Revision Response: The Cost of Representation by Subset Repairs

We thank the reviewers for giving us the opportunity to revise the paper, which helped us significantly improve it. We first describe a summary of changes in the revision under Meta-Review, and then list detailed responses to comments from each reviewer along with pointers to changes.

Updates made in response to specific comments are color-coded in the paper as follows:

- Changes in response to multiple reviewers as well as generic improvements in the presentation are in Purple.
- Changes for comments from Reviewer #1 in Blue.
- Changes for comments from Reviewer #2 in Magenta.
- Changes for comments from Reviewer #3 in Red.

Meta-Review

Comment: The reviewers were positively impressed by the topic and the paper, however they require some changes, well-detailed in their reviews. In particular, the reviewers suggest that the authors try to justify the deletion semantic in the context of bias mitigation and fix the issues raised about the experiments. Moreover, the paper supports bias only in a single attribute, and the reviewers would like to see an effort to address that, or at least an appropriate discussion of the topic. Please look carefully at all the reviewers' critiques, well explained in the reviews, and follow their suggestions.

Reply: Thank you for summarizing the comments. We first describe the main changes to the paper and later reply to each referee individually.

Our focus on deletions has two main reasons. First, we utilize and rely on the results of prior work that has theoretically analyzed subset repairs (S-repairs) or deleting tuples to fix FD violations in [23, 31, 41, 47], and the complexity of achieving an optimal S-repair based on the structure of the input FD set [41]. This focus allows us to infer conclusions about our model by extending these existing results. Second, S-repairs possess the advantage of keeping only original tuples in the repair and do not introduce new combinations of values within the same tuple, which may happen with other repair models. We have added this discussion to Section 3.1. Additionally, please refer to the newly added example in Section 8 for a case where information beyond the original data was introduced by the repair process.

We have significantly extended our experimental study on the existing datasets (ACS and COMPAS) and have added two new datasets (Flight [37] in Section 6 and Credit Card Transaction [10] in the full version [38]). The new dataset, Flight, showcases the repairing scenarios when FD violations and value distributions of the sensitive attribute are all inherent to the original dataset. Specifically, this dataset contains FD violations caused by conflicts across data sources from which the data was collected. These conflicts remove the need for artificial noise injections. We also articulated the process of generating the sample and injecting the noise into the ACS and COMPAS datasets, and further extended the study of the cost of representation by varying the original distribution of the sensitive attribute in the input relation and the level of injected noise in different groups.

Finally, extending our model to support multiple sensitive attributes requires overcoming several non-trivial challenges. We have added a discussion containing several new insights on the matter in the second paragraph in Section 8.

Reviewer #1

Comment: R1-D2. It is not clear whether deleting information is a realistic approach to preserve representation of some attribute. I understand that deletion might be acceptable to fix inconsistency in the data, but this is because we know that some tuples contain incorrect information. The issue is not that clear when it comes to preserving data distribution. I suggest that the authors make a stronger case about the applicability of this approach versus other semantics, e.g., using real-world scenarios and downstream tasks in which users are ok with deleting information to preserve representation.

Reply: We thank the reviewer for this observation. We have added a discussion to Section 3.1 ("Choice of repair model for cost of representation") in common color purple, where we explain our choice to focus on S-repairs in this work. In essence, subset repairing has been extensively studied as one of the two prevalent methods for data repair in the literature [23, 31, 41, 47]. Since this is the first work to consider representation as an integral part of the repair problem, we chose the deletion model where the complexity is well-understood and the results can be extended to our setting. Indeed, other repair models are important future work, as well as effects on downstream tasks under both deletion and updates. We have added a corresponding discussion to Section 8. For also simplified the discussion in Section 1 ("Contributions") saying that we focus on S-repairs in this paper.

Comment: R1-D3. Using the number of deleted tuples as the metric for cost might be too simplistic for many downstream tasks as some tuples might be contain more important information relevant to the task than others, e.g., if the downstream task is a classification, this metric is not useful as some tuples may be more important than others to learn an accurate classifier. I suggest that the authors provide a real-world example in which this cost is reasonable and reliable metric of information loss.

Reply: Since our focus is on the study of the cost of representation under the deletion semantics by S-repair, we employ the standard manner in which such repairs are measured. In general, the amount of intervention depends on the repair model. For example, for tuple deletions this is the number of deleted tuples, but for cell value updates, it is customary to use the number of updated cells [22, 23, 31, 41, 52]. A

natural extension of our model is to consider the (still task-independent) weighted cost of tuple deletion where different tuples have different weights or different costs, which can be helpful to the downstream tasks after repairing. Moreover, the effect of S-repair with representations in an end-to-end application including downstream tasks, and in general, the exploration of other metrics of cost of representations are important future directions. This is discussed now in Section 8.

Comment: R1-D4. The proposed method is limited to a single sensitive attribute. I understand that this is an initial work regarding deletion semantic. But the proposed solution does not seem to be usable as datasets often have multiple sensitive attributes. Also, these attributes are often correlated (e.g., race, age, income). Maintaining the representation of one attribute might help or hurt another sensitive attribute's representation. To make their case, I suggest that the authors provide real-world use cases in which the repaired datasets have only a single attribute. I also suggest that the authors provide and analyze some initial solution for the case of multiple attributes or at least provide more comprehensive discussion on the possible extensions of their approach to multiple attributes and their possible shortcomings.

Reply: Indeed, we focus on a single sensitive attribute, e.g., cost of representation for gender. We have added a discussion about multiple sensitive attributes in Section 8, while a longer discussion with initial observations on extending our model and algorithms appears in the full version of this paper [38] due to space limitations. In a nutshell, some of our definitions, theoretical results, and algorithms can generalize to multiple attributes, while other results are more challenging to generalize and may become intractable. For example, if the sensitive attributes are disjoint from the FDs and the goal is to preserve the joint distribution, the problem reduces to a single sensitive attribute assuming data complexity. On the other hand, while preserving marginal distributions on multiple sensitive attributes can be expressed as an ILP, this approach will likely not scale.

Comment: R1-D5. The empirical study uses injected FDs as opposed to the FDs that appear in real-world data. This approach may not necessarily reflect the pattern of noise in practice. I suggest that authors report some experiments using real-world datasets with FDs.

Reply: Thank you for the suggestion. We have added experiments with two new datasets: the Flight dataset [6, 37] in Section 6, and the Credit Card Transaction dataset [10] in the full version [38] due to space limitations. The Flight dataset was used in prior work on data fusion and data cleaning [6, 37, 52]. The noise in the data is caused by the conflicts across data sources. These conflicts result in FD violations without the need for artificial noise injections. The dataset description and FD set for the Flight dataset are in Section 6.1.1 and Table 1 respectively. The results are shown in Tables 3 and 6, and the deletion overhead and repair quality are respectively analyzed in Sections 6.2 and 6.3.

Regarding ACS and COMPAS in our submission, the FD sets shown in Table 1 are obtained by exploring the data description documents and verifying them on the datasets (see e.g., the dictionary for the ACS dataset [8]). They reflect the constraints implicitly expressed in the schema. We use noise injections to introduce FD violations into ACS and COMPAS datasets, which is a common approach to generate noisy inputs in the literature of data repair [15, 22, 52]. We have clarified this point in Section 6.1.4 ("Obtaining noisy input data").

We have further added new experiments, varying relative noise distributions for the ACS dataset in Section 6.2 and for the COMPAS dataset in the full version [38]. These experiments vary both the level of noise in each population segment categorized by the sensitive attribute, and vary the proportion of each segment, i.e., the same number of cells are changed in each sensitive group, or different numbers of cells are changed in each sensitive group. Through these experiments (detailed in Tables 2 and 5) we obtain two general observations: First, when there is relatively more noise injected into the under-represented group categorized by the sensitive attribute, the deletion overhead (number of tuples deleted compared to the optimal S-repair) of RS-repairs is high. Second, FDCleanser maintains high repair quality (over 94%), while other heuristics fell behind. Please see Tables 2 and 5.

The changes are in the common color purple in the experiments Section 6 (also discussed in R2-D3).

Comment: R1-D6. There paper has some typos, I suggest that the authors proof read the paper, e.g., page 2: "1.6 times of tuples than the" should be "1.6 times more tuples than the"

Reply: Thank you. We proofread the paper and fixed typos like the one you mentioned.

Reviewer #2

Comment: R2-D1. Why not including more datasets? There are more approaches under test than datasets for testing them (9 approaches to test over 2 datasets), this is clearly unbalanced. It is true that sometimes it is difficult to find datasets, but considering that the authors argue that their targeted problem is very relevant for avoiding Machine Learning (ML) biases, and the huge number of public datasets available for ML, they should be able to find more datasets for experimentation. Hence, I really, really believe that including more datasets is feasible and can deeply reinforce the evidence of the effectivity of this approach.

Reply: Thank you for the suggestion. We have added two new datasets to our experimental study: Flight [6, 37] with a chain FD set and real noise, and Credit Card Transaction [10] with a non-chain FD set and injected noise. The experiments on the Credit Card Transaction dataset are presented in the full version [38] due to limited space, and the observations are similar to ACS and COMPAS datasets discussed in Section 6.

The Flight dataset was used in prior work on data fusion and data cleaning [6, 37, 52]. The noise in the data is caused by the conflicts across data sources. These conflicts result in FD violations without the need for artificial noise injections. The dataset description and FD set

for the Flight dataset are in Section 6.1.1 and Table 1 respectively. The results are shown in Tables 3 and 6, and the deletion overhead and repair quality are respectively analyzed in Sections 6.2 and 6.3.

For conciseness and readability, and to make room for the new content, we omitted the discussion and the experimental result for LP + GreedyRounding from the main paper and moved them to the full version [38], decreasing the number of examined approaches in the paper to eight. Also please note that some approaches apply to chain FD sets only and some other approaches apply to non-chain FD sets only, as we now have clarified in Section 6.1.6. Please also see R2-D2 below.

Comment: R2-D2. Why including discarded approaches? During the discussion of your heuristics solutions in section 5.2 you seem to discard the algorithms LP+GreedyRounding and LP+ReprRounding. Then, why do you include them in the experiments? In the experiments I expect to see your defended approach (which I believe is algorithm FDCleaner, for non-chain FDs) versus the other approaches from the literature. Including discarded approaches distracts your reader. I totally agree on including evidences than FDCleaner is better than your other came out solutions, but maybe I would include them in the full version of the paper, rather than in this one.

Reply: Thank you for the suggestion. We have revised the paper to use a simpler manner for presenting our algorithms in multiple ways.

We have revised Section 5.1 by merging the descriptions for ILP, LP relaxations and moving the algorithm LP + GreedyRounding to the full version. The motivation for this change is that the performance of LP + GreedyRounding is close to that of LP + ReprRounding, and the latter has a representation-aware rounding step, which is a novel contribution.

We have moved all experimental results related to LP + GreedyRounding to the full version [38] simplifying the discussion in Section 6.

Additionally, we have now clarified our findings as they pertain to the best-performing approach in each setting (the "Summary of findings" paragraph at the beginning of Section 6). In particular, we have several polynomial-time heuristics for estimating the cost of representation for general (non-chain) FD sets as the problem is NP-hard —FDCleanser, LP + ReprRounding, and LP + GreedyRounding (moved to full version [38]). Among them, FDCleanser demonstrates the best trade-off between repair quality and runtime. For example, in "ACS, non-chain FD set, 80%-20% sensitive attribute distribution" (Table 5), FDCleanser consistently provided RS-repairs with more than 94% repair quality (compared to the optimal RS-repair) and in less than 10s (see the red line in Figure 4b). However, we also spotted other cases from the experiments where FDCleanser lagged behind (the cells with better quality than FDCleanser are boldfaced in Table 5). For example, FDCleanser provided RS-repair with slightly lower qualities compared to LP + ReprRounding in COMPAS and non-chain FD set ((d) in Table 5). However, as shown in Figure 4d, LP + ReprRounding (green line) is much slower than FDCleanser (red line) in this setting (e.g., for the 30K input size in Figure 4d, LP + ReprRounding took 18,485 seconds, whereas FDCleanser took 2,111 seconds).

Comment: R2-D3-a. Some interesting data I would have liked to see on the experiments: When adding "noise" to your data, how have you distributed the noise? Have you introduced more "noise" in the underrepresented tuples, or have you distributed the noise uniformly?

Reply: We have now expanded the experimental setup section (Section 6.1) and clarified the process of generating noisy input data in Section 6.1.4 ("Obtaining noisy input data") that is highlighted in the common color purple.

We use uniform random samples of different sizes from the new Flight dataset since the noise is real and does not have to be injected. The results are in Tables 3 and 6.

The ACS and COMPAS datasets, on the other hand, initially satisfy their FDs, and hence we inject random noise (FD violations) in a similar manner to prior work [15, 22, 52] to generate our noisy input data. We consider binary sensitive attributes in our experiments and vary two parameters: (1) *value distribution of the sensitive attribute* by stratified samples from the "majority" and the "minority" groups (50%-50%, 60%-40%, 80%-20%), and (2) *relative noise distribution of the majority and minority groups defined by the sensitive attribute* (20%-80%, 40%-60%, 50%-50%, 60%-40%, 80%-20%). This simulates the scenarios when one group in the sensitive attribute is more, equally, or less noisy than the other group. Please see Tables 2 and 5 with the new results (more settings are in the full version [38]).

In the results of ACS and COMPAS datasets shown in the original submission, we had 80%-20% data distribution of the majority and the minority groups, where we had changed the same number of cells in each sub-population defined by the sensitive attribute (i.e., the relative noise distribution was 20%-80%), simulating the real-world situations where the underrepresented group has lower data quality [3, 35]. However, now we have significantly expanded the experiments on noise injection for both deletion overhead and RS-repair-quality, so we do not make any assumption that the underrepresented tuples have more noise.

Comment: R2-D3-b. Some interesting data I would have liked to see on the experiments: How bad are current approaches that do not consider the representativity of the tuple? It is true that you have included a small Figure in the introduction to argue that, but I would have liked a better/formal discussion in the form of experiments. In this way, the reader can be convinced about the real necessity of your approach. However, under this setting, I am not sure to what extend could you use "randomly generated noise" but would need to find real data including real noise.

Reply: We have simplified Figure 8 that motivates the paper. As suggested by the reviewer, we extended the study with three questions: *"How bad S-repairs are for preserving the representation?"*, *"For inputs with different value distributions of the sensitive attribute and different noise distribution (by injections), how do our algorithms perform compared to the baselines that do not consider representation?"*, and *"How do the algorithms perform in a dataset with real noise?"*. To answer these questions, we have added Tables 2 to 4 to Section 6.2, and Table 6 to Section 6.3, with more settings in the full version [38].

To answer the first question, we added Table 4 for a specific scenario extending the example in the introduction, "ACS, 6K, 80%-20% value distribution of Nativity, 20%-80% relative noise distribution," to show the changes of the value distribution of the binary sensitive attribute in the S-repairs while increasing the overall level of noise. We observe that S-repair worsens when we increasingly inject more noise to the dataset. Further, the new Table 3 on Flight data with real noise shows that an optimal S-repair can significantly change the distribution of the groups defined by the sensitive attribute after repair.

To answer the second question, i.e., to see the effect of injected noise, we added an extended study of the cost of representation, covering more real-world scenarios in two types: (1) the two groups defined by the sensitive attribute are uniform/skewed (value distribution), and (2) the noise is distributed uniformly/non-uniformly among the groups categorized by the sensitive attribute (relative noise distribution). The results of deletion overhead (Tables 2 and 3) are shown and analyzed in Section 6.2. We observe that the deletion overhead is high when there is more noise distributed to the under-represented group defined by the sensitive attribute. This is as expected, since more noise injected into the under-represented group makes it harder to preserve tuples from this group. Therefore, we need to remove more tuples from the other group in the RS-repair in order to satisfy the RC.

To answer the third question, we added a new dataset, namely, Flight [6, 37]. It contains real noise caused by conflicts across data sources, which saves us from the process of noise injection. The setup information for Flight is in Section 6.1. In Sections 6.2 and 6.3, we have analyzed Flight's results for deletion overhead (Table 3) and for repair quality across different algorithms. (Table 6)

Comment: As a final comment, if the paper suffers from a lack of space, there are some observations from the experiments that, in my opinion, are not necessary. For instance, observing that, fixed an algorithm, the set of chain FDs takes lower time than the set of all FDs is quite expected since one set contains fewer constraints than the other.

Reply: Thank you for the suggestion! We have made our observations from the experiments more concise.

Comment: R2-D4-a. Some clarifications regarding the current results

- In section 6.4, last paragraph, it seems that your heuristics LP+GreedyRounding and LP+ReprRounding are worse than the GlobapILP algorithm with large data (that is, the heuristics seems to be defeated by the complete algorithm), but in section 6.4.1 you say that such heuristics are efficient and that GlobalIP does not scale. Please, clarify.

Reply: Thank you for the comment. This pattern shows up only in Figure 4d, while LP + GreedyRounding (now moved from the main paper to the full version [38]) and LP + ReprRounding outperform GlobalILP in terms of runtime in all other cases (as shown in Figure 4b). The behavior of these three approaches varies depending on the dataset. Even though LP relaxation and rounding are theoretically known as polynomial-time, in this specific case, the rounding steps took a large amount of time, simply because the conflict graph (constructed by FD violations) is very dense and a majority of tuples are involved in multiple FD violations. We have rephrased the explanation for this special case in Section 6.4 (last paragraph). Also, as input relation size increases, GlobalILP becomes impractical due to its exponential time complexity.

Comment: R2-D4-b

- In section 6.5 it seems that your algorithms deletes a lot of tuples. That is, it seems to remove more than 50% of them. How can this happen when you introduce so "little" noise (about 5% in some cases)?

Reply: We have now clarified the process of "Obtaining noisy input data" in Section 6.1.4. Specifically, 5% noise indicates that 5% of the total 'cells' (attribute value) belonging to the attributes that appear in the LHS or RHS of some FD have been changed to another value from the domain of the corresponding attribute. Since there are more than one attribute involved in the FD set, there can be more than 5% of the total tuples changed. The FD sets for different datasets are described in Table 1, and the number of attributes involved varies based on the dataset and the choice of chain vs. non-chain FD sets. Hence the number of cells and tuples that have been changed in different settings can be different for the same level of overall noise (5% or 10%).

For instance, in Table 2c, for ACS data with non-chain FD set, 80%-20% value distribution, and 20%-80% relative noise distribution, 52.83% tuples are deleted in the optimal RS-repair. However, from Table 1, note that the non-chain FD set for ACS data has 9 attributes. Hence 5% of the cells of 9 attributes are changed by noise injection, which may change at least one attribute value of many more tuples than 5%, which may in turn introduce conflicts with other tuples. As Table 2 shows, since the non-chain FD sets for ACS and COMPAS have more attributes than their chain counterparts, therefore, the percentage of tuples deleted by optimal RS-repair in tables (a) and (c) for chain FDs is much less than that in tables (b) and (d) for non-chain FDs.

Further note that, the number of tuples deleted in the optimal S-repair shown in Table 2 is a lower bound on the number of tuples deleted in an optimal RS-repair that also preserves the representation of the sensitive attribute. The above case where the optimal RS-repair removes 52.83% tuples, the optimal S-repair also removes 25.91% tuples. In this particular case, the minority group has 4-times more noise than the majority group (20%-80% relative noise distribution). Since many noisy tuples might have been deleted in the minority group, the optimal RS-repair had to also delete a large number of tuples from the majority group to preserve the original representation. These are the reasons why even 5% noise can lead to a large number of deletions. We have explained the results of Table 2 in Section 6.2 and added the explanation in Section 6.5.

Comment: R2-D5. Sometimes the paper excludes some information that makes it not self-contained, but provides some other content which, in my opinion, can be omitted. In my opinion, if you defend your algorithm FDCleaner, which is your best general algorithm, I would

totally omit section 5.2 from this version of the paper. Indeed, it seems to me that you can speak about FDCleaner without such heuristic algorithms. Furthermore, I would also omit the LHS marriage simplification, which I think is not used in the paper. In contrast, I would include the PostClean algorithm, a better explanation of the common LHS simplification, or include/enlarge the proof sketches of the Lemmas and Theorems. Finally, as a minor comment, in Page 5, "We prove (Lemma 10) that the optimal RS-repair [...]" should it be "an optimal"?

Reply: Thank you very much for these suggestions. As suggested by the reviewer, we moved LP + GreedyRounding and its associated results to the full version [38] and moved the content about LHS marriage, which is not used by our algorithms, to the full version [38] as well. As rightly pointed out by the reviewer, after the experiments, we observed that FDCleanser demonstrates the best trade-off between repair quality and runtime among all the heuristics and baselines. We have summarized this and other major findings in the beginning of Section 6 (paragraph titled "Summary of findings"). Please also see more details in R2-D2 for the comparison between FDCleanser and LP + ReprRounding. Finally, we have fixed the typo pointed out by the reviewer.

Reviewer #3

Comment: R3-W1. For the datasets used, you generate samples up to 30K and 1M rows, respectively. Is it a good size for experimentation? I.e., can we work with bigger inputs (the current ones sound small enough)?

Reply: Thank you for the comment. As suggested by the reviewer, we have extended our scalability experiment to input relations of size 1.5M. Specifically, for ACS, we have added LhsChain-DP's runtime for "chain FD set, 1.5M size, 5% noise" and FDCleanser's runtime for "non-chain FD set, 1.5M size, 5% noise" (see the last column in Table 7 and the associated discussion in Section 6.4). The other alternative approaches with reasonable RS-repair quality do not scale to large datasets. Nevertheless, both LhsChain-DP and FDCleanser are DP-based algorithms, thus they indeed require large space and do not scale well to larger datasets. Designing better space and time-efficient algorithms for RS-repair remains an interesting future work. We have added this point as the last sentence of Section 6.4.

Comment: R3-W2. The functional dependencies that are used in the experiments are from one attribute to another attribute. Can we consider pairs (or even bigger sets) of attributes? Any discussion on this?

Reply: We allow multiple attributes on the LHS of an FD, whereas we assume without loss of generality that the RHS has only one attribute (e.g., an FD AB $\rightarrow$ CD can be decomposed into AB $\rightarrow$ C and AB $\rightarrow$ D). For instance, as shown in Table 1 in Section 6, our experiments involve the functional dependency (COMPAS, non-chain FD set) `FirstName`, `LastName`, `DOB` $\rightarrow$ `Sex`, which is an FD that has multiple attributes on its LHS. To explain this better, we have clarified the definition of functional dependencies in Section 2, where we articulate that our algorithms support the FD that has more than one attribute on the LHS, and assumes one attribute on the RHS without loss of generality.

Comment: R3-W3. The solution considers deleting tuples. How possible is to consider additions of tuples to achieve the same goals?

Reply: Thank you for the comment. We have added a justification of our choice of repair model in Section 3.1 ("Choice of repair model for cost of representation") in common color purple. First, as the first exploration of data repair with the added aspect of representation, we begin with S-repairs (or deletion-based approaches), because this problem (without the added aspect of sensitive representation) has been well-studied by the literature [23, 31, 41, 47] and the complexity of computing an optimal S-repair is well-understood [41]. Second, repairing data through deletions is conservative as the first attempt, because it does not introduce any new information that is not included in the original dataset.

We note that tuple additions might introduce new information to the dataset which may incur new challenges. Specifically, in our model, we extend S-repair which relies on the anti-monotonic property of FDs. Namely, if a relation satisfies a set of FDs, then all its subsets do too. Once new tuples are introduced, this property cannot be used and our results will not directly extend. Therefore, the interesting future direction of exploring other repair models presents several novel challenges that need to be addressed. We have added a discussion about this direction in Section 8.

Comment: R3-W4. For a relevant recent survey on algorithmic fairness, you can cite: E. Pitoura et al. Fairness in Rankings and Recommendations: An Overview. VLDBJ 2022.

Reply: Thank you for the reference. We have referred to and cited this relevant survey paper [51] in Section 7.

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 [29, 48]. Therefore, the problem of automatic data repairing has been the focus of much prior work [1, 5, 13, 19, 22, 23, 33, 42, 52]. 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 [7, 11, 32], changing the database in different forms [5, 11, 13, 31, 42, 45], and even relaxing it in various manners [52, 53]. 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*) [31, 41, 42, 46, 47], 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 [41, 42] has characterized the tractable and intractable cases in this setting.

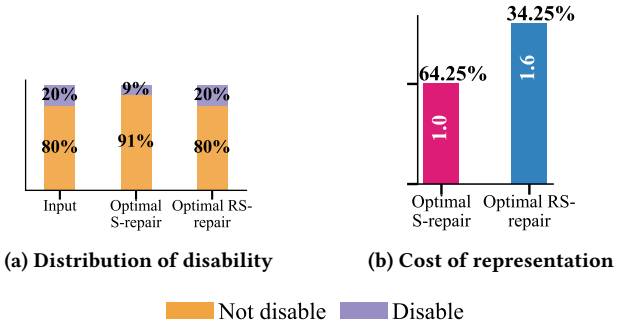


Figure 1: Disability status and cost of representation for ACS data. In (b), the % above the bars indicates remaining tuples, and the numbers on the bars show the deletion ratio relative to optimal S-repair (that does not satisfy representation).

Database repair may drastically sway the proportions of different populations in the data, specifically the proportions of various sensitive sub-populations. For instance, the repair may end up deleting more females than males, or more tuples from a certain age group, race, or disability status, as illustrated in the sequel in an example. This may happen simply by chance while selecting one of many feasible optimal repairs, or, due to varying levels of inconsistency in different sub-populations in the collected data, which may arise due to varying familiarity with the data collection technology, imputing varying amounts of missing data in different groups due to concerns for ageism and other biases, etc. If data is repaired agnostic to the representations, it can lead to biased decisions for downstream (e.g., ML prediction) tasks [24, 25, 56], 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 [36, 57]. 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 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

* Both authors contributed equally.

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 8a, 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 [42, 47], no people with disability would stay in the repaired data. Consequently, both S-repairs may introduce biases against people with disability in downstream tasks that use the repaired datasets.*

The above example shows that representation-agnostic S-repair methods can badly affect representations of groups. Our aim is to answer the following question: *Can we both obtain an S-repair and enforce a desired representation?* Figure 8 shows that "Optimal S-repair" retains 64.25% of the original tuples, but does not satisfy the representation, while it is possible to obtain an RS-repair that retains 34.25% of the tuples. Although retaining only 34.25% of the original tuples after the optimal RS-repair looks pessimistic, this cost is necessary in this example. In other words, for this dataset and specific noise, if we insist on preserving the 20%/80% ratio of representation among disabled and non-disabled people respectively, 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[1]. Since computing the optimal RS-repair is not always possible, in this paper, we study multiple other approaches for obtaining the RS-repairs.

Contributions. In this paper, we focus on understanding the cost of representations for S-repair[2], where the number of deleted tuples is used to measure the repair quality. Algorithms and complexity of S-repair, which is NP-hard in general, has been studied by multiple prior works [23, 41, 45, 47]. 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. [41, 42], who established a complete classification of complexity (dichotomy) based on the structural properties of the input FD set. We show that finding an RS-repair can be hard even if the FDs are such that an optimal S-repair can be computed in polynomial time.

Next we investigate the complexity of computing an optimal RS-repair for special cases of FDs and RC. We present a polynomial-time dynamic-programming-based algorithm for a well studied class of FDs, namely the *LHS-chains* [40, 43], 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 using three real-world datasets. We demonstrate the effect of representation-agnostic S-repairs on the representations of the sensitive attribute and show that optimal RS-repairs delete 1× to 2× tuples compared to optimal S-repairs in Section 6.2, depending on how the noises are distributed. We conduct a thorough comparison between our algorithms, existing baselines, and a version of these baselines with post-hoc processing that further deletes tuples until the RC is satisfied, and analyze the quality and runtime of different approaches. We show that representation-aware subset-repair algorithms can find superior RS-repairs in terms of the number of deletions. In summary, our contributions are as follows. *(1)* We introduce the problem of finding an optimal RS-repair as a way of measuring the cost of representation. *(2)* We present complexity analysis of the problem. *(3)* We devise algorithms, including polynomial-time and ILP algorithms with optimality guarantees, and heuristics. *and (4)* We conduct a thorough experimental study of our solutions and show their effectiveness compared to the baselines.

Due to space limitation, all proofs appear in the full version [38].

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

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

[1] More settings varying noise and representations are given in the experiments (Section 6) and full version [38].

[2] We note that "update repair" methods, e.g., Holoclean [52] and Nadeef [15], have their own limitations or challenges when considering the cost of representations as discussed in Section 8.

multiple FDs. Note that the LHS X can contain multiple attributes. We also assume that the FDs $X \rightarrow Y$ are non-trivial, i.e., $Y \notin X$.

Two types of database repairs have been mainly studied. A *subset repair (S-repair)* [23, 31, 41, 47] changes a noisy relation by removing tuples, while an *update repair* [22, 42, 52] changes values of cells. Each of the two has pros and cons. While the update repair retains the size of the dataset, it may generate invalid tuples (as discussed in Section 1). An S-repair uses original tuples, but at the cost of losing others. The complexity of computing an optimal S-repair is well understood [41], whereas the picture for update repairs is still quite partial [31, 41]. Hence, we focus on S-repairs and leave the update repairs for future work. Formally, given R and Δ, an S-repair is a subset of tuples[3] $R' \subseteq R$ such that $R' \models \Delta$. R' is called an *optimal subset repair* (or *optimal S-repair*) if for all S-repairs R'' of R given Δ, $|R'| \geq |R''|$ (there is a weighted version when there is a weight $w(t)$ associated with each input tuple t).

Computing optimal S-repairs. Livshits et al. [41] proposed an exact algorithmic characterization (dichotomy) for computing an optimal S-repair. Moreover, it showed that when a specific procedure is not able to return an answer, the problem is NP-hard for the input Δ in data complexity [59]. We briefly discuss these concepts and algorithms as we will use them in the sequel.

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

Using the above concepts, three simplification methods of Δ 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) ρ is a finite set of lower-bound constraints. A relation R satisfies $\%a \geq p$ if at least $p \cdot |R|$ of the tuples $t \in R$ satisfy $t[A_s] = a$. A relation R satisfies an RC ρ, denoted $R \models \rho$, if R satisfies every lower-bound constraint in ρ.*

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

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

For simplicity, our implementation restricts attention so that each proportion (p_ℓ) in the input is a rational number, represented by an integer numerator and an integer denominator. Moreover, if one does not specify a lower-bound constraint for some a_o, then we treat it as 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 ρ, but both the subset $R_1 = \{(1, a, 3), (2, b, 5), (3, c, 9)\}$ and the subset $R_2 = \{(2, b, 5), (3, c, 9), (4, d, 3)\}$ satisfy it.*

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

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

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

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

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

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

Choice of repair model for cost of representation. In this initial study of repairs with representation, we consider S-repair (deletion) as the repair model. Multiple prior works have theoretically analyzed S-repairs (without the cost of representation) [23, 31, 41, 47], and the complexity of achieving an optimal S-repair based on the structure of the input FD set is well understood [41]. We focus on the framework and theoretical analysis for this simpler variant of the repair model with the representation criteria. Additionally,

[3] We note that in prior work [41, 46], 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.

[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).

S-repairs keep tuples from the original dataset and do not introduce new combinations of values within the same tuple while repairing the data. We use the number of deletions to define the optimal RS-repair as an extension for defining S-repair models from prior work. Extensions to other repair models and other cost functions (e.g., based on the effects on downstream tasks) are important and challenging future work (see Section 8).

3.1.1 NP-Hardness of Computing Optimal RS-Repairs. In this section, we consider the relation R and the RC ρ as inputs, while the schema $\mathcal{S}$ and the FD set Δ are fixed. We first note that since the problem of finding an optimal RS-repair is a generalization of the problem of finding an optimal S-repair, as expected, in all cases where finding an optimal S-repair is NP-hard, finding an optimal RS-repair is also NP-hard. Theorem 1 shows that computing an optimal RS-repair is NP-hard even for a single FD. We prove this theorem by a reduction from 3-SAT (proof in [38]).

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

It is important to note that if Δ contains a single FD, computing an optimal S-repair can be done in polynomial time [41]. 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. [41], computing an optimal S-repair is poly-time solvable if Δ can be reduced to $\emptyset$ by repeated applications of three simplification processes: (i) consensus FDs (remove FDs of the form $\emptyset \rightarrow Y$), (ii) common LHS (remove attribute A from Δ, such that A belongs to the LHS of all FDs in Δ), and (iii) LHS marriage, which is slightly more complex. We will show that reduction to $\emptyset$ only by the first two simplification processes entails a polynomial time algorithm for computing optimal RS-repairs when the sensitive attribute A_s has a fixed number of distinct values (e.g., for common sensitive attributes gender, race, disability status, etc.). Before we formally state the theorem, we take a closer look at the class of FD sets that reduces to $\emptyset$ by the first two simplifications.

Definition 3. *An FD set Δ is an* LHS-chain *[39, 41] 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 [39, 41]. [39] showed that the class of LHS-chains consists of precisely the FD sets where the S-repairs can be counted in polynomial time (assuming $\mathsf{P} \neq \#\mathsf{P}$). [41] observed that FD sets that form an LHS-chain can be simplified to the empty set by repeatedly applying simplifications on only the common LHS and the consensus FD. We show in the following proposition that the converse also holds:

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

The following theorem states our main algorithmic result.

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

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

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

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

Overview of the PostClean algorithm. The PostClean algorithm intuitively works as follows (pseudo-code and analysis are in [38]). Recall from Section 3.1 that $\rho(a_\ell)$ denotes the lower bound on the fraction of the value a_ℓ in the tuples retained by the RS-repair. Further note that sum of $\rho(a_\ell)$ may be smaller than 1, i.e., the RC ρ may only specify the desired lower bounds for a subset of the sensitive values, and the rest can have arbitrary proportions as long as a minimum set of tuples is removed to obtain the optimal RS-repair. Moreover, the fractions are computed w.r.t. the final repair size $|R'|$ and not w.r.t. the input relation size $|R|$. PostClean iterates over all possible sizes T of R' from $|R|$ to 1. For each T, it checks if the lower bound on the number of tuples with a_ℓ, i.e., $\tau_\ell = \lceil T \cdot \rho(a_\ell) \rceil$, is greater than the number of tuples with value a_ℓ in the original relation R. If yes, then no repair R' of size T can satisfy ρ, and it goes to the next value of T (or $T \leftarrow T - 1$). Otherwise, if there are sufficient tuples for all sensitive values a_ℓ, and if the sum of the lower bounds on numbers, formally $T_0 = \sum \tau_\ell$ is $\leq T$, then we have a feasible T. Finally, the algorithm arbitrarily fills R' with more tuples from R if $T_0 < T$ and returns the final R'. Note that if all values of T between $|R|$ and 1 are invalid, then an $\emptyset$ is returned because it is the only subset of R that (trivially) satisfies the ρ. The following states the optimality and runtime of PostClean.

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

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

Example 6. *Consider a relation R with $\mathcal{S} = (A, B, \mathsf{sex})$, a set Δ of FDs $\{A \rightarrow B\}$, and an exact RC $\rho = \{\%\mathtt{male} = \frac{1}{2}, \%\mathtt{female} = \frac{1}{2}\}$. Let $R = \{(1, a, \mathtt{male}), (1, b, \mathtt{female}), (2, c, \mathtt{male}), (2, d, \mathtt{female})\}$.*

An optimal S-repair of R w.r.t. Δ is $R' = \{(1, a, \texttt{male}), (2, c, \texttt{male})\}$. However, PostClean$(R', \rho)$ returns $\emptyset$ since R' does not have any female tuples. Conversely, $\{(1, a, \texttt{male}), (2, d, \texttt{female})\}$ is an optimal RS-repair, which satisfies both Δ and ρ.

4 DYNAMIC PROGRAMMING FOR LHS-CHAINS

We now prove Theorem 2 by presenting a DP-based 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, Δ can be reduced to $\emptyset$ by repeated application of only consensus FD simplification and common LHS simplification.

Overview of LhsChain-DP. For a relation R and a set Δ of FDs, let $\mathcal{A}_{R,\Delta} = \{R' \subseteq R \mid R' \models \Delta\}$ be the set of all S-repairs of R for Δ. Intuitively, if we could enumerate all S-repairs R' from $\mathcal{A}_{R,\Delta}$, we could compute $R'' = \text{PostClean}(R', \rho)$ (Section 3.3) for each of them and return the R'' with the maximum number of tuples. Since PostClean optimally returns a maximum subset satisfying ρ for every R', and since any RS-repair w.r.t. Δ and ρ must be an S-repair w.r.t. Δ, such an R'' is guaranteed to be an optimal RS-repair. However, even when the domain size of the sensitive attribute A_s is fixed, the size of $\mathcal{A}_{R,\Delta}$ can be exponential in $|R|$. Therefore, it is expensive to enumerate the set of all S-repairs. Hence, we find a *candidate set* $C_{R,\Delta} \subseteq \mathcal{A}_{R,\Delta}$ of S-repairs that is sufficient to inspect. Then, we apply PostClean to each element of $C_{R,\Delta}$, and return the final solution having the maximum size. We formally define candidate set $C_{R,\Delta}$ in Section 4.1, along with associated definitions. The basic idea is that there are no two S-repairs in $C_{R,\Delta}$ where one is "clearly better" than the other or that the two "are equivalent to each other". Further, for any S-repair that is not in the candidate set i.e., $R'' \in \mathcal{A}_{R,\Delta} \setminus C_{R,\Delta}$, there is an S-repair $R' \in C_{R,\Delta}$ that is "clearly better" or "equivalent to" R''. We prove (in Lemma 9) that an optimal RS-repair can be obtained by applying PostClean to each S-repair in $C_{R,\Delta}$ and returning the one with maximum size. Moreover, the size of the candidate set is $O(|R|^k)$, when the domain size $|Dom(A_s)| = k$ is fixed (proofs in [38]).

Algorithm 1 LhsChain-DP(R, Δ, ρ)

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

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

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

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

4.1 Candidate Set for Optimal RS-Repairs

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

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

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

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

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

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

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

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

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

ReprInsert. A subroutine ReprInsert(C, R_0) (pseudocode in [38]) 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 Δ is an LHS-chain. Since an LHS-chain Δ can be reduced to $\emptyset$ by repeated applications of consensus FD and common LHS simplifications, Reduce calls ConsensusReduction (Section 4.2.1) and CommonLHSReduction (Section 4.2.2) until Δ becomes empty. When Δ is empty, it returns $\{R\}$ as the singleton candidate set since R itself is an S-repair and representatively dominates all other S-repairs. The following lemma states the correctness of the Reduce procedure.

Algorithm 2 Reduce(R, Δ)

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

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

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

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

Algorithm 3 ConsensusReduction(R, Δ, f)

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

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

4.2.2 Reduction for Common LHS. Consider a common LHS attribute X that appears on the LHS of all FDs in Δ. Suppose that $Dom(X) = \{x_1, x_2, \ldots, x_n\}$, $R_{x_\ell} = \sigma_{X=x_\ell} R$ denotes the subsets of R that have value $X = x_\ell$, and $R_{x_1,\ldots,x_\ell} = \sigma_{X=x_1 \vee \cdots \vee X=x_\ell} R$ as an extension. Also suppose Δ_{-X} denotes that the common LHS attribute X is removed from all FDs in Δ. If we were to consider S-repairs, we could optimally repair each R_{x_ℓ} w.r.t. Δ_{-X} independently (and recursively), and then take the union of their optimal S-repairs to obtain an optimal S-repair for R w.r.t. Δ (as done in [41]).

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 ρ, we do not know what the value distribution of A_s should be in each disjoint piece (e.g., some R_{x_ℓ}) of the final repair before we get one. Therefore, in each step of the recursion (either CommonLHSReduction here or ConsensusReduction above), the candidate set preserves all the possible distributions of A_s from the S-repairs that could provide the final optimal solution.

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 ℓ, the algorithm obtains the candidate set $C_{R_{x_1,\ldots,x_\ell},\Delta}$ by combining S-repairs for $R_{x_1}, \cdots, R_{x_\ell}$. Note that the union of S-repairs for $R_{x_1}, \cdots, R_{x_\ell}$ is an S-repair for $R_{x_1,\ldots,x_\ell}$ and consequently R w.r.t. Δ , but we have to ensure that the properties of a candidate set 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, Δ, X)

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

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

5 ALGORITHMS FOR THE GENERAL CASE

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

5.1 LP-based Algorithms

ILP-based Optimial Algorithm and LP-based Heuristic. 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.

$$\begin{aligned} &\textbf{Maximize} \sum_{i \in [1,|R|]} x_i \quad \textbf{subject to:} && (1)\\ &x_i + x_j \leq 1 && \text{for all conflicting } t_i \text{ and } t_j \\ \sum_{i:t_i[A_s]=a_\ell} x_i \;&\geq\; p_\ell \times \sum_i x_i && \text{for all } a_\ell \in Dom(A_s) \\ &x_i \in \{0, 1\} && \text{for all } i \in [1, |R|] \end{aligned}$$

The objective maximizes the number of tuples retained. The first constraint ensures that the solution does not violate Δ. The

following set of constraints correspond to the RC ρ, by ensuring each lower-bound constraint is satisfied, i.e. $\%a_\ell \geq p_\ell$, where $p_\ell = \rho(a_\ell)$, is satisfied for every $a_\ell \in Dom(A_s)$.

The ILP in Equation (1) can be relaxed to an LP by replacing the integrality constraints $x_i \in \{0, 1\}$ with $x_i \in [0, 1]$, for every $i \in [1, |R|]$. We propose rounding procedures to derive an integral solution from fractional x_is and refer to the heuristic as LP + ReprRounding (pseudocode and running time analysis in [38]).

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

5.2 FDCleanser: A DP-based Algorithm

The combinatorial DP-based FDCleanser algorithm is motivated by the ideas behind the CommonLHSReduction and ConsensusReduction procedures in Section 4. An FD set Δ with only one FD can be reduced to the empty set using LhsChain-DP by first applying CommonLHSReduction and then ConsensusReduction, and hence is poly-time solvable by Theorem 2. FDCleanser therefore calls LhsChain-DP with one FD at a time from Δ until all FDs are taken care of. Further, it prioritizes the FD which has a most frequent LHS column X (among all the columns) in its LHS. This greedy approach may not be optimal as demonstrated empirically in Section 6. As highlighted, Since Δ is fixed and each call to LhsChain-DP runs in polynomial time for fixed $Dom(A_s)$, FDCleanser terminates in polynomial time for any (R, Δ, ρ) where $Dom(A_s)$ is fixed. FDCleanser provides a practical and scalable heuristic approach for handling large instances of the problem of computing RS-repairs, by leveraging the efficient subroutine, LhsChain-DP. Its effectiveness and efficiency will be empirically evaluated in Section 6.

Algorithm 5 FDCleanser(R, Δ, ρ)

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

In our implementation, the heuristics described in Section 5.2 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 to preserve representation in subset repairs, and the quality and performance of our algorithms. In particular, we study the following questions:

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

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

6.1 Setup

We implement 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 [26]. The code is publicly available[5].

6.1.1 Datasets. We use samples of different sizes from three real datasets (ACS, COMPAS, and Flight) that are commonly used for the study of fairness or data repair. For ACS and COMPAS, the noise is injected (Section 6.1.4), whereas for the Flight dataset, the noise is real and inherent to the dataset. Additional experiments with a fourth dataset on Credit Card Transactions [10] appear in the full version [38] due to space limitations.

- ACS-PUMS (in short ACS) [16]: The American Community Survey Public Use Microdata Sample dataset. We use 9 attributes and samples from 2K to 1.5M in our experiments (described below).
- COMPAS [34]: The ProPublica COMPAS recidivism dataset. We use 10 attributes and samples from 4K to 30K in our experiments.
- Flight [6, 37][6]: The Flight data contains information on scheduled and actual departure time and arrival time from different sources, and has been used in prior work of data fusion [37] and data cleaning [6, 52]. We use 6 attributes and samples from 2K to 8K.

6.1.2 Functional dependencies. We consider two types of FD sets in our experiments: (1) chain FD sets (i.e., LHS-chains, Definition 3,

[5] https://github.com/louisja1/RS-repair.
[6] We download and use the version of data from Boeckling and Bronselaer [6].

Section 4), and (2) non-chain FD sets (Section 5). For ACS and COMPAS, FDs are inferred from the data description documents and verified in the original clean datasets [8]. The FD sets used in all three datasets are shown in Table 1[7]. For example, the FD for the ACS dataset ST → DIV implies that each state (ST) has a unique division (DIV). For the Flight dataset, FDs are from prior work [6]. For ACS and COMPAS, the complete set of FDs is non-chain, and we also use a subset of the FDs that forms a chain for these two datasets. As for Flight, the complete set of FDs forms a chain, so only the chain FD set is considered.

Table 1: FD sets used in experiments

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

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

6.1.4 Obtaining noisy input data. For the Flight dataset, the noise is real, inherent, and caused by conflicts coming from data sources, so these conflicts naturally result in FD violations. Hence, we only generate uniform random samples from the Flight dataset of size 2K, 4K, 6K, and 8K tuples as the noisy input data. These samples have slightly different value distributions of Source, ranging from {`%wunderground` = 58%, `%flightview` = 42%} to {`%wunderground` = 54%, `%flightview` = 46%}.

The ACS and COMPAS datasets, on the other hand, initially satisfy their FDs, and hence we inject random noise (FD violations) in a similar manner to prior work [15, 22, 52] to generate our noisy input data. We vary two parameters: (1) *value distribution of the sensitive attribute*, and (2) *relative noise distribution of the two groups defined by the sensitive attribute*, as we describe next.

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

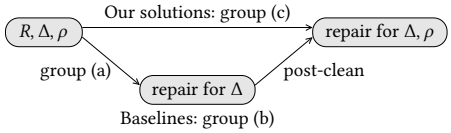


Figure 2: Our solutions vs. baselines

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

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

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

- ILP-Baseline [44] (for both chain and non-chain FD sets): Formulates an ILP (with only the first constraint for FD in Equation (1)) and computes the *optimal* S-repair satisfying a given FD set.
- VC-approx-Baseline [2] (for non-chain FD sets only): An approximation algorithm that translates the problem of computing an optimal S-repair to finding a minimum vertex cover of the conflict graph by its known 2-approximation algorithm.
- DP-Baseline [42] (for chain FD sets only): A dynamic programming (DP) algorithm that computes the *optimal* S-repair in polynomial time when the FD set forms an LHS-chain.
- MuSe-Baseline [23] (for both chain and non-chain FD sets): An S-repair framework with deletion rules. We use step semantics.

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

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

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

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

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

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

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

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

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

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

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

For chain FD sets, we examine the polynomial-time optimal algorithm LhsChain-DP (Section 4). For non-chain FD sets LhsChain-DP is not applicable, and we examine the end-to-end heuristic algorithms LP + ReprRounding and FDCleanser (Section 5). Recall that when applied to chain FD sets, FDCleanser reduces to LhsChain-DP. We repeat each experiment twice for each data point and take the average.

6.2 Deletion Overhead of RS-repairs

In this section, we investigate how many additional deletions are required to satisfy the representation of the sensitive attribute by defining *deletion overhead* as follows:

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

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

First, we examine the deletion overhead of RS-repairs in various scenarios for ACS in Table 2 (results for COMPAS appear in full version [38] due to space limitations). Table 2 also shows the deletion overhead and the percentage of tuples deleted by the optimal S-repair and optimal RS-repair for ACS for both chain and non-chain FD sets (5% overall noise and 10K tuples). We vary the data distribution of the majority and minority groups as 80%-20% and 50%-50%, and for each vary their relative noise distribution 20%-80%, 40%-60%, 50%-50%, 60%-40%, 80%-20%, investigating the cases when the minority group has more, equal, and less noise than the majority group. We observe that (1) When noise is uniform (50%-50%), irrespective of the data distribution in Tables 2a to 2d, RS-repair does not delete many additional tuples, hence the deletion overhead is close to 1. However, as the noise distribution becomes unbalanced in all tables, the deletion overhead increases since more tuples are deleted to satisfy the representation. (2) In Tables 2a and 2c, when the data distribution of the majority and minority groups is 80%-20%, the overhead for relative noise distribution 20%-80% is larger than the overhead for the relative noise distribution 80%-20% because. Intuitively, the fraction of noisy tuples from the minority group is much larger requiring more deletions from the minority group compared to the majority group in the optimal S-repair. Hence, many tuples from the majority groups are required to be removed to get the data distribution back to 80%-20% in the optimal RS-repair. (3) Although both chain and non-chain FD sets show similar trends on the deletion overhead, the deletion percentages of tuples (S-rep.% and RS-rep%) are lower for the chain FD set than those for the non-chain FD. This is because with the same noise level but a lower number of attributes involved in the chain FD set (Table 1), less noise is injected.

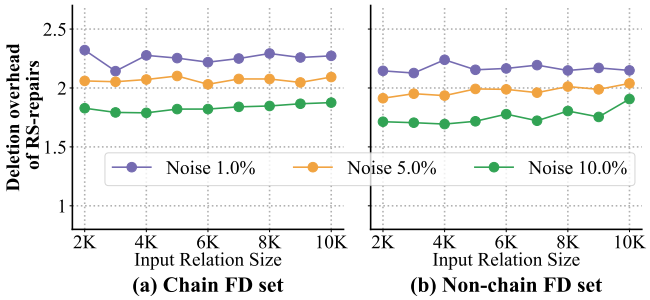


Figure 3: Deletion overhead varying overall noise and input size (ACS, 80%-20% value distr., 20%-80% relative noise distr.)

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

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

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

Table 3 considers the Flight data with real noise. It presents the deletion overhead, deletion percentage of the optimal S-repair and optimal RS-repair, and the value distribution of the majority and the minority groups of the input and after S-repair, RS-repair. (1) As we vary the input size (2K to 8K) and show that the deletion overhead is consistently slightly larger than 1, many tuples are deleted in both S-repair and RS-repair. (2) Interestingly, although the deletion overhead is close to 1, the optimal S-repair returned by the ILP does not preserve the original value distribution and

deletes more from the minority group. On the other hand, when the representation constraint is specified in the ILP, the optimal RS-repair preserves the value distribution after repair.

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

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

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

6.3 RS-repair Quality of Various Approaches

Although GlobalILP provides the optimal RS-repair, solving ILP is NP-hard and not scalable (e.g., GlobalILP took 18 hours to repair ACS data for the non-chain FD set, 10% overall noise, with only 10K tuples). Hence, we report the *RS-repair quality* (*repair quality* in short) defined below, i.e., how our proposed algorithms perform compared to the optimal RS-repair returned by GlobalILP, on relatively smaller datasets where running GlobalILP was possible.

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

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

In Table 5, we examine the repair quality varying value distributions of the sensitive attribute and FD types for ACS and COMPAS data. Since there are many potential scenarios, we present a set of results (more results are in the full version [38]) for overall 10% noise level, value distributions of the majority and minority groups X%-Y% ("80%-20%", "60%-40%", and "50%-50%") with relative noise distributions Y%-X% (i.e., the same number of cells are changed in both majority and minority groups). The optimal GlobalILP has 100% repair quality in all scenarios. The results by FDCleanser proposed in Section 5.2 for general FD sets, which is identical to the optimal LhsChain-DP in Section 4 are **boldfaced** in all tables. The repair quality by other algorithms that are better than FDCleanser (non-chain) or LhsChain-DP (chain) is also **boldfaced**.

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

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

6.4 Running Time Analysis

Comparison of runtime of different RS-repair methods: First, we evaluate the runtime performance of the RS-repair algorithms from groups (b) and (c) in Figure 2. Figure 4 shows the runtime results of the ACS and COMPAS for 80%-20% value distribution of the sensitive attribute and 10% overall noise level (runtime for more settings and Flight are in the full version [38]). If no data points are shown, execution took longer than 12 hours. (1) The optimal GlobalILP for RS-repair has a high runtime in all settings as ILP is not scalable. (2) For chain FD sets in Figures 4a and 4c the optimal LhsChain-DP (= FDCleanser) is much faster than GlobalILP, e.g., GlobalILP takes 1, 046s for size 10K ACS data, while LhsChain-DP takes only 12s, (3) For non-chain FD sets in Figures 4b and 4d, FDCleanser has a good scalability. While VC-approx-Baseline+PostClean has a slightly better runtime than FDCleanser in Figure 4c, Table 5 shows its poor quality. Additionally, for COMPAS and non-chain FD set () in size 30K due to a lagged rounding step.

Scalability for bigger data: Since ILP is known not to be scalable, in Table 7 we evaluate the scalability of the other methods over bigger samples up to 1.5M tuples of the ACS dataset. This table shows that FDCleanser for non-chain FD sets, which is the same as the optimal LhsChain-DP for chain FD sets can repair bigger datasets, while the other methods face out-of-memory issues. While DP-Baseline+PostClean has much better scalability for chain FD sets, as discussed in Section 6.3, it suffers from poor quality. For the chain FD set, LhsChain-DP provides the optimal RS-repair, in 8 hours for 1M tuples and in 48 hours for 1.5M tuples. For the non-chain FD set, FDCleanser takes 5.9 hours for 1M tuples, and 34 hours for 1.5M tuples. Although it takes a long time to repair the data, it returns results, which may be permissible for offline data repair tasks. In contrast, VC-approx-Baseline + PostClean and LP + ReprRounding encounter issues with memory usage over 100K tuples, which is as expected given that the number of constraints can be as large as $|R|^2$. Nevertheless, both LhsChain-DP and FDCleanser are DP-based algorithms requiring large space

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

(a) ACS: chain FD set

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

(b) ACS: non-chain FD set

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

(c) COMPAS: chain FD set

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

(d) COMPAS: non-chain FD set

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

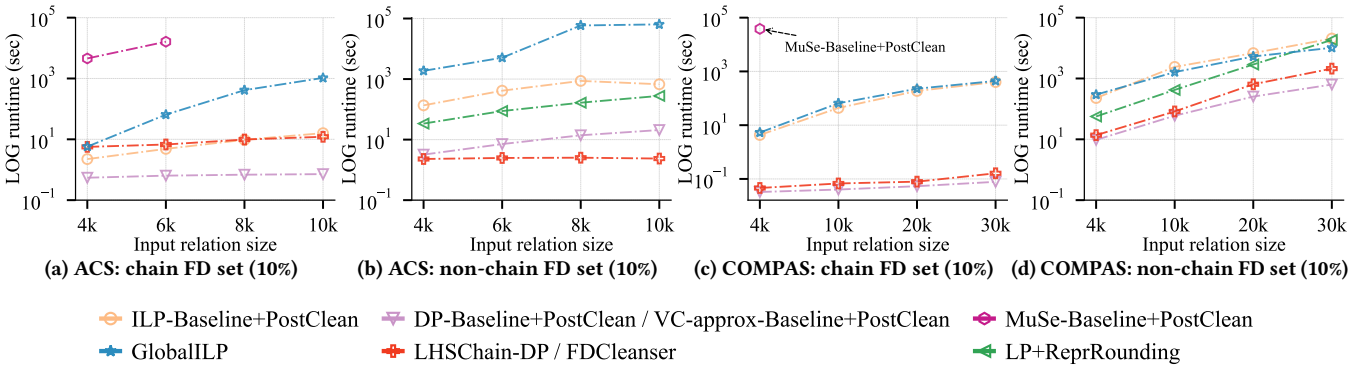


Figure 4: Runtimes for ACS and COMPAS data (10% noise level, 80%-20% value distribution) with chain and non-chain FD sets

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

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

and do not scale well to larger datasets, and designing better space and time-efficient algorithms for RS-repair remains an interesting future work.

6.5 Varying Representation Constraints

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

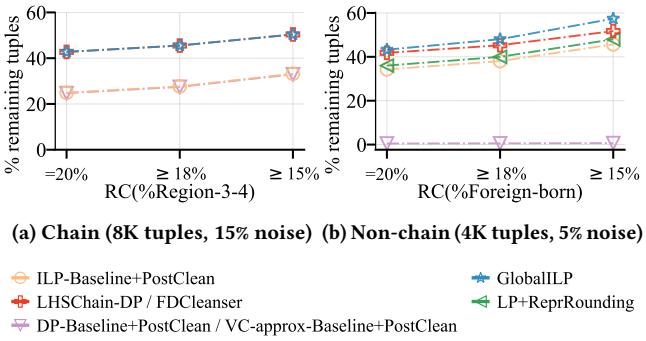
Figure 5 shows that for both chain and non-chain FD sets, the percentage of remaining tuples increases as the constraint is relaxed. The percentage of tuples deleted for non-chain FDs is high even for 5% noise, since the FDs have 9 attributes (Table 1), possibly changing a large number of tuples in noise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

7 RELATED WORK

Data repairing and constraints. Past work on data repairing [1, 5, 13, 19, 52] aimed at minimizing the number of changes in the

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

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



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

Figure 5: Percentage of remaining tuples varying the RC (ACS, 80%-20% value distr. and 20%-80% relative noise distr.)

database in terms of tuple deletions [1, 11, 23, 42, 45, 47] and value updates [5, 13, 15, 22, 31, 52]. The complexity of the former is better understood [42]. We intend to study extensions of our approach to the value-change model as well in future work. The literature considered a wide variety of constraints, such as FDs and versions thereof [7, 32], denial constraints [11], tuple-generating dependencies [18, 20], and equality-generating dependencies [4]. Different from these, our representation constraints considers the statistical proportions of an attribute *over the entire database*.

Representation constraints. Various types of constraints involving probability distributions on the data have been proposed in the literature. A predominant part of these focuses on *algorithmic fairness* [9, 21, 51, 54, 55]: given a classification algorithm, and sensitive attributes, determine whether the classification is *fair* according to some definition of *fairness* [14, 27]. This vein of literature focuses 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 [17, 30] and true positive rate balance [12, 58]. Capturing some fairness definitions using complex RCs is an intriguing subject for future work

8 DISCUSSION 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. This paper conducts an initial study of data repair with representations using S-repair, and there are many interesting future directions.

First, one can study the complexity and algorithms for generalization of representations to *multiple sensitive attributes with arbitrary overlap with the FDs*. While the problem will remain NP-hard and ILP can give a solution for smaller data and some settings, a closer study of the complexity, algorithms (e.g., the PostClean process), and impact in practice will be challenging. We have a detailed discussion on multiple sensitive attributes in the full version [38].

Second, it will be interesting to study the cost of representations for other repair models, and in particular, for update repairs. Update repair cleans the data by updating values of some cells (attributes of tuples), which preserves the original data size unlike S-repair. However, update repair approaches also have their own challenges with or without representations. In Example 1 discussed in the introduction, a popular update repair method Holoclean [52] does not make any changes in the data, and therefore does not eliminate any violations, since it treats the FDs as soft constraints while preserving the data distribution. On the other hand, another update repair method Nadeef [15] provides a repair satisfying the FDs, and since the sensitive attribute Disability does not participate in the FDs, it preserves its representations. However, in our experiment, Nadeef changed a `(Asian, foreign-born)` person to `(White, native)` while keeping the other attribute values the same `(female, born in Philippines, lives in CA, DISABLE, ···)`. This is not faithful to the reality (People born in Philippines should be "`foreign born`" not "`native`" and are likely to be Asian (not White)). Data repair under updates might be inherently connected to preserving distributions of multiple attributes discussed above.

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

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] Solon Barocas and Andrew D Selbst. 2016. Big data's disparate impact. *Calif. L. Rev.* 104 (2016), 671.

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

[5] 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.

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

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

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

[9] 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.

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

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

[12] 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

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

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

[15] 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.

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

[17] 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

[18] 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

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

[20] 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.

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

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

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

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

[25] 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

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

[27] 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

[28] 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.

[29] 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

[30] 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

[31] 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

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

[33] 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

[34] 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.

[35] Jonas Lerman. 2013. Big data and its exclusions. *Stan. L. Rev. Online* 66 (2013), 55.

[36] 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

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

[38] 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.

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

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

[41] 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.

[42] 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.

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

[44] 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.

[45] 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.

[46] 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

[47] 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

[48] 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

[49] 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.

[50] 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.

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

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

[53] 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.

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

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

[56] 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

[57] 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

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

[59] 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 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 Proposition 5 (Section 3.3).

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

A.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* $\Delta = \{A \to B\}$.

Proof. Recall the decision problem of computing an optimal RS-repair. Suppose $\mathcal{S}$ and Δ are fixed. Given a relation R, a RC ρ, 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 ϕ 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 ϕ evaluates to true.

We construct an instance for our decision problem as follows. The relation $R(A, B, C)$ contains three attributes A, B, C where the sensitive attribute is C. For a variable v_i and 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 ρ contains the lower bounds $\%c_j \geq \frac{1}{m}$, for all $c_j \in Dom(C)$. Note that ρ is an exact constraint 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 C-value if the repair satisfies ρ. We set $T = m$.

We show that ϕ has a satisfying assignment if and only if R has a RS-repair R' for Δ and ρ whose size is at least $T = m$.

(If). Consider any RS-repair R' of R w.r.t. Δ, ρ such that $|R'| \geq m$. R' satisfies ρ, 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 Δ, 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 ϕ, since clause c_j is satisfied for every $j \in [1, m]$, and consequently ϕ evaluates to true.

(Only if). Suppose that we have a satisfying assignment for variables $v_1, \ldots, v_n$ so that ϕ 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 ϕ, every clause c_ℓ is satisfied by at least one literal v_i, hence each C-value c_ℓ 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 $R' \models \rho$.

□

A.2 Reduction from Color-Balanced Matching to Simple LHS Marriage

As mentioned in Section 3.2, in this section we show that when the FDs Δ is a *simple LHS marriage* as defined below, which has been shown to be poly-time solvable for optimal S-repair, is as hard as the color-balanced matching problem studied in the algorithms literature.

- **Simple LHS Marriage:** Given relation $R(A, B, C)$, the FD set is $\Delta = \{A \to B, B \to A, A \to C\}$, C is the sensitive attribute with values $c_1, \cdots, c_k$ (k is constant), and the RC is $\rho = \{\rho_1, \rho_2, \cdots, \rho_k\}$, where $\sum_i \rho_i = 1$ (an exact RC). We assume set semantics, i.e., R does not have any duplicate tuples.

The S-repair problem for R and Δ has been shown to be polytime-solvable in [41].

The *exact-color bipartite matching problem* is defined as follows:

Definition 6 (Exact Colored Matching [28]). *Given an instance of an unweighted bipartite graph* $G = (U \cup V, E)$ *and* k *colors, where each edge is colored by one color. The color constraint is* $\phi = (\phi_1, \phi_2, \cdots, \phi_k)$, *where* ϕ_i *is the number of edges in color* i *in a matching* M *and* $\phi_i \geq 0$. *The number of colors* k *is a constant. A matching* M *is an exact colored matching in* G *if it satisfies the color constraint* ϕ.

The exact colored matching problem for two colors is known to be in RNC [49]. A recent study by K. A. et al. [28] shows that the problem for a constant number of colors k has a randomized polynomial-time algorithm by generalizing the algorithm presented in [49], and hence it is not NP-hard unless `NP = RP`.

The following lemma shows that the simple LHS marriage RS-repair is at least as hard as the exact colored matching problem.

Lemma 11. *The exact colored matching problem with color constraint can be reduced to the decision problem of finding an optimal RS-repair for simple LHS marriage in polynomial time.*

Proof. Recall the decision problem of computing an optimal RS-repair from Theorem 1: suppose $\mathcal{S}$ and Δ are fixed. Given a relation R, a RC ρ, 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 an instance of the exact colored matching problem with an unweighted bipartite graph $G = (U \cup V, E)$ and a color constraint $\phi = (\phi_1, \phi_2, \cdots, \phi_k)$, we construct an instance of the decision problem of finding an RS-repair for simple LHS marriage as follows:

- Create a relation $R(A, B, C)$, where C is the sensitive attribute with domain $\{c_1, c_2, \cdots, c_k\}$. For each edge $(u, v) \in E$ with color c_i, add a tuple (u, v, c_i) to R.
- Set the FD set $\Delta = \{A \to B, B \to A, A \to C\}$.

- Set the RC $\rho = \{\rho_1, \rho_2, \cdots, \rho_k\}$, where $\rho_i = \frac{\phi_i}{\sum_{i\in[1,k]}\phi_i}$ for $i \in \{1, 2, \cdots, k\}$.
- Set $T = \sum_{i\in[1,k]}\phi_i$

We claim that G has an exact colored matching M satisfying ϕ if and only if there is an RS-repair R' w.r.t. Δ and ρ such that $|R'| \geq T$.

(If). Given an RS-repair R' of R w.r.t. Δ and ρ with size at least T, we can construct an exact colored matching M in G as follows: for each tuple (u, v, c_i) in R', add the edge (u, v) with color c_i to M. Since R' satisfies the FDs in Δ, for any two tuples (u_1, v_1, c_{i_1}) and (u_2, v_2, c_{i_2}) in R', if $u_1 = u_2$, then $v_1 = v_2$, and if $v_1 = v_2$, then $u_1 = u_2$. This implies that no two edges in M share a common endpoint, and thus M is a matching. As R' satisfies ρ, the number of edges in M with color c_i is at least $|R'| \cdot \rho_i \geq \sum_{i\in[1,k]}\phi_i \cdot \frac{\phi_i}{\sum_{i\in[1,k]}\phi_i} = \phi_i$. If the number of edges in M with c_i is strictly greater than ϕ_i, we can remove arbitrary edges with color c_i from M to satisfy the exact color constraint ϕ_i. After this process, M becomes an exact colored matching satisfying ϕ.

(Only if). Given an exact colored matching M in G satisfying ϕ, we can construct an RS-repair R' of R as follows: for each edge (u, v) in M with color c_i, add the tuple (u, v, c_i) to R'. Since M is a matching, for any two tuples (u_1, v_1, c_{i_1}) and (u_2, v_2, c_{i_2}) in R', we have $u_1 \neq u_2$ and $v_1 \neq v_2$. Therefore, R' satisfies the FDs in Δ. As M satisfies ϕ, the number of tuples in R' with sensitive value c_i is exactly ϕ_i for each c_i, hence R satisfies that $\%c_i = \frac{\phi_i}{\sum_{i\in[1,k]}\phi_i} = \rho_i$. Therefore R' satisfies both Δ and ρ. Moreover, $|R'| = \sum_{i=1}^{k}\phi_i = T$, so R' is an RS-repair with size at least T.

The reduction takes polynomial time as it scans the edges in E once to construct the relation R and the RC ρ are directly set based on the given color constraint ϕ. □

A.3 Pseudocode and Walkthrough of PostClean

In this section, Example 12 gives a simple scenario about how PostClean works and the pseudocode of the PostClean(R, ρ) procedure from Section 3.3 is shown in Algorithm 6 followed by a detailed walk-through of the algorithm.

Example 12. *Assume that an input (R, ρ) 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 ρ, 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 algorithm computes the size (denoted by T) of the maximum subset (denoted by R') of R that satisfies the ρ. 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.

Algorithm 6 PostClean(R, ρ)

Input: A relation R and a RC ρ
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 ∅;
```

Specifically, τ_ℓ is assigned to the lower-bound value for the number of tuples with $A_s = a_\ell$. Then τ_ℓ 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.

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 τ_ℓs, representing the minimum size of R' that satisfies every lower-bound constraint. Note that T_0 can be larger than T, because τ_ℓ takes the ceiling for every a_ℓ. 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 ρ. If $T_0 < T$ (line 12), then there is a slack between the expected size T and the T_0 tuples by concatenating τ_ℓ tuples for every a_ℓ. This slack is fulfilled by retaining additional tuples with arbitrary value a_ℓ 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_ℓ and of size τ_ℓ.

Here is an example for how PostClean works:

A.4 Proof of Proposition 5

Proposition 5. *Given R and ρ, PostClean(R, ρ) returns in polynomial time a maximum subset R' of R 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_ℓ 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 τ_ℓ, 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 τ_ℓ, 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_ℓ 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 ρ:

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

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

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_ℓ in the active domain of A_s, which is $O(k) = O(|R|)$. In lines 4-8, we compute τ_ℓ 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 $O(|R|^2)$. □

A.5 Proof of Proposition 4

Proposition 4. *A set Δ of FDs reduces to the $\emptyset$ by repeated applications of consensus FD and common LHS simplifcations if and only if Δ 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 [41], 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 Δ that is not an LHS chain but was reduced to $\emptyset$ by consensus FD and common LHS simplifications. Since Δ is not an LHS chain, it has two FDs $f_1 : X_1 \rightarrow Y_1$ and $f_2 : X_2 \rightarrow Y_2$ such that $X_1 \not\subseteq X_2$ and $X_2 \not\subseteq X_1$. Hence there is 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 Δ until no more of either of these two simplifications can be applied. Without loss of generality, suppose f_1 was removed altogether from Δ first by the simplifications, and consider the last step j that removed the last attribute from X_1. Since $A_1 \in X_1 \setminus X_2$, A_1 cannot be removed by common LHS simplification in any step $\leq j$. Since at least A_1 remains in the LHS of f_1 in step j, f_1 cannot be removed from Δ by consensus FD since the LHS is not $\emptyset$. This contradicts the assumption that Δ was reduced to $\emptyset$, hence proved. □

B DETAILS IN SECTION 4

We present the missing details of Section 4 as follows:

- Size of Candidate Sets: Lemma 13
- Proof of Lemma 7 (Section 4)
- Proof of Lemma 9 (Section 4.1)
- Pseudocode of ReprInsert (Section 4.1)
- Proof of Lemma 10 (Section 4.2)
- Proof of Theorem 2 (Section 3.2)

B.1 Size of Candidate Sets

Lemma 13. *Given a relation R, an FD set Δ, 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[8], 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_ℓ in attribute A_s for every a_ℓ, 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:

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

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

B.2 Proof of Lemma 7

The time complexity of Reduce (Lemma 7) 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_{RDCE}, F_{COSNS}, F_{COLHS} respectively.

B.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:*

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

, 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 Δ, $|R|$ 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_ℓ}, and R_{y_ℓ} is a subset of R, so $|R_{y_\ell}| \le |R|$. Without loss of accuracy, the size of the candidate set of R_{y_ℓ} is also $O(|R|^k)$ as implied in Lemma 13.

In lines 2-7, ConsensusReduction iterates over each distinct y_ℓ in $O(|R|)$. Within the loop, it first computes $C_{R_{y_\ell}, \Delta - f}$, which is the candidate set of relation R_{y_ℓ} 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). □

B.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:*

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

, where $R_{x_\ell} = \sigma_{X=x_\ell} R$, X is the common LHS attribute, Δ_{-X} is removing column X from Δ, $|R|$ 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_ℓ}, $R_{x_1,\ldots,x_{\ell-1}}$ 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_ℓ in $O(|R|)$. Within the loop, it computes $C_{R_{x_\ell}, \Delta_{-X}}$, which is the candidate set of relation R_{x_ℓ} and FD set Δ_{-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). □

B.2.3 Time Complexity of Reduce and LhsChain-DP. In Lemma 7, 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 7. *LhsChain-DP terminates in $O(m \cdot |\Delta| \cdot k \cdot |R|^{3k+2})$ time, where m is the number of attributes in R, $|\Delta|$ is the number of FDs, and $k = |Dom(A_s)|$ is the domain size of the sensitive attribute A_s.*

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 ρ in $O(|R|^2)$ (Proposition 5) 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 Δ is empty in $O(1)$;
- otherwise, if Δ has a consensus FD by going ever each FD in Δ in $O(|\Delta|)$;

[8] 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.

- otherwise, if Δ has a common LHS by checking each attribute for common columns in the LHS of each FD in Δ in $O(m \cdot |\Delta|)$.

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

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

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

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

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

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

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

Intuitively, Reduce can be viewed as a tree traversal, where each node is a call of Reduce and the root corresponds to Reduce(R, Δ). Note that the call of Reduce by each leaf node is on a clean sub-relation and empty Δ. 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 Δ 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 $O(m \cdot |\Delta| \cdot k \cdot |R|^{3k+2})$. □

B.3 Proof of Lemma 9

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

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

$$S \coloneqq \{\text{PostClean}(R', \rho) \mid \forall R' \in C_{R,\Delta}\}, \tag{21}$$

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 Proposition 5, the output satisfies the RC ρ 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 ρ. 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 $|\text{PostClean}(R'', \rho)| = |\text{PostClean}(R', \rho)|$. 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 $|\text{PostClean}(R'', \rho)| \geq |\text{PostClean}(R', \rho)|$. 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 ρ.

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. □

B.4 Pseudocode of ReprInsert

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' ∈ C do
2:     if R' ≻_Repr R* or R' =_Repr R* then
3:         return C;
4:     else if R* ≻_Repr R' then
5:         C ← C \ {R'};
6:     end if
7: end for
8: return C ∪ R*;
```

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.

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 $O(k \cdot |R|^k)$. □

B.5 Proof of Lemma 10

In Lemma 10, we prove the correctness of Reduce, which relies on the correctness of ConsensusReduction and the correctness of CommonLHSReduction. Figures 6 and 7 show the workflow of these two reductions and repeat the notations used in the algorithms.

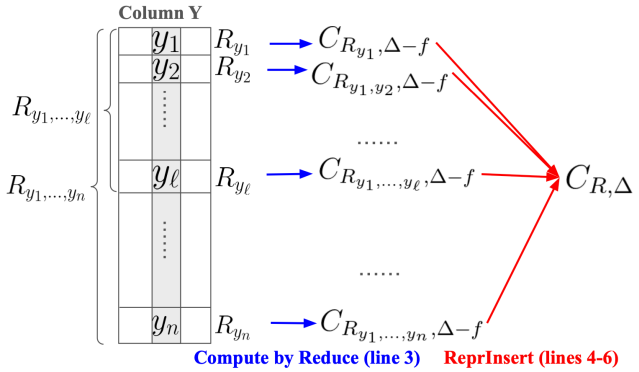


Figure 6: Notations and Workflow of ConsensusReduction. f is the consensus FD and Y is its RHS. $y_1, \ldots, y_n$ are distinct values of column Y. $R_{y_\ell} = \sigma_{Y=y_\ell} R$ for every $\ell \in [1, n]$. $C_{Ry_\ell,\Delta-f}$ is the candidate set of relation Ry_ℓ and FD set $\Delta - f$. $C_{R,\Delta}$ is the output of ConsensusReduction.

B.5.1 Correctness of Reduce (Lemma 10).

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

Proof. We prove it by induction on w, the number of reductions applied to Δ by Reduce. Note that because Reduce works on LHS chains, Δ must be able to reduced to the $\emptyset$ with a finite w. Assume that at each step of w, Reduce works on R and Δ, 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 Δ 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, Δ) will apply one of the following reductions:

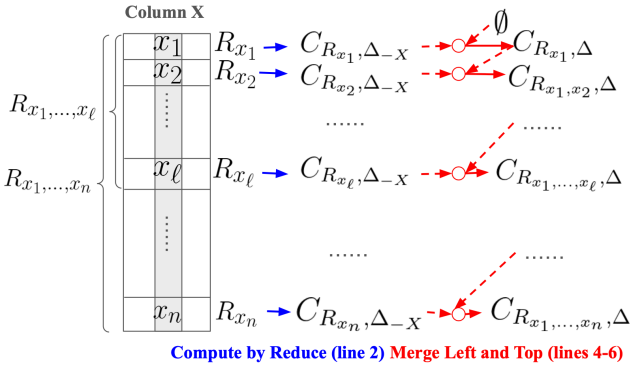


Figure 7: Notations and Workflow of CommonLHSReduction. X is the common LHS column. $x_1, \ldots, x_n$ are the distinct values of column X. $R_{x_\ell} = \sigma_{X=x_\ell} R$ for every $\ell \in [1, n]$. $Rx_1, \ldots, x_\ell = \sigma_{X=x_1 \vee \cdots \vee X=x_\ell} R$ for every $\ell \in [1, n]$. C_{Rx_ℓ,Δ_X} is the candidate set of relation Rx_ℓ and FD set Δ. $C_{Rx_1,\ldots,x_\ell,\Delta_{-X}}$ is the candidate set of relation $Rx_1, \ldots, x_\ell$ and FD set Δ_{-X} (i.e. removing X from Δ). $C_{R,\Delta}$ is the output of CommonLHSReduction.

(If Δ 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_ℓ 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, Δ):

- $R' \models \{f\}$, implying all tuples in R' agree on the same Y-value (say y_ℓ w.l.o.g.) and consequently R' is also an S-repair of R_{y_ℓ}.
- 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 Δ has a common LHS X). CommonLHSReduction is called (corresponding to lines 3-4). By the induction hypothesis, CommonLHSReduction(R, Δ) 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_ℓ in $Dom(X)$. To achieve this, for each $\ell \in [1, n]$, $C_{R_{x_1,\dots,x_\ell},\Delta}$ is derived from $C_{R_{x_1,\dots,x_{\ell-1}},\Delta}$ and $C_{R_{x_\ell},\Delta_{-X}}$ by inserting each candidate in

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

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

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 Δ as well as Δ_{-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 $C_{R_{x_1,\dots,x_\ell},\Delta}$, we argue that:

- $\bar{R} = (\bar{R} \cap R_{x_\ell}) \cup (\bar{R} \cap R_{x_1,\dots,x_{\ell-1}})$, because R_{x_ℓ} and $R_{x_1,\dots,x_{\ell-1}}$ 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 $\bar{R} \cap R_{x_1,\dots,x_{\ell-1}}$ is a candidate of $(R_{x_1,\dots,x_{\ell-1}}, \Delta)$ (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_ℓ} (resp. $R_{x_1,\dots,x_{\ell-1}}$). Secondly, $\bar{R} \cap R_{x_\ell}$ (resp. $\bar{R} \cap R_{x_1,\dots,x_{\ell-1}}$) is not representatively dominated by any other candidate in $C_{R_{x_\ell},\Delta_{-X}}$ (resp. $C_{R_{x_1,\dots,x_{\ell-1}},\Delta}$). Otherwise, if such a candidate R_0 exists in $C_{R_{x_\ell},\Delta_{-X}}$ (resp. $C_{R_{x_1,\dots,x_{\ell-1}},\Delta}$), then $R_0 \cup (\bar{R} - R_{x_\ell})$ must be a candidate of $C_{R_{x_1,\dots,x_\ell},\Delta}$ 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

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

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 $C_{R_{x_1,\dots,x_\ell},\Delta}$ for every $\ell \in [1, n]$. Since $R_{x_1,\dots,x_n} = R$, we have $C_{R,\Delta} \leftarrow C_{R_{x_1,\dots,x_n},\Delta}$.

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

□

B.6 Proof of Theorem 2

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

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 ρ is fixed and Δ forms an LHS chain. The proof consists of two parts: (a) Reduce correctly computes the candidate set $C_{R,\Delta}$ (proved in Lemma 10) and (b) Lines 2-3 of LhsChain-DP compute the optimal RS-repair by applying PostClean to $C_{R,\Delta}$ (proved in Lemma 9).

(Time complexity). The time complexity of LhsChain-DP is $O(m \cdot |\Delta| \cdot k \cdot |R|^{3k+2})$ as implied by Lemma 7. □

C DETAILS FROM SECTION 5

C.1 LP Rounding-based Algorithms

Besides LP + ReprRounding, we also show another polynomial-time rounding procedure, LP + GreedyRounding, in Appendix C.1.2. They differs in whether the rounding considers representation or not. From the experiments, we see that they have close performance while LP + ReprRounding is better in scenarios where the dataset contains a large number of errors.

C.1.1 Representation-Aware Rounding (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_is to 1, tuples from underrepresented groups are prioritized. Inspired by stratified sampling [50], 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. To alleviate this issue in LP + GreedyRounding, we propose a representat-ion-aware enhancement in LP + ReprRounding as outlined in Algorithm 8. 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 [50], 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_ℓ to the expected proportion p_ℓ. 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

Algorithm 8 LP + ReprRounding(R, Δ, ρ)

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', ρ);
10: **end if**
11: **return** $\emptyset$;

additional representation considerations in this step. Similarly, the complexity of this rounding is $O(|R|^2)$.

C.1.2 Greedy Rounding (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 Δ are satisfied, it calls PostClean (Section 3.3) to ensure that ρ 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 ρ.

Algorithm 9 LP + GreedyRounding(R, Δ, ρ)

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', ρ);
9: **end if**
10: **return** $\emptyset$;

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.

D MORE EXPERIMENTAL RESULTS

D.1 More results for Example 1

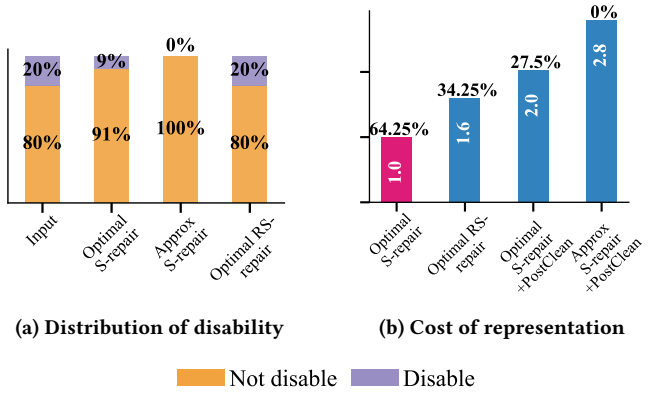


Figure 8: Cost of representation for ACS data and disability status (relative noise distribution 20%-80%).

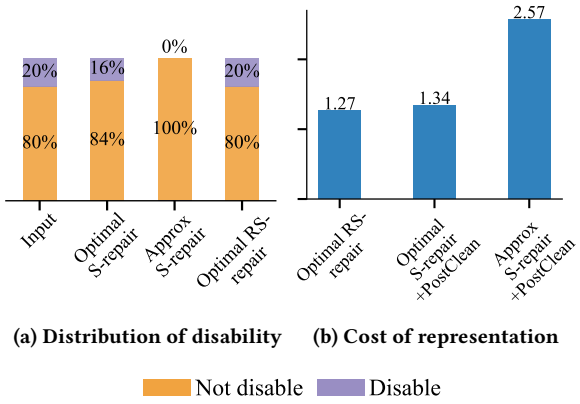


Figure 9: Cost of representation for ACS data and disability status (relative noise distribution 33%-67%)

D.2 Additional experiments from Section 6.3

We first present the repair quality for ACS and COMPAS data of different value distributions of the sensitive attribute for 5% noise level in Table 8.

D.3 Additional experiments from Section 6.4

Runtime on ACS and COMPAS Data. Table 9 demonstrates the runtime for ACS and COMPAS data of different value distributions of sensitive attribute (5% and 10% noise level, chain and non-chain FD sets).

Table 8: Repair quality of different algorithms for ACS data and COMPAS data (5% noise level) with chain and non-chain FD sets.

(a) ACS: chain FD set												
Sensitive attribute distribution	**80%-20%**				**60%-40%**				**50%-50%**			
Algorithm / Size (K)	**4**	**6**	**8**	**10**	**4**	**6**	**8**	**10**	**4**	**6**	**8**	**10**
GlobalILP	100	100	100	100	100	100	100	100	100	100	100	100
LhsChain-DP (= FDCleanser)	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
ILP-Baseline+PostClean	93.48	93.73	94.62	94.65	98.95	99.30	99.21	99.28	99.94	99.95	99.99	99.93
DP-Baseline+PostClean	93.48	93.73	94.62	94.65	98.95	99.30	99.21	99.28	99.94	99.95	99.99	99.93
MuSe-Baseline+PostClean	92.87	94.31	94.79	-	-	-	-	-	-	-	-	-

(b) ACS: non-chain FD set												
Sensitive attribute distribution	**80%-20%**				**60%-40%**				**50%-50%**			
Algorithm / Size (K)	**4**	**6**	**8**	**10**	**4**	**6**	**8**	**10**	**4**	**6**	**8**	**10**
GlobalILP	100	100	100	100	100	100	100	100	100	100	100	100
FDCleanser	**96.68**	**98.63**	**98.75**	**99.15**	**97.96**	**97.89**	**97.76**	**98.00**	**98.04**	**98.79**	**98.84**	**98.98**
LP + ReprRounding	81.21	90.07	90.71	90.51	93.78	88.28	87.11	92.73	**99.36**	98.67	**99.45**	**99.08**
ILP-Baseline+PostClean	79.34	84.93	85.43	86.01	92.94	93.67	93.60	94.31	**98.26**	98.69	98.55	**99.13**
VC-approx-Baseline+PostClean	0.43	6.34	6.72	5.09	36.68	37.84	36.74	36.94	48.86	50.74	49.72	50.50

(c) COMPAS: chain FD set											
Sensitive attribute distribution	**80%-20%**				**60%-40%**				**50%-50%**		
Algorithm / Size (K)	**4**	**10**	**20**	**30**	**4**	**10**	**20**	**30**	**4**	**10**	**20**
GlobalILP	100	100	100	100	100	100	100	100	100	100	100
LhsChain-DP	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
ILP-Baseline+PostClean	99.61	100	100	100	100	100	100	100	100	100	100
DP-Baseline+PostClean	99.61	100	100	100	100	100	100	100	100	100	100
MuSe-Baseline+PostClean	98.83	-	-	-	-	-	-	-	-	-	-

(d) COMPAS: non-chain FD set											
Sensitive attribute distribution	**80%-20%**				**60%-40%**				**50%-50%**		
Algorithm / Size (K)	**4**	**10**	**20**	**30**	**4**	**10**	**20**	**30**	**4**	**10**	**20**
GlobalILP	100	100	100	100	100	100	100	100	100	100	100
FDCleanser	**99.54**	**96.06**	**98.15**	**97.44**	**99.28**	**98.70**	**97.60**	**97.08**	**99.13**	**99.60**	**99.33**
LP + ReprRounding	100	100	100	100	100	100	100	100	100	100	100
ILP-Baseline+PostClean	99.08	95.87	93.78	91.35	99.19	98.59	97.47	96.98	99.80	99.73	99.55
VC-approx-Baseline+PostClean	16.09	13.26	10.44	8.34	33.12	30.99	30.86	29.97	43.62	46.07	44.76

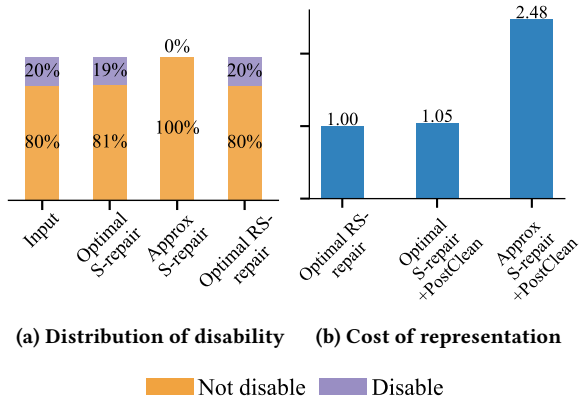


Figure 10: Cost of representation for ACS data and disability status (relative noise distribution 50%-50%)

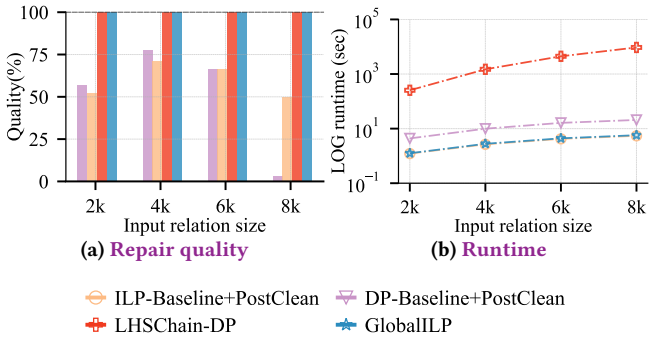


Figure 11: Repair quality and runtime for Flight data (chain FDs)

Runtime on Flight Data. Figure 11b shows the runtimes for the Flight dataset that has chain FDs. While both LhsChain-DP and GlobalILP provides the optimal RS-repair, GlobalILP is faster than LhsChain-DP (and other baselines) for smaller datasets GlobalILP. Since the tuples are quite distinct on the attribute "DF" (only 2 or 3 tuples sharing the same key), LhsChain-DP takes a much longer runtime to loop over all new repairs that are potential to be a candidate in the step of CommonLHSReduction. LhsChain-DP seems to take a longer runtime when the value of common LHS has a sparse domain.

D.4 Experiments on Credit Card Transaction Dataset

D.4.1 Setup. Credit card transaction [10] dataset provides detailed transaction records, including associated personal and merchant information, which can be used for fraud detection. 7 attributes are involved in our experiments.

We consider a non-chain FD set here: {Merchant → Category; City → State; First, Last → Gender; First, Last → City}. we use an RC on the sensitive attribute Gender, {%Male = $\frac{20}{100}$, %Female =

Table 9: Runtime (sec) of different algorithms for ACS data (5% and 10% noise levels) with chain and non-chain FD sets

(a) ACS: non-chain FD set (5% noise)												
Sensitive attribute distribution	**80%-20%**				**60%-40%**				**50%-50%**			
Algorithm / Size (K)	**4**	**6**	**8**	**10**	**4**	**6**	**8**	**10**	**4**	**6**	**8**	**10**
GlobalILP	313.49	1003.48	3188.59	17840.45	64.97	260.79	1113.37	3781.58	30.08	161.76	903.63	2017.56
FDCleanser	2.53	2.43	3.57	4.33	4.60	7.87	12.09	15.23	5.72	10.36	13.47	27.20
LP + ReprRounding	31.51	69.82	117.88	206.39	21.69	52.70	93.31	174.77	11.36	28.30	48.38	133.10
ILP-Baseline+PostClean	62.60	101.64	250.12	619.45	34.40	100.44	209.47	475.07	24.58	124.47	182.43	518.43
VC-approx-Baseline+PostClean	3.20	3.82	7.38	14.02	2.08	4.77	6.87	14.82	2.11	3.39	5.90	27.98

(b) ACS: non-chain FD (10% noise)												
Sensitive attribute distribution	**80%-20%**				**60%-40%**				**50%-50%**			
Algorithm / Size (K)	**4**	**6**	**8**	**10**	**4**	**6**	**8**	**10**	**4**	**6**	**8**	**10**
GlobalILP	1869.85	5072.64	58977.39	64039.06	261.93	1344.63	4293.89	44326.58	139.27	855.29	1227.35	3328.55
FDCleanser	2.29	2.48	2.52	2.36	5.72	10.33	12.52	16.64	9.50	12.09	13.85	23.14
LP + ReprRounding	33.83	87.84	165.72	278.52	39.40	97.13	248.89	397.02	44.67	114.99	117.65	250.03
ILP-Baseline+PostClean	135.41	412.93	869.86	673.26	104.51	307.65	1000.11	2961.43	116.17	358.94	909.23	2404.93
VC-approx-Baseline+PostClean	3.19	7.13	13.88	20.72	3.34	7.31	13.37	19.87	3.84	11.57	11.42	18.89

(c) ACS: chain FD (5% noise)												
Sensitive attribute distribution	**80%-20%**				**60%-40%**				**50%-50%**			
Algorithm / Size (K)	**4**	**6**	**8**	**10**	**4**	**6**	**8**	**10**	**4**	**6**	**8**	**10**
GlobalILP	5.02	13.37	94.12	90.16	3.27	14.11	47.89	308.90	1.92	4.53	14.30	49.97
LhsChain-DP (= FDCleanser)	3.49	4.80	8.41	9.06	3.73	5.88	7.14	14.61	4.78	7.72	8.24	11.75
DP-Baseline+PostClean	0.41	0.50	0.59	0.62	0.81	0.52	0.56	0.88	0.53	0.55	0.60	0.69
ILP-Baseline+PostClean	1.17	2.74	5.78	8.15	1.49	2.82	5.48	9.76	1.59	3.51	6.65	13.74
MuSe-Baseline+PostClean	460.05	3904.73	15161.43	-	-	-	-	-	599.76	3850.74	14413.26	-

(d) ACS: chain FD (10% noise)												
Sensitive attribute distribution	**80%-20%**				**60%-40%**				**50%-50%**			
Algorithm / Size (K)	**4**	**6**	**8**	**10**	**4**	**6**	**8**	**10**	**4**	**6**	**8**	**10**
GlobalILP	5.74	64.79	413.00	1045.78	11.40	209.54	544.95	657.28	9.97	13.10	116.37	129.90
LhsChain-DP (= FDCleanser)	5.64	6.75	9.84	12.12	5.32	8.71	11.08	28.91	7.11	9.62	12.32	14.35
DP-Baseline+PostClean	0.55	0.64	0.69	0.71	0.55	0.64	0.86	1.02	0.63	0.67	0.73	0.91
ILP-Baseline+PostClean	2.24	4.85	9.65	15.95	2.28	5.94	12.61	19.60	2.86	7.10	12.68	26.69
MuSe-Baseline+PostClean	4558.15	16187.80	-	-	3075.89	23414.92	-	-	12328.36	14197.39	-	-

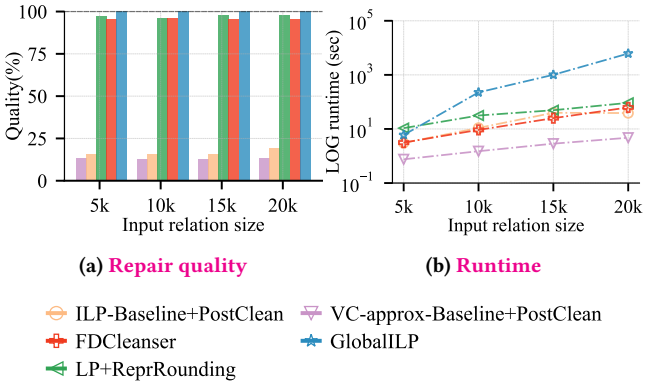
(e) COMPAS: non-chain FD set (5% noise)											
Sensitive attribute distribution	**80%-20%**				**60%-40%**				**50%-50%**		
Algorithm / Size (K)	**4**	**10**	**20**	**30**	**4**	**10**	**20**	**30**	**4**	**10**	**20**
GlobalILP	124.48	1724.06	3038.94	8083.23	114.57	1145.06	3689.82	6727.07	130.72	1625.59	6766.93
FDCleanser	18.73	236.34	1413.43	5237.60	24.86	240.79	1826.81	5966.77	31.47	278.60	1985.86
LP + ReprRounding	23.48	148.77	601.35	1434.94	26.45	186.49	1656.01	3558.59	30.14	186.87	1104.72
ILP-Baseline+PostClean	131.51	1428.27	5331.15	12498.97	101.97	1046.61	5898.54	13983.64	123.99	1280.14	11003.02
VC-approx-Baseline+PostClean	5.41	34.79	148.39	311.44	6.92	40.30	316.29	469.01	8.35	34.84	359.49

(e) COMPAS: non-chain FD set (10% noise)											
Sensitive attribute distribution	**80%-20%**				**60%-40%**				**50%-50%**		
Algorithm / Size (K)	**4**	**10**	**20**	**30**	**4**	**10**	**20**	**30**	**4**	**10**	**20**
GlobalILP	294.09	1620.48	5289.97	10235.06	221.40	1672.16	5294.01	10742.42	242.20	1415.72	8104.47
FDCleanser	13.70	82.28	646.44	2110.72	18.13	179.43	1518.88	3924.87	23.66	328.31	3365.94
LP + ReprRounding	56.36	421.65	2881.62	18485.24	65.67	373.81	5688.91	19661.11	47.12	309.59	1789.07
ILP-Baseline+PostClean	294.09	1620.48	5289.97	10235.06	221.40	1672.16	5294.01	10742.42	242.20	1415.72	8104.47
VC-approx-Baseline+PostClean	9.58	59.64	254.31	632.66	8.80	62.63	286.61	909.84	8.95	56.94	271.85

(c) COMPAS: chain FD set (5% noise)											
Sensitive attribute distribution	**80%-20%**				**60%-40%**				**50%-50%**		
Algorithm / Size (K)	**4**	**10**	**20**	**30**	**4**	**10**	**20**	**30**	**4**	**10**	**20**
GlobalILP	2.48	21.12	96.14	211.90	2.86	25.25	122.54	234.58	2.49	26.85	114.82
LhsChain-DP (= FDCleanser)	0.04	0.06	0.07	0.09	0.04	0.06	0.09	0.10	0.04	0.05	0.07
ILP-Baseline+PostClean	2.09	18.06	86.95	177.53	2.81	20.09	121.65	214.16	2.58	19.78	95.52
DP-Baseline+PostClean	0.03	0.04	0.05	0.07	0.04	0.04	0.06	0.09	0.05	0.04	0.05
MuSe-Baseline+PostClean	11071.22	-	-	-	-	-	-	-	-	-	-

(c) COMPAS: chain FD set (10% noise)											
Sensitive attribute distribution	**80%-20%**				**60%-40%**				**50%-50%**		
Algorithm / Size (K)	**4**	**10**	**20**	**30**	**4**	**10**	**20**	**30**	**4**	**10**	**20**
GlobalILP	5.32	65.48	226.84	443.41	7.18	63.17	234.84	417.15	5.97	89.14	244.40
LhsChain-DP (= FDCleanser)	0.05	0.07	0.08	0.16	0.05	0.06	0.07	0.08	0.04	0.05	0.07
ILP-Baseline+PostClean	4.31	43.77	187.56	401.77	5.83	45.29	241.47	364.47	4.88	43.49	217.83
DP-Baseline+PostClean	0.03	0.04	0.05	0.08	0.04	0.04	0.05	0.07	0.06	0.04	0.05
MuSe-Baseline+PostClean	38218.90	-	-	-	-	-	-	-	-	-	-

$\frac{80}{100}$}. The method of noise input generation is the same as that used in ACS and COMPAS data (Section 6.1.4).



(a) Repair quality **(b) Runtime**

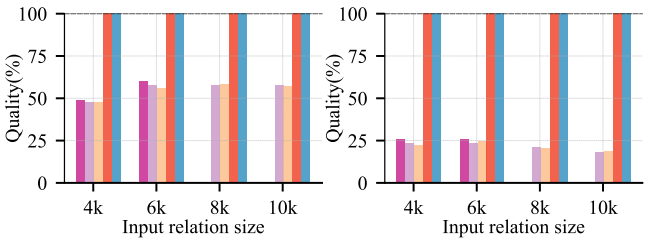
Figure 12: Repair quality and runtime of different algorithms for Credit card transaction data (5% noise)

D.4.2 RS-repair Quality and Runtime Cost. Figure 12 presents the repair quality and runtime of different algorithms in the Credit card transaction dataset under the noise level 5%. As shown in Figure 12a, both LP + ReprRounding and FDCleanser achieve high quality above 95%, while LP + ReprRounding performs slightly better than FDCleanser. However, the repair quality of ILP-Baseline+PostClean and VC-approx-Baseline+PostClean achieved is significantly lower than the optimal 100% repair quality provided by GlobalILP.

Figure 12b demonstrates that the runtime of GlobalILP increases exponentially as the input size increases, but LP + ReprRounding and LhsChain-DP provide much more efficient alternatives. Moreover, LhsChain-DP takes less time than LP + ReprRounding while achieving high repair quality as well.

D.5 Additional experiments varying noise levels

In this section, we present more results for larger noise levels, 15% and 20% for chain FDs in Figure 13 and Tables 10 and 11



(a) ACS: chain FD set (15% noise) **(b) ACS: chain FD set (20% noise)**

Figure 13: Repair Quality while varying the noise level

E DETAILS FROM SECTION 8

E.1 Discussion on Multiple Sensitive Attributes

We discuss several next steps for the extension of multiple sensitive attributes. First, a few definitions need to be extended: (i) b sensitive attributes, denoted by $A_{s_1}, \ldots, A_{s_b}$, of $\mathcal{S}$ and b associated RCs, denoted by $\rho_1, \ldots, \rho_b$; (ii) the second item in Definition 2, i.e., the definition of RS-repair, will be changed to *R' satisfies all b RCs, i.e., $\rho_1, \ldots, \rho_b$*. In other words, a RS-repair preserves the marginal distribution, given by the RC, for every sensitive attribute. Second, cases where handling a single sensitive attribute is NP-hard remain NP-hard when extended to multiple sensitive attributes. The rationale behind this is that the single sensitive attribute problem can be viewed as a special instance of the multiple sensitive attributes problem. Third, GlobalILP and its ILP (Equation (1)) can be extended to handle the problems with b RCs, $\rho_1, \ldots, \rho_b$. The constraint in the middle will be changed to:

$$\sum_{i:t_i[A_{s_j}]=a_\ell} x_i \geq \rho_j(a_\ell) \times \sum_i x_i \quad \text{for all } j \in [1,b] \text{ and } a_\ell \in Dom(A_{s_j})$$

Fourth, the representation of a tuple becomes multi-dimensional, so the impact of deleting it or preserving it is also multi-dimensional, which further complicates the extension of PostClean and candidate set. To be specific, in the cases with a single sensitive attribute, PostClean can greedily decide on the tuples to delete until it satisfies the RC, but it is unclear that which set of tuples to remove when there are multiple RCs. The cause is, for multiple sensitive attributes, deleting a tuple reduces the representation of a combination of sensitive values. For example, considering a tuple with two sensitive values, `Asian` and `Female`, deleting it might not be ideal if `Asian` is over-represented on Race but `Female` is under-represented in Sex. Hence, the extension of DP is possible but not trivial. Essentially, we can keep track of more information for all the sensitive attributes, but it is unclear how to extend the definition of candidate set (Definition 4), which means it is uncertain to declare if one S-repair is representatively equivalent to or representatively dominates the other. Therefore, the size of the candidate set is unknown and consequently the complexity of DP is unknown.

Table 10: COMPAS: (Chain FD set) Repair Quality

noise				15%							20%			
size	4k	6k	8k	10k	20k	30k	40k	4k	6k	8k	10k	20k	30k	40k
MuSe-Baseline+PostClean	95.50													
VC-approx-Baseline+PostClean	93.84	96.46	95.04	95.77	95.29	96.95	97.48	91.04	93.01	93.37	92.35	92.40	93.50	94.55
DP-Baseline+PostClean	93.84	96.46	95.04	95.77	95.29	96.95	97.48	91.04	93.01	93.37	92.35	92.40	93.50	94.55
ILP-Baseline+PostClean	93.84	96.46	95.04	95.77	95.29	96.95	97.48	91.04	93.01	93.37	92.35	92.40	93.50	94.55
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
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

Table 11: COMPAS: (Chain FD set) Run time cost

noise				15%							20%			
size	4k	6k	8k	10k	20k	30k	40k	4k	6k	8k	10k	20k	30k	40k
MuSe-Baseline	74113.50													
VC-approx-Baseline	0.03	0.04	0.04	0.04	0.05	0.07	0.09	0.03	0.04	0.04	0.04	0.05	0.07	0.09
DP-Baseline	0.03	0.03	0.04	0.04	0.06	0.11	0.23	0.03	0.03	0.04	0.04	0.05	0.09	0.12
ILP-Baseline	6.70	18.10	36.64	65.74	336.20	770.82	1811.51	8.92	32.38	49.07	79.95	331.32	932.78	2463.53
LHSChain-DP	0.05	0.06	0.08	0.07	0.11	0.15	0.18	0.06	0.08	0.08	0.11	0.11	0.28	0.36
GlobalILP	8.33	30.18	85.78	179.88	463.69	1376.47	1114.64	11.72	47.65	100.34	202.19	390.46	1590.58	2390.69