Sampling from repairs of Denial Constraints

Hemant Saxena
University of Waterloo
h2saxena@uwaterloo.ca

ABSTRACT

Integrity constraint violations in databases have become a major problem, primarily caused as a side affect of data integration or Web data extraction from varied sources. Resolving these violations is challenging for various reasons, primarily because of exponential number of possible repairs for a given violation. Some of the previous data cleaning approaches have tried to generate a single most optimal repair, or a set of possible repairs, but they focused on the violations caused by certain specific constraints like, functional dependencies, conditional functional dependencies, and matching dependencies. In this paper we extend the idea of generating multiple repairs for a general set of constraints defined in form of denial constraints. When we are generating multiple repairs for a violation, it is important to provide the repairs which have minimal number of changes. Given that the problem of generating minimal repairs for denial constraint violations is MaxSNP-Hard, we propose a constrained randomized algorithm that is capable of generating sample repairs from a defined space of possible repairs. We have used the idea of repair expressions, used in holistic data cleaning of denial constraint violations. Our algorithm ensures two things: first, all the repairs for a single repair expression are minimal in some sense and second, it generates all possible repair expressions such that, their repairs avoid any new violations.

1. INTRODUCTION

It has been observed that data stored in databases is not always trustworthy [18]. Data quality experts estimate that as much as 10% to 20% of the total system implementation budget of a company can be wasted due to the dirty tuples in the database. To some US businesses it costs 600 billion dollars each year [11]. With the increasing scale of data, the problem of maintaining data quality is becoming more challenging: with more number of sources, data integration becomes a weak point and can potentially introduce duplicate values, integrity constraint violating tuples and missing values. Therefore, to maintain an state-of-the-art data warehouse it becomes essential to come up with a data cleaning system. A typical data cleaning system is responsible of two major tasks, detecting the dirty data and correcting them in the data. Numerous data cleaning approaches have aimed at defining the data quality rules and data cleaning algorithms for those rules (e.g. [10, 12, 16, 8].

Some of the common ways of expressing data quality rules are Functional dependencies (FDs), Conditional functional dependencies (CFDs) and Denial constraints (DCs) [8]. In majority of the literature FDs and CFDs have been considered as constraints encoding semantics, and any FD and CFD violations indicate a deviation in the database semantics, hence presence of dirty data. The idea of Denial constraints was introduced by Xu et. al. [9] and is fairly new. Denial constraints are same as universally quantified first order logic forms, and hence are capable of expressing varied type of constraints which can involve predicates like "*greater-than*" or ""not-equal-to" also, including FDs and CFDs. Most of the previous data cleaning algorithms [10, 12, 16] considered the integrity constraints in isolation while generating the repairs. Later work by Xu et. al.[8] proposed a holistic data cleaning approach, that is capable of repairing all violations together using a single objective function. Consider the example database instance given in Table 1 and the following Constraints:

$$\text{c1} : \neg(t_i.A = t_j.A \wedge t_i.C \neq t_j.C)$$

$$\text{c2} : \neg(t_i.B = t_j.B \wedge t_i.C \neq t_j.C)$$

$$\text{c3} : \neg(t_i.D = 5 \wedge t_i.C \neq 5)$$

$$\text{c4} : \neg(t_i.C < t_i.D)$$

A holistic data cleaning approach can generate the following repair: $t_1.C = t_2.C = t_3.C = t_4.C = 5$, also shown in Table 2. However, there exist numerous possible repairs for one instance of unclean database. For example, Table 2 shows two possible repairs of data instance in Table 1. A cell with value "V" can be modified to several values in order to satisfy the DC.

In this paper, we present an approach to data repair, that is capable of generating various possible repairs from a defined set of repair space. One should note that number of possible repairs are exponential for a given number of cells in the data instance. Some "unnecessary" cells can be changed which do not take part in the violation and can still be a candidate repair. For example, in the repairs shown in Table 2 one can also change cell $t_4.A$ to 4, and it will be valid repair.

	A	B	C	D
t_1	1	2	1	3
t_2	1	1	2	1
t_3	2	1	3	1
t_4	3	1	3	5

Table 1: Sample Data.

	A	B	C	D
t_1	1	2	5	3
t_2	1	1	5	1
t_3	2	1	5	1
t_4	3	1	5	5

	A	B	C	D
t_1	1	2	3	3
t_2	1	1	3	1
t_3	2	1	3	1
t_4	3	1	3	$V \leq 3$

Table 2: Two possible repairs.

The main challenge while generating a sample repair from the space of possible repairs is to generate meaningfull repairs, repairs which do not involve "unnecessary" changes. We will now show some examples which motivate the approach of sampling from a suitable set of repairs.

1.1 Motivating Examples

The previous data repair approaches can be majorly categorized into frameworks. One, stems on the idea of generating single, near-optimal repair, in terms of number of deletions or attribute modifications [16] [5]. For example, second repair in Table 2 will be optimal as it has fewer changes. Second approach is of consistent query answering, that computes answers to a selected classes of queries that seems to be valid in every suitable repair [1, 7, 13, 19].

The following examples show that the existing approaches are not suitable for them, and hence motivates a sampling approach described in the paper [3]:

Interactive Data Cleaning - An interactive data repair is a process where a sequence of repairs are suggested to the user as a guide. User may change some values according to one suggestion and demands more suggestions for other unclean attributes. For example, a user may repair tuple t_4 according to repair 1 in Table 2 and tuple t_2 according to repair 2. Therefore, the application requires several suggested repairs from a suitable space of repairs. Hence, generating a single repair will not help in this case, A consistent query approach will also not work here because, an interactive approach is not associated with a particular query.

Data Integration - Data integration is another scenario where user-defined constraints are required to control the data modification during integration process. Data expert often have a prior knowledge about the sanity of the data resources and can dictate rules to trust some sources over others. Although previous approaches [5, 16] assign weights to the sources depending upon their trustworthiness and penalize changing a trusted data source. They still cannot totally prevent the modification of trusted data.

Uncertain Query Answering - The notion of consistent query answering can be generalized to probabilistic query answering. Problem of generating all repairs is intractable, but a meaningful subset of possible repairs can still be computed and will be sufficient to allow probabilistic query answering. Again, computing a single repair or a consistent query answer is not sufficient for this application. The Monte Carlo Database (MCDB) [15] is one such example.

1.2 Challenges

Major challenges in sampling from repairs of denial constraints are:

- A repair expression used in holistic approach [8] has exponential number of solutions. Therefore it is important to define a meaningful space of candidate repairs. A meaningful repair is one which involves minimal number of changes with respect to the original database. The task of finding a repair with minimal changes for DCs and numerical values only is known to be *MaxSNP*-hard problem [2]. Therefore, an approximation algorithm is needed which can compute nearly-optimal repairs within a sufficiently large subset of repairs.

- The present holistic data cleaning approach presented in [8] uses a minimum vertex cover idea which is capable of generating nearly-optimal repairs, but does not guarantee to cover all the cardinality-minimal-repairs [3]. Consider the data instance given in Table 1, the holistic data cleaning approach uses minimum vertex cover and finds out $t_1.C = t_2.C = t_3.C = t_4.C = 5$ as the solution, but repair 2 in Table 2 shows that a there exist a repair which has fewer number of changes. Hence, holistic approach can miss out some cardinality-minimal repairs.

- The previous work on sampling repairs [3] considered FDs and CFDs. The idea of building equivalence classes is not possible for DCs, because DCs are more expressive and for predicates like "*greater-than*" and "*less-than*" equivalence classes cannot be defined.

1.3 Contributions

Our main contribution in this paper are:

- We first show that the minimality definitions defined in [4] also makes sense with respect to Denial-Constraints. For a given Instance I and a set $\sum$ of DCs, the space of possible repairs can be divided into Set-Minimal, Cardinality-Set-Minimal and Cardinality-Minimal repairs.

- We prove that the holistic data cleaning approach described in [8] does not guarantee to generate Cardinality-Minimal repair and can also generate repairs which are Not-Set-Minimal.

- We give an algorithm based on randomized selection of cells as compared to Minimum-vertex-cover selection in [8]. We prove that our algorithm is capable of generating repairs from Cardinality-Minimal space and also, it restricts all the sample repairs to be within the Set-Minimal space.

The remainder of the paper is organized as follows. In Section 2 we describe the notations used in the paper. In Section 3 we reason out why previous approaches do not fit for our task and also provide an overview of our algorithm with help of an example. In Section 4 we provide a sketch of our algorithm along with the details. A short experimental

study is provided in Section 5. We provide the related work in Section 6 and conclude with some ideas for future work in Section 7.

2. NOTATION AND DEFINITIONS

Consider a database scheme of the form $S = (U, R, B)$, where U is a set of database domains, R is a set of database predicates or relations and B is a set of finite built-in predicates. For example, in this paper, $B = \{=, <, >, \neq, \leq, \geq, \approx\}$. Let $Dom(A)$ be the domain of an attribute $A \in U$. We denote by I and instance of S consisting of n tuples. Let t be a single tuple from the instance I, we denote $t[A]$ or $I(t[A])$ the value of the cell of the tuple t under attribute A.

The integrity constraints in this paper are identified by *denial-constraints* over relational databases. Denial constraints are first-order formulae of the form $\varphi = \forall \overline{x} \neg(R_1(\overline{x_1}) \wedge ... \wedge R_n(\overline{x_n}) \wedge P_1 \wedge ... \wedge P_m)$, where $R_i \in R$ is a relation atom, and $\overline{x} = \cup \overline{x_i}$, and each P_i of the form $v_i \theta c$ or $v_i \theta v_j$, where $v_i, v_j \in \overline{x}$, c is a constant, and $\theta \in B$. Similarity predicate $\approx$ is possitive when the edit distance between two strings is above a user-defined threshold δ. Note that sungle-tuple constraints, Functional Dependencies, Matching Dependencies and Conditional Functional Dependencies are special cases of unary and binary denial constraints with equality and similarity predicates.

For a given database instance I of schema S and a DC φ, if I satisfies φ, we write $I \models \varphi$. For a given set of DCs $\sum$, we say $I \models \sum$ if and only if $\forall \varphi \in \sum, I \models \varphi$. A repair I' of an inconsistent instance I is an instance that satisfies $\sum$ and has the same set of tuple identifiers in I. The values of attributes in I' can be different from I. Also, for attributes with infinite domains, there can be infinte number of possible repairs. In cases of infinite domain attributes we represent the repairs using a variable that can accomodate *fresh-value* for its domain. A fresh value variable can have value from $Dom(A)/Dom^a(A)$, where $Dom^a(A)$ is the domain of the values for A which satisfy at least a predicate for each denial constraint involving *fresh-value*.

Number of possible ways to get a repair I' from I are infinite. Therefore, when sampling a repair I' for an inconsistent instance I it is crucial to filter the repairs that have minimal number of modifications. A widely used criterion is the *minimality of changes*. In literature [5, 16, 7, 14, 3], three main terms used to define minimality of changes are: *Cardinality-Minimal Repair, Cardinality-Set-Minimal Repair* and *Set-Minimal Repair*. In [4] these terms have been defined in context of FDs and CFDs. We will show with the help of an example that these minimality definitions still hold in cases of Denial constraints with predicates in the set $B = \{=, <, >, \neq, \leq, \geq, \approx\}$. Consider a database instance I in Table 3 and the following DC: $\varphi_1 : \forall t_i, t_j \in R, \neg(t_i.A = t_j.A \wedge t_i.B > t_j.B \wedge t_i.C < t_j.C)$. Figure 1 shows sample repairs and the repair space to which they belong. Repair $I1$ in Fig. 1 is a cardinality-minimal repair because it has clearly the minimum number of changes to repair I, and is also a Cardinality-Set-Minimal and Set-Minimal repair, [4] shows that Cardinality-Minimal is a subset of Cardinality-Set-Minimal space and Cardinality-Set-Minimal is a subset of Set-Minimal space. Repair $I2$ is Not Cardinality-Minimal because it has more changes than $I1$, but it is still a Cardinality-SetMinimal and Set-Minimal. Repair $I3$ is Not Cardinality-Set-Minimal and neither a Cardinality-Minimal repair, because the value of cell $t_2[C]$ can be changed back to

	A	B	C
t_1	1	8	9
t_2	1	3	5
t_3	1	4	2
t_4	1	5	4

Table 3: Inconsistent Instance I and DC $\varphi_1 : \forall t_i, t_j \in R, \neg(t_i.A = t_j.A \wedge t_i.B > t_j.B \wedge t_i.C < t_j.C)$

I1

	A	B	C
t_1	1	8	9
t_2	1	3	1
t_3	1	4	2
t_4	1	5	4

Set-Minimal
Cardinality-Set-Minimal
Cardinality-Minimal

I2

	A	B	C
t_1	1	8	9
t_2	1	3	5
t_3	1	4	7
t_4	1	5	8

Set-Minimal
Cardinality-Set-Minimal
Not Cardinality-Minimal

I3

	A	B	C
t_1	1	8	9
t_2	1	3	3
t_3	1	4	4
t_4	1	5	5

Set-Minimal
Not Cardinality-Set-Minimal
Not Cardinality-Minimal

I4

	A	B	C
t_1	1	8	9
t_2	2	6	7
t_3	1	4	2
t_4	1	5	4

Not Set-Minimal
Not Cardinality-Set-Minimal
Not Cardinality-Minimal

Figure 1: Sample repairs from different repair space.

5 and value of cell $t_3[C]$ and $t_4[C]$ can be changed to something else, say 6 and 7 respectively, and the repair still stays valid. Repair $I4$ is Not Set-Minimal, because the values of cell $t_2[B]$ and $t_2[C]$ can be reverted to their original values 3 and 5 respectively, and the resulting instance will still be a valid repair of I.

3. SOLUTION OVERVIEW

Finding minimal repairs has been shown to be NP-Complete even for FDs only [6]. Moreover, finding minimal repair for DCs and numerical values only is known to be MaxSNP-hard [2]. The work in [8] proposes an approximate holistic algorithm that is capable of generating nearly-optimal repairs for a given set of DCs. The algorithm we propose for generating sample repairs for DCs is built upon two ideas: first, we try to generate a repair in a holistic manner as proposed in [8], Second, we uses the idea of generating random samples from the space of repairs, introduced in [3]. Before going into the algorithm specifications, we will first see why the two of the above ideas cannot solve the problem alone.

3.1 Previous Approaches

First we will see why the repair sampling approach described in [3] is not self-sufficient to generate repairs for Denial-Constraints as well. The idea in [3] is to grow the set of clean cells by merging the cells belonging to same equivalence class. It is easy to build equivalence classes in case of FDs and CFDs, in other words when the set of predicates $B = \{=, \neq\}$; but the equivalence class cannot be determined when the predicates are more loose, such as from the set $B = \{=, \neq <, >, \leq, \geq, \approx\}$.

Second, the approximate holistic algorithm in [8], has two issues due to which it cannot be used for generating repairs.

	A	B	C	D
t_1	3	5	6	2
t_2	3	5	4	2
t_3	3	5	7	6

{A -> B
D -> A
B < C
C > D}

Figure 2: Unclean database instance and a set of DCs. Dotted line is a hyperedge, showing violation.

One, the approximate algorithm uses minimum vertex cover (MVC) to find an approximately minimal repair, but it still does not guarantee to cover all cardinality-Minimal space. For example, consider the data instance given in Table 1, the MVC for the conflict hyper-edges is set $\{t_1.C, t_2.C, t_4.C\}$. The holistic algorithm will pick one cell from MVC, say $t_2.C$ and form a frontier set and generates a repair expression for it. The output repair from this will be the first repair in Table 2, i.e. $t_1.C = t_2.C = t_3.C = t_4.C = 5$. Although the cardinality-minimal repair is the second repair in Table 2, i.e. $t_1.C = t_2.C = 3, t_4.D = V \leq 3$. This cardinality-Minimal repair could have been achieved if the algorithm would have picked up cell $t_4.D$ first (that is not in MVC) and then $t_2.C$ to generate repair, we will show this in detail in Section 3.2.

The other issue with the holistic approach is that, it can also generate repairs which are not even Set-Minimal. The cell modifications governed by the repair expression can generate new violations which are handled in the next iteration of the algorithm. We will see an example where a series of new violations can end up generating a Not-Set-Minimal repair. Consider a database instance and a set of DCs as shown in figure 2. The dotted box represents a violation hyperedge of a conflict hypergraph (CH). The MVC for a single hyperedge CH can be any cell in that hyperedge. Say, algorithm selects cell $t_2.B$ as MVC and generates a repair, as shown in Figure 3(i), but introduces a new violation. In the next iteration cell $t_2.A$ is a part of MVC and Figure 3(ii) is the repair with a new violation, and so on, Figure 3(i-iv) shows the repair and new violations introduced at each step of the iteration. Figure 4 shows all the cells changed in the repair process. Now observe that the value of cell $t_2.B$ can be reverted back to its original value 5, and the resulting repair is shown in Figure 5 which is still a valid repair. This proves that the holistic data cleaning algorithm can generate repairs which are not Set-Minimal.

3.2 Algorithm Overview

The algorithm we propose is inspired from the holistic data cleaning algorithm of [8], with two major modifications. One, we introduce randomness in selecting an hyperedge and a cell in an hyperedge, where as [8] used minimum vertex cover to select cells to start the repair. The random selection of cells help us to generate Cardinality-Minimal repairs as well. Second, we have put restrictions on which cells can be included in the frontier set. For each of the randomly picked cell, we do one step look-ahead and check if the cell can produce further violations. If yes, we mark that cell "unsafe" and choose another cell from the edge. In Section 3.1 we saw that if the a repair is introducing new violations then we can get into a sequence of violations that can result into a Not-Set-Minimal repair. Therefore, the insight behind the constrained selection of cells is to avoid a sequence of

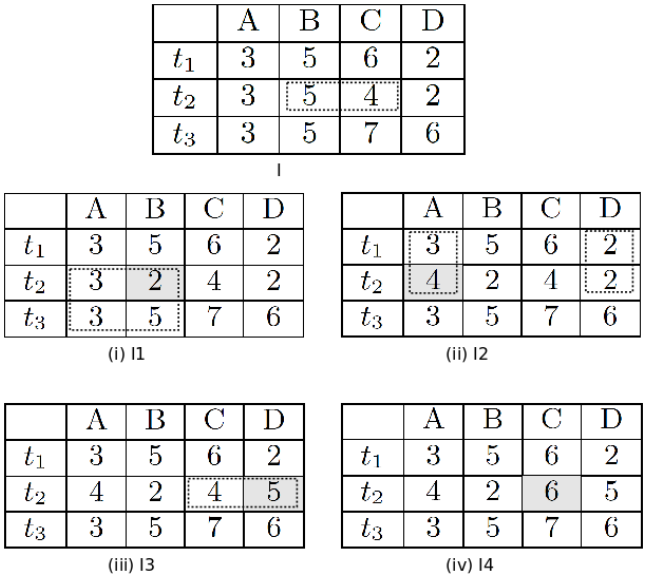

Figure 3: Shows the repaired cell at each iteration and the new violation.

	A	B	C	D
t_1	3	5	6	2
t_2	4	2	6	5
t_3	3	5	7	6

Figure 4: All the cell changed with respect to original I.

	A	B	C	D
t_1	3	5	6	2
t_2	4	5	6	5
t_3	3	5	7	6

Figure 5: Value of Cell $t_2.B$ can be reverted and the repair is still valid.

repairs which can end up in a Not-Set-Minimal repair.

We represent the violations in the database instance using hyperedges, similar to [8, 16]. After all the violations have been detected we get a set of hyperedges, and the graph induced by them is called a *Conflict Hypergraph (CH)*. It is an undirected hypergraph with a set of nodes P representing the cells and a set of annotated hyperedges E representing the relationships among cells violating a constraint. More precisely, a *hyperedge* is a set of violating cells from which one of them must change to repair the constraint, and contains: (a) the constraint c, which induced the conflict on the cells; (b) the list of nodes involved in the conflict.

Consider the example database instance given in Table 1 and the following set of DCs $\{A \rightarrow C, B \rightarrow C, R[D = 5] \rightarrow R[C = 5], C \geq D\}$. Figure 6 represents a conflict hypergraph of the present state of the database violations w.r.t the given DCs. Each hyperedge in the graph contains only the violating cells, and in order to repair it, at least one of its cells must get a new value. We start our algorithm by picking an hyperedge at random and a random cell within that edge. Algorithm assumes that this cell needs to be changed and hence identify all the other cells that are involved in the repair of this cell. This starting set of newly identified cells is called *frontier*. We call *repair expressions* the list of constant assignments and constraints among the frontier. Similar to [8], the frontier and the repair expressions form a *Repair Context* (RC). Once we are done with the repair context of the first chosen cell, we *freeze* all the edges which got repaired using that RC. By freeze, we mean that the cells belonging to those edges, if encountered in the frontier of any other RC later, cannot be changed. If there are more edges remaining in the hypergraph, our algorithm again chooses a remaining hyperedge at random and chooses a cell that is not marked *freeze* from that edge, and forms the RC for this new cell in the similar fashion. The algorithm continues till there are no more hyperedges left in the graph.

For example, consider the DCs and hypergraph in Figure 6. Lets say, the algorithm starts by choosing Edge $e4$ and cell $t_4[D]$. The repair expression for this is simple, i.e. $t_4[D] \leq 3$, so we will assign a fresh value to it, call it FV and cell $t_4[C]$ will be marked *freeze*. Then we move onto choose the next edge, say this time we choose edge $e2$ and cell $t_2[C]$. If we have to change $t_2[C]$ then the cell $t_3[C]$ in edge $e2$ should be changed, this gives us the expression $t_2[C] = t_3[C]$. Cell $t_2[C]$ is related to edges $e1$ as well, and we get the expression $t_2[C] = t_1[C]$. Cell $t_2[C]$ is related to edges $e4$ as well, and we get the expression $t_2[C] = t_4[C] = 3$, remember that cell $t_4[C]$ is frozen and cannot be changed. The cell $t_1[C]$ is in edge $e5$ as well, hence the relation for cell $t_1[C]$ will be $t_1[C] \geq 3$. Therefore the overall repair expression becomes $t_2[C] = t_1[C], t_2[C] = t_3[C], t_2[C] = t_4[C], t_4[C] = 3$and $t_1[C] \geq 3$. The cells and their new values will be: $t_2[C] = 3, t_1[C] = 3$ and $t_4[D] = FV$.

Notice that, the repair we obtained above is same as the second repair shown in Table 2, which is actually a Cardinality-Minimal repair. This indicates that the randomized selection of cells is capable of generating Cardinality-Minimal repairs also.

Proposition: Randomized selection of cells guarantee to cover Cardinality-Minimal repairs also.

Proof Sketch: There are exponential number of repair expressions possible in a single CH depending upon the cell we start with. Also, each repair expression has different satisfiability with different cardinality. By using random selection we are allowing all the possible repair expressions, some of which will be Cardinality-Minimal.

	A	B	C	D
t_1	1	2	1	3
t_2	1	1	2	1
t_3	2	1	3	1
t_4	3	1	3	5

e1
e2
e3
e4
e5

{A -> C, B -> C, C >=D, R[D=5] < R[C=5]}

Figure 6: Conflict hypergraph.

Now we will see an example where we demonstrate a constrained selection of cells to limit the space of repairs to Set-Minimal. Consider the unclean database instance in Fig. 2, cell $(t_2[B], t_2[C])$ forms the only edge in the CH. We can either choose cell $t_2[B]$ or $t_2[C]$, notice that choosing $t_2[B]$ resulted into a new violation, see Fig. 3(i). Hence it is called an "unsafe" cell and will not be chosen. To check if the cell is "unsafe" we use a one step look-ahead, i.e. we assume that the cell has been modified according to the DETERMINATION step and construct a new conflict hypergraph, if that cell introduces a new hyperedge then we mark it "unsafe". Whereas, in the same example if we select cell $t_2[C]$, the result of DETERMINATION step will be $t_2[C] > 5$ and it will not create a new violation. This observation leads to the following proposition.

Proposition: A repair context that does not introduce new violations will provide a Set-Minimal repair.

Proof Sketch: If DETERMINATION step is using a Cardinality-Minimality or Distance-Minimality methods to satisfy the expression, then all the cell changes are minimal and necessary. Therefore, if we avoid any new violations then the cell changes suggested by the DETERMINATION step are the necessary changes. Hence, the repair will be Set-Minimal.

4. ALGORITHM

This section will give the details of our algorithms. Our algorithm consists of two main algorithms, first algorithm guarantees that the repair for a given repair expression changes minimal number of cells, i.e. it will be cardinality-minimal. Second algorithm generates all possible repair expressions, such that their repairs do not introduce new violations. Together, they guarantee a sample repair within a Set-Minimal space. For ease of understanding we give details of the second algorithm first.

4.1 Repair expression generation

Given a database and a set of DCs, we use Algorithm 1 to carefully generate all possible repair expressions that do not introduce new violations. It starts by computing the violations and constructing a conflict hypergraph. The outer loop (lines 3-20) traverse through all the hyperedges of the

CH. For each hyperedge we select one of its "non-frozen" cell (line 5), and construct a repair context and get assignments for it (lines 6-12). After receiving the assignments we do a look-ahead and check if the assignment is safe and will not cause new violations (line 13). If it is safe we go ahead and apply the updates and mark the cells of the updated edges as "frozen" (lines 14-16). If it is not safe we choose another vertex of the current hyperedge, and repeat the process. The LOOKUP and DETERMINATION procedures are same as described in [8]. However, we have used a new procedure called, $isSafeAssignment()$ to do a one step look-ahead and determine if the assignment will introduce new violations or not. Algorithm 2 shows the details of it. We basically apply the assignments on a copy $data'$ of the actual data and re-create a CH and check for any new hyperedge.

Algorithm 1 Holistic Repair

Require: Database Data, Set of Denial Constraints DCs
Ensure: Repaired Data
1: Compute violations, Conflict Hypergraph (CH)
2: allEdges ← $CH(Edges)$
3: **while** allEdges is not Empty **do**
4: currEdge ← Get one random edge from allEdges
5: **while** exists (currCell ← Get one cell from currEdge) AND currCell.freeze = FALSE **do**
6: rc ← Initialize a new repair context for currCell
7: edges ← Get all hyperedges for that cell
8: **while** edges is not empty **do**
9: edge ← Get an edge from edges
10: $LOOKUP(cells, edges, rc)$
11: **end while**
12: assignments ← $DETERMINATION(cells, exps)$
13: **if** isSafeAssignment(assignments) **then**
14: data.updates(assignments)
15: allEdges ← allEdges − assignments.edges
16: $\cup assignment.edges.cells.freeze \leftarrow TRUE$
17: Break
18: **end if**
19: **end while**
20: **end while**
21: **return** data.$PostProcess()$

Algorithm 2 isSafeAssignment

Require: Set of assignments for data
Ensure: Boolean value, TRUE if assignments do not cause new violations.
1: $data'.updates(assignments)$
2: rebuild CH for $data'$
3: **if** any new hyperedges **then**
4: **return** FALSE
5: **else**
6: **return** TRUE
7: **end if**

4.2 Repairs for Expressions

This section provide details about determining a valid and minimal assignment for a repair expression. For this part we borrow the idea of DETERMINATION algorithm in [8]. The determination algorithm provides two types of minimality when assigning values for a repair expression.

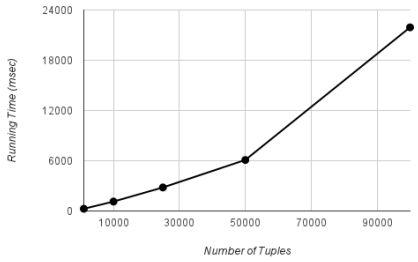

Figure 7: Running time of generating a repair.

Distance minimality. In this case we minimize the distance given by the following function:

$$\sum_{t \in I, t' \in I_r, A \in A^R} dis_A(I(t[A]), I(t'[A]))$$

We use a squared distance function in our approach. In particular, we solve the following objective function: for each cell with value v involved in a predicate of the DC, a variable x is added to the function with $(x-v)^2$. As the given objective function is quadratic, it has a positive definite matrix, and can be efficiently solved by the quadratic programing technique.

Cardinality Minimality. In this case we want (i) to minimize the number of changed cells, and (ii) to minimize the distance for those changing cells. In order to achieve cardinality minimality we gradually increase the number of cells that can be changed until the quadratic equation becomes solvable. For those variables we decide not to change, we add constraint to enforce it to be equal to original value. It can be seen that this process is exponential in the number of cells in the repair context.

We use one of the above two minimality while generating an assignment for a repair expression. Say, we always use the Distance Minimality for any expression, now from our Algorithm 1 we generate all possible expressions which do not cause new violations. Our claim is that all the possible repairs we generate are Distance Minimal with respect to their expression and none of the repairs are Not-Set-Minimal.

5. EXPERIMENTAL STUDY

In this section we present a small performance evaluation of our algorithm. The experiments were conducted on an AMD-6300 Six core machine with 16GB of RAM. All the computations were executed in memory. We have used a real world data set of hospital records from US Department of Health and Human Services. It has 19 attributes and we designed 9 FDs for it. We varied the number of tuples to test the scalability of our algorithm. Figure 7 shows the running time of generating a single repair when number of tuples in the dataset were changed from 1,000 to 100,000 tuples. We also tested the precision of the repairs generated by our algorithm, we used the same dataset with 10,000 tuples and ran the algorithm 500 times, because we get random repair at each run. Figure 8 shows the variation in precision with number of changed cells in the repair.

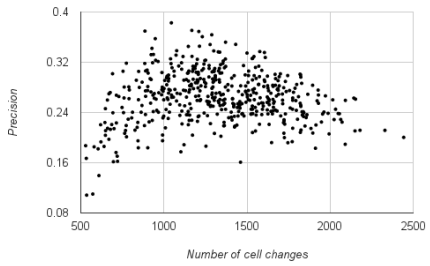

Figure 8: Precision variation with number of changed cells.

6. RELATED WORK

One of the most popular approach of obtaining the repair in the context of FDs and DCs has been to obtain a single most optimal repair possible [16, 8]. Both of these work aimed at either minimizing the number of changes or the distance of the repair with the original database. The main drawback of single-repair approaches is that there is usually no unique optimal repair. Work done by George et. al. [3] solve this problem by generating a sample of possible repairs of the input instance in context of FDs. In this work we extend the same idea of sampling repairs in the context of more general set of integrity constraints defined using Denial Constraints.

Approaches that provide consistent query answers perform query rewriting [1, 13], or construct a condensed representation of all the repairs that allows obtaining consistent answers [19]. Usually, a restricted class of queries can be answered efficiently while harder classes are answered using approximate methods (e.g., [17]). The approach of sampling repairs overcome shortcomings of consistent query answering such as returning empty query results when no common query answers are found.

In [14], the authors have imposed strong constraint on the defined FDs that, any attribute that appears in the right-hand side of an FD cannot appear in the left-hand side of another FD. These limitations have allowed to get the cardinality-minimal repairs in PTIME. Our solution uses a more generic way of expressing constraints, we use denial constraints to express simple constraints like FDs and CFDs, but it can also express constraints which have predicates like $<, > \leq, \geq$ and $\neq$.

A related topic to our work is answering queries over uncertain data. Several works have addressed the problem of modelling data uncertainty in a compact way to allow efficient query answering. One of the related works is the Monte-Carlo Database [15], which allows answering user queries efficiently using a sample of possible realizations of a database by avoiding redundant computations. For example, query planning is performed only once for the entire set of possible repairs.

7. CONCLUSION

In this paper we extended the idea of generating a sample of repairs for a varied set of constraints which are expressed using Denial Constraints. We motivated the need of multiple repairs rather than a single optimal repair. The challenge in generating a set of repairs is to filter the reasonable set of repairs out of the exponential space of all the possible repairs. We saw that the existing algorithms for generating samples from FDs and CFDs can not be extended to DCs and the holistic data repair approach is not capable of generating repairs within a reasonable space of possible repairs. Therefore, we proposed a constrained randomized algorithm that generates repairs that are within the space of Set-Minimal repairs. Our algorithm relies on a one step look ahead to restrict the sampling space to Set-Minimal repairs, as an important direction for future work we would like to come up with a more robust algorithm that can help us define the space of possible repairs. We will also consider to come up with an approach that can define the sensitivity of the cells depending upon the violations it can potentially cause, and select the less sensitive cells. A threshold can determine what cells to choose, and that will indirectly determine the space of our sample repairs. One drawback in the existing algorithm is that, in the scenario when all the cells of an hyperedge causes a new violation then the algorithm cannot generate any solution. Whereas if we know the sensitivity of the cells, we can select a less sensitive cell and continue our algorithm with some defined probability of getting a not-set-minimal repair.

8. REFERENCES

[1] M. Arenas, L. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *Proceedings of the Eighteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '99, pages 68–79, New York, NY, USA, 1999. ACM.

[2] L. Bertossi, L. Bravo, E. Franconi, and A. Lopatenko. Complexity and approximation of fixing numerical attributes in databases under integrity constraints. In G. Bierman and C. Koch, editors, *Database Programming Languages*, volume 3774 of *Lecture Notes in Computer Science*, pages 262–278. Springer Berlin Heidelberg, 2005.

[3] G. Beskales, I. F. Ilyas, and L. Golab. Sampling the repairs of functional dependency violations under hard constraints.

[4] G. Beskales, I. F. Ilyas, L. Golab, and A. Galiullin. Sampling from repairs of conditional functional dependency violations. *The VLDB Journal*, 23(1):103–128, Feb. 2014.

[5] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD '05, pages 143–154, New York, NY, USA, 2005. ACM.

[6] F. Chiang and R. J. Miller. Discovering data quality rules. *Proc. VLDB Endow.*, 1(1):1166–1177, Aug. 2008.

[7] J. Chomicki, J. Marcinkowski, and S. Staworko. Computing consistent query answers using conflict hypergraphs. In *Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management*, CIKM '04, pages 417–426, New York, NY, USA, 2004. ACM.

[8] X. Chu, I. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 458–469, April 2013.

[9] X. Chu, I. F. Ilyas, and P. Papotti. Discovering denial constraints. *Proc. VLDB Endow.*, 6(13):1498–1509, Aug. 2013.

[10] G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. Improving data quality: Consistency and accuracy. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 315–326. VLDB Endowment, 2007.

[11] W. Eckerson. Data Quality and the Bottom Line: Achieving Business Success through a Commitment to High Quality Data. Technical report, The Data Warehousing Institute, 2002.

[12] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Interaction between record matching and data repairing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 469–480, New York, NY, USA, 2011. ACM.

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

[14] S. Greco and C. Molinaro. Probabilistic query answering over inconsistent databases. *Annals of Mathematics and Artificial Intelligence*, 64(2-3):185–207, Mar. 2012.

[15] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. Jermaine, and P. J. Haas. Mcdb: A monte carlo approach to managing uncertain data. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 687–700, New York, NY, USA, 2008. ACM.

[16] S. Kolahi and L. V. S. Lakshmanan. On approximating optimum repairs for functional dependency violations. In *Proceedings of the 12th International Conference on Database Theory*, ICDT '09, pages 53–62, New York, NY, USA, 2009. ACM.

[17] A. Lopatenko and L. Bertossi. Complexity of consistent query answering in databases under cardinality-based and incremental repair semantics. In T. Schwentick and D. Suciu, editors, *Database Theory ICDT 2007*, volume 4353 of *Lecture Notes in Computer Science*, pages 179–193. Springer Berlin Heidelberg, 2006.

[18] T. C. Redman. The impact of poor data quality on the typical enterprise. *Commun. ACM*, 41(2):79–82, Feb. 1998.

[19] J. Wijsen. Condensed representation of database repairs for consistent query answering. In D. Calvanese, M. Lenzerini, and R. Motwani, editors, *Database Theory ICDT 2003*, volume 2572 of *Lecture Notes in Computer Science*, pages 378–393. Springer Berlin Heidelberg, 2003.