Sampling from repairs of Denial Constraints

Hemant Saxena
University of Waterloo
h2saxena@uwaterloo.ca

ABSTRACT

Write abstract here.

1. INTRODUCTION

It has been observed that data stored in the database is not always trustworthy [17]. 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 has costed 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.

	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.

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: $t_1.C = t_2.C = t_3.C = t_4.C = 5$. However, there exist numerous possible repairs for one instance of unclean database. For example, Table 1 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 reapir 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 in Table 1 also change cell $t_4.A$ to 4, and it will be valid repair. The main challenge while generating a sample repair from the space of possible repair 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, described in this paper.

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 term sof number of deletions or attribute modifications [16] [5]. For example, second repair in Table 1 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, 18].

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 1 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 also will 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 weight 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 can 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 gaurantee 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 1 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] consider 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 gaurantee to generate Cardinality-Minimal repair and can also generate repairs which are Not-Set-Minimal.

- We give an efficient algorithm tha can

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*.

	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 $\varphi_1 : \forall t_i, t_j \in R, \neg(t_i.A = t_j.A \wedge t_i.B > t_j.B \wedge t_i.C < t_j.C)$

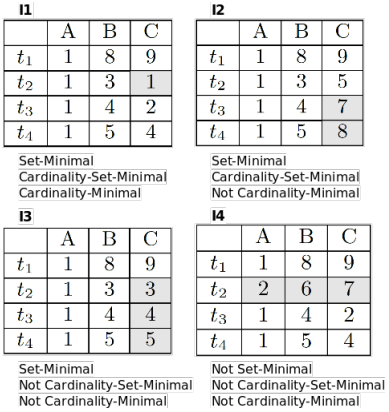

Figure 1: Sample repairs from different repair space.

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 2 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 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 repairs 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 sapmle 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 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. One, the approximate algorithm uses minimum vertex cover (MVC) to find an approximately minimal repair, but it still does not gaurantee 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 $\{t_1.C, t_2.Candt_4.C\}$. The holistic algorithm will pick one cell from MVC, say $t_2.C$ and form a frontier set and generate a repair expression for it. The output repair from this will be the first repair in Table 1, 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 1, i.e. $t_1.C = t_2.C = 3, t_4.D = V \leq 3$. This cardinality-Minimal repair could have been acheived 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 more detail in Section **??**.

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 hendled 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 chose 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 modifica-

	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.

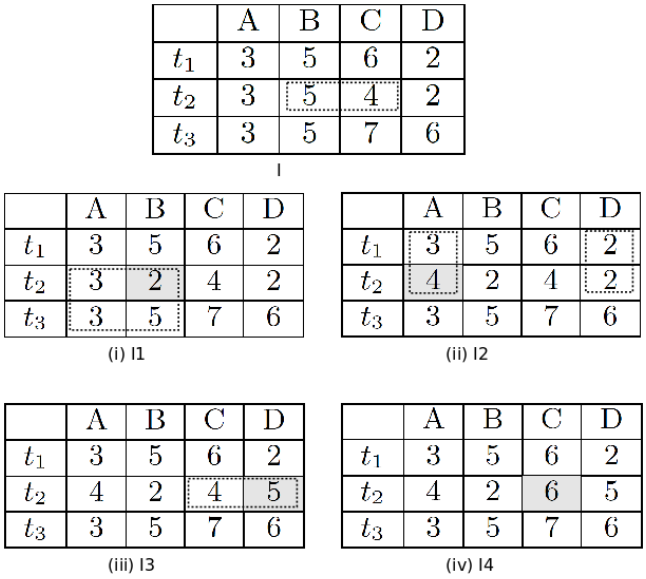

Figure 3: Shows the repaired cell at each iteration and the new violation.

	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.

tions. One, we have introduced a 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. Second, we have put constraints on which cells can be included in the frontier set. [MIGHT CHANGE —]

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 non freezed cell 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 it to a fresh value, call it FV and cell $t_4[C]$ will be freezed. Then we move onto choose the next edge, say this time we chose 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 1, which is ctually 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 gaurantee to

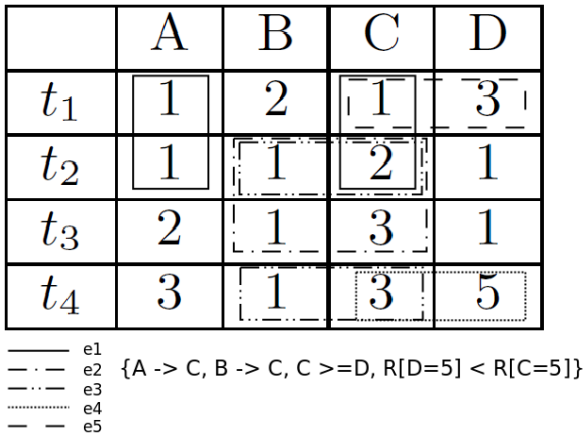

Figure 6: Conflict hypergraph.

cover Cardinality-Minimal repairs also.

Proof Sketch: There are exponential possible repair expression in a CH depending upon the cells 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.

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

[18] 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.