. $D = D^n \dot{\cup} D^x$, with $D^n = \{Q(a,b), R(a,c)\}$ and $D^x = \{P(a), P(e)\}$. In this case, the *endogenous-repair semantics* that allows only a minimum number of deletions of endogenous tuples, defines the class of repairs: $Crep^n(D,\Sigma) = \{D_2\}$, with D_2 as above. In this case,[12] $inc\text{-}deg^{c,n,g_3}(D,\Sigma) = \frac{4-2}{4} = \frac{1}{2}$. Similarly, if now $D^n = \{P(a), Q(a,b)\}$ and $D^x = \{P(e), R(a,c)\}$, there are no endogenous repairs, and $inc\text{-}deg^{c,n,g_3}(D,\Sigma) = 1$. □

8 Adapting $inc\text{-}deg^{s,g_3}$ to Attribute-Based Repairs

Database repairs that are based on changes of attribute values in tuples have been considered in [6,33], and implicitly in [4]. In this section we adapt the inconsistency measure we have considered so far, to make it depend upon attribute-repairs. We emphasize that these repairs may not be subinstances of the initial instance even in the presence of DCs. We rely here on repairs introduced in [2], which we show with an example.[13]

Example 6. For the database instance $D = \{S(a_2), S(a_3), R(a_3,a_1), R(a_3,a_4), R(a_3,a_5)\}$, and the DC $\kappa : \neg\exists x\exists y(S(x) \wedge R(x,y))$, it holds $D \not\models \kappa$. Notice that value a_3 matters here in that it enables the join, e.g. $D \models S(a_3) \wedge R(a_3,a_1)$, which could be avoided by replacing it by a null value as used in SQL databases.

More precisely, for the instance $D_1 = \{S(a_2), S(a_3), R(null,a_1), R(null,a_4), R(null,a_5)\}$, where *null* stands for the null value, which cannot be used to satisfy a join, it holds $D_1 \models \kappa$. Similarly with $D_2 = \{S(a_2), S(null), R(a_3,a_1), R(a_3,a_4), R(a_3,a_5)\}$, and $D_3 = \{S(a_2), S(null), R(null,a_1), R(null,a_4), R(null,a_5)\}$, among others obtained from D through replacement of attribute values by *null*. □

In relation to the special constant *null* we assume that all atoms with built-in comparisons, say $null\,\theta\,null$, and $null\,\theta\,c$, with c a non-null constant, are all false for $\theta \in \{=, \neq, <, >, \ldots\}$. In particular, since a join, say $R(\ldots,x) \wedge S(x,\ldots)$, can be written as $R(\ldots,x) \wedge S(x',\ldots) \wedge x = x'$, it can never be satisfied through *null*. This assumption is compatible with the use of NULL in SQL databases (cf. [5, sect. 4] for a detailed discussion, also [4, sect. 2]). Changes of attribute values by *null* as repair actions offer a natural and deterministic solution. It appeals to *the* generic data value used in SQL databases to represent the uncertainty and

[12] For certain forms of *prioritized repairs*, such as endogenous repairs, the normalization coefficient $|D|$ might be unnecessarily large. In this particular case, it might be better to use $|D^n|$.

[13] We believe the developments in this section could be applied to inconsistency measures based on repairs that update attribute values using other constants from the domain [6,33].

incompleteness of the database that inconsistency produces. In order to keep track of changes, we introduce numbers as first arguments in tuples, as global, unique tuple identifiers (tids).

Example 7. (Example 6 cont.) With tids D becomes $D = \{S(1; a_2), S(2; a_3), R(3; a_3, a_1), R(4; a_3, a_4), R(5; a_3, a_5)\}$; and D_1 becomes $D_1 = \{S(1; a_2), S(2; a_3), R(3; null, a_1), R(4; null, a_4), R(5; null, a_5)\}$. The changes are collected in $\Delta^{null}(D, D_1) := \{R[3; 1], R[4; 1], R[5; 1]\}$, showing that (the original) tuple (with tid) 3 has its first-argument changed into *null*, etc. Similarly, $\Delta^{null}(D, D_2) := \{S[2; 1]\}$, and $\Delta^{null}(D, D_3) := \{S[2; 1], R[3; 1], R[4; 1], R[5; 1]\}$.

D_1 and D_2 are the only repairs based on attribute-value changes (into *null*) that are minimal under set inclusion of changes. More precisely, they are consistent, and there is not other consistent repaired version of this kind D' for which $\Delta^{null}(D, D') \subsetneqq \Delta^{null}(D, D_1)$. Similarly for D_2. We denote this class of repairs (and the associated repair semantics) by $Srep^{null}(D, \Sigma)$. Since $\Delta^{null}(D, D_1) \subsetneqq \Delta^{null}(D, D_3)$, $D_3 \notin Srep^{null}(D, \{\kappa\})$. So, $Srep^{null}(D, \{\kappa\}) = \{D_1, D_2\}$.

As with S-repairs, we can consider the subclass of repairs that minimize the number of changes, denoted $Crep^{null}(D, \Sigma)$. In this example, $Crep^{null}(D, \{\kappa\}) = \{D_2\}$ □

Inspired by (4), we define:

$$inc\text{-}deg^{c,null,g_3}(D, \Sigma) := \frac{min\{|\Delta^{null}(D, D')| \ : \ D' \in Crep^{null}(D, \Sigma)\}}{|atv(D)|},$$

where $atv(D)$ is the number of values in attributes of tuples in D.

Example 8. (Example 7 cont.) $inc\text{-}deg^{c,null,g_3}(D, \{\kappa\}) = \frac{1}{8}$, but $inc\text{-}deg^{c,g_3}(D, \{\kappa\}) = \frac{1}{5}$. Here, it is easy to restore consistency: only one attribute value has to be changed. □

The computation of this measure can be done on the basis of ASPs that specify null-based attribute repairs that were introduced in [2], to specify and compute causes for query answers at the attribute level.

9 Tuple-Level Inconsistency Degrees

The inconsistency measure is global in that it applies to the whole database. However, one could also investigate and measure the contribution by individual tuples to the degree of inconsistency of the database. Such local measures have been investigated before in a logical setting [22]. In our case, the global inconsistency measure can be expressed in terms of the *responsibility* of tuples as *causes* for the violation of the DCs.

Connections between database causality [29] and repairs were investigated in [7], where it was established that the *responsibility* of a tuple τ as a cause for $D \not\models \Sigma$ is:

$$\rho_{D,\Sigma}(\tau) = \frac{1}{|D| - \max(|S|)},$$

where $S \subseteq D$ is an S-repair of D wrt. Σ and $\tau \notin S$ (but $\rho_{D,\Sigma}(\tau) := 0$ if there is not such an S). Combining this with (1) and (2), we can see that

$$inc\text{-}deg^{c,g_3}(D,\Sigma) = \frac{1}{\rho_{D,\Sigma}(\tau) \times |D|}, \tag{5}$$

where τ is one and any of the *maximum-responsibility* tuples τ as causes for $D \not\models \Sigma$. We can also consider the responsibility of tuple, $\rho_{D,\Sigma}(\tau)$, as its degree of contribution to the inconsistency of the database, and those with the highest responsibility as those with a largest degree of contribution. According to (5), the global inconsistency measure turns out to be an aggregation over local, tuple-level, degrees of inconsistency.

10 Conclusions

We have scratched the surface of some of the problems and research directions we considered in this work. Certainly all of them deserve further investigation, most prominently, the analysis of other possible distance-based inconsistency measures along the lines of our work; and also the relationships between those measures. Also a deeper analysis of the incremental case (cf. Sect. 6) would be interesting. It is also left for ongoing and future research establishing a connection to the problem of computing specific repairs, and using them [27]. The same applies to the use of the inconsistency measure to explore the *causes for inconsistency*, in particular, to analyze how the measure changes when tuples or combinations thereof are removed from the database. Such an application sounds natural given the established connection between database repairs, causality and causal responsibility [2,7].

It is natural to think of a principled, postulate-based approach to inconsistency measures, similar in spirit to postulates for belief-updates [23]. This has been done in logic-based knowledge representation [20], but as we argued before, a dedicated, specific approach for databases becomes desirable.

In relation to the abstract setting of Sect. 7, we could consider a class $Rep^{\mathsf{S}\preceq}(D,\Sigma)$ of *prioritized repairs* [31], and through them introduce *prioritized measures of inconsistency*. Repair programs for the kinds of priority relations $\preceq$ investigated in [31] could be constructed from the ASPs introduced and investigated in [17] for capturing different optimality criteria. The repair programs could be used to specify and compute the corresponding prioritized inconsistency measures.

Acknowledgments. Research supported by NSERC Discovery Grant #06148. The author is grateful to Jordan Li for his help with DLV; and to Benny Kimelfeld, Sudeepa Roy and Ester Livshits for stimulating general conversations. The author appreciates the support from RelationalAI, and its excellent human and research environment.

References

1. Bertossi, L.: Repair-based degrees of database inconsistency: computation and complexity. Corr arxiv Paper cs.DB/1809.10286 (2018). (extended version of this work)
2. Bertossi, L.: Characterizing and computing causes for query answers in databases from database repairs and repair programs. In: Ferrarotti, F., Woltran, S. (eds.) FoIKS 2018. LNCS, vol. 10833, pp. 55–76. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-90050-6_4
3. Bertossi, L.: Database Repairing and Consistent Query Answering. Synthesis Lectures on Data Management. Morgan & Claypool, San Rafael (2011)
4. Bertossi, L., Li, L.: Achieving data privacy through secrecy views and null-based virtual updates. IEEE Trans. Knowl. Data Eng. **25**(5), 987–1000 (2013)
5. Bertossi, L., Bravo, L.: Consistency and trust in peer data exchange systems. Theory Pract. Log. Program. **17**(2), 148–204 (2017)
6. Bertossi, L., Bravo, L., Franconi, E., Lopatenko, A.: The complexity and approximation of fixing numerical attributes in databases under integrity constraints. Inf. Syst. **33**(4), 407–434 (2008)
7. Bertossi, L., Salimi, B.: From causes for database queries to repairs and model-based diagnosis and back. Theory Comput. Syst. **61**(1), 191–232 (2017)
8. Bertossi, L., Rizzolo, F., Jiang, L.: Data quality is context dependent. In: Castellanos, M., Dayal, U., Markl, V. (eds.) BIRTE 2010. LNBIP, vol. 84, pp. 52–67. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22970-1_5
9. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Commun. ACM **54**(12), 93–103 (2011)
10. Buccafurri, F., Leone, N., Rullo, P.: Enhancing disjunctive datalog by constraints. IEEE Trans. Knowl. Data Eng. **12**(5), 845–860 (2000)
11. Calimeri, F., Cozza, S., Ianni, G., Leone, N.: An ASP system with functions, lists, and sets. In: Erdem, E., Lin, F., Schaub, T. (eds.) LPNMR 2009. LNCS (LNAI), vol. 5753, pp. 483–489. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04238-6_46
12. Caniupan-Marileo, M., Bertossi, L.: The consistency extractor system: answer set programs for consistent query answering in databases. Data Knowl. Eng. **69**(6), 545–572 (2010)
13. Chomicki, J., Marcinkowski, J.: Minimal-change integrity maintenance using tuple deletions. Inf. Comput. **197**(1–2), 90–121 (2005)
14. Eiter, T., Mannila, H.: Distance measures for point sets and their computation. Acta Informatica **34**, 109–133 (1997)
15. El Oualia, M., Fohlin, H., Srivastav, A.: A randomised approximation algorithm for the hitting set problem. Theor. Comput. Sci. **555**, 23–34 (2014)
16. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006). https://doi.org/10.1007/3-540-29953-X
17. Gebser, M., Kaminski, R., Schaub, T.: Complex optimization in answer set programming. Theory Pract. Log. Program. **11**(4–5), 821–839 (2011)
18. Gelfond, M., Kahl, Y.: Knowledge Representation and Reasoning, and the Design of Intelligent Agents. Cambridge University Press, Cambridge (2014)
19. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Gener. Comput. **9**(3/4), 365–386 (1991)
20. Grant, J., Martinez, M.V. (eds.): Measuring Inconsistency in Information. College Publications (2018)

21. Grant, J., Hunter, A.: Analysing inconsistent information using distance-based measures. Int. J. Approx. Reason. **89**, 3–26 (2017)
22. Hunter, A., Konieczny, S.: On the measure of conflicts: shapley inconsistency values. Artif. Intell. **174**(14), 1007–1026 (2010)
23. Katsuno, H., Mendelzon, A.O.: Propositional knowledge base revision and minimal change. Artif. Intell. **52**(3), 263–294 (1992)
24. Khot, S., Regev, O.: Vertex cover might be hard to approximate to within 2-epsilon. J. Comput. Syst. Sci. **74**(3), 335–349 (2008)
25. Kivinen, J., Mannila, H.: Approximate inference of functional dependencies from relations. Theor. Comput. Sci. **149**, 129–l49 (1995)
26. Leone, N., et al.: The DLV system for knowledge representation and reasoning. ACM Trans. Comput. Logic. **7**(3), 499–562 (2006)
27. Livshits, E., Kimelfeld, B., Roy, S.: Computing optimal repairs for functional dependencies. In: Proceedings of PODS 2018, pp. 225–237 (2018)
28. Lopatenko, A., Bertossi, L.: Complexity of consistent query answering in databases under cardinality-based and incremental repair semantics. In: Schwentick, T., Suciu, D. (eds.) ICDT 2007. LNCS, vol. 4353, pp. 179–193. Springer, Heidelberg (2006). https://doi.org/10.1007/11965893_13
29. Meliou, A., Gatterbauer, W., Moore, K.F., Suciu, D.: The complexity of causality and responsibility for query answers and non-answers. In: Proceedings of VLDB 2010, pp. 34–41 (2010)
30. Papadimitriou, C.: Computational Complexity. Addison-Wesley, Boston (1994)
31. Staworko, S., Chomicki, J., Marcinkowski, J.: Prioritized repairing and consistent query answering in relational databases. Ann. Math. Artif. Intell. **64**(2–3), 209–246 (2012)
32. Thimm, M.: On the compliance of rationality postulates for inconsistency measures: a more or less complete picture. Künstliche Intelligenz **31**(1), 31–39 (2017)
33. Wijsen, J.: Database repairing using updates. ACM Trans. Database Syst. **30**(3), 722–768 (2005)