Complexity and Efficient Algorithms for Data Inconsistency Evaluating and Repairing

Dongjing Miao
Harbin Institute of Technology
P.O. Box 321
92 Xidazhi Street
Harbin, China
miaodongjing@hit.edu.cn

Zhipeng Cai
Georgia State University
P.O. Box 5060
Atlanta, GA, USA
zcai@gsu.edu

Jianzhong Li
Harbin Institute of Technology
P.O. Box 321
92 Xidazhi Street
Harbin, China
lijzh@hit.edu.cn

Xiangyu Gao
Harbin Institute of Technology
P.O. Box 321
92 Xidazhi Street
Harbin, China
gaoxy@hit.edu.cn

Xianmin Liu
Harbin Institute of Technology
P.O. Box 321
92 Xidazhi Street
Harbin, China
liuxianmin@hit.edu.cn

ABSTRACT

As the basic computing task of data repairing and inconsistency evaluating, optimal subset repair is not only applied for cost estimation during the progress of database repairing, but also directly used to derive the evaluation of database inconsistency. Computing an optimal subset repair is to find a minimum tuple set from an inconsistent database whose remove results in a consistent subset left. Tight bound on the complexity and efficient algorithms are still unknown. In this paper, we show the tight upper and lower bounds, together with a fast estimation on the size of optimal subset repair. We first strengthen the dichotomy for optimal subset repair computation problem, we show that it is not only `APXcomplete`, but also `NPhard` to approximate an optimal subset repair with a factor better than 17/16 for most cases. We second show a $(2-0.5^{\sigma-1})$-approximation whenever given σ functional dependencies, and a $(2-\eta_k+\frac{\eta_k}{k})$-approximation when an η_k-portion of tuples have the k-quasi-Turán property for some $k>1$. We finally show a sublinear estimator on the size of optimal S-repair for subset queries, thus deriving an estimation of FD-inconsistency degree of a ratio $2+\epsilon$. Experiment results show the efficiency of our FD-inconsistency degree estimator.

1. INTRODUCTION

A database instance I is said to be inconsistent if it violates some given integrity constraints, that is, I contains conflicts or inconsistencies. Those database inconsistencies can occur in various scenarios due to many causes. For example, a typical scenario is information integration, where data are integrated from different sources, some of them may be low-quality or imprecise, so that conflicts or inconsistencies arise.

In the principled approach managing inconsistencies [5], the notion of *repair* was first introduced decades ago. A repair of an inconsistent instance I is a consistent instance I' obtained by performing a minimal set of operations on I so as to satisfy all the given integrity constraints. Repairs could be defined under different settings of *operations* and *integrity constraints*. We follow the setting of [27], where we take functional dependencies, also a most typical one, as the integrity constraints, and deletions as the operations, so that a repair of I here is a subset of I obtained by minimal tuple deletions, and an optimal repair of I is a subset of it obtained by deleting minimum tuples. Computing an optimal subset repair with respect to functional dependencies is the major concern in this paper. The significance of study on computing optimal repair is twofold.

Computing optimal repairs would be the basic task in data cleaning and repairing. For data repairing, existing methods could be roughly categorized into two classes, fully automatic and semi automatic ways [14]. In fully automatic repairing methods, optimal subset repairs are always considered as optimization objectives [15, 32, 31]. Given an inconsistent database, one needs automated methods to make it consistent, *i.e.*, find a repair that satisfies the constraints and minimally differs from the originated input, optimal subset repair is right one of the choices [27]. On the other side, optimal subset repairs are also preferred candidates picked by automatic data cleaning or repairing system when dealing with inconsistency errors. Instead of the fully automatic way, the human-in-loop semi-automatic repairing is another prevailing way [6, 7, 18, 22], and the complement of an optimal subset repair is an ideal lower bound of repairing cost which could be used as to estimate the amount of necessary effort to eliminate all the inconsistency, sometimes even enlighten them how to choose specific operations.

Besides optimal repairs, measuring inconsistency motivates the computation on the size of optimal repairs. Intuitively, for the same schema and with the same integrity

constraints, given two databases instances, it is natural to know which one is more inconsistent than the other. This comparison can be accomplished by assigning a measure of inconsistency to a database. Hence, measuring database inconsistency has been revisited and generalized recently by data management community. [9] argued that both the admissible repair actions and how close we want stay to the instance at hand should be taken into account when defining such measure. To achieve this, database repairs [8] could be applied to define degrees of inconsistency. Among a series of numerical measurements proposed in [9], subset repair based inconsistency degree $inc\text{-}deg^S$ is the most typical one. According to [9], subset repair based inconsistency degree is defined as the ratio of minimum number of tuple deleted in order to remove all inconsistencies, *i.e.*, the size of the complement of an optimal subset repair. Therefore, computing optimal subset repair is right the fundamental of inconsistency degree computation. Previous studies does not provide fine-grained complexity on this problem and efficient algorithm for large databases. Thus, we in this paper give a careful analysis on the computational complexity and fast computation on the size of an optimal subset repair. Contributions of this paper are detailed as follows.

We first study the data complexity of optimal subset repair problem including the lower and upper bounds in order to understand how hard the problem is and how good we could achieve. The most recent work [27] develops a simplification algorithm OSRSucceeds and establishes a dichotomy based on it to figure the complexity of this problem. Simply speaking, they show that, for the space of combinations of database schemas and functional dependency sets, (i) it is polynomial to compute an optimal subset repair, if the given FD set can be simplified into an empty set; (ii) the problem is APX-complete, otherwise.

As the computation accuracy of the size of an optimal subset is very crucial to our motivation, we strengthen the dichotomy in this paper by improving the lower bound into concrete constants. Specifically, we show that it is NPhard to obtain a $(17/16-\epsilon)$-optimal subset repair for most input cases, and $(69246103/69246100-\epsilon)$–optimal subset repair for all the others. We show that a simple reduction could unify most cases and improve the low bound We then consider approximate repairing. For this long standing problem, it is always treated as a vertex cover problem equivalently, and admits the upper bound of ratio 2. However, we take a step further, show that (i) an $(2-0.5^{\sigma-1})$-approximation of an optimal subset repair could be obtained for given σ functional dependencies, more than that, (ii) it is also polynomial to find an $(2-\eta_k+\frac{\eta_k}{k})$-approximation, which is much better if an η_k-portion of tuples have the k-quasi-Turán property for some $k>2$.

Then, we turn to the most related problem, to estimate the subset repair based FD-inconsistency degree efficiently. For an integrated database instance, it is helpful to measure the inconsistency degree of any part of it locally, in order to let users know and understand well the quality of their data. Consider an inconsistent database I integrated by data from two organizations A and B, we need to know the main cause of the conflicts. If we know the inconsistency degree of some part A is very low but that of B is as high as the inconsistency degree of I, then we could conclude that the cause of inconsistencies is mainly on the conflicts in between, but not in any single source. That is when we find the inconsistency degree of some local part is approximately equal to that of the global one, then it is reasonable to take this part as a primary cause of inconsistency, so that we may focus on investigating what happens in B.

To this motivation, in this paper, we focus on fast estimating the subset-repair based FD-inconsistency degree. Concretely, it seems a same problem as computing an optimal repair itself, so that the complexity result of optimal subset repair computing indicates it is hard to be approximated within a better ratio polynomially, not to mention linear or even a sublinear running time. However, we observe that, the value of inconsistency degree is a ratio to the size of input data, say n, hence, an n-fold accuracy loss of optimal subset repair size estimating is acceptable. Based on this, we develop a sample-based method to estimate the size of an optimal subset repair with a error of $(2+\epsilon)n$ so as to break through the limitation of linear time complexity while achieving an approximation with an additive error ϵ. To support a variety of subset queries, especially for whose result is very large, we model those queries as the $\subseteq$-oracle which can answer membership-query, and return k tuples uniformly sampled whenever given a number k.

2. PROBLEM STATEMENT

The necessary definitions, notations and problem definition are formally given in this section.

Schemas and Tables. A k-ary relation schema is represented by $R(A_1,...,A_k)$, where R is the relation name and $A_1,\ldots,A_k$ are distinct attributes of R. In the following part of this paper, we refer $R(A_1,...,A_k)$ to R for simplicity. We customarily use capital letters from the beginning of the English alphabet to denote individual attribute, such as "A,B,C", and use capital letters from the end of the English alphabet individual attribute to denote a set of attributes, such as "$\mathsf{X},\mathsf{Y},\mathsf{Z}$", sometimes with subscripts. A set of attributes are conventionally written without curly braces and commas, such as X can be written as AB if $\mathsf{X}=\{A,B\}$. We assume the domain of each attribute, $dom(A_i)$, is countably infinite, then, any instance I over relation R is a collection of k-ary tuples $\{a_1,a_2,\ldots,a_k\}$, where each value a_i are taken from the set $dom(A_i)$. Let $\mathbf{t}.A_i$ refer to the value a_i on attribute A_i, and $\mathbf{t}.\mathsf{X}$ refer to the sequence of attribute values $a_1,a_2,\ldots,a_i$ when $\mathsf{X}=A_1A_2\ldots A_i$. We use $[\mathbf{t}.\mathsf{X}]$ to denote the set of all tuples from I sharing the same value of X. The size of an instance is the number of tuples in it, denoted as $|I|$. In this paper, any instance I of a relation schema R is a single table corresponding to R.

Functional Dependencies. Let X and Y be two arbitrary sets of attributes in a relation schema R, then Y is said to be functionally determined by X, written as $\mathsf{X}\rightarrow\mathsf{Y}$, if and only if each X-value in R is associated with precisely one Y-value in R. Usually X is called the *determinant* set and Y the *dependent* set, but in this paper, for the sake of simple, we just call them *determinant* and *dependent* respectively. A functional dependency $\mathsf{X}\rightarrow\mathsf{Y}$ is called *trivial* if Y is a subset of X.

Given a functional dependency $\varphi\colon \mathsf{X}\rightarrow\mathsf{Y}$ over R, any instance I corresponding to R is said to satisfy φ, denoted as $I\models\varphi$, such that for any two tuples $\mathbf{s},\mathbf{t}$ in I, $\mathbf{s}.\mathsf{Y}=\mathbf{t}.\mathsf{Y}$ if $\mathbf{s}.\mathsf{X}=\mathbf{t}.\mathsf{X}$. That is, two tuples sharing the same values of X will necessarily have the same values of Y. Otherwise, I does not satisfy φ, denoted as $I\not\models\varphi$. As a special case,

	id	name	PR	AC	PN	STR	CTY	CT	ST	zip
t_1	14	Gaudi	30.5	217	779-9956	KATY	COOK	Orland Park	IL	60462
t_2	15	Paris	29.99	217	928-9725	SUNNY	COOK	Orland Park	IL	60462
t_3	16	ART	65.99	217	898-1817	155TH	COOK	Orlad Park	IL	60462
t_4	17	Formall	18.99	217	823-0934	HARLEM	COOK	Orladn Park	IL	60462
t_5	18	Sailing	59.5	217	597-1328	HELEN	COOK	Orlad Park	IL	60462
t_6	19	Ocean	44.49	217	861-9836	HRON	COOK	Orladn Park	ILA	60462

(a) Example instance order

conflict	conflict
$\{t_1, t_3\}$	$\{t_2, t_6\}$
$\{t_1, t_4\}$	$\{t_3, t_4\}$
$\{t_1, t_5\}$	$\{t_3, t_6\}$
$\{t_1, t_6\}$	$\{t_4, t_5\}$
$\{t_2, t_3\}$	$\{t_4, t_6\}$
$\{t_2, t_4\}$	$\{t_5, t_6\}$
$\{t_2, t_5\}$	

(b) All conflicts

Figure 1: *dist* **with different** ρ**,** n **and** σ

any two-tuple subset J of I is called a φ-*conflict* in I with respect to φ if $J \not\models \varphi$.

Let Σ be a set of functional dependencies, we usually use σ to refer to the number of functional dependencies in Σ, *i.e.* $\sigma = |\Sigma|$. Given a set of functional dependency Σ, an instance I is said to be consistent with respect to Σ if I satisfies every functional dependencies in Σ. Otherwise, I is inconsistent, denoted as $I \not\models \Sigma$. As a special case, any two-tuple subset J of I is called a *conflict* in I if there is some φ such that J is a φ-*conflict*. That is, I contains one or more *conflicts* if I is inconsistent.

Example 1. Our running example is around the schema order(id, name, AC, PR, PN, STR, CTY, CT, ST, zip). Each tuple contains information about an item sold (a unique item id, name and price PR), and the phone number (area code AC, phone number PN) and the address of the customer who purchased the item (street STR, country CTY, city CT, state ST). An instance I of the schema order is shown in figure 1(a). Some functional dependencies on the order database include:

$\mathsf{fd}_1 : [\mathsf{AC}, \mathsf{PN}] \rightarrow [\mathsf{STR}, \mathsf{CT}, \mathsf{ST}]$ $\quad$ fd_2: $[\mathsf{zip}] \rightarrow [\mathsf{CT}, \mathsf{ST}]$

$\mathsf{fd}_3 : [\mathsf{id}] \rightarrow [\mathsf{name}, \mathsf{PR}]$ $\quad$ fd_4: $[\mathsf{CT}, \mathsf{STR}] \rightarrow [\mathsf{zip}]$

The database of figure 1(a) is inconsistent since there are 13 fd_2-conflicts in total as listed in figure 1(b). The meaning of the number assigned to each conflict will be clarified later.

Equivalence Class. Given an FD φ: $\mathsf{X} \rightarrow \mathsf{Y}$, an instance I can be partitioned horizontally into several *determinant equivalence classes* according to the X-values, that is, tuples in each determinant equivalence class share the same value of X. Moreover, any determinant equivalence class can be further partitioned into several *determinant-dependent equivalence classes* according to the Y-values, denoted as $[xy]$, that is, tuples in each determinant equivalence class share the same value of XY. It is obviously that, for an FD $\varphi : \mathsf{X} \rightarrow \mathsf{Y}$ and any instance I, two tuples $\mathbf{s}$ and $\mathbf{t}$ not in any φ-conflict in I must be in different determinant equivalence classes $[\mathbf{s}.\mathsf{X}]$ and $[\mathbf{t}.\mathsf{X}]$ respectively.

Example 2. With respect to $\mathsf{fd}_2 : [\mathsf{zip}] \rightarrow [\mathsf{CT}, \mathsf{ST}]$, instance I can be partitioned into one determinant equivalence class $[60462] = \{\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{t}_4, \mathbf{t}_5, \mathbf{t}_6\}$ and 4 determinant-dependent equivalence classes $\{\mathbf{t}_1, \mathbf{t}_2\}$, $\{\mathbf{t}_3, \mathbf{t}_5\}$, $\{\mathbf{t}_4\}$ and $\{\mathbf{t}_6\}$.

Repair. Let I be an instance over a relation schema R, a subset of I is an instance J obtained from I by eliminating some tuples. If J is a subset of I, then the distance from J to I, denoted $dist_{\text{sub}}(J, I)$, is the number of tuples missing from I, and it is for sure that $J \subseteq I$, thus,

$$dist_{\text{sub}}(J, I) = |I \setminus J| = |I| - |J|$$

Let I be an instance over schema R, and let Σ be a set of FDs. A consistent subset of I with respect to Σ is a subset J of I such that $J \models \Sigma$. A subset repair (*s-repair*, for short) is a consistent subset that is not strictly contained in any other consistent subset. An optimal subset repair of I is a consistent subset J of I such that $dist_{\text{sub}}(J, I)$ is minimum among all consistent subsets of I. Note that, each optimal subset repair is a repair, but not necessarily vice versa. Clearly, any consistent subset can be polynomially transformed into a subset repair, with no increase of distance. Unless explicitly stated otherwise, in this paper, we do not distinguish between a subset repair and a consistent subset.

Example 3. Both $S_1 = \{\mathbf{t}_4\}$ and $S_2 = \{\mathbf{t}_1, \mathbf{t}_2\}$ are s-repairs of I. It is easy to verify that S_2 is an optimal s-repair such that $dist_{\text{sub}}(S_2, I) = 4$.

Now, we formally define the first problem studies in this paper as follows,

Definition 1 (OSR Computing). *Input an instance I over a relation schema R, a functional dependency set Σ, OSR computing problem is to compute an optimal s-repair J of I with respect to Σ.*

Inconsistency Measurement. Computing an optimal s-repair helps estimating database FD-inconsistency degree. As in literature [9], given a functional dependency set Σ, one of subset repair based measurements on the FD-inconsistency degree of input database I is defined as following,

$$\textit{inc-deg}^S(I, \Sigma) = \min_{\substack{J \subseteq I, \\ J \models \Sigma}} \left\{ \frac{dist_{\text{sub}}(J, I)}{|I|} \right\} = \frac{dist_{\text{sub}}(J_{opt}, I)}{|I|}$$

Moreover, this measurement could be also applied for any part H of the input database I in order to evaluate its corresponding FD-inconsistency degree as following,

$$\textit{inc-deg}^S(I, H, \Sigma) = \min_{\substack{J \subseteq H, \\ J \models \Sigma}} \left\{ \frac{dist_{sub}(J, H)}{|H|} \right\} = \textit{inc-deg}^S(H, \Sigma)$$

The local degree does not depends on the whole of the input data, thus leads to the right equation. Our FD-inconsistency degree of any part is defined locally, hence, we use notation $\textit{inc-deg}^S(H, \Sigma)$ instead of $\textit{inc-deg}^S(I, H, \Sigma)$ by omitting the first parameter. Then, we here formally define the second problem studied in this paper as follows,

Definition 2 (FD-inconsistency Evaluation). *Input a relation schema R, an FD set Σ, an instance I over R and a subset query Q on I, FD-inconsistency evaluation is to compute inc-deg$^S(Q(I), \Sigma)$ of the query result $Q(I)$ with respect to Σ.*

Example 4. As mentioned in Example 3, $S_2 = \{\mathbf{t}_1, \mathbf{t}_2\}$ is an optimal s-repair of I, then $\textit{inc-deg}^S(I, \Sigma) = \frac{dist_{\text{sub}}(S_2, I)}{|I|} = \frac{2}{3}$. Given a range query $Q = [15, 45]$ on attribute PR in order, the result set $Q(I) = \{\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_4, \mathbf{t}_6\}$. S_2 ia also an optimal s-repair of $Q(I)$, then $\textit{inc-deg}^S(Q(I), \Sigma) = \frac{dist_{\text{sub}}(S_2, I)}{|Q(I)|} = \frac{1}{2}$.

Approximation. We follow the convention of approximation definition, to define the approximation of optimal repairs explicitly. For a constant $c \geq 1$, a c-optimal *s-repair* is an *s-repair* J of I such that

$$dist_{\text{sub}}(J, I) \leq c \cdot dist_{\text{sub}}(J', I)$$

for all *s-repairs* J' of I. In particular, an optimal *s-repair* is the same as a 1-optimal *s-repair*.

According to the definition of subset repair based FD-inconsistency degree, for an arbitrary $0 \leq \epsilon \leq 1$ and a constant $c \geq 1$, $\tilde{\textit{inc-deg}}^S(I, \Sigma)$ is a (c, ϵ)-approximation of $\textit{inc-deg}^S(I, \Sigma)$ such that

$$\textit{inc-deg}^S(I, \Sigma) \leq \tilde{\textit{inc-deg}}^S(I, \Sigma) \leq c \cdot \textit{inc-deg}^S(I, \Sigma) + \epsilon$$

Complexity. The conventional measure of *data complexity* are adopted to perform the computational complexity analysis of optimal subset repair computing problem in this paper. That is, the relation schema $R(A_1, \ldots, A_k)$ and the functional dependency set Σ are fixed in advance, and the instance data I over R is the only input. Therefore, an polynomial running time may have an exponential dependency on k and $|\Sigma|$. In such context of data complexity, each distinct setting of $R(A_1, \ldots, A_k)$ and Σ indicates a distinct problem of finding an optimal repair, so that different setting may indicate different complexities. Recall that, in the measurement of combined complexity, the relation schema and the functional dependency set are considered as inputs, hence, the hardness of OSR computing problem equals to that of vertex cover problem. However, this is not the case under data complexity.

After showing the hardness, we still adopt data complexity to be the measurement on running times and approximation ratios, however, the difference is that we fix only the size of the functional dependency set Σ, but not itself and the schema. Note that, this is reasonable in practical, the input functional dependencies may vary with time, but the number of given functional dependencies are always much smaller than the size of input data I, so that we could consider it to be bounded within some constant.

3. COMPUTING AN OPTIMAL S-REPAIR

In this section, we show the improved lower bound and upper bound of OSR.

3.1 The Strengthened Dichotomy for OSR

Livshits *et al.* gave a procedure OSRSucceed(Σ) [27] to simplify a given functional dependency set Σ. Any functional dependency set can either be simplified polynomially into a set containing only *trivial* functional dependencies, or not. The procedure OSRSucceed(Σ) returns `true` for the former case, otherwise `false`. OSR is polynomially tractable for functional dependency sets that can be simplified into trivial ones. For all the other functional dependency sets, OSR computing problem is hard as in not only `NPhard` but also `APXcomplete`.

Specifically, any functional dependency set that cannot be simplified further can be classified into one of five certain classes of functional dependency sets. And OSR is shown in `APXcomplete` for any such functional dependency set by *fact-wise* reductions from one of the following four fixed schemas.

$$\begin{aligned}
\Sigma_{\mathsf{A}\to\mathsf{B}\to\mathsf{C}} &= \{A \to B, B \to C\} \\
\Sigma_{\mathsf{A}\to\mathsf{B}\leftarrow\mathsf{C}} &= \{A \to B, C \to B\} \\
\Sigma_{\mathsf{AB}\to\mathsf{C}\to\mathsf{B}} &= \{AB \to C, C \to B\} \\
\Sigma_{\mathsf{AB}\leftrightarrow\mathsf{AC}\leftrightarrow\mathsf{BC}} &= \{AB \to C, AC \to B, BC \to A\}
\end{aligned}$$

By showing the inapproximability of such four schemas, the following dichotomy follows immediately.

Theorem 1 (Dichotomy for OSR computing [27]). *Let Σ be a set of FDs, then*

- *An optimal subset repair can be computed polynomially, if* OSRSucceed(Σ) *returns* `true`*;*
- *Computing an optimal subset repair is* `APXcomplete`*, if* OSRSucceed(Σ) *returns* `false`*.*

In this paper, we give a more careful analysis to show a concrete constant for each of the four schemas, thus strengthening this dichotomy.

Lemma 1. *For FD sets $\Sigma_{\mathsf{A}\to\mathsf{B}\to\mathsf{C}}$, $\Sigma_{\mathsf{A}\to\mathsf{B}\leftarrow\mathsf{C}}$, and $\Sigma_{\mathsf{AB}\to\mathsf{C}\to\mathsf{B}}$, there is no polynomial-time $(\frac{17}{16}-\epsilon)$-approximation algorithm for computing an optimal subset repair for any $\epsilon > 0$, unless* `NP=P`*.*

Lemma 2. *For FD set $\Sigma_{\mathsf{AB}\leftrightarrow\mathsf{AC}\leftrightarrow\mathsf{BC}}$, there is no polynomial time $(\frac{69246103}{69246100} - \epsilon)$-approximation algorithm for computing an optimal subset repair for any $\epsilon > 0$, unless* `NP=P`*.*

For detailed proofs of lemma 1 and 2, we refer to [**?**]. Based on Lemma 1,2 and Theorem 1, a strengthened dichotomy for OSR computing can be stated as follows.

Theorem 2 (A strengthened dichotomy for OSR). *Let Σ be a set of FDs, then*

- *An optimal subset repair can be computed polynomially, if* OSRSucceed(Σ) *returns* `true`*;*
- *There is no poly-time $(\frac{69246103}{69246100} - \epsilon)$-approximation to compute an optimal subset repair, if* OSRSucceed(Σ) *returns* `false` *and Σ can be classified into the class having a fact-wise reduction from $\Sigma_{\mathsf{AB}\leftrightarrow\mathsf{AC}\leftrightarrow\mathsf{BC}}$ to itself;*
- *There is no poly-time $(\frac{17}{16} - \epsilon)$-approximation to compute an optimal subset repair, otherwise.*

For the polynomial-intractable side, one can simply verify that if the size of FD set is unbounded, then the OSR computing is as hard as classical vertex cover problem on general inputs which is `NPhard` to be approximate within $2 - \epsilon$ for any $\epsilon > 0$. A simple approximation algorithm can provide a ratio of 2 when the input FD set is unbounded.

However, in practical, the size of FD set is usually much smaller than the size of data, so that it can be treated as fixed, especially in the context of big data. Unfortunately, it is still unclear how good we could arrive when the size of FD set is bounded. Therefore, to study the upper bound of its data complexity, we next give a carefully designed approximation to archive a ratio of $2 - 0.5^{\sigma-1}$ when the number of given FDs is σ, or even better sometimes.

3.2 Approximation

To investigate the upper bound of optimal s-repair computing problem, we start from a basic linear programming to provide a ratio of $2-0.5^{\mathcal{X}(n)}$, for an input instance I over a given relation schema R and an input FD set Σ, where $\mathcal{X}(I)$ is the number of all possible *determinant-dependent equivalence classes* of an input instance I with respect to the input FD set Σ. Then, an improved the approximation ratio $2-0.5^{\sigma-1}$ could be derived by means of triad elimination. Finally, we find another $(2-\eta_k+\frac{\eta_k}{k})$-approximation which is sometimes, but not always, better than $2-0.5^{\sigma-1}$, based on a k-quasi-Turán characterization of the input inconsistent instance with respect to the input Σ.

3.2.1 A basic approximation algorithm

We start from the basic linear programming model which is equivalent to the classical one solving minimum vertex cover problem.

Let x_i be a 0-1 variable indicating the elimination of tuple $\mathbf{t}_i$ such that, $x_i = 1$ if eliminate $\mathbf{t}_i$; $x_i = 0$ otherwise. Then we formulate the OSR computing problem as followings,

$$\begin{aligned} minimize \quad & \textstyle\sum_{\mathbf{t}_i \in I} x_i && (1)\\ s.\,t. \quad & x_i + x_j \geq 1, \quad \forall\ \{\mathbf{t}_i, \mathbf{t}_j\} \not\models \Sigma, && (2)\\ & x_i \geq 0, \quad \forall\ \mathbf{t}_i \in I && (3) \end{aligned}$$

It is well-known that every extreme point of this model takes value of 0 or 0.5 or 1, hence, we can relax it with condition:

$$x_i \in \{0, 0.5, 1\}$$

thus getting

$$OPT^{relax} \leq OPT$$

A trivial rounding derives a ratio of 2 immediately. However, based on a partition of instance I with respect to FD set Σ, a better ratio depending on the size of partition could be obtained.

Obviously, for any FD $\varphi_i : \mathsf{X}_i \rightarrow \mathsf{Y}_i$ of Σ with a size of σ, each tuple $\mathbf{t}$ belongs to one and only one distinct determinant-dependent equivalence class with respect to φ_i, say $[\mathbf{t}.\mathsf{X}_i\mathsf{Y}_i]$, then we have

$$\mathbf{t} \in [\mathbf{t}.\mathsf{X}_1\mathsf{Y}_1] \cap \cdots \cap [\mathbf{t}.\mathsf{X}_\sigma\mathsf{Y}_\sigma] = [\mathbf{t}.\mathsf{Z}],$$

where $\mathsf{Z} = \mathsf{X}_1 \cup \cdots \cup \mathsf{X}_\sigma \cup \mathsf{Y}_1 \cup \cdots \cup \mathsf{Y}_\sigma$. Hence, we observe that if any two tuples $\mathbf{s}$ and $\mathbf{t}$ are in some conflict, then there must be

$$\mathbf{s} \notin [\mathbf{t}.\mathsf{Z}], \mathbf{t} \notin [\mathbf{s}.\mathsf{Z}],$$

and vice versa, since they disagree on at least one attribute in some Y_i but agree on all the attributes in X.

Further more, another observation is that all the tuples in conflict with $\mathbf{t}$ are included in the determinant equivalence classes

$$[\mathbf{t}.\mathsf{X}] = [\mathbf{t}.\mathsf{X}_1] \cup \cdots \cup [\mathbf{t}.\mathsf{X}_\sigma]$$

Because all tuples in each $[\mathbf{t}.\mathsf{X}_i]$ may be inconsistent with each other at worst, hence, every tuple in $[\mathbf{t}.\mathsf{X}]$ may be inconsistent with at most $|[\mathbf{t}.\mathsf{X}_1]| \times \cdots \times |[\mathbf{t}.\mathsf{X}_\sigma]| - 1$ tuples.

Let $\mathcal{X}(\mathbf{t})$ be the numbers of tuples who are in conflict with $\mathbf{t}$, and $\mathcal{X}(I)$ be the numbers of consistent classes that I could be partitioned into, such that each class is consistent. This observation implies the following claims immediately,

CLAIM 1. $\mathcal{X}(I) \leq \max_{\mathbf{t}\in I}\{\mathcal{X}(\mathbf{t})\} \leq \max_{\mathbf{t}\in I}\{|[\mathbf{t}.\mathsf{X}_1]| \times \cdots \times |[\mathbf{t}.\mathsf{X}_\sigma]|\}$

This claim implies that all the tuples in I could be partitioned into at most $\mathcal{X}(I)$ classes such that tuples in each class are consistent with each other.

Then, we improve the ratio by using the $\mathcal{X}(I)$ partitions of the input instance. Based on the rounding technique similar with [29], an improved approximated algorithm could be stated as follows.

Algorithm 1 Baseline LP-OSR

Input: n-tuple instance I over schema R, FD set Σ
Output: optimal subset repair J of I with respect to Σ

1: Solve the linear programming (2)-(4) to obtain a solution $x_1 \dots x_n$ such that $x_i \in \{0, 0.5, 1\}$ for all $1 \leq i \leq n$.
2: Let P_j is the set of tuples of some consistent partition of I with respect to Σ
3: $j \leftarrow \arg\max_j |\{x_i |\ \mathbf{t}_i \in P_j \wedge x_i = 0.5\}|$
4: **for** each $\mathbf{t}_i \in I$ **do**
5: **if** $x_i = 1$ or ($x_i = 0.5$ and $\mathbf{t}_i \notin P_j$) **then**
6: add $\mathbf{t}_i$ into $\hat{\Delta}$
7: **end if**
8: **end for**
9: $\hat{J} \rightarrow I \setminus \hat{\Delta}$
10: **return** $\hat{J}$

Obviously, OPT^{relax} can be returned in polynomial time as shown in [33], and it is easy to see that $\hat{J}$ is a s-repair. In fact, if an $x_i = 0.5$ and not be picked into deletion $\hat{\Delta}$, then all the tuples in conflicts with $\mathbf{t}_i$ must be added into $\hat{\Delta}$ because they are not in partition P_j, and the sum of two variables of tuples in any conflict should be no less than 1. Therefore, we claim that the approximation ratio is $2-\frac{2}{\mathcal{X}(I)}$.

LEMMA 3. *Algorithm 1 returns a* $\left(2-\frac{2}{\mathcal{X}(I)}\right)$*-optimal subset repair.*

The number $\mathcal{X}(I)$ is unbounded, in the worst case, could be as large as $|I|$ so that it is a factor depending on the size of input.

3.2.2 Improved ratio by triad eliminating

Reducing the number of consistent partitions will improve the approximation. We introduce triad elimination in this section to decrease the partition number into a factor which is independent with the size of input but only depending on the number $\sigma = |\Sigma|$ of input functional dependencies.

Data reduction. Let $\mathbf{r}, \mathbf{s}, \mathbf{t}$ be three tuples in I, then they are called a *triad* if any two of them are in a conflict with respect to Σ. An important observation is that any s-repair contains at most one tuple of a triad in I, especially in an optimal s-repair, hence, any triad elimination yields a 1.5-optimal s-repair of itself. Therefore, we could preform a data

reduction by eliminating all the disjoint triads without the loss of an approximation ratio 1.5.

Based on the data reduction, the improved algorithm can be shown as follow.

Algorithm 2 TE LP-OSR

Input: n-tuple instance I over schema R, FD set Σ
Output: optimal subset repair J of I with respect to Σ

1: Find a maximal tuple set of disjoint triads Δ from I
2: $I' \leftarrow I \setminus \Delta$
3: $\hat{J} \leftarrow$ Baseline LP-OSR(I')
4: **return** $\hat{J}$

Let σ be the number of functional dependencies in Σ, then we claim that TE LP-OSR will return a better approximation as the following theorem.

THEOREM 3. *Algorithm* TE LP-OSR *returns a* $\left(2 - 0.5^{\sigma-1}\right)$*-optimal subset repair.*

Note that this ratio depends on only the size of functional dependency set other than the scale of input data. Therefore, a simple corollary implies a ratio of 1.5 for $\Sigma_{\mathsf{A}\to\mathsf{B}\to\mathsf{C}}$, $\Sigma_{\mathsf{A}\to\mathsf{B}\leftarrow\mathsf{C}}$, and $\Sigma_{\mathsf{AB}\to\mathsf{C}\to\mathsf{B}}$, and 1.75 for $\Sigma_{\mathsf{AB}\leftrightarrow\mathsf{AC}\leftrightarrow\mathsf{BC}}$, no matter how large of the input data.

A naive enumeration of triad is time wasting. In our algorithm, as in the proof of theorem 3, it is not necessary to eliminate all disjoint triads as possible. Instead, to obtain a good ratio, it needs only eliminate all disjoint triads with respect to each single functional dependency. Then, for each single functional dependency, sorting or hashing techniques could be utilized to speed up the triad eliminating, and skip the finding of triads across different functional dependencies.

3.2.3 Improved ratio by k-quasi-Turán property

Triad elimination based TE LP-OSR does not capture the characteristic of input data instance. We next give another approximation algorithm QT LP-OSR. In fact, we found that constraints could be derived to strengthen LP formula. Intuitively, for each functional dependency, each determinant equivalence class contains several determinant-dependent equivalence classes, say k, hence, tuples in at least $k-1$ classes should be eliminated from I to obtain an s-repair. Therefore, constraints could be invented to limit the lower bound of variables taking value 1 according to the $k-1$ classes, so that a better ratio could be obtained for some featured cases.

Formally, consider a determinant equivalence class $[p]$ containing m determinant-dependent equivalence classes $[pq_1]$, ..., $[pq_m]$, hence,

$$|[p]| = |[pq_1]| + \cdots + |[pq_m]|$$

k-quasi-Turán. Given $k > 1$, a tuple $\mathbf{t} \in I$ is of *k-quasi-Turán* property if and only if there is some functional dependency φ and a determinant equivalence class $[p]$ with respect to φ such that

$$\mathbf{t} \in [p], m \geq 3, \forall i, 1 \leq i \leq m, |[p]| - |[pq_i]| \geq k\,|[pq_i]|$$

Example 5. As mentioned in Example 2, given the functional dependency $\mathsf{fd}_2 : [\mathsf{zip}] \to [\mathsf{CT}, \mathsf{ST}]$, the determinant equivalence class [60462] is partitioned into $m = 4$ determinant-dependent equivalence classes. It is easy to verify that [60462] is a 2-*quasi-Turán*

Then we characterize the data with parameter η_k which is the portion of k-quasi-Turán tuples in I. A strengthened LP could be formulated as follows,

$$\begin{aligned}
\textit{minimize} \quad & \textstyle\sum_{\mathbf{t}_i \in I} x_i & \\
\textit{s. t.} \quad & x_i \geq 0, & \forall\ \mathbf{t}_i \in I \\
& x_i + x_j \geq 1, & \forall\ \{\mathbf{t}_i, \mathbf{t}_j\} \not\models \Sigma, \\
& \textstyle\sum_{\mathbf{t}_i \in [p]} x_i > |[p]| - \max_j |[pq_j]| - \epsilon, & \text{for every } [p].
\end{aligned}$$

In this model, pick a small enough $\epsilon > 0$, the inequality guarantees that in any integral solution, at least $|[p]| - \max_j |[pq_j]|$ variables taking value of 1. However, in the fractional solution, we could not limit the number of 1-variables, for example, a slop line cannot distinguish points $(0.5, 0.5)$, $(0, 1)$ and $(1, 0)$. However, even so, we will show that this number could still be limited to improve the approximation ratio.

CLAIM 2. *Every extreme point of any solution to the linear programming is in* $\{0, 0.5, 1\}$.

One can simply verify the correctness and prove it by contradiction, we omit the proof here.

Every solution of this strengthened linear programming still admits the half-integral property, hence, we take the basic rounding strategy such that

$$x_i = \begin{cases} 0, & \text{if} \quad x_i = 0, \\ 1, & \text{if} \quad x_i = 0.5, \\ 1, & \text{if} \quad x_i = 1. \end{cases}$$

then, for tuples in each determinant equivalence class $[p]$, at most $\max_j |[pq_j]|$ variables will be rounded as 1 wrongly. Formally, for each determinant equivalence class $[p]$, define $S_1^{[p]}$ and $S_{0.5}^{[p]}$ as follows,

$$S_1^{[p]} \coloneqq \{x_i |\ \mathbf{t}_i \in [p], x_i = 1\}, \quad S_{0.5}^{[p]} \coloneqq \{x_i |\ \mathbf{t}_i \in [p], x_i = 0.5\}$$

then we have the following lemma,

LEMMA 4. $\left|S_1^{[p]}\right| \geq |[p]| - 2\max_j |[pq_j]|$, $\left|S_{0.5}^{[p]}\right| \leq 2\max_j |[pq_j]|$

This lemma derives the a ratio depending on the portion of k-quasi-Turán tuples η_k where $0 < \eta_k \leq 1$ for any $k \geq 2$.

THEOREM 4. QT LP-OSR *returns a* $(2 - \eta_k + \frac{\eta_k}{k})$*-optimal subset repair.*

For detailed proofs of Lemma 4 and Theorem 4 we refer to [**?**].

Remarks. Combine the approximations based on the strengthened LP with triad elimination, a better approximation is provided. Note that, it is polynomial-time to find a best pair (k, η_k) to capture the data characteristic as possible, so as to improve the ratio as much as possible.

4. FAST ESTIMATE FD-INCONSISTENCY DEGREE

The hardness of OSR computing implies FD-inconsistency degree evaluation is also hard. Therefore, we take effort to find an approximation of such degree. Fortunately, an observation is that we aim to compute the ratio, but not any OSR itself, hence, to achieve a constant relative ratio, a relaxation of approximation ratio with an $O(n)$ factor is

allowed. In this section, we show a fast FD-inconsistency evaluation of subset query result. To obtain a good approximation in sublinear complexity, we allow a relative ratio 2 and an additional additive error ϵ where $0 < \epsilon < 1$, *i.e.*, given an FD set Σ and a subset query Q on an instance I, the algorithm computes an estimation $\widetilde{inc\text{-}deg}^S(Q(I), \Sigma)$ such that with high constant probability such that

$$inc\text{-}deg^S(Q(I), \Sigma) \leq \widetilde{inc\text{-}deg}^S(Q(I), \Sigma) \leq 2 \cdot inc\text{-}deg^S(Q(I), \Sigma) + \epsilon$$

4.1 Subset Query Oracle

As the diversity of subset queries, we model them as a $\subseteq$-oracle, such that, query complexity of the algorithm can be analyzed in terms of operations supported by the oracle. The rest work is to find out the way of implementing the $\subseteq$-oracle for a specific subset query. The time complexity of FD-Inconsistency evaluation for this kind of subset query then can be derived by combining query complexity and time complexity of the oracle.

Given an instance I of a relation schema R and a subset query Q, the corresponding $\subseteq$-oracle $O(I, Q)$ is required to answer three queries about the result $Q(I)$:

$\mathsf{O(I, Q).sample\_tuple()}$. Since the algorithm introduced later is sample-based, the oracle has to provide a uniform sample on the result set $Q(I)$. But sampling after the evaluation of Q is incompetent to obtain a sublinear approximation, since the retrieval of $Q(I)$ will take at least linear time. A novel method of sampling is essential to implement the oracle.

$\mathsf{O(I, Q).in\_result(t)}$. It is to check the membership of a tuple $\mathbf{t}$ of $Q(I)$, such that, it returns true if the input tuple $\mathbf{t}$ belongs to $Q(I)$, otherwise , it returns *false*. As we shown in the next subsection, it is mostly used to check if $Q(I)$ contains the conflict $\{\mathbf{t}, \mathbf{s}\}$.

$\mathsf{O(I, Q).size()}$. Recall the definition of $inc\text{-}deg^S(Q(I), \Sigma)$, the result size is in the denominator. It only returns the number of tuples in $Q(I)$. Obviously, it is intolerable to compute the size by evaluating the query.

As an example, we next show a concrete implementation of $\subseteq$-oracle for range queries.

An implement of $\subseteq$-oracle. In the following, we present an indexing-based implement of $\subseteq$-oracle for range queries. Without the loss of generality, let $[low, high]$ be the query range of attribute A, hence, the corresponding query result consists of all tuples $\mathbf{t}$ such that $low \leq \mathbf{t}.A \leq high$. Then B^+-tree index with in-node binary search is sufficient to implement the $\subseteq$-oracle.As mentioned before, the most challenge is to implement the three operations in sublinear time, for the detailed explanation, one can refer to our report [**?**].

Nevertheless, based on our model, one could also be free from the consideration on materialization of query result, such as we shown for range queries, the materialization of query result could be avoided.

4.2 Ranking and $(2, \epsilon)$-Estimation

Recall that, a two-tuple subset $J = \{\mathbf{t}, \mathbf{s}\}$ is a conflict if and only if $\mathbf{s} \in [\mathbf{t}.\mathsf{X}] \setminus [\mathbf{t}.\mathsf{XY}]$ with respect to some FD: $\mathsf{X} \to \mathsf{Y}$ in Σ. Let C be the set of all conflicts in an instance I, then an *s-repair* S of I can be derived in the following way: ranking all the conflicts of C in an ascendant order Π, *pick* the current first conflict $J = \{\mathbf{t}, \mathbf{s}\}$ and *remove* tuples $\mathbf{t}$ and $\mathbf{s}$ from I, then *eliminate* all the conflicts containing $\mathbf{t}$ or $\mathbf{s}$ from C, repeat the *pick-remove-eliminate* procedure until no any conflict left in C, then the I left is a repair S.

We claim that S is a 2-optimal s-repair of I. The proof is quite straightforward, observe that, any repair has to eliminate at least one tuple of the conflicts picked in such procedure, then we have

$$\frac{1}{2} dist_{\text{sub}}(S, I) = \frac{1}{2}(|I| - |S|) \leq dist_{\text{sub}}(S_{opt}, I),$$

thus achieving a 2-optimal s-repair.

By applying Chernoff bound, if we uniformly sample $p = \Theta(\frac{1}{\epsilon^2})$ tuples, and count the number of tuple not in S, say q, then with high probability

$$|S| - \frac{\epsilon}{2} n \leq \frac{p - q}{p} \cdot n \leq |S| + \frac{\epsilon}{2} n.$$

Hence we can obtain a $(2, \epsilon)$-approximation of $inc\text{-}deg^S(I, \Sigma)$ as defined previously. And observe each ranking Π decides a 2-optimal s-repair S so that we could scan the ranking once, *verify* the membership of each tuple in S, and count the number q, however, in such a trivial way, it takes a linear time complexity. In the next subsection, we will give a sublinear time implementation of the verification step.

We argue that the sampling method mentioned above still works for the 2-optimal s-repair of $Q(I)$ with the same probabilistic error bound. Consider a subset query Q on I, a

conflict	conflict
$\langle\{t_2, t_4\}, 1\rangle$	$\langle\{t_3, t_4\}, 8\rangle$
$\langle\{t_5, t_6\}, 2\rangle$	$\langle\{t_2, t_3\}, 9\rangle$
$\langle\{t_1, t_4\}, 3\rangle$	$\langle\{t_4, t_5\}, 10\rangle$
$\langle\{t_3, t_6\}, 4\rangle$	$\langle\{t_4, t_6\}, 11\rangle$
$\langle\{t_2, t_5\}, 5\rangle$	$\langle\{t_1, t_3\}, 12\rangle$
$\langle\{t_2, t_6\}, 6\rangle$	$\langle\{t_1, t_5\}, 13\rangle$
$\langle\{t_1, t_6\}, 7\rangle$	

Figure 2: Ranking of conflicts in I and Ranking of conflicts in $Q(I)$ (yellow shaded)

conflict J in any $Q(I)$ is still a conflict in I, and the ranking induced by any $Q(I)$ from Π is still ascendant. Continue with Example 4, given a range query $Q = [15, 45]$ on PR, the result set $Q(I)$ is $\{\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_4}, \mathbf{t_6}\}$. As shown in the figure 2, all conflicts in $Q(I)$ are yellowed faded and the ranking induced by them from Π is still ascendant. Let S be an optimal s-repair of $Q(I)$, then ranking Π could be reused to compute a $(2, \epsilon)$-approximation of $inc\text{-}deg^S(Q(I), \Sigma)$.

4.3 Fast Estimate FD-inconsistency Degree

We first settle the preprocessing method, then show an efficient implementation of the *sample-and-verify* method for subset queries.

4.3.1 Preprocessing

Ranking-based method mentioned above implies a predefined rank could be reused for the FD-inconsistency degree evaluation of every subset query result. Therefore, we discover all the conflicts in I with respect to Σ in the preprocessing step, and assign a unified rank to C in advance. That is a distinct rank r is assigned to each conflict $J = \{\mathbf{t}, \mathbf{s}\}$ as shown in figure 1(b).

Algorithm 3 Preprocessing Procedure

Input: An instance I of a relation schema R, and a set of functional dependencies Σ
Output: Set C of conflicts in I and a ascendant ranking Π on C

1: **for** each two tuples subset $J = \{\mathbf{t}, \mathbf{s}\}$ of I **do**
2: **if** J is a *φ-conflict* for some $\varphi \in \Sigma$ in I **then**
3: Generate a unique and no duplicated ranking r;
4: Append $\langle\{\mathbf{t}, \mathbf{s}\}, r\rangle$ to C;
5: **end if**
6: **end for**
7: Sort C according to r;

Algorithm 3 illustrates the preprocessing procedure. Let n be the size of I. The running time of Preprocessing is at worst $O(n^2 \log n)$. Because there are at most $O(n^2)$ tuple pairs of I and we consider data complexity in this paper, it takes $O(n^2)$ time to find out all conflicts of I. And Step 7 may take $O(n^2 \log n)$ time. Note that, the number of conflicts are usually not that large in practice, techniques like hash-based partition could be taken as a tool to find all possible conflicts, so that the time cost of preprocessing could be further lower, but we do not emphasis them in this paper.

4.3.2 Verification Locally

Recall the *sampling-and-verification* procedure, for any tuple $\mathbf{t} \in Q(I)$ sampled uniformly, it is to check if $\mathbf{t}$ is in the 2-optimal s-repair of $Q(I)$ derived by given ranking Π. During this procedure, every conflict J in $Q(I)$ eliminated in the checking procedure has a lowest rank when we turn to check it. That is, any conflict J' in $Q(I)$ intersecting with J are either already eliminated or having a rank higher than J. It is easy for the sequential implementation if we scan the ranking from its beginning. However, it is difficult without scanning the entire ranking, since the sampled tuple may not locate in the beginning.

To enable a sublinear evaluation, we need a *start-from-anywhere* implement method. Fortunately, the locality of a ranking can be utilized to avoid scanning the entire ranking. We employ a recursive verification starting from the conflicts involving current sampled tuple.

Basically, we begin with the sampled tuple $\mathbf{t}$ and check the conflicts in $Q(I)$ caused by $\mathbf{t}$ in turn from rank lowest to highest. For each conflict J we currently considering, J should be eliminated if one of the following conditions holds,

- J is lowest among all the conflicts in $Q(I)$ intersecting with it,
- Every conflict J' in $Q(I)$ intersecting with J and lower than J are already known to be eliminated.

Otherwise, recursively check J' by the same procedure. At last, if none of conflicts in $Q(I)$ containing $\mathbf{t}$ is decided to be eliminated, then $\mathbf{t}$ should stay in S, otherwise not, and count it into q. The correctness of this method is obviously, the only concern is the running time which depends on the number of recursive calls. We next formally describe it and bound the number of recursive calls.

4.3.3 Sublinear Estimation

Algorithm 4 Fast-IncDeg performs s sampling-and-verifying operations and counts the number of tuples sampled but not in S by calling **function** NotInSR($\mathbf{t}, C$). Since S is a 2-optimal s-repair of I, Fast-IncDeg outcomes an $(2, \epsilon)$-estimation of FD-inconsistency degree of $Q(I)$.

Algorithm 4 Fast-IncDeg

Input: Set C of conflicts in I, subset query Q, error ϵ
Output: $\tilde{inc\text{-}deg}^S(Q(I), \Sigma)$

1: $\tilde{dist}_{\text{sub}} := 0$;
2: **for** $i := 1$ to $8/\epsilon^2$ **do**
3: $\mathbf{t} := O(I, Q)$.sample_tuple();
4: **if** NotInSR($\mathbf{t}, C$) **then**
5: $\tilde{dist}_{\text{sub}} := \tilde{dist}_{\text{sub}} + 1$;
6: **end if**
7: **end for**
8: **return** $\frac{\tilde{dist}_{\text{sub}}}{O(I,Q).\text{size}()} + \frac{\epsilon}{2}$;

Subroutine 1 NotInSR($\mathbf{t}, C$)

1: Let $\langle\{\mathbf{t}, \mathbf{t_1}\}, r_1\rangle, \cdots, \langle\{\mathbf{t}, \mathbf{t}_l\}, r_l\rangle$ be the tuples in C including $\mathbf{t}$ in order of increasing r;
2: **for** $i := 1$ to l **do**
3: **if** $O(I, Q)$.in_result($\mathbf{t}_i$) **and** Eliminate($\{\mathbf{t}, \mathbf{t}_i\}$) **then**
4: **return true**;
5: **end if**
6: **end for**
7: **return false**;
8: **function** Eliminate($\{\mathbf{t}, \mathbf{s}\}$)
9: Let $\langle\{\mathbf{t}_1, \mathbf{s}_1\}, r_1, \rangle, \cdots, \langle\{\mathbf{t}_l, \mathbf{s}_l\}, r_l\rangle$ be the tuples in C such that $\mathbf{t}_i \in \{\mathbf{t}, \mathbf{s}\}$ in order of increasing r;
10: **while** $r_i < r$ **do**
11: **if** $O(I, Q)$.in_result($\mathbf{s}_i$) **and** Eliminate($\{\mathbf{t}_i, \mathbf{s}_i\}$) **then**
12: **return false**;
13: **end if**
14: $i := i + 1$;
15: **end while**
16: **return true**;
17: **end function**

Subroutine 1 implements the *verification* by calling Eliminate recursively. Namely, as introduced in subsection-*4.3.2*, give a conflict $J = \{\mathbf{t}, \mathbf{s}\}$, it considers all conflicts which include $\mathbf{t}$ or $\mathbf{s}$ with lower ranking. If there are no such conflicts, it returns *true*. Otherwise, it performs recursive calls to these conflicts in the order of their ranking. If any one-step recursion returns *true*, it returns *false*; Otherwise, it returns *true*. With the help of Eliminate, the subroutine 1 checks if a tuple $\mathbf{t}$ belongs to S. Concretely, it performs Eliminate on all conflicts in $Q(I)$ including $\mathbf{t}$ in the order of their rankings, and if there exists a conflict such that Eliminate returns *true*, it returns *true*; otherwise, it returns *false*.

Now, we bound the number of recursive calls of Eliminate. First, we derive an important corollary from [30]. For a ranking injection $\pi : J \to r$ of all conflicts of an instance I and a tuple $\mathbf{t} \in I$, let $N(\pi, \mathbf{t})$ denote the number of conflicts that a call Eliminate(J) was made on in the course of the computation of NotInSR($\mathbf{t}$). Let Π denote the set of all ranking injections π over the conflicts of I. Given a tuple $\mathbf{t}$, let $\delta_{\mathbf{t}}$ be the number of conflicts containing $\mathbf{t}$. Then we define the maximum conflict number of I as $\delta_I = \max_{\mathbf{t} \in I}\{\delta_{\mathbf{t}}\}$

The average value of $N(\pi, \mathbf{t})$ taken over all ranking injections π and tuples $\mathbf{t}$ is $O(\delta_I^2)$, *i.e.*,

$$\frac{1}{m!} \cdot \frac{1}{n!} \cdot \sum_{\pi \in \Pi} \sum_{\mathbf{t} \in I} N(\pi, \mathbf{t}) = O(\delta_I^2) \tag{4}$$

THEOREM 5. *Algorithm Fast-IncDeg returns an estimate* $\tilde{incDeg}(Q(I), \Sigma)$ *with a probability at least* 2/3 *such that,*

$$inc\text{-}deg^S(Q(I), \Sigma) \leq \tilde{inc\text{-}deg}^S(Q(I), \Sigma) \leq 2 \cdot inc\text{-}deg^S(Q(I), \Sigma) + \epsilon.$$

The average query complexity taken over all rankings π*, subset queries* Q *and tuples* $\mathbf{t}$ *of* $Q(I)$ *is* $O(\frac{\delta_I^2}{\epsilon^2})$*, where the algorithm uses only queries supported by the* $\subseteq$*-oracle.*

PROOF. By applying an additive Chernoff bound, suppose that it is sampled uniformly and independently $s = \Theta(\frac{1}{\epsilon^2})$ tuples $\mathbf{t}$ from $Q(I)$, with probability more than 2/3,

$$\frac{dist_{\text{sub}}(S, Q(I))}{|Q(I)|} - \frac{\epsilon}{2} \leq \frac{\tilde{dist}_{\text{sub}}}{|Q(I)|} \leq \frac{dist_{\text{sub}}(S, Q(I))}{|Q(I)|} + \frac{\epsilon}{2}. \tag{5}$$

And with the fact that S is a 2-optimal *s-repair*, it is obtained that,

$$inc\text{-}deg^S(Q(I), \Sigma) \leq \tilde{inc\text{-}deg}^S(Q(I), \Sigma) \leq 2 \cdot inc\text{-}deg^S(Q(I), \Sigma) + \epsilon. \tag{6}$$

For query complexity, we first bound the number of calls of Eliminate(). Given the result $Q(I)$ of a subset query Q, let n' be the number of tuples in $Q(I)$, and m' be the number of conflicts contained in $Q(I)$, and the maximum conflict number of $Q(I)$. Now, consider the ranking Π' induced by $Q(I)$ from Π, then equation 4 implies,

$$\frac{1}{m'!} \cdot \frac{1}{n'} \cdot \sum_{\pi \in \Pi'} \sum_{\mathbf{t} \in Q(I)} N(\pi, \mathbf{t}) = O(\delta_{Q(I)}^2)$$

Notice that since the conflicts in $Q(I)$ is a subset of the conflicts in I, for each $\pi' \in \Pi'$, there are $\frac{m!}{m'!}$ number of $\pi \in \Pi$ can produce the same ranking on the conflicts of $Q(I)$. Group Π into $m'!$ groups $\{\Pi_1, \cdots, \Pi_{m'!}\}$, and for each $\pi \in \Pi_i$ and a fixed $\mathbf{t} \in Q(I)$, $N(\pi, \mathbf{t})$ has the same value. So we have,

$$\begin{aligned}
&\frac{1}{m!} \sum_{\pi \in \Pi} \frac{1}{|Q(I)|} \sum_{\mathbf{t} \in Q(I)} N(\pi, \mathbf{t}) \\
=&\frac{1}{m!} \sum_{\pi \in \{\Pi_1, \cdots, \Pi_{m'!}\}} \frac{1}{|Q(I)|} \sum_{\mathbf{t} \in Q(I)} N(\pi, \mathbf{t}) \\
=&\frac{1}{m!} \cdot \frac{m!}{m'!} \sum_{\pi \in \Pi'} \frac{1}{Q(I)} \sum_{\mathbf{t} \in Q(I)} N(\pi, \mathbf{t}) \\
=&O(\delta_{Q(I)}^2)
\end{aligned}$$

The we could derive the query complexity. Let $\mathcal{Q}$ be the space of queries, then for any $Q(I)$, we have $\delta_{Q(I)} \leq \delta_I$, so that the *average query complexity* is that

$$\begin{aligned}
&\frac{1}{m!} \sum_{\pi \in \Pi} \frac{1}{|\mathcal{Q}|} \sum_{Q \in \mathcal{Q}} \frac{1}{|Q(I)|} \sum_{\mathbf{t} \in Q(I)} N(\pi, \mathbf{t}) \\
=&\frac{1}{|\mathcal{Q}|} \sum_{Q \in \mathcal{Q}} \frac{1}{m!} \sum_{\pi \in \Pi} \frac{1}{|Q(I)|} \sum_{\mathbf{t} \in Q(I)} N(\pi, \mathbf{t}) \\
=&\frac{1}{|\mathcal{Q}|} \sum_{Q \in \mathcal{Q}} O(\delta_{Q(I)}^2) \\
\leq&O(\delta_I^2)
\end{aligned}$$

Obviously, there are $O(\frac{1}{\epsilon^2})$ calls to sample a tuple from the result set. For each sampled tuple $\mathbf{t}$, the average number of calls to Eliminate is $O(\delta_I^2)$. So the number of calls to in_result() is $O(\frac{\delta_I^2}{\epsilon^2})$. □

In addition, inspired by the methodology proposed in [30], a pre-defined ranking in the preprocessing can be saved by ranking on the fly, that is, we only need to discover all conflicts in the preprocessing step and ranking whenever it is required. Since the basic idea is similar with our method, we omit the detail here, instead, we compare the two different implements in our experiments to show the efficiency of our method.

5. EXPERIMENTS

This section experimentally evaluates the performace of our algorithms for OSR computing and FD-inconsistency evaluation.

5.1 Experimental Settings

All experiments are conducted on a machine with eight 16-core Intel Xeon processors and 3072GB of memory.

Dataset. We used two datasets to evaluate the performance of algorithms for OSR computing and FD-Inconsistency evaluaction experimentally.

Dataset 1: ORDER *data* is an instance of the schema order shown in Example 1. Our set Σ consists of 4 FDs taken from Example 1. To populate the relation we scraped product informations from AMAZON and collected real-life data: the zip and area codes for major cities and twons for all US states[1] and street informations for all the United States[2]. We generated datasets of various size, ranging from 10M to 100M tuples.

Dataset 2: DBLP *data* was extracted from DBLP Bibliography[3]. It consists of 40M tuples and the format is as follows:

dblp(title, authors, year, publication, pages, ee, url)

Each DBLP tuple contains the title of an article, the authors and the information of publication (year, publication venue, pages, electronic edition and, url) We designed 4 FDs for DBLP.

fd_1 : [title] → [author] $\quad$ fd_2: [ee] → [title]

fd_3 : [year, publication, pages] → [title, ee, url] $\quad$ fd_4: [url] → [title]

To add noise to a dataset, we randomly selected an attribute of a "correct" tuple and changed it either to a close value or to an existing value taken from another tuple. We appended such "dirty" tuples which violate at least one or more functional dependencies to the dataset. We set a parameter ρ ranging from 1% to 10% to control the noise rate.

Methods. We implemented the following algorithms: (a) the basic approximation algorithm BL LP-OSR and the improved approximation algorithm by triad elimination TE LP-OSR for OSR computing; (b) the sublinear estimation algorithm Fast-IncDeg based on two implements of $\subseteq$-oracle for range query with $O(1)$ and $O(\log n)$ time complexity respectively, and its variation mentioned in the subsection-*4.3.3* Fast-IncDeg_ol for FD-Inconsistency evaluation. Hence, there

[1] http://www.geonames.org/
[2] http://results.openaddresses.io/
[3] https://dblp.org/xml/

are totally 4 implements of Algorithm 4 for range query denoted by Fast-IncDeg$_c$, Fast-IncDeg$_{log}$, Fast-IncDeg$_c$-ol and Fast-IncDeg$_{log}$-ol respectively.

Metrics. Since a dataset I with n tuples is polluted by appending ρn dirty tuples, where ρ is noise rate, the number of tuples in the optimal repair S_{opt} must be larger than n, *i.e.*, $dist_{\text{sub}}(S_{opt}, I) \leq \rho n$. Hence, we calculate $dist_{\text{sub}}(\hat{J}, I)$ of BL LP-OSR and TE LP-OSR and use $2\rho n$ to evaluate the approximation ratio of them. What's more, according to the definition of FD-inconsistency degree, we treat $2\rho + \epsilon$ as the upper bound of FD-inconsistency degree to ensure the correctness of the Algorithm 4. To evaluate the efficiency of Algorithm 4, we issue 300 queries for each algorithm and each parameter set, and record the average of the query time.

5.2 Experimental Results

We report our findings concerning about the accuracy and efficiency of our algorithms.

Accuracy. We first show the accuracy of BL LP-OSR, TE LP-OSR and QT LP-OSR. In figure 3, we ran them on datasets consisting of 10K to 40K tuples with noise rate ρ ranging from 1% to 10% and calculated $\text{UB}_1 = 2\rho n$. The $dist_{\text{sub}}(\hat{J}, I)$ of BL LP-OSR, TE LP-OSR and QT LP-OSR are much less than UB_1. Because the approximation ratio only bounds the relation between worst case output of an algorithm and the optimal solution, BL LP-OSR sometimes performs better than the other two improved algorithm. What's more, it is discovered that, after triad elimination, the ratio of 0.5 solution becomes much less since they only appear when some conflicts form a cycle with odd length.

We also evaluate the accuracy of Fast-IncDeg. We ran Fast-IncDeg$_c$ with parameter $\epsilon = 0.01$ on the same datasets and calculated $\text{UB}_2 = (2\rho + \epsilon)n$. As shown in figure 3 the value return by Fast-IncDeg$_c$ is less than UB_2 even less than UB_1 mostly since the upper bound is loose. And it is greater than the values of BL LP-OSR, TE LP-OSR and QT LP-OSR since it returns an estimate with an additive error.

Efficiency. We evaluate the efficiency of our algorithms for FD-inconsistency evaluation. We first ran Fast-IncDeg$_c$, Fast-IncDeg$_{log}$, Fast-IncDeg-ol$_c$ and Fast-IncDeg-ol$_{log}$ on datasets with various size, noise rate ρ ranging from 1% to 10% and $\epsilon = 0.01$. 300 large queries were issued per dataset and the average query time was recorded.

Figure 4(a), 4(b), 4(e), and 4(f) indicate that the average query time increase with the number of tuples and noise rate since both of them influence the maximum conflicts number in $Q(I)$. Further experiments were performed to evaluate the impact of the maximum conflict number $\delta_{Q(I)}$ on the average query time. Since queries were generated randomly, we only bounded the maximum conflict number of the dataset δ_D. Therefore, the average query time shown in figures 4(d) and 4(h) remain basically the same with the increasing δ_D. As shown in figures 4(c) and 4(g), the average query time of Fast-IncDeg$_c$ and Fast-IncDeg-ol$_c$ change slightly due to the impact of the number of tuples on $\delta_{Q(I)}$. And the average query time of Fast-IncDeg$_{log}$ and Fast-IncDeg-ol$_{log}$ grow with the number of tuples.

Figure 4 also illustrates that no matter how the $\subseteq$-oracle is implemented, Fast-IncDeg performs better then Fast-IncDeg-ol. It is because that in Fast-IncDeg-ol the ranking is assigned on-the-fly when a conflict is queried and it is expensive to keep the ranking consistent in every tuple which the conflict concerned about. In addition, an efficient $\subseteq$-oracle indeed makes the average query time drop a lot.

6. RELATED WORK

As a principled approach managing inconsistency, Arenas et al. [5] introduced the notions of repairs to define consistent query answering. The definitions of repair differ in settings of integrity constraints and operation gain [3]. The most general form of integrity constraints are denial constraints [21], they are able to express the classic functional dependencies [1], inclusion dependencies [25], and so on. Data complexities of computing optimal repairs are widely studied in the past. The complexity of tuple-level deletion based subset repair [13, 27] is studied respectively in the past. And the complexity of cell-level update based v-repair is also studied in [27, 26]. APXcompleteness of both optimal subset repair and v-repair computation has been shown for in these works.

For the upper bound, the best approximation on subset repair is still 2 obtained by solving the corresponding vertex cover problem [27] without the limitation on the number of given FDs. For the setting of fixed number of FDs, there are still no existing algorithmic result.

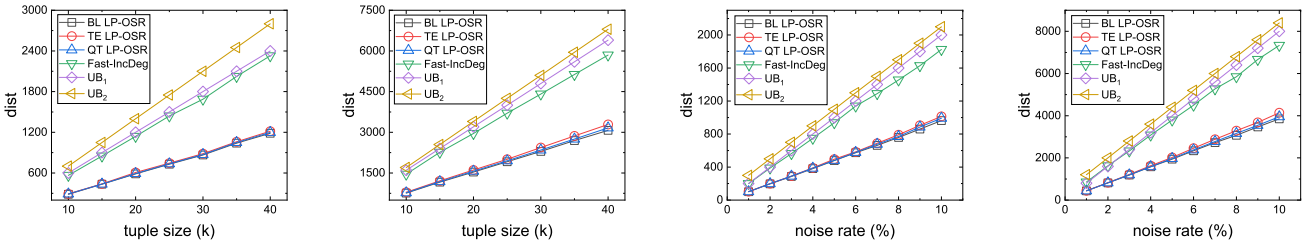
For the data repairing frameworks [2], there are two kinds of works which are based on FDs, they both aim to directly resolve the inconsistency of database. One kind of methods is to repair data based on minimizing the repair cost, *e.g.*, [5, 11, 16, 28, 34].

Given the data edit operations (including tuple-level and cell-level), minimum cost repair will output repaired data with minimizing the difference between it and the original one. But these work also do not provide us tight lower and upper bounds for data repairing. There are some other type of repairs not related with this paper, such as "minimum description length" [12], "relative trust" [10] and so on.

For inconsistency detection, there exists some detection techniques which are able to detect errors efficiently. SQL techniques for detecting FD violations were given in [13], practical algorithms for detecting violations of FDs in fragmented and distributed relations were provided in [19], and a incremental detection algorithm were proposed by [20]. In contrast to inconsistency detection, inconsistency evaluation need to compute the quantized dirtiness value of the data, rather than finding all violations.

7. CONCLUSIONS

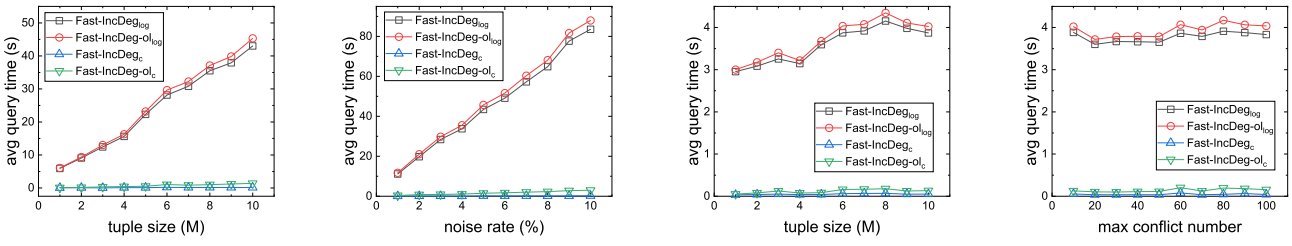
We revisit computing an optimal s-repair problem and fast estimate of s-repair based FD-inconsistency degree of subset query results. For the lower bound, we improve the inapproximability of optimal s-repair computing problem over most cases of FDs and schemas. For the upper bound, we developed two LP-based algorithms to compute a near optimal s-repair based on different characterization of input FDs and schemas respectively. Complexity results implies it is hard to obtain a good approximation polynomially, not to mention sublinear time for large data. For the FD-inconsistency degree, we present a fast $(2, \epsilon)$-estimation with an average sublinear query complexity, and achieve a sublinear time complexity whenever incorporating a sublinear implementation of the subset query oracle. This results give a way to estimate FD-inconsistency degree efficiently with theoretical guarantee.



(a) ORDER $\rho = 0.03$ (b) ORDER $\rho = 0.08$ (c) ORDER $n = 10$K (d) ORDER $n = 40$K

(e) DBLP $\rho = 0.03$ (f) DBLP $\rho = 0.08$ (g) DBLP $n = 10$K (h) DBLP $n = 40$K

Figure 3: *dist* with different ρ, n and σ



(a) ORDER $\rho = 0.05$ (b) ORDER $n = 10$M (c) ORDER $d = 50, \rho = 0.03$ (d) ORDER $n = 10$M, $\rho = 0.03$

(e) DBLP $\rho = 0.05$ (f) DBLP $n = 4$M (g) DBLP $d = 50, \rho = 0.03$ (h) DBLP $n = 4$M, $\rho = 0.03$

Figure 4: average query time with different n, ρ and d

8. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases: The Logical Level*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[2] F. N. Afrati and K. P. G. Repair checking in inconsistent databases: Algorithms and complexity. 2009.

[3] F. N. Afrati and P. G. Kolaitis. Repair checking in inconsistent databases: Algorithms and complexity. In *Proceedings of the 12th International Conference on Database Theory*, ICDT 09, pages 31–41, New York, NY, USA, 2009. ACM.

[4] O. Amini, S. Pérennes, and I. Sau. Hardness and approximation of traffic grooming. *Theoretical Computer Science*, 410(38-40):3751–3760, 2009.

[5] M. Arenas, L. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *Proceedings of the 18th ACM Symposium on Principles of Database Systems*, pages 68–79. ACM, 1999.

[6] A. Assadi, T. Milo, and S. Novgorodov. Dance: data cleaning with constraints and experts. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 1409–1410. IEEE, 2017.

[7] M. Bergman, T. Milo, S. Novgorodov, and W.-C. Tan. Qoco: A query oriented data cleaning system with oracles. *Proceedings of the VLDB Endowment*, 8(12):1900–1903, 2015.

[8] L. Bertossi. *Database Repairing and Consistent Query Answering*. Morgan Claypool Publishers, 2011.

[9] L. Bertossi. Repair-based degrees of database inconsistency. In *Logic Programming and Nonmonotonic Reasoning*, pages 195–209. Springer, Cham, 2019.

[10] G. Beskales, I. F. Ilyas, L. Golab, and A. Galiullin. On the relative trust between inconsistent data and inaccurate constraints. *arXiv preprint arXiv:1207.5226*, 2012.

[11] 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*, pages 143–154. ACM, 2005.

[12] F. Chiang and R. J. Miller. A unified model for data and constraint repair. In *ICDE*, 2011.

[13] J. Chomicki and J. Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput.*, 197(1-2), 2005.

[14] X. Chu, I. F. Ilyas, S. Krishnan, and J. Wang. Data cleaning: Overview and emerging challenges. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, page 2201–2206, New York, NY, USA, 2016. Association for Computing Machinery.

[15] 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, page 315–326. VLDB Endowment, 2007.

[16] 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*, pages 315–326. VLDB Endowment, 2007.

[17] P. Crescenzi. A short guide to approximation preserving reductions. In *Proceedings of Computational Complexity. Twelfth Annual IEEE Conference*, pages 262–273. IEEE, 1997.

[18] M. Dallachiesa, A. Ebaid, A. Eldawy, A. Elmagarmid, I. F. Ilyas, M. Ouzzani, and N. Tang. Nadeef: a commodity data cleaning system. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 541–552. ACM, 2013.

[19] W. Fan, F. Geerts, S. Ma, and H. Müller. Detecting inconsistencies in distributed data. 2010.

[20] W. Fan, J. Li, N. Tang, and W. Y. qa. Incremental detection of inconsistencies in distributed data. *IEEE Trans. on Knowl. and Data Eng.*, 26(6), 2014.

[21] T. Gaasterland, P. Godfrey, and J. Minker. An overview of cooperative answering. *Journal of Intelligent Information Systems*, 1(2):123–157, 1992.

[22] F. Geerts, G. Mecca, P. Papotti, and D. Santoro. The llunatic data-cleaning framework. *Proceedings of the VLDB Endowment*, 6(9):625–636, 2013.

[23] V. Guruswami and S. Khot. Hardness of max 3sat with no mixed clauses. In *Proceedings of the 20th Annual IEEE Conference on Computational Complexity*, pages 154–162. IEEE Computer Society, 2005.

[24] V. Kann. Maximum bounded 3-dimensional matching is max snp-complete. *Information Processing Letters*, 37(1):27–35, 1991.

[25] H. Koehler and S. Link. Inclusion dependencies and their interaction with functional dependencies in sql. *J. Comput. Syst. Sci.*, 85(C):104–131, 2017.

[26] S. Kolahi and L. V. S. Lakshmanan. On approximating optimum repairs for functional dependency violations. ICDT, 2009.

[27] E. Livshits, B. Kimelfeld, and S. Roy. Computing optimal repairs for functional dependencies. In *Proceedings of the 37th ACM Symposium on Principles of Database Systems*, pages 225–237. ACM, 2018.

[28] A. Lopatenko and L. Bravo. Efficient approximation algorithms for repairing inconsistent databases. In *2007 IEEE 23rd international conference on data engineering*, pages 216–225. IEEE, 2007.

[29] G. L. Nemhauser and L. E. Trotter. Vertex packings: Structural properties and algorithms. *Mathematical Programming*, 8(4):232–248, 1975.

[30] K. Onak, D. Ron, M. Rosen, and R. Rubinfeld. A near-optimal sublinear-time algorithm for approximating the minimum vertex cover size. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1123–1131. Society for Industrial and Applied Mathematics, 2012.

[31] C. D. Sa, I. F. Ilyas, B. Kimelfeld, C. Ré, and T. Rekatsinas. A formal framework for probabilistic unclean databases. In *22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal*, pages 6:1–6:18, 2019.

[32] B. Salimi, L. Rodriguez, B. Howe, and D. Suciu. Interventional fairness: Causal database repair for algorithmic fairness. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, page 793–810, New York, NY, USA, 2019. Association for Computing Machinery.

[33] D. P. Williamson and D. B. Shmoys. *The design of approximation algorithms*. Cambridge University Press, Cambridge, England, 2011.

[34] W. E. Winkler. Methods for evaluating and creating data quality. *Information Systems*, 29(7):531–550, 2004.