From Minimum Change to Maximum Density: On S-Repair under Integrity Constraints

Yu Sun
Tsinghua University
sy17@mails.tsinghua.edu.cn

Shaoxu Song
Tsinghua University
sxsong@tsinghua.edu.cn

***Abstract*—To clean dirty data, integrity constraints are often employed. A typical S-repair model removes a minimal set of tuples (to avoid excessive removal and information loss) such that the integrity constraints are no longer violated in the remaining tuples. However, multiple candidates of minimal removal sets exist and are difficult to determine. We intuitively notice that a clean tuple often has more close neighbors (i.e., higher density) than dirty tuples. In this sense, our study proposes to return the S-repair under integrity constraints with the highest density, among various minimal removal sets. We explicitly analyze the hardness of maximizing S-repair density under integrity constraints, together with efficient approximation. Extensive experiments over real datasets collected from industry with real-world errors show that our proposal can achieve higher accuracy in cleaning dirty tuples, compared to the state-of-the-art methods.**

I. Introduction

Dirty data have been widely observed for various reasons, such as value misplacement [19], sensor failure [18], disguised missing value [15], etc. For instance, Figure 1 illustrates some example subway data, which are collected by our partner company. It collects Load sensor readings in subway trains and the corresponding passenger Count in factory pressure testing. Owing to sensor failure, the Load values in tuples t_1, t_2, t_3, t_4 are missing and recorded using a disguised value 0. Moreover, the Count values of t_5, t_6 are misplaced in entry. It is not surprising that such dirty tuples affect downstream applications, such as passenger count monitoring or query processing [9].

Integrity constraints are often employed to identify the errors. For example, in Figure 1, a functional dependency (FD) Load $\rightarrow$ Count identifies t_1, t_2, t_3, t_4 in violation with t_0, which have the same Load 0 but different Count. Similar violations are noticed in t_5, t_7 w.r.t. the FD as well as in t_6, t_8.

To tackle the errors, the S-repair model [11] removes a minimal set of tuples such that the remaining tuples satisfy the constraints, in order to avoid excessive removal and information loss [2]. For instance, by removing t_1, t_2, t_3, t_4, t_5, t_6, it is sufficient to eliminate the violations w.r.t. the FD, and there is no need to further remove t_0, t_7, t_8 in the aforesaid violations. However, multiple candidates of minimal removal sets exist. That is, other minimal removal sets include $\{t_0, t_2, t_3, t_4, t_5, t_6\}$, $\{t_0, t_1, t_3, t_4, t_5, t_6\}$, $\{t_0, t_1, t_2, t_4, t_5, t_6\}$, and so on. Unfortunately, without further knowledge, it is difficult for S-repair to determine which removal set is indeed proper.

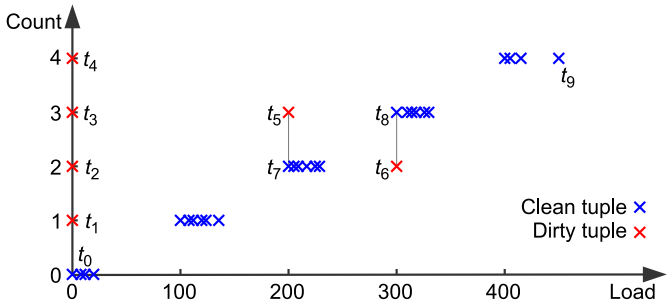

Fig. 1. Example subway data with tuples t_0, t_1, t_2, t_3, t_4 in violation to an FD Load $\rightarrow$ Count

The widely used minimum change criteria [11] does not help, given the same size 6 of all the minimal removal sets.

Recent study [19] shows that distances to neighbors could be a good signal of identifying dirty data. Intuitively, a clean tuple (e.g., t_0) often has more close neighbors than dirty tuples (such as t_1, t_2, t_3, t_4). Therefore, in this study, we investigate the density of tuples w.r.t. their nearest neighbors. Informally, the closer a tuple is to its k neighbors, the higher the density is (see formal Definition 3). Figure 2 ranks the density of sample tuples from the subway data. As shown, dirty (red) tuples with errors have generally lower density than the clean ones in blue.[1] In this sense, we can determine a proper S-repair by observing the corresponding density.

Hence, in this paper, we propose to find the optimal S-repair under integrity constraints with the highest density. That is, the minimal removal set $\{t_1, t_2, t_3, t_4, t_5, t_6\}$ will be returned, while the remaining tuples $\{t_0, t_7, t_8, \ldots\}$ satisfy the FD and have the highest density.

Contribution: To summarize, we make the following contributions. (1) We formalize the density maximization problem for S-repair under integrity constraints in Section II. It is to determine the minimal removal set such that the remaining tuples satisfy the constraints and have the maximum density. The hardness of the problem is explicitly analyzed (Theorem 2). (2) We devise a heuristic solution in Section III, referring to the aforesaid hardness analysis. The heuristic algorithm indeed returns the optimal solution in certain cases (Proposition 5). (3) We conduct an extensive experimental evaluation in Section

[1] Outliers such as t_9 in Figure 1 may have low density in Figure 2. However, it does not violate the FD and thus will not be repaired.

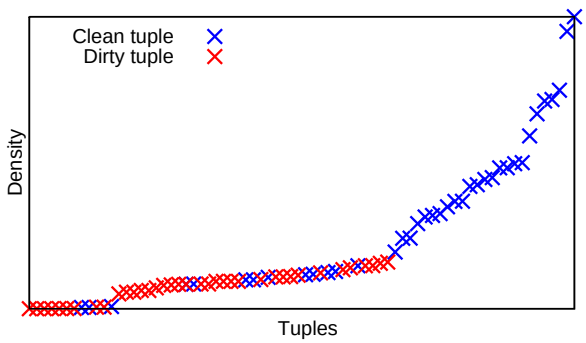

Fig. 2. Density of some clean (blue) and dirty (red) tuples in the subway data

IV, over real datasets collected from industry with real-world errors. The experiments demonstrate the superiority of our proposal in accurately cleaning dirty tuples.

II. Problem Statement

In this section, we introduce the terminology used in this paper and formalize the density maximization problem. Consider a relational instance $I = \{t_1, \ldots, t_n\}$ over schema $A = (A_1, \ldots, A_m)$. Each tuple $t_i \in I$ is a collection of cells $\{t_i[A_1], t_i[A_2], \ldots, t_i[A_m]\}$, where $t_i[A_j]$ denotes the value of attribute A_j in tuple t_i. Let us first introduce S-repair for cleaning and density over I.

A. S-repair under Integrity Constraints

We employ a general notation of denial constraints (DC) [5] to represent the integrity constraints. The reason is that DC subsumes several types of integrity constraints such as functional dependencies (FD), conditional functional dependencies (CFD) [3], fixing rules [20] and metric functional dependencies [10]. Therefore, DC is widely used in various state-of-the-art data cleaning approaches [6], [16]. Given a set of operators $B = \{=, \neq, >, <, \geq, \leq, \approx\}$, the binary denial constraint is a formula of the form $\varphi : \forall t_i, t_l \in I, \neg(P_1 \wedge \cdots \wedge P_j \wedge \cdots \wedge P_J)$, where each predicate P_j is of the form $v_1 \phi v_2$ or $v_1 \phi h$ with $v_1, v_2 \in t_i[A_j], t_i \in I, A_j \in A$, and h is a constant.

For any tuples $t_i, t_l \in I$ and a denial constraint φ defined over A, we say tuples t_i and t_l violate the constraint φ, denoted by $(t_i, t_l) \not\models \varphi$, if these two tuples satisfy all the violation conditions defined in the predicates $(P_1 \wedge \cdots \wedge P_j \wedge \cdots \wedge P_J)$ in φ. Otherwise, we write $(t_i, t_l) \models \varphi$ to denote tuples t_i and t_l satisfy the constraint φ. For a relational instance I of A, we write $I \not\models \varphi$ if there exists at least a pair of tuples $t_i, t_l \in I$ having $(t_i, t_l) \not\models \varphi$. Otherwise, if no tuple in I violates the constraint φ, we use $I \models \varphi$ to denote that the instance I satisfies constraint φ. If we have a set of denial constraints Σ, $I \models \Sigma$ if and only if I satisfies each constraint in Σ, i.e., $\forall \varphi \in \Sigma, I \models \varphi$. Otherwise, we say the instance I violates the constraints Σ, i.e., $I \not\models \Sigma$.

To eliminate the violations, the removal set I_N is a subset of I such that the remaining tuples in $I \setminus I_N$ satisfy the constraints of Σ.

Definition 1 (Removal Set)**.** *Given a set of integrity constraints Σ defined over schema A, and an instance I of A, $I_N \subset I$ is a* removal set *w.r.t. Σ if $I \setminus I_N \models \Sigma$.*

Intuitively, the removal set denotes the tuples with potential errors, introducing violations to the integrity constraints Σ. Obviously, the removal should not be excessive to avoid information loss [2]. The S-repair model [11] thus considers the minimal removal set.

Definition 2 (Minimal Removal Set)**.** *A removal set I_N is* minimal *if there does not exist any other removal set I'_N of I, such that $I'_N \subset I_N$.*

That is, a minimal removal set I_N does not contain any tuple that can be put back without introducing violations to the integrity constraints Σ. Unfortunately, multiple candidates of minimal removal sets may exist for eliminating the violations. Without further knowledge, it is often difficult for S-repair to determine which minimal removal set is indeed proper.

Example 1. *Consider the relational instance I in Figure 1. The* FD *Load $\rightarrow$ Count described in Section I can be expressed by the following* DC

$$\forall t_i, t_l \in I, \neg(t_i[\mathit{Load}] = t_l[\mathit{Load}] \wedge t_i[\mathit{Count}] \neq t_l[\mathit{Count}]).$$

It states that for any two points having the same Load value, there would be a violation if their Count values are different. As shown, tuples $t_0 - t_8$ are involved in violation. Constraint-based S-repair can generate all the possible minimal removal sets $\{t_1, t_2, t_3, t_4, t_5, t_6\}$, $\{t_0, t_2, t_3, t_4, t_5, t_6\}$, $\{t_0, t_1, t_3, t_4, t_5, t_6\}$, $\{t_0, t_1, t_2, t_4, t_5, t_6\}$, $\{t_0, t_1, t_2, t_3, t_5, t_6\}$, and so on, but fail to identify which one denotes the true error tuples.

B. Density Maximization for S-repair

Let $d_j(t_i, t_l)$ be any distance metric [14] to evaluate the distance between two tuples over an attribute $A_j \in A$, having $0 \leq d_j(t_i, t_l) \leq 1$. We denote $L_I^k(t_i)$ the k neighbors with the minimum distances to t_i in I. The density of instance I is thus defined as the aggregation of $1 - d_j(t_i, t_l)$ of k nearest neighbors t_l for each tuple t_i in I.

Definition 3 (Density)**.** *The density of a relational instance I is*

$$\mathcal{D}(I) = \sum_{t_i \in I} \sum_{t_l \in L_I^k(t_i)} \sum_{A_j \in A} (1 - d_j(t_i, t_l)), \tag{1}$$

where $L_I^k(t_i)$ is the k nearest neighbors for each tuple $t_i \in I$.

The smaller the distances are to the k nearest neighbors, the higher the density is. Likewise, for a removal set I_N of I, the density of remaining tuples in $I \setminus I_N$ is

$$\mathcal{D}(I \setminus I_N) = \sum_{t_i \in I \setminus I_N} \sum_{t_l \in L_{I \setminus I_N}^k(t_i)} \sum_{A_j \in A} (1 - d_j(t_i, t_l)). \tag{2}$$

As illustrated in Section I, among multiple candidates of minimal removal sets, we propose to determine the S-repair such that the remaining tuples have the maximum density.

Problem 1 (Density Maximization)**.** *Given a relational instance I over schema A that violates a set of integrity constraints Σ, the* density maximization *problem is to find a removal set I_N of I w.r.t. Σ such that the density $\mathcal{D}(I \setminus I_N)$ is maximized.*

It is worth noting that Problem 1 does not explicitly state whether the optimal removal set I_N of I is minimal or not. Indeed, we show below that the returned removal set I_N with the maximum density is always a minimal removal set.

Proposition 1. *The removal set I_N^* with the maximum density $\mathcal{D}(I \setminus I_N^*)$ is always a minimal removal set.*

In order to investigate the hardness of Problem 1, we consider its corresponding decision problem.

Problem 2 (Density Checking)**.** *Given a relational instance I over schema A that violates a set of integrity constraints Σ, and a constant ζ, the* density checking *problem is to decide whether there exists a removal set I_N with density $\mathcal{D}(I \setminus I_N) \geq \zeta$.*

As illustrated below, density checking is generally hard.

Theorem 2. *The density checking problem is* NP*-complete.*

Example 2. *Consider the relational instance I in Figure 1. Suppose that we have a removal set $I_N = \{t_1, t_2, t_3, t_4, t_5, t_6\}$. Given $k = 2$, according to Formula 2, the density of remaining tuples in $I \setminus I_N$ is computed by considering k nearest neighbors $L^k_{I \setminus I_N}(t_i)$ for each tuple $t_i \in I \setminus I_N$, having*

$$\begin{aligned}
&\mathcal{D}(I \setminus I_N)\\
=&((1-0.028)+(1-0)+(1-0.032)+(1-0))+\\
&((1-0.016)+(1-0)+(1-0.024)+(1-0))+\\
&((1-0.020)+(1-0)+(1-0.024)+(1-0))+\dots\\
=&99.002.
\end{aligned}$$

Consider other possible removal sets $I'_N = \{t_0, t_2, t_3, t_4, t_5, t_6\}$ and $I''_N = \{t_0, t_1, t_3, t_4, t_5, t_6\}$. The corresponding density values are computed,

$$\begin{aligned}
&\mathcal{D}(I \setminus I'_N)\\
=&((1-0.200)+(1-0)+(1-0.224)+(1-0))+\dots\\
=&98.638,
\end{aligned}$$

and similarly

$$\begin{aligned}
&\mathcal{D}(I \setminus I''_N)\\
=&((1-0.200)+(1-0.200)+(1-0.416)+(1-0))+\dots\\
=&98.246.
\end{aligned}$$

Since the removal set I_N leads to a higher density for the remaining tuples in $I \setminus I_N$, it will be returned as errors.

III. Heuristic Solution

Recognizing the hardness in Theorem 2, in this section, we first propose the HEURISTIC algorithm in Section III-A, which always returns a minimal removal set in polynomial time. Moreover, we study a special case of clique conflicts[2] and non-violation neighbors in Section III-B, where the HEURISTIC algorithm indeed returns the optimal solution.

A. Heuristic Algorithm by Solving Conflicts

Let $I_C \subset I$ be the set of conflict tuples w.r.t. the integrity constraints Σ, having

$$I_C = \{t_i \in I | \exists t_l \in I, i \neq l, (t_i, t_l) \nvDash \Sigma\}.$$

The following proposition first narrows down the search space of the optimal solution, i.e., only need to search inside the conflict tuples I_C.

Proposition 3. *Any optimal removal set I_N^* with the maximum density always has $I_N^* \subset I_C$.*

Intuitively, in order to maximize the density of the remaining tuples, we may remove first those tuples in I_C with lower density. However, referring to the density definition in Formula 1, the density of a tuple t_i may be affected by the removal of other tuples in I_C, which are the k neighbors of t_i. To approximate, we simply ignore such updates on density, and heuristically consider the k neighbors only from $I \setminus I_C$ of non-conflict tuples, denoted by $L^k_{I \setminus I_C}(t_i)$. Let

$$\rho_i = \sum_{t_l \in L^k_{I \setminus I_C}(t_i)} \sum_{A_j \in A} (1 - d_j(t_i, t_l)), \tag{3}$$

be the approximate density of tuple t_i w.r.t. its k neighbors in $I \setminus I_C$.

The HEURISTIC Algorithm 1 iteratively removes tuples in I_C with lower density ρ_i, till all the conflicts are solved. We first sort the conflict tuples $t_i \in I_C$ in the ascending order ρ_i in Line 1. Lines 3-5 consider all the pairwise conflict tuples, and the one with lower density between two conflict tuples will be removed. Finally, Lines 6-8 guarantee the HEURISTIC algorithm returns a minimal removal set.

Algorithm 1: HEURISTIC(I,Σ)

Input: a (dirty) relational instance I, and a set of integrity constraints Σ
Output: a removal set I_N of I

```
1 sort conflict tuples t_i ∈ I_C in an ascending order of ρ_i;
2 I_N ← ∅;
3 for each t_i ∈ I_C do
4 |   if ∃t_l ∈ I_C \ I_N, (t_i, t_l) ⊭ Σ then
5 |   |   I_N ← I_N ∪ {t_i};
6 for each t_i ∈ I_N do
7 |   if ∀t_l ∈ I_C \ I_N, (t_i, t_l) ⊨ Σ then
8 |   |   I_N ← I_N \ {t_i};
9 return I_N
```

Example 3. *Consider the relational instance I in Figure 1 with the* DC *specified in Example 1. Line 1 in Algorithm 1 first initializes ρ_i for tuples $t_i \in I_C = \{t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8\}$*

[2]See the formal definition of clique conflicts in Definition 4.

and sort them in an ascending order of ρ_i. For instance, given $k = 2$, we have $\rho_0 = 3.940$ for tuple $t_0 \in I_C$ and $\rho_1 = 3.576$ for tuple $t_1 \in I_C$. During removal in Line 3, we first consider the conflict tuple t_4, whose $\rho_4 = 2.400$ is smallest. Then Line 5 selects t_4 as an error tuple by checking whether it has conflicts with the other tuples in $I_C \setminus I_N$, i.e., $\{t_0, t_1, t_2, t_3, t_5, t_6, t_7, t_8\}$. In the second round, the next smallest t_3 with $\rho_3 = 2.784$ is considered in Line 3. Similarly, t_3 having conflicts with remaining tuples $\{t_0, t_1, t_2, t_5, t_6, t_7, t_8\}$ is identified as a removal tuple in Line 5. Finally, since t_1, t_2, t_3, t_4, t_5 and t_6 have conflicts with the remaining tuple $t_0, t_7, t_8 \in I_C \setminus I_N$, Lines 6-8 do not put back any tuple in I_N to the remaining tuple set. The final result is indeed a minimal removal set.

While HEURISTIC algorithm considers the repairing over pairwise conflict tuples, we show below that the returned removal set is always minimal.

Proposition 4. *The* HEURISTIC *algorithm returns a minimal removal set with time complexity $O(cnm + cn \log k)$, where $c = |I_C|$, $n = |I|$, $m = |A|$ and k is the number of considered neighbors.*

B. Special Case Performance

We show below (in Proposition 5) that the HEURISTIC algorithm indeed returns the optimal solution, in a special case of clique conflicts and non-violation neighbors. Let us first formally define clique conflicts.

Definition 4 (Clique Conflicts)**.** *We call $I'_C \subseteq I_C$* clique conflicts*, if $\forall t_i, t_l \in I'_C$, $i \neq l$, it always has $(t_i, t_l) \nvDash \Sigma$.*

That is, tuples in a clique I'_C conflict with each other. Such clique conflicts are prevalent, e.g., an FD (Load $\rightarrow$ Count) always leads to clique conflicts. Tuples sharing the same **y** value but with different **x** form a clique I'_C of conflicts.

Proposition 5. *If we have $L^k_I(t_i) = L^k_{I \setminus I_C}(t_i)$ for each tuple $t_i \in I$, and all the tuples in I_C are involved in disjoint clique conflicts,* HEURISTIC *algorithm returns an optimal solution.*

The condition $L^k_I(t_i) = L^k_{I \setminus I_C}(t_i)$ states that each tuple $t_i \in I$ only has neighbors not in violation, i.e., not in I_C. The efficient solution in such a case is not surprising, since the density of any tuple is no longer affected by (removing) any tuples in I_C.

Example 4. *Consider again the relational instance I in Figure 1 with the* DC *specified in Example 1, which is indeed an* FD *Load $\rightarrow$ Count. The constraint detects the conflicts among tuples in I, having $I_C = \{t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8\}$ as illustrated in Example 3. It is notable that any tuple $t_i \in I'_C = \{t_0, t_1, t_2, t_3, t_4\} \subset I_C$ has conflicts with all the other tuples in $I'_C \setminus \{t_i\}$. Similar clique conflicts exist in $\{t_5, t_7\} \subset I_C$ and $\{t_6, t_8\} \subset I_C$.*

IV. EXPERIMENTS

In this section, we compare our approach HEURISTIC against the state-of-the-art methods, in terms of data cleaning performance, on a variety of real datasets with real-world and synthetic errors.

TABLE I
DATASETS USED IN EXPERIMENTS

Name	Size	#Attr	#FD	#DC	Error
Restaurant	864	4	2	7	artificial
Tax	200k	15	5	5	artificial
WindTurbine	31k	7	3	3	real/labeled
Iris	150	4	4	6	KEEL/labeled

A. Experimental Setup

1) Datasets: Table I lists the datasets used in the experiments. Datasets Restaurant[3] and Tax[4] are assumed to be originally clean. Artificial errors are injected for evaluation (see details in Section IV-A2). WindTurbine dataset is collected by our partner company, where real-world errors are embedded and labeled. Similarly, KEEL[5] provides different versions of dataset Iris with various errors embedded.

2) Error Generation: Since errors are not embedded in Restaurant and Tax, to evaluate the data cleaning performance, we use BART [1], a state-of-the-art system for benchmarking data cleaning algorithms, to inject synthetic errors. Given a dataset, FDs are first discovered by TANE [8]. BART then generates FD-detectable errors.

3) Evaluation Methodology: To show the effectiveness of S-repair, we measure the accuracy of removing errors by Precision (Prec.), Recall (Rec.) and F1-score (F1). Referring to the ground truth of each dataset, we determine tp - the correctly removed tuples of errors, fp - clean tuples that are falsely determined as errors, and fn - error tuples that are not marked as such. The evaluation metrics are computed as

$$Rec. = \frac{tp}{tp + fn}, \ Prec. = \frac{tp}{tp + fp}, \ F1 = \frac{2 \times Prec. \times Rec.}{Prec. + Rec.}.$$

For the data cleaning methods which identify the specific error cells, e.g., Holistic [6], HC [16] and Raha [12], we count the corresponding tuples as errors. If multiple cells in one tuple are indicated having error data, this tuple is only counted once.

4) Competing Methods: We compare our proposed HEURISTIC algorithm with the following approaches. (1) **Constraint-based MS** [11]. It returns the minimum S-repair under the guidance of FDs, following the minimum change principle. (2) **Constraint-based Holistic** [6]. It performs U-repair under denial constraints (DC). The U-repair model updates the values of specific cells in order to eliminate violations to the given integrity constraints. (3) **Machine Learning System HC** [16]. HC is a state-of-the-art machine learning based data cleaning system with integrity constraints, external data and statistical analysis. (4) **Machine Learning System Raha** [12]. Raha is a machine learning based error

[3]http://www.cs.utexas.edu/users/ml/riddle/data.html
[4]http://db.unibas.it/projects/bart/
[5]https://sci2s.ugr.es/keel/attributeNoise.php

TABLE II
PRECISION, RECALL AND F1-SCORE OF S-REPAIR OVER VARIOUS DATASETS

Approach	WindTurbine (1,759 error tuples)			Restaurant (34 error tuples)			Tax (20,000 error tuples)		
	$Precision$	$Recall$	$F1-score$	$Precision$	$Recall$	$F1-score$	$Precision$	$Recall$	$F1-score$
DORC	0.877	0.405	0.544	0.038	0.909	0.073	0.233	0.064	0.101
LS	0.647	0.809	0.719	0.032	0.318	0.058	0.466	0.018	0.036
LOF	0.843	0.594	0.697	0.031	0.727	0.059	0.316	0.461	0.375
dBoost	0.568	0.521	0.544	0.047	0.147	0.071	0.414	0.049	0.087
MS	0.505	0.215	0.302	0.706	0.545	0.615	0.723	0.758	0.740
Holistic	0.584	0.239	0.340	0.722	0.591	0.650	0.726	0.760	0.743
HC	0.651	0.106	0.182	n/a	n/a	n/a	0.742	0.015	0.029
Raha	0.990	0.663	0.794	0.132	0.151	0.141	0.174	0.253	0.206
HEURISTIC	0.999	0.983	**0.991**	0.857	0.706	**0.774**	0.738	0.768	**0.753**

detection system, which needs the existing error detection algorithms, e.g., dBoost [13], to generate sets of data errors, and errors labeled for training model. (5) **Clustering-based DORC** [17]. It divides all the tuples into three types, referring to the clustering results. All the tuples, which are neither core points nor border points, are identified as errors. (6) **Clustering-based LS** [7]. It performs k-means clustering with outliers. Again, the identified outliers are considered as errors. (7) **Outlier-based LOF** [4]. It defines a score for each tuple t_i based on the density of those tuples in t_i's neighborhood, in order to evaluate whether this tuple t_i should be classified as an outlier. Outliers are ranked based on their LOF scores and treated as errors. (8) **Outlier-based dBoost** [13]. It is a widely used outlier detection tool that integrates several algorithms, including histogram, gaussian and multivariate gaussian mixture models.

B. Cleaning Performance

Table II reports the S-repair accuracy of removing error tuples. DORC, LS, LOF and dBoost, relying on statistical information to identify errors, usually cannot achieve a good performance compared to the constraint-based techniques, such as MS or Holistic. The reason is that they often fail to detect non-outlying errors but excessively identify all the outlier tuples as errors.

Raha combines the efforts of existing error detection algorithms, e.g., dBoost, and the manual labels. It thus achieves a high accuracy in the WindTurbine dataset where extreme outlier values present. Unfortunately, although it can conduct label propagation with only 20 labeled tuples, these manually labeled tuples cannot always cover all the error patterns, especially for a large dataset such as Tax.

HC can also detect the conflict tuples with the help of denial constraints. It is difficult to further distinguish which one of the conflict tuples indeed has errors. Without strong statistical signals supporting, e.g., in Restaurant[6], the corresponding results are not available (denoted by n/a). Our HEURISTIC algorithm performs significantly better than the other approaches, e.g., in WindTurbine and Restaurant. The constraint-based approaches MS and Holistic achieve comparable performances with HEURISTIC over Tax dataset, since there are duplicate values where the constraints apply to detect violations.

[6]The Restaurant dataset is too small with heterogeneous values.

TABLE III
F1-SCORE OF S-REPAIR UNDER VARIOUS ERROR RATES OVER IRIS DATA

Approach	5%	10%	15%	20%
DORC	0.473	0.439	0.428	0.419
LS	0.476	0.465	0.422	0.405
LOF	0.403	0.377	0.316	0.313
dBoost	0.224	0.193	0.172	0.166
MS	0.291	0.245	0.236	0.055
Holistic	0.483	0.478	0.428	0.423
HC	0.134	0.130	0.106	0.098
Raha	0.358	0.338	0.319	0.300
HEURISTIC	**0.547**	**0.521**	**0.517**	**0.513**

Table III reports the error removal accuracy over the Iris dataset under various error rates, where errors with various rates are labeled by KEEL in advance. First, it is not surprising that with the increase of error rate, the repair performance of each method drops. The error repair accuracy in Table III is generally proportional to the results over other datasets in Table II. Again, our proposed HEURISTIC method is more effective in identifying errors than the other data cleaning approaches, with various error rates in the dataset.

Table IV presents the median runtime of different methods. As expected, the iterative approaches such as DORC and Holistic are slower than non-iterative ones. Our HEURISTIC algorithm is more efficient than most of competing methods, which is not surprising referring to the complexity analysis in Proposition 4.

V. RELATED WORK

Constraint-based cleaning. Constraint-based methods [2], [3], [6], [11] use integrity constraints to tackle errors. The

TABLE IV
RUNTIME (IN SECONDS) OF REPAIRING (the tiny time costs over the very small Restaurant and Iris datasets are omitted)

Approach	WindTurbine	Tax
DORC	4289.39	417609.18
LS	3337.94	150659.43
LOF	13048.33	279181.31
dBoost	22.34	59.29
MS	1546.02	99142.21
Holistic	3515.70	157432.03
HC	1410.21	83256.02
Raha	286.36	16393.07
HEURISTIC	241.35	23976.83

S-repair model [11] removes a minimal set of tuples such that the remaining tuples satisfy the constraints, in order to avoid excessive removal and information loss. The U-repair model Holistic [6] conducts the iterative repairing with the help of denial constraints to construct repair context. The minimum change criteria is often employed to avoid excessive change and information loss [2]. Unfortunately, as illustrated in Section I, without further knowledge, it is often difficult for constraint-based approaches to determine a proper removal between conflict tuples.

Labeling-based cleaning. Labeling based methods [12] often cast the error detection task to a binary classification problem. Raha [12] first generates a wide range of algorithm configurations and encodes the output into feature vectors for all data cells. Then the method combines clustering, label propagation, and classification per attribute, and learns to predict the labels of cells. HC [16] employs not only the statistical learning but also the integrity constraints and probabilistic inference to repair error data. However, as shown in Section IV, it still cannot perform well without strong statistics supporting, especially for the datasets with heterogeneous values.

Outlier-based cleaning. Outlier detection methods [4], [13] identify the tuples containing values significantly different from the others. For example, dBoost [13] relies on several classical outlier detection approaches, such as histogram, gaussian modeling and multivariate gaussian mixture models, to analyze the data distributions for error detection. LOF [4] utilizes the k-nearest neighbors of each tuple to measure its density-based outlier factor. This interesting idea motivates us to define directly the density by considering the neighbors in our proposal. However, outliers are not always errors, while errors may not be outlying.

Clustering-based cleaning. DORC [17] treats outliers as errors and moves them into clusters as repairs. While core points and border points are clustered, all the other outlying points will be repaired. Similarly, LS [7] performs k-means clustering with outliers. It iteratively checks if swapping one of the current centers with a non-center and any further local swap with the additional removal of outliers could improve the clustering performance. If it is the case, the swap and removal will be performed. The major issue of DORC and LS is to excessively repair all outliers but fail to identify non-outlying errors.

VI. CONCLUSION

In this paper, we propose to find the S-repair w.r.t. integrity constraints that maximizes density. To solve the density maximization problem under integrity constraints, we analyze the hardness of the problem in Theorem 2 and devise a heuristic algorithm that returns the optimal solution in certain cases in Proposition 5. Extensive experiments over datasets from industry with real-world errors demonstrate that our proposal improves the data cleaning accuracy compared to the existing techniques.

Acknowledgement: This work is supported in part by the National Key Research and Development Plan (2019YFB1705301) and the National Natural Science Foundation of China (62072265, 61572272, 71690231).

REFERENCES

[1] P. C. Arocena, B. Glavic, G. Mecca, R. J. Miller, P. Papotti, and D. Santoro. Messing up with BART: error generation for evaluating data-cleaning algorithms. *PVLDB*, 9(2):36–47, 2015.
[2] G. Beskales, I. F. Ilyas, and L. Golab. Sampling the repairs of functional dependency violations under hard constraints. *PVLDB*, 3(1):197–207, 2010.
[3] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *ICDE*, pages 746–755, 2007.
[4] M. M. Breunig, H. Kriegel, R. T. Ng, and J. Sander. LOF: identifying density-based local outliers. In *SIGMOD*, pages 93–104, 2000.
[5] J. Chomicki and J. Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput.*, 197(1-2):90–121, 2005.
[6] X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. In *ICDE*, pages 458–469, 2013.
[7] S. Gupta, R. Kumar, K. Lu, B. Moseley, and S. Vassilvitskii. Local search methods for k-means with outliers. *PVLDB*, 10(7):757–768, 2017.
[8] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. TANE: an efficient algorithm for discovering functional and approximate dependencies. *Comput. J.*, 42(2):100–111, 1999.
[9] Y. Ji, Y. Chai, X. Zhou, L. Ren, and Y. Qin. Smart intra-query fault tolerance for massive parallel processing databases. *Data Sci. Eng.*, 5(1):65–79, 2020.
[10] N. Koudas, A. Saha, D. Srivastava, and S. Venkatasubramanian. Metric functional dependencies. In *ICDE*, pages 1275–1278, 2009.
[11] E. Livshits, B. Kimelfeld, and S. Roy. Computing optimal repairs for functional dependencies. In *PODS*, pages 225–237, 2018.
[12] M. Mahdavi, Z. Abedjan, R. C. Fernandez, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang. Raha: A configuration-free error detection system. In *SIGMOD*, pages 865–882, 2019.
[13] Z. Mariet, R. Harding, S. Madden, et al. Outlier detection in heterogeneous datasets using automatic tuple expansion. 2016.
[14] G. Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, 2001.
[15] A. A. Qahtan, A. K. Elmagarmid, M. Ouzzani, and N. Tang. FAHES: detecting disguised missing values. In *ICDE*, pages 1609–1612, 2018.
[16] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré. Holoclean: Holistic data repairs with probabilistic inference. *PVLDB*, 10(11):1190–1201, 2017.
[17] S. Song, C. Li, and X. Zhang. Turn waste into wealth: On simultaneous clustering and cleaning over dirty data. In *SIGKDD*, pages 1115–1124, 2015.
[18] S. Song, A. Zhang, J. Wang, and P. S. Yu. SCREEN: stream data cleaning under speed constraints. In *SIGMOD*, pages 827–841, 2015.
[19] Y. Sun, S. Song, C. Wang, and J. Wang. Swapping repair for misplaced attribute values. In *ICDE*, pages 721–732, 2020.
[20] J. Wang and N. Tang. Towards dependable data repairing with fixing rules. In *SIGMOD*, pages 457–468, 2014.