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

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

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

together with with two key dependencies:

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

In an instance of this schema a tuple (x, y, z, v) denotes the fact that x manages the department y, receives a salary z, and is 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 may result from 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* [18]. 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 [20]. 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 [24]: 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),
- the source of the information of the tuple (a user can consider the data from one source more reliable than the data from the other).

Applying 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 does not allow to utilize the incomplete information often present in inconsistencies.

The framework of *repairs* and *consistent query answers* [1] incorporates an alternative approach to deal with inconsistent databases, geared toward utilizing incomplete information. A *repair* is a consistent database minimally different from the given one, and a *consistent answer* to a query is the answer present in *every* repair. This approach does not 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 [5, 13], for the surveys of the area see [4, 3, 8]).

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 attempt to first clean the database and then use the consistent query answers approach. However, this is a radical approach: removing tuples may lead to information loss. Instead, we propose to use additional user input in the form of preferences to select only the *preferred* repairs. Query answers present in every preferred repair are called *preferred consistent query answers*.

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 r' 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, however, 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 as the only preferred repairs, then true is the preferred consistent answer to Q_2.

In this paper we extend the framework of consistent query answers with additional input consisting of preference information Φ. We use Φ to define the set of preferred repairs Rep^{Φ}. When we compute preferred 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. Φ is *total* if it cannot be extended further. We identify the following desirable properties of families of preferred repairs:

$\mathcal{P}1$ **Non-emptiness**

$$Rep^{\Phi} \neq \varnothing.$$

$\mathcal{P}2$ **Monotonicity**: extending preferences can only narrow the set of preferred repairs

$$\Phi \subseteq \Psi \Rightarrow Rep^{\Psi} \subseteq Rep^{\Phi}.$$

$\mathcal{P}3$ **Non-discrimination**: if no preference information is given, then no repair is removed from consideration

$$Rep^{\varnothing} = Rep.$$

$\mathcal{P}4$ **Categoricity**: given maximal preference information we obtain exactly one repair

$$\Phi \text{ is total} \Rightarrow |Rep^{\Phi}| = 1.$$

We also note that properties $\mathcal{P}2$ and $\mathcal{P}3$ imply an important property

$\mathcal{P}5$ **Conservativeness**: preferred repairs are a subset of all repairs

$$Rep^{\Phi} \subseteq Rep.$$

In Section 3 we also study various 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 [9] 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 (FD). We use $X \to Y$ to denote the following constraint:

$$\forall t_1, t_2 \in R. \bigwedge_{A \in X} t_1.A = t_2.A \Rightarrow \bigwedge_{B \in Y} t_1.B = t_2.B \tag{1}$$

We identify conflicts as follows: tuples t_1 and t_2 are *mutually conflicting* in the database r w.r.t. the set of functional dependencies F if t_1 and t_2 belong to r and there exists a functional dependency of the form (1) in F such that $t_1.A = t_2.A$ 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 w.r.t. F. Otherwise, the database is *consistent*.

In the framework of [1] when repairing a database two operations are considered: adding or removing a tuple. In the presence of functional dependencies adding new tuples cannot remove conflicts and hence only repairs obtained by deleting tuples have to be considered.

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 $Rep(r, F)$ 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 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* $G(r, E)$ is a graph whose vertices are the tuples of r and two tuples are adjacent if only if they are mutually conflicting w.r.t. F. Conflict graphs are *compact representations of repairs* because the set of all repairs is equal to the set of all maximal independent 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.

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 $v(t) = \{t\} \cup n(t)$.

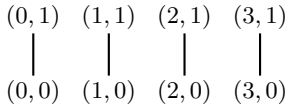

Fig. 1. A conflict graph.

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 acyclic orientations of some (not necessarily all) edges of the conflict graph. Orientations allow us to express the preferences at the level of single conflicts and acyclicity ensures unambiguity of the preference.

Definition 2 (Priority). *A* priority *(in r w.r.t. F) is a binary relation $\succ \subseteq r \times r$ such that $\succ$ is acyclic and $x \succ y$ implies that x and y are mutually conflicting (in r w.r.t. F). If $x \succ y$ we say that that x* dominates *y. A priority $\succ$ is* total *if for every pair x, y of mutually conflicting tuples (in r w.r.t. F) either $x \succ y$ or $y \succ x$.*

From the point of the user interface it is often more natural to define the priority as some acyclic binary relation on r and then consider the 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 mutually conflicting tuples. Also observe that a priority cannot be extended further if and only if it is total.

Conflict resolution A total priority provides an unambiguous information on how each conflict should be resolved. The Algorithm $\mathcal{CR}$ uses a total priority to construct a consistent database by iteratively selecting tuples that are not dominated by any other tuples, i.e. tuples selected by the *winnow operator* [7]:

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

After selecting a tuple t, t is removed together with its neighbors from further consideration.

Proposition 1. *Given a total priority $\succ$, the Algorithm $\mathcal{CR}$ computes a unique repair for any sequence of choices in Step 3.*

Algorithm $\mathcal{CR}$: Conflict Resolution

```
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'
```

Preferred repairs In our work we investigate families of preferred repairs. Formally, a *family of preferred repairs* is a function X-*Rep* defined on triplets $(r, F, \succ)$, where $\succ$ is a priority in r w.r.t. a set of FDs F, such that X-$Rep(r, F, \succ)$ is a set of repairs. We say that a family X_1-*Rep subsumes* a family X_2-*Rep*, denoted X_1-$Rep \sqsubseteq X_2$-*Rep*, if for every $(r, F, \succ)$ we have that X_1-$Rep(r, F, \succ) \subseteq X_2$-$Rep(r, F, \succ)$.

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, 9]. 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 (X-Consistent query answer). *Given a database r, a set of FDs F, a closed query Q, a priority $\succ$, and a family of repairs X-Rep,* true *(*false*) is the X-*consistent query answer *to a query Q in r w.r.t. F and $\succ$ if for every repair $r' \in X$-$Rep(r, F, \succ)$ we have $r' \models Q$ (resp. $r' \not\models Q$).*

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

3 Priority-based repairing

The main purpose of introducing $\mathcal{P}1$–$\mathcal{P}4$ is the identification of the desired properties of families of preferred repairs. We note that all properties except for $\mathcal{P}4$ do not require any use of the priority itself to eliminate repairs. This makes it possible to construct a family of preferred repairs which satisfies $\mathcal{P}1$–$\mathcal{P}4$ but 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 $\mathcal{CR}$ and for non-total priorities it consists of all repairs. This family of repairs fulfills properties $\mathcal{P}1$–$\mathcal{P}4$.

Thus we investigate a number of increasingly complex notions of repair optimality that ensure an 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 *Pareto 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 Pareto 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

By *L-Rep* we denote the family selecting 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 dependency.

Example 7. Consider the relational schema $R(A, B)$ with a key dependency $F = \{A \to B\}$ 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 $Rep(r, F) = \{r_1 = \{t_a\}, r_2 = \{t_b\}, r_3 = \{t_c\}\}$. Only r_1 is locally preferred.

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

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

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 $Rep(r, F) = \{r_1 = \{t_a, t_b\}, r_2 = \{t_c\}\}$. All repairs are locally optimal.

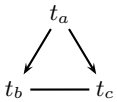

Fig. 2. Use of *L-Rep*.

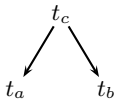

Fig. 3. Non-categoricity of *L-Rep*.

3.2 Pareto optimal repairs

In Example 8, we note that even though the priority suggests 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 Pareto optimality, on the other hand, effectively applies the priority in the situations of violations of one non-key functional dependency: the repair r_1 is not Pareto optimal and r_2 is. By *P-Rep* we denote the family selecting all Pareto optimal repairs. We note that *P-Rep* is as effective in enforcing priorities as *L-Rep*.

Proposition 3. *P-Rep satisfies properties $\mathcal{P}1$–$\mathcal{P}4$. Moreover P-Rep $\sqsubseteq$ L-Rep and for one key dependency L-Rep coincides with P-Rep.*

3.3 Globally optimal repairs

Pareto optimality selects repairs whose compliance with the priority cannot be improved by exchanging a set of tuples with a dominating tuple. The following example presents a situation where one may decide to use global optimality to provide a finer selection of repairs.

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: $r = \{t_a = (1, 1, 0, 0), t_b = (1, 2, 0, 0), t_c = (1, 1, 1, 1), t_d = (1, 2, 2, 1), t_e = (0, 0, 2, 2)\}$ with a 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 $Rep(r, F) = \{r_1 = \{t_b, t_e\}, r_2 = \{t_b, t_d\}, r_3 = \{t_a, t_c, t_e\}\}$. Repairs r_2 and r_3 are Pareto optimal. However, only the repair r_3 is globally optimal.

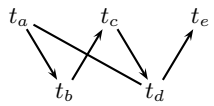

Fig. 4. Pareto vs. global optimality.

Let *G-Rep* be the family selecting all globally optimal repairs.

Proposition 4. *G-Rep satisfies properties $\mathcal{P}1$–$\mathcal{P}4$. Moreover G-Rep $\sqsubseteq$ P-Rep and for one functional dependency G-Rep coincides with P-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 is $\ll$-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 [22], or for a preference among different worlds [17].

3.4 Common optimal repairs

Now, we investigate the question whether there are repairs common for any family of optimal repairs that satisfies the properties $\mathcal{P}1$ and $\mathcal{P}2$, i.e. given any $(r, F, \succ)$ is there a repair r' which is in $X\text{-}Rep(r, F, \succ)$ for any family $X\text{-}Rep$ of optimal repairs satisfying $\mathcal{P}1$ and $\mathcal{P}2$? The answer is negative for families of locally optimal repairs. For instance we can construct two families of locally optimal repairs that define the same set of preferred repairs as $L\text{-}Rep$ except that for the setting in Example 8 one returns only r_1 while the other only r_2. Surprisingly, the situation is different for families of Pareto (and thus also globally) optimal repairs.

Theorem 1. *For every instance r, every set of functional dependencies F, and every priority $\succ$ in r w.r.t. F, there exists a repair r' such that $r' \in X\text{-}Rep(r, F, \succ)$ for any family $X\text{-}Rep$ of Pareto optimal repairs that satisfies $\mathcal{P}1$ and $\mathcal{P}2$.*

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

Proposition 6. *$C\text{-}Rep$ is a family of globally optimal repairs, i.e. $C\text{-}Rep \sqsubseteq G\text{-}Rep$, and $C\text{-}Rep$ satisfies properties $\mathcal{P}1$-$\mathcal{P}4$.*

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 $\succ$, the set $C\text{-}Rep(r, F, \succ)$ consists of all results of Algorithm $\mathcal{CR}$ for any sequence of choices in Step 3.*

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

Theorem 2. *$C\text{-}Rep$ and $G\text{-}Rep$ 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 (they can be found in [10] or derived using the techniques presented there).

4.1 Data complexity

In our paper we use the notion of *data complexity* [23] 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 *X-Rep* of preferred repairs we study two fundamental computational problems:

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

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

(*ii*) *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

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

4.2 Negative results

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

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

Proof. We reduce computing X-consistent query answers to the complement of SAT. Take then any CNF formula $\varphi = c_1 \wedge \ldots \wedge c_k$ over variables $x_1, \ldots, x_n$ and let $c_j = l_{j,1} \vee \ldots \vee l_{j,m_j}$. We assume that there are no repetitions of literals in a clause (i.e., $l_{j,k_1} \neq l_{j,k_2}$). We construct a relation instance r_φ over the schema $R(A_1, B_1, A_2, B_2)$ in the presence of two functional dependencies $F = \{A_1 \rightarrow B_1, A_2 \rightarrow B_2\}$. The instance r_φ consists of the following tuples:

- $w_i = (i, 1, i, 1)$ corresponds to the positive valuation of the variable x_i (for every $i = 1, \ldots, n$),
- $\bar{w}_i = (i, -1, -i, 1)$ corresponds to the negative valuation of the variable x_i (for every $i = 1, \ldots, n$),
- $v_i^j = (n + j, 1, -i, 0)$ corresponds to the use of the literal x_i in the clause c_j,
- $v_i^j = (n + j, 1, i, 0)$ corresponds to the use of the literal $\neg x_i$ in the clause c_j,
- $d_j = (n + j, 1, 0, 1)$ corresponds to the clause c_j,
- $b = (0, 0, 0, 0)$ corresponds to the formula φ.

The constructed priority $\succ_\varphi$ is the minimal priority on r_φ (w.r.t. F) such that:

$$\bar{w}_i \succ_\varphi v_i^j, \qquad v_i^j \succ_\varphi d_j, \qquad d_j \succ_\varphi b,$$
$$w_i \succ_\varphi \bar{v}_i^j, \qquad \bar{v}_i^j \succ_\varphi d_j.$$

The query we consider is $Q = \neg R(b)$. On Figure 5 we can find a conflict graph of an instance received from reduction of a formula $\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_3 \vee \neg x_4 \vee x_5 \vee \neg x_6)$.

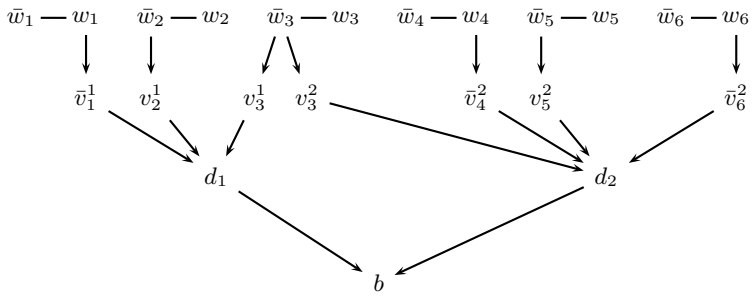

Fig. 5. Conflict graph for $\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_3 \vee \neg x_4 \vee x_5 \vee \neg x_6)$ with $\succ_\varphi$.

Now we show that

$$(r_\varphi, \succ_\varphi) \in \mathcal{D}_{F,Q} \iff \forall r' \in X\text{-}Rep(r_\varphi, F, \succ_\varphi).b \notin r' \iff \varphi \notin SAT.$$

$\Rightarrow$ Suppose there exists a valuation V such that $V \models \varphi$. Consider then the following instance

$$r' = \{w_i | V(x_i) = true\} \cup \{\bar{w}_i | V(x_i) = false\} \cup$$
$$\{v_i^j | V(x_i) = true\} \cup \{\bar{v}_i^j | V(x_i) = false\} \cup \{b\}.$$

We claim that $r \in X\text{-}Rep(r_\varphi, F, \succ_\varphi)$. To prove this consider the following priority $\succ' = \succ_\varphi \cup \{(v_i, \bar{v}_i) | V(x_i) = true\} \cup \{(\bar{v}_i, v_i) | V(x_i) = false\}$. By $\mathcal{P}1$ we have that $X\text{-}Rep(r_\varphi, F, \succ')$ is non-empty. We can prove that r' belongs to $X\text{-}Rep(r_\varphi, F, \succ')$ using the fact that $X\text{-}Rep$ is a family of Pareto optimal repairs (for brevity we skip this step). And from $\mathcal{P}2$ we get that $X\text{-}Rep(r_\varphi, F, \succ') \subseteq X\text{-}Rep(r_\varphi, F, \succ_\varphi)$ and hence r' belongs to $X\text{-}Rep(r_\varphi, F, \succ')$. Since $b \in r$ this contradicts $(r_\varphi, \succ_\varphi) \in \mathcal{D}_{F,Q}$.

$\Leftarrow$ For brevity we just sketch this part of the proof. Suppose there exists a repair $r' \in X\text{-}Rep(r_\varphi, F, \succ_\varphi)$ such that $b \in r'$. We can prove that the following

valuation

$$V(x_i) = \begin{cases} true & \text{if } v_i \in r', \\ false & \text{if } \bar{v}_i \in r', \\ true & \text{otherwise} \end{cases}$$

satisfies φ which is a contradiction.

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

Another argument for this conjecture is the intractability of computing L-consistent query answers (the proof of co-NP-hardness is omitted here, however, it uses the reduction from the proof of Theorem 3.)

Theorem 4. *L-repair checking is in PTIME and L-consistent query answers are co-NP-complete.*

To find if a repair r' is Pareto optimal we seek a tuple $y \in r \setminus r'$ whose all neighbors in r' are dominated by y. Such a tuple exists if and only if r' is not Pareto optimal. The tractability of P-checking implies that computing P-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 Pareto optimal; the machine finds the answer to the query in r' (if r' is not Pareto then the machine halts with the answer 'yes'). With Theorem 3 we obtain:

Corollary 1. *P-repair checking is in PTIME and P-consistent query answers are 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 (proofs of the following claims can be found in [10].)

Theorem 5. *G-repair checking is co-NP-complete and G-consistent query answers are Π_2^p-complete.*

The procedural nature of common repairs makes it possible to check if a repair r' belongs to C-$Rep(r, F, \succ)$ with a simulation of Algorithm $\mathcal{CR}$ 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. *C-repair checking is in PTIME and C-consistent query answers is co-NP-complete.*

4.3 Positive results

We show how to compute consistent query answers if only one key dependency is present. Note that under one key dependency all previously introduced families of repairs coincide and hence we simply talk about preferred repairs. We note that under one key dependency the corresponding conflict graph consists of disjoint cliques. A repair is obtained by choosing one tuple from each clique. If a priority is given then a preferred repair is obtained by choosing a non-dominated tuple from each clique, i.e. a tuple selected with the winnow operator.

Now, we show how to adopt the algorithm from [8] to find if true is the preferred consistent answer to a query Φ in r for a given priority $\succ$ w.r.t. one key dependency. We assume that the query is in CNF: $\Phi = \Phi_1 \wedge \ldots \Phi_n$. We note that true is not a preferred consistent query answer if and only if there exists a preferred repair r' such that $r' \not\models \Phi_i$ for some i. The algorithm attempts to find if such a repair exists for every i. If the algorithm is successful for some i then the answer is false; otherwise the answer is true. Let's fix i and consider $\neg\Phi_i$

$$\neg\Phi_i = R(t_1) \wedge \ldots \wedge R(t_k) \wedge \neg R(t_{k+1}) \wedge \ldots \wedge \neg R(t_m).$$

We use the following test:

Claim. For every $j \in \{1, \ldots, m\}$ if $t_j \in r$ we identify the clique C_j the tuple t_j belongs to. A preferred repair r' such that $r' \models \neg\Phi_i$ exists if and only if:

1. $\{t_1, \ldots, t_k\} \subseteq r$,
2. $\{t_1, \ldots, t_k\}$ is independent,
3. $\{t_1, \ldots, t_k\} \cap \{t_{k+1}, \ldots, t_n\} = \varnothing$,
4. $t_j \in \omega_\succ(C_j)$ for every $j \in \{1, \ldots, k\}$,
5. $\omega_\succ(C_j) \setminus \{t_{k+1}, \ldots, t_m\} \neq \varnothing$ for every $j \in \{k+1, \ldots, n\}$ such that $t_j \in r$.

Proof. $\Rightarrow$ The conditions 1, 2, and 3 are trivially implied. We observe that every tuple in r' is selected among non-dominated tuples from the clique it belongs to (with $\{t_1, \ldots, t_k\} \subseteq r'$ this proves 4) and r' is maximal, i.e. r' contains a tuple selected from every clique (with $\{t_{k+1}, \ldots, t_m\} \cap r' = \varnothing$ this proves 5).

$\Leftarrow$ We construct the repair r' as follows. First, for every $j \in \{1, \ldots, k\}$ we select the tuple t_j from C_j (feasible by 1, 2, and 3). The condition 4 guarantees that none of the tuples $t_{k+1}, \ldots, t_m$ is selected in this step. Next, for every $j \in \{k+1, \ldots, m\}$, such that $t_j \in r$, from the clique C_j we select a tuple different from $t_{k+1}, \ldots, t_m$ (feasible by 5). Finally, we select an arbitrary non-dominated tuple from every remaining clique. Obviously, $r' \models \neg\Phi_i$.

We also observe that the described test can be performed in time polynomial in the size of the database.

Along the lines of [2, 8] this algorithm can be further extended to handle one functional dependency. Because for one FD the family of locally optimal repairs does not coincide with other families of preferred repairs, we can extend this algorithm is two different directions: computing the preferred consistent query answers w.r.t. locally optimal repairs and w.r.t. Pareto optimal repairs.

Theorem 6. *For one functional dependency computing preferred consistent query answers is in PTIME for* L*-Rep,* P*-Rep,* G*-Rep, and* C*-Rep.*

5 Related work

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

The first article to notice the importance of priorities in information systems is [11]. 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 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 constraint 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 [6], 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 C-repairs. Furthermore, this approach is generalized to priorities being a partial order, by considering all extensions to weak orders. Again, however, this approach assumes the transitivity of priority on conflicts, which as we explained previously may be considered a significant restriction.

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 Proposition 5, [22] lifts a total order on rules to a preference on (extended) answers sets. When computing answers only maximally preferred answers sets are considered.

[21] investigate disjunctive logic programs with priorities on facts. 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 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). We note that a

family of preferred repairs defined in analogous manner does not satisfy $\mathcal{P}2$ and $\mathcal{P}4$ but satisfies $\mathcal{P}1$, $\mathcal{P}3$, and $\mathcal{P}5$.

A simpler approach to the problem of inconsistent logic programs is presented in [16]. 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 $\mathcal{P}1$ and $\mathcal{P}4$). However, if the priority is not total, the returned instance is not a repair and therefore the $\mathcal{P}5$ is not satisfied. Such an approach leads to a loss of (disjunctive) information and does not satisfy $\mathcal{P}2$ and $\mathcal{P}3$.

[12] 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 $\mathcal{P}1$ and $\mathcal{P}3$ and doesn't satisfy $\mathcal{P}2$ and $\mathcal{P}4$. [12] also describes how to translate conditioned active integrity constraints into a prioritized logic program [21], 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 environments (like general disjunctive logic programs) can be used to compute preferred answers.

[19] 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 $\mathcal{P}4$). 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 [15]. 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 properties $\mathcal{P}3$ and $\mathcal{P}5$ are trivially satisfied, but because this form of preference information does not have natural notions of extensions and maximality, it is hard to discuss postulates $\mathcal{P}2$ and $\mathcal{P}4$. 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 [14]. 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 $\mathcal{P}2$ but not $\mathcal{P}1$. A method of weakening the repair constraints is propose to get $\mathcal{P}1$, however this comes at the price of losing $\mathcal{P}2$.

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: *L-Rep*, *P-Rep*, *G-Rep*, and *C-Rep*. Figure 6 summarizes the computational complexity results; its first row is taken from [8].

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

Fig. 6. 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 of preferences [14] leads to losing the monotonicity. A modified, conditional, version of monotonicity may be necessary to capture non-trivial families of repairs.

Finally, one can extend our framework to handle a broader class of constraints. Conflict graphs can be generalized to hypergraphs [8], necessary to deal with 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 (TCS)*, 296(3):405–434, 2003.
3. L. Bertossi. Consistent Query Answering in Databases. *SIGMOD Record*, 2006. *To appear.*
4. L. Bertossi and J. Chomicki. Query Answering in Inconsistent Databases. In J. Chomicki, R. van der Meyden, and G. Saake, editors, *Logics for Emerging Applications of Databases*, pages 43–83. Springer-Verlag, 2003.

5. 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.
6. G. Brewka. Preferred Subtheories: An Extended Logical Framework for Default Reasoning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1043 – 1048, 1989.
7. J. Chomicki. Preference Formulas in Relational Queries. *ACM Transactions on Database Systems (TODS)*, 28(4):427–466, December 2003.
8. J. Chomicki and J. Marcinkowski. Minimal-Change Integrity Maintenance Using Tuple Deletions. *Information and Computation*, pages 90–121, 2005.
9. 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. ACM Press, November 2004.
10. 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.
11. 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.
12. 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.
13. A. Fuxman, E. Fazli, and R. J. Miller. ConQuer: Efficient Management of Inconsistent Databases. In *ACM SIGMOD International Conference on Management of Data*, 2005.
14. G. Greco and D. Lembo. Data Integration with Preferences Among Sources. In *International Conference on Conceptual Modeling (ER)*, pages 231–244. Springer, November 2004.
15. 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.
16. B. N. Grosof. Prioritized Conflict Handling for Logic Programs. In *International Logic Programming Symposium*, pages 197–211, 1997.
17. J. Y. Halpern. Defining Relative Likehood in Partially-Ordered Preferential Structures. *Journal of Artificial Intelligence Research*, 1997.
18. D. B. Lomet. Letter from the Editor-in-Chief. *IEEE Data Eng. Bull.*, 23(4), 2000.
19. A. Motro, P. Anokhin, and A. C. Acar. Utility-based Resolution of Data Inconsistencies. In *International Workshop on Information Quality in Information Systems (IQIS)*, pages 35–43. ACM, 2004.
20. E. Rahm and H. H. Do. Data Cleaning: Problems and Current Approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.
21. C. Sakama and K. Inoue. Prioritized logic programming and its application to commonsense reasoning. *Artificial Intelligence*, 123:185–222, 2000.
22. 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.
23. M. Y. Vardi. The Complexity of Relational Query Languages. In *ACM Symposium on Theory of Computing (STOC)*, pages 137–146, 1982.

24. 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.