Approximate Query Answering over Inconsistent Knowledge Bases

Sergio Greco, Cristian Molinaro, Irina Trubitsyna

DIMES, University of Calabria
{greco,cmolinaro,trubitsyna}@dimes.unical.it

Abstract. Consistent query answering is a principled approach for querying inconsistent knowledge bases. It relies on the central notion of *repair*, that is, a maximal consistent subset of the facts in the knowledge base. One drawback of this approach is that entire facts are deleted to resolve inconsistency, even if they may still contain useful "reliable" information.
To overcome this limitation, we propose an inconsistency-tolerant semantics for query answering based on a new notion of repair, allowing values within facts to be updated for restoring consistency. This more fine-grained repair primitive allows us to preserve more information in the knowledge base. We also introduce the notion of a *universal repair*, which is a compact representation of all repairs and can be computed in polynomial time. Then, we show that consistent query answering in our framework is intractable (coNP-complete). In light of this result, we develop a polynomial time approximation algorithm for computing a sound (but possibly incomplete) set of consistent query answers.

1 Introduction

Reasoning in the presence of inconsistent information is a problem that has attracted much interest in the last decades. Many inconsistency-tolerant semantics for query answering have been proposed, and most of them rely on the notions of *consistent query answer* and *repair*. A consistent answer to a query is a query answer that is entailed by every repair, where a repair is a "maximal" consistent subset of the facts of the knowledge base. Different maximality criteria have been investigated, but all the resulting notions of repair share the same drawback: a fact is either kept or deleted altogether, and deleting entire facts can cause loss of "reliable" information, as illustrated below.

Example 1. Consider the knowledge base (D, Σ), where D contains the following facts:

works

john	cs	nyc
john	math	rome
mary	math	sidney

and Σ is an ontology consisting of the following equality-generating dependency σ:

$$\mathsf{works}(\mathsf{E}_1, \mathsf{D}, \mathsf{C}_1) \wedge \mathsf{works}(\mathsf{E}_2, \mathsf{D}, \mathsf{C}_2) \rightarrow \mathsf{C}_1 = \mathsf{C}_2.$$

SEBD 2018, June 24-27, 2018, Castellaneta Marina, Italy.

As an example, the fact works(john, cs, nyc) *states that* john *is an employee working in the* cs *department located in* nyc*. The dependency σ says that every department must be located in a single city. Clearly, the last two facts violate σ, so every repair would discard either of them.*

If we pose a query asking for the employees' name, the only consistent answer is john*. However, we might consider reliable the information on* mary *being an employee, as the only uncertainty concerns the* math *department and its city—roughly speaking, the information in the first column of the* works *table can be considered "clean".*

To overcome the drawback illustrated above, we propose a notion of repair based on updating values within facts. Update-based repairing allows for rectifying errors in facts without deleting them altogether, thereby preserving consistent values.

Example 2. Consider again the knowledge base of Example 1. Using value updates as the primitive to restore consistency, and assuming that the only uncertain values are math*'s cities, we get the following two repairs:*

john	cs	nyc
john	math	rome
mary	math	rome

john	cs	nyc
john	math	sidney
mary	math	sidney

If we ask again for the employees' name, both mary *and* john *are consistent answers.*

We show that consistent query answering in this setting is coNP-complete. Then, we show how to compute a "universal" repair, which compactly represents all repairs and can be used as a valuable tool to compute approximate query answers.

Example 3. A universal repair for the two repairs of Example 2 is reported below:

john	cs	nyc
john	math	$\perp_1$
mary	math	$\perp_1$

$\perp_1 = \mathsf{rome} \vee \perp_1 = \mathsf{sidney}$

where $\perp_1$ is a labeled null, and the "global" condition at the bottom restricts the admissible values for $\perp_1$, which are rome *and* sidney*.*

2 Related Work

Consistent query answering was first proposed in [1]. Query answering under various inconsistency-tolerant semantics for ontologies expressed in DL languages has been studied in several works (see, e.g., [13,14,3,4]), and in [17,18] for ontologies expressed by fragments of Datalog+/–. [4] have proposed an approach for the approximation of consistent query answers from above and from below. [7] proposed an approach based on three-valued logic to compute a sound but possibly incomplete set of consistent query answers. Different from our proposal, all the approaches above adopt the most common notion of repair, where whole facts are removed. This can cause loss of information, as illustrated in the toy scenario of Example 1—in real-life scenarios, it might well be

the case that facts have much more attributes and only a few of them are involved in inconsistencies.

There have also been different proposals adopting a notion of repair that allows values to be updated [5,2,6]. Our repair strategy behaves similar to the one of [5] in that values on the right-hand side of functional dependencies (FDs) are updated. However, [5] focus on FDs only, while we consider more general EGDs, and they consider repairs that are minimal according to a specific cost model. [2] allow only numerical attributes to be updated, one primary key per relation is allowed, but keys are assumed to be satisfied by the original DB: thus, no repairing is possible w.r.t. keys, while we allow it, and we allow much more general constraints. Our repair strategy can be seen as an instantiation of the value-based family of policies proposed in [19], even though the two approaches differ in how multiple dependencies are handled, and while they focus on FDs only, we consider EGDs. [6] consider numerical databases and a different class of (aggregate) constraints. However, the main difference between this paper and the aforementioned ones is that none of them has investigated the approximate computation of consistent query answers.

Approximation algorithms providing sound but possibly incomplete query answers in the presence of nulls have been proposed in [11,15,16,9], but no dependencies are considered therein, and thus data is assumed to be consistent.

3 Preliminaries

We assume the existence of the following pairwise disjoint (countably infinite) sets: a set Const of *constants*, a set Var of *variables*, and a set Null of *labeled nulls*. Nulls are denoted by the symbol $\perp$ subscripted. A *term* is a constant, variable, or null. We also assume a set of *predicates*, disjoint from the aforementioned sets, with each predicate being associated with an *arity*, which is a non-negative integer.

An *atom* A is of the form $p(t_1, \ldots, t_n)$, where p is an n-ary predicate and the t_i's are terms. We write an atom also as $p(\mathbf{t})$, where $\mathbf{t}$ is a sequence of terms. An atom without variables is also called a *fact*. An *instance* is a finite set of facts. A *database* is an instance containing constants only.

A *homomorphism* is a mapping $h : \mathsf{Const} \cup \mathsf{Var} \cup \mathsf{Null} \to \mathsf{Const} \cup \mathsf{Var} \cup \mathsf{Null}$ that is the identity on Const. Homomorphisms are also applied to atoms and set of atoms in the natural fashion, that is, $h(p(t_1, \ldots, t_n)) = p(h(t_1), \ldots, h(t_n))$, and $h(S) = \{h(A) \mid A \in S\}$ for every set S of atoms. A *valuation* is a homomorphism ν whose image is Const, that is, $\nu(t) \in \mathsf{Const}$ for every $t \in \mathsf{Const} \cup \mathsf{Var} \cup \mathsf{Null}$.

Conditional instances. *Conditional instances* (also known as "conditional tables" [12,8]) are instances augmented with conditions restricting the set of admissible values for nulls. Let $\mathcal{E}$ be the set of all expressions, called *conditions*, that can be built using the standard logical connectives $\wedge$, $\vee$, $\neg$, $\Rightarrow$ and expressions of the form $t_i = t_j$, true, and false, where $t_i, t_j \in \mathsf{Const} \cup \mathsf{Null}$. We will also use $t_i \neq t_j$ as a shorthand for $\neg(t_i = t_j)$.We say that a valuation ν *satisfies* a condition ϕ, denoted $\nu \models \phi$, if its assignment of constants to nulls makes ϕ true. Formally, a *conditional instance* (CI) is a pair (I, Φ), where I is an instance and $\Phi \in \mathcal{E}$. The semantics of $C = (I, \Phi)$ is given by the set of its *possible worlds*, that is, the set of databases $pw(C) = \{\nu(I) \mid \nu \text{ is a valuation and } \nu \models \Phi\}$.

Equality generating dependencies. An *equality generating dependency* (EGD) σ is a first-order formula of the form $\forall \mathbf{x}\, \varphi(\mathbf{x}) \rightarrow x_i = x_j$, where $\varphi(\mathbf{x})$ is a conjunction of atoms (without labeled nulls) whose variables are exactly $\mathbf{x}$, and x_i and x_j are variables from $\mathbf{x}$. We call $\varphi(\mathbf{x})$ the *body* of σ, and call $x_i = x_j$ the *head* of σ. We will omit the universal quantification in front of dependencies and assume that all variables are universally quantified. With a slight abuse of notation, we sometimes treat a conjunction as the *set* of its atoms.

An instance I *satisfies* σ, denoted $I \models \sigma$, if whenever there exists a homomorphism h s.t. $h(\varphi(\mathbf{x})) \subseteq I$, then $h(x_i) = h(x_j)$. A instance I *satisfies* a set Σ of EGDs, denoted $I \models \Sigma$, if $I \models \sigma$ for every $\sigma \in \Sigma$.

Knowledge bases. A *knowledge base* (KB) is a pair (D, Σ), where D is a database and Σ is a finite set of EGDs. It is *consistent* if $D \models \Sigma$, otherwise it is *inconsistent*.

4 Repairing and Querying Inconsistent KBs

In this section, we present our notion of repair and analyze the computational complexity of two central problems: repair checking and consistent query answering.

We start by defining the EGDs we consider. Let Σ be a set of EGDs. An *argument* of Σ is an expression of the form $p[i]$, where p is an n-ary predicate appearing in Σ and $1 \leq i \leq n$.

Definition 1 (Argument and dependency graphs). *The* argument graph *of a set Σ of EGDs is a directed graph $G_\Sigma = (V, E)$, where V is the set of all arguments of Σ, and E contains a directed edge from $p[i]$ to $q[j]$ labeled σ iff there is an EGD $\sigma \in \Sigma$ such that:*

- *the body of σ contains an atom $p(t_1, \ldots, t_n)$ such that either t_i is a constant or t_i is a variable occurring more than once in the body of σ, and*
- *the body of σ contains an atom $q(u_1, \ldots, u_m)$ such that u_j is a variable also appearing in the head of σ.*

The dependency graph *of Σ is a directed graph $\Gamma_\Sigma = (\Sigma, \Omega)$, where Ω is the set $\{(\sigma_1, \sigma_2) \mid G_\Sigma = (V, E) \wedge (p([i], q[j], \sigma_1), (q[j], r[k], \sigma_2) \in E\}$. We say that Σ is* acyclic *if its dependency graph is acyclic.*

In the following, we write $\sigma_i < \sigma_j$ if $(\sigma_i, \sigma_j) \in \Omega$ or there is σ_k such that $\sigma_i < \sigma_k$ and $\sigma_k < \sigma_j$.

Example 4. Consider the following set of EGDs Σ:

$$\sigma_1 \colon\ \mathsf{works}(\mathsf{X}, \mathsf{Y}_1, \mathsf{Z}_1) \wedge \mathsf{works}(\mathsf{X}, \mathsf{Y}_2, \mathsf{Z}_2) \rightarrow \mathsf{Y}_1 = \mathsf{Y}_2$$
$$\sigma_2 \colon\ \mathsf{works}(\mathsf{X}_1, \mathsf{Y}, \mathsf{Z}_1) \wedge \mathsf{works}(\mathsf{X}_2, \mathsf{Y}, \mathsf{Z}_2) \rightarrow \mathsf{Z}_1 = \mathsf{Z}_2$$

Then, G_Σ has two edges: one from works[1] *to* works[2] *labeled σ_1, and another one from* works[2] *to* works[3] *labeled σ_2. Moreover, Γ_Σ has only the edge (σ_1, σ_2). Thus, Σ is acyclic and a topological sorting of EGDs is (σ_1, σ_2).*

In the rest of the paper we consider acyclic sets of EGDs only. Thus, from now on, Σ is understood to be acyclic. Before introducing our notion of repair, we provide some intuitions in the following example.

Example 5. Consider the database below and the set of EGDs of Example 4.

john	cs	rome
john	math	rome
mary	math	sidney

By enforcing σ_1 into the first two facts, either cs *or* math *can be chosen as* john*'s department. If the latter is chosen, then the database D' below is obtained.*

Suppose now that σ_2 is enforced into the last two facts of D'. Then, either rome *or* sidney *can be chosen as* math*'s city. If the former is chosen, then the database D'' below is obtained.*

$D' =$

john	math	rome
john	math	rome
mary	math	sidney

$D'' =$

john	math	rome
john	math	rome
mary	math	rome

No further dependency enforcement is applicable at this point and thus D'' is a repair.

When repairing inconsistent databases, conditional instances can be used to keep track of which values must be equal and which constants can be assigned to nulls. For instance, in Example 5 above, after the last dependency enforcement, we have to make sure that the last column contains the same city (which can be either rome or sidney).

Below we introduce the technical definitions, starting with a *repair step*.

Definition 2 (Repair step). *Let (I, Φ) be a CI, Σ a set of EGDs, and σ an EGD $\varphi(\mathbf{x}) \rightarrow x_i = x_j \in \Sigma$. Let h be a homomorphism s.t. $h(\varphi(\mathbf{x})) \subseteq h(I)$, $h \models \Phi$, and $h(x_i) \neq h(x_j)$.*

Moreover, let $\perp_m$ be a fresh null and (I', Φ') the CI obtained from (I, Φ) as follows:

1. *for each fact $p(u_1, ..., u_n)$ in I, if $\varphi(\mathbf{x})$ contains an atom $p(t_1, ..., t_n)$ s.t. $h(p(t_1, ..., t_n)) = h(p(u_1, ..., u_n))$, then for every $1 \leq k \leq n$, if $t_k \in \{x_i, x_j\}$,*
   - *replace u_k in $p(u_1, ..., u_n)$ with $\perp_m$;*
   - *if $u_k \in$ Null, replace every occurrence of u_k elsewhere with $\perp_m$;*
2. *Either $\Phi' = \Phi \wedge (\perp_m = h(x_i))$ or $\Phi' = \Phi \wedge (\perp_m = h(x_j))$.*

We say that $(I, \Phi) \xrightarrow{\sigma,h} (I', \Phi')$ is a repair step.

Intuitively, a repair step enforces an EGD that is not satisfied by the knowledge base.

A *repair sequence* of a knowledge base (D, Σ) is a (possibly empty) finite sequence of repair steps $C_i \xrightarrow{\sigma_i,h_i} C_{i+1}$ $(0 \leq i < m)$ such that $C_0 = (D, \mathsf{true})$ and $\sigma_i \in \Sigma$ for every i. We also say that the repair sequence is *from C_0 to C_m*. We call C_m the *result* of the repair sequence. Also, we say that the repair sequence is *maximal* if there does not exist a repair step of the form $C_m \xrightarrow{\sigma_m,h_m} C_{m+1}$.

It is easy to see that for each repair step $(I, \Phi) \xrightarrow{\sigma,h} (I', \Phi')$, if h' is a homomorphism s.t. $h' \models \Phi'$, then it assigns constants to nulls as dictated by Φ' (see item 2 of Definition 2).

Thus, all h' satisfying Φ' yield the same database $h'(I')$. For any conditional instance (I, Φ), we use the notation $\Phi(I)$ to denote the database derived from I by iteratively replacing nulls with constants or other nulls as dictated by Φ.

For ease of presentation, we assume that one can keep of a given fact f in the database during the repair process despite the value changes. Thus, we use $D[f, i]$ to denote the i-th term of a fact f in a database D. Given a repair sequence S from $C_0 = (D, \mathsf{true})$ to $C_m = (I_m, \Phi_m)$, we define the set of changes made by S as $update(S) = \{(f, i) \mid D[f, i] \neq \Phi_m(I_m)[f, i]\}$, where f is a fact of the database.

Example 6. Consider the database of Example 5 and the set of EGDs of Example 4. By enforcing first σ_1 using the first two facts, then σ_2 using the last two facts, and finally σ_2 using the first two facts, we get, respectively, the CIs C_1, C_2 and C_3 reported below:

C_1

john	$\perp_1$	rome
john	$\perp_1$	rome
mary	math	sidney

$\perp_1 = \mathsf{math}$

C_2

john	$\perp_1$	rome
john	$\perp_1$	$\perp_3$
mary	math	$\perp_3$

$\perp_1 = \mathsf{math} \wedge \perp_3 = \mathsf{sidney}$

C_3

john	$\perp_1$	$\perp_4$
john	$\perp_1$	$\perp_4$
mary	math	$\perp_4$

$\perp_1 = \mathsf{math} \wedge \perp_3 = \mathsf{sidney} \wedge \perp_4 = \mathsf{rome}$

No further repair steps are applicable at this point. Notice that $\perp_3 = \mathsf{sidney}$ can be deleted from the condition of C_3 because it does not play a role anymore. A repair is obtained from C_3 by replacing every occurrence of $\perp_1$ with math*, and every occurrence of $\perp_4$ with* rome*.*

Definition 3 (Repair). *A* repair *for a knowledge base $K = (D, \Sigma)$ is a database $\Phi(J)$ such that there exists a maximal repair sequence S of K from (D, true) to (J, Φ) and $update(S)$ is minimal w.r.t. $\subseteq$.*

We use $repair(D, \Sigma)$ to denote the set of all repairs of a knowledge base (D, Σ). Every repair is indeed consistent [10].

The *consistent answers* to a query Q over a knowledge base (D, Σ) are defined in the standard way as follows: $Q(D, \Sigma) = \bigcap\{Q(D') \mid D' \in repair(D, \Sigma)\}$.

In the context of managing inconsistent knowledge bases, two fundamental problems are *repair checking* and *consistent query answering*, which are defined as follows. Let (D, Σ) be a knowledge base, D' a database, Q a query, and f a fact of constants. Then, the following two problems are defined: *repair checking*: decide whether $D' \in repair(D, \Sigma)$; *consistent query answering*: decide whether $f \in Q(D, \Sigma)$.

Repair checking is in PTIME while conjunctive query answering is coNP-complete in the data complexity [10].

[10] introduced the notion of a *universal repair*, which is a compact representation of all repairs for a given knowledge base, and can be computed in polynomial time. An example is provided below.

Example 7. Consider the database of Example 5 and the EGDs of Example 4. The following CI is a universal repair:

john	$\perp_1$	$\perp_5$
john	$\perp_1$	$\perp_5$
mary	math	$\perp_4$

$(\perp_1 = \mathsf{cs} \wedge \perp_4 = \mathsf{sidney} \wedge \perp_5 = \mathsf{rome}) \vee$
$(\perp_1 = \mathsf{math} \wedge \perp_5 = \perp_4 \wedge (\perp_4 = \mathsf{rome} \vee \perp_4 = \mathsf{sidney}))$

5 Approximation Algorithm

Since consistent query answering in intractable in our framework, [10] developed a polynomial time approximation algorithm to compute a sound (but possibly incomplete) set of consistent query answers. The basic idea is illustrated in the following example.

Example 8. Consider the knowledge base of Example 1 and the query asking for the pairs of departments located in different cities.

A universal repair, say U, is shown in Example 3. The first step of the approximation algorithm consists in assigning the "local condition" true *to every fact of U as follows:*

john	cs	nyc	true
john	math	$\perp_1$	true
mary	math	$\perp_1$	true

$\perp_1 = \mathsf{rome} \vee \perp_1 = \mathsf{sidney}$

Then, the "conditional evaluation" (cf. [8,10] for more details) of the query is performed—roughly speaking, it means that the query is evaluated using conditions that express how facts are derived. The result is the following set of facts associated with conditions:

cs	cs	$\mathsf{true} \wedge \mathsf{true} \wedge \mathsf{nyc} \neq \mathsf{nyc}$
cs	math	$\mathsf{true} \wedge \mathsf{true} \wedge \mathsf{nyc} \neq \perp_1$
math	cs	$\mathsf{true} \wedge \mathsf{true} \wedge \perp_1 \neq \mathsf{nyc}$
math	math	$\mathsf{true} \wedge \mathsf{true} \wedge \perp_1 \neq \perp_1$

Finally, conditions are evaluated in such a way that each of them yields one of the truth values true, false, unknown. *If the evaluation of a fact's condition yields* true, *then the fact is a consistent query answer. In the scenario above, the approximate consistent answers are* $(\mathsf{cs}, \mathsf{math})$ *and* $(\mathsf{math}, \mathsf{cs})$, *which are indeed consistent query answers.*

6 Conclusion

We proposed a framework for query answering over inconsistent KBs that is based on *(i)* a notion of repair that preserves more information than classical repair strategies, *(ii)* a compact representation of all repairs called universal repair, *(iii)* an approximation algorithm to compute under-approximations of consistent query answers.

As a direction for future work, we plan to investigate further approximation algorithms based on different conditions' evaluations. Another issue to be investigated is the generalization of our framework to more general classes of dependencies, such as TGDs or arbitrary EGDs.

References

1. Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *Symposium on Principles of Database Systems (PODS)*, pages 68–79, 1999.

2. Leopoldo E. Bertossi, Loreto Bravo, Enrico Franconi, and Andrei Lopatenko. The complexity and approximation of fixing numerical attributes in databases under integrity constraints. *Information Systems*, 33(4-5):407–434, 2008.
3. Meghyn Bienvenu. On the complexity of consistent query answering in the presence of simple ontologies. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2012.
4. Meghyn Bienvenu and Riccardo Rosati. Tractable approximations of consistent query answering for robust ontology-based data access. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 775–781, 2013.
5. Philip Bohannon, Michael Flaster, Wenfei Fan, and Rajeev Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *International Conference on Management of Data (SIGMOD)*, pages 143–154, 2005.
6. Sergio Flesca, Filippo Furfaro, and Francesco Parisi. Querying and repairing inconsistent numerical databases. *ACM Transactions on Database Systems*, 35(2):14:1–14:50, 2010.
7. Filippo Furfaro, Sergio Greco, and Cristian Molinaro. A three-valued semantics for querying and repairing inconsistent databases. *Annals of Mathematics and Artificial Intelligence*, 51(2-4):167–193, 2007.
8. Gösta Grahne. *The Problem of Incomplete Information in Relational Databases*, volume 554 of *Lecture Notes in Computer Science*. Springer, 1991.
9. Sergio Greco, Cristian Molinaro, and Irina Trubitsyna. Computing approximate certain answers over incomplete databases. In *Alberto Mendelzon International Workshop (AMW)*, 2017.
10. Sergio Greco, Cristian Molinaro, and Irina Trubitsyna. Computing approximate query answers over inconsistent knowledge bases. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1838–1846, 2018.
11. Paolo Guagliardo and Leonid Libkin. Making SQL queries correct on incomplete databases: A feasibility study. In *Symposium on Principles of Database Systems (PODS)*, pages 211–223, 2016.
12. Tomasz Imielinski and Witold Lipski. Incomplete information in relational databases. *Journal of the ACM*, 31(4):761–791, 1984.
13. Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. Inconsistency-tolerant semantics for description logics. In *International Conference on Web Reasoning and Rule Systems (RR)*, pages 103–117, 2010.
14. Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. Inconsistency-tolerant query answering in ontology-based data access. *Journal of Web Semantics*, 33:3–29, 2015.
15. Leonid Libkin. How to define certain answers. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4282–4288, 2015.
16. Leonid Libkin. Certain answers as objects and knowledge. *Artificial Intelligence*, 232:1–19, 2016.
17. Thomas Lukasiewicz, Maria Vanina Martinez, Andreas Pieris, and Gerardo I. Simari. From classical to consistent query answering under existential rules. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 1546–1552, 2015.
18. Thomas Lukasiewicz, Maria Vanina Martinez, and Gerardo I. Simari. Inconsistency handling in Datalog+/– ontologies. In *European Conference on Artificial Intelligence (ECAI)*, pages 558–563, 2012.
19. Maria Vanina Martinez, Francesco Parisi, Andrea Pugliese, Gerardo I. Simari, and V. S. Subrahmanian. Policy-based inconsistency management in relational databases. *International Journal of Approximate Reasoning*, 55(2):501–528, 2014.