THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

Preference-Driven Querying of Inconsistent Relational Databases

Citation for published version:
Staworko, S, Chomicki, J & Marcinkowski, J 2006, Preference-Driven Querying of Inconsistent Relational Databases. in *Current Trends in Database Technology: EDBT 2006, EDBT 2006 Workshops PhD, DataX, IIDB, IIHA, ICSNW, QLQP, PIM, PaRMA, and Reactivity on the Web, Munich, Germany, March 26-31, 2006, Revised Selected Papers.* vol. 4254, Springer Berlin Heidelberg, pp. 318-335. https://doi.org/10.1007/11896548_26

Digital Object Identifier (DOI):
10.1007/11896548_26

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Current Trends in Database Technology

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

OPEN ACCESS

Download date: 08. Jun. 2023

Preference-Driven Querying of Inconsistent Relational Databases ⋆

Slawomir Staworko[1], Jan Chomicki[1], and Jerzy Marcinkowski[2]

[1] University at Buffalo,
{staworko,chomicki}@cse.buffalo.edu
[2] Wroclaw University
Jerzy.Marcinkowski@ii.uni.wroc.pl

Abstract. One of the goals of cleaning an inconsistent database is to remove conflicts between tuples. Typically, the user specifies how the conflicts should be resolved. Sometimes this specification is incomplete, and the cleaned database may still be inconsistent. At the same time, data cleaning is a rather drastic approach to conflict resolution: It removes tuples from the database, which may lead to information loss and inaccurate query answers.
We investigate an approach which constitutes an alternative to data cleaning. The approach incorporates preference-driven conflict resolution into query answering. The database is not changed. These goals are achieved by augmenting the framework of consistent query answers through various notions of preferred repair. We axiomatize desirable properties of preferred repair families and propose different notions of repair optimality. Finally, we investigate the computational complexity implications of introducing preferences into the computation of consistent query answers.

1 Introduction

In many novel database applications, violations of integrity constraints cannot be avoided. A typical example is integration of two consistent data sources that contribute conflicting information. At the same time the sources are autonomous and cannot be changed. Inconsistencies also occur in the context of long running operations. Finally, integrity enforcement may be neglected because of efficiency considerations.

Integrity constraints, however, often capture important semantic properties of the stored data. These properties directly influence the way a user formulates a query. Evaluation of the query over an inconsistent database may negatively affect the meaning of the answers.

Example 1. Consider the schema

$$Mgr(Name, Dept, Salary, Reports)$$

consistent with two key dependencies:

$$Dept \rightarrow Name\,Salary\,Reports, \qquad (fd_1)$$
$$Name \rightarrow Dept\,Salary\,Reports, \qquad (fd_2)$$

⋆ Research supported by NSF Grants IIS-0119186 and IIS-0307434.

In an instance of this schema a tuple (x,y,z,v) denotes a manager x of the department y with the salary z required to write v reports annually.

Now suppose we integrate the following (consistent) sources:

$$s_1 = \{(Mary, R\&D, 40k, 3)\}, \quad s_2 = \{(John, R\&D, 10k, 2)\},$$
$$s_3 = \{(Mary, IT, 20k, 1), (John, PR, 30k, 4)\}.$$

The integrated instance $r = s_1 \cup s_2 \cup s_3$ contains 3 conflicts:

1. $(Mary, R\&D, 40k, 3)$ and $(John, R\&D, 10k, 2)$ w.r.t. fd_1,
2. $(Mary, R\&D, 40k, 3)$ and $(Mary, IT, 20k, 1)$ w.r.t. fd_2,
3. $(John, R\&D, 10k, 2)$ and $(John, PR, 30k, 4)$ w.r.t. fd_2.

These inconsistencies can be a result of changes that are not yet fully propagated. For example *Mary* may have been promoted to manage *R&D* whose previous manager *John* was moved to manage *PR*, or conversely *John* may have been moved to manage *R&D*, while *Mary* was moved from *R&D* to manage IT.

Consider the query Q_1 asking if *John* earns more than *Mary*:

$$\exists x_1, y_1, z_1, x_2, y_2, z_2 . Mgr(Mary, x_1, y_1, z_1) \wedge Mgr(John, x_2, y_2, z_2) \wedge y_1 < y_2.$$

The answer to Q_1 in r is true but this is misleading because r may not correspond to any actual state of the world.

One way to deal with the impact of inconsistencies in the results of the query evaluation is *data cleaning* [16]. Although there exist a wide variety of tools for automatic elimination of duplicates, extraction and standardization of information, there are practically no tools that automatically resolve integrity constraint violations [18]. Usually, the user is responsible for providing a procedure that decides how the conflicts should be resolved. The standard repertoire of actions that can be performed on a conflicting tuple is [23]: removing the tuple, leaving the tuple, or reporting the tuple to an auxiliary (*contingency*) table. Typically, the data cleaning system provides useful information which may include:

- the timestamp of creation/last modification of the tuple (the conflicts can be resolved by removing from consideration old, outdated tuples),
- source of the information of the tuple (a user can consider the data from one source more reliable than the data from the other).

The approach of data cleaning has several shortcomings:

- If the user provides insufficient information to resolve all the conflicts then data cleaning results in an inconsistent database; this again may lead to misleading answers.
- Physically removing the tuples from the database may lead to information loss.
- Data cleaning doesn't allow to utilize the incomplete information often expressed in inconsistencies.

The framework of *repairs* and *consistent query answers* [1] proposes an alternative approach to deal with inconsistent databases geared towards utilizing incomplete information. A *repair* is a minimally changed consistent database and a *consistent answer* to a query is the answer present in *every* repair. This approach doesn't remove physically any tuples from the database. The framework of [1] has served as a foundation for most of the subsequent work in the area of querying inconsistent databases (for recent developments see[3, 11]).

Example 2. The instance r of Example 1 has 3 repairs:

$$r_1 = \{(Mary, R\&D, 40k, 3), (John, PR, 30k, 4)\},$$
$$r_2 = \{(John, R\&D, 10k, 2), (Mary, IT, 20k, 1)\},$$
$$r_3 = \{(Mary, IT, 20k, 1), (John, PR, 30k, 4)\}.$$

Because Q_1 is false in r_1 and r_2, true is not a consistent answer to Q_1.

The standard framework of consistent query answers does not contain any way to incorporate additional user input about how to resolve some conflicts. One can try to first clean the database and then use the consistent query answers approach. This is a radical approach: removing tuples may lead to information loss. Additional user input in the form of preferences can be used in the framework of consistent query answers to benefit the correctness of consistent query answers by considering only the *preferred* repairs.

Example 3. Suppose the user finds the source s_3 to be less reliable than s_1 and less reliable than s_2. The user does not know, however, the relative reliability of the sources s_1 and s_2. The cleaning of r with this information yields an inconsistent database:

$$r' = \{(Mary, R\&D, 40k, 3), (John, R\&D, 10k, 2)\}.$$

Consider the query Q_2 asking if *Mary* earns more and has fewer reports to write than *John*:

$$\exists x_1, y_1, z_1, x_2, y_2, z_2 . Mgr(Mary, x_1, y_1, z_1) \wedge Mgr(John, x_2, y_2, z_2) \wedge y_1 > y_2 \wedge z_1 < z_2.$$

The answer to this query in the "cleaned" database is false. False is also the consistent answer to Q_2 in r'. Note, however, that neither false nor true is a consistent answer to Q_2 in r.

Intuitively the repairs r_1 and r_2 incorporate more of reliable information than the repair r_3 (all tuples of r_3 come from a less reliable source s_3). If we consider r_1 and r_2 at the only preferred repairs, then true is the *preferred consistent answer* to Q_2.

In our paper we extend the framework of consistent query answers with an additional input consisting of preference information Φ. We use Φ to define the set of preferred repairs Rep^{Φ}. When we compute consistent answers, instead of considering the set of all repairs Rep, we use the set of preferred repairs. We assume that there exists a (possibly partial) operation of extending Φ with some additional preference information and we write $\Phi \subseteq \Psi$ when Ψ is an *extension* of Φ. We consider Φ to be *total* when it cannot be extended further. We identify the following desirable properties of families of preferred repairs:

1. **Non-emptiness**
$$Rep^{\Phi} \neq \varnothing. \qquad (\mathscr{P}1)$$
2. **Monotonicity**: extending preferences can only narrow the set of preferred repairs
$$\Phi \subseteq \Psi \Rightarrow Rep^{\Psi} \subseteq Rep^{\Phi}. \qquad (\mathscr{P}2)$$
3. **Non-discrimination**: if no preference information is given, then no repair is removed from consideration
$$Rep^{\varnothing} = Rep. \qquad (\mathscr{P}3)$$
4. **Categoricity**: given maximal preference information we obtain exactly one repair
$$\Phi \text{ is total} \Rightarrow |Rep^{\Phi}| = 1. \qquad (\mathscr{P}4)$$

In Section 3 we observe, however, that these properties do not enforce practically any use of preference information. To do so we also study different notions of repair *optimality* which ensure a proper use of preference information to select preferred repairs.

2 Preliminaries

In this paper, we work with databases over a schema consisting of only one relation R with attributes from U. We use $A, B, \ldots$ to denote elements of U and $X, Y, \ldots$ to denote subsets of U. We consider two disjoint domains: uninterpreted names D and natural numbers N. Every attribute in U is typed. We assume that constants with different names are different and that symbols $=$, $\neq$, $<$, $>$ have the natural interpretation over N.

The instances of R, denoted by $r, r', \ldots$, can be seen as finite, first-order structures, that share the domains D and N. For any tuple t from r by $t.A$ we denote the value associated with the attribute A. In this paper we consider first-order queries over the alphabet consisting of R and binary relation symbols $=$, $\neq$, $<$, and $>$.

The limitation to only one relation is made only for the sake of clarity and along the lines of [7] the framework can be easily extended to handle databases with multiple relations.

2.1 Inconsistency and repairs

The class of integrity constraints we study consists of functional dependencies. We use $X \to Y$ to denote the following constraint:
$$\forall t_1, t_2 \in R.\ \bigwedge_{A \in X} t_1.A = t_2.A \Rightarrow \bigwedge_{B \in Y} t_1.B = t_2.B \qquad (1)$$

We identify conflicts created by (1) as follows: tuples t_1 and t_2 are *conflicting* if $t_1.A = t_2.A$ for all $A \in X$ and $t_1.B \neq t_2.B$ for some $B \in Y$. A database r is *inconsistent* with a set of constraints F if and only if r contains some conflicting tuples with a constraint from F. Otherwise, the database is *consistent*.

In the general framework when repairing a database we consider two operations: adding or removing a tuple. Because in the presence of functional dependencies adding new tuples cannot remove conflicts, we only consider repairs obtained by deleting tuples from the original instance.

Definition 1 (Repair). *Given a database r and a set of integrity constraints F, a database r' is a* repair *of r w.r.t. F if r' is a maximal subset of r consistent with F. By $Rep_F(r)$ we denote the set of all repairs of r w.r.t F.*

A repair can be viewed as the result of a process of cleaning the input relation. Note that since every conflict can be resolved in two different ways and conflict are often independent, there may be an exponential number of repairs.

Example 4. For any natural number n consider an instance

$$r_n = \{(0,0),(0,1),\ldots,(n-1,0),(n-1,1)\}$$

of the schema $R(A,B)$. Note that the set of all repairs of r_n w.r.t. the functional dependency $A \to B$ is equal to the set $\{0,1\}^n$ of all functions from $\{0,\ldots,n-1\}$ to $\{0,1\}$.

Also note that the set of repairs of a consistent relation r contains only r.

Given a relation instance r and a set of functional dependencies F, a *conflict graph* is a graph whose vertices are the tuples of r and two tuples are adjacent only if they are conflicting w.r.t. a constraint from F. Conflict graphs are *compact representations of repairs* because the set of all repairs is equal to the set of all maximal sets of the corresponding conflict graph.

Example 5. The conflict graph for the instance r_n for $n = 4$ and the functional dependency $A \to B$ from Example 4 is presented in Figure 1.

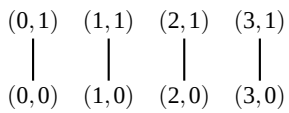

Fig. 1. A conflict graph.

For a given tuple t, by $n(t)$ we denote its *neighborhood* in the conflict graph, i.e. all tuples conflicting with t; and the *vicinity* of t is $v(t) = \{t\} \cup n(t)$.

2.2 Priorities and preferred repairs

For the clarity of presentation we assume a fixed database instance r with a fixed set of functional dependencies F.

To represent the preference information, we use (possibly partial) acyclic orientations of the conflict graph. Orientations allows us to express the preferences at the level of single conflicts and acyclicity ensures unambiguity of the preference.

Definition 2 (Priority). *A* priority *is a binary relation $\succ \subseteq r \times r$ that is defined only on conflicting tuples and is acyclic, i.e. there there does not exist $x \in r$ such that $x \succ^* x$, where $\succ^*$ is the transitive closure of $\succ$. If $x \succ y$ we say that that x* dominates *over y. A priority $\succ$ is* total *if every pair of conflicting tuples $\{x,y\}$ either $x \prec y$ or $y \prec x$.*

From the point of user interface it is often more natural to define the priority as an arbitrary acyclic binary relation on r and then use such a priority relation only on conflicting tuples. Naturally, those approaches are equivalent.

Extending an orientation consists of orienting some conflicting edges that were not oriented before; formally, a priority $\succ'$ is an *extension* of $\succ$ if $\succ' \supseteq \succ$. Note that an extension $\succ'$ is also a priority and therefore $\succ'$ is acyclic and defined only on pairs of conflicting tuples. Also observe that a priority that cannot be extended further is total (i.e. all edges of the conflict graph are oriented).

Preferred repairs In our work we investigate families of preferred repairs: subsets of repairs selected with priorities. For the clarity we adapt the following naming convention. For each investigated way of selecting preferred repairs we use one letter name to refer to it, e.g. $\mathcal{X}$. For a given relation r, a given set of functional dependencies F and a given priority $\succ$, by $\mathcal{X}\text{-}Rep_F^{\succ}(r)$ we denote the selected set of preferred repairs. We drop r, F, and $\succ$ if they are known from the context.

Database cleaning A total priority represent an unambiguous information on how each conflict should be resolved. With Algorithm 1 a total priority is used to construct a "clean" database by iteratively selecting tuples that are not dominated by any other, i.e. tuples selected by the *winnow operator* [5]:

$$\omega_{\succ}(r) = \{t \in r | \neg \exists t' \in r . t' \succ t\}.$$

After selecting a tuple we remove it and its neighbors from further considerations.

Algorithm 1 Cleaning the database

```
1: r' ← ∅
2: while ω_≻(r) ≠ ∅ do
3:     choose any x ∈ ω_≻(r)
4:     r' ← r' ∪ {x}
5:     r ← r \ ({x} ∪ n(x))          ▷ where n(x) – the neighborhood of x.
6: return r'
```

Proposition 1. *For a total priority $\succ$ Algorithm 1 computes a unique repair for any sequence of choices in Step 3.*

2.3 Preferred consistent query answers

We generalize the notion of consistent query answer [1] by considering only preferred repairs when evaluating a query (instead of all repairs). We only study closed first-order

logic queries. We can easily generalize our approach to open queries along the lines of [1, 7]. For a given query Q we say that *true* is an answer to Q in r, if $r \models Q$ in the standard model-theoretic sense.

Definition 3 ($\mathcal{X}$-Consistent query answer). *Given a closed query Q and a family of repairs $\mathcal{X}$-Rep,* true *is the $\mathcal{X}$-*consistent query answer *to a query Q if for every repair $r' \in \mathcal{X}$-Rep we have $r' \models Q$.*

Note that we obtain the original notion of consistent query answer [1] if we consider the whole set of repairs $Rep_F(r)$.

3 Optimal use of the priority

The main purpose of introducing $\mathscr{P}1$–$\mathscr{P}4$ is identification of desired properties of families of preferred repairs. We note that all properties except for $\mathscr{P}4$ do not require any use of the priority to eliminate any repairs. This makes it possible to construct a family of preferred repairs which satisfies $\mathscr{P}1$–$\mathscr{P}4$ which practically makes no use of the given priority.

Example 6. Consider a family of repairs which for a total priority consists of the clean database obtained with Algorithm 1 and otherwise it consists of all repairs. This family of repairs fulfills properties $\mathscr{P}1$–$\mathscr{P}4$.

Thus we investigate a number of increasingly complex notions of repair optimality that ensure effective use of the preference information:

1. r' is a *locally optimal* repair, if no tuple x from r' can be replaced with a tuple y such that $y \succ x$ and the resulting set of tuples is consistent;
2. r' is a *semi-globally optimal* if no nonempty subset X of tuples from r' can be replaced with a tuple y such that $\forall x \in X.y \succ x$ and the resulting set of tuples is consistent;
3. r' is a *globally optimal* if no nonempty subset X of tuples from r can be replaced with a set of tuples Y such that $\forall x \in X.\exists y \in Y.y \succ x$ and the resulting set of tuples is consistent.

We note that global optimality implies semi-global optimality which in turn implies local optimality. Intuitively, global optimality makes an aggressive use of priorities to select repairs, while local optimality does so in a less aggressive manner.

3.1 Locally optimal repairs

With $\mathcal{L}$-*Rep* we denote the set of all locally optimal repairs. The following example illustrates that the notion of local optimality allows to effectively use priorities to handle relations with one key.

Example 7. Consider the relational schema $R(A,B)$ with a key dependency $F = \{A \rightarrow B\}$ and take an instance $r = \{t_a = (1,1), t_b = (1,2), t_c = (1,3)\}$ with the priority $\succ = \{(t_a, t_c), (t_a, t_b)\}$. Figure 2 contains the corresponding conflict graph and its orientation. The repairs are $Rep_F(r) = \{r_1 = \{t_a\}, r_2 = \{t_b\}, r_3 = \{t_c\}\}$. Only r_1 is locally preferred.

Proposition 2. *$\mathcal{L}$-Rep satisfies properties $\mathscr{P}1$–$\mathscr{P}3$.*

As it's shown on the following example, locally optimal repairs do not satisfy $\mathscr{P}4$.

Example 8. Consider the relational schema $R(A,B,C)$ with a functional dependency $A \to B$ and take an instance $r = \{t_a = (1,1,1), t_b = (1,1,2), t_c = (1,2,3)\}$ with the total priority $\succ = \{(t_c,t_a),(t_c,t_b)\}$. The corresponding conflict graph can be found in Figure 3. The set of repairs consists of two repairs $Rep_F(r) = \{r_1 = \{t_a,t_b\}, r_2 = \{t_c\}\}$. All the repairs are locally optimal.

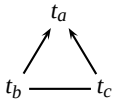

Fig. 2. Use of $\mathcal{L}$-*Rep*.

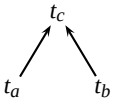

Fig. 3. Non-categoricity of $\mathcal{L}$-*Rep*.

3.2 Semi-globally optimal repairs

In Example 8, we note that even though the priority suggest rejecting r_1 from consideration, the notion of local optimality is too weak to do so. The main reason is the existence of violations of functional dependency with duplicates (t_a and t_b which are not conflicting, but both of them conflict with t_c). The notion of semi-global optimality, however, effectively applies the priority in the situations of violations of one non-key functional dependency: the repair r_1 is not semi-globally optimal and r_2 is. We denote the family of all semi-globally optimal repairs by $\mathcal{S}$-*Rep* and we note that $\mathcal{S}$-*Rep* is as effective in enforcing priorities as $\mathcal{L}$-*Rep*.

Proposition 3. *$\mathcal{S}$-Rep satisfies properties $\mathscr{P}1$–$\mathscr{P}3$. Moreover $\mathcal{S}$-Rep $\subseteq$ $\mathcal{L}$-Rep and for one key dependency $\mathcal{L}$-Rep coincides with SRep.*

Also this family of preferred repairs does not satisfy $\mathscr{P}4$.

Example 9. Consider the schema $R(A,B,C,D)$ with two functional dependencies $F = \{A \to B, C \to D\}$ and suppose we have a database: $r = \{t_a = (1,1,0,0), t_b = (1,2,1,1), t_c = (2,1,1,2), t_d = (2,2,2,1), t_e = (0,0,2,2)\}$ with a total priority $\succ = \{(t_a,t_b),(t_b,t_c),(t_c,t_d),(t_d,t_e)\}$. The conflict graph is presented on Figure 4. The set of repairs is $Rep_F(r) = \{r_1 = \{t_a,t_c,t_e\}, r_2 = \{t_b,t_d\}\}$. This is also the set of semi-globally optimal repairs.

3.3 Globally optimal repairs

Situations similar to Example 9 are encountered in the setting where a relation has more than one functional dependency which are violated by mutual conflicts (a tuple

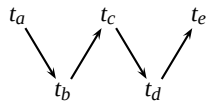

Fig. 4. Non-categoricity of $\mathcal{S}$-*Rep*.

may be involved in conflicts generated by more than one functional dependency) and the user provides priority only for some of the violated functional dependencies. In those settings the notion of global optimality follows our intuitions: r_2 is not globally optimal and r_1 is.

Let $\mathcal{G}$-*Rep* be the family of globally optimal repairs. This family satisfies $\mathscr{P}4$.

Proposition 4. *$\mathcal{G}$-Rep satisfies properties $\mathscr{P}1$–$\mathscr{P}4$. Moreover $\mathcal{G}$-Rep $\subseteq$ $\mathcal{S}$-Rep and for one functional dependency $\mathcal{G}$-Rep coincides with $\mathcal{S}$-Rep.*

Globally optimal repairs can be characterized in an alternative way.

Proposition 5. *For a given priority $\succ$ and two repairs, we say that r_2 is* preferred *over r_1, denoted $r_1 \ll r_2$, if*

$$\forall x \in r_1 \setminus r_2.\ \exists y \in r_2 \setminus r_1.\ y \succ x.$$

A repair r' is globally optimal if and only if it's $\ll$-maximal (there is no repair r'' such that $r' \ll r''$).

This particular "lifting" of a preference on objects to a preference on sets of objects can be found in other contexts. For example, a similar definition is used for a preference among different models of a logic program [21], or for a preference among different worlds [15].

3.4 Importance of monotonicity

In Section 4 we study the computational implications of using priorities to handle inconsistent databases. Restricting our choice when constructing a family of repairs to one of the optimal classes of repairs, still does not prevent us to construct trivial families of optimal repairs.

Example 10. For any instance r, any set of functional dependencies F, and any priority $\succ$ for r and F, choose one extension $\succ'$ that is total for r and F. Now, consider the family $\mathcal{T}$-*Rep* which for an instance r, a set of functional dependencies F, and a priority $\succ$ consists of the only repair constructed with Algorithm 1 for r, F, and the corresponding total priority $\succ'$.

We can easily show that the repair obtained with Algorithm 1 for a total priority is a globally optimal repair. Therefore $\mathcal{T}$-*Rep* is a family of globally optimal repairs that satisfies $\mathscr{P}1$, $\mathscr{P}3$, and $\mathscr{P}4$.

We conclude here that while optimality enforces use of priorities to eliminate repairs from considerations, the monotonicity prevents from groundless elimination. Hence, in the context of preferred consistent query answers it is natural to restrict our attention to families of optimal repairs which satisfy the essential properties $\mathscr{P}1$ and $\mathscr{P}2$.

3.5 Common optimal repairs

Now, we investigate the question whether there are repairs common for any family of optimal repairs that satisfies the properties $\mathscr{P}1$ and $\mathscr{P}2$, i.e. for a given instance r, a given set of functional dependencies, and a given priority $\succ$, is there a repair r' which is in $\mathcal{X}\text{-}Rep_F^{\succ}(r)$ for any family $\mathcal{X}\text{-}Rep$ of optimal repairs satisfying $\mathscr{P}1$ and $\mathscr{P}2$. The answer is negative for families of semi-globally (and thus also locally) optimal repairs. For instance we can construct two families of semi-globally optimal repairs that define the same set of preferred repairs as $\mathcal{S}\text{-}Rep$ except that for the setting in Example 9 one returns only r_1 while the other only r_2. Surprisingly, the situation is different for families of globally optimal repairs.

Theorem 1. *For every instance r, every set of functional dependencies F, and any every priority $\succ$, there exists a repair r' such that $r' \in \mathcal{X}\text{-}Rep_F^{\succ}(r)$ for any family $\mathcal{X}\text{-}Rep$ of globally optimal repairs that satisfies $\mathscr{P}1$ and $\mathscr{P}2$.*

We define a new family of $\mathcal{C}\text{-}Rep$ which selects only *common* repairs of all families of globally optimal repairs satisfying the essential properties $\mathscr{P}1$ and $\mathscr{P}2$. $\mathcal{C}\text{-}Rep$ is another family of preferred repairs that satisfies all properties.

Proposition 6. *$\mathcal{C}\text{-}Rep$ satisfies properties $\mathscr{P}1$ and $\mathscr{P}4$ and $\mathcal{C}\text{-}Rep \subseteq \mathcal{G}\text{-}Rep$*

Interestingly the family of common repairs has an alternative *procedural* characterization.

Proposition 7. *For a given instance r, a given set of functional dependencies F, and a given priority $\prec$, the set $\mathcal{C}\text{-}Rep_F^{\prec}(r)$ consists of all results of Algorithm 1 for any sequence of choices in Step 3.*

We also note that under some conditions, the properties $\mathscr{P}1$ and $\mathscr{P}2$ specify exactly one family of globally optimal repairs.

Theorem 2. *$\mathcal{C}\text{-}Rep$ and $\mathcal{G}\text{-}Rep$ coincide for priorities that cannot be extended to a cyclic orientation of the conflict graph.*

4 Computational properties

In this section we study the computational implications of using priorities to handle inconsistent databases. Because of space restriction we skip the proofs (most of them can be found in or easily based on reductions presented in [8]).

4.1 Data complexity

In our paper we use the notion of *data complexity* [22] which captures the complexity of a problem as a function of the number of tuples in the database. The input consists of the relation instance and the priority relation, while the database schema, the integrity constraints, and the query are assumed to be fixed. For a family $\mathcal{X}\text{-}Rep$ of preferred repairs we study two fundamental computational problems:

(i) $\mathcal{X}$*-repair checking* – determining if a database is a preferred repair of a given database i.e., the complexity of the following set

$$\mathscr{B}_F^{\mathcal{X}} = \{(r, \succ, r') : r' \in \mathcal{X}\text{-}Rep_F^{\succ}(r)\}.$$

(ii) $\mathcal{X}$*-consistent query answers* – checking if *true* is an answer to a given query in every preferred repair i.e., the complexity of the following set

$$\mathscr{D}_{F,Q}^{\mathcal{X}} = \{(r, \succ) : \forall r' \in \mathcal{X}\text{-}Rep_F^{\succ}(r).r' \models Q\}.$$

4.2 Complexity results

First we state that computing preferred consistent query answer with any family of semi-globally (and thus also globally) optimal repairs that satisfies $\mathscr{P}1$ and $\mathscr{P}2$ leads to intractability.

Theorem 3. *For any family $\mathcal{X}$-Rep of semi-globally optimal repairs that satisfies $\mathscr{P}1$ and $\mathscr{P}2$, there exists a set of two functional dependencies F and a quantifier-free ground query Q (consisting of one atom) to which computing the $\mathcal{X}$-consistent answer is co-NP-hard.*

It's an open question whether a similar statement holds for families of locally optimal repairs. We note that computing preferred consistent query answers is co-NP-hard if we consider a slightly restricted locally optimal repairs: locally optimal repairs for which there doesn't exists a pair of tuples x_1, x_2 which can be replaced with a tuple y such that $y \succ x_1$ and $y \succ x_2$ and the resulting set of tuples is consistent. Therefore we state the following conjecture.

Conjecture 1. For any family $\mathcal{X}$-*Rep* of preferred repairs satisfying $\mathscr{P}1$, $\mathscr{P}2$, and global local optimality computing $\mathcal{X}$-consistent answers is co-NP-hard.

Another argument for this conjecture is the intractability of computing $\mathcal{L}$-consistent query answers.

Theorem 4. *The $\mathcal{L}$-repair checking is in PTIME . There exists a set of two functional dependencies and a quantifier-free query (consisting of one atom only) for which computing $\mathcal{L}$-consistent answers co-NP-complete.*

To find if a repair r' is semi-globally optimal we seek a tuple $yr \setminus r'$ whose all neighbors in r' are dominated by y. Such a tuple exists if and only if r' is not semi-globally optimal. The tractability of $\mathcal{S}$-checking implies that computing $\mathcal{S}$-consistent answers is in co-NP: the nondeterministic machine uses a polynomial in the size of r number of nondeterministic steps to construct a repair r', checks if r' is semi-globally optimal; the machine finds the answer to the query in r' (if r' is not semi-optimal then the machine halts with the answer 'yes'). With Theorem 3 we obtain:

Corollary 1. *The $\mathcal{S}$-repair checking is in PTIME and computing $\mathcal{S}$-consistent answers is co-NP-complete.*

Checking if a repair is globally optimal requires, however, an essential use of nondeterminism. This also promotes computing preferred consistent query answers to a higher level of the polynomial hierarchy.

Theorem 5. *There exists a set of five functional dependencies for which the $\mathcal{G}$-repair checking is co-NP-complete. There exists a set of four functional dependencies and a quantifier-free query (consisting of one atom only) for which computing $\mathcal{G}$-consistent answers is Π_2^p-complete.*

The procedural nature of common repairs makes it possible to check if a repair r' belongs to $\mathcal{C}$-$Rep_F^{\succ}(r)$ with a simulation of Algorithm 1 with the choices in Step 3 restricted to $\omega_{\succ}(r) \cap r'$. Naturally this process can be performed in polynomial time. Again using Theorem 3 we get:

Corollary 2. *The $\mathcal{C}$-repair checking is in PTIME and computing $\mathcal{C}$-consistent answers is co-NP-complete.*

5 Related work

We limit our discussion to work on using priorities to maintain consistency and facilitate resolution of conflicts.

The first to notice the importance of priorities in information systems is [9]. The authors study there the problem of updates of databases containing propositional sentences. The priority is expressed by storing a natural number with each clause. If during an update (inserting or deleting a sentence) the inconsistency arises, then the priorities are used in a fashion similar to $\mathcal{G}$-repairs to select minimally different repairs. We note, however, that the chosen representation of priorities imposes a significant restriction on the class of considered priorities. In particular it assumes transitivity of the priority on conflicting facts i.e. if facts *a*, *b*, and *c* are pair-wise conflicting and *a* has a higher priority than *b* and *b* has a higher priority than *c*, then the priority of *a* is higher than *c*. This assumption cannot be always fulfilled in the context of inconsistent databases. For example the conflicts between *a* and *b*, and between *b* and *c* may be caused by violation of one integrity constraints while the conflict between *a* and *c* is introduced by a different constraint. While the user may supply us with a rule assigning priorities to conflicts created by the first integrity constraint, the user may not wish to put any priorities on any conflicts created by the other constraint.

A similar representation of priorities used to resolve inconsistency in first-order theories is studied in [4], where the inconsistent set of clauses is stratified (again the lowest strata has the highest priority). Then preferred maximal consistent subtheories are constructed in a manner analogous to $\mathcal{C}$-repairs. Furthermore, this approach is generalized to priorities being a partial orders, by considering all extensions to weak orders. Again, however, this approach assumes transitivity of priority on conflicts, which as we explained previously may be considered a significant restriction.

In [19] priorities are studied to facilitate the process of *belief revision*. A belief state is represented as an ordered list of propositional formulae and the revision operation simply adds the given sentence at the end of the given belief state. This representation

of belief state allows to keep track of revision history, which is later used to impose a preference order on the possible interpretations of the belief state. Only maximally preferred interpretations are used when defining the entailment relation.

In the context of logic programs, priorities among rules can be used to handle inconsistent logic programs (where rules imply contradictory facts). More preferred rules are satisfied, possibly at the cost of violating less important ones. In a manner analogous to $\ll$, [21] lifts a total order on rules to a preference on (extended) answers sets. When computing answers only maximally preferred answers sets are considered.

[20] investigate disjunctive logic programs with priorities on facts. A a transitive and reflexive closure of user supplied priorities on facts is used to define a relation of preference on models of the program. The definition of preference on models of the disjunctive program is essentially different from the characterization of globally optimal repairs in Proposition 5. The answer to a program in the extended framework consists of all maximally preferred answer sets. The main shortcoming of using this framework is it's computational infeasibility (which is specific to decision problems involving general disjunctive programs): computing answers to ground queries to disjunctive prioritized logic programs under cautious (brave) semantics is Π_3^p-complete (resp. Σ_3^p-complete).

A simpler approach to the problem of inconsistent logic programs is presented in [14]. There conflicting facts are removed from the model unless the priority specifies how to resolve the conflict. Because only programs without disjunction are considered, this approach always returns exactly one model of the input program. Constructing preferred repairs in a corresponding fashion (by removing all conflicts unless the priority indicates a resolution) would similarly return exactly one database instance (fulfillment of $\mathscr{P}1$ and $\mathscr{P}4$). However, if the priority does not specify how to resolve every conflict, the returned instance is not a maximal set of tuples and therefore it is not a repair. Such an approach leads to a loss of (disjunctive) information and do not satisfy $\mathscr{P}2$ and $\mathscr{P}3$.

[10] proposes a framework of *conditioned active integrity constraints*, which allows the user to specify the way some of the conflicts created with the constraint can be resolved. This framework satisfies properties $\mathscr{P}1$ and $\mathscr{P}3$ and doesn't satisfy $\mathscr{P}2$ and $\mathscr{P}4$. [10] also describes how to translate conditioned active integrity constraints into a prioritized logic program [20], whose preferred models correspond to maximally preferred repairs. We note that the framework of prioritized logic programming is computationally more powerful (computing answers under the brave semantics is Σ_3^p-complete) than required by the problem of finding if an atom is present in any repair (Σ_2^p-complete). It is yet to be seen if less powerful programming environment (like general disjunctive logic programs) can be used to compute preferred answers.

[17] uses ranking functions on tuples to resolve conflicts by taking only the tuple with highest rank and removing others. This approach constructs a unique repair under the assumption that no two different tuples are of equal rank (satisfaction of $\mathscr{P}4$). If this assumption is not satisfied and the tuples contain numeric values, a new value, called the fusion, can be calculated from the conflicting tuples (then, however, the constructed instance is not a repair in the sense of Definition 1 which means a possible loss of information).

A different approach based on ranking is studied in [13]. The authors consider polynomial functions that are used to rank repairs. When computing preferred consistent

query answers, only repairs with the highest rank are considered. The property $\mathscr{P}3$ is trivially satisfied, but because this form of preference information does not have natural notions of extensions and maximality, it is hard to discuss postulates $\mathscr{P}2$ and $\mathscr{P}4$. Also, the preference among repairs in this method is not based on the way in which the conflicts are resolved.

An approach where the user has a certain degree of control over the way the conflicts are resolved is presented in [12]. Using repair constraints the user can restrict considered repairs to those where tuples from one relation have been removed only if similar tuples have been removed from some other relation. This approach satisfies $\mathscr{P}2$ but not $\mathscr{P}1$. A method of weakening the repair constraints is propose to get $\mathscr{P}1$, however this comes at the price of losing $\mathscr{P}2$.

6 Conclusions and future work

In this paper we proposed a general framework of preferred repairs and preferred consistent query answer. We also proposed a set of desired properties a family of preferred repairs should satisfy. We presented 4 families of preferred repairs: $\mathcal{L}$-*Rep*, $\mathcal{S}$-*Rep*, $\mathcal{G}$-*Rep*, and $\mathcal{C}$-*Rep*. Figure 5 summarizes the computational complexity results; its first row is taken from [6].

	Repair Check	Consistent Answers to		Possible Applications
		$\{\forall,\exists\}$-free queries	conjunctive queries	
Rep	PTIME	PTIME	co-NP-complete	no priorities given
$\mathcal{L}$-*Rep*	PTIME	co-NP-complete		key (no duplicates)
$\mathcal{S}$-*Rep*	PTIME	co-NP-complete		one FD (duplicates)
$\mathcal{G}$-*Rep*	co-NP-complete	Π_p^2-complete		many FDs with mutual conflicts
$\mathcal{C}$-*Rep*	PTIME	co-NP-complete		

Fig. 5. Summary of complexity results.

We envision several directions for further work. Along the lines of [2], the computational complexity results could be further studied, by assuming the conformance of functional dependencies with BCNF.

Extending our approach to cyclic priorities is an interesting and challenging issue. Including priorities in similar frameworks [12] of preferences leads to loosing the monotonicity. A modified, conditional, version of monotonicity may be necessary to capture non-trivial families of repairs.

The last is a generalization of our framework to a broader class of constraints. Conflict graphs can be generalized to hypergraphs [6], which allow to handle broader class of denial constraints. Then, more than two tuples can be involved in a single conflict and the current notion of priority does not have a clear meaning.

References

1. M. Arenas, L. Bertossi, and J. Chomicki. Consistent Query Answers in Inconsistent Databases. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 68–79, 1999.
2. M. Arenas, L. Bertossi, J. Chomicki, X. He, V. Raghavan, and J. Spinrad. Scalar Aggregation in Inconsistent Databases. *Theoretical Computer Science*, 296(3):405–434, 2003.
3. P. Bohannon, M. Flaster, W. Fan, and R. Rastogi. A Cost-Based Model and Effective Heuristic for Repairing Constraints by Value Modification. In *ACM SIGMOD International Conference on Management of Data*, 2005.
4. G. Brewka. Preferred Subtheories: An Extended Logical Framework for Default Reasoning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 1989.
5. J. Chomicki. Preference Formulas in Relational Queries. *ACM Transactions on Database Systems*, 28(4):427–466, December 2003.
6. J. Chomicki and J. Marcinkowski. Minimal-Change Integrity Maintenance Using Tuple Deletions. *Information and Computation*, pages 90–121, 2005.
7. J. Chomicki, J. Marcinkowski, and S. Staworko. Computing Consistent Query Answers Using Conflict Hypergraphs. In *International Conference on Information and Knowledge Management (CIKM)*, pages 417–426, November 2004.
8. J. Chomicki, J. Marcinkowski, and S. Staworko. Priority-Based Conflict Resolution in Inconsistent Relational Databases. Technical Report cs.DB/0506063, arXiv.org e-Print archive, June 2004.
9. R. Fagin, J. D. Ullman, and M. Y. Vardi. On the Semantics of Updates in Databases. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 352–356, 1983.
10. S. Flesca, S. Greco, and E. Zumpano. Active Integrity Constraints. In *ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP)*, pages 98–107, 2004.
11. A. Fuxman, E. Fazli, and R. J. Miller. ConQuer: Efficient Management of Inconsistent Databases. In *ACM SIGMOD International Conference on Management of Data*, 2005.
12. G. Greco and D. Lembo. Data Integration with Preferences Among Sources. In *International Conference on Conceptual Modeling (ER)*, pages 231–244, November 2004.
13. S. Greco, C. Sirangelo, I. Trubitsyna, and E. Zumpano. Feasibility Conditions and Preference Criteria in Quering and Repairing Inconsistent Databases. In *International Conference on Database and Expert Systems Applications (DEXA)*, pages 44–55, 2004.
14. B. N. Grosof. Prioritized Conflict Handling for Logic Programs. In *International Logic Programming Symposium*, pages 197–211, 1997.
15. J. Y. Halpern. Defining Relative Likehood in Partially-Ordered Preferential Structures. *Journal of Artificial Intelligence Research*, 1997.
16. D. B. Lomet. Letter from the Editor-in-Chief. *IEEE Data Eng. Bull.*, 23(4), 2000.
17. A. Motro, P. Anokhin, and A. C. Acar. Utility-based Resolution of Data Inconsistencies. In *International Workshop on Information Quality in Information Systems (IQIS)*, 2004.
18. E. Rahm and H. H. Do. Data Cleaning: Problems and Current Approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.
19. M. Ryan. Belief Revision and Ordered Theory Presentations. In *Logic, Action, and Information*, pages 129–151, 1996.
20. C. Sakama and K. Inoue. Prioritized logic programming and its application to commonsense reasoning. *Artificial Intelligence*, 123:185–222, 2000.
21. D. Van Nieuwenborgh and D. Vermeir. Preferred Answer Sets for Ordered Logic Programs. In *European Conference on Logics for Artificial Intelligence (JELIA)*, pages 432–443. Springer-Verlag, LNCS 2424, 2002.
22. M. Y. Vardi. The Complexity of Relational Query Languages. In *ACM Symposium on Theory of Computing (STOC)*, pages 137–146, 1982.
23. P. Vassiliadis, Z. Vagena, S. Skiadopoulos, and N. Karayannidis. ARKTOS: A Tool For Data Cleaning and Transformation in Data Warehouse Environments. *IEEE Data Eng. Bull.*, 23(4):42–47, 2000.