The Cost of Representation by Subset Repairs

Yuxi Liu*
Duke University
yuxi.liu@duke.edu

Fangzhu Shen*
Duke University
fangzhu.shen@duke.edu

Kushagra Ghosh
Duke University
kushagra.ghosh@duke.edu

Amir Gilad
Hebrew University
amirg@cs.huji.ac.il

Benny Kimelfeld
Technion
bennyk@cs.technion.ac.il

Sudeepa Roy
Duke University
sudeepa@cs.duke.edu

ABSTRACT

Datasets may include errors, and specifically violations of integrity constraints, for various reasons. Standard techniques for "minimal-cost" database repairing resolve these violations by aiming for a minimum change in the data, and in the process, may sway representations of different sub-populations. For instance, the repair may end up deleting more females than males, or more tuples from a certain age group or race, due to varying levels of inconsistency in different sub-populations. Such repaired data can mislead consumers when used for analytics, and can lead to biased decisions for downstream machine learning tasks. We study the "cost of representation" in subset repairs for functional dependencies. In simple terms, we target the question of how many additional tuples have to be deleted if we want to satisfy not only the integrity constraints but also representation constraints for given sub-populations. We study the complexity of this problem and compare it with the complexity of optimal subset repairs without representations. While the problem is NP-hard in general, we give polynomial-time algorithms for special cases, and efficient heuristics for general cases. We perform a suite of experiments that show the effectiveness of our algorithms in computing or approximating the cost of representation.

PVLDB Reference Format:
Yuxi Liu, Fangzhu Shen, Kushagra Ghosh, Amir Gilad, Benny Kimelfeld, and Sudeepa Roy. The Cost of Representation by Subset Repairs. PVLDB, 18(2): 475-487, 2024.
doi:10.14778/3705829.3705860

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/louisja1/RS-repair.

1 INTRODUCTION

Real-world datasets may violate integrity constraints that are expected to hold in the dataset for various reasons such as noisy sources, imprecise extraction, integration of conflicting sources, and synthetic data generation. Among the basic data science tasks, repairing such noisy data is considered as one of the most time-consuming and important steps, as it lays the foundation for subsequent tasks that rely on high-quality data [27, 44]. Therefore, the problem of automatic data repairing has been the focus of much prior work [1, 4, 12, 18, 21, 22, 31, 38, 46]. The existing literature on data repairing typically has the following high-level goal: given a source database that violates a set of integrity constraints, find the closest database that satisfies the constraints. This problem has been studied in many settings, by varying the type of integrity constraints [6, 10, 30], changing the database in different forms [4, 10, 12, 29, 38, 41], and even relaxing it in various manners [46, 47]. Among these, a fundamental and well-studied instance of the problem is that of data repairing with tuple deletions (called *subset repair* or *S-repair*) [29, 38, 42, 43], when the integrity constraints are Functional Dependencies (FDs), e.g., a zip code cannot belong to two different cities. The aim is to find the minimum number of deletions in the input database so that the resulting database satisfies the FDs. In particular, previous work [38] has characterized the tractable and intractable cases in this setting.

Database repair may drastically sway the proportions of different populations in the data, specifically the proportions of various sensitive sub-populations. For instance, the repair may end up deleting more females than males, or more tuples from a certain age group, race, or disability status, as illustrated in the sequel in an example. This may happen simply by chance while selecting one of many feasible optimal repairs, or, due to varying levels of inconsistency in different sub-populations in the collected data, which may arise due to varying familiarity with the data collection technology, imputing varying amounts of missing data in different groups due to concerns for ageism and other biases, etc. If data is repaired agnostic to the representations, it can lead to biased decisions for downstream (e.g., ML prediction) tasks [23, 24, 50], and mislead consumers when used for analytics. Thus, it is only natural to require that the process of data repair that ensures the satisfaction of FDs, will also guarantee desired representation of different groups. Recent works have proposed ways of ensuring representation of different sub-populations (especially sensitive ones) and diversity in query results by considering different types of constraints [33, 51]. However, to our knowledge, such aspects have not been considered in the context of data repairing.

In this work, we embark on an exploration of the problem of measuring the cost of representation in data repair. In particular, we treat the representation of sub-populations on par with data consistency. Our framework allows for FDs as well as a novel type of constraint called *representation constraint (RC)*. This constraint

Proceedings of the VLDB Endowment, Vol. 18, No. 2 ISSN 2150-8097.
doi:10.14778/3705829.3705860

* Both authors contributed equally.

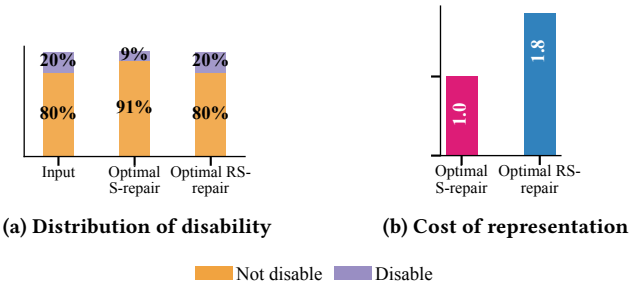


Figure 1: Disability status and cost of representation for ACS data.

specifies the proportions of the different values in a sensitive attribute (e.g., gender, race, disability status), such as "*the percentage of population with disability should be at least 20%*," "*the percentages of population with disability vs. non-disability should be exactly 20%/80%*," etc. We then formally define the problem of finding an *optimal representative S-repair (RS-repair for short)* as finding the minimum number of deletions required to satisfy *both* the FDs and the RC. We devise algorithms that consider RCs as an integral part of the repairing process and compare the cost of optimal S-repair with the cost of optimal RS-repair to understand the cost of representation that one has to pay for maintaining representations of a sensitive attribute in a dataset after repair.

Example 1. *Consider a noisy dataset constructed from the ACS PUMS, with data collected from the US demographics survey by the Census Bureau. It is expected to satisfy a set of 9 FDs:* Citizenship *to* Nativity, State *to* Division, *etc. (more details in Section 6). We focus on the sensitive attribute* Disability, *which has a* 20%/80% *distribution of disabled and non-disabled people in the dataset, where the data for the disabled group is 4-times more noisy than the non-disabled group.*

Suppose that we wish to repair the dataset by subset repair (S-repair) such that all FDs are satisfied. One can write a simple integer linear program (ILP) to find the maximum S-repair so that for each pair of tuples that violate an FD, at least one is removed. Although the ILP method finds an optimal (maximal-size) S-repair, as shown in Figure 1a, a side-effect of this repair is the drop in the proportion of people with disabilities from 20% *to* 9%*, which makes a minority group less represented further. ILP is not a scalable method for S-repair; if we were to use an efficient approximate algorithm [38, 43], no people with disability would stay in the repaired data. Consequently, both S-repairs may introduce biases against people with disability in downstream tasks that use the repaired datasets.*

The above example shows that representation-agnostic S-repair methods can badly affect representations of groups. Our aim is to answer the following question: *What would it take to repair the database by an S-repair if we insisted on the representation?* Figure 1 shows that "Optimal S-repair" retains 64.25% of the original tuples, but does not satisfy the representation, while it is possible to obtain an RS-repair that retains 34.25% of the tuples. Although retaining only 34.25% of the original tuples after the optimal RS-repair looks pessimistic, this cost is necessary in this example. In other words, for this dataset and specific noise, if we insist on preserving the 20%/80% ratio of representation among disabled and non-disabled people respectively,[1] we can retain at most 34.25% of the original tuples, and will be forced to remove 1.8 times the number of tuples than a representation-agnostic optimal S-repair. Since computing the optimal RS-repair is not always possible, in this paper, we study multiple other approaches for obtaining the RS-repairs.

Contributions. We focus on understanding the cost of representations for S-repair,[2] where the number of deleted tuples is used to measure the repair quality. Algorithms and complexity of S-repair, which is NP-hard in general, has been studied in prior work [22, 38, 41, 43]. Whenever finding an optimal (largest) S-repair is an intractable problem, so is the problem of finding an optimal RS-repair. The complexity of finding an optimal S-repair has been studied by Livshits et al. [38], who established a complete classification of complexity (dichotomy) based on the structural properties of the input FD set. We show that finding an RS-repair can be computationally hard even if the FDs are such that an optimal S-repair can be computed in polynomial time.

Next we investigate the complexity of computing an optimal RS-repair for special cases of FDs and RC. We present a polynomial-time dynamic-programming-based algorithm for a well-studied class of FDs, namely the *LHS-chains* [37, 39], when the domain of the sensitive attribute is bounded (e.g., gender, race, disability status). For the general case, we phrase the problem as an ILP, and devise heuristic algorithms that produce RS-repairs by rounding the ILP and by using the algorithm for LHS-chains.

Finally, we perform an experimental study using three real-world datasets. We demonstrate the effect of representation-agnostic S-repairs on the representations of the sensitive attribute and show that optimal RS-repairs delete 1× to 2× tuples compared to optimal S-repairs in Section 6.2, depending on how the noises are distributed. We conduct a thorough comparison between our algorithms, existing baselines, and a version of these baselines with post-hoc processing that further deletes tuples until the RC is satisfied, and analyze the quality and runtime of different approaches. We show that representation-aware subset-repair algorithms can find superior RS-repairs in terms of the number of deletions. In summary, our contributions are as follows. *(1)* We introduce the problem of finding an optimal RS-repair as a way of measuring the cost of representation. *(2)* We present a complexity analysis of the problem. *(3)* We devise algorithms, including polynomial-time and ILP algorithms with optimality guarantees, and heuristics. *and (4)* We conduct a thorough experimental study of our solutions and show their effectiveness compared to the baselines.

Due to space limitation, all proofs appear in the full version [35].

2 PRELIMINARIES

In this section, we present the background concepts and notations used in the rest of the paper.

A relation (or table) R is a set of tuples over a relation schema $\mathcal{S} = (A_1, \ldots, A_m)$, where $A_1, \ldots, A_m$ are the attributes of R. A tuple t in R maps each A_i, $1 \leq i \leq m$, to a value that we denote by $t[A_i]$.

[1] More settings varying noise and representations are given in the experiments (Section 6) and full version [35].

[2] We note that "update repair" methods, e.g., Holoclean [46] and Nadeef [14], have their own limitations or challenges when considering the cost of representations as discussed in Section 8.

This is referred to as a *cell* in R in the sequel. We use $|R|$ to denote the number of tuples in R. The domain of an attribute A_i, denoted by $Dom(A_j)$, is the range of values that A_i can be assigned to by t. Abusing notation, we denote by $t[X]$ the *projection* of tuple t over the set X of attributes, i.e., $t[X] = \pi_X t$.

A functional dependency (FD) is denoted as $X \rightarrow Y$, where X and Y are disjoint sets of attributes. X is termed the left-hand side (LHS), and Y is the right-hand side (RHS) of the FD. A relation R satisfies $X \rightarrow Y$ if every pair of tuples that agree on X also agree on Y. Formally, for every pair t_i, t_j in R, if $t_i[X] = t_j[X]$, then $t_i[Y] = t_j[Y]$. A relation R satisfies a set Δ of FDs, denoted $R \models \Delta$, if it satisfies each FD from Δ. When Δ is clear from the context, we refer to R as *clean* (respectively, *noisy*) if it satisfies (respectively, violates) Δ. Without loss of generality, we assume that the RHS Y of each FD is a single attribute, otherwise we break the FD into multiple FDs. Note that the LHS X can contain multiple attributes. We also assume that the FDs $X \rightarrow Y$ are non-trivial, i.e., $Y \notin X$.

Two types of database repairs have been mainly studied. A *subset repair (S-repair)* [22, 29, 38, 43] changes a noisy relation by removing tuples, while an *update repair* [21, 38, 46] changes values of cells. Each of the two has pros and cons. While the update repair retains the size of the dataset, it may generate invalid tuples (as discussed in Section 1). An S-repair uses original tuples, but at the cost of losing others. The complexity of computing an optimal S-repair is well understood [38], whereas the picture for update repairs is still quite partial [29, 38]. Hence, we focus on S-repairs and leave the update repairs for future work. Formally, given R and Δ, an S-repair is a subset of tuples[3] $R' \subseteq R$ such that $R' \models \Delta$. R' is called an *optimal subset repair* (or *optimal S-repair*) if for all S-repairs R'' of R given Δ, $|R'| \geq |R''|$ (there is a weighted version when there is a weight $w(t)$ associated with each input tuple t).

Computing optimal S-repairs. Livshits et al. [38] proposed an exact algorithmic characterization (dichotomy) for computing an optimal S-repair. Moreover, it showed that when a specific procedure is not able to return an answer, the problem is NP-hard for the input Δ in data complexity [53]. We briefly discuss these concepts and algorithms as we will use them in the sequel.

A *consensus FD* $\emptyset \rightarrow A$ is an FD where the LHS is the empty set, which means that all values of the attribute A must be the same in the relation. A *common LHS* of an FD set Δ is an attribute A shared by the LHS of all FDs in Δ, e.g., A is a common LHS in $\Delta = \{A \rightarrow B, AC \rightarrow D\}$. An *LHS marriage* is a pair of distinct left-hand sides (X_1, X_2) such that every FD in Δ contains either X_1 or X_2 (or both), and $cl_\Delta(X_1) = cl_\Delta(X_2)$, where $cl_\Delta(X)$ is the *closure* of X under Δ, i.e. all attributes that can be inferred starting with X using Δ. For instance, (A, B) forms a LHS marriage in $\Delta = \{A \rightarrow B, B \rightarrow A, A \rightarrow C\}$ where $cl_\Delta(A) = cl_\Delta(B) = \{A, B, C\}$.

Using the above concepts, three simplification methods for Δ are proposed, preserving the complexity of computing an optimal S-repair. This leads to a dichotomy: cases where computing an optimal S-repair is either polynomial-time solvable or NP-hard. In the polynomial-time cases, dynamic programming can be used to recursively find an optimal S-repair.

[3]We note that in prior work [38, 42], an S-repair has been defined as a "*maximal*" subset $R' \subseteq R$ such that $R' \models \Delta$. We consider even non-maximal subsets as valid S-repairs since in our problem, additional tuples from S-repairs may have to be removed to satisfy both FDs and representations.

3 REPRESENTATIVE REPAIRS

We now formally define the problem of finding an optimal RS-repair, and give an overview of complexity results and algorithms.

3.1 Representation of a Sensitive Attribute

Sensitive Attribute. Without loss of generality, we denote the last attribute A_s of $\mathcal{S}$ as the sensitive attribute.[4] We denote the domain of A_s in R as $Dom(A_s) = \{a_1, \cdots, a_k\}$, therefore $k = |Dom(A_s)|$ representing the size. A special case is when A_s is binary, i.e., the sensitive attribute has two values: (1) a minority or protected group of interest (e.g., female, people with disability, and other underrepresented groups), and (2) the non-minority group or others.

Definition 1 (Representation Constraint). *Let $\mathcal{S}$ be a schema and A_s a sensitive attribute. A* lower-bound constraint *is an expression of the form $\%a \geq p$ where a is a value and p is a number in $[0, 1]$. A* Representation Constraint *(RC) ρ is a finite set of lower-bound constraints. A relation R satisfies $\%a \geq p$ if at least $p \cdot |R|$ of the tuples $t \in R$ satisfy $t[A_s] = a$. A relation R satisfies an RC ρ, denoted $R \models \rho$, if R satisfies every lower-bound constraint in ρ.*

We assume that ρ contains only constraints $\%a \geq p$ where a is in the active domain of A_s; otherwise, for $p > 0$ the constraint is unsatisfiable by any S-repair, and for $p = 0$ the constraint is trivial and can be ignored. We also assume that ρ has no redundancy, that is, it contains at most one lower-bound $\%a \geq p$ for every value a. We can also assume that the numbers p in ρ sum up to at most one, since otherwise the constraint is, again, infeasible.

The RC ρ is called an *exact* RC if $\sum_{i=1}^{k} p_i = 1$ because the only way to satisfy the individual constraints $\%a_i \geq p_i$ is to match p_i exactly, i.e., $\%a_i = p_i$ for all $1 \leq i \leq k$. We refer to the lower-bound proportion p_ℓ of value a_ℓ in ρ by $\rho(a_\ell)$.

For simplicity, our implementation restricts attention so that each proportion (p_ℓ) in the input is a rational number, represented by an integer numerator and an integer denominator. Moreover, if one does not specify a lower-bound constraint for some a_o, then we treat it as a trivial lower-bound constraint in the form of $\%a_o \geq 0$ or formally $\rho(a_o) = p_o = 0$.

Example 2. *Suppose that a relation R with the schema $\mathcal{S} = (A_1, A_2, A_3)$ and the sensitive attribute A_3 contains four tuples $\{(1, a, 3), (2, b, 5), (3, c, 9), (4, d, 3)\}$. The RC is $\rho = \{\%3 \geq \frac{1}{3}, \%5 \geq \frac{1}{3}, \%9 \geq \frac{1}{3}\}$. R does not satisfy ρ, but both the subset $R_1 = \{(1, a, 3), (2, b, 5), (3, c, 9)\}$ and the subset $R_2 = \{(2, b, 5), (3, c, 9), (4, d, 3)\}$ satisfy it.*

Next we define an RS-repair for a set of FDs and an RC:

Definition 2 (RS-repair). *Given a relation R with the sensitive attribute A_s, a set Δ of FDs, and an RC ρ, a subset $R' \subseteq R$ is called an RS-repair (representative subset repair) w.r.t. Δ and ρ if:*

- *R' is an S-repair of R, i.e., $R' \models \Delta$,*
- *R' satisfies the RC ρ on A_s, i.e., $R' \models \rho$.*

[4]This initial study of cost of representations by subset repairs considers representations of sub-populations defined on a single sensitive attribute. Representations on a set of sensitive attributes will be an interesting future work (Section 8).

We call R' an optimal RS-repair *of R w.r.t. Δ and ρ if for all RS-repairs R'' of R, we have $|R''| \leq |R'|$.*

Example 3. *Continuing Example 2, if $\Delta = \{A_1 \rightarrow A_2\}$, then either R_1 or R_2 in can be an optimal RS-repair of R w.r.t. Δ and ρ.*

We study the problem of computing an optimal RS-repair. For our complexity analysis, we assume that $\mathcal{S}$ and Δ are fixed, and the input consists of the relation R and the RC ρ.

Choice of repair model for the cost of representation. In this initial study of repairs with representation, we consider S-repair (deletion) as the repair model. Multiple prior works have theoretically analyzed S-repairs (without the cost of representation) [22, 29, 38, 43], and the complexity of achieving an optimal S-repair based on the structure of the input FD set is well understood [38]. We focus on the framework and theoretical analysis for this simpler variant of the repair model with the representation criteria. Additionally, S-repairs keep tuples from the original dataset and do not introduce new combinations of values within the same tuple while repairing the data. We use the number of deletions to define the optimal RS-repair as an extension for defining S-repair models from prior work. Extensions to other repair models and other cost functions (e.g., based on the effects on downstream tasks) are important and challenging future work (see Section 8).

3.1.1 NP-Hardness of Computing Optimal RS-Repairs. In this section, we consider the relation R and the RC ρ as inputs, while the schema $\mathcal{S}$ and the FD set Δ are fixed. We first note that since the problem of finding an optimal RS-repair is a generalization of the problem of finding an optimal S-repair, as expected, in all cases where finding an optimal S-repair is NP-hard, finding an optimal RS-repair is also NP-hard. Theorem 1 shows that computing an optimal RS-repair is NP-hard even for a single FD. We prove this theorem by a reduction from 3-SAT (proof in [35]).

Theorem 1. *The problem of finding an optimal RS-repair is NP-hard already for $\mathcal{S} = (A, B, C)$ and $\Delta = \{A \rightarrow B\}$.*

It is important to note that if Δ contains a single FD, computing an optimal S-repair can be done in polynomial time [38]. The key distinction is on (the size of) the active domain of the sensitive attribute. Theorem 2 in the next subsection describes a tractable scenario when the active domain of the sensitive attribute is bounded.

3.2 Overview of Our Algorithms for Computing Optimal RS-Repairs

As shown by Livshits et al. [38], computing an optimal S-repair is poly-time solvable if Δ can be reduced to $\emptyset$ by repeated applications of three simplification processes: (i) consensus FDs (remove FDs of the form $\emptyset \rightarrow Y$), (ii) common LHS (remove attribute A from Δ, such that A belongs to the LHS of all FDs in Δ), and (iii) LHS marriage, which is slightly more complex. We will show that reduction to $\emptyset$ only by the first two simplification processes entails a polynomial time algorithm for computing optimal RS-repairs when the sensitive attribute A_s has a fixed number of distinct values (e.g., for common sensitive attributes gender, race, disability status, etc.). Before we formally state the theorem, we take a closer look at the class of FD sets that reduces to $\emptyset$ by the first two simplifications.

Definition 3. *An FD set Δ is an* LHS-chain *[36, 38] if for every two FDs $X_1 \rightarrow Y_1$ and $X_2 \rightarrow Y_2$, either $X_1 \subseteq X_2$ or $X_2 \subseteq X_1$ holds.*

For instance, the FD set $\Delta_1 = \{A \rightarrow B, AC \rightarrow D\}$ is an LHS-chain. LHS-chains have been studied for S-repairs in prior work [36, 38]. [36] showed that the class of LHS-chains consists of precisely the FD sets where the S-repairs can be counted in polynomial time (assuming $\mathsf{P} \neq \#\mathsf{P}$). [38] observed that FD sets that form an LHS-chain can be simplified to the empty set by repeatedly applying simplifications on only the common LHS and the consensus FD. We show in the following proposition that the converse also holds:

Proposition 4. *A set Δ of FDs reduces to the $\emptyset$ by repeated applications of consensus FD and common LHS simplifications if and only if Δ is an LHS-chain.*

The following theorem states our main algorithmic result.

Theorem 2. *Let $\mathcal{S}$ be a fixed schema and Δ be a fixed FD set that forms an LHS chain. Suppose that the domain size of the sensitive attribute A_s is fixed. Then, an optimal RS-repair can be computed in polynomial time.*

We present and analyze a dynamic programming (DP)-based algorithm LhsChain-DP(R, Δ, ρ) in Section 4 to prove the above theorem. LhsChain-DP not only gives an optimal algorithm for the special case of LHS-chains, but will also be used in Section 5 as a procedure to obtain efficient heuristics for general FD sets where computing an optimal RS-repair can be NP-hard. We give another (non-polynomial-time) optimal algorithm and several polynomial-time heuristics for cases with general FD sets in Section 5.

3.3 Can we Convert an S-repair to an RS-repair?

As discussed in the introduction for Example 1, an intuitive heuristic to compute an RS-repair is (i) first compute an S-repair R' (optimal or non-optimal) of R w.r.t. Δ, and (ii) then delete additional tuples from R' to obtain R'' that also satisfies the RC ρ. Following this idea, we present the PostClean algorithm, which takes a relation R and an RC ρ, and returns a maximum subset R' of R such that $R' \models \rho$. PostClean has a dual use in this paper. First, it is used as a subroutine in several algorithms in the later sections when an S-repair of R is used as the input relation to PostClean. Second, in Section 6, we also compose PostClean with several known approaches for computing S-repairs to create baselines for our algorithms.

Overview of the PostClean algorithm. The PostClean algorithm intuitively works as follows (pseudo-code and analysis are in [35]). Recall from Section 3.1 that $\rho(a_\ell)$ denotes the lower bound on the fraction of the value a_ℓ in the tuples retained by the RS-repair. Also recall that the sum of $\rho(a_\ell)$ may be smaller than 1, i.e., the RC ρ may only specify the desired lower bounds for a subset of the sensitive values, and the rest can have arbitrary proportions as long as a minimum set of tuples is removed to obtain the optimal RS-repair. Moreover, the fractions are computed w.r.t. the final repair size $|R'|$ and not w.r.t. the input relation size $|R|$. PostClean iterates over all possible sizes T of R' from $|R|$ to 1. For each T, it checks if the lower bound on the number of tuples with a_ℓ, i.e., $\tau_\ell = \lceil T \cdot \rho(a_\ell) \rceil$, is greater than the number of tuples with value a_ℓ in the original relation R. If yes, then no repair R' of size T can satisfy ρ, and it goes to the next value of T (or $T \leftarrow T - 1$). Otherwise, if there are

Algorithm 1 LhsChain-DP(R, Δ, ρ)

Input: a relation R, an LHS-chain FD set Δ, and a RC ρ
Output: an optimal RS-repair of (R, Δ, ρ)

1: $C_{R,\Delta} \leftarrow$ Reduce(R, Δ); ▷ Algorithm 2 in Section 4.2
2: $S \leftarrow \{\text{PostClean}(R', \rho) \mid R' \in C_{R,\Delta}\}$; ▷ Section 3.3
3: **return** $\arg\max_{s \in S} |s|$;

sufficient tuples for all sensitive values a_ℓ, and if the sum of the lower bounds on numbers, formally $T_0 = \sum \tau_\ell$ is $\leq T$, then we have a feasible T. Finally, the algorithm arbitrarily fills R' with more tuples from R if $T_0 < T$ and returns the final R'. Note that if all values of T between $|R|$ and 1 are invalid, then an $\emptyset$ is returned because it is the only subset of R that (trivially) satisfies the ρ. The following states the optimality and runtime of PostClean.

Proposition 5. *Given R and ρ, PostClean(R, ρ) returns in polynomial time a maximum subset R' of R such that $R' \models \rho$.*

Applying PostClean on an optimal S-repair may not lead to an optimal RS-repair as illustrated below (and in Example 1).

Example 6. *Consider a relation R with $\mathcal{S} = (A, B, \text{sex})$, a set Δ of FDs $\{A \rightarrow B\}$, and an exact RC $\rho = \{\%\texttt{male} = \frac{1}{2}, \%\texttt{female} = \frac{1}{2}\}$. Let $R = \{(1, a, \texttt{male}), (1, b, \texttt{female}), (2, c, \texttt{male}), (2, d, \texttt{female})\}$. An optimal S-repair of R w.r.t. Δ is $R' = \{(1, a, \texttt{male}), (2, c, \texttt{male})\}$. However, PostClean$(R', \rho)$ returns $\emptyset$ since R' does not have any female tuples. Conversely, $\{(1, a, \texttt{male}), (2, d, \texttt{female})\}$ is an optimal RS-repair, which satisfies both Δ and ρ.*

4 DYNAMIC PROGRAMMING FOR LHS-CHAINS

We now prove Theorem 2 by presenting a DP-based exact algorithm, LhsChain-DP (Algorithm 1), that finds an optimal RS-repair for LHS-chains (Section 3.2, Definition 3) in polynomial time when the sensitive attribute has a fixed domain size. By the property of an LHS-chain, Δ can be reduced to $\emptyset$ by repeated application of only consensus FD simplification and common LHS simplification.

Overview of LhsChain-DP. For a relation R and a set Δ of FDs, let $\mathcal{A}_{R,\Delta} = \{R' \subseteq R \mid R' \models \Delta\}$ be the set of all S-repairs of R for Δ. Intuitively, if we could enumerate all S-repairs R' from $\mathcal{A}_{R,\Delta}$, we could compute $R'' = \text{PostClean}(R', \rho)$ (Section 3.3) for each of them and return the R'' with the maximum number of tuples. Since PostClean optimally returns a maximum subset satisfying ρ for every R', and since any RS-repair w.r.t. Δ and ρ must be an S-repair w.r.t. Δ, such an R'' is guaranteed to be an optimal RS-repair.

However, even when the domain size of the sensitive attribute A_s is fixed, the size of $\mathcal{A}_{R,\Delta}$ can be exponential in $|R|$. Therefore, it is expensive to enumerate the set of all S-repairs. Hence, we find a *candidate set* $C_{R,\Delta} \subseteq \mathcal{A}_{R,\Delta}$ of S-repairs that is sufficient to inspect. Then, we apply PostClean to each element of $C_{R,\Delta}$, and return the final solution having the maximum size. We formally define candidate set $C_{R,\Delta}$ in Section 4.1, along with associated definitions. The basic idea is that there are no two S-repairs in $C_{R,\Delta}$ where one is "clearly better" than the other or that the two "are equivalent to each other". Further, for any S-repair that is not in the candidate set i.e., $R'' \in \mathcal{A}_{R,\Delta} \setminus C_{R,\Delta}$, there is an S-repair $R' \in C_{R,\Delta}$ that is "clearly better" or "equivalent to" R''. We prove (in Lemma 9) that an optimal RS-repair can be obtained by applying PostClean to each S-repair in $C_{R,\Delta}$ and returning the one with maximum size. Moreover, the size of the candidate set is $O(|R|^k)$, when the domain size $|Dom(A_s)| = k$ is fixed (proofs in [35]).

Algorithm 1 has two steps: Line 1 computes the candidate set $C_{R,\Delta}$ by the recursive Reduce procedure (Algorithm 2 in Section 4.2), that divides the problem into smaller sub-problems by DP. Then Line 2 applies PostClean to each S-repair in $C_{R,\Delta}$ and returns the maximum output as an optimal RS-repair in Line 3. Section 4.2 describes the Reduce procedure. Since Δ is an LHS-chain, it reduces to $\emptyset$ by repeated reductions of consensus FD (Section 4.2.1) and common LHS (Section 4.2.2). The correctness of LhsChain-DP follows from Lemmas 9 and 10 stated later.

Lemma 7. *LhsChain-DP terminates in $O(m \cdot |\Delta| \cdot k \cdot |R|^{3k+2})$ time, where m is the number of attributes in R, $|\Delta|$ is the number of FDs, and $k = |Dom(A_s)|$ is the domain size of the sensitive attribute A_s.*

4.1 Candidate Set for Optimal RS-Repairs

Recall that $\mathcal{A}_{R,\Delta}$ denotes the set of all S-repairs R w.r.t. Δ. We define a candidate set as the subset of $\mathcal{A}_{R,\Delta}$ such that every S-repair in the candidate set is neither *representatively dominated* by nor *representatively equivalent to* other S-repairs in terms of the sensitive attribute as defined below.

Definition 4. *For a relation R, FD set Δ, and $R_1, R_2 \in \mathcal{A}_{R,\Delta}$:*

- R_1 *is* representatively equivalent to R_2, *denoted* $R_1 =_{Repr} R_2$, *if for all* $a_\ell \in Dom(A_s)$, $|\sigma_{A_s=a_\ell} R_1| = |\sigma_{A_s=a_\ell} R_2|$, *i.e. the same number of tuples for each sensitive value.*
- R_1 representatively dominates R_2, *denoted* $R_1 \succ_{Repr} R_2$, *if for all* $a_\ell \in Dom(A_s)$, $|\sigma_{A_s=a_\ell} R_1| \geq |\sigma_{A_s=a_\ell} R_2|$, *and there exists* $a_c \in Dom(A_s)$, $|\sigma_{A_s=a_c} R_1| > |\sigma_{A_s=a_c} R_2|$.

Example 8. *Consider three S-repairs for the relation R with the schema $(A, B, \texttt{race})$ and FD set $\Delta = \{A \rightarrow B\}$: (1) $R'_1 = \{(1, 2, \texttt{white}), (2, 3, \texttt{black})\}$; (2) $R'_2 = \{(1, 3, \texttt{black}), (1, 3, \texttt{white})\}$; and (3) $R'_3 = \{(1, 1, \texttt{black}), (2, 2, \texttt{white}), (3, 3, \texttt{asian})\}$. Here $R'_1 =_{Repr} R'_2$ since they have the same number of* black *and* white *tuples. $R'_3 \succ_{Repr} R'_1$ and $R'_3 \succ_{Repr} R'_2$ since R'_3 has one more* asian *than R'_1 and R'_2.*

Definition 5 (candidate set). *Given a relation R and any FD set Δ, a* candidate set *denoted by $C_{R,\Delta}$ is a subset of $\mathcal{A}_{R,\Delta}$ such that*

(1) For all $R_1, R_2 \in C_{R,\Delta}$, $R_1 \neq_{Repr} R_2$, $R_1 \not\succ_{Repr} R_2$, and $R_2 \not\succ_{Repr} R_1$;
(2) For any $R'' \in \mathcal{A}_{R,\Delta} \setminus C_{R,\Delta}$, there exists $R' \in C_{R,\Delta}$ such that $R' =_{Repr} R''$ or $R' \succ_{Repr} R''$.

Each S-repair $R' \in C_{R,\Delta}$ is called a candidate.

For the correctness of LhsChain-DP, we use the following lemma.

Lemma 9. *For any relation R and any FD set Δ, if $C_{R,\Delta}$ is computed correctly in Line 1, LhsChain-DP (Algorithm 1) returns an optimal RS-repair of R w.r.t. Δ and ρ.*

ReprInsert. A subroutine ReprInsert(C, R_0) (pseudocode in [35]) will be used in the following subsections. Intuitively, it safely processes an insertion to a set of candidates and maintains the properties in Definition 5. Specifically, it takes a set of candidates C, where no representative equivalence or representative dominance exists, and an S-repair $R_0 \in \mathcal{A}_{R,\Delta}$ as inputs. ReprInsert compares

Algorithm 2 Reduce(R, Δ)

Input: a relation R, a FD set Δ that forms an LHS chain
Output: a candidate set $C_{R,\Delta}$ w.r.t. R and Δ

1: **if** Δ is empty **then**
2: **return** $C_{R,\Delta} := \{R\}$;
3: **else if** Identify a consensus FD $f : \emptyset \rightarrow Y$ **then**
4: **return** ConsensusReduction(R, Δ, f); ▷ Algorithm 3
5: **else if** Identify a common LHS X **then**
6: **return** CommonLHSReduction(R, Δ, X); ▷ Algorithm 4
7: **end if**

R_0 with every $R' \in C$. If there is an R' such that $R' \succ_{Repr} R_0$ or $R' =_{Repr} R_0$, it returns the existing C. Otherwise it removes all R' from C where $R_0 =_{Repr} R'$ and returns $C \cup \{R_0\}$.

4.2 Recursive Computation of Candidate Set

The procedure Reduce (Algorithm 2) computes a candidate set recursively when Δ is an LHS-chain. Since an LHS-chain Δ can be reduced to $\emptyset$ by repeated applications of consensus FD and common LHS, Reduce calls ConsensusReduction (Section 4.2.1) and CommonLHSReduction (Section 4.2.2) until Δ is empty. When Δ is empty, it returns $\{R\}$ as the singleton candidate set since R itself is an S-repair and representatively dominates all other S-repairs. The following lemma states the correctness of the Reduce procedure.

Lemma 10. *Given relation R and FD set Δ that forms an LHS-chain, Reduce(R, Δ) correctly computes the candidate set $C_{R,\Delta}$.*

4.2.1 Reduction for Consensus FD. Consider a consensus FD $f : \emptyset \rightarrow Y$. Within an S-repair, all values of Y should be the same. Suppose that $Dom(Y) = \{y_1, \cdots, y_n\}$ and $R_{y_\ell} = \sigma_{Y=y_\ell} R$ denotes the subset of R that has the value $Y = y_\ell$. The procedure ConsensusReduction (Algorithm 3) computes the candidate set $C_{R_{y_\ell}, \Delta - f}$ by calling Reduce$(R_{y_\ell}, \Delta - f)$ for every y_ℓ. Note that any S-repair $R' \in C_{R_{y_\ell}, \Delta - f}$ is also an S-repair of R for Δ, i.e., $R' \in \mathcal{A}_{R,\Delta}$, since the FD f is already taken care of in R_{y_ℓ}. Hence, Line 5 combines these sets from smaller problems (i.e., Reduce$(R_{y_\ell}, \Delta - f)$ for every $y_\ell \in Dom(Y)$) by inserting their candidates into $C_{R,\Delta}$ one by one so that the properties of a candidate set are maintained in $C_{R,\Delta}$.

4.2.2 Reduction for Common LHS. Consider a common LHS attribute X that appears on the LHS of all FDs in Δ. Suppose that $Dom(X) = \{x_1, x_2, \ldots, x_n\}$, $R_{x_\ell} = \sigma_{X=x_\ell} R$ denotes the subsets of R that have value $X = x_\ell$, and $R_{x_1,\ldots,x_\ell} = \sigma_{X=x_1 \vee \cdots \vee X=x_\ell} R$ as an extension. Also suppose Δ_{-X} denotes that the common LHS attribute X is removed from all FDs in Δ. If we were to consider S-repairs, we could optimally repair each R_{x_ℓ} w.r.t. Δ_{-X} independently (and recursively), and then take the union of their optimal S-repairs to obtain an optimal S-repair for R w.r.t. Δ (as done in [38]).

Yet, this is not the case for computing RS-repairs, since maximum size is not the only requirement—while we know the final repair will satisfy ρ, we do not know what the value distribution of A_s should be in each disjoint piece (e.g., some R_{x_ℓ}) of the final repair before we get one. Therefore, in each step of the recursion (either CommonLHSReduction here or ConsensusReduction above), the candidate set preserves all possible distributions of A_s from the S-repairs that could provide the final optimal solution.

Algorithm 3 ConsensusReduction(R, Δ, f)

Input: a relation R, a FD set Δ, a consensus FD $f : \emptyset \rightarrow Y$ in Δ
Output: A candidate set $C_{R,\Delta}$

1: $C_{R,\Delta} \leftarrow \emptyset$;
2: **for** each value $y_\ell \in Dom(Y)$ **do**
3: $C_{R_{y_\ell}, \Delta - f} \leftarrow$ Reduce$(R_{y_\ell}, \Delta - f)$; ▷ Algorithm 2
4: **for** all R' in $C_{R_{y_\ell}, \Delta - f}$ **do**
5: $C_{R,\Delta} \leftarrow$ ReprInsert$(C_{R,\Delta}, R')$; ▷ Section 4.1
6: **end for**
7: **end for**
8: **return** $C_{R,\Delta}$.

CommonLHSReduction (Algorithm 4) constructs the candidate set $C_{R,\Delta}$ recursively from smaller problems by building solutions cumulatively in n stages (the outer loop). In particular, after stage ℓ, the algorithm obtains the candidate set $C_{R_{x_1,\ldots,x_\ell},\Delta}$ by combining S-repairs for $R_{x_1}, \cdots, R_{x_\ell}$. Note that the union of S-repairs for $R_{x_1}, \cdots, R_{x_\ell}$ is an S-repair for $R_{x_1,\ldots,x_\ell}$ and consequently R w.r.t. Δ, but we have to ensure that the properties of a candidate set are maintained while combining these S-repairs. Line 3 computes the candidate set $C_{R_{x_\ell},\Delta_{-X}}$ recursively by calling Reduce$(R_{x_\ell}, \Delta_{-X})$. Since $C_{R_{x_1,\ldots,x_{\ell-1}},\Delta}$ is already formed in the previous stage, in Lines 4-7, it goes over all combinations of $R' \in C_{R_{x_1,\ldots,x_{\ell-1}},\Delta}$ and $R'' \in C_{R_{x_\ell},\Delta_{-X}}$, takes their union $R_0 = R' \cup R''$, and inserts it to $C_{R_{x_1,\ldots,x_\ell},\Delta}$ by ReprInsert ensuring that the property of a candidate set is maintained. Finally, $C_{R_{x_1,\ldots,x_n},\Delta}$ is returned as the final set $C_{R,\Delta}$.

5 ALGORITHMS FOR THE GENERAL CASE

Computing optimal RS-repairs for arbitrary Δ and ρ is NP-hard (Section 3.1.1). We now present a collection of end-to-end algorithms capable of handling general inputs. We begin with an exact algorithm based on integer linear programming (ILP) and then present a heuristic utilizing LP relaxation and rounding. Next, we present another heuristic using procedures from the previous section for LHS-chains as a subroutine (Section 5.2).

5.1 LP-Based Algorithms

We use $|R|$ binary random variables $x_1, x_2, \ldots, x_{|R|}$, where $x_i \in \{0, 1\}$ denotes whether tuple $t_i \in R$ is retained (if $x_i = 1$) or deleted

Algorithm 4 CommonLHSReduction(R, Δ, X)

Input: A relation R, a FD set Δ, a common LHS X for all FDs in Δ
Output: a candidate set $C_{R,\Delta}$

1: **for** $\ell = 1$ to n **do** ▷ Suppose $Dom(X) = \{x_1, x_2, \ldots, x_n\}$
2: $C_{R_{x_1,\ldots,x_\ell},\Delta} \leftarrow \emptyset$; ▷ Initialize a candidate set for Δ that only considers values $x_1, \cdots, x_\ell$ of X
3: $C_{R_{x_\ell},\Delta_{-X}} \leftarrow$ Reduce$(R_{x_\ell}, \Delta_{-X})$; ▷ Algorithm 2
4: **for** all R' in $C_{R_{x_1,\ldots,x_{\ell-1}},\Delta}$ and all R'' in $C_{R_{x_\ell},\Delta_{-X}}$ **do**
5: $R_0 \leftarrow R' \cup R''$; ▷ Combine prior and current S-repairs
6: $C_{R_{x_1,\ldots,x_\ell},\Delta} \leftarrow$ ReprInsert$(C_{R_{x_1,\ldots,x_\ell},\Delta}, R_0)$; ▷ Section 4.1
7: **end for**
8: **end for**
9: $C_{R,\Delta} \leftarrow C_{R_{x_1,\ldots,x_n},\Delta}$
10: **return** $C_{R,\Delta}$.

Algorithm 5 FDCleanser(R, Δ, ρ)

```
while Δ is not empty do
    Select the most frequent LHS column X in Δ;
    Choose one arbitrary FD f whose LHS contains X;
    R ← LhsChain-DP(R, {f}, ρ);
    Δ ← Δ − f
end while
return R;
```

(if $x_i = 0$) in the RS-repair. From the result of the following ILP we take the tuples with $x_i = 1$. We refer to this algorithm as GlobalILP.

$$\textbf{Maximize} \sum_{i \in [1,|R|]} x_i \quad \textbf{subject to:} \tag{1}$$

$$x_i + x_j \leq 1 \quad \text{for all conflicting } t_i \text{ and } t_j$$

$$\sum_{i : t_i[A_s] = a_\ell} x_i \;\geq\; p_\ell \times \sum_i x_i \quad \text{for all } a_\ell \in Dom(A_s)$$

$$x_i \in \{0, 1\} \quad \text{for all } i \in [1, |R|]$$

The objective maximizes the number of tuples retained. The first constraint ensures that the solution does not violate Δ. The following set of constraints correspond to the RC ρ, by ensuring each lower-bound constraint is satisfied, i.e. $\%a_\ell \geq p_\ell$, where $p_\ell = \rho(a_\ell)$, is satisfied for every $a_\ell \in Dom(A_s)$.

The ILP in Equation (1) can be relaxed to an LP by replacing the integrality constraints $x_i \in \{0, 1\}$ with $x_i \in [0, 1]$, for every $i \in [1, |R|]$. We propose rounding procedures to derive an integral solution from fractional x_is and refer to the heuristic as LP + ReprRounding (pseudocode and running time analysis in [35]).

Limitations. While GlobalILP provides an exact optimal solution, its scalability is limited by the size of the ILP. And ILP in general is not poly-time solvable. Each pair of tuples (t_i, t_j) that conflict on some FD introduces a constraint $x_i + x_j \leq 1$ to the ILP, therefore it can have $O(|R|^2)$ constraints, leading to a large program that does not scale. In Section 6, we show that GlobalILP finds optimal RS-repairs but does not scale to large datasets. For LP + ReprRounding, we observe in Section 6 that, even with the state-of-the-art LP solvers, solving our LP can be slow and sometimes encounters out-of-memory issues due to large number of constraints. Hence, we propose a DP-based heuristic using ideas from Section 4 to explore the possibility of avoiding solving the large LP.

5.2 FDCleanser: A DP-Based Algorithm

The combinatorial DP-based FDCleanser algorithm is motivated by the ideas behind the CommonLHSReduction and ConsensusReduction procedures in Section 4. An FD set Δ with only one FD can be reduced to the empty set using LhsChain-DP by first applying CommonLHSReduction and then ConsensusReduction, and hence is poly-time solvable by Theorem 2. FDCleanser therefore calls LhsChain-DP with one FD at a time from Δ until all FDs are taken care of. Further, it prioritizes the FD which has a most frequent LHS column X (among all the columns) in its LHS. This greedy approach may not be optimal as demonstrated empirically in Section 6.

As highlighted, Since Δ is fixed and each call to LhsChain-DP runs in polynomial time for fixed $Dom(A_s)$, FDCleanser terminates in polynomial time for any (R, Δ, ρ) where $Dom(A_s)$ is fixed. FDCleanser provides a practical and scalable heuristic approach for handling large instances of the problem of computing RS-repairs, by leveraging the efficient subroutine, LhsChain-DP. Its effectiveness and efficiency will be empirically evaluated in Section 6.

In our implementation, the heuristics described in Section 5.2 first employ ConsensusReduction and CommonLHSReduction until no feasible reductions are possible. This decomposes the problem into sub-problems. Then the heuristics are applied to the sub-problems and return a single RS-repair (and consequently a singleton candidate set) for each of them. These candidate sets are later merged during the backtracking stage of reductions. Finally, all the end-to-end algorithms will return a candidate set as what LhsChain-DP does, and rely on PostClean to ensure satisfying the RC.

6 EXPERIMENTS

In this section, we evaluate the deletion overhead to preserve representation in subset repairs, and the quality and performance of our algorithms. In particular, we study the following questions:

(1) Section 6.2: How many additional tuple deletions are required (i.e., deletion overhead) for computing optimal RS-repairs compared to computing optimal S-repairs?
(2) Section 6.3: How effective is each algorithm in minimizing tuple deletions compared to an optimal RS-repair algorithm (i.e., RS-repair quality)?
(3) Section 6.4: What is the runtime cost of our algorithms?
(4) Section 6.5: What is the impact of considering non-exact RCs on the number of deletions, i.e., $\%a \geq p$ instead of $\%a = p$?

Summary of findings. The experiments show the following. First, the deletion overhead is high when the noise distribution is imbalanced in the input (more noise in one subgroup), especially when the under-represented group defined by the sensitive attribute has relatively more noise than the majority group. Second, the DP-based algorithms proposed in this paper, FDCleanser for general FD sets (Section 5), and LhsChain-DP to which the former reduces to for chain FD sets (Section 4), present the best trade-off of high RS-repair quality and runtime compared to the other alternatives that we have examined. Third, as the constraint on exact RC is relaxed, fewer deletions are needed.

6.1 Setup

We implement[5] our algorithms in Python 3.11 and experiment on a machine with a commodity EPYC CPU (AMD EPYC 7R13 48-Core Processor @2.6GHz, Boost 3.73GHz, 192MB L3 cache; 256GiB DDR4-3200 memory). We use Gurobi[25] as the LP/ILP solver.

6.1.1 Datasets. We use samples of different sizes from three real datasets (ACS, COMPAS, and Flight) that are commonly used for the study of fairness or data repair. For ACS and COMPAS, the noise is injected (Section 6.1.4), whereas for the Flight dataset, the noise is real and inherent to the dataset. Additional experiments with a fourth dataset on Credit Card Transactions [9] appear in the full version [35] due to space limitations.

[5] Code publicly available at https://github.com/louisja1/RS-repair.

Table 1: FD sets used in experiments.

Dataset	Chain FD Set	Non-chain FD Set
ACS	{ST → DIV, DIV → Region}	{CIT → Nativity,ST → DIV, DIV → Region, POBP → WAOB,RAC2P → RAC1P}
COMPAS	{DecileScore → ScoreText}	{DecileScore → ScoreText, ScaleID → DisplayText,RSL → RSLT, DisplayText → ScaleID,RSLT → RSL, FirstName, LastName, DOB → Sex}
Flight	**Chain FD Set**	
	{DF → ActualDeparture, DF → ActualArrival, DF → ScheduledDeparture, DF → ScheduledArrival}	

- ACS-PUMS (in short ACS) [15]: The American Community Survey Public Use Microdata Sample dataset. We use 9 attributes and samples from 2K to 1.5M in our experiments (described below).
- COMPAS [32]: The ProPublica COMPAS recidivism dataset. We use 10 attributes and samples from 4K to 30K in our experiments.
- Flight [5, 34][6]: The Flight data contains information on scheduled and actual departure time and arrival time from different sources, and has been used in prior work of data fusion [34] and data cleaning [5, 46]. We use 6 attributes and samples from 2K to 8K.

6.1.2 Functional dependencies. We consider two types of FD sets in our experiments: (1) chain FD sets (i.e., LHS-chains, Definition 3, Section 4), and (2) non-chain FD sets (Section 5). For ACS and COMPAS, FDs are inferred from the data description documents and verified in the original clean datasets [7]. The FD sets used in all three datasets are shown in Table 1.[7] For example, the FD ST → DIV for ACS implies that each state has a unique division. For the Flight dataset, FDs are from prior work [5]. For ACS and COMPAS, the set of FDs is a non-chain, and we also use a subset of the FDs that forms a chain for these two datasets. For Flight, the set of FDs forms a chain, so only the chain FD set is considered.

6.1.3 Sensitive attribute selection. For the ACS dataset, we consider Nativity as the sensitive attribute (with the values `Native-born` and `Foreign-born`) for the non-chain FD set, and Region (with the values `Region-1-2` and `Region-3-4`) for the chain FD set, so that the sensitive attribute is always included in the FD set. (Nativity is not in the chain FD set.) For the COMPAS dataset, we use Sex as the sensitive attribute with values `Male` and `Female` that appear in the dataset. For the Flight dataset, there is no natural sensitive attribute so we choose the Source of the data as the sensitive attribute with two values `wunderground` and `flightview`. We use binary values of the sensitive attribute in our experiments since we vary both the data and noise distribution of two groups (Section 6.1.4), but our algorithms can handle multiple values of the sensitive attribute.

6.1.4 Obtaining noisy input data. The Flight dataset has violations of the FD, so we only generate uniform random samples of 2K, 4K, 6K, and 8K tuples as the noisy input data. These samples have slightly different value distributions of Source, ranging from {%`wunderground` = 58%, %`flightview` = 42%} to {%`wunderground` = 54%, %`flightview` = 46%}.

[6]We download and use the version of data from Boeckling and Bronselaer [5].

[7]ST, DIV, CIT, RAC1P, RAC2P, POBP, and WAOB are in short for state code, division, citizenship status, race code 1, race code 2, place of birth, world area of birth respectively for dataset ACS. DOB, RSL, and RSLT are short for date of birth, recommended level of supervision, and its text respectively for dataset COMPAS. DF is in short for date collected + flight number for dataset Flight.

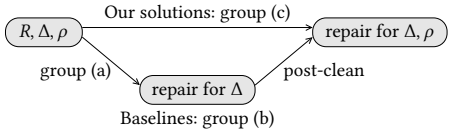


Figure 2: Our solutions vs. baselines.

The ACS and COMPAS datasets satisfy their FDs, and we inject random noise (FD violations) similarly to prior work [14, 21, 46]. We vary two parameters: (1) *value distribution of the sensitive attribute*, and (2) *relative noise distribution of the two groups defined by the sensitive attribute*, as we describe next.

(1) Value distribution of the sensitive attribute for ACS and COMPAS: We consider binary values of their sensitive attributes denoted by `Group-1` and `Group-2`. The notation "X%-Y%" denotes the value distribution of these two groups, i.e., $\frac{\%\text{no. of tuples from } \texttt{Group-1}}{\%\text{no. of tuples from } \texttt{Group-2}} = \frac{X}{Y}$ where $X + Y = 100$. We generate stratified samples for these two groups in the ACS and COMPAS datasets where the percentages "X%-Y%" are varied as "80%-20%", "60%-40%" to "50%-50%" for different sizes of the datasets. Therefore, `Group-1` is called the majority group (`Native-born, Region-1-2, Male` in Section 6.1.3) having possibly higher percentage of tuples, and `Group-2` (`Foreign-born, Region-3-4, Female` in Section 6.1.3) is called the minority group.

(2) Relative noise distribution of the sensitive attribute in ACS and COMPAS: First, we choose an overall noise level. We only change the values of the attributes that appear in the LHS or RHS of FDs (Table 1). x% noise level means that x% of the cells (attribute values) belonging to these attributes from all tuples in the data generated in the previous step have been changed to another value from the domain of the corresponding attribute. Then, we choose a relative noise distribution of the values `Group-1` and `Group-2` of the binary sensitive attribute. The distribution "x%-y%" means that $\frac{\%\text{noise level of } \texttt{Group-1}}{\%\text{noise level of } \texttt{Group-2}} = \frac{x}{y}$, where $x + y = 100$. The values of x%-y% are chosen from "20%-80%", "40%-60%", "50%-50%", "60%-40%", "80%-20%" to study different levels of noise in the majority and the minority groups (Section 6.2). For example, "20%-80%" means that the noise level of `Group-2` (the minority group, e.g., `Foreign-born`, `Female`, etc.) is four times that of `Group-1` (the majority group, e.g., `Native-born`, `Male`, etc.). Since the sensitive attribute values are changed, the data distribution in the previous step of the majority and the minority group may change following the noise injection.

6.1.5 Representation constraints. We use exact RCs in Sections 6.2 to 6.4 to preserve the original distribution of the two groups of the sensitive attribute, and examine RCs with inequality in Section 6.5 to preserve the original distribution of the minority group.

6.1.6 Algorithms. We classify the repair algorithms into three groups (Figure 2): (a) S-repairs, (b) S-repairs with PostClean (Section 3.3) to satisfy the RC, and (c) RS-repair algorithms from Sections 4 and 5. Group (a) includes the following:

- ILP-Baseline [40] (for both chain and non-chain FD sets): Formulates an ILP (with only the first constraint for FD in Equation (1)) and computes the *optimal* S-repair satisfying a given FD set.
- VC-approx-Baseline [2] (for non-chain FD sets only): An 2-approximation algorithm that translates the problem to finding a minimum vertex cover of the conflict graph.

Table 2: Deletion overhead for varying representations (80%-20% and 50%-50%) and relative noise distributions (ACS data, 5% overall noise, 10k tuples). "S-rep. %" and "RS-rep. %" stand for deletion percentage of S-repair and and for RS-repair, respectively.

(a) Chain FD set: 80%-20%

noise	del. ratio	S-rep. %	RS-rep. %
20%-80%	2.112	13.52	28.55
40%-60%	1.254	13.88	17.40
50%-50%	1.003	13.89	13.92
60%-40%	1.044	13.91	14.52
80%-20%	1.134	13.87	15.73

(b) Chain FD set: 50%-50%

noise (%)	del. ratio	S-rep. %	RS-rep. %
20%-80%	1.514	13.60	20.70
40%-60%	1.152	13.80	15.90
50%-50%	1.004	13.90	13.90
60%-40%	1.153	13.90	16.00
80%-20%	1.511	13.60	20.60

(c) Non-chain FD set: 80%-20%

noise (%)	del. ratio	S-rep. %	RS-rep. %
20%-80%	2.039	25.91	52.83
40%-60%	1.197	28.65	34.30
50%-50%	1.026	28.81	29.55
60%-40%	1.009	28.71	28.97
80%-20%	1.199	28.02	30.80

(d) Non-chain FD set: 50%-50%

noise (%)	del. ratio	S-rep. %	RS-rep. %
20%-80%	1.423	28.63	40.74
40%-60%	1.091	30.02	32.75
50%-50%	1.005	29.94	30.08
60%-40%	1.071	29.56	31.64
80%-20%	1.395	27.97	39.02

- DP-Baseline [38] (for chain FD sets only): A dynamic programming (DP) algorithm that computes the *optimal* S-repair in polynomial time when the FD set forms an LHS-chain.
- MuSe-Baseline [22] (for both chain and non-chain FD sets): An S-repair framework with deletion rules. We use step semantics.

Group (b) is defined as using PostClean to post-process the S-repairs obtained by the Group (a) approaches. These baselines are denoted as baseline+PostClean, e.g. ILP-Baseline + PostClean. Group (c) in Figure 2 consists of the algorithms proposed in Sections 4 and 5, which incorporate the RC in their core procedures.

For chain FD sets, we examine the polynomial-time optimal algorithm LhsChain-DP (Section 4). For non-chain FD sets, we examine the end-to-end heuristic algorithms LP + ReprRounding and FDCleanser (Section 5). Recall that when applied to chain FD sets, FDCleanser reduces to LhsChain-DP. We repeat each experiment twice for each data point and take the average.

6.2 Deletion Overhead of RS-repairs

In this section, we examine the number of additional deletions required to satisfy the representation of the sensitive attribute. We do so by defining *deletion overhead* as follows:

$$\text{deletion overhead} = \frac{\#_{\text{del}}(\text{an optimal RS-repair of } R \text{ w.r.t. } \Delta \text{ and } \rho)}{\#_{\text{del}}(\text{an optimal S-repair of } R \text{ w.r.t. } \Delta)} \quad (2)$$

Here $\#_{\text{del}}()$ is the number of tuple deletions of a repair. This ratio quantifies the additional deletions required by an optimal RS-repair compared to an optimal S-repair for the same (R, Δ). This ratio is at least 1 as an optimal RS-repair may have to delete more tuples also to satisfy ρ. A large ratio indicates a high overhead, i.e., many extra deletions are needed to satisfy ρ, whereas a ratio close to 1 indicates that an optimal S-repair is likely to satisfy ρ. We use ILP-Baseline and GlobalILP for optimal S-repairs and RS-repairs respectively.

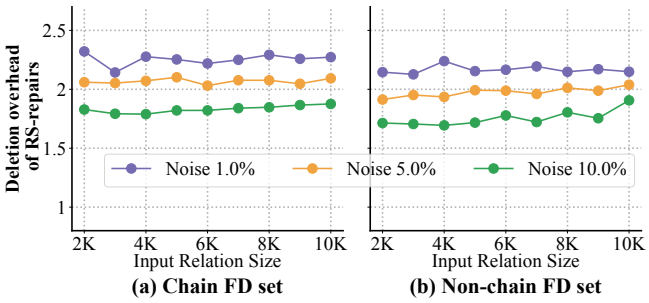


Figure 3: Deletion overhead varying overall noise and input size (ACS, 80%-20% value distr., 20%-80% relative noise distr.).

Table 3: Deletion overhead and value distribution of sensitive attribute for Flight data with real noise.

Size	Deletion overhead			Value distribution		
	del. ratio	S-rep. %	RS-rep. %	Original	S-rep.	RS-rep.
2K	1.005	47.25	47.50	58%-42%	**69%-31%**	58%-42%
4K	1.006	48.20	48.50	55%-45%	**61%-39%**	55%-45%
6K	1.035	48.30	50.00	54%-46%	**57%-43%**	54%-46%
8K	1.034	48.36	50.00	54%-46%	**65%-35%**	54%-46%

First, we examine the deletion overhead in various scenarios for ACS in Table 2. (Results for COMPAS appear in full version [35] due to space limitations.) Table 2 also shows the deletion overhead and the percentage of tuples deleted by the optimal S-repair and optimal RS-repair for ACS for both chain and non-chain FD sets (5% overall noise and 10K tuples). We vary the data distribution of the majority and minority groups as 80%-20% and 50%-50%, and for each vary their relative noise distribution 20%-80%, 40%-60%, 50%-50%, 60%-40%, 80%-20%, investigating the cases when the minority group has more, equal, and less noise than the majority group.

We observe the following. (1) When noise is uniform (50%-50%), irrespective of the data distribution in Tables 2a to 2d, RS-repair does not delete many additional tuples, hence the deletion overhead is close to 1. As the noise distribution becomes unbalanced, deletion overhead increases as more tuples are deleted for representation. (2) In Tables 2a and 2c, when the data distribution of the majority and minority groups is 80%-20%, the overhead for relative noise distribution 20%-80% is larger than the overhead for 80%-20%. Intuitively, the fraction of noisy tuples from the minority group is much larger requiring more deletions from the minority group compared to the majority group. Hence, many tuples from the majority groups need to be removed to get the data distribution back to 80%-20% in the optimal RS-repair. (3) Although both chain and non-chain FD sets show similar trends on the deletion overhead, the deletion percentages of tuples (S-rep.% and RS-rep%) are lower for the chain FD sets. This is because a lower number of attributes is involved in the chain FD set (Table 1), so less noise is injected.

Next, in Figure 3, we vary the overall noise level (1%, 5%, 10%) and the input size (2K to 10K) for ACS data with 80%-20% value distribution and 20%-80% relative noise distribution. The relation size does not significantly impact the deletion overhead of RS-repairs. However, higher noise levels lead to lower deletion overhead, since as noise increases, S-repair tends to delete more tuples from both majority and minority groups, almost matching the deletions by RS-repair maintaining their representation, which reduces the ratio.

Table 3 presents the deletion overhead and percentage for the optimal S-repair and RS-repair on Flight, alongside the value distribution of the majority and minority groups before and after repair.

Table 4: Value distributions by optimal S-repairs varying overall noise (ACS, 6K tuples, 20%-80% relative noise distr.).

FD Set	Original	Overall noise level			
		1%	5%	10%	15%
Chain FD set	80%-20%	81%-19%	84%-16%	88%-11%	92%-8%
Non-chain FD set	80%-20%	82%-18%	89%-11%	96%-4%	100%-0%

(1) Across input sizes (2K to 8K), the deletion overhead remains slightly above 1, indicating significant tuple deletions in both S-repair and RS-repair. (2) Notably, while S-repair does not preserve the original value distribution and disproportionately deletes from the minority group, RS-repair maintains the distribution when the representation constraint is applied.

In Table 4, we report the value distributions of the majority and minority groups after optimal S-repair varying the noise level for ACS data with 80%-20% value and 20%-80% relative noise distribution (one example is in Example 1). At 15% noise level, the minority group drops from 20% to 8% for chain FDs, and to 0% for non-chain FDs. This shows that S-repairs may significantly change the representation of a sensitive attribute and highlights the importance of considering the representation constraint in the repair algorithms.

6.3 RS-Repair Quality of Various Approaches

The algorithm GlobalILP computes an optimal RS-repair, but does not scale (e.g., it took 18 hours to repair ACS data for the non-chain FD set with 10% noise and 10K tuples). In this section, we report the *RS-repair quality* (*repair quality* in short) defined below to assess how our proposed algorithms perform compared to the optimal RS-repair. The experiments are conducted on relatively small datasets where GlobalILP terminated in a reasonable time.

$$\text{repair quality} = \frac{|R| - \#_{\text{del}}(R')}{|R| - \#_{\text{del}}(\text{an optimal RS-repair of } R \text{ w.r.t. } \Delta \text{ and } \rho)} \quad (3)$$

The expression compares the number of tuples retained by an RS-repair R' to that of an optimal RS-repair (via GlobalILP), indicating how well R' approximates the optimum. This quality is upper-bounded by 100%, and a ratio close to 100% indicates high quality when the output of an RS-repair algorithm is close to optimal.

In Table 5, we examine the repair quality for varying value distributions of the sensitive attribute and FD types for ACS and COMPAS data. Since there are many potential scenarios, we present a set of results (more results are in the full version [35]) for overall 10% noise level, value distributions of the majority and minority groups X%-Y% ("80%-20%", "60%-40%", and "50%-50%") with relative noise distributions Y%-X% (i.e., the same number of cells are changed in both majority and minority groups). The results by FDCleanser proposed in Section 5.2 for general FD sets, which is identical to the optimal LhsChain-DP in Section 4, are **boldfaced** in all tables. The repair quality by other algorithms that are better than FDCleanser (non-chain) or LhsChain-DP (chain) is also boldfaced.

We conclude the following. (1) LhsChain-DP (=FDCleanser) is optimal for chain FDs, so has 100% repair quality in all settings for chain FDs in ACS and COMPAS (Table 5 (a) and (c)). (2) For non-chain FDs, FDCleanser has consistently high repair quality (> 94%) in all settings for both ACS and COMPAS (Table 5 (b) and (d)). (3) For the baselines that apply PostClean after an S-repair is obtained (i.e., group (b) in Figure 2), repair quality significantly varies in different scenarios. They perform relatively better for chain FD sets where LhsChain-DP already gives optimal results (MuSe-Baseline+PostClean exceeded 12 hours for larger data denoted by "-"), but may give poor quality for non-chain FD sets. For both chain and non-chain FDs, their quality mostly degrades when the value distribution of the sensitive attribute is imbalanced. While ILP-Baseline+PostClean has better quality than FDCleanser in some non-chain FD settings in Table 5 (b) and (d), ILP is not a scalable method (Section 6.4). The efficient approximation VC-approx-Baseline+PostClean has poor quality. (4) The other LP-rounding-based algorithm LP + ReprRounding from Section 5.1 also performs better than FDCleanser in some settings in Table 5 (b) and (d), especially when the value distribution of the sensitive attribute is close to 50%-50%, but is not scalable (Section 6.4). Table 5 shows that FDCleanser gives high quality across datasets and settings with better scalability as shown in Section 6.4.

Table 6 presents the repair quality for the Flight dataset. GlobalILP and LhsChain-DP provide 100% repair quality consistently as expected, while the baselines DP-Baseline + PostClean and ILP-Baseline+PostClean have a significantly lower quality across all input sizes (e.g., for $8K$, the quality of DP-Baseline+PostClean is 3.5% and the quality of ILP-Baseline+ PostClean is 60%).

6.4 Running Time Analysis

Comparison of runtime of different RS-repair methods: First, we evaluate the runtime performance of the RS-repair algorithms from groups (b) and (c) in Figure 2. Figure 5 shows the runtime results of the ACS and COMPAS for 80%-20% value distribution of the sensitive attribute and 10% overall noise level (runtime for more settings and Flight are in the full version [35]). If no data points are shown, execution took longer than 12 hours. (1) The optimal GlobalILP for RS-repair has a high runtime in all settings as ILP is not scalable. (2) For chain FD sets in Figures 5a and 5c the optimal LhsChain-DP (= FDCleanser) is much faster than GlobalILP, e.g., GlobalILP takes 1, 046s for size 10K ACS data, while LhsChain-DP takes only 12s, (3) For non-chain FD sets in Figures 5b and 5d, FDCleanser has a good scalability. While VC-approx-Baseline+PostClean has a slightly better runtime than FDCleanser in Figure 5c, Table 5 shows its poor quality. Additionally, for COMPAS and non-chain FD set () in size 30K due to a lagged rounding step.

Scalability for bigger data: Since ILP is known to be unscalable, in Table 7 we evaluate the scalability of the other methods over bigger samples up to 1.5M tuples of the ACS dataset. We see that FDCleanser for non-chain FD sets, which is identical to the optimal LhsChain-DP for chain FD sets, can repair larger datasets, while the other methods face out-of-memory issues. While DP-Baseline+PostClean has much better scalability for chain FD sets, as discussed in Section 6.3, it suffers from poor quality. For the chain FD set, LhsChain-DP provides the optimal RS-repair in 8 hours for 1M tuples and in 48 hours for 1.5M tuples. For the non-chain FD set, FDCleanser takes 5.9 hours for 1M tuples, and 34 hours for 1.5M tuples. Although it takes a long time to repair the data, it returns results, which may be permissible for offline data repair tasks. In contrast, VC-approx-Baseline + PostClean and LP + ReprRounding encounter issues with memory usage over 100K tuples, which is as expected given that the number of constraints can be as large as $|R|^2$. Yet, both LhsChain-DP and FDCleanser

Table 5: Repair quality of different algorithms for ACS and COMPAS data (10% noise level) with chain and non-chain FD sets. Numbers are boldfaced if they are from LhsChain-DP, FDCleanser, or other algorithms that beat them in the same setup.

(a) ACS: chain FD set

Sensitive attribute distribution	80%-20%				60%-40%				50%-50%			
Algorithm / Size (K)	**4**	**6**	**8**	**10**	**4**	**6**	**8**	**10**	**4**	**6**	**8**	**10**
GlobalILP	100	100	100	100	100	100	100	100	100	100	100	100
LhsChain-DP (= FDCleanser)	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
ILP-Baseline+PostClean	76.54	79.57	80.60	80.90	96.95	97.60	97.52	97.63	99.79	99.93	99.98	99.93
DP-Baseline+PostClean	76.54	79.57	80.60	80.90	96.95	97.60	97.52	97.63	99.79	99.93	99.98	99.93
MuSe-Baseline+PostClean	77.80	80.64	-	-	96.68	65.54	-	-	99.66	99.91	-	-

(b) ACS: non-chain FD set

Sensitive attribute distribution	80%-20%				60%-40%				50%-50%			
Algorithm / Size (K)	**4**	**6**	**8**	**10**	**4**	**6**	**8**	**10**	**4**	**6**	**8**	**10**
GlobalILP	100	100	100	100	100	100	100	100	100	100	100	100
FDCleanser	**94.25**	**95.41**	**96.19**	**99.14**	**94.45**	**96.05**	**96.12**	**95.60**	**96.45**	**97.22**	**96.65**	**97.83**
LP + ReprRounding	64.94	66.39	69.25	79.14	81.33	82.26	75.33	87.75	**98.15**	90.74	90.21	84.29
ILP-Baseline+PostClean	29.60	45.57	46.78	73.62	84.03	86.03	85.79	85.79	95.7	96.30	**97.57**	**98.10**
VC-approx-Baseline+PostClean	0	0	0	0	5.00	4.14	3.81	4.51	18.15	15.73	13.93	15.11

(c) COMPAS: chain FD set

Sensitive attribute distribution	80%-20%				60%-40%				50%-50%		
Algorithm / Size (K)	**4**	**10**	**20**	**30**	**4**	**10**	**20**	**30**	**4**	**10**	**20**
GlobalILP	100	100	100	100	100	100	100	100	100	100	100
LhsChain-DP (= FDCleanser)	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
ILP-Baseline+PostClean	97.96	98.60	99.19	99.01	99.92	100	100	100	100	100	100
DP-Baseline+PostClean	97.96	98.60	99.19	99.01	99.92	100	100	100	100	100	100
MuSe-Baseline+PostClean	98.93	-	-	-	-	-	-	-	-	-	-

(d) COMPAS: non-chain FD set

Sensitive attribute distribution	80%-20%				60%-40%				50%-50%		
Algorithm / Size (K)	**4**	**10**	**20**	**30**	**4**	**10**	**20**	**30**	**4**	**10**	**20**
GlobalILP	100	100	100	100	100	100	100	100	100	100	100
FDCleanser	**98.94**	**94.21**	**97.39**	**96.45**	**95.99**	**96.90**	**97.39**	**96.37**	**97.11**	**99.24**	**99.19**
LP + ReprRounding	98.58	**97.92**	**98.04**	**98.44**	**99.61**	**99.54**	**98.25**	**98.02**	**100**	**100**	**100**
ILP-Baseline+PostClean	96.81	**94.36**	90.01	85.36	**99.10**	**98.32**	96.95	95.86	**99.55**	99.17	**99.28**
VC-approx-Baseline+PostClean	26.24	25.96	22.81	20.85	9.95	9.81	8.73	9.01	11.12	10.40	8.91

Table 6: Repair quality for Flight data (chain FD set).

Algorithm / Size (K)	2	4	6	8
GlobalILP	100	100	100	100
LhsChain-DP(= FDCleanser)	**100**	**100**	**100**	**100**
DP-Baseline+PostClean	57.14	77.67	66.67	2.93
ILP-Baseline+PostClean	52.38	70.87	66.67	50.00

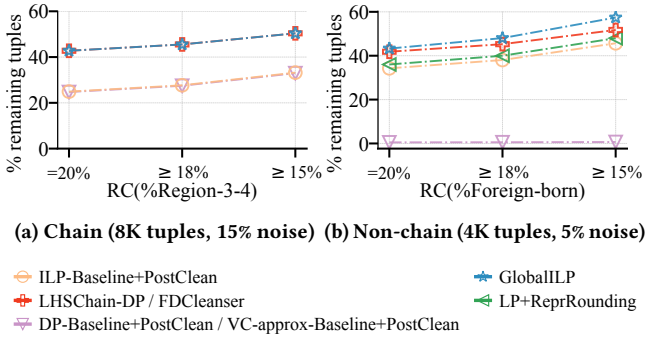
are DP-based algorithms that require significant space and do not scale well. Developing more space- and time-efficient algorithms for RS-repair remains a promising direction for future research.

6.5 Varying Representation Constraints

In this section, we vary the RCs to study the effect on the number of tuples remaining in the database after repair in ACS data. We relax the RC by gradually reducing the proportion of the minority group from the original "= 20%" to "≥ 18%" and " ≥ 15%."

Table 7: Runtime (mins) for ACS (5% overall noise, 80%-20% value distr. and 20%-80% relative noise distr.). "OOM" = out-of-memory issues.

Chain FD set					
Name/DB Size	10K	100K	500K	1M	1.5M
DP-Baseline+PostClean	0.01	0.02	0.07	0.14	0.20
LhsChain-DP	0.14	3.52	84.76	**479.50**	**2873.09**
Non-chain FD set					
Name/DB Size	10K	100K	500K	1M	1.5M
VC-approx-Baseline+PostClean	0.29	OOM	OOM	OOM	OOM
LP + ReprRounding	3.18	OOM	OOM	OOM	OOM
FDCleanser	0.15	10.56	142.80	**351.79**	**2049.64**



(a) Chain (8K tuples, 15% noise) (b) Non-chain (4K tuples, 5% noise)

Figure 4: Percentage of remaining tuples varying the RC (ACS, 80%-20% value distr. and 20%-80% relative noise distr.).

Figure 4 shows that for both chain and non-chain FD sets, the percentage of remaining tuples increases as the constraint is relaxed. The percentage of tuples deleted for non-chain FDs is high even for 5% noise, since the FDs have 9 attributes (Table 1), possibly changing a large number of tuples in noise injection (Section 6.1.4), and the 20%-80% relative noise has a high cost of RS-repair (Section 6.2).

7 RELATED WORK

Data repairing and constraints. Past work on data repairing [1, 4, 12, 18, 46] aimed at minimizing the number of changes in the database in terms of tuple deletions [1, 10, 22, 38, 41, 43] and value

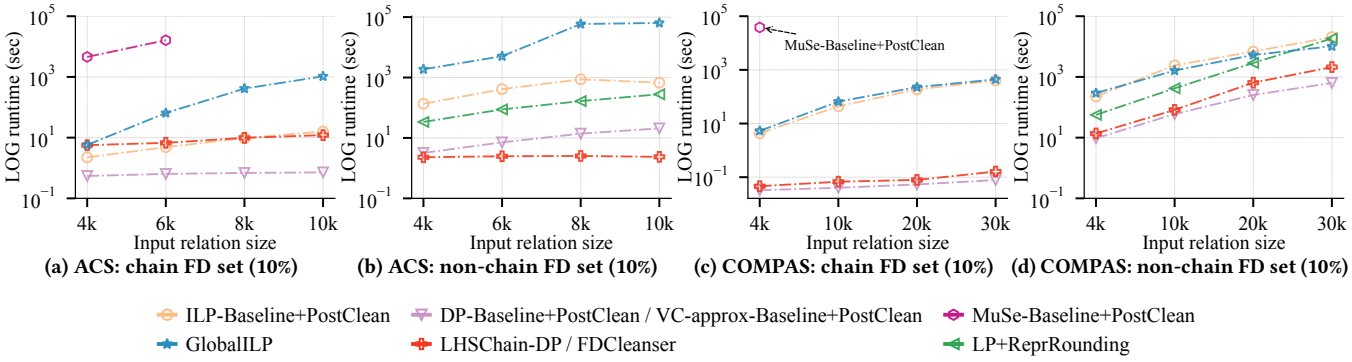


Figure 5: Runtimes for ACS and COMPAS data (10% noise level, 80%-20% value distribution) with chain and non-chain FD sets.

updates [4, 12, 14, 21, 29, 46]. The complexity of the former is better understood [38]. We intend to study extensions of our approach to the value-change model as well in future work. The literature considered a wide variety of constraints, such as FDs and versions thereof [6, 30], denial constraints [10], tuple-generating dependencies [17, 19], and equality-generating dependencies [3]. Different from these, our representation constraints consider the statistical proportions of an attribute *over the entire database*.

Representation constraints. Various types of constraints involving probability distributions on the data have been proposed in the literature. This vein of literature focuses on fairness toward specific downstream tasks. A predominant part of that work focuses on *algorithmic fairness* [8, 20, 45, 48, 49]: given a classification algorithm and sensitive attributes, determine whether the classification satisfies some definition of *fairness* [13, 26]. They consider the outcome or predicted attribute in the data and some sensitive attributes (like race or sex), and aim to maintain some equality of conditional probabilities for different values of these attributes. Several fairness definitions can be expressed as comparisons of conditional probabilities, e.g., demographic parity [16, 28] and true positive rate balance [11, 52]. Our hypothesis here is that maintaining representation while repairing the data is a precursor to reducing biases in all data-dependent tasks. Yet, capturing some fairness definitions using complex RCs is an intriguing subject for future work.

8 DISCUSSION AND FUTURE WORK

We proposed a framework for estimating the cost of representation for S-repairs—how many extra tuples have to be deleted to satisfy the FDs as well as a representation constraint (RC) on a sensitive attribute. We studied the complexity of computing an optimal RS-repair, presented polynomial-time algorithms for FD sets that form LHS-chains for a bounded domain of the sensitive attribute, devised efficient heuristics for the general cases, and evaluated them experimentally. This is an initial study of data repair with representation, and there are many interesting future directions.

First, one can study the complexity and algorithms for generalization of representations to *multiple sensitive attributes*. While the problem is still NP-hard, a closer study of the complexity, algorithms (e.g., the PostClean process), and practical performance would be beneficial. We have a detailed discussion on multiple sensitive attributes in the full version [35].

Second, it will be interesting to study the cost of representations for other repair models, and in particular, for update repairs. Update repair cleans the data by updating values of some cells (attributes of tuples), which preserves the original data size unlike S-repair. However, update repair approaches also have their own challenges with or without representations. In Example 1 discussed in the introduction, a popular update repair method Holoclean [46] does not make any changes in the data, and therefore does not eliminate any violations, since it treats the FDs as soft constraints while preserving the data distribution. On the other hand, another update repair method Nadeef [14] provides a repair satisfying the FDs, and since the sensitive attribute Disability does not participate in the FDs, it preserves its representations. However, in our experiment, Nadeef changed a `(Asian, foreign-born)` person to `(White, native)` while keeping the other attribute values the same `(female, born in Philippines, lives in CA, DISABLE, ···)`. This is not faithful to the reality (People born in Philippines should be "`foreign born`" not "`native`" and are likely to be Asian (not White)). Data repair under updates might be inherently connected to preserving distributions of multiple attributes discussed above.

Third, this paper considers the cost of repair preserving representation on the data without considering any tasks where the data is used. A natural extension of our model and future work is to consider the (still task-independent) weighted cost of tuple deletion where different tuples have different weights or different costs, which can be helpful to the downstream tasks after repairing. A more challenging future work is to study both the cost and effect of data repair with and without representation on *downstream tasks*, e.g., when the repaired data is used for ML-based tasks like predictions and classification. It will be interesting to see the implication of preserving representations in the input dataset to preserve fairness of the ML tasks for different sub-populations.

ACKNOWLEDGMENTS

The work of Amir Gilad was supported by the Israel Science Foundation (ISF) under grant 1702/24 and the Alon Scholarship. The work of Benny Kimelfeld was supported by the Israel Science Foundation (ISF) under grant 768/19 and the German Research Foundation (DFG) under grant KI 2348/1-1. This work of Sudeepa Roy was partially supported by NSF awards IIS-1552538, IIS-1703431, IIS-2008107, and IIS-2147061.

REFERENCES

[1] Foto N. Afrati and Phokion G. Kolaitis. 2009. Repair Checking in Inconsistent Databases: Algorithms and Complexity. In *ICDT*. 31–41.

[2] R Bar-Yehuda and S Even. 1981. A linear-time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms* 2, 2 (1981), 198–203. https://doi.org/10.1016/0196-6774(81)90020-1

[3] Catriel Beeri and Moshe Y. Vardi. 1984. Formal Systems for Tuple and Equality Generating Dependencies. *SIAM J. Comput.* 13, 1 (1984), 76–98.

[4] Leopoldo E. Bertossi, Solmaz Kolahi, and Laks V. S. Lakshmanan. 2013. Data Cleaning and Query Answering with Matching Dependencies and Matching Functions. *Theory Comput. Syst.* 52, 3 (2013), 441–482.

[5] Toon Boeckling and Antoon Bronselaer. 2024. Cleaning data with Swipe. *arXiv preprint arXiv:2403.19378* (2024).

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

[7] US Census Bureau. [n.d.]. *PUMS Documentation*. https://www.census.gov/programs-surveys/acs/microdata/documentation.html Section: Government.

[8] Flávio P. Calmon, Dennis Wei, Bhanukiran Vinzamuri, Karthikeyan Natesan Ramamurthy, and Kush R. Varshney. 2017. Optimized Pre-Processing for Discrimination Prevention. In *NIPS*. 3992–4001.

[9] Priyam Choksi. 2023. Credit Card Transactions Dataset. https://www.kaggle.com/datasets/priyamchoksi/credit-card-transactions-dataset Accessed: 2024-08-04.

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

[11] Alexandra Chouldechova. 2017. Fair Prediction with Disparate Impact: A Study of Bias in Recidivism Prediction Instruments. *Big Data* 5, 2 (2017), 153–163. https://doi.org/10.1089/big.2016.0047

[12] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Holistic data cleaning: Putting violations into context. In *ICDE*. 458–469.

[13] Equal Employment Opportunity Commission et al. 1990. Uniform guidelines on employee selection procedures. *Fed Register* 1 (1990), 216–243.

[14] Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, Ahmed K. Elmagarmid, Ihab F. Ilyas, Mourad Ouzzani, and Nan Tang. 2013. NADEEF: a commodity data cleaning system. In *SIGMOD*. 541–552.

[15] Frances Ding, Moritz Hardt, John Miller, and Ludwig Schmidt. 2022. Retiring Adult: New Datasets for Fair Machine Learning. arXiv:2108.04884 [cs.LG]

[16] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard S. Zemel. 2012. Fairness through awareness. In *ITCS*. ACM, 214–226.

[17] Ronald Fagin. 2009. Tuple-Generating Dependencies. In *Encyclopedia of Database Systems*, Ling Liu and M. Tamer Özsu (Eds.). Springer US, 3201–3202. https://doi.org/10.1007/978-0-387-39940-9_1274

[18] Ronald Fagin, Benny Kimelfeld, and Phokion G. Kolaitis. 2015. Dichotomies in the Complexity of Preferred Repairs. In *PODS*. 3–15.

[19] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. 2005. Data exchange: semantics and query answering. *Theor. Comput. Sci.* 336, 1 (2005), 89–124.

[20] Michael Feldman, Sorelle A. Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. 2015. Certifying and Removing Disparate Impact. In *SIGKDD*. 259–268.

[21] Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. 2013. The LLUNATIC Data-Cleaning Framework. *Proc. VLDB Endow.* 6, 9 (2013), 625–636.

[22] Amir Gilad, Daniel Deutch, and Sudeepa Roy. 2020. On Multiple Semantics for Declarative Database Repairs. In *SIGMOD*. 817–831.

[23] Stefan Grafberger, Julia Stoyanovich, and Sebastian Schelter. 2021. Lightweight Inspection of Data Preprocessing in Native Machine Learning Pipelines. In *CIDR*.

[24] Shubha Guha, Falaah Arif Khan, Julia Stoyanovich, and Sebastian Schelter. 2023. Automated Data Cleaning Can Hurt Fairness in Machine Learning-based Decision Making. In *ICDE*. IEEE, 3747–3754.

[25] LLC Gurobi Optimization. 2024. Gurobi Optimizer. https://www.gurobi.com Accessed: 2024-03-01.

[26] Moritz Hardt, Eric Price, and Nati Srebro. 2016. Equality of Opportunity in Supervised Learning. In *NeurIPS*, Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett (Eds.). 3315–3323.

[27] Sean Kandel, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. 2011. Wrangler: interactive visual specification of data transformation scripts. In *CHI*, Desney S. Tan, Saleema Amershi, Bo Begole, Wendy A. Kellogg, and Manas Tungare (Eds.). ACM, 3363–3372.

[28] Jon M. Kleinberg, Sendhil Mullainathan, and Manish Raghavan. 2017. Inherent Trade-Offs in the Fair Determination of Risk Scores. In *ITCS (LIPIcs)*, Vol. 67. 43:1–43:23.

[29] Solmaz Kolahi and Laks V. S. Lakshmanan. 2009. On Approximating Optimum Repairs for Functional Dependency Violations. In *ICDT*. ACM, 53–62.

[30] Nick Koudas, Avishek Saha, Divesh Srivastava, and Suresh Venkatasubramanian. 2009. Metric functional dependencies. In *ICDE*.

[31] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J. Franklin, and Ken Goldberg. 2016. ActiveClean: Interactive Data Cleaning For Statistical Modeling. *Proc. VLDB Endow.* 9, 12 (2016), 948–959.

[32] Jeff Larson, Surya Mattu, Lauren Kirchner, and Julia Angwin. 2016. How We Analyzed the COMPAS Recidivism Algorithm. https://www.propublica.org/article/how-we-analyzed-the-compas-recidivism-algorithm. Accessed: 2023-04-07.

[33] Jinyang Li, Yuval Moskovitch, Julia Stoyanovich, and H. V. Jagadish. 2023. Query Refinement for Diversity Constraint Satisfaction. *Proc. VLDB Endow.* 17, 2 (2023), 106–118.

[34] Xian Li, Xin Luna Dong, Kenneth Lyons, Weiyi Meng, and Divesh Srivastava. 2015. Truth finding on the deep web: Is the problem solved? *arXiv preprint arXiv:1503.00303* (2015).

[35] Yuxi Liu, Fangzhu Shen, Kushagra Ghosh, Amir Gilad, Benny Kimelfeld, and Sudeepa Roy. 2024. The Cost of Representation by Subset Repairs. arXiv:2410.16501 [cs.DB] https://arxiv.org/abs/2410.16501

[36] Ester Livshits and Benny Kimelfeld. 2017. Counting and Enumerating (Preferred) Database Repairs. In *PODS*. 289–301.

[37] Ester Livshits and Benny Kimelfeld. 2021. The Shapley Value of Inconsistency Measures for Functional Dependencies. In *ICDT*, Vol. 186. 15:1–15:19.

[38] Ester Livshits, Benny Kimelfeld, and Sudeepa Roy. 2020. Computing optimal repairs for functional dependencies. *ACM Transactions on Database Systems (TODS)* 45, 1 (2020), 1–46.

[39] Ester Livshits, Benny Kimelfeld, and Jef Wijsen. 2021. Counting subset repairs with functional dependencies. *J. Comput. Syst. Sci.* 117 (2021), 154–164.

[40] Ester Livshits, Rina Kochirgan, Segev Tsur, Ihab F. Ilyas, Benny Kimelfeld, and Sudeepa Roy. 2021. Properties of Inconsistency Measures for Databases. In *SIGMOD*. 1182–1194.

[41] Andrei Lopatenko and Leopoldo E. Bertossi. 2007. Complexity of Consistent Query Answering in Databases Under Cardinality-Based and Incremental Repair Semantics. In *ICDT*. 179–193.

[42] Dany Maslowski and Jef Wijsen. 2014. Counting Database Repairs that Satisfy Conjunctive Queries with Self-Joins. In *ICDT*, Nicole Schweikardt, Vassilis Christophides, and Vincent Leroy (Eds.). OpenProceedings.org, 155–164.

[43] Dongjing Miao, Zhipeng Cai, Jianzhong Li, Xiangyu Gao, and Xianmin Liu. 2020. The computation of optimal subset repairs. *Proc. VLDB Endow.* 13, 12 (jul 2020), 2061–2074.

[44] Michael J. Muller, Ingrid Lange, Dakuo Wang, David Piorkowski, Jason Tsay, Q. Vera Liao, Casey Dugan, and Thomas Erickson. 2019. How Data Science Workers Work with Data: Discovery, Capture, Curation, Design, Creation. In *CHI*. ACM, 126.

[45] Evaggelia Pitoura, Kostas Stefanidis, and Georgia Koutrika. 2022. Fairness in rankings and recommendations: an overview. *The VLDB Journal* (2022), 1–28.

[46] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *PVLDB* 10, 11 (2017), 1190–1201.

[47] Christopher De Sa, Ihab F. Ilyas, Benny Kimelfeld, Christopher Ré, and Theodoros Rekatsinas. 2019. A Formal Framework for Probabilistic Unclean Databases. In *ICDT (LIPIcs)*, Vol. 127. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 6:1–6:18.

[48] Ricardo Salazar, Felix Neutatz, and Ziawasch Abedjan. 2021. Automated Feature Engineering for Algorithmic Fairness. *Proc. VLDB Endow.* 14, 9 (2021), 1694–1702.

[49] Babak Salimi, Luke Rodriguez, Bill Howe, and Dan Suciu. 2019. Interventional Fairness: Causal Database Repair for Algorithmic Fairness. In *SIGMOD*. 793–810.

[50] Sebastian Schelter, Yuxuan He, Jatin Khilnani, and Julia Stoyanovich. 2020. FairPrep: Promoting Data to a First-Class Citizen in Studies on Fairness-Enhancing Interventions. In *EDBT*. OpenProceedings.org, 395–398.

[51] Suraj Shetiya, Ian P. Swift, Abolfazl Asudeh, and Gautam Das. 2022. Fairness-Aware Range Queries for Selecting Unbiased Data. In *ICDE*. IEEE, 1423–1436.

[52] Camelia Simoiu, Sam Corbett-Davies, and Sharad Goel. 2017. The problem of infra-marginality in outcome tests for discrimination. (2017).

[53] Moshe Y. Vardi. 1982. The Complexity of Relational Query Languages (Extended Abstract). In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing (STOC '82)*. ACM, New York, NY, USA, 137–146.