# 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 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.

## 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 [25, 42]. Therefore, the problem of automatic data repairing has been the focus of much prior work [1, 4, 9, 15, 18, 19, 29, 36, 45]. 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 [5, 7, 28], changing the database in different forms [4, 7, 9, 27, 36, 39], and even relaxing it in various manners [45, 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*) [27, 35, 36, 40, 41], 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 [35, 36] has characterized the tractable and intractable cases in this setting.

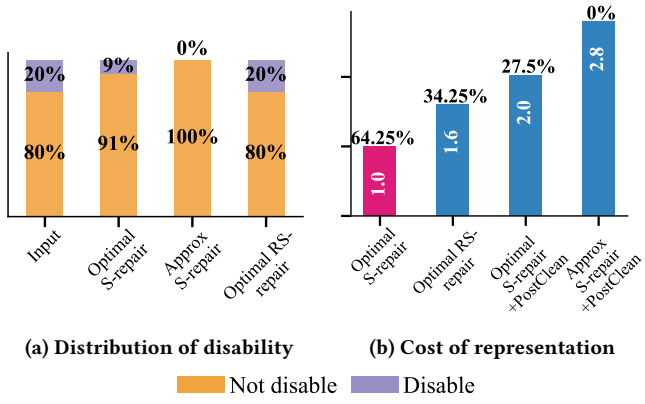



**Figure 1: Cost of representation for ACS data and disability status. In (b), the % above the bars show the percentage of remaining tuples, and the numbers on the bars show the ratio of the number of tuples deleted compared to that in an optimal S-repair (that does not satisfy representation).**

While data repair is considered a crucial part of data science pipeline, it may drastically sway the proportions of different populations in the data, and 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 [20, 21, 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 [31, 51]. However, to our knowledge, such aspects have not been considered in the context of data repairing.

In this work, we introduce the problem of understanding the cost of representation in data repair, that considers representations

* Both authors contributed equally.

of sub-populations in the repair as a first-class citizen just like the cost of changing the data. Our framework allows for FDs as well as a novel type of constraint called *representation constraint (RC)*. This constraint 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 disable and non-disable people in the dataset, where the data for the disabled group is 4-times more noisy than the non-disabled group.*

*Suppose a data analyst wants 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 [36, 41], 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. *Can we enforce a desired representation, say* 20%/80% *as in the original dataset, after obtaining an S-repair by removing additional tuples?* In this setting for optimal S-repair, we can indeed remove additional fewest non-disable tuples to change the ratio from 9%/91% to 20%/80% (called "PostClean", Section 3.3), which is referred to as the "Optimal S-repair+ PostClean" in Figure 1b. Figure 1b shows that "Optimal S-repair" retains 64.25% of the original tuples, but does not satisfy the representation. "Optimal S-repair+ PostClean" retains only 27.5% of the tuples, and removes twice as many tuples as the vanilla optimal S-repair. A natural question to ask is if this drastic reduction on the number of remaining tuples (from 64.25% to 27.5%) is an artifact of the "Optimal S-repair+ PostClean" method, or if we can do better. Figures 1a and 1b show that an "Optimal RS-repair" (studied in this paper) that maintains the representation of 20%/80% in the Disability attribute and considers FDs and RC together, will delete 1.6 times of tuples than the optimal S-repair and retain 34.25% of the tuples of the original database, which is better than applying "PostClean" on an optimal S-repair. Although retaining only 34.25% of the original tuples after the optimal RS-repair still looks pessimistic, this is the *cost of representation* we have to pay in this setting. In other words, for this dataset and noise level, if we insist on preserving the 20%/80% ratio of representation for disable and non-disable people in the dataset, we can retain at most 34.25% of the original tuples, and will be forced to remove 1.6 times the number of tuples than a representation-agnostic optimal S-repair. If we desire to retain more tuples, we have to compromise, either for the FDs or for the representation after repair[1]. For the approximate S-repair method, no disable tuples survive, so "Approx S-repair + PostClean" cannot result in the desired distribution for the Disability attribute until all tuples are removed (2.8 times of optimal S-repair) and the trivial empty set is returned as the RS-repair.

As an alternative approach to S-repair, prominent "update-repair" methods vary in their treatment of maintaining representations. For example, Holoclean [45] treats the FDs as soft constraints focusing on data distribution, and in the above example, does not make any changes in the data and does not eliminate any violations. On the other hand, while the repair given by Nadeef [11] by value updates preserves the distribution of Disability as this attribute does not participate in the input FDs, updating individual cells generate tuples that are invalid or unlikely[2] to occur in the real world. In this work, we focus on understanding the cost of representations for S-repair, and leave the study of representations under update repair or other repair methods as an interesting future direction.

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. [35, 36], who established a complete classification of complexity (dichotomy) for the whole class of FD sets. We show that finding an RS-repair can be hard even if the FDs are such that an optimal S-repair can be computed in polynomial time.

Next we study the possibility of an efficient algorithm for 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* [34, 37], 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, and test their performance in the experiments.

Finally, we perform an experimental study for cost of representation using two real-world datasets. We compare the size of the optimal RS-repairs and the optimal S-repairs. 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 complexity analysis of the problem.

[1] More settings varying noise and representations are given in the full version [32].
[2] As an example, in this case, 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, ···)`. People in Philippines are likely to be Asian (Brown), and people born in Philippines should be "`foreign born`" not "`native`", so the new tuple does not reflect reality.

(3) We devise algorithms, including polynomial-time and ILP algorithms with optimality guarantees, and heuristics.
(4) We conduct a thorough experimental study of our solutions and show their effectiveness compared to the baselines.

Due to space constraints, all proofs appear in the full version [32].

## 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 $t[A_i]$. 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., $t[X] = \pi_X t$.

*Functional dependency.* A functional dependency (FD) is denoted as $X \rightarrow Y$, where (without loss of generality) $X$ and $Y$ are disjoint sets of attributes of relation $R$. $X$ is termed the left-hand side (LHS), and $Y$ is the right-hand side (RHS) of the FD. A relation $R$ satisfies the FD $X \rightarrow Y$ 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 $t_i[Y] = t_j[Y]$ must hold. A relation $R$ satisfies a set $\Delta$ of FDs, denoted $R \models \Delta$, if it satisfies each FD from $\Delta$, otherwise it is said to *violate* $\Delta$. When $\Delta$ is clear from the context, we refer to $R$ as *clean* (respectively, *noisy*) if it satisfies (respectively, violates) $\Delta$. Without loss of generality (wlog.), we assume that the RHS $Y$ of each FD is a single attribute, otherwise we break the FD into multiple FDs. We also assume that the FDs $X \rightarrow Y$ are non-trivial, i.e., $Y \notin X$.

*Subset repair.* Much literature has studied two types of data repairs to repair a noisy relation $R$: *subset repair (S-repair)* [19, 27, 35, 41] that repairs a noisy relation by removing a set of tuples, and *update repair* [18, 36, 45] that repairs a noisy relation by changing the values of some cells. These two methods have different pros and cons. While update repairs keep the size of the dataset unchanged, it may generate tuples that are not valid in reality as discussed in Section 1. On the other hand, S-repairs generate a dataset where the tuples are faithful to the original dataset at the cost of losing a subset of the tuples. The complexity of computing an optimal S-repair has been fully studied in [35], whereas the complexity of computing an optimal update repair remains an open question [27, 35]. Since S-repair is well-understood and has been thoroughly studied in the literature, we initiate the study of cost of representation by building our model on this solid base, and leave extension to update repair as a future work.

Formally, given $R$ and $\Delta$, a *subset repair* or *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 $\Delta$, $|R'| \geq |R''|$ (which can also be defined in terms of weights when there is a weight $w(t)$ associated with each input tuple $t$).

[3]We note that in prior work [35, 40], 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.

*Computing optimal S-repairs.* Livshits et al. [35] proposed an exact algorithmic characterization (dichotomy) for computing an optimal RS-repair. Moreover, it showed that when a specific procedure is not able to return an answer, the problem is NP-hard for the input $\Delta$ 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 $\Delta$ is an attribute $A$ shared by the LHS of all FDs in $\Delta$, 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 $\Delta$ 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 $\Delta$, i.e. all attributes that can be inferred starting with $X$ using $\Delta$. 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 of $\Delta$ are proposed such that they preserve the complexity of computing an optimal S-repair, yielding a dichotomy of cases where computing an optimal S-repair is either polynomial-time or NP-hard. Further, for polynomial-time solvable cases, an optimal S-repair can be obtained recursively by dynamic programming.

## 3 REPRESENTATIVE REPAIRS

In this section, we formally define the problem of finding an optimal RS-repair, and give an overview of complexity 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) $\rho$ 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 $\rho$, denoted $R \models \rho$, if $R$ satisfies every lower-bound constraint in $\rho$.*

We assume that $\rho$ 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 $\rho$ 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 $\rho$ sum up to at most one, since otherwise the constraint is, again, infeasible.

$\rho$ 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 $p_\ell$ of value $a_\ell$ in $\rho$ by $\rho(a_\ell)$.

[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).

For simplicity, our implementation restricts attention so that each proportion ($p_\ell$) 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 an 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 $\rho$, 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 $\Delta$ of FDs, and an RC $\rho$, a subset $R' \subseteq R$ is called an RS-repair (representative subset repair) w.r.t. $\Delta$ and $\rho$ if:*
- *$R'$ is an S-repair of $R$, i.e., $R' \models \Delta$,*
- *$R'$ satisfies the RC $\rho$ on $A_s$, i.e., $R' \models \rho$.*

*We call $R'$ an* optimal RS-repair *of $R$ w.r.t. $\Delta$ and $\rho$ 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. $\Delta$ and $\rho$.*

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

*3.1.1 NP-Hardness of Computing Optimal RS-Repairs.* In this section, we consider the relation $R$ and the RC $\rho$ as inputs, while the schema $\mathcal{S}$ and the FD set $\Delta$ 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 [32]).

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 $\Delta$ contains a single FD, computing an optimal S-repair can be done in polynomial time [35]. 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. [35], computing an optimal S-repair is poly-time solvable if $\Delta$ 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 $\Delta$, such that $A$ belongs to the LHS of all FDs in $\Delta$), 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 $\Delta$ is an* LHS-chain *[33, 35] 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 [33, 35]. [33] showed that the class of LHS-chains consists of precisely the FD sets where the S-repairs can be counted in polynomial time (assuming P $\neq$ #P). [35] 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 $\Delta$ of FDs reduces to the $\emptyset$ by repeated applications of consensus FD and common LHS simplifcations if and only if $\Delta$ is an LHS-chain.*

The following theorem states our main algorithmic result.

Theorem 2. *Let $\mathcal{S}$ be a fixed schema and $\Delta$ 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$(R, \Delta, \rho)$ 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.

A natural question is what happens for LHS marriage, which is polynomial-time solvable for standard subset repair using polynomial-time algorithms for bipartite matching. We show in the full version [32] that the simplest case of LHS marriage instance for representative repair on $R(A, B, C)$ with $\Delta_0 = \{A \rightarrow B, B \rightarrow A\}$ and $C$ has only two values, is as hard as a well-known problem studied in algorithms called color-balanced matching [24, 43], which is known to have a randomized polynomial time (RP) algorithm (and therefore, is not NP-hard unless NP = RP), but does not have a known deterministic polynomial-time algorithm.

## 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 a RS-repair is (i) first compute an S-repair $R'$ (optimal or non-optimal) of $R$ w.r.t. $\Delta$, and (ii) then delete additional tuples from $R'$ to obtain $R''$ that also satisfies the RC $\rho$. Following this idea, we present the PostClean algorithm, which takes a relation $R$ and an RC $\rho$, 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 is in [32]). Recall from Section 3.1 that $\rho(a_\ell)$ denotes the lower bound on the fraction of the value $a_\ell$ in the tuples retained by the RS-repair. Further note that sum of $\rho(a_\ell)$ may be smaller than 1, i.e., the RC $\rho$ 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 $a_\ell$, i.e., $\tau_\ell = \lceil T \cdot \rho(a_\ell) \rceil$, is greater than the number of tuples with value $a_\ell$ in the original relation $R$. If yes, then no repair $R'$ of size $T$ can satisfy $\rho$, and it goes to the next value of $T$ (or $T \leftarrow T-1$). Otherwise, if there are sufficient tuples for all sensitive values $a_\ell$, 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 $\rho$.

Example 5. *Assume that an input $(R, \rho)$ of PostClean satisfies: $|R| = 12, |\sigma_{A_s=\texttt{red}}R| = 2, |\sigma_{A_s=\texttt{blue}}R| = |\sigma_{A_s=\texttt{green}}R| = 5$, and $\rho = \{\%\texttt{red} \geq \frac{1}{4}, \%\texttt{blue} \geq \frac{1}{4}, \%\texttt{green} \geq \frac{1}{4}\}$. PostClean iterates over $T$ from 12 to 0. When $T$ is between 12 and 9, it requires the subset $R'$ has at least $\tau_{\texttt{red}} = \lceil \frac{T}{4} \rceil = 3$ red tuples, but $R$ cannot fulfill that number. Next, when $T = 8$, $\tau_{\texttt{red}} = \tau_{\texttt{blue}} = \tau_{\texttt{green}} = 2$, which can be supplied by $R$ and thus $T_0 = \tau_{\texttt{red}} + \tau_{\texttt{blue}} + \tau_{\texttt{green}} = 6$. $T_0 < T$ indicates that we have to add two extra tuples that can have arbitrary $A_s$-values from $R$. Moreover, adding these two tuples will not violate the RC $\rho$, since $\tau_{\texttt{red}}, \tau_{\texttt{blue}}, \tau_{\texttt{green}}$ are already larger than or equal to $\frac{T}{4}$. Hence, PostClean will collect two more blue tuples, or two more green tuples, or one more blue and one more green from $R$.*

The following lemma proves the optimality and polynomial runtime of PostClean.

Lemma 6. *Given a relation $R$ and an RC $\rho$, PostClean$(R, \rho)$ returns a maximum subset $R'$ of $R$ in polynomial time 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 7. *Consider a relation $R$ with $\mathcal{S} = (A, B, \text{sex})$, a set $\Delta$ 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. $\Delta$ 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 $\Delta$ and $\rho$.*

# 4 POLYNOMIAL DP-BASED OPTIMAL ALGORITHM FOR LHS-CHAINS

In this section we prove Theorem 2 by presenting a DP-based optimal 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, $\Delta$ can be reduced to $\emptyset$ by repeated application of only consensus FD simplification and common LHS simplification.

**Algorithm 1** LhsChain-DP$(R, \Delta, \rho)$

**Input:** a relation $R$, an LHS-chain FD set $\Delta$, and a RC $\rho$
**Output:** an optimal RS-repair of $(R, \Delta, \rho)$
1: $C_{R,\Delta} \leftarrow \text{Reduce}(R, \Delta)$; ▷ Algorithm 2 in Section 4.2
2: $S \leftarrow \{\text{PostClean}(R', \rho) \mid R' \in C_{R,\Delta}\}$; ▷ Section 3.3
3: **return** $\arg\max_{s \in S} |s|$;

*Overview of LhsChain-DP.* For a relation $R$ and a set $\Delta$ of FDs, let $\mathcal{A}_{R,\Delta} = \{R' \subseteq R \mid R' \models \Delta\}$ be the set of all S-repairs of $R$ for $\Delta$. 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 as the optimal RS-repair. Since PostClean optimally returns a maximum subset satisfying $\rho$ for every $R'$, and since any RS-repair w.r.t. $\Delta$ and $\rho$ must be an S-repair w.r.t. $\Delta$, 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 (Lemma 10) that the 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 [32]).

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 $\Delta$ 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 computation of $C_{R,\Delta}$ is obtained by the recursive reductions.

The correctness of LhsChain-DP follows from Lemmas 10 and 11 stated later. The running time of LhsChain-DP is as follows:

Lemma 8. *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. $\Delta$. 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 $\Delta$, 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* $a_c \in Dom(A_s)$, $|\sigma_{A_s=a_c} R_1| > |\sigma_{A_s=a_c} R_2|$.

We show an example of representative equivalence and representative dominance below.

Example 9. *Consider three S-repairs for the relation $R$ with the schema* $(A, B,$ race$)$ *and FD set* $\Delta = \{A \to B\}$*: (1)* $R'_1 = \{(1, 2,$ white$), (2, 3,$ black$)\}$*; (2)* $R'_2 = \{(1, 3,$ black$), (1, 3,$ white$)\}$*; and (3)* $R'_3 = \{(1, 1,$ black$), (2, 2,$ white$), (3, 3,$ 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 $\Delta$, 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 10. *For any relation $R$ and any FD set $\Delta$, if $C_{R,\Delta}$ is computed correctly in Line 1, LhsChain-DP (Algorithm 1) returns an optimal RS-repair of $R$ w.r.t. $\Delta$ and $\rho$.*

*ReprInsert.* A subroutine ReprInsert$(C, R_0)$ (pseudocode in [32]) 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 $R_0$ 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 $\Delta$ is an LHS-chain. Since an LHS-chain $\Delta$ can be reduced to $\emptyset$ by repeated applications of consensus FD and common LHS simplifications, for the former, Reduce calls ConsensusReduction (Section 4.2.1), and for the latter, it calls CommonLHSReduction (Section 4.2.2) until $\Delta$ becomes empty. When $\Delta$ 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 shows that Reduce correctly computes $C_{R,\Delta}$ when $\Delta$ is an LHS-chain.

Lemma 11. *Given relation $R$ and FD set $\Delta$ that forms an LHS-chain, Reduce$(R, \Delta)$ correctly computes the candidate set $C_{R,\Delta}$.*

The proof is by an induction on the number of reductions applied to $\Delta$ and correctness of CommonLHSReduction and ConsensusReduction procedures that we describe below.

**Algorithm 2** Reduce$(R, \Delta)$

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

```
1: if Δ is empty then
2:     return C_{R,Δ} := {R};
3: else if Identify a consensus FD f : ∅ → 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
```

*4.2.1 Reduction for Consensus FD.* Consider a consensus FD $f : \emptyset \to Y$. 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 $y_\ell$. Note that any S-repair $R' \in C_{R_{y_\ell},\Delta-f}$ is also an S-repair of $R$ for $\Delta$, i.e., $R' \in \mathcal{A}_{R,\Delta}$, since the FD $f$ is already taken care of in $R_{y_\ell}$. 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 is 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 $\Delta$. 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 $\Delta_{-X}$ denotes that the common LHS attribute $X$ is removed from all FDs in $\Delta$. If we were to consider S-repairs, we could optimally repair each $R_{x_\ell}$ w.r.t. $\Delta_{-X}$ independently (and recursively), and then take the union of their optimal S-repairs to obtain an optimal S-repair for $R$ w.r.t. $\Delta$ (as done in [35]).

However, this is not the case for computing RS-repairs, since, unlike optimal S-repairs, the maximum size is not the only requirement of optimal RS-repairs. In other words, although we know the final repair will satisfy $\rho$, we do not know what the value distribution of $A_s$ should be in each disjoint piece (e.g., some $R_{x_\ell}$) 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 the possible distributions of $A_s$ from the S-repairs that could provide the final optimal solution.

**Algorithm 3** ConsensusReduction$(R, \Delta, f)$

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

```
1: C_{R,Δ} ← ∅;
2: for each value y_ℓ ∈ Dom(Y) do
3:     C_{R_{y_ℓ},Δ−f} ← Reduce(R_{y_ℓ}, Δ − f);   ▷ Algorithm 2
4:     for all R' in C_{R_{y_ℓ},Δ−f} do
5:         C_{R,Δ} ← ReprInsert(C_{R,Δ}, R');   ▷ Section 4.1
6:     end for
7: end for
8: return C_{R,Δ}.
```

The procedure CommonLHSReduction (Algorithm 4) in this section 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 $\ell$, 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. $\Delta$ , but we have to ensure that the properties of a candidate set is 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 candidate set $C_{R,\Delta}$.

**Algorithm 4** CommonLHSReduction$(R, \Delta, X)$

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

# 5 ALGORITHMS FOR THE GENERAL CASE

Computing optimal RS-repairs for arbitrary $\Delta$ and $\rho$ is NP-hard (Section 3.1.1). In this section we present a collection of end-to-end algorithms capable of handling general inputs. We first present an exact algorithm based on integer linear programming (ILP) (Section 5.1) that is not polynomial-time. Then we present efficient heuristics utilizing LP relaxations and greedy rounding (Section 5.2), and using procedures from the previous section for LHS-chains as a subroutine (Section 5.3).

## 5.1 ILP-based Optimal Algorithm

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 (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 $\Delta$. The following set of constraints correspond to the RC $\rho$, 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 $a_\ell \in Dom(A_s)$.

*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.

## 5.2 LP Rounding-based Algorithms

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 $i \in [1, |R|]$. It can be solved in polynomial time, but may return fractional variable values. We, therefore, propose two polynomial-time rounding procedures to derive an integral solution from fractional $x_i$s (pseudocode and running time analysis in [32]).

*LP + GreedyRounding.* The greedy rounding algorithm LP + GreedyRounding greedily chooses a fractional $x_i$, where $0 < x_i < 1$, that participates in the smallest number of LP constraints. Then it rounds $x_i$ to 1 and for all constraints $x_i + x_j \leq 1$, it rounds $x_j$ to 0, thereby satisfying the linear constraints and consequently resolving the FD violations. Once all the FDs in $\Delta$ are satisfied, it calls PostClean (Section 3.3) to ensure that $\rho$ is also satisfied. When dealing with datasets containing a large number of errors, this simple greedy rounding approach does not work well as it fails to give priority to sub-populations that are under-representative compared to the desired proportions specified by $\rho$.

*LP + ReprRounding.* To address the issue in LP + GreedyRounding, we propose a representation-aware greedy algorithm, which is called LP + ReprRounding. Here, for rounding fractional $x_i$s to 1, tuples from underrepresented groups are prioritized. Inspired by stratified sampling [44], we use the ratio $\frac{\sum_{i:t_i[A_s]=a_\ell} x_i}{p_\ell}$ to prioritize $x_i$ for rounding, where $p_\ell = \rho(a_\ell)$. A group with smaller ratio is less representative in the retained tuples. From these less representative groups, LP + ReprRounding chooses $x_i$ associated with the fewest conflict constraints, then employing similar techniques as LP + GreedyRounding does, including applying a final PostClean. While the computational cost is usually dominated by solving the LP, the rounding for LP + ReprRounding takes slightly longer than that in LP + GreedyRounding due to the additional representation considerations in this step.

*Limitations.* Even with the state-of-the-art LP solvers, solving our LP is not scalable for large input relations (with hundreds of thousands of tuples) and with large number of constraints introduced by FD violations that can be quadratic in the size of the input relation even when considered as an offline task as observed in our experiments. Hence in the next subsection, we propose combinatorial DP-based efficient heuristics using ideas from Section 4.

## 5.3 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 $\Delta$ 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 $\Delta$ 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.

**Algorithm 5** FDCleanser$(R, \Delta, \rho)$

1: **while** $\Delta$ is not empty **do**
2: Select the most frequent LHS column $X$ in $\Delta$;
3: Choose one arbitrary FD $f$ whose LHS contains $X$;
4: $R \leftarrow$ LhsChain-DP$(R, \{f\}, \rho)$;
5: $\Delta \leftarrow \Delta - f$
6: **end while**
7: **return** $R$;

As highlighted, Since $\Delta$ is fixed and each call to LhsChain-DP runs in polynomial time for fixed $Dom(A_s)$, FDCleanser terminates in polynomial time for any $(R, \Delta, \rho)$ 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 Sections 5.2 and 5.3 first employ ConsensusReduction and CommonLHSReduction until no feasible reductions are possible. This splits the original problem into a set of sub-problems that can be solved independently. Then the heuristics are applied on these 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 of representation in subset repairs and the performance of our proposed algorithms. In particular, we study the following questions:

(1) Section 6.2: How many additional tuple deletions are required for computing optimal RS-repairs compared to computing optimal S-repairs?
(2) Section 6.3: How effective is each algorithm in minimizing tuple deletions for computing RS-repairs?
(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 $\%a = p$?

In Sections 6.2 to 6.4, all results presented are based on exact RCs (Definition 1), where the required proportions match the value distribution of the sensitive attribute in the input relation. This choice aims to to highlight the motivation of preserving the distribution of the sensitive attribute as it was before the repairing process. In Section 6.5, we delve into a more general case where the required proportions do not exactly match the input distribution.

**Table 1: FD sets used in experiments**

| Dataset | chain FD Set | Non-chain FD Set |
|---|---|---|
| **ACS** | {ST → DIV, ST → 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} |

## 6.1 Setup

*Implementation details.* We implement our algorithms in Python 3.11 and conduct experiments 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 Optimizer as the solver of ILP and LP [22]. The code is publicly available[5].

*Datasets.* We use two datasets that are commonly used for fairness and representation analysis.

- ACS: The American Community Survey Public Use Microdata Sample [12]. 9 attributes are involved in our experiments. We generate random samples of sizes from 2K rows to 1M rows.
- COMPAS: The ProPublica COMPAS recidivism dataset [30]. 10 attributes are involved in our experiments. We generate random samples from 4K to 30K rows.

*Functional dependencies (FD).* As shown in Table 1[6], we consider two types of FD sets: (1) chain FD sets, that is, where the FD set forms an LHS-chain and (2) non-chain FD sets. Initially, the datasets and consequently all of their samples satisfy the FDs.

*Noise injection.* Inspired by prior work [11, 18, 46], we inject random noises into the samples to introduce FD violations. Specifically, we change the values of randomly selected cells to other values in the active domain of the corresponding attribute. Formally, $x\%$ noise, called the *level of noise*, indicates that $x\%$ of the cells in columns involved in either the LHS or the RHS of FDs in the set are disturbed. Note that the same level of noise may result in a different number of violations, as the set of FDs changes.

*Representation constraints.* We consider the following RCs. For the ACS dataset, we use an RC on the sensitive attribute Nativity, $\{\%\texttt{Native-born} = \frac{80}{100}, \%\texttt{Foreign-born} = \frac{20}{100}\}$ and another RC on Region, $\{\%\texttt{Region 1 and 2} = \frac{80}{100}, \%\texttt{Region 3 and 4} = \frac{20}{100}\}$ for non-chain FD sets and chain FD sets respectively. We use different RCs based on the type of FD sets since we want (1) a natural sensitive attribute and (2) the sensitive attribute to always be included in the FD set. Therefore, for non-chain FDs, we choose the Nativity attribute. However, since this attribute is not included in the chain FDs, we resort to an RC over Region. For the COMPAS dataset, we use an RC on the sensitive attribute Sex, $\{\%\texttt{Male} = \frac{80}{100}, \%\texttt{Female} = \frac{20}{100}\}$. As stated previously, we use exact

[5] https://github.com/louisja1/RS-repair.

[6] 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.

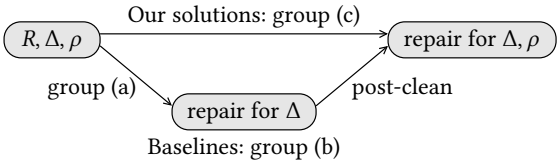



**Figure 2: Our solutions vs. baselines**

RCs in Sections 6.2 to 6.4, and we examine general RCs in Section 6.5.

*Algorithms.* We classify the different repairing approaches into three categories: (a) S-repairs, (b) S-repairs with PostClean (Section 3.3) to satisfy the RC, and (c) designated RS-repairs, as shown in Figure 2. Group (a) includes the examined baseline approaches for computing S-repairs:

- ILP-Baseline [38]: Formulates the problem as an ILP and computes the *optimal* S-repair satisfying a given FD set.
- VC-approx-Baseline [2]: An approximation algorithm that translates the problem of computing an optimal S-repair to the problem of finding a minimum vertex cover of the conflict graph and utilizes its known 2-approximation algorithm.
- DP-Baseline [36]: A DP algorithm that computes the *optimal* S-repair in polynomial time when the FD set forms an LHS-chain.
- MuSe-Baseline [19]: A repair framework for S-repairs composed of multiple semantics. We use step semantic.

Group (b) is defined as using PostClean as post-processing for 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 exact algorithm LhsChain-DP (Section 4). While for non-chain FD sets where LhsChain-DP is not applicable, we examine end-to-end heuristic algorithms LP + GreedyRounding, LP + ReprRounding, and FDCleanser (Section 5). Recall that when applied to chain FD sets, these three algorithms reduce to LhsChain-DP and, consequently, provide the same RS-repairs and performance. Therefore, to avoid redundancy, we do not include the results of LP + GreedyRounding, LP + ReprRounding, and FDCleanser on chain FD sets. We repeat each experiment twice and take the average for each data point.

## 6.2 Deletion Overhead of RS-repairs

We report the deletion overhead of RS-repairs, as indicated by the following ratio ($\#_{\text{del}}()$ is the number of tuple deletions of a repair):

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

This ratio quantifies the additional deletions required by an optimal RS-repair compared to an optimal S-repair for the same $(R, \Delta)$. This ratio is always at least 1, as the optimal RS-repair necessarily deletes more tuples than the optimal S-repair on the same input. A large ratio indicates a high overhead, i.e., too many extra deletions are required for the representation. We use ILP-Baseline for optimal S-repairs and GlobalILP for optimal RS-repairs.

Figure 3 depicts the deletion overhead of RS-repairs on the ACS dataset, varying: (i) input relation size (2K to 10K), (ii) levels of noise (1%, 5%, 10%) and (iii) FD set types (chain and non-chain). Each point

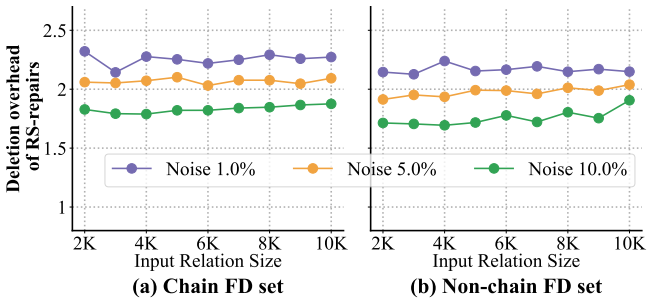



**Figure 3: Deletion overhead results on the ACS dataset, (a) chain FD set and (b) non-chain FD set. The overhead is 2.25, 2.06, and 1.83 on average for 1%, 5%, and 10% noise, respectively, across input relation sizes and types of FD sets.**

is averaged over two runs. For example, the yellow point at $2K$ for the chain FD set (Figure 3a) indicates that the optimal RS-repairs delete 2.06× tuples in average compared to the optimal S-repairs under the 5% noise level. The relation size does not significantly impact the deletion overhead of RS-repairs. Second, higher noise levels lead to lower deletion overhead of RS-repairs. As the level of noise increases, S-repairs tend to delete more minority tuples, while RS-repairs maintain their representation, consequently reducing the ratios. Third, the type of FD sets, i.e., chain or non-chain, does not significantly affect the deletion overhead of RS-repairs.

## 6.3 RS-repair Quality

Although GlobalILP provides the optimal RS-repair for arbitrary inputs and consequently measures the deletion overhead of RS-repairs accurately, solving ILP is NP-hard in general and is not scalable. For example, GlobalILP took 18 hours to repair a relation of only 10K tuples and 10% noise for a non-chain FD set. Hence, we cannot rely on GlobalILP for repairing large relations in practice. This raises the question: how effective are other proposed algorithms in minimizing tuple deletions while satisfying the RC?

We report the following measure that captures *repair quality*:

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

The expression compares the number of tuples retained by an RS-repair $R'$ to that of an optimal RS-repair (provided by GlobalILP) for the same input, indicating how well $R'$ approximates the optimum. This quality is upper-bounded by 100%, as optimal RS-repairs have the minimum number of deletions. Therefore, a ratio close to 100% indicates that the RS-repair is of high quality. Figure 4 presents repair quality for both chain and non-chain FD sets in two datasets. We provide more repair quality results for various settings in [32].

*ACS.* In Figures 4a and 4b, we vary the level of noise (5%, 10%) and relation size (4K to 10K) for chain FD sets. First, GlobalILP (in blue) demonstrates its 100% repair quality, serving as the optimal benchmark. LhsChain-DP (in red) also maintains 100% repair quality across all cases of the chain FD set as expected. The baselines retain a high quality 93% under 5% noise, while they soon drop to about 77% compared to optimal RS-repairs for 10% noise. Additional results for noise levels of 15% and 20% (in [32]) show that the repair quality of the baselines deteriorates significantly as the level

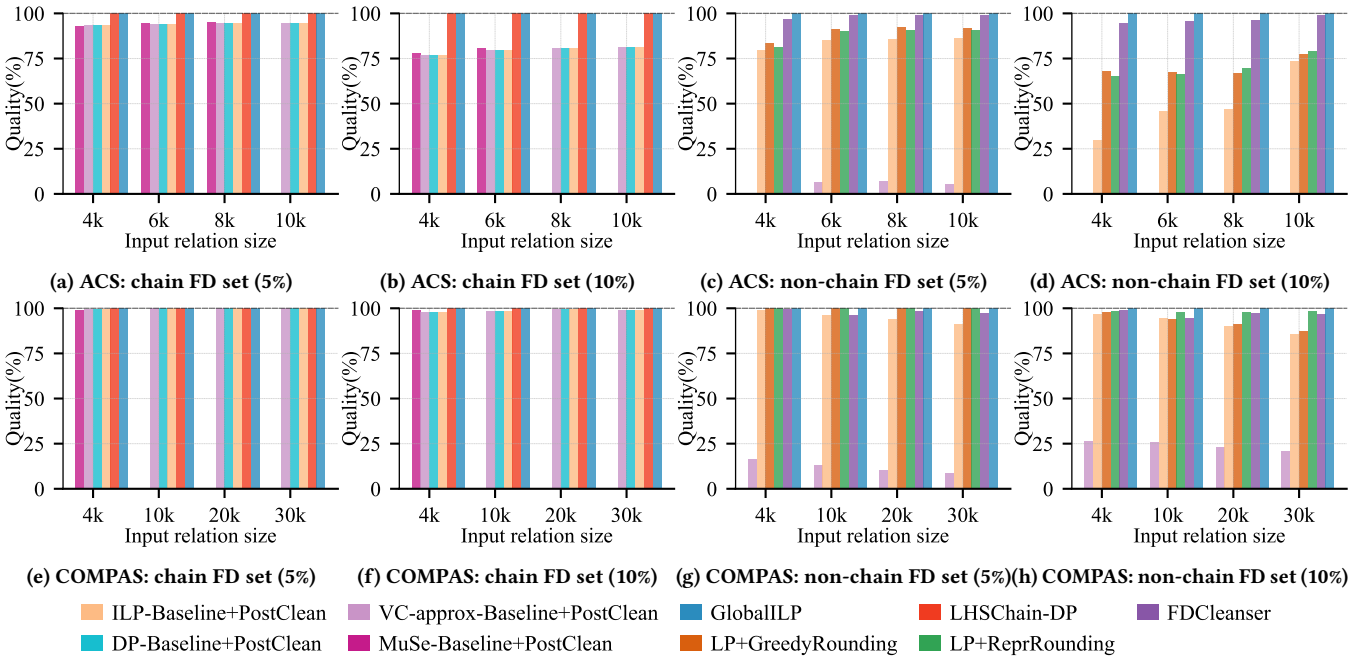



**Figure 4: Repair quality of different algorithms for ACS and COMPAS data, chain and non-chain FD sets, 5% and 10% noise levels**

of noise increases. MuSe-Baseline+PostClean is omitted for 8K (10% noise) and 10K (5% and 10% noise) due to running for more than 24 hours.

In Figures 4c and 4d, we evaluate the same noise and sizes for non-chain FD sets. GlobalILP continues to provide 100% repair quality, while the heuristics approaches, LP + GreedyRounding and LP + ReprRounding perform similarly, achieving about 80% (resp. 65%) repair quality for 5% (resp. 10%) noise. FDCleanser consistently maintains over 91% quality across all scenarios. On the other hand, the repair quality of ILP-Baseline+PostClean deteriorates significantly with high levels of noise, dropping below 50% for input size 4K, 6K, 8K, and 10% noise. VC-approx-Baseline+PostClean performs with almost 0% repair quality across all scenarios. MuSe-Baseline+PostClean runs more than 24 hours even for 4K tuples.

*COMPAS.* In Figures 4e to 4h, we demonstrate the results of the same noise levels (5%, 10%) from input relation size 4K to 30K. Interestingly, baselines in group (b) achieve high repair quality in most cases. Except for VC-approx-Baseline+PostClean in the non-chain FD set, all algorithms perform over 80% repair quality across all the scenarios. For the chain FD set, DP-Baseline+PostClean and ILP-Baseline+ PostClean reach 92% repair quality and even achieve 99% repair quality for sizes ranging from 10K to 30K and 5% noise. However, the baselines do not consistently provide 100% repair quality like LhsChain-DP does. GlobalILP, LhsChain-DP and FDCleanser are still the top performers in most cases for chain and non-chain FD sets.

## 6.4 Runtime Analysis

We now measure the runtime for computing RS-repairs using algorithms from groups (b) and (c). We present the results in two parts. First, we focus on ACS dataset with input relation sizes of 4K to 10K and on COMPAS dataset with sizes of 4K to 30K, using the same measurement points in Section 6.3. Then, we focus on our more scalable algorithms and measure their runtime for ACS dataset of sizes 10K to 1M. For Group (b), we note that PostClean takes negligible time (less than 0.2s even for 1M tuples) compared to computing S-repairs.

As shown in Figure 5, ILP-Baseline+PostClean and GlobalILP spend more time on repairs for non-chain FD sets than for chain FD sets for both ACS and COMPAS. Moreover, the level of noise has a greater impact on runtime for the non-chain FD sets. Last, MuSe-Baseline+PostClean is slow across all scenarios. In all cases where no data points are shown, execution takes more than 24 hours. Next, we detail our results for each scenario.

*ACS.* In Figures 5a and 5b for the chain FD set, GlobalILP has a longer runtime than LhsChain-DP across all cases. For example, GlobalILP takes 1,046s for size 10K with 10% noise, while LhsChain-DP only needs 12s to achieve the same repair quality. GlobalILP has a longer runtime than ILP-Baseline+PostClean does, because it contains more constraints in the ILP due to the RC. This also results in a longer runtime of LhsChain-DP compared to that of DP-Baseline+PostClean. DP-Baseline+PostClean and VC-approx-Baseline + PostClean have similar runtime (the trend lines overlap). Both of them run faster than other algorithms.

In Figures 5c and 5d for the non-chain FD set, LP + GreedyRounding and LP + ReprRounding are both faster than GlobalILP. For example, in 10K size with 10% noise, GlobalILP (64,039s) takes more than 300 times the runtime of LP + GreedyRounding (278s) and the runtime of LP + ReprRounding (166s), respectively. FDCleanser, which consistently takes less than 8s across all scenarios, demonstrates its good scalability.

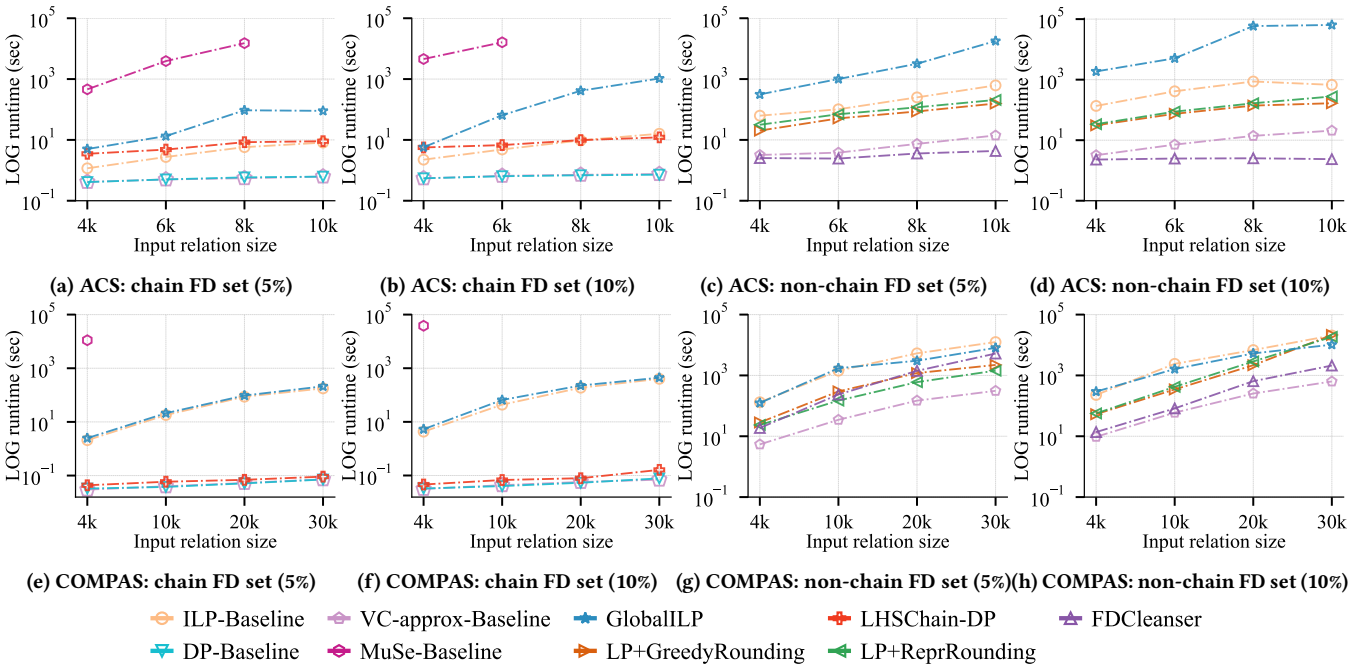



**Figure 5: Runtime cost of different algorithms for ACS and COMPAS data, chain and non-chain FD sets, 5% and 10% noise levels.**

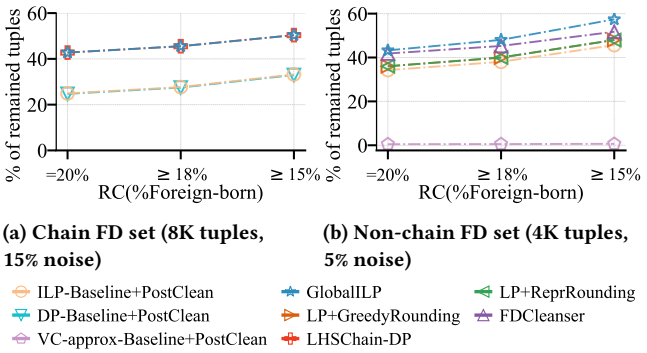
*COMPAS.* In Figures 5e and 5f for the chain FD set, the gap between GlobalILP and ILP-Baseline+PostClean and the gap between LhsChain-DP and DP-Baseline+PostClean are much smaller compared to the gaps in Figures 5a and 5b. For example, GlobalILP takes 443s for size 30K with 10% noise while ILP-Baseline+PostClean takes 117s. It indicates that the extra runtime cost to incorporate representation into the procedure of repairing (instead of fully relying on PostClean) for COMPAS is lower compared to ACS.

Regarding the non-chain FD set, as shown in Figures 5g and 5h, the runtime of both LP + GreedyRounding and LP + ReprRounding increases as the input relation size increases, especially since they have a longer runtime than GlobalILP in the case of a 30K input relation size and 10% noise. This pattern is caused by the rounding step, which runs longer than under 5% noise. It also explains the longer runtime of LP + ReprRounding compared to LP + GreedyRounding in the same case.

*6.4.1 Scalability analysis of non-ILP Algorithms.* GlobalILP consistently provides the optimal RS-repair, but its NP-hardness results in impracticality for large inputs. As shown in Figure 5d, GlobalILP takes 18 hours to run over 10K tuples in the non-chain FD case with 10% noise. Despite its poor scalability, previous results show that our polynomial-time algorithm LhsChain-DP for the chain FD set and the heuristic algorithms LP + GreedyRounding, LP + ReprRounding, and FDCleanser for the non-chain FD set are efficient while providing high-quality repairs. Therefore, in Table 2, we examine the runtime for these scalable algorithms for large input relation sizes up to 1M tuples on the ACS dataset. VC-approx-Baseline+PostClean and DP-Baseline+PostClean demonstrate better scalability (less than 15s), but they might provide RS-repairs of low repair quality. For example, for ACS dataset with 10% noise shown in Figure 4b, both provide RS-repairs with quality only around 75%. Our algorithm, LhsChain-DP consistently provides the optimal RS-repair, i.e., of 100% repair quality, in 8 hours for 1M input relation size. Next, for the non-chain FD set, VC-approx-Baseline + PostClean, LP + GreedyRounding, LP + ReprRounding encounter issues with memory usage when $|R| \geq$ 100K, which is as expected given that the number of constraints can be as large as $|R|^2$. Moreover, FDCleanser is faster than the LP relaxations, where it executes for only 1.1 min. while LP relaxations need more than 70 min. It is well scalable and takes 5.9 hours for size 1M. In summary, LhsChain-DP and FDCleanser demonstrate their scalability for chain FD sets and non-chain FD sets, respectively.

**Table 2: Runtime cost (minutes) for the ACS with 5% noise. "OOM" means we experienced out-of-memory issues.**

| **Chain FD set** | | | | | |
|---|---|---|---|---|---|
| Name/DB Size | 10K | 40K | 100K | 500K | 1M |
| VC-approx-Baseline +PostClean | 0.01 | 0.02 | 0.02 | 0.13 | 0.25 |
| DP-Baseline +PostClean | 0.01 | 0.02 | 0.02 | 0.07 | 0.14 |
| **LhsChain-DP** | 0.14 | 0.77 | 3.52 | 84.76 | **479.50** |
| **Non-chain FD set** | | | | | |
| Name/DB Size | 10K | 40K | 100K | 500K | 1M |
| VC-approx-Baseline + PostClean | 0.29 | 5.46 | OOM | OOM | OOM |
| LP + GreedyRounding | 3.16 | 71.42 | OOM | OOM | OOM |
| LP + ReprRounding | 3.18 | 85.82 | OOM | OOM | OOM |
| **FDCleanser** | 0.15 | 1.12 | 10.56 | 142.80 | **351.79** |




**(a) Chain FD set (8K tuples, 15% noise)**

**(b) Non-chain FD set (4K tuples, 5% noise)**

**Figure 6: Percentage of remained tuples while varying the RC for different algorithms on ACS dataset**

## 6.5 Varying Representation Constraints

We now relax the $\rho$ by gradually reducing the lower bound proportion of some individual constraints in $\rho$. Specifically, one of the RCs for ACS is $\{\%\texttt{Native-born} \geq \frac{80}{100}, \%\texttt{Foreign-born} \geq \frac{20}{100}\}$, which is identical to the distribution of Nativity in the input relations. As for relaxing this RC, we gradually reduce the required proportion of `Foreign-born` (the minority group) to 18% and then 15%.

As shown in Figure 6, the percentage of tuples retained by the RS-repairs for every algorithm increases as the specified lower-bound proportion of `Foreign-born` reduces, regardless of the type of FD sets. The reason is that preserving the population of the minority group is the bottleneck that prevents the algorithm from less deletions. In the end, although the algorithms are able to delete fewer tuples by relaxing the RCs, we claim that it is a trade-off between the strictness of $\rho$ and the tuple deletions needed. The other side of the coin is that the RS-repairs might lose more tuples with the sensitive values that belong to the minority groups.

## 7 RELATED WORK

*Data repairing and constraints.* A common theme past work on data repairing [1, 4, 9, 15, 45] is the aim to minimize the number of changes in the database (intervention) in order to reach a version that satisfies a set of constraints. Yet, the literature on data repairing can be partitioned into two main veins, based on the approach used for intervention. One approach allows for tuple deletions to repair the data [1, 7, 19, 36, 39, 41], while others allow for value updates [4, 9, 11, 18, 27, 45]. Tuple deletion has a clearer theoretical characterization [36], thus allowing us to build on this model to extend the classic setting to support representation constraints. We intend to study extensions of our approach to the value-change model as well in future work.

The literature also proposed a wide variety of constraints, such as FDs and versions thereof [5, 28], denial constraints [7], tuple-generating dependencies [14, 16], and equality-generating dependencies [3]. These are first-order logic statements that point to *local issue* in the database, e.g., a specific set of tuples that violate a constraint or a specific tuple-pattern that has to be present in the database. Conversely, a representation constraint considers the statistical proportions of an attribute *over the entire database.*

*Representation constraints.* Multiple definitions of constraints involving probability distributions on the data have been proposed in the literature. A predominant part of these focuses on *algorithmic fairness*: given a classification algorithm, and sensitive attributes, determine whether the classification is *fair* according to some definition of *fairness* [10, 23]. Algorithmic fairness and data repair for algorithmic fairness [6, 17, 48, 49] focus on fairness in light of *a specific downstream task*. These works consider an 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. For RS-repairs, we do not have a pre-defined task or application, and aim to create a dataset with good quality for many subsequent applications similar to the goal of the standard data repair approaches in the literature. In addition, the RCs can be defined for any attribute and it is not required to have an outcome attribute in the data.

One of our hypotheses is that maintaining representations of different sub-populations while repairing the data is a precursor to reducing biases in all data-dependent tasks. If a sub-population becomes underrepresented during repair, an algorithm (e.g., ML-based predictions) trained on this data has the potential to be biased. Of course, the requirement of maintaining representations may vary among applications requiring updating RCs. Several fairness definitions can be expressed as comparisons of (combinations of) conditional probabilities, e.g., demographic parity [13, 26] and true positive rate balance [8, 52]. Capturing some fairness definitions using complex RCs is an intriguing subject for future work

## 8 CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a novel framework for estimating the cost of representation for S-repairs, i.e., how many extra tuples have to be deleted in order to satisfy both the FDs as well as a representation constraint (RC) on the values of a sensitive attribute. We studied the complexity of computing an optimal RS-repair, presented poly-time optimal algorithms for FD sets that form LHS-chains for bounded domain of the sensitive attribute, devised efficient heuristics for the general cases, and evaluated our approaches experimentally. Since data repair with representations is a novel problem, there are many interesting future directions. First, the exact complexity characterization (like for S-repair [35]) for cost of representations for RS-repair remains an open problem. Second, it will be interesting to study the cost of representation for update repair. As illustrated in the example in the introduction, if the sensitive attribute and the FD set do not have an overlap, the distribution of the sensitive attribute values is automatically preserved for update repair, however, it may generate unrealistic tuples, which is not possible for subset repair. Hence the problem setup for update repair should consider the overall distribution of values over all attributes. Third, one can study the complexity and algorithms for generalization of representations to multiple sensitive attributes (for arbitrary overlap with the FDs, preserving marginal distribution vs. joint distribution, etc.). Finally, it will be interesting to study the implication of preserving representations in the input dataset to preserving fairness in the output of an application such as an ML model built on the repaired dataset.

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

[6] 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.

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

[8] 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

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

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

[11] 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.

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

[13] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard S. Zemel. 2012. Fairness through awareness. In *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, Shafi Goldwasser (Ed.). ACM, 214–226. https://doi.org/10.1145/2090236.2090255

[14] 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

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

[16] 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.

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

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

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

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

[21] Shubha Guha, Falaah Arif Khan, Julia Stoyanovich, and Sebastian Schelter. 2023. Automated Data Cleaning Can Hurt Fairness in Machine Learning-based Decision Making. In *39th IEEE International Conference on Data Engineering, ICDE 2023, Anaheim, CA, USA, April 3-7, 2023*. IEEE, 3747–3754. https://doi.org/10.1109/ICDE55515.2023.00303

[22] Gurobi. [n.d.]. https://www.gurobi.com/. ([n. d.]).

[23] Moritz Hardt, Eric Price, and Nati Srebro. 2016. Equality of Opportunity in Supervised Learning. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett (Eds.). 3315–3323. https://proceedings.neurips.cc/paper/2016/hash/9d2682367c3935defcb1f9e247a97c0d-Abstract.html

[24] Santhini K. A., Govind S Sankar, and Meghana Nasre. 2022. Optimal Matchings with One-Sided Preferences: Fixed and Cost-Based Quotas. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*. 696–704.

[25] Sean Kandel, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. 2011. Wrangler: interactive visual specification of data transformation scripts. In *Proceedings of the International Conference on Human Factors in Computing Systems, CHI 2011, Vancouver, BC, Canada, May 7-12, 2011*, Desney S. Tan, Saleema Amershi, Bo Begole, Wendy A. Kellogg, and Manas Tungare (Eds.). ACM, 3363–3372. https://doi.org/10.1145/1978942.1979444

[26] Jon M. Kleinberg, Sendhil Mullainathan, and Manish Raghavan. 2017. Inherent Trade-Offs in the Fair Determination of Risk Scores. In *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA (LIPIcs)*, Christos H. Papadimitriou (Ed.), Vol. 67. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 43:1–43:23. https://doi.org/10.4230/LIPIcs.ITCS.2017.43

[27] Solmaz Kolahi and Laks V. S. Lakshmanan. 2009. On Approximating Optimum Repairs for Functional Dependency Violations. In *Proceedings of the 12th International Conference on Database Theory* (St. Petersburg, Russia) *(ICDT '09)*. Association for Computing Machinery, New York, NY, USA, 53–62. https://doi.org/10.1145/1514894.1514901

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

[29] 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. https://doi.org/10.14778/2994509.2994514

[30] 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.

[31] 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. https://www.vldb.org/pvldb/vol17/p106-li.pdf

[32] Yuxi Liu, Fangzhu Shen, Kushagra Ghosh, Amir Gilad, Benny Kimelfeld, and Sudeepa Roy. [n.d.]. The Cost of Representation by Subset Repairs - Full Version. https://github.com/louisja1/RS-repair/blob/main/The_Cost_of_Representation_by_Subset_Repairs_full_version.pdf.

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

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

[35] Ester Livshits, Benny Kimelfeld, and Sudeepa Roy. 2018. Computing Optimal Repairs for Functional Dependencies. In *SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. 225–237.

[36] 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.

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

[38] 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.

[39] 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.

[40] Dany Maslowski and Jef Wijsen. 2014. Counting Database Repairs that Satisfy Conjunctive Queries with Self-Joins. In *Proc. 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24-28, 2014*, Nicole Schweikardt, Vassilis Christophides, and Vincent Leroy (Eds.). OpenProceedings.org, 155–164. https://doi.org/10.5441/002/ICDT.2014.18

[41] 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. https://doi.org/10.14778/3407790.3407809

[42] 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 *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI 2019, Glasgow, Scotland, UK, May 04-09, 2019*, Stephen A. Brewster, Geraldine Fitzpatrick, Anna L. Cox, and Vassilis Kostakos (Eds.). ACM, 126. https://doi.org/10.1145/3290605.3300356

[43] Ketan Mulmuley, Umesh V Vazirani, and Vijay V Vazirani. 1987. Matching is as easy as matrix inversion. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. 345–354.

[44] Jerzy Neyman. 1992. On the two different aspects of the representative method: the method of stratified sampling and the method of purposive selection. In *Breakthroughs in statistics: Methodology and distribution*. Springer, 123–150.

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

[46] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *Proc. VLDB Endow.* 10, 11 (aug 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 *Proceedings of the 23rd International Conference on Extending Database Technology, EDBT 2020, Copenhagen, Denmark, March 30 - April 02, 2020*, Angela Bonifati, Yongluan Zhou, Marcos Antonio Vaz Salles, Alexander Böhm, Dan Olteanu, George H. L. Fletcher, Arijit Khan, and Bin Yang (Eds.). OpenProceedings.org, 395–398. https://doi.org/10.5441/002/EDBT.2020.41

[51] Suraj Shetiya, Ian P. Swift, Abolfazl Asudeh, and Gautam Das. 2022. Fairness-Aware Range Queries for Selecting Unbiased Data. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*. IEEE, 1423–1436. https://doi.org/10.1109/ICDE53745.2022.00111

[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* (San Francisco, California, USA) *(STOC '82)*. ACM, New York, NY, USA, 137–146. https://doi.org/10.1145/800070.802186

# A DETAILS IN SECTION 1

We present more results by varying noise distributions and representations in Figures 7 to 9. Figure 7a shows that when the data for the disabled group is twice as much noise as the non-disabled group, the optimal (maximal-size) S-repair drops the proportion of people with disabilities from 20% to 16%. No people with disabilities stay in the repaired data in the "Approx S-repair". Figure 7b shows that the "Optimal S-repair" retains 61.10% of the original tuples, but does not satisfy the representation. If we apply PostClean to remove additional fewest non-disable tuples to restore the ratio to 20%/80%, only 47.75% of original tuples are retained (referred to as "Optimal S-repair+PostClean"), removing 1.34 times compared to vanilla optimal S-repair. In contrast, the "Optimal RS-repair" (studied in this paper) deletes 1.27 times tuples deleted by the optimal S-repair and retains 50.75% of the original tuples, while maintaining the 20%/80% representation and satisfying all FDs.

Figures 8a and 8b demonstrate that when the data is uniformly noisy across both disabled and non-disabled groups, the optimal S-repair still deviates from 20%/80% to 19%/81% and retains 59.62% of original tuples. The "Approx S-repair" also retains no tuples with disabilities. Applying PostClean to the vanilla optimal S-repair ("Optimal S-repair+PostClean") removes 1.46 times the number of tuples deleted by the optimal S-repair and retains 57.75% of the original tuples. However, the "Optimal RS-repair" deletes only 1.003 times of the optimal S-repair and retains 59.50% of the original tuples. As shown in Figures 7 and 8, when we vary the noise, the cost of representation we need to pay decreases.

Figure 9 shows that compared to Example 1, when we relax the representation constraint from 20%/80% to 10%/90%, the cost of representation we have to pay decreases accordingly. Figure 9a has the same distribution as Figure 1a due to their same noise distribution. However, when enforcing the representation 10%/90%, "Optimal S-repair+PostClean" removes 1.26 times the number of tuples deleted by the vanilla "Optimal S-repair" and retains 55.00% of the original tuples. In comparison, "Optimal RS-repair" removes only 1.02 times compared to the optimal S-repair and retains 64.50% of the original tuples.

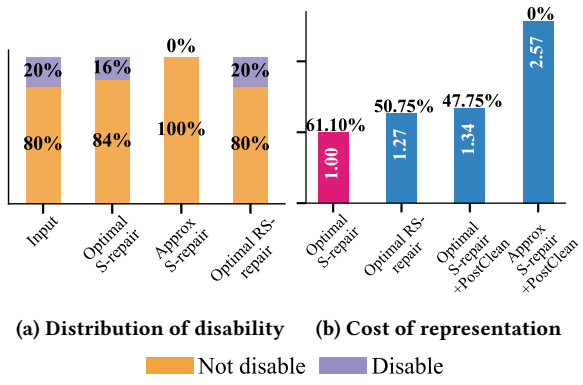



**Figure 7: Cost of representation for ACS data and disability status (the disabled group is twice as much noise as the non-disabled group)**

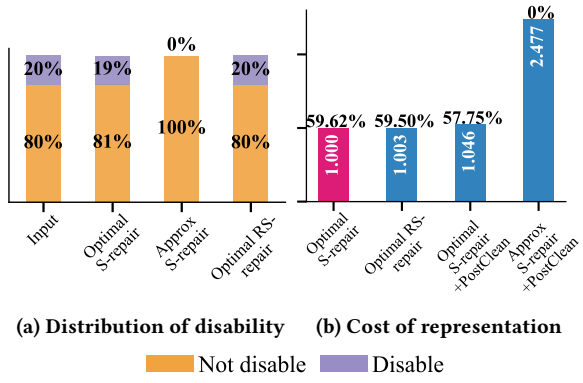



**Figure 8: Cost of representation for ACS data and disability status (uniformly noisy)**

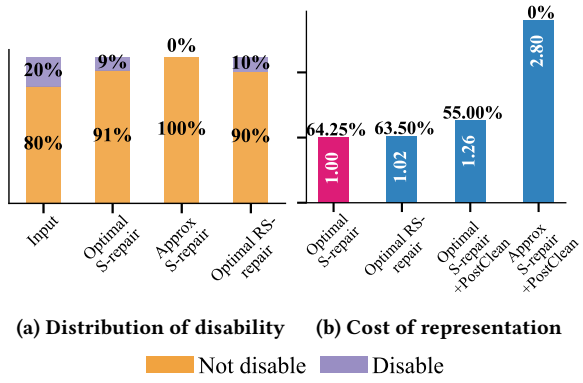



**Figure 9: Cost of representation for ACS data and disability status (representation constraint as {%`not disable` = 90%, %`disable` = 10%})**

# B DETAILS IN SECTION 3

We present the missing details of Section 3 in this section in the following order:

- Proof of Theorem 1 (Section 3.1),
- Proof of Proposition 4 (Section 3.2),
- Simple LHS marriage is at least as hard as color-balanced matching (Section 3.2)
- Pseudocode of PostClean (Section 3.3),
- Proof of Lemma 6 (Section 3.3).

The proof of Theorem 2 is given in Appendix C.6, since it uses the lemmas proved in Appendix C.

## B.1 Proof of Theorem 1

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

Proof. Recall the decision problem of computing an optimal RS-repair. Suppose $\mathcal{S}$ and $\Delta$ are fixed. Given a relation $R$, a RC $\rho$, and a non-negative integer $T$, the answer is yes if and only if there exists an RS-repair that retains at least $T$ tuples.

We show a reduction from 3-satisfiability (3-SAT) to the decision problem of computing the optimal RS-repair.

Consider an input $\phi$ of 3-CNF with clauses $c_1, c_2, \cdots, c_m$ and variables $v_1, v_2, \cdots, v_n$. The goal is to decide if there is an assignment of each variable to $\{0, 1\}$ (i.e., false or true) so that $\phi$ evaluates to true.

We construct an instance for our decision problem as follows. The relation $R$ with schema $\mathcal{S} = (A, B, C)$ has a sensitive attribute $C$. For a variable $v_i$ and a clause $c_j$, if the positive literal $v_i$ appears in $c_j$, we add a tuple $(v_i, 1, c_j)$ to $R$; otherwise if the negative literal $\overline{v_i}$ appears in $c_j$, we add a tuple $(v_i, 0, c_j)$ to $R$. Overall, there are $3m$ tuples in $R$. The FD set $\Delta = \{A \to B\}$. The RC $\rho$ contains the lower-bound constraint $\%c_j \geq \frac{1}{m}$, for every $c_j \in Dom(C)$. Note that $\rho$ is an exact RC since there are $m$ clauses $c_j$ and $m \cdot \frac{1}{m} = 1$, i.e., $\%c_j = \frac{1}{m}$ must hold for every value in $Dom(C)$ if the repair satisfies $\rho$. We set $T = m$.

We show that $\phi$ has a satisfying assignment if and only if $R$ has an RS-repair $R'$ for $\Delta$ and $\rho$ such that $R'$ retains at least $T$ ($= m$) tuples.

*(If).* Consider any RS-repair $R'$ of $R$ w.r.t. $\Delta$ and $\rho$ such that $|R'| \geq T = m$. $R'$ satisfies $\rho$, so there are at least $\lfloor \frac{|R'|}{m} \rfloor \geq 1$ tuples in $R'$ such that $t[C] = c_j$ for each distinct $c_j$. Pick any such tuple $t$. If $t = (v_i, u, c_j)$, we assign the Boolean value $u$ to $v_i$. Since $R'$ satisfies $\Delta$, so the assignment to every variable $v_i$ is unique. The variables that are not retained by $R'$ can be assigned to either 0 or 1. Such assignment of the variables is a satisfying assignment for $\phi$, since clause $c_j$ is satisfied for every $j \in [1, m]$, and consequently $\phi$ evaluates to true.

*(Only if).* Suppose that we have a satisfying assignment for variables $v_1, \ldots, v_n$ so that $\phi$ evaluate to true, represented by Boolean values $u_1, \ldots, u_n$. We form a relation $R'$ with tuples of the form $(v_i, u_i, *)$ where $*$ can take any value. Note that $R'$ satisfies $\Delta = \{A \to B\}$ since each $v_i$ is associated with a unique $u_i$. Since the assignments $v_i = u_i$ for all $i \in [1, n]$ form a satisfying assignment for $\phi$, every clause $c_j$, $1 \leq j \leq m$, is satisfied by at least one literal $v_i$, hence each $C$-value $c_j$ appears at least once in $R'$ and consequently $|R'| \geq m$. For the cases where $|R'| > m$, we ensure that $R'$ has exactly one tuple with value $C = c_j$ for every $c_j \in Dom(C)$ by picking one arbitrary tuple and discarding the rest. Then every value $C = c_j$ for $c_j \in Dom(C)$ appears exactly once, i.e., $\%c_j = \frac{1}{m}$ holds, and therefore $R' \models \rho$.

□

## B.2 Proof of Proposition 4

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

Proof. Since the 'if' direction (an LHS-chain reduces to $\emptyset$ by repeated applications of consensus FD and common LHS) was observed in [35], we show the 'only if' direction here, i.e., show that, if an FD set reduces to $\emptyset$ by repeated applications of consensus FD and common LHS, it must be an LHS-chain.

We prove 'only if' by contradiction. Consider an FD set $\Delta$ that is not an LHS-chain but was reduced to $\emptyset$ by consensus FD and common LHS simplifications. Since $\Delta$ is not an LHS-chain, it has two FDs $f_1 : X_1 \to Y_1$ and $f_2 : X_2 \to Y_2$ such that $X_1 \not\subseteq X_2$ and $X_2 \not\subseteq X_1$. Hence there exists an attribute $A_1 \in X_1$ such that $A_1 \notin X_2$, and an attribute $A_2 \in X_2$ such that $A_2 \notin X_1$. Apply consensus FD and common LHS simplifications on $\Delta$ until no more of either of these two simplifications can be applied. Without loss of generality, suppose $f_1$ was removed altogether from $\Delta$ before $f_2$ was removed by the simplifications. Therefore, the attribute $A_1$ is removed from $\Delta$ either by a common LHS simplification or by a consensus FD simplification. However, at the time point when attribute $A_1$ is about to removed from $\Delta$,

- $A_1$ cannot be removed by a common LHS simplification, because $A_1 \in X_1 \setminus X_2$, and consequently $A_1$ is not in the LHS of $X_2$ (and at that point $f_2$ is still in $\Delta$);
- and $A_1$ cannot be removed by a consensus FD simplification, because at least $A_1$ is still in the LHS of $f_1$,

which contradicts the assumption that $\Delta$ was reduced to $\emptyset$, hence proved. □

## B.3 Reduction from Color-Balanced Matching to Simple LHS Marriage

Continuing Section 3.2, we show a special case for $\mathcal{S}$ and $\Delta$ called *simple LHS marriage*: the schema $\mathcal{S} = \{A, B, C\}$ and the FD set $\Delta = \{A \to B, B \to C, A \to C\}$. Note that when $\mathcal{S}$ and $\Delta$ forms a simple LHS marriage, computing an optimal S-repair for arbitrary $R$ has been shown to be polynomial-time solvable by [35].

Next, we define the problem called *Exact Colored Matching* (as a specific problem that belongs to color-balanced matchings) that has been studied by [24]:

Definition 6 (Exact Colored Matching [24]). *Given an unweighted bipartite graph $G = (U \cup V, E)$, a positive integer $k$ that represents the number of color, a coloring function $c : E \to \{1, 2, \ldots, k\}$ that assigns a single color to each edge in $E$, and a color constraint denoted by a $k$-dimensional vector $\vec{\phi} = (\phi_1, \ldots, \phi_k) \in \mathcal{Z}^k_{\geq 0}$, an subset $M$ of edges of $E$ is an exact colored matching if*

- *$M$ is a matching of $G$,*
- *and for every $i \in [1, k]$, the number of edges of $M$ in color $i$ is exactly $\phi_i$.*

Lemma 12. *Let $\mathcal{S}$ be a fixed schema and $\Delta$ be a fixed FD set such that they are a simple LHS marriage. Suppose that the relation $R$ is duplicate-free, the sensitive attribute $A_s$ is $C$, and the RC $\rho$ is an exact RC. Then, computing an optimal RS-repair is at least as hard as finding an exact colored matching.*

Proof. Recall the decision problem of computing an optimal RS-repair: suppose $\mathcal{S}$ and $\Delta$ are fixed. Given a relation $R$, a RC $\rho$, and a non-negative integer $T$, the answer is yes if and only if there exists an RS-repair that contains at least $T$ tuples.

Given any instance $(G, k, c, \vec{\phi})$ of the exact colored matching problem, we construct an instance of the decision problem of finding an RS-repair for simple LHS marriage as follows:

- A relation $R$ with schema $\mathcal{S} = (A, B, C)$ is defined by
$$\{(u, v, c(e)) \mid \forall e = (u, v) \in E\}.$$
Note that since $G$ is unweighted, $R$ is duplicate-free.
- A FD set $\Delta = \{A \to B, B \to A, A \to C\}$.

- The sensitive attribute $A_s := C$, and its domain is $\{1, \ldots, k\}$.
- A RC $\rho$ is represented by $k$ individual constraints. They are in the form of $\%i \geq p_i$ and $p_i := \frac{\phi_i}{\sum_{i \in [1,k]} \phi_i}$ for every $i \in [1, k]$.
- $T := \sum_{i \in [1,k]} \phi_i$

We claim that there exists an exact colored matching $M$ for $(G, k, c, \vec{\phi})$ if and only if there is an RS-repair $R'$ w.r.t. $\Delta$ and $\rho$ such that $|R'| \geq T$.

*(If).* Given an RS-repair $R'$ of $R$ w.r.t. $\Delta$ and $\rho$ with the size at least $T$, we can construct an exact colored matching $M$ in $G$ as follows: for each tuple $(u, v, w)$ in $R'$, add the edge $(u, v)$ with color $w$ to $M$. $R' \models \Delta$ guarantees that $M$ is a matching. Futhermore, $R' \models \rho$ implies that the number of edges in $M$ with color $i$, $1 \leq i \leq k$ is at least $|R'| \cdot p_i \geq (\sum_{i \in [1,k]} \phi_i) \cdot \frac{\phi_i}{\sum_{i \in [1,k]} \phi_i} = \phi_i$. If the number of edges in $M$ with some color $i$ is greater than $\phi_i$, we can remove arbitrary edges of that color to satisfy the constraint regarding $\phi_i$. After that, $M$ is an exact colored matching.

*(Only if).* Given an exact colored matching $M$, we can construct an RS-repair $R'$: for each edge $(u, v)$ in $M$ with color $i$, add the tuple $(u, v, i)$ to $R'$. Since $M$ is a matching, $R' \models \Delta$. As the number of edges of $M$ in each color is restricted by $\phi$, the number of tuples in $R'$ with each sensitive value $i$, $1 \leq i \leq k$ is exactly $\phi_i$, therefore for every $i \in [1, k]$, $\%i = |\sigma_{C=i} R'| = \frac{\phi_i}{\sum_{i \in [1,k]} \phi_i} = p_i$, indicating that $R' \models \rho$. Moreover, $|R'| = \sum_{i=1}^{k} \phi_i = T$, so $R'$ is an RS-repair that retains at least $T$ tuples. □

The problem of finding an exact colored matching for two colors is known to be in RNC [43]. A recent study by [24] shows that the problem for a constant number $k$ of colors has a randomized polynomial-time algorithm by generalizing the algorithm presented in [43], and hence it is not NP-hard unless NP = RP.

## B.4 Pseudocode of PostClean

In this section we give the pseudocode of the PostClean$(R, \rho)$ procedure from Section 3.3 in Algorithm 6 followed by a walkthrough of the algorithm.

The algorithm computes the size (denoted by $T$) of the maximum subset (denoted by $R'$) of $R$ that satisfies the $\rho$. To achieve this, the main part of Algorithm 6 (lines 1-20) is a loop that enumerates the size $T$ from $|R|$ to 1, which terminates once a feasible $R'$ is found. A Boolean variable $b$ is initialized in line 2. In lines 3-9, we check if there are sufficient number of tuples for every value $A_s = a_\ell$ in $R$. Specifically, $\tau_\ell$ is assigned to the lower-bound value for the number of tuples with $A_s = a_\ell$. Then $\tau_\ell$ is compared with $|\sigma_{A_s = a_\ell} R|$, where the latter denotes the number of tuples in $R$ with value $A_s = a_\ell$. If the lower-bound requirement is larger than the number of available tuples with $A_s = a_\ell$ (line 5), then the verification fails, $b$ is falsified and the inner loop is stopped early. In this case, the current $T$ is not valid, therefore lines 11-19 will be skipped and the procedure will go to next value of $T$ if $T > 1$; otherwise, the loop (lines 1-20) is finished and it is the case where $\emptyset$ is the only valid subset of $R$ that trivially satisfies all the lower-bound constraints. At that point, the $\emptyset$ is returned in line 21.

**Algorithm 6** PostClean$(R, \rho)$

**Input:** A relation $R$ and a RC $\rho$
**Output:** A subset $R' \subseteq R$ of largest size such that $R' \models \rho$

```
1:  for Size T from |R| to 1 do
2:      b ← True;
3:      for Every value a_ℓ in Dom(A_s) do
4:          τ_ℓ ← ⌈T · ρ(a_ℓ)⌉;
5:          if τ_ℓ > |σ_{A_s=a_ℓ} R| then
6:              b ← False;
7:              break
8:          end if
9:      end for
10:     T_0 ← Σ_{ℓ∈[1,k]} τ_ℓ;
11:     if b is True and T_0 ≤ T then
12:         while T_0 < T do
13:             Arbitrarily choose an a_ℓ from Dom(A_s) where the
    corresponding τ_ℓ < |σ_{A_s=a_ℓ} R|);
14:             τ_ℓ ← τ_ℓ + 1;
15:             T_0 ← T_0 + 1;
16:         end while
17:         R' ← ⋃_{a_ℓ∈Dom(A_s)} an arbitrary subset of σ_{A_s=a_ℓ} R of
    size τ_ℓ;
18:         return R';
19:     end if
20: end for
21: return ∅;
```

Next, we discuss the case when $b$ is true for all values $A_s = a_\ell$ for the current value of $T$. In line 10, $T_0$ is computed as the sum of $\tau_\ell$s, representing the minimum size of $R'$ that satisfies every lower-bound constraint. Note that $T_0$ can be larger than $T$, because $\tau_\ell$ takes the ceiling for every $a_\ell$. In that case, it is impossible to derive a $R'$ of size $T$ and the algorithm fails in the condition check in line 11. On the other hand, $T_0$ is smaller than or equal to $T$, representing that $R$ does have a subset that satisfies $\rho$. If $T_0 < T$ (line 12), then there is a slack between the expected size $T$ and the $T_0$ tuples by concatenating $\tau_\ell$ tuples for every $a_\ell$. This slack is fulfilled by retaining additional tuples with arbitrary value $a_\ell$ as long as $R$ has more tuples with $A_s = a_\ell$ than what it already took by that $T_0$ tuples (lines 12-16). In line 17, the algorithm forms $R'$ by concatenating the samples of tuples with $A_s = a_\ell$ for every $a_\ell$ and of size $\tau_\ell$.

## B.5 Proof of Lemma 6

Lemma 6. *Given a relation $R$ and an RC $\rho$, PostClean$(R, \rho)$ returns a maximum subset $R'$ of $R$ in polynomial time such that $R' \models \rho$.*

Proof. We first prove the optimality, and then the polynomial running time.

*(Optimality).* By contradiction, suppose that there is a $R'' \subseteq R$ where $R'' \models \rho$ and $|R''| > |R'|$. Note that $T = |R''|$ is checked earlier than $T = |R'|$ in the outermost for-loop. Hence, $T = |R''|$ failed the validation in Algorithm 6 in line 11, either by $b$ = false or by $T_0 > T$.

(i) If $b$ = false, then there exists an $a_\ell$ where $\tau_\ell > |\sigma_{A_s=a_\ell}R|$. Furthermore, since $R'' \subseteq R$ indicates that $|\sigma_{A_s=a_\ell}R| \geq |\sigma_{A_s=a_\ell}R''|$, we have

$$\tau_\ell > |\sigma_{A_s=a_\ell}R''| \tag{4}$$

Additionally, $R'' \models \rho$ indicates that

$$|\sigma_{A_s=a_\ell}R''| \geq |R''| \cdot \rho(a_\ell). \tag{5}$$

According to the value assignment of $\tau_\ell$, we have

$$\tau_\ell = \lceil T \cdot \rho(a_\ell) \rceil = \lceil |R''| \cdot \rho(a_\ell) \rceil \tag{6}$$

Combining Equation (5) and Equation (6), we have

$$\tau_\ell \leq \lceil |\sigma_{A_s=a_\ell}R''| \rceil. \tag{7}$$

Combining Equation (4) and Equation (7) that are related to $\tau_\ell$, we have

$$\lceil |\sigma_{A_s=a_\ell}R''| \rceil > |\sigma_{A_s=a_\ell}R''|. \tag{8}$$

However, $|\sigma_{A_s=a_\ell}R''|$ is exactly an integer representing the number of tuples with value $a_\ell$ in $R''$, so we have (by removing the ceiling function)

$$|\sigma_{A_s=a_\ell}R''| > |\sigma_{A_s=a_\ell}R''|, \tag{9}$$

which leads to a contradiction.

(ii) Consider the other case that $T_0 > T$. We express $T_0$ and $T$ by $R''$ and $\rho$:

$$T_0 = \sum_{\ell \in [1,k]} \tau_\ell = \sum_{\ell \in [1,k]} \lceil |R''| \cdot \rho(a_\ell) \rceil, \tag{10}$$

$$T = |R''|. \tag{11}$$

By substituting them into the inequality $T_0 > T$, we have

$$\sum_{\ell \in [1,k]} \lceil |R''| \cdot \rho(a_\ell) \rceil > |R''|. \tag{12}$$

Again, $R'' \models \rho$ indicates that $|\sigma_{A_s=a_\ell}R''| \geq |R''| \cdot \rho(a_\ell)$. Combining it with Equation (12), we have

$$\sum_{\ell \in [1,k]} \lceil |\sigma_{A_s=a_\ell}R''| \rceil > |R''|. \tag{13}$$

Since $|\sigma_{A_s=a_\ell}R''|$ is exactly an integer, we can simplify Equation (13) into $\sum_{\ell \in [1,k]} |\sigma_{A_s=a_\ell}R''| > |R''|$, which implies $|R''| > |R''|$, which is a contradiction.

This completes the proof of Optimality.

*(Time complexity).* First, it is important to know that $|R'|$ is bounded by $|R|$. In line 1, we iterate over $T$ in $O(|R|)$. Within the loop, we enumerate all distinct $a_\ell$ in the active domain of $A_s$, which is $O(k) = O(|R|)$. In lines 4-8, we compute $\tau_\ell$ and validate it in $O(1)$. The second part within the loop is the process of distributing the slack, which can be bounded in $O(|R|)$. Finally in line 19, we form $R''$ by sequentially taking samples from $R'$ and concatenations, this step is $O(|R|)$. Overall, the complexity is $O(|R|^2)$. □

# C DETAILS IN SECTION 4

We present the missing details of Section 4 as follows:

- Size of Candidate Sets: Lemma 13
- Proof of Lemma 8 (Section 4)
- Proof of Lemma 10 (Section 4.1)
- Pseudocode of ReprInsert (Section 4.1)
- Proof of Lemma 11 (Section 4.2)
- Proof of Theorem 2 (Section 3.2)

## C.1 Size of Candidate Sets

Lemma 13. *Given a relation $R$, an FD set $\Delta$, and the domain size $k$ of the sensitive attribute $A_s$ of $R$, the size of the candidate set $C_{R,\Delta}$ is $O(|R|^k)$.*

Proof. With the property that there are no two candidates in $C_{R,\Delta}$ that are representatively equivalent to each other and a candidate in $C_{R,\Delta}$ must be a subset of $R$, the size of $C_{R,\Delta}$ is bounded by the maximum number of subsets of $R$ such that no two subsets have the same number of tuples for every $a_\ell \in Dom(A_s)$. By the rule of product[7], it is

$$\prod_{a_\ell \in Dom(A_s)} (|\sigma_{A_s=a_\ell}R| + 1) \tag{14}$$

Specifically, given that the input relation $R$ has $|\sigma_{A_s=a_\ell}R|$ tuples with value $a_\ell$ in attribute $A_s$ for every $a_\ell$, for every $a_\ell \in Dom(A_s)$, $(|\sigma_{A_s=a_\ell}R| + 1)$ means the possible numbers of tuples with $A_s = a_\ell$ from any subsets of $R$, i.e. no tuple with $A_s = a_\ell$, 1 tuple with $A_s = a_\ell$, and so on.

Next, we utilize the AM-GM inequality:

$$\begin{aligned}\prod_{a_\ell \in Dom(A_s)} (|\sigma_{A_s=a_\ell}R| + 1) &\leq \left(\frac{\sum_{a_\ell \in Dom(A_s)} (|\sigma_{A_s=a_\ell}R| + 1)}{k}\right)^k \\ &= \left(\frac{|R| + k}{k}\right)^k \\ &= \left(\frac{|R|}{k} + 1\right)^k\end{aligned} \tag{15}$$

where $|R|$ is the input relation size and $k = |Dom(A_s)|$ is the active domain size of $A_s$, which is fixed. When $|R| \geq 2$ and $k \geq 2$, i.e., any relation with at least two tuples and at least two distinct sensitive values of $A_s$, we have $(\frac{|R|}{k} + 1)^k = O(|R^k|)$. □

## C.2 Proof of Lemma 8

The time complexity of Reduce (Lemma 8) and LhsChain-DP depends on the time complexity of ConsensusReduction (Lemma 14) and that of CommonLHSReduction (Lemma 15). Since Reduce and ConsensusReduction (resp. CommonLHSReduction) call each other recursively, the time complexity analysis is based on recurrence relations. The recurrence relations for Reduce, ConsensusReduction, and CommonLHSReduction are $F_{\text{RDCE}}$, $F_{\text{COSNS}}$, $F_{\text{COLHS}}$ respectively.

### C.2.1 *Time Complexity of ConsensusReduction.*

We analyze the time complexity of ConsensusReduction through a recurrence relation in Lemma 14.

Lemma 14. *The recurrence relation for ConsensusReduction is given by the following expression:*

$$F_{COSNS}(R, \Delta) = \sum_{y_\ell \in Dom(Y)} F_{RDCE}(R_{y_\ell}, \Delta - f) + O(k \cdot |R|^{2k+1}) \tag{16}$$

[7] Given a relation with 3 red tuples, 2 blue tuples, and 1 green tuple. For any subest of this relation, it has 0 red, 1 red, 2 red, or 3 red (4 possibilities), independently 0 blue, 1 blue, or 2 blue (3 possibilities), and independently 0 green, 1 green (2 possibilities). By taking the product $4 \times 3 \times 2 = 24$, there are at most 24 subsets such that each of them has a unique distribution of the color.

*, where $R_{y_\ell} = \sigma_{Y=y_\ell} R$, $f$ is the consensus FD and $Y$ is its RHS, $\Delta - f$ is removing $f$ from $\Delta$, $|R|$ is the relation size, $k$ is the domain size of the sensitive attribute $A_s$, and $F_{RDCE}(\cdot,\cdot)$ is the recurrence relation of Reduce.*

Proof. First of all, the algorithm involves candidate sets on $R_{y_\ell}$, and $R_{y_\ell}$ is a subset of $R$, so $|R_{y_\ell}| \leq |R|$. Without loss of accuracy, the size of the candidate set of $R_{y_\ell}$ is also $O(|R|^k)$ as implied in Lemma 13.

In lines 2-7, ConsensusReduction iterates over each distinct $y_\ell$ in $O(|R|)$. Within the loop, it first computes $C_{R_{y_\ell},\Delta-f}$, which is the candidate set of relation $R_{y_\ell}$ and FD set $\Delta - f$ in line 3. Next, in line 4-6, the algorithm enumerates each candidate $R'$ in $C_{R_{y_\ell},\Delta-f}$ in $O(|R|^k)$, since the size of a candidate set is $O(|R|^k)$ (Lemma 13). For each candidate $R'$, in line 5, the algorithm utilizes ReprInsert to insert $R'$ in $O(k \cdot |R|^k)$ (Lemma 16). Combining them, we obtain Equation (16). □

*C.2.2 Time Complexity of CommonLHSReduction.* The time complexity of CommonLHSReduction is analyzed using a recurrence relation in Lemma 15.

Lemma 15. *The recurrence relation for CommonLHSReduction is given by the following expression:*

$$F_{COLHS}(R,\Delta) = \sum_{x_\ell \in Dom(X)} F_{RDCE}(R_{x_\ell}, \Delta_{-X}) + O(k \cdot |R|^{3k+1}) \quad (17)$$

*, where $R_{x_\ell} = \sigma_{X=x_\ell} R$, $X$ is the common LHS attribute, $\Delta_{-X}$ is removing column $X$ from $\Delta$, $|R|$ is the relation size, $k$ is the domain size of the sensitive attribute $A_s$, and $F_{RDCE}(\cdot,\cdot)$ is the recurrence relation of Reduce.*

Proof. First of all, the algorithm involves candidate sets on $R_{x_\ell}$, $R_{x_1,\ldots,x_{\ell-1}}$ and $R_{x_1,\ldots,x_\ell}$. All of them are subsets of $R$, so their sizes are bounded by $|R|$. Without loss of accuracy, the size of the candidate set of each of them is also $O(|R|^k)$ as implied in Lemma 13.

In lines 1-8, CommonLHSReduction iterates over each distinct value $x_\ell$ in $O(|R|)$. Within the loop, it computes $C_{R_{x_\ell},\Delta_{-X}}$, which is the candidate set of relation $R_{x_\ell}$ and FD set $\Delta_{-X}$ in line 3. Next, in lines 4-7, it enumerates each candidate $R'$ from $C_{R_{x_1,\ldots,x_{\ell-1}},\Delta}$ and each candidate $R''$ from $C_{R_{x_\ell},\Delta}$. Note that the size of candidate set is $O(|R|^k)$ (Lemma 13), so this enumeration will be $O(|R|^{2k})$. In line 5, it compute the union $R_0$ of $R'$ and $R''$ in $O(|R|)$. In line 6, it inserts $R_0$ into the candidate set $C_{R_{x_1,\ldots,x_\ell},\Delta}$ by ReprInsert in $O(k \cdot |R|^k)$ (Lemma 16). Combining them, we obtain Equation (17). □

*C.2.3 Time Complexity of Reduce and LhsChain-DP.* In Lemma 8, we present the time complexity of LhsChain-DP, including the time complexity of Reduce, which serves as the core sub-procedure of LhsChain-DP.

Lemma 8. *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 $A_s$.*

Proof. The complexity of LhsChain-DP consists of three parts:

(1) Reduce computes a candidate set in $O(m \cdot |\Delta| \cdot k \cdot |R|^{3k+2})$ (proof below).
(2) PostClean computes a maximum subset that satisfies the $\rho$ in $O(|R|^2)$ (Lemma 6) for every candidate from a candidate set of size $O(|R|^k)$ (Lemma 13). In total, PostClean runs in $O(|R|^{k+2})$.
(3) The selection of the maximum RS-repair after post-cleaning is linear in the size of the candidate set, $O(|R|^k)$.

For part (1), Reduce is an if/else statement with three branches. Therefore, the complexity of Reduce consists of two parts: if-condition checking and branching. The if-condition checking part checks:

- if $\Delta$ is empty in $O(1)$;
- otherwise, if $\Delta$ has a consensus FD by going ever each FD in $\Delta$ in $O(|\Delta|)$;
- otherwise, if $\Delta$ has a common LHS by checking each attribute for common columns in the LHS of each FD in $\Delta$ in $O(m \cdot |\Delta|)$.

Depending on the results of condition checking, the recurrence relation of Reduce has three possibilities:

- if $\Delta$ is empty, then the algorithm simply returns $\{R\}$ and

$$F_{\text{RDCE}}(R,\Delta) = O(|R|); \quad (18)$$

- otherwise, if $\Delta$ has a consensus FD, the sub-procedure ConsensusReduction is called, as implied by Lemma 14 and Equation (16),

$$\begin{aligned} F_{\text{RDCE}}(R,\Delta) &= F_{\text{COSNS}}(R,\Delta) \\ &= \sum_{y_\ell \in Dom(Y)} F_{\text{RDCE}}(R_{y_\ell}, \Delta - f) + O(k \cdot |R|^{2k+1}); \end{aligned} \quad (19)$$

- otherwise, if $\Delta$ has a common LHS, the sub-routine CommonLHSReduction is called, implied by Lemma 15 and Equation (17),

$$\begin{aligned} F_{\text{RDCE}}(R,\Delta) &= F_{\text{COLHS}}(R,\Delta) \\ &= \sum_{x_\ell \in Dom(X)} F_{\text{RDCE}}(R_{x_\ell}, \Delta_{-X}) + O(k \cdot |R|^{3k+1}). \end{aligned} \quad (20)$$

Intuitively, Reduce can be viewed as a tree traversal, where each node is a call of Reduce and the root corresponds to Reduce$(R,\Delta)$. Note that the call of Reduce by each leaf node is on a clean sub-relation and empty $\Delta$. The number of leaves is bounded by $|R|$, as the associated sub-relations are disjoint. The tree height, equivalent to the number of reductions applied, is bounded by $O(m \cdot |\Delta|)$ as at least one attribute is removed from the $\Delta$ per reduction. The number of nodes in each level must be larger than the number of nodes in the parent level, so the tree size is bounded by $O(m \cdot |\Delta| \cdot |R|)$ and therefore $F_{\text{RDCE}}(R,\Delta)$ is equal to the sum of the Big O terms of the associated $F_{\text{RDCE}}(\cdot,\cdot)$ of all non-root tree nodes. Therefore, by combining Equations (18) to (20), $F_{\text{RDCE}}(R,\Delta)$ is bounded by $O((m \cdot |\Delta| \cdot |R|) \cdot (k \cdot |R|^{3k+1}))$. To sum up, the branching step is $O(m \cdot |\Delta| \cdot k \cdot |R|^{3k+2})$. Compared to this, the time complexity of if-condition checking, $O(m \cdot |\Delta|)$, can be ignored.

Overall, the time complexity of LhsChain-DP is dominated by Reduce, which is $O(m \cdot |\Delta| \cdot k \cdot |R|^{3k+2})$. □

## C.3 Proof of Lemma 10

Lemma 10. *For any relation $R$ and any FD set $\Delta$, if $C_{R,\Delta}$ is computed correctly in Line 1, LhsChain-DP (Algorithm 1) returns an optimal RS-repair of $R$ w.r.t. $\Delta$ and $\rho$.*

Proof. Algorithm 1 returns $\arg\max_{s \in S} |s|$ as the output RS-repair where $S$ is defined as follows:

$$S := \{\mathsf{PostClean}(R', \rho) \mid \forall R' \in C_{R,\Delta}\}, \tag{21}$$

Since $C_{R,\Delta} \subseteq \mathcal{A}_{R,\Delta}$, each $R' \in C_{R,\Delta}$ is an S-repair. Moreover, since the final output of Algorithm 1 is obtained by applying PostClean on such S-repairs, by Lemma 6, the output satisfies the RC $\rho$ and thus is an RS-repair. Next, we prove the optimality of the final answer when $C_{R,\Delta}$ is correctly computed.

We argue that applying Equation (21) on $\mathcal{A}_{R,\Delta}$ instead of $C_{R,\Delta}$ yields the optimal RS-repair. To see this, note that any RS-repair is also an S-repair, so an optimal RS-repair $S^*$ must belong to $\mathcal{A}_{R,\Delta}$ that already satisfies the RC $\rho$. If we return an RS-repair $S^0$ by applying PostClean on every S-repair in $\mathcal{A}_{R,\Delta}$ and choosing the one with the maximum size, $|S^0| \geq |S^*|$, and since $S^*$ is the optimal RS-repair, $|S^0| = |S^*|$. Hence the final output $S^0$ would be an optimal RS-repair.

Finally, we argue that S-repairs in $\mathcal{A}_{R,\Delta} \setminus C_{R,\Delta}$ cannot derive an RS-repair that deletes strictly fewer tuples. Consider any S-repair $R'' \in \mathcal{A}_{R,\Delta} \setminus C_{R,\Delta}$. By the candidate set definition (Definition 5), $R'' \notin C_{R,\Delta}$ due to one of the following reasons:

(1) There exists an $R' \in C_{R,\Delta}$ where $R' =_{Repr} R''$. In this case, we argue that $|\mathsf{PostClean}(R'', \rho)| = |\mathsf{PostClean}(R', \rho)|$. Because $R''$ and $R'$ will obtain the same $\{\tau_1, \ldots, \tau_k\}$, which are the minimum numbers required for each sensitive value to satisfy the RC, in PostClean regardless of which $T$ is enumerated, therefore PostCleanwill return a RS-repair with the same number of tuples for each distinct value of the sensitive attribute $A_s$ and consequently the same $T$ for $R'$ and $R''$ respectively.

(2) There exists an $R' \in C_{R,\Delta}$ where $R' \succ_{Repr} R''$. In this case, we argue that $|\mathsf{PostClean}(R'', \rho)| \geq |\mathsf{PostClean}(R', \rho)|$. Since there exists some $a_c$ where

$$|R'| \cdot |\sigma_{A_s = a_c} R'| > |R''| \cdot |\sigma_{A_s = a_c} R''|, \tag{22}$$

we first take tuples with $A_s$-value $a_c$ in $R'$ away as a backup, until $R'$ has the same value distribution of $A_s$ as $R''$. As implied by Item 1 above, now $R'$ and $R''$ will generate RS-repairs with the same $T$. Moreover, $R'$'s backup offers $R'$ a chance to retain more tuples (consequently deleting fewer tuples) while not conflicting $\rho$.

Therefore, no matter how we satisfy the RC by additional deletions (through PostClean) on $R''$, we can do the same or even better on some candidate in $C_{R,\Delta}$. Hence, $C_{R,\Delta}$ is sufficient to compute the RS-repair as the entire $\mathcal{A}_{R,\Delta}$ does. □

## C.4 Pseudocode of ReprInsert

Algorithm 7 considers an insertion of a candidate $R^*$ into a set $C$ of candidates and eliminates any representative dominance and representative equivalence. First, in line 1, the algorithm enumerates each candidate $R'$ in $C$. Then in line 2, it checks if $R'$ representatively dominates $R^*$ or is representatively equivalent to $R^*$. If either condition is true, indicating that $R^*$ must be redundant or sub-optimal, ReprInsert returns the current $C$. Conversely, if $R^*$ representatively dominates $R'$ (line 4), $R'$ is removed from $C$, and the loop continues. Finally, the updated $C$ is returned in line 8.

**Algorithm 7** ReprInsert$(C, R^*)$

**Input:** a set $C$ of candidates where no representative dominance or representative equivalence exist, a candidate $R^*$ to be inserted
**Output:** the set of candidates after the insertion

1: **for** every $R' \in C$ **do**
2: **if** $R' \succ_{Repr} R^*$ or $R' =_{Repr} R^*$ **then**
3: **return** $C$;
4: **else if** $R^* \succ_{Repr} R'$ **then**
5: $C \leftarrow C \setminus \{R'\}$;
6: **end if**
7: **end for**
8: **return** $C \cup R^*$;

Lemma 16. *ReprInsert terminates in $O(k \cdot |R|^k)$, where $k$ is the domain size of the sensitive attribute and $|R|$ is the input relation size.*

Proof. In line 1, we iterate over a set of candidates in $O(|R|^k)$ as implied by Lemma 13. Within the loop, we sequentially check the conditions (line 2 and line 4) in $O(k)$. Therefore, the overall time complexity is $O(k \cdot |R|^k)$. □

## C.5 Proof of Lemma 11

In Lemma 11, we prove the correctness of Reduce, which relies on the correctness of ConsensusReduction and the correctness of CommonLHSReduction. Figures 10 and 11 show the workflow of these two reductions and repeat the notations used in the algorithms.

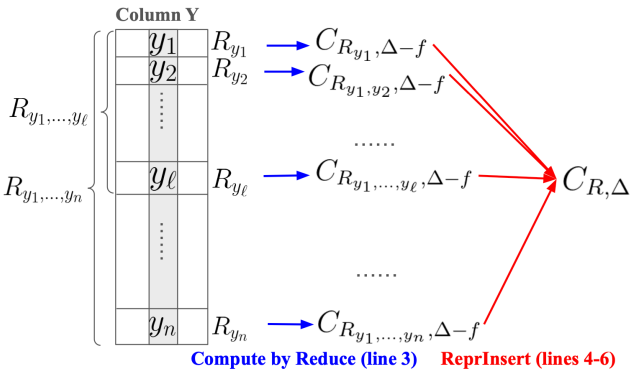



**Figure 10: Notations and Workflow of ConsensusReduction. $f$ is the consensus FD and $Y$ is its RHS. $y_1, \ldots, y_n$ are distinct values of column $Y$. $R_{y_\ell} = \sigma_{Y=y_\ell} R$ for every $\ell \in [1, n]$. $C_{R_{y_\ell},\Delta-f}$ is the candidate set of relation $R_{y_\ell}$ and FD set $\Delta - f$. $C_{R,\Delta}$ is the output of ConsensusReduction.**

### C.5.1 Correctness of Reduce (Lemma 11).

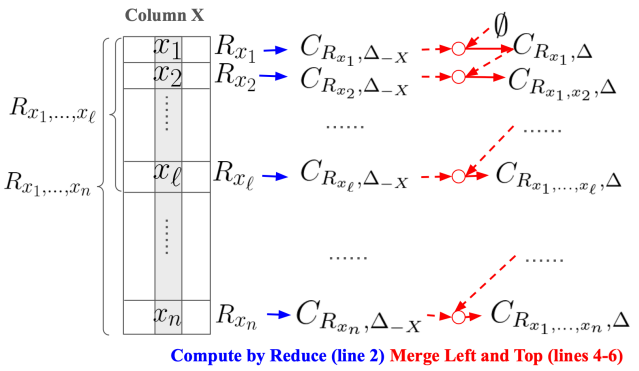



**Figure 11: Notations and Workflow of CommonLHSReduction. $X$ is the common LHS column. $x_1, \ldots, x_n$ are the distinct values of column $X$. $R_{x_\ell} = \sigma_{X=x_\ell}R$ for every $\ell \in [1,n]$. $Rx_1, \ldots, x_\ell = \sigma_{X=x_1 \vee \cdots \vee X=x_\ell}R$ for every $\ell \in [1,n]$. $C_{Rx_\ell,\Delta_X}$ is the candidate set of relation $Rx_\ell$ and FD set $\Delta$. $C_{Rx_1,\ldots,x_\ell,\Delta_{-X}}$ is the candidate set of relation $Rx_1, \ldots, x_\ell$ and FD set $\Delta_{-X}$ (i.e. removing $X$ from $\Delta$). $C_{R,\Delta}$ is the output of CommonLHSReduction.**

Lemma 11. *Given relation $R$ and FD set $\Delta$ that forms an LHS-chain, Reduce$(R,\Delta)$ correctly computes the candidate set $C_{R,\Delta}$.*

Proof. We prove it by induction on $w$, the number of reductions applied to $\Delta$ by Reduce. Note that because Reduce works on LHS chains, $\Delta$ must be able to reduced to the $\emptyset$ with a finite $w$. Assume that at each step of $w$, Reduce works on $R$ and $\Delta$, which can be the same as inputted or different, i.e. a sub-relation and a portion of the input FD set.

In the base case of induction, when $w = 0$, the FD set $\Delta$ must be empty, so $R$ is the only candidate in $C_{R,\Delta}$, which will be returned by Reduce.

For the inductive step, assume Reduce works for all $w \in [0, \beta-1]$, we prove that it also works for $w = \beta$. In this case, Reduce$(R,\Delta)$ will apply one of the following reductions:

*(If $\Delta$ has a consensus FD $f : \emptyset \rightarrow Y$).* the condition of line 6 is satisfied and the sub-routine ConsensusReduction is called. Similarly, ConsensusReduction depends on Reduce$(R_{y_\ell}, \Delta - f)$ for every $y_\ell \in Dom(Y)$, whom work because they applies $w-1$ reductions. Next, we argue that ConsensusReduction derives a $C_{R,\Delta}$ from the results of Reduce$(R_{y_\ell}, \Delta - f)$ for every $y_\ell$ in $Dom(Y)$: we first discuss the relationship between $C_{R,\Delta}$ and $\bigcup_{\forall y_\ell \in Dom(Y)} C_{R_{y_\ell},\Delta-f}$.

For any candidate $R'$ from $C_{R,\Delta}$, since $R'$ is a S-repair of $(R,\Delta)$:

- $R' \models \{f\}$, implying all tuples in $R'$ agree on the same $Y$-value (say $y_\ell$ w.l.o.g.) and consequently $R'$ is also an S-repair of $R_{y_\ell}$.
- We claim that $R'$ is not dominated by any other candidates: otherwise if there exists an $R''$ from $C_{R_{y_\ell},\Delta-f}$ such that $R'' \succ_{Repr} R'$, then $R''$ must also be a candidate of $C_{R,\Delta}$, contradicting the assumption that $R'$ is a candidate of $C_{R,\Delta}$.
- $R' \models \Delta - f$.

Combining these three points, we claim that $R'$ either is a member of $C_{R_{y_\ell},\Delta-f}$ (and consequently $\bigcup_{\forall y_\ell \in Dom(Y)} C_{R_{y_\ell},\Delta-f}$) or shares the same distribution of $A_s$ values with another candidate in $C_{R_{y_\ell},\Delta-f}$ (and consequently $\bigcup_{\forall y_\ell \in Dom(Y)} C_{R_{y_\ell},\Delta-f}$).

Hence, $C_{R,\Delta}$ can be derived from $\bigcup_{\forall y_\ell \in Dom(Y)} C_{R_{y_\ell},\Delta-f}$ by inserting each candidate through ReprInsert, because

- there is neither representative dominance nor representative equivalence in $C_{R,\Delta}$ (guaranteed by ReprInsert),
- any candidate $R'' \in \mathcal{A}_{R,\Delta} \setminus C_{R,\Delta}$ must be either representatively dominated by or representatively equivalent to one candidate in $C_{R,\Delta}$, because $R'' \in C_{R_{\pi_Y R''},\Delta-f} \subseteq \bigcup_{\forall y_\ell \in Dom(Y)} C_{R_{y_\ell},\Delta-f}$ as $R'' \models f$. Therefore, according to ReprInsert, $R''$, as a candidate from $\bigcup_{\forall y_\ell \in Dom(Y)} C_{R_{y_\ell},\Delta-f}$, does not exists in $C_{R,\Delta}$ only because of one the two reasons mentioned above.

*(If $\Delta$ has a common LHS $X$).* CommonLHSReduction is called (corresponding to lines 3-4). By the induction hypothesis, CommonLHSReduction$(R,\Delta)$ relies on Reduce$(R_{x_\ell}, \Delta_{-X})$ for each $x_\ell \in Dom(X)$. Note that those smaller instances work because they applies $w-1$ reductions. Furthermore, we argue that CommonLHSReduction derives a $C_{R,\Delta}$ from the results of Reduce$(R_{x_\ell}, \Delta_{-X})$ for every $x_\ell$ in $Dom(X)$. To achieve this, for each $\ell \in [1,n]$, $C_{R_{x_1,\ldots,x_\ell},\Delta}$ is derived from $C_{R_{x_1,\ldots,x_{\ell-1}},\Delta}$ and $C_{R_{x_\ell},\Delta_{-X}}$ by inserting each candidate in

$$\{R' \cup R'' \mid R' \in C_{R_{x_\ell},\Delta_{-X}}, R'' \in C_{R_{x_1,\ldots,x_{\ell-1}},\Delta}\}$$

into $C_{R_{x_1,\ldots,x_\ell},\Delta}$ using ReprInsert by CommonLHSReduction. We prove it by induction on $\ell$.

When $\ell = 1$, we argue that for any candidate $R'_1$ from $C_{R_{x_1},\Delta}$, there must exist an $R'_2$ from $C_{R_{x_1},\Delta_{-X}}$ where $R'_1 =_{Repr} R'_2$, because:

- $R'_1$ is an S-repair of $R_{x_1}$ as $R'_1$ has no violation on $\Delta$ as well as $\Delta_{-X}$.
- $R'_1$ is not dominated by any candidate in $C_{R_{x_1},\Delta_{-X}}$. Otherwise, there must exist a candidate from $C_{R_{x_1},\Delta}$ also dominates $R'_1$, which conflicts with the assumption.
- $R'_1$ is not in $C_{R_{x_1},\Delta_{-X}}$ only if $R'_1$ shares the same value distribution of $A_s$ with someone else.

This step guarantees the properties of a candidate set by ReprInsert as well.

When $(\ell > 1)$, for any candidate $\bar{R}$ from $C_{R_{x_1,\ldots,x_\ell},\Delta}$, we argue that:

- $\bar{R} = (\bar{R} \cap R_{x_\ell}) \cup (\bar{R} \cap R_{x_1,\ldots,x_{\ell-1}})$, because $R_{x_\ell}$ and $R_{x_1,\ldots,x_{\ell-1}}$ are two separate parts of $R$.
- $\bar{R} \cap R_{x_\ell}$ is a candidate of $(R_{x_\ell}, \Delta_{-X})$ (or shares a value distribution of $A_s$ with another candidate),
- and $\bar{R} \cap R_{x_1,\ldots,x_{\ell-1}}$ is a candidate of $(R_{x_1,\ldots,x_{\ell-1}}, \Delta)$ (or shares a value distribution of $A_s$ with another candidate).

These three bullet points are proved by: firstly, an intersection with $\bar{R}$ is equivalent to taking a subset of $\bar{R}$, and we know that a subset of an S-repair is still an S-repair over the entire relation $R$ as well as the sub-relation $R_{x_\ell}$ (resp. $R_{x_1,\ldots,x_{\ell-1}}$). Secondly, $\bar{R} \cap R_{x_\ell}$ (resp. $\bar{R} \cap R_{x_1,\ldots,x_{\ell-1}}$) is not representatively dominated by any

other candidate in $C_{R_{x_\ell},\Delta_{-X}}$ (resp. $C_{R_{x_1,\ldots,x_{\ell-1}},\Delta}$). Otherwise, if such a candidate $R_0$ exists in $C_{R_{x_\ell},\Delta_{-X}}$ (resp. $C_{R_{x_1,\ldots,x_{\ell-1}},\Delta}$), then $R_0 \cup (\bar{R} - R_{x_\ell})$ must be a candidate of $C_{R_{x_1,\ldots,x_\ell},\Delta}$ that representatively dominates $\bar{R}$, resulting in the contradiction with the definition of $\bar{R}$. Finally, the only reason that $\bar{R}$ is not included in

$$\{R' \cup R'' \mid \forall R' \in C_{R_{x_\ell},\Delta_{-X}}\ \forall R'' \in C_{R_{x_1,\ldots,x_{\ell-1}},\Delta}\}$$

is that it shares the same distribution of $A_s$ values with another candidate in this candidate set. Similarly, this step guarantees the properties of a candidate set by ReprInsert as well.

Combining $\ell = 1$ and $\ell > 1$, we prove that CommonLHSReduction computes $C_{R_{x_1,\ldots,x_\ell},\Delta}$ for every $\ell \in [1, n]$. Since $R_{x_1,\ldots,x_n} = R$, we have $C_{R,\Delta} \leftarrow C_{R_{x_1,\ldots,x_n},\Delta}$.

Therefore, Reduce is correct when $w = \beta$.

□

## C.6 Proof of Theorem 2

Theorem 2. *Let $\mathcal{S}$ be a fixed schema and $\Delta$ 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.*

Proof. We first prove the correctness and then the time complexity of LhsChain-DP.

*(Correctness).* We argue that LhsChain-DP computes the optimal RS-repair when $\rho$ is fixed and $\Delta$ forms an LHS chain. The proof consists of two parts: (a) Reduce correctly computes the candidate set $C_{R,\Delta}$ (proved in Lemma 11) and (b) Lines 2-3 of LhsChain-DP compute the optimal RS-repair by applying PostClean to $C_{R,\Delta}$ (proved in Lemma 10).

*(Time complexity).* The time complexity of LhsChain-DP is $O(m \cdot |\Delta| \cdot k \cdot |R|^{3k+2})$ as implied by Lemma 8. □

# D DETAILS FROM SECTION 5

## D.1 LP-based Rounding Algorithms

**Algorithm 8** LP + GreedyRounding$(R, \Delta, \rho)$

1: Build and optimize LP;
2: **if** Find a solution $\vec{x} \in \mathbb{R}^{|R|}$ **then**
3: **while** $\exists\ 0 < x_i < 1$ **do**
4: Identify $x_i$ that is involved in the smallest number of LP constraints;
5: $x_i \leftarrow 1$ and $x_j \leftarrow 0$ for all $j$ where there is a constraint $x_i + x_j \leq 1$ exists;
6: **end while**
7: $R' \leftarrow \{t_i \mid x_i = 1, \forall i \in [1, |R|]\}$;
8: **return** PostClean$(R', \rho)$;
9: **end if**
10: **return** $\emptyset$;

*D.1.1 Greedy Rounding (LP + GreedyRounding).* A greedy algorithm is proposed in Algorithm 8 that guarantees no violations of FDs occur and greedily retains more (deletes fewer) tuples in the repair. The algorithm ends with PostClean to ensure satisfying the RC. It is important to note that representation is not considered before line 8. The time cost consists of two parts. The time cost of building and optimizing the LP depends on the number of constraints, which is at most $O(|R|^2)$ in our case. On the other hand, the rounding step consists of an enumeration of undecided variables and their neighbors, where the complexity of the rounding step is $O(|R|^2)$.

However, we have observed performance degradation with the greedy rounding approach, especially when dealing with datasets containing a large number of errors. The greedy rounding fails to preserve the minor sub-groups because the central focus of rounding is primarily on maximizing the number of tuples retained.

**Algorithm 9** LP + ReprRounding$(R, \Delta, \rho)$

1: Build and optimize LP;
2: **if** Find a solution $\vec{x} \in \mathbb{R}^{|R|}$ **then**
3: **while** $\exists\ 0 < x_i < 1$ **do**
4: $a_\ell \leftarrow \arg\min_{a_\ell \in Dom(A_s)} \frac{\sum_{t_i[A_s]=a_\ell} x_i}{p_\ell}$;
5: Identify $x_i$ with $t_i[A_s] = a_\ell$ and involved in the smallest number of LP constraints;
6: $x_i \leftarrow 1$ and $x_j \leftarrow 0$ for all $j$ where there is a constraint $x_i + x_j \leq 1$ exists;
7: **end while**
8: $R' := \{t_i \mid x_i = 1, \forall i \in [1, |R|]\}$;
9: **return** PostClean$(R', \rho)$;
10: **end if**
11: **return** $\emptyset$;

*D.1.2 Representation-Aware Rounding (LP + ReprRounding).* To alleviate this issue in LP + GreedyRounding, we propose a representation-aware enhancement in LP + ReprRounding as outlined in Algorithm 9. The primary modification lies in lines 4 and 5, where considerations of representation are incorporated by prioritizing the selection of tuples from underrepresented sensitive groups. Inspired by stratified sampling in [44], we introduce $\frac{\sum_{t_i[A_s]=a_\ell} x_i}{p_c}$, representing the ratio of the tuple with a given $A_s$-value $a_\ell$ to the expected proportion $p_\ell$. A smaller ratio indicates a less representative group in the retained tuples. From these less representative groups, LP + ReprRounding chooses $x_i$ associated with the fewest conflict constraints, then employing similar techniques as LP + GreedyRounding does. While the computational cost remains primarily on solving the LP for most cases, the rounding takes slightly longer than that in LP + GreedyRounding due to the additional representation considerations in this step. Similarly, the complexity of this rounding is $O(|R|^2)$.

# E DETAILS IN SECTION 6

We present the repair qualities for chain FD sets with larger noise (15% and 20%) in **??**. The repair qualities of ILP-Baseline+ PostClean, VC-approx-Baseline+PostClean, DP-Baseline+ PostClean, MuSe-Baseline+PostCleandrop as the noise level increases. Tables 3 to 8 display detail results of different sizes and noise levels for both chain and non-chain FD sets.

**Table 3: Repair quality (%) of ACS: chain FD set**

| noise | 5% | | | | 10% | | | | 15% | | | | 20% | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| size | 4k | 6k | 8k | 10k | 4k | 6k | 8k | 10k | 4k | 6k | 8k | 10k | 4k | 6k | 8k | 10k |
| **MuSe-Baseline+PostClean** | 92.87 | 94.31 | 94.79 | | 77.80 | 80.64 | | | | | | | | | | |
| **VC-approx-Baseline+PostClean** | 93.48 | 93.73 | 94.62 | 94.65 | 76.54 | 79.57 | 80.60 | 80.90 | 47.47 | 57.54 | 57.82 | 57.54 | 23.27 | 23.30 | 21.23 | 17.96 |
| **DP-Baseline+PostClean** | 93.48 | 93.73 | 94.62 | 94.65 | 76.54 | 79.57 | 80.60 | 80.90 | 47.47 | 57.54 | 57.82 | 57.54 | 23.27 | 23.30 | 21.23 | 17.96 |
| **ILP-Baseline+PostClean** | 93.48 | 93.73 | 94.49 | 94.65 | 76.89 | 79.57 | 80.60 | 80.90 | 47.47 | 55.81 | 58.26 | 56.83 | 22.44 | 24.55 | 20.21 | 18.78 |
| **LHSChain-DP** | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 |
| **GlobalILP** | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 |

**Table 4: Repair quality (%) of COMPAS: chain FD set**

| noise | 5% | | | | 10% | | | | 15% | | | | 20% | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| size | 4k | 10k | 20k | 30k | 4k | 10k | 20k | 30k | 4k | 10k | 20k | 30k | 4k | 10k | 20k | 30k |
| **MuSe-Baseline+PostClean** | 98.83 | | | | 98.93 | | | | | | | | | | | |
| **VC-approx-Baseline+PostClean** | 99.61 | 100.00 | 100.00 | 100.00 | 97.96 | 98.60 | 99.19 | 99.01 | 93.84 | 95.77 | 95.29 | 96.95 | 91.04 | 92.35 | 92.40 | 93.50 |
| **DP-Baseline+PostClean** | 99.61 | 100.00 | 100.00 | 100.00 | 97.96 | 98.60 | 99.19 | 99.01 | 93.84 | 95.77 | 95.29 | 96.95 | 91.04 | 92.35 | 92.40 | 93.50 |
| **ILP-Baseline+PostClean** | 99.61 | 100.00 | 100.00 | 100.00 | 97.96 | 98.60 | 99.19 | 99.01 | 93.84 | 95.77 | 95.29 | 96.95 | 91.04 | 92.35 | 92.40 | 93.50 |
| **LHSChain-DP** | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 |
| **GlobalILP** | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 |

**Table 5: Runtime cost (in minutes) of ACS: chain FD set**

| noise | 5% | | | | 10% | | | | 15% | | | | 20% | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| size | 4k | 6k | 8k | 10k | 4k | 6k | 8k | 10k | 4k | 6k | 8k | 10k | 4k | 6k | 8k | 10k |
| **MuSe-Baseline** | 460.05 | 3904.73 | 15161.43 | | 4558.15 | 16187.80 | | | | | | | | | | |
| **VC-approx-Baseline** | 0.42 | 0.50 | 0.55 | 0.62 | 0.54 | 0.66 | 0.71 | 0.75 | 0.63 | 0.71 | 0.75 | 0.85 | 0.68 | 0.74 | 0.80 | 0.83 |
| **DP-Baseline** | 0.41 | 0.50 | 0.59 | 0.62 | 0.55 | 0.64 | 0.69 | 0.72 | 0.63 | 0.71 | 0.81 | 0.81 | 0.74 | 0.89 | 0.88 | 0.86 |
| **ILP-Baseline** | 1.17 | 2.74 | 5.78 | 8.15 | 2.24 | 4.85 | 9.65 | 15.95 | 2.93 | 6.88 | 15.33 | 33.62 | 7.81 | 11.35 | 38.87 | 31.69 |
| **LHSChain-DP** | 3.49 | 4.80 | 8.41 | 9.06 | 5.64 | 6.75 | 9.84 | 12.12 | 6.42 | 7.34 | 11.39 | 12.01 | 8.78 | 8.84 | 14.96 | 14.04 |
| **GlobalILP** | 5.02 | 13.37 | 94.12 | 90.16 | 5.74 | 64.79 | 413.00 | 1045.77 | 23.60 | 83.43 | 683.59 | 482.75 | 25.20 | 421.60 | 734.71 | 327.48 |

**Table 6: Runtime cost (in minutes) of COMPAS: chain FD set**

| noise | 5% | | | | 10% | | | | 15% | | | | 20% | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| size | 4k | 10k | 20k | 30k | 4k | 10k | 20k | 30k | 4k | 10k | 20k | 30k | 4k | 10k | 20k | 30k |
| **MuSe-Baseline** | 11071.22 | | | | 38218.90 | | | | | | | | | | | |
| **VC-approx-Baseline** | 0.03 | 0.04 | 0.05 | 0.07 | 0.03 | 0.04 | 0.06 | 0.07 | 0.03 | 0.04 | 0.05 | 0.07 | 0.03 | 0.04 | 0.05 | 0.07 |
| **DP-Baseline** | 0.03 | 0.04 | 0.05 | 0.07 | 0.03 | 0.04 | 0.05 | 0.08 | 0.03 | 0.04 | 0.06 | 0.11 | 0.03 | 0.04 | 0.05 | 0.09 |
| **ILP-Baseline** | 2.09 | 18.06 | 86.95 | 177.53 | 4.31 | 43.77 | 187.56 | 401.77 | 6.70 | 65.74 | 336.20 | 770.82 | 8.92 | 79.95 | 331.32 | 932.78 |
| **LHSChain-DP** | 0.04 | 0.06 | 0.07 | 0.09 | 0.05 | 0.07 | 0.08 | 0.16 | 0.05 | 0.07 | 0.11 | 0.15 | 0.06 | 0.11 | 0.11 | 0.28 |
| **GlobalILP** | 2.48 | 21.12 | 96.14 | 211.90 | 5.32 | 65.48 | 226.84 | 443.41 | 8.33 | 179.88 | 463.69 | 1376.47 | 11.72 | 202.19 | 390.46 | 1590.58 |

**Table 7: Repair quality (%) of ACS: non-chain FD set**

| noise | 1% | | | | | 5% | | | | | 10% | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| size | 2k | 4k | 6k | 8k | 10k | 2k | 4k | 6k | 8k | 10k | 2k | 4k | 6k | 8k | 10k |
| **VC-approx-Baseline+PostClean** | 69.81 | 68.82 | 73.84 | 74.01 | 73.68 | 1.94 | 0.43 | 6.34 | 6.72 | 5.09 | 1.68 | 0.00 | 0.00 | 0.00 | 0.00 |
| **ILP-Baseline+PostClean** | 97.05 | 97.45 | 97.78 | 97.72 | 97.66 | 80.83 | 79.34 | 84.93 | 85.43 | 86.01 | 47.49 | 29.60 | 45.57 | 46.78 | 73.62 |
| **LP+GreedyRounding** | 99.85 | 99.55 | 100.00 | 99.86 | 99.77 | 90.28 | 83.24 | 91.10 | 92.16 | 91.63 | 68.16 | 67.82 | 67.54 | 66.49 | 77.07 |
| **GlobalILP** | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 |
| **LP+ReprRounding** | 100.00 | 99.55 | 100.00 | 99.89 | 99.80 | 85.28 | 81.21 | 90.07 | 90.71 | 90.51 | 64.25 | 64.94 | 66.39 | 69.25 | 79.14 |
| **FDCleanser** | 99.41 | 98.95 | 99.52 | 99.42 | 99.19 | 97.22 | 96.68 | 98.63 | 98.75 | 99.15 | 91.06 | 94.25 | 95.41 | 96.19 | 99.14 |

**Table 8: Runtime cost (in minutes) of ACS: non-chain FD set**

| noise | 1% | | | | | 5% | | | | | 10% | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| size | 2k | 4k | 6k | 8k | 10k | 2k | 4k | 6k | 8k | 10k | 2k | 4k | 6k | 8k | 10k |
| **LP+GreedyRounding** | 2.79 | 15.67 | 81.07 | 43.29 | 74.63 | 4.05 | 20.22 | 50.97 | 86.22 | 157.23 | 6.37 | 31.01 | 74.40 | 139.08 | 166.43 |
| **LP+ReprRounding** | 5.69 | 15.47 | 100.46 | 92.27 | 158.59 | 8.16 | 31.51 | 69.82 | 117.88 | 206.39 | 9.66 | 33.83 | 87.84 | 165.72 | 278.52 |
| **FDCleanser** | 0.75 | 1.16 | 1.32 | 2.10 | 2.35 | 1.12 | 2.53 | 2.43 | 3.57 | 4.33 | 1.06 | 2.29 | 2.48 | 2.52 | 2.36 |
| **GlobalILP** | 2.06 | 14.23 | 86.99 | 235.19 | 547.00 | 13.64 | 313.49 | 1003.48 | 3188.69 | 17840.45 | 63.39 | 1869.85 | 5072.64 | 58977.39 | 64039.06 |
| **VC-approx-Baseline** | 0.18 | 0.52 | 0.91 | 1.59 | 2.58 | 0.35 | 3.20 | 3.82 | 7.38 | 14.02 | 0.61 | 3.19 | 7.13 | 13.88 | 20.72 |
| **ILP-Baseline** | 1.08 | 5.84 | 9.93 | 21.63 | 40.45 | 4.93 | 62.60 | 101.64 | 250.12 | 619.45 | 9.67 | 135.41 | 412.93 | 869.96 | 673.23 |