On Approximating Optimum Repairs for Functional Dependency Violations

Solmaz Kolahi
University of British Columbia
solmaz@cs.ubc.ca

Laks V. S. Lakshmanan
University of British Columbia
laks@cs.ubc.ca

ABSTRACT

We study the problem of repairing an inconsistent database that violates a set of functional dependencies by making the smallest possible value modifications. For an inconsistent database, we define an optimum repair as a database that satisfies the functional dependencies, and minimizes, among all repairs, a distance measure that depends on the number of corrections made in the database and the weights of tuples modified. We show that like other versions of the repair problem, checking the existence of a repair within a certain distance of a database is NP-complete. We also show that finding a constant-factor approximation for the optimum repair for any set of functional dependencies is NP-hard. Furthermore, there is a small constant and a set of functional dependencies, for which finding an approximate solution for the optimum repair within the factor of that constant is also NP-hard. Then we present an approximation algorithm that for a fixed set of functional dependencies and an arbitrary input inconsistent database, produces a repair whose distance to the database is within a constant factor of the optimum repair distance. We finally show how the approximation algorithm can be used in data cleaning using a recent extension to functional dependencies, called conditional functional dependencies.

Categories and Subject Descriptors

H.2.0 [**Database Management**]: General- Security, integrity, and protection; H.2.4 [**Database Management**]: Systems- Relational databases

General Terms

Algorithms, Theory

Keywords

Functional Dependency Violation, Inconsistent Databases, Repair, Approximation Algorithm

1. INTRODUCTION

Data dependencies or integrity constraints capture constraints that data values appearing in a database are required to satisfy, and have long been used for the purpose of database design. Key constraints and functional and inclusion dependencies are prime examples of such constraints. Recently, with the popularity of data integration systems, we often encounter large databases that violate an underlying set of constraints and hence are inconsistent.

Salvaging the information in an inconsistent database has been attacked in several ways. The different attempts at dealing with inconsistent data can be categorized into three different approaches according to a recent survey by Fan [15]. Most of these attempts rely on the notion of *minimal repair* [3, 5] to an inconsistent database, which is a database satisfying the constraints but at the same time is as close as possible to the original inconsistent database in the sense that a minimal set of modifications have been applied to the original database to yield the repair.

The first approach, often referred to as *consistent query answering* [3, 7, 11], tries to extract the most trustworthy answer to a query posed to an inconsistent database. A consistent answer to a query corresponds to running the query against *all* possible minimal repairs and taking the intersection of the resulting answer sets. The complexity of consistent query answering has been studied for different classes of integrity constraints [7, 3, 5, 12], and practical implementations have been proposed that compute consistent answers, for some restricted classes of first-order queries and constraints, using *query rewriting* [19, 20].

The second approach tries to actually *repair* an inconsistent database by minimally modifying it, so that the resulting database satisfies the integrity constraints. This modification can take the form of deleting or inserting tuples [12, 28] or value modifications [10, 8, 27]. Typically, we would like the repair to be as "close" to the original inconsistent database as possible. To achieve this, one defines a cost measure. The measure can make use of a weight associated with every tuple in the original database or with every attribute value of every tuple. The weight may reflect the confidence the data owner places in the original tuple of values. Additionally, the cost measure uses a distance measure between values. This distance may reflect, e.g., the edit distance or the numerical distance between an original value and the changed value. The overall distance of a repair to an inconsistent database is the weighted sum over tuples that are changed in the repair, of the distance between the original values and their changed values. In this line of study, we are typically interested in the *minimum* repair, i.e., a repair that enjoys the smallest distance to the original database among all its repairs.

The third approach aims at producing a *nucleus*, which is a condensed representation of all repairs that can be used for consistent query answering [27, 26, 4]. Computing consistent answers, minimum repairs, and nuclei has been shown to be not tractable for most classes of constraints and queries, and efficient heuristic [10] and approximation [8, 23, 18] approaches have been proposed for dealing with inconsistent data in practice.

In this paper, we focus on the problem of repairing a database that violates a set of functional dependencies by modifying attribute values. An important property of functional dependencies is that correcting a violation may introduce a new one, and this makes finding a minimum repair more challenging, compared to repairing w.r.t. some of the other integrity constraints (such as *local denial constraints* considered in [8]).

To deal with repairing an inconsistent database w.r.t. functional dependencies, we introduce *V-repairs*, which are databases that contain variables representing incomplete information. A V-repair reflects two types of changes made to the original database to resolve functional dependency violations: changing a constant to another constant whenever there is enough information for doing so, and changing a constant to a variable whenever we cannot suggest a constant for an incorrect value. This is done in a way that a ground repair will be produced if different values not in the active domain are assigned to different variables. So, basically, a V-repair represents a set of repairs obtained by constant-to-constant modifications. For an inconsistent database, we would ideally want to produce an *optimum V-repair* that is as close as possible to the original database.

We first illustrate V-repairs with an example.

Example 1. Figure 1(a) shows a database instance over *name*, country (*cnt*), province/state (*prov*), region (*reg*), area code (*arCode*), and *phone*. In the instance, 'Van' refers to Vancouver/Lower mainland, 'Vic' to Victoria region, etc. The database instance in Figure 1(a) violates the functional dependencies $\Sigma = \{cnt, arCode \rightarrow reg, \quad cnt, reg \rightarrow prov\}$. The V-repair shown in Figure 1(b) reflects two necessary value modifications to resolve the violations. In one modification, we change the value of *reg* 'Man' to the correct value of 'Van', and in the other, we replace the value 'CAN' with a variable v_1, showing that to obtain an optimum repair, one viable option is to change the value of country to something else. The semantics is that v_1 stands for a value outside the active domain of *cnt*. □

We define a measure of distance between a database and its repair that depends only on the number of value modifications made in the database and the weights of tuples modified. This measure is somewhat simpler compared to other measures [10, 8]. Following previous work [10], recall that tuple weights correspond to the confidence that the data owner places in their correctness. Our measure simply counts the number of value modifications, and thus amounts to treating all modifications as equally expensive. One key motivation for looking at simpler distance measures is that it helps us understand whether the previously established results on the hardness of finding a minimum repair can be sharpened, and in some cases, whether efficient approximation algorithms can be developed.

	name	*cnt*	*prov*	*reg*	*arCode*	*phone*
t_1	Smith	CAN	BC	Van	604	123 4567
t_2	Adams	CAN	BC	Van	604	765 4321
t_3	Simpson	CAN	BC	Man	604	345 6789
t_4	Rice	CAN	AB	Vic	604	987 6543

(a)

	name	*cnt*	*prov*	*reg*	*arCode*	*phone*
t_1	Smith	CAN	BC	Van	604	123 4567
t_2	Adams	CAN	BC	Van	604	765 4321
t_3	Simpson	CAN	BC	Van	604	345 6789
t_4	Rice	v_1	AB	Vic	604	987 6543

(b)

Figure 1: (a) A database instance violating $\Sigma = \{cnt, arCode \rightarrow reg, \quad cnt, reg \rightarrow prov\}$. (b) An optimum V-repair.

Our first result is that even with a simpler notion of repair and distance measure between database instances, we still cannot efficiently find an optimum V-repair or even a good approximation. Specifically, we show that like previously studied versions of minimum repair [10, 8], deciding whether there is a V-repair within distance k of a database is NP-complete, even for unary functional dependencies. We also show that if functional dependencies are part of the input, finding a repair whose distance to the original database is always within a constant factor of the optimum repair distance is NP-hard. Moreover, for a fixed set of functional dependencies, finding a $(1 + \varepsilon)$-approximation is also NP-hard for some $\varepsilon > 0$. In other words, finding an optimum V-repair is Max-$\mathcal{SNP}$-hard (for a background on hardness of approximation for optimization problems, see [14, 25, 24]).

After establishing the hardness results, we ask whether we can leverage the simplified distance measure, and develop efficient approximation algorithms for special cases. Indeed, the combination of V-repairs and simpler distance measure enables us to design an approximation algorithm that for a fixed set of functional dependencies and an arbitrary inconsistent database produces a V-repair, whose distance to the original database is within a constant factor of the distance of an optimum V-repair, where the constant depends on the set of functional dependencies.

Traditional database dependencies have recently been revisited to make them more expressive for the purpose of cleaning an inconsistent database [15]. *Conditional func-*

tional dependencies (CFDs) [9, 13] are extensions to traditional functional dependencies in the following ways: first, CFDs make it possible to restrict the application of a functional dependency to only a subset of a relation, and second, CFDs allow forcing attribute values not only to be equal to each other, but also to be equal to constants. For example, for the database instance of Figure 1, one can express the following constraints as CFDs: if country (*cnt*) is Canada ('CAN'), then the value of region (*reg*) determines the value of province (*prov*).[1] Furthermore, if country is Canada ('CAN') and area code (*arCode*) is 604, then *reg* must be 'Van'. We show that our approximation framework is applicable to conditional functional dependencies. More precisely, for a fixed set of CFDs, we can produce a V-repair for an arbitrary inconsistent database, whose distance to the original database is within a constant factor of an optimum V-repair distance.

1.1 Related Work

Here we compare our work with previous works on repairs that are most similar to ours. As already pointed out above, our distance measure is simpler than the one studied by Fan et al. [10, 13] and Bertossi et al. [8]. Hardness results for finding minimum repairs under value modifications were already established in previous work [8, 10, 16]. However, since our distance measure only counts the number of changes in the repair (similar to measure used in [16, 17]) and the hardness proof uses only (unary) functional dependencies, our intractability results cannot be derived from previous results.

Max-$\mathcal{SNP}$-hardness for approximating a minimum repair by modifying numerical attributes for a fixed set of *denial constraints* [22] has been proved by Bertossi et al. [8]. In this paper, we present a similar result by focusing only on a fixed set of functional dependencies and a simpler distance measure for databases. Approximation algorithms for finding the minimum repair for a fixed set of *local* denial constraints have also been presented [8, 23]. Local denial constraints have the property that by resolving a constraint violation no new violation could be generated. Since functional dependencies do not have this property, we need a new strategy for approximating the best repair.

Repairing a database using functional dependencies and conditional functional dependencies has been studied by others (including [10, 9, 13]). However, these works rely on heuristic approaches for finding a good repair. We are not aware of any approximation algorithms in this context. We believe that our approximation framework would be useful for practical cases when a guarantee of small number of changes is required, and it complements the heuristic approach developed in [10, 13].

Finally, we would like to add that our notion of V-repair is different from that of *fixes with variables* in [26, 27]. While our V-repairs are homomorphic to a consistent relation, they are not necessarily homomorphic to the original relation, unlike fixes with variables. A V-repair can be thought of as a special case of *conditioned tables* [2], for representing incomplete information, with no local conditions and a conjunction of inequalities, one for each pair of distinct variables.

[1]It is possible in some countries, regions may straddle state/provincial boundaries.

The rest of the paper is organized as follows: we give preliminary notations and definitions in Section 2. Then in Section 3, we introduce V-repairs for functional dependency violations, and we study the hardness of finding an optimum V-repair or an approximate solution to it. We present an approximation algorithm for finding an optimum V-repair for a fixed set of functional dependencies in Section 4, and in Section 5, we show how the approximation algorithm can be used for repairing with CFDs. We give concluding remarks in Section 6.

2. BACKGROUND AND NOTATIONS

An instance I of a relation schema $R(A_1, \ldots, A_m)$ is a set of tuples in $Dom(A_1) \times \ldots \times Dom(A_m)$, where $Dom(A_i)$ denotes the domain of attribute A_i, $i \in [1, m]$. The active domain of A_i is denoted by $ADom(A_i)$. For convenience, we associate every tuple in an instance with an identifier t that remains unchanged even if some values in the tuple change. Given an instance I, the set of positions of I is defined as $Pos(I) = \{(t, A_i) \mid t$ is an identifier for a tuple in I and $i \in [1, m]\}$. The set $\{A_1, \ldots, A_m\}$ is sometimes referred to as $sort(R)$. We denote the value contained in position $p = (t, A) \in Pos(I)$ by $I(t, A)$. If the instance I is clear from the context, we may write $t[A]$ instead of $I(t, A)$. We sometimes use the term tuple interchangeably with tuple identifier.

A functional dependency (FD) over attributes of R is an expression of the form $X \to Y$, where $X, Y \subseteq sort(R)$. An FD $X \to Y$ is *unary* if $|X| = 1$. An instance I of R satisfies $X \to Y$, denoted $I \models X \to Y$, if for every two tuples t_1, t_2 in I with $t_1[X] = t_2[X]$, we have $t_1[Y] = t_2[Y]$. An instance I satisfies a set of FDs Σ, if it satisfies all FDs in Σ. An FD $X \to Y$ is called trivial if $Y \subseteq X$. We say that an FD $X \to A$ is implied by Σ, written $\Sigma \models X \to A$, if for every instance I satisfying Σ, I satisfies $X \to A$. In this paper, we always assume that Σ is *minimal* (see [1]), i.e., a set of FDs of the form $X \to A$ (with a single attribute on the right-hand side), such that $\Sigma \not\models X' \to A$ for every $X' \subsetneq X$, and $\Sigma - \{X \to A\} \not\models X \to A$. We usually denote sets of attributes by X, Y, Z and single attributes by A, B, C, or D.

We deal with *inconsistent* instances of R that violate some of the FDs in Σ. Following other works on repairs [10], we assume that each tuple t in I is associated with a weight $w(t) > 0$, which could have different meanings such as the accuracy of data or a confidence value placed by a user. In the absence of such weights, we can simply assume they are all equal to 1.

For an attribute $A \in sort(R)$, an *implicant* X of A is a subset of $sort(R)$, such that $\Sigma \models X \to A$, $A \not{in} X$, and for every $X' \subsetneq X$, $\Sigma \not\models X' \to A$. Then $Imp(A)$ denotes the set of all implicants of A. An attribute A is called *primitive* if $Imp(A) = \emptyset$.

3. V-REPAIRS AND OPTIMUM REPAIR PROBLEM

We assume that we have an infinite set of variables $\boldsymbol{V} = \{v_1, v_2, \ldots\}$. Given an instance I of R that violates FDs in

Σ ($I \not\models \Sigma$), we define a *V-instance* for I to be an instance I_V with the same tuple identifiers (and therefore $Pos(I_V) = Pos(I)$),[2] such that for every position $p = (t, A_i)$ in $Pos(I_V)$, $I_V(p) = a$ for some $a \in ADom(A_i)$, or $I_V(p) = v_j$ for some $v_j \in \boldsymbol{V}$.

For a V-instance I_V, a *ground substitution* is a mapping σ from variables in I_V to $Dom(A_1) \cup \ldots \cup Dom(A_m)$, such that for each position $p = (t, A_i)$ in $Pos(I_V)$ with $I_V(p) = v_j$, $\sigma(v_j) = a$ for some $a \in Dom(A_i) \setminus ADom(A_i)$, and $\sigma(v_j) \neq \sigma(v_k)$ whenever $j \neq k$. Ground substitutions can be lifted to V-instances in the obvious way. The instance obtained by applying a ground substitution σ, denoted $\sigma(I_V)$, is called a *ground instance*. A V-instance is called a *V-repair* if for some ground substitution σ, $\sigma(I_V) \models \Sigma$. If for every ground substitution σ, $\sigma(I_V)$ violates an FD $X \to A$ in Σ, we simply say that I_V violates $X \to A$. Note that if the attribute domains are not finite, V-repairs always exist, and the satisfaction of an FD by a V-instance could be checked in polynomial time.

Our approach to repairing a database instance I that violates a set of functional dependencies is finding a V-repair that is as close as possible to I. More precisely,

Definition 2. *Given an instance I of schema R that violates a set of FDs Σ over attributes of R, we define the distance between I and a V-repair I_V as*

$$\Delta(I, I_V) = \sum_{(t,A) \in Pos(I)} w(t) \cdot dist(I(t, A), I_V(t, A)), \text{ where}$$

$$dist(a, b) = \begin{cases} 1 & \text{if } a \neq b \\ 0 & \text{if } a = b. \end{cases}$$

Repair Checking *for a set of FDs is the problem of deciding, given an integer $k \geq 0$ and an instance I, whether there is a V-repair I_V with $\Delta(I, I_V) \leq k$. A V-repair I_V is an* optimum repair *if the distance $\Delta(I, I_V)$ is the minimum among all V-repairs of I.* Optimum Repair *for a set of FDs is the problem of finding an optimum repair for a given instance I.*

In this section, we show that like other definitions of "best" repair for an inconsistent database, finding an optimum repair does not seem to be possible in polynomial time in the size of the input database, even when the FDs are unary. We will also show that if the set of functional dependencies is given as an input, it is NP-hard to approximate the optimum repair within any constant factor. Furthermore, if we fix a set of FDs, it is NP-hard to approximate the optimum repair within an arbitrary small constant factor. It is worth mentioning that all of our hardness results can be obtained in the absence of tuple weights. These results motivate the next section, where we present a constant-factor approximation algorithm for finding an optimum V-repair for any fixed set of FDs and an arbitrary violating instance.

Theorem 3. *There is a set of unary functional dependencies for which* Repair Checking *is NP-complete.*

[2]Technically, we need to define an isomorphism between I, I_V, but for simplicity, we deliberately assume that the set of tuple identifiers are equal.

Proof. For every set of functional dependencies, it is easy to see that Repair Checking is in NP: given a V-instance I_V for an input inconsistent instance I, it is possible to check whether I_V satisfies the FDs and measure $\Delta(I, I_V)$ in polynomial time.

Now consider relation schema $R(Edge, Vertex, Dummy)$ and the set of FDs $\Sigma = \{Edge \to Vertex,\ Vertex \to Dummy\}$. To prove hardness, we reduce from the vertex cover problem. Given a graph $\mathcal{G}(U, E)$, a vertex cover is a subset C of U such that for every edge $(u_1, u_2) \in E$, C contains at least one of u_1 or u_2.

Given a graph $\mathcal{G}(U, E)$, we create an instance I of R that violates the set of FDs Σ as follows: for every edge $e_r = (u_i, u_j)$ in E, I contains two tuples $t_{ri}(e_r, u_i, 0)$ and $t_{rj}(e_r, u_j, 0)$. Moreover, for every vertex $u_i \in U$, I contains a tuple $t_i(f_i, u_i, 1)$, where f_i is an edge identifier not in E, and $f_i \neq f_j$ for $i \neq j$. Every tuple t in I has a weight $w(t) = 1$. Obviously, there is a pair of tuples for every edge in the graph that violates $Edge \to Vertex$, and for every vertex connected to an edge, there is at least a pair of tuples violating $Vertex \to Dummy$. We can see that the graph has a vertex cover C of size $|C| \leq k$ if and only if I has a V-repair I_V with $\Delta(I, I_V) \leq |E| + k$. First, let C be a vertex cover of size k. We produce V-repair I_V as follows: for every edge $e_r = (u_i, u_j)$ with $u_i \in C$, I_V has tuples $t_{ri}(e_r, u_i, 0)$ and $t_{rj}(e_r, u_i, 0)$ (no matter if u_j is also in C). For every vertex $u_i \in C$, I_V contains tuple $t_i(f_i, v_l, 1)$ for a fresh variable $v_l \in \boldsymbol{V}$, and for every vertex $u_i \notin C$, I_V contains the unchanged tuple $t_i(f_i, u_i, 1)$. Clearly, $\Delta(I, I_V)$ is exactly $|E| + k$.

Conversely, suppose that there is a V-repair I_V with $\Delta(I, I_V) = |E| + k$ for some k smaller than k', the size of a minimum vertex cover. Let C be the set of vertices u_i such that their corresponding tuple t_i shows at least one change in I_V. It is easy to see that for every edge $e_r \in E$, I_V should have changed at least one position in each pair of tuples t_{ri}, t_{rj}, due to the violation of $Edge \to Vertex$. Therefore, $\Delta(I, I_V) \geq |E| + |C|$, and thus $|C| \leq k < k'$. Since C cannot be a vertex cover due to $|C| < k'$, there are at least $k' - |C|$ edges between the remaining vertices u_i in $V - C$, whose corresponding tuples t_i show no change in I_V. For each edge e_r among these $k' - |C|$ edges, we need to see at least *two* changes in the pair of tuples t_{ri}, t_{rj} due to the violation of $Vertex \to Dummy$. We thus have: $\Delta(I, I_V) \geq 2 \cdot (k' - |C|) + (|E| - k' + |C|) + |C| = |E| + k'$, which is a contradiction. □

The following theorem shows that approximating Optimum Repair for all sets of FDs within any constant factor is NP-hard.

Theorem 4. *Let $\alpha > 1$ be any constant. Then unless P=NP, there is no algorithm that approximates* Optimum Repair *for any given set of FDs within a factor of α, and runs in polynomial time in the size of the input FDs and inconsistent instance.*

Proof. We give a *gap-preserving reduction* (see [14, 25]) from the hypergraph vertex cover problem (or equivalently the set cover problem), whose approximation within any constant factor is known to be NP-hard [6]. Given a hypergraph $\mathcal{G}(U, E)$, a vertex cover (or a hitting set) is a subset C of U that overlaps every edge $e \in E$. A minimum vertex cover is a vertex cover with the smallest cardinality.

Given a hypergraph $\mathcal{G}(U, E)$, we create a relation schema R with $sort(R) = U \cup \{D\}$, a set of functional dependencies Σ, and an instance I of R as follows. For every edge $e_i = \{u_1, \ldots, u_m\}$ in E, there is a functional dependency $u_1 \ldots u_m \to D$ in Σ and a tuple t_i in I with $w(t_i) = W$ for a large positive W, where $t_i[u_j] = 0$ for $u_j \in e_i$, $t_i[u_j] = i$ for $u_j \notin e$, and $t_i[D] = i$. There is also another tuple t_0 in I with $w(t_0) = 1$, where $t_0[u_j] = 0$ for all $u_j \in U$ and $t_0[D] = 0$. It is easy to see that hypergraph $\mathcal{G}$ has a vertex cover of size k if and only if I has a V-repair I_V with $\Delta(I, I_V) = k$. Let $I_V^{\min}$ and $MinVertCover(\mathcal{G})$ denote an optimum V-repair for I and a minimum vertex cover for $\mathcal{G}$, respectively. Then every approximation algorithm that produces a V-repair I_V with $\Delta(I, I_V) \leq \alpha \cdot \Delta(I, I_V^{\min})$, for some constant $\alpha > 1$, will produce a vertex cover whose size is at most α times the size of $MinVertCover(\mathcal{G})$. More precisely, the reduction is gap-preserving since we have:

1. if $|MinVertCover(\mathcal{G})| \leq k$, then $\Delta(I, I_V^{\min}) \leq k$, and
2. if $|MinVertCover(\mathcal{G})| > \alpha \cdot k$, then $\Delta(I, I_V^{\min}) > \alpha \cdot k$.

□

The next theorem rules out the possibility of approximating Optimum Repair for a fixed set of FDs within an arbitrarily small factor in polynomial time in the size of the input inconsistent instance. To show that an optimization problem P is Max-$\mathcal{SNP}$-hard [24], it is enough, by definition, to show that P behaves just like MAX3SAT in terms of approximability, i.e., there is a gap-preserving reduction from MAX3SAT to P. Then we can immediately conclude that there exists $\varepsilon > 0$ such that achieving an approximation factor $(1 + \varepsilon)$ for P is NP-hard (see [14]).

Theorem 5. *For a fixed relation schema and set of FDs,* Optimum Repair *is Max-$\mathcal{SNP}$-hard. In other words, there exists $\varepsilon > 0$, a relation schema and a set of FDs, for which approximating* Optimum Repair *within a factor of $(1 + \varepsilon)$ is NP-hard.*

Proof. We give a gap-preserving reduction from MAX3SAT problem, whose approximation within $(1 - \varepsilon')$ factor, for some $\varepsilon' > 0$, is known to be NP-hard (see [14, 25]). Given a CNF formula, whose clauses have at most three literals, MAX3SAT is the problem of finding an assignment that maximizes the number of satisfied clauses.

Let $C = C_1 \wedge \ldots \wedge C_N$ be a CNF formula, where $C_i = l_{i1} \vee l_{i2} \vee l_{i3}$ for every clause C_i ($i \in [1, N]$), and each literal l_{ij} is either x or $\neg x$ for some variable $x \in X$ (we assume each clause has exactly three literals). We create an instance I of relation schema $R(Cls, Asn, Var, Lit)$, which violates FDs $\Sigma = \{Var, Asn \to Lit,\ Cls, Asn \to Var\}$, as follows: for every clause C_i and every literal l_{ij} of variable x_{ij} ($j \in [1, 3]$) in C_i, there is a tuple t_{ij} in I, where $t_{ij}[Cls] = C_i$, $t_{ij}[Asn] = 1$, $t_{ij}[Var] = x_{ij}$, $t_{ij}[Lit] = l_{ij}$ (x_{ij} or $\neg x_{ij}$). All tuples have weight equal to 1. Let $MAX3SAT(C)$ denote the maximum number of clauses satisfiable over all assignments, and $\Delta_{\min}$ be the distance between I and any optimum V-repair. Then it is enough to show that:

1. if $MAX3SAT(C) = k$, then $\Delta_{\min} = 3N - k$, and
2. if $MAX3SAT(C) < (1 - \varepsilon')k$ for some $\varepsilon' > 0$, then $\Delta_{\min} > (1 + \varepsilon)(3N - k)$ for some $\varepsilon > 0$.

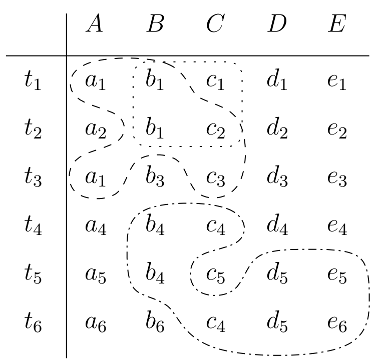
Suppose $MAX3SAT(C) = k$. Then there exists a truth assignment that satisfies exactly k clauses. First, we build a V-repair I_V with $\Delta(I, I_V) = 3N - k$ as follows: for every satisfied clause $C_i = l_{i1} \vee l_{i2} \vee l_{i3}$, assuming l_{i1} is assigned true, I_V has three tuples $t_{i1} = (C_i, 1, x_{i1}, l_{i1})$, and $t_{ij} = (C_i, v_{ij}, x_{ij}, l_{ij})$ for $j = 2, 3$, where v_{ij}s are fresh variables taken from $\boldsymbol{V}$. For every unsatisfied clause $C_i = l_{i1} \vee l_{i2} \vee l_{i3}$, I_V has three tuples $t_{ij} = (C_i, v_{ij}, x_{ij}, l_{ij})$ for $j = 1, 2, 3$, where v_{ij}s are fresh variables taken from $\boldsymbol{V}$. Clearly, I_V is a V-repair with $\Delta(I, I_V) = 2k + 3(N - k) = 3N - k$. Second, we show that for every V-repair I'_V, $\Delta(I, I'_V) \geq 3N - k$. Let D be the set of clauses C_i with at least one tuple t_{ij} whose all positions has remained unchanged in I'_V. Then there is a truth assignment that satisfies at least $|D|$ clauses. We thus have $|D| \leq k$. For every clause $C_i \in D$, there are at least two positions p_1, p_2 in the three tuples t_{ij} ($j \in [1, 3]$), such that $I(p_1) \neq I'_V(p_1)$ and $I(p_2) \neq I'_V(p_2)$. This is due to violations of FD $Cls, Asn \to Var$. Moreover, for every clause $C_i \notin D$, each of the three corresponding tuples have at least one changed position. Therefore, $\Delta(I, I'_V) \geq 2 \cdot |D| + 3 \cdot (N - |D|) = 3N - |D| \geq 3N - k$. To prove part 2, using the fact that $MAX3SAT(C) \geq N/2$, we have $\Delta_{\min} = 3N - MAX3SAT(C) > 3N - (1 - \varepsilon')k = 3N - k + \varepsilon'/5(5k) \geq (1 + \varepsilon'/5)(3N - k)$. □

4. A CONSTANT-FACTOR APPROXIMATION

Theorem 5 rules out the possibility of finding a *polynomial-time approximation scheme* for the Optimum Repair problem of a set of FDs. We show, in this section, that we were nevertheless able to find a repair algorithm that runs is polynomial time in the size of the input inconsistent instance, and produces a V-repair whose distance to the input instance is within a constant factor of the optimum repair distance, where the constant factor depends on the set of FDs.

The algorithm starts by finding all FD violations in the instance, and puts each of them in a hyperedge of a *conflict hypergraph*. Let I be an instance of relation schema $R(A_1, \ldots, A_m)$ that is inconsistent w.r.t. a set of functional dependencies Σ. We assume, without loss of generality, that Σ is minimal. Note that this is an important assumption for the correctness of the algorithm.

Definition 6. *The* initial conflict hypergraph *of instance I is defined to be $\mathcal{G}_I = (V, E)$, where V is the set of positions $p = (t, A)$ in $Pos(I)$ with weight $w(t)$, and*



	A	B	C	D	E
t_1	a_1	b_1	c_1	d_1	e_1
t_2	a_2	b_1	c_2	d_2	e_2
t_3	a_1	b_3	c_3	d_3	e_3
t_4	a_4	b_4	c_4	d_4	e_4
t_5	a_5	b_4	c_5	d_5	e_5
t_6	a_6	b_6	c_4	d_5	e_6

Figure 2: Hyperedges in an initial conflict hypergraph.

- *for every functional dependency $X \rightarrow A \in \Sigma$ and two tuples t_1, t_2 in I that violate $X \rightarrow A$, the set of positions $\{p \mid p = (t_1, B)$ or $p = (t_2, B), B \in XA\}$ form a hyperedge in E, which is called an* **initial pair conflict**.

- *for every three tuples t_1, t_2, t_3 in I such that:*
  - *there exists an FD $X \rightarrow A$ such that $t_1[X] = t_2[X]$, and*
  - *there exists an FD $Y \rightarrow A$ such that $t_1[Y] = t_3[Y]$, and*
  - *$t_2[A] \neq t_3[A]$,*

  the set of positions $\{p = (t_1, C) \mid C \in XY\} \cup \{p = (t_2, C) \mid C \in XA\} \cup \{p = (t_3, C) \mid C \in YA\}$, form a hyperedge in E, which is called an **initial triple conflict**.

- *for every three tuples t_1, t_2, t_3 in I such that:*
  - *there exists an FD $X \rightarrow A$ such that $t_1[X] = t_2[X]$, and*
  - *there exists an FD $Y \rightarrow B$ such that $A \in Y$, $t_2[A] = t_3[A]$, and $t_1[Y - A] = t_3[Y - A]$, and*
  - *$t_1[B] \neq t_3[B]$,*

  the set of positions $\{p = (t_1, C) \mid C \in (XYB - \{A\})\} \cup \{p = (t_2, C) \mid C \in XA\} \cup \{p = (t_3, C) \mid C \in YB\}$, form a hyperedge in E, which is also called an **initial triple conflict**.

Intuitively, each hyperedge in the initial conflict hypergraph of an instance represents a conflicting set of positions that cannot completely remain unchanged in any repair. In other words, in every repair of the instance, at least one position in every hyperedge should get a new value. As a useful consequence we have:

PROPOSITION 7. *The weight of a minimum vertex cover of the initial conflict hypergraph $\mathcal{G}_I$ is a lower bound on the distance between instance I and any optimum V-repair $I_V^{\min}$.*

```
Input: An instance $I$ of schema $R$ violating FDs $\Sigma$.
Output: A V-repair $I_V$ for $I$.

# STEP 1: CAPTURING INITIAL VIOLATIONS:
(1)  create initial conflict hypergraph $\mathcal{G}_I$ for $I$;
(2)  find an approximation $VC$ for minimum vertex cover
     in $\mathcal{G}_I$;
(3)  $change := VC$;
(4)  $I_{vc} := I$; $i := 1$;
(5)  while there are tuples $t_1, t_2$ in $I_{vc}$ violating an FD
     $X \rightarrow A \in \Sigma$ s.t. $(t_1, A)$ is the only position in $VC$ do
(6)    $I_{vc}(t_1, A) := I_{vc}(t_2, A)$;
(7)    $change := change - \{(t_1, A)\}$;
(8)  for every position $p \in change$ do
(9)    $I_{vc}(p) := v_i$; $i := i + 1$;
(10) $I_V := I_{vc}$;

# STEP 2: RESOLVING NEW VIOLATIONS:
(11) while there are violating tuples do
(12)   let $p = (t, B)$ be a position in $VC$ with smallest
       ratio $\frac{w(t)}{|\text{violations involving } p|}$;
(13)   let $CI$ be a set of attributes in $MinCorImp_\Sigma(B)$;
(14)   for every attribute $C \in (CI \cup \{B\})$ do
(15)     $I_V(t, C) := v_i$; $i := i + 1$;
(16) return $I_V$.
```

Figure 3: Algorithm FINDVREPAIR.

The notion of conflict hypergraph was previously used in managing inconsistency in databases (e.g., see [12, 8, 5]). The main difference is that the hypergraphs considered before contained database tuples as vertices and constraint violations as hyperedges. Since we believe that functional dependency violations could be resolved at the level of changing attribute values, versus deleting whole tuples, we create conflict hypergraphs in which vertices are positions in an inconsistent database.

EXAMPLE 8. Consider relation schema $R(A, B, C, D, E)$ with FDs $\Sigma = \{A \rightarrow C, B \rightarrow C, CD \rightarrow E\}$. In instance I shown in Figure 2, we can see three different types of hyperedges in the initial conflict hypergraph (not all hyperedges are shown). It is easy to see that in any repair of this instance, the value of at least one position in each hyperedge should be modified. $\square$

The basic idea of the FINDVREPAIR algorithm, shown in Figure 3, is as follows. First, we construct the initial conflict hypergraph $\mathcal{G}_I$ for the input inconsistent instance I. Then we find an approximation VC for a minimum vertex cover in $\mathcal{G}_I$ using the known algorithms in the graph theory literature. Positions in VC are the initial set of positions whose values need to be changed. For each position in VC, we either put a value from the active domain if there are FDs forcing it, or we put a fresh variable from $\boldsymbol{V}$. The resulting V-instance may still contain FD violations, which we resolve in the last step by changing the values in some positions in a way that no new violation may be introduced.

Before we explain how we resolve the FD violations that arise after fixing values in the vertex cover VC, we introduce the notion of *core implicant* for attributes in a relation schema. Let R be a relation schema and Σ be a set of functional dependencies on the attributes of R.

Definition 9. *For every attribute A in $sort(R)$, a* core implicant *of A is defined as a minimal set CI of attributes that intersects with every implicant of A. That is, $CI \cap X \neq \emptyset$ for every X such that Σ implies the nontrivial FD $X \rightarrow A$. A* minimum core implicant *for an attribute A is a core implicant with the smallest cardinality. The sets of all core and all minimum core implicants for an attribute A is denoted by $CorImp_\Sigma(A)$ and $MinCorImp_\Sigma(A)$ respectively.*

To resolve new FD violations by doing a limited number of value modifications, the FindVRepair algorithm uses the following property of core implicants:

Proposition 10. *For every non-primitive attribute B in a core implicant $CI \in CorImp_\Sigma(A)$, $CI \cup \{A\}$ contains a core implicant $CI' \in CorImp_\Sigma(B)$.*

Proof. Suppose for some attribute $B \in CI$, there is an implicant Y such that $Y \cap (CI \cup \{A\}) = \emptyset$. Let X be an implicant of A that contains B (such implicants exist since CI is minimal). If B is the only attribute in the intersection of X and CI (i.e., $X \cap CI = \{B\}$), then $Y \cup (X - \{B\}) \rightarrow A$ is a nontrivial functional dependency whose left-hand side does not have any intersection with CI, which means CI cannot be a core implicant. Therefore, every such implicant X of A that contains B must have at least one more attribute in common with CI. This means B is a redundant attribute in CI, which contradicts with the assumption of minimality for CI.

Consequently, every implicant Y of a non-primitive attribute $B \in CI$ must have a non-empty intersection with $(CI \cup \{A\})$, which concludes the proof. □

We now explain step 2 of the algorithm (lines (11)–(15)) that tries to resolve FD violations that arise as a result of running step 1 (lines (1)–(9)). Let I_V be the V-instance after making corrections for positions in the vertex cover VC, and t_i, t_j be two tuples in I_V violating a functional dependency $X \rightarrow A$. Since this is a violation that happens after we resolve the initial pair and triple conflicts, at least two positions (t_i, B) and (t_j, D) are in the vertex cover VC, where $B, D \in XA$. This is due to the way that the initial pair and triple conflict hyperedges were picked.

Lemma 11. *For every two tuples t_i, t_j in I_V violating an FD $X \rightarrow A$, there are at least two positions p_1, p_2 in the intersection of vertex cover VC and the set $S = \{p \mid p = (t_i, B)$ or $p = (t_j, B), B \in XA\}$.*

Proof. First, if none of the positions in S are in VC, this would be an initial violation, and S forms an initial pair conflict hyperedge, which means it should have contained at least one position in the vertex cover. Now suppose that only one position $p = (t_i, A) \in S$ is in the vertex cover VC. Clearly, $I_V(p)$ cannot contain a variable, because the FD $X \rightarrow A$ would force $I_V(p)$ to be equal to $I_V(t_j, A)$. If $I_V(p) = a$ for some $a \in ADom(A)$, it means that there is a tuple t_k in I and an FD $Y \rightarrow A$ such that $t_k[Y] = t_i[Y]$, $t_j[X] = t_i[X]$ and $t_k[A] = a \neq t_j[A]$. Then positions $\{p = (t_i, C) \mid C \in XY\} \cup \{p = (t_j, C) \mid C \in XA\} \cup \{p = (t_k, C) \mid C \in YA\}$ form an initial triple conflict hyperedge, which means either $I_V(p) \neq a$, or t_i, t_j in I_V cannot violate $X \rightarrow A$.

If the only position from S in VC is of the form $p = (t_i, B)$ for some $B \in X$, $I_V(p)$ cannot contain a variable, because that would not lead to a violation of $X \rightarrow A$. Therefore, $I_V(p) = b$ for some $b \in ADom(B)$, and thus there is a tuple t_k in I and an FD $Y \rightarrow B$ such that $t_k[Y] = t_i[Y]$, $t_i[X - B] = t_j[X - B]$, $t_j[B] = t_k[B]$, and $t_i[A] \neq t_j[A]$. Then positions $\{p = (t_i, C) \mid C \in (XYA - \{B\})\} \cup \{p = (t_j, C) \mid C \in XA\} \cup \{p = (t_k, C) \mid C \in YB\}$ form an initial triple conflict hyperedge, which means either $I_V(p) \neq b$, or t_i, t_j in I_V cannot violate $X \rightarrow A$. □

To resolve the violation, it is enough to pick one of (t_i, B) or (t_j, D) (say (t_i, B)), and change the value of $I_V(t_i, B)$ to a fresh variable from V. In order not to make new violations by this change, we also change the value in $I_V(t_i, C)$ for every attribute C in a minimum core implicant of B to a fresh variable. Then t_i, t_j no longer agree on X, and thus the violation is gone. Furthermore, no new violation will be introduced by these changes since for every attribute value that we change into a fresh variable, we also change the value of a core implicant of the attribute into a fresh variable. This is due to Proposition 10.

In step 2 of the algorithm, we basically need to resolve a number of violations, each of which involving at least two positions in vertex cover VC. If we do this in a greedy manner by picking the most cost-effective position at each step (smallest ratio $\frac{w(t)}{|\text{violations involving } p|}$), we manage to cover all violations by picking a set of positions in VC whose total weight is at most half of the total weight of all postions in VC. This is similar to the idea used in the greedy approximation algorithm for weighted set cover problem [25].

Theorem 12. *For every set of FDs,* FindVRepair *is a polynomial-time constant-factor approximation algorithm for* Optimum Repair.

Proof. It is easy to see that the output of the algorithm is a V-repair for an input instance I, and is produced in polynomial time in the size of I. In step 1 we capture all initial violations by changing the values in positions that fall in the vertex cover. Then in step 2 we keep resolving new violations by changing values to fresh variables, causing no new violation since we do not use a variable for two positions and we also assume $v_i \neq v_j$ for $i \neq j$ (more precisely, no ground substitution assigns equal values to different variables). Clearly, this process eventually terminates outputting a V-instance that does not contain any FD violation.

Now we need to show that for every instance I of a schema R that violates FDs in Σ, the algorithm outputs a V-repair I_V, such that:

$$\Delta(I, I_V) \leq \alpha \cdot \Delta(I, I_V^{\min}),$$

where α is a constant that does not depend on I. We know that if $I_V^{\min}$ is a V-repair, in each initial pair and triple conflict in I, there is at least one position p whose value is not the same in I and $I_V^{\min}$. Therefore, the weight of a minimum vertex cover of the initial conflict hypergraph $\mathcal{G}_I$ is a lower bound on the distance between I and $I_V^{\min}$ (Proposition 7):

$$\Delta(I, I_V^{\min}) \geq \mathit{weight}(\mathit{MinVertCover}(\mathcal{G}_I)).$$

We know that if the size of the hyperedges in a hypergraph is bounded by f (or equivalently, in the set cover problem, the frequency of occurrence of every element in the sets is bounded by f), then there is an approximation algorithm for finding the minimum vertex cover with approximation factor f (see [14, 25]). In our problem, the size of the hyperedges does not depend on the instance, and is determined by the number of attributes involved in a violated functional dependency. Let MFS be the maximum number of attributes involved in a functional dependency in Σ, which is minimal. It is easy to see that each hyperedge of an initial pair conflict contains at most $2 \cdot \mathit{MFS}$ positions, and each hyperedge of an initial triple conflict contains at most $4 \cdot \mathit{MFS} - 2$ positions. Therefore,

$$f \leq 4 \cdot \mathit{MFS} - 2.$$

Let VC be the vertex cover selected by the approximation algorithm that we apply to the initial conflict hypergraph. Then for I_{vc}, which is the V-instance that we obtain by replacing values in positions selected by the vertex cover algorithm, we have

$$\begin{aligned}\Delta(I, I_{vc}) &\leq \mathit{weight}(VC)\\ &\leq f \cdot \mathit{weight}(\mathit{MinVertCover}(\mathcal{G}_I))\\ &\leq (4 \cdot \mathit{MFS} - 2) \cdot \mathit{weight}(\mathit{MinVertCover}(\mathcal{G}_I)).\end{aligned}$$

During step 2 of the algorithm, for each remaining violation involving a position (t_1, B) in the vertex cover, we change the value of at most $|CI|$ additional positions into fresh variables for some core implicant $CI \in \mathit{MinCorImp}_\Sigma(B)$, which are all in the same tuple t_1 sharing the weight $w(t_1)$. We do this in a way that these positions never involve in any FD violation from the moment we fix them. Furthermore, at each step we resolve remaining violations by picking the most cost-effective position of VC. If MCI denotes the size of the largest minimum core implicant over all attributes, then after resolving all violations in I_V, we have added at most $\frac{1}{2} \cdot \mathit{MCI} \cdot \mathit{weight}(VC)$ to the distance between the original instance I and instance I_{vc}. Putting everything together, we have

$$\begin{aligned}\Delta(I, I_V) &\leq (\tfrac{1}{2} \cdot \mathit{MCI} + 1) \cdot \Delta(I, I_{vc})\\ &\leq (\tfrac{1}{2} \cdot \mathit{MCI} + 1) \cdot (4 \cdot \mathit{MFS} - 2) \cdot\\ &\quad \mathit{weight}(\mathit{MinVertCover}(\mathcal{G}_I))\\ &\leq (\mathit{MCI} + 2) \cdot (2 \cdot \mathit{MFS} - 1) \cdot \Delta(I, I_V^{\min}),\end{aligned}$$

which completes the proof. □

It would be interesting to ask whether any natural condition on FDs (e.g., a normal form) guarantees a small constant factor for the approximation algorithm. For instance, we can guarantee $\mathit{MCI} = 1$ if we manage to partition the set of attributes $\mathit{sort}(R)$ into disjoint sets $\{A_1, \ldots, A_k\}$, such that there is no FD involving attributes from two different sets, and for each set there is an FD $A_1 \ldots A_{k-1} \rightarrow A_k$. Interestingly, this is similar to a recently-studied restricted version of third normal form, called 3NF$^+$ [21].

5. REPAIRING WITH CONDITIONAL FUNCTIONAL DEPENDENCIES

An extension to traditional functional dependencies called *conditional functional dependencies* (CFDs) has recently been introduced [9, 13] that is capable of representing accurate information and is useful in data cleaning. Using a CFD we are able to express constraints that bind a set of semantically related values. Here, we would like to show that given a database instance that violates a set of CFDs, we can use a similar technique to find an approximation for an optimum V-repair.

Like traditional FDs, every set of CFDs has a minimal cover [9]. We therefore assume that we are given a minimal set of CFDs that contain constraints of the form $(X \rightarrow A, t_p)$, where $X \rightarrow A$ is a standard FD, and t_p is a *pattern tuple* on attributes XA. For every attribute $B \in XA$, $t_p[B]$ is either a constant in the domain of B or the symbol '_'.

To define the semantics of CFDs, we need an operator $\asymp$. For two symbols u_1, u_2, we have $u_1 \asymp u_2$ if either $u_1 = u_2$ or one of u_1, u_2 is '_'. This operator naturally extends to tuples. Let I be an instance of relation schema R, and Σ be a set of CFDs over the attributes of R. Instance I satisfies a CFD $(X \rightarrow A, t_p)$ if for every two tuples t_1, t_2 in I, $t_1[X] = t_2[X] \asymp t_p[X]$ implies $t_1[A] = t_2[A] \asymp t_p[A]$. Instance I satisfies Σ if it satisfies all CFDs in Σ. For example, for relation schema $R(\mathit{name}, \mathit{cnt}, \mathit{prov}, \mathit{reg}, \mathit{arCode}, \mathit{phone})$, where the attributes represent name, country, province, region, area code, and phone number, we can express the following two CFDs: 1. $(\mathit{cnt}, \mathit{arCode} \rightarrow \mathit{reg}, (\mathrm{CAN}, 604, \mathrm{Van}))$, which means whenever country is Canada and area code is 604, then the region must be Vancouver/Lower mainland. 2. $(\mathit{cnt}, \mathit{reg} \rightarrow \mathit{prov}, (\mathrm{CAN}, \_, \_))$, which means whenever country is Canada, the value of region uniquely determines the province.

The notion of V-repairs can be extended for CFDs in a natural way. We now show how the approximation algorithm of Section 4 can be extended, so it can be used for repairing an instance using a set of CFDs. The new algorithm starts with a step 0 (shown in Figure 4), which is necessary to ensure that corrections are made based on the information provided by pattern tuples. This step adds pattern tuples of CFDs that enforce a fixed constant for an attribute to the original instance. These added tuples have a very large weight to ensure that their positions never fall in the vertex cover.

The initial conflict hypergraph for an instance violating a set of CFDs can be built in a very similar way after artificially adding pattern tuples to the instance. Again, we need to

```
# STEP 0: ENFORCING PATTERN TUPLES
(i)   for every CFD $(X \to A, t_p)$ s.t. $t_p[A]$ is not '_' do
(ii)      create tuple $t$ with $t[XA] := t_p[XA]$, and for every
          attribute $B \notin XA$, $t[B] := v$ for some fresh variable
          $v \in \mathbf{V}$, and $w(t) := W$ for some large positive
          integer $W$;
(iii)     $I := I \cup \{t\}$;
```

Figure 4: Additional step for extending algorithm FINDVREPAIR **for CFDs.**

form sets of positions violating a CFD, whose values cannot all together remain unchanged in any repair. After finding a vertex cover for the initial conflict hypergraph and proposing new values for positions in the vertex cover that are forced to some value according to CFDs, we need to resolve possible new violations.

The notion of core implicant can also be extended for CFDs. Intuitively, a core implicant of an attribute A is a set of attributes X such that whenever we change the value in positions (t, B), $B \in XA$, for a tuple t, into fresh variables, we ensure that those positions will never get involved in any violation. We also wish to keep this set X of attributes as small as possible.

A naive way to find core implicants for CFDs is by treating them as standard FDs and ignoring pattern tuples, which obviously does not necessarily give us the smallest core implicant sets. Let Σ_{FD} be a set of all FDs $X \to A$, where $(X \to A, t_p)$ is a CFD in Σ, and $CorImp_{\Sigma_{FD}}(A)$, $MinCorImp_{\Sigma_{FD}}(A)$ represent the set of all core and all minimum core implicants for an attribute A with respect to FDs in Σ_{FD}. Then

DEFINITION 13. *For every attribute A in $sort(R)$, a* CFD core implicant *CI is defined to be a set in $CorImp_{\Sigma_{FD}}(A)$, and a* minimum CFD core implicant *for an attribute A is a set in $MinCorImp_{\Sigma_{FD}}(A)$.*

The following lemma shows a property of CFD core implicants, as defined above, that helps us resolve new violations by changing the value in a limitted number of positions into fresh variables.

LEMMA 14. *Let t be a tuple in V-instance I_V, such that for some attribute A and every attribute B in a minimum CFD core implicant CI of A, we have $I_V(t, B) = v_i$ for some variable $v_i \in \mathbf{V}$. Then for every CFD $(Y \to C, t_p) \in \Sigma$ and tuple t' in I_V such that t, t' violate $(Y \to C, t_p)$, we have $YC \cap (A \cup CI) = \emptyset$.*

Proof. Let D be an attribute in the intersection of YC and $(A \cup CI)$. If $D \in Y$, then $I_V(t, D)$ contains a fresh variable and cannot be equal to $I_V(t', D)$ $(t \neq t')$ or $t_p[D]$. Therefore, the violated CFD must be of the form $(Y \to D, t_p)$. Then $(A \cup CI)$ should contain an attribute $B \in Y$ (Proposition 10). Since $I_V(t, B)$ contains a fresh variable, again, $I_V(t, B)$ cannot be equal to $I_V(t', B)$ $(t \neq t')$ or $t_p[B]$. Therefore, t, t' cannot violate $(Y \to C, t_p)$ unless the intersection is empty. □

This lemma is the key to showing that running step 0 (Figure 4) followed by FINDVREPAIR (Figure 3) produces a V-repair whose distance to the original inconsistent instance is within a constant factor of an optimum repair distance.

THEOREM 15. *For every set of CFDs,* STEP 0 + FINDVREPAIR *is a polynomial-time constant-factor approximation algorithm for* OPTIMUM REPAIR.

Proof. Follows from Lemma 14 and an argument similar to the proof of Theorem 12. □

6. CONCLUSIONS

Databases often tend to be inconsistent in the sense of violating the set of integrity constraints they are supposed to satisfy. This is typically the case with data obtained as a result of integration of multiple data sources. There has been considerable recent interest in dealing with inconsistent databases. An inconsistent database can be repaired, i.e., turned into a consistent database, by updates in the form of tuple insertion/deletions and/or value modifications. Prior art in dealing with inconsistent databases has focused on one of two things. The consistent query answering approach produces answers to queries that are true in all minimal repairs, by means of query rewriting or finding a nucleus. In the second approach, researchers have studied the hardness of finding minimum repairs, and in some cases proposed heuristic solutions. Being minimum is w.r.t. distance measures that have been proposed to gauge how far away the repair is from the original inconsistent database.

In this paper, we focused on repairs w.r.t. functional dependencies and proposed a model of repair distance which is based on the number of value modifications and the weight/confidence associated with the tuples being modified. This model is simpler than the one studied previously [10, 13, 8] in that all value modifications are treated alike. Yet we showed that under this model, repair checking is NP-complete, even when the functional dependencies are unary. We also showed that if FDs are part of the input, finding a constant-factor approximation for an optimum repair, i.e., a repair whose distance from the original database is minimum, is NP-hard. Furthermore, for a fixed set of functional dependencies, finding a $(1 + \varepsilon)$-approximation for an optimum repair is NP-hard, for some $\varepsilon > 0$. On the positive side, we developed a polynomial time algorithm that produces a constant-factor approximation for an optimum repair, when the set of functional dependencies is fixed. We also showed that our constant-factor approximation algorithm can be easily extended for repairs w.r.t. conditional functional dependencies. All our results are couched in terms of a notion of V-rapairs we proposed in this paper. V-repairs are based on the idea of changing values to other constants or to fresh variables.

Several questions remain open. In practice, a DBA or data owner may have some domain knowledge which enables her to prefer certain (optimum) repairs over others. E.g., in Figure 1(a), the data owner might know: $t_3[reg]$ cannot be 'Van', whereas $t_4[cnt]$ must be 'CAN' and $t_4[reg]$ = 'Vic' is suspect. It would be nice to formalize such preferences coming from user's background knowledge, and investigate whether we can find constant-factor approximations to optimum repairs satisfying user preferences.

Our extension to the approximation algorithm to handle conditional functional dependencies is not tight. By carefully reflecting pattern tuples in computing core implicants, we may be able to improve the constant factor of the approximation. Other interesting questions are as follows. Can the approximation algorithm be extended to work with more sophisticated distance measures, such as the cost model of Fan et al. [13]? What can we say about finding (approximate) optimum repairs w.r.t. functional and inclusion dependencies? We believe that the same approximation framework can at least be extended for equality-generating dependencies. Our ongoing work addresses some of these problems.

Acknowledgments. The authors would like to thank the anonymous reviewers for their helpful comments. This work was supported by a grant from NSERC Canada, which is gratefully acknowledged.

7. REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] Serge Abiteboul, Paris C. Kanellakis, and Gösta Grahne. On the representation and querying of sets of possible worlds. *Theor. Comput. Sci.*, 78(1):158–187, 1991.

[3] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79, 1999.

[4] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Answer sets for consistent query answering in inconsistent databases. *TPLP*, 3(4-5):393–424, 2003.

[5] Marcelo Arenas, Leopoldo E. Bertossi, Jan Chomicki, Xin He, Vijay Raghavan, and Jeremy Spinrad. Scalar aggregation in inconsistent databases. *Theor. Comput. Sci.*, 3(296):405–434, 2003.

[6] Mihir Bellare, Shafi Goldwasser, Carsten Lund, and A. Russeli. Efficient probabilistically checkable proofs and applications to approximations. In *STOC*, pages 294–304, 1993.

[7] Leopoldo E. Bertossi. Consistent query answering in databases. *SIGMOD Record*, 35(2):68–76, 2006.

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

[9] Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. Conditional functional dependencies for data cleaning. In *ICDE*, pages 746–755, 2007.

[10] Philip Bohannon, Michael Flaster, Wenfei Fan, and Rajeev Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD Conference*, pages 143–154, 2005.

[11] Jan Chomicki. Consistent query answering: Five easy pieces. In *ICDT*, pages 1–17, 2007.

[12] Jan Chomicki and Jerzy Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput.*, 197(1-2):90–121, 2005.

[13] Gao Cong, Wenfei Fan, Floris Geerts, Xibei Jia, and Shuai Ma. Improving data quality: Consistency and accuracy. In *VLDB*, pages 315–326, 2007.

[14] Dorit S. Hochbaum (Editor). *Approximation Algorithms for NP-Hard Problems*. PWS, 1997.

[15] Wenfei Fan. Dependencies revisited for improving data quality. In *PODS*, pages 159–170, 2008.

[16] Sergio Flesca, Filippo Furfaro, and Francesco Parisi. Consistent query answers on numerical databases under aggregate constraints. In *DBPL*, pages 279–294, 2005.

[17] Enrico Franconi, Antonio Laureti Palma, Nicola Leone, Simona Perri, and Francesco Scarcello. Census data repair: a challenging application of disjunctive logic programming. In *LPAR*, pages 561–578, 2001.

[18] Filippo Furfaro, Sergio Greco, and Cristian Molinaro. A three-valued semantics for querying and repairing inconsistent databases. *Ann. Math. Artif. Intell.*, 51(2-4):167–193, 2007.

[19] Ariel Fuxman, Elham Fazli, and Renée J. Miller. Conquer: Efficient management of inconsistent databases. In *SIGMOD Conference*, pages 155–166, 2005.

[20] Ariel Fuxman and Renée J. Miller. First-order query rewriting for inconsistent databases. *J. Comput. Syst. Sci.*, 73(4):610–635, 2007.

[21] Solmaz Kolahi and Leonid Libkin. On redundancy vs dependency preservation in normalization: an information-theoretic study of 3nf. In *PODS*, pages 114–123, 2006.

[22] Gabriel M. Kuper, Leonid Libkin, and Jan Paredaens, editors. *Constraint Databases*. Springer, 2000.

[23] Andrei Lopatenko and Loreto Bravo. Efficient approximation algorithms for repairing inconsistent databases. In *ICDE*, pages 216–225, 2007.

[24] Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. Syst. Sci.*, 43(3):425–440, 1991.

[25] Vijay V. Vazirani. *Approximation Algorithms*. Springer, 2003.

[26] Jef Wijsen. Condensed representation of database repairs for consistent query answering. In *ICDT*, pages 375–390, 2003.

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

[28] Jef Wijsen. Project-join-repair: An approach to consistent query answering under functional dependencies. In *FQAS*, pages 1–12, 2006.