Swapping Repair for Misplaced Attribute Values

Yu Sun, Shaoxu Song, Chen Wang, Jianmin Wang
BNRist, School of Software, Tsinghua University, Beijing, China
{sy17, sxsong, wang_chen, jimwang}@tsinghua.edu.cn

***Abstract*—Misplaced data in a tuple are prevalent, e.g., a value "Passport" is misplaced in the passenger-name attribute, which should belong to the travel-document attribute instead. While repairing *in-attribute* errors have been widely studied, i.e., to repair the error by other values in the attribute domain, misplacement errors are surprisingly untouched, where the true value is simply misplaced in some other attribute of the same tuple. For instance, the true passenger-name is indeed misplaced in the travel-document attribute of the record. In this sense, we need a novel swapping repair model (to swap the misplaced passenger-name and travel-document values "Passport" and "John Adam" in the same tuple). Determining a proper swapping repair, however, is non-trivial. The minimum change criterion, evaluating the distance between the swapping repaired values, is obviously meaningless, since they are from different attribute domains. Intuitively, one may examine whether the swapped value ("John Adam") is similar to other values in the corresponding attribute domain (passenger-name). In a holistic view of all (swapped) attributes, we propose to evaluate the likelihood of a swapping repaired tuple by studying its distances (similarity) to neighbors. The rationale of distance likelihood refers to the Poisson process of nearest neighbor appearance. The optimum repair problem is to find a swapping repair with the maximum likelihood on distances. Experiments over datasets with real-world misplaced attribute values demonstrate the effectiveness of our proposal in repairing misplacement.**

I. Introduction

Misplaced attribute values are commonly observed, e.g., owing to filling mistakes in Web forms, mis-plugging cables of sensors, or missing values of sensors during transfer. Downstream applications built upon the misplaced data are obviously untrusted. Cleaning such misplacement is demanded.

A. Sources of Misplaced Attribute Errors

Misplaced attribute values could be introduced generally in all ETL steps, ranging from data production to consumption.

1) Entry Error: Misplaced attribute values may occur when data are entered into the database. For instance, a value "Passport", which should be input in the attribute travel-document, is mistakenly filled in attribute passenger-name. Similar examples are also observed in medical data [31] and procurement data [7]. Even in the IoT scenarios, since workers may occasionally mis-plug the cables of sensors during equipment maintenance, misplacement occurs frequently (200 out of 5.2k tuples in the real Turbine dataset used in the experiments as introduced in Section VII-A1).

2) Extraction Error: When integrating data from various sources, information extraction and conversion frequently introduce misplaced attribute values. For instance, CSV files from different sources often have various delimiters and abuse is difficult to avoid [40]. It needs great effort to manually correct them step by step. Similarly, misplaced errors may also occur when performing Optical Character Recognition (OCR) on handwritten forms [23]. Inaccurate rectangle labeling obviously leads to misplaced errors.

3) Shift Error: In the IoT scenario, data are often transferred in the form of comma separated records, and parsed as database tuples when received (see Example 1 below). If an attribute value is missing owing to sensor failure or replacement, the values next to the missing/replaced one are shifted to wrong places, a.k.a. shift error. Similar examples are also observed in medical data [31] and government data [20]. In the real FEC dataset [20], since commas in values are mistakenly interpreted as separators, a number of 3.3k tuples are observed with misplacement out of 50k (see details in Section VII-A1 of experiment datasets).

B. Challenges

1) In-attribute errors vs. Misplaced-attribute errors: While misplaced attribute values are commonly observed in practice, they are surprisingly untouched in research studies. To the best of our knowledge, existing data repairing approaches [15], [25], [33], [38], [39], [41] (see Section VIII-A for a short survey) often focus on *in-attribute* errors, and thus repair the error by other values in the attribute domain. For example, use some value of passenger-name to repair the value "Passport". Or similarly, use some other voltage value to repair $t_0[\text{voltage}] = 13.7$ in Figure 1 below.

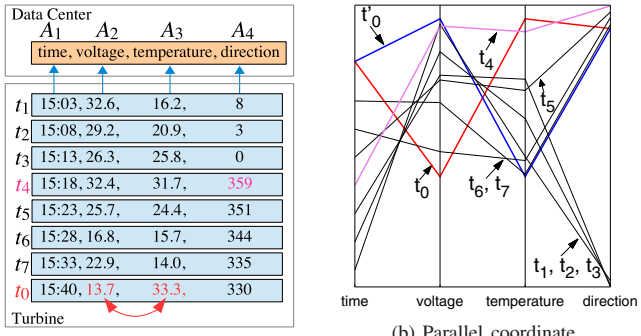
For the *misplaced-attribute errors*, however, the true values are indeed in the tuple but in wrong places. That is, we can significantly narrow down the candidates for repairing misplaced attribute values. Obviously, **swapping** the data in the tuple is preferred to repair the misplaced attribute values. For instance, the value "Passport" in the misplaced attribute passenger-name should be swapped with the value "John Adam" in attribute travel-document in the same tuple. [1]

2) Minimum change vs. Maximum likelihood: To evaluate whether misplacement is correctly repaired, the minimum change criteria [9], widely considered for repairing in-attribute errors, does not help. Measuring the swapping repaired values "Passport" and "John Adam" is meaningless, since they are from the domains of different attributes.

Intuitively, we may study the likelihood of a tuple by investigating how similar/distant its values are to the values in other tuples. The rationale of the distance likelihood refers

[1]Swapping may apply to multiple misplaced attributes (see Definition 1).

2375-026X/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDE48307.2020.00068



(a) Comma separated records

(b) Parallel coordinate

Fig. 1. Sensor readings from wind turbine, where misplaced attribute values 13.7, 33.3 occur in t_0, and should be repaired by swapping voltage and temperature values as in t'_0

to the Poisson process of nearest neighbor appearance, where the neighbors of a given tuple t_i are the tuples t_j having the minimum tuple distances defined in Formula 1 to t_i (see Section II-B for details). In this sense, t_0 with misplacement in Figure 1(b) in Example 1 could be identified, since its (time, voltage, temperature, direction) value combination is distant from other tuples in the dataset, i.e., low likelihood.

C. Our Proposal

We notice that tuples with misplaced values are often distant from other tuples (see motivation Example 1 below). Intuitively, one may apply the multivariate outlier detection techniques, e.g., distance-based [19], to detect tuples deviating markedly from others. However, directly applying the multivariate outlier detection may return false-positives, i.e., true outliers without misplaced values. In this sense, we propose to further investigate the detected tuple by swapping its attribute values. If the tuple after swapping has closer neighbors, e.g., becomes inliers, it is more confident to assert misplacement and apply the swapping as repairs.

Informally, the swapping repair problem is thus: for each tuple (say t_0) in relation instance r, see whether there exists a tuple t'_0 by swapping the attribute values in t_0, such that t'_0 is more likely (more similar to the neighbors in r) than t_0; if yes, we return the most likely swapping repair having the least distances to neighbors in r.

Example 1. *Consider a collection of sensor readings in wind turbine in Figure 1, where the sensor data are transferred from devices to data center through wireless communication networks, in a form of comma separated records. Misplaced values are frequently observed for various reasons. For instance, shifting errors occur when the power supply of some sensor is interrupted or some packages of a tuple are lost in data transmission, as discussed in Section I-A. Moreover, during equipment maintenance, workers may occasionally mis-plug the cables of sensors for monitoring temperature and voltage, as shown in Figure 1(a). In addition, sensor may be reordered in the upgrade of wind turbine. While the data collection protocol is updated immediately in the device, the modification of schema definition in the data center is delayed. Misplaced values are observed in a short period of schema updating.*

As shown in Figure 1(a), the voltage and temperature values in the latest record (denoted by t_0) are misplaced, which are very different to those in the nearby tuples t_5, t_6, t_7. A false alarm will be triggered, owing to the sudden "changes".

As plotted in the parallel coordinate in Figure 1(b), by swapping the voltage and temperature values of t_0 with true misplacement, it will accord perfectly with other tuples having similar timestamps, e.g., t_5, t_6, t_7. In this sense, we propose to evaluate the likelihood of repaired tuple by whether having values (on all attributes time, voltage, temperature, direction) similar to other tuples.

The existing in-attribute repair, e.g., constraint-based [38], uses the value in the same attribute to repair the misplaced error in t_0, i.e., t'_0[voltage] =22.9 and t'_0[temperature] =14.0. As shown, the repair is not as accurate as the swapping repair, where 33.3 and 13.7 are indeed the true values of voltage and temperature, respectively, but simply misplaced.

Attribute direction reports the direction of a wind turbine measured in degrees, with domain values ranging from 0 to 359. As shown in Figure 1(a), data entry simply changes its pattern starting at t_4, from values around 0 to values near 359. We have t_4[direction] = 359, which is distant from the previous direction values in tuples t_1 to t_3. However, swapping repair will not be performed on t_4, since by swapping the value of direction with any other value in the tuple, it is still distinct from the nearby tuples such as t_1, t_2, t_3. That is, the likelihood of the swapped tuple does not increase.

D. Contributions

Our major contributions in this study are as follows.

We formalize the optimum swapping repair problem in Section III. A pipeline is further presented to jointly repair both misplacement and in-attribute errors.

We show that, if considering all the n tuples in r as neighbors in evaluating a repair, the optimum repair problem is polynomial time solvable (Proposition 1) in Section IV. This special case is not only theoretically interesting, but also used to efficiently solve (Algorithm 1) or approximate (Algorithm 2) the problem with any number κ of neighbors.

We present that, if considering a fixed number κ of neighbors, the optimum repair problem can be solved in polynomial time (Proposition 2) in Section V. Bounds of neighbor distances are devised (Proposition 3), which enable pruning for efficient repairing.

We develop an approximation algorithm, by considering a fixed set of neighbors, in Section VI.

We conduct an extensive evaluation in Section VII, on datasets with real-world misplaced attribute values. The experiments demonstrate that our proposal complements the existing data repairing by effectively handling misplacement.

Table I lists the frequently used notations.

TABLE I
NOTATIONS

Symbol	Description
R	relation schema, with m attributes
r	relation instance over R, with n tuples
t_0	tuple in r to detect and repair misplaced attribute values
κ	number of considered nearest neighbors
$N_r^\kappa(t)$	κ-nearest-neighbors (κ-NN) of t from r, for simplicity $N(t)$
$\mathbf{x}$	swapping repair of t_0, having repaired tuple $t_0' = \mathbf{x}t_0$
$\Theta_r^\kappa(\mathbf{x})$	distance cost of swapping repair $\mathbf{x}$ over tuples r with κ-NN, for simplicity $\Theta(\mathbf{x})$
T	potential set of κ-NN, $T \subseteq r$
SRAN	Swapping Repair with All Neighbors, in Section IV-B
SRKN	Swapping Repair with κ Neighbors, Algorithm 1
SRFN	Swapping Repair with Fixed Neighbors, Algorithm 2

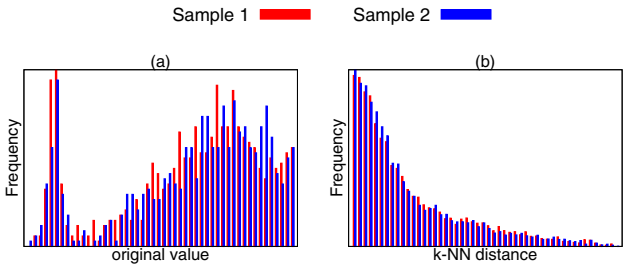


Fig. 2. Inconsistent value distribution vs. consistent distance distribution of two different samples from the same dataset

II. DISTANCE-BASED LIKELIHOOD EVALUATION

In this section, we first illustrate the deficiencies of evaluating the likelihood of a tuple w.r.t. value distribution. It leads to the intuition of considering the distances of the tuple to its neighbors. We use the likelihood on distance to evaluate a repair in the following Section III.

A. Why Not Using Value Distribution

To evaluate the likelihood of a tuple, a natural idea is to investigate how likely each value in the tuple belongs to the claimed attribute. By studying the joint distribution of values in multiple attributes, the likelihood of the tuple is calculated [25]. A tuple with misplaced attribute values is outlying in the value distribution, and thus has a low likelihood.

Unfortunately, as mentioned in the Introduction, owing to data sparsity and heterogeneity, the value distribution could be unreliable. For instance, in Figure 2(a), we observe the value distributions of two different samples (i.e., Sample 1 and Sample 2) with 4k tuples randomly sampled from the Magic dataset [4], respectively. As shown, the value distribution of Sample 1 (red) is largely different from that of Sample 2 (blue), which are indeed two samples of the same dataset. Some value in Sample 1 even does not appear in Sample 2. The likelihood of a value computed based on these inconsistent value distributions would obviously be inaccurate.

Intuitively, instead of directly evaluating how likely a tuple contains attribute values appearing exactly in the value distribution, we may alternatively check whether the tuple has values similar to other tuples, in order to be tolerant to data sparsity and heterogeneity. If the tuple is distant from others, (either misplaced-attribute or in-attribute) errors are likely to occur. By coincidence, the tuple becomes similar to some neighbors after swapping certain attribute values. We would assure the misplacement and repair, such as t_0 in Figure 1 in Example 1.

Therefore, in this study, we propose to learn the distribution of distances between a tuple and its neighbors. As illustrated in Figure 2(b), more consistent distance distributions are observed in two different samples. The consistent distance distributions (in contrast to the inconsistent value distributions) are not surprising referring to the Poisson process of nearest neighbor appearance [28] (see explanation in Section II-B). The likelihood computed based on the consistent distance distribution would be more reliable.

B. Likelihood on Distances to Neighbors

Consider a relation instance $r = \{t_1, \ldots, t_n\}$ over schema $R = (A_1, \ldots, A_m)$. For each attribute $A \in R$, let Δ be any distance metric having $0 \leq \Delta(t_i[A], t_j[A]) \leq 1$, where $t_i[A]$ and $t_j[A]$ are values from the domain $\mathsf{dom}(A)$ of attribute A. For instance, we may use edit distance [26] or pre-trained embedding technique [30] with normalization [22] for string values, or the normalization distance [10] for numerical values. By considering L^1 norm the Manhattan distance [16] as the distance function on all attributes in R, we obtain the tuple distance $\Delta(t_i[R], t_j[R])$, or simply $\Delta(t_i, t_j)$,

$$\Delta(t_i[R], t_j[R]) = \sum_{A \in R} \Delta(t_i[A], t_j[A]). \tag{1}$$

Let $N_r^\kappa(t_i)$ be the κ-nearest-neighbors (κ-NN) of t_i from r, denoted by $N(t_i)$ for simplicity. As illustrated in [28], the random generation of data from a particular underlying density function induces a density function of distance to the nearest neighbors. In relational data, it means that the appearance of nearest neighbors $N_r^\kappa(t_i)$ of a tuple t_i can be approximated with Poisson process [28]. That is, the distances between a tuple t_i and its nearest neighbors $t_j \in N_r^\kappa(t_i)$ follow an exponential distribution, $\Delta(t_i, t_j) \sim Exp(\lambda)$, and we can write the probability distribution of the distances as

$$f(\Delta(t_i, t_j)) = \lambda \exp^{-\lambda\Delta(t_i, t_j)}, \tag{2}$$

where $\lambda > 0$ is the parameter of the distribution, often called the rate parameter.

The (log) likelihood of a tuple t_0 in r is thus computed by aggregating the probabilities $f(\Delta(t_0, t_l))$ on the distances to its neighbors t_l

$$\begin{aligned}\hat{\ell}(t_0) &= \sum_{t_l \in N_r^\kappa(t_0)} \log f(\Delta(t_0, t_l)) \\ &= \kappa \log(\lambda) - \lambda \sum_{t_l \in N_r^\kappa(t_0)} \Delta(t_0, t_l).\end{aligned} \tag{3}$$

Example 2. *Consider the relation* $r = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_0\}$ *with schema* $R = \{$*time, voltage, temperature, direction*$\}$ *in Figure 1. The normalized distance between* t_0 *and* t_7 *on* *voltage* *is* $\Delta(t_0[\textit{voltage}], t_7[\textit{voltage}]) = 1 - \exp^{-|13.7-22.9|} = 1$. *By aggregating the distances on all attributes in* R *referring to Formula 1, we have* $\Delta(t_0[R], t_7[R]) = 3.061$. *Given* $\kappa = 3$, *the* κ-NN *of* t_0 *is* $N(t_0) = \{t_0, t_4, t_7\}$, *where* $\Delta(t_0[R], t_0[R]) = 0$, $\Delta(t_0[R], t_4[R]) = 2.995$, *and* $\Delta(t_0[R], t_7[R]) = 3.061$. *Suppose that* $\lambda = 2$ *of the exponential distribution* $f(\Delta(t_i, t_j))$ *in Formula 2. The likelihood of distances between* t_0 *and* $N(t_0)$ *is* $\hat{\ell}(t_0) = 3 * \log(2) - 2 * (0 + 2.995 + 3.061) = -9.112$.

III. Problem Statement

As introduced in Section I-B1, misplacement can be repaired by simply swapping attribute values in a tuple. We first formally define the swapping repair in Definition 1. Recognizing the deficiencies of existing criteria in Sections I-B2 and II-A, we introduce the distance likelihood in Formula 3 to evaluate a swapping repaired tuple. The optimum repair problem is thus to find the swapping repair with the maximum likelihood.

A. Swapping Repair

We use a $m \times m$ matrix $\mathbf{x}$ to denote the swapping repair of attribute values in a tuple t_0 with m attributes.

Definition 1. Swapping repair *of tuple* t_0 *is a matrix* $\mathbf{x}$ *having*

$$\begin{aligned} &\sum_{i=1}^{m} x_{ij} = 1, && 1 \le j \le m \\ &\sum_{j=1}^{m} x_{ij} = 1, && 1 \le i \le m \\ &x_{ij} \in \{0, 1\}, && 1 \le i \le m, 1 \le j \le m \end{aligned}$$

The swapping repaired tuple

$$t_0' = \mathbf{x} t_0$$

has $t_0'[A_i] = t_0[A_j]$ *if* $x_{ij} = 1$.

For a reliable [41] or certain [15] attribute A_i that should not be modified in repairing, we may simply set $x_{ii} = 1$.

Example 3. *Consider tuple* t_0 *in Figure 1. Suppose that we have a swapping repair* $\mathbf{x}$ *for* t_0 *as follows,*

$$\mathbf{x} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

where $\sum_{i=1}^{4} x_{ij} = 1$, $1 \le j \le 4$, *and* $\sum_{j=1}^{4} x_{ij} = 1$, $1 \le i \le 4$. *Each* $x_{ij} = 1$, *for instance* $x_{32} = 1$, *denotes that* $t_0'[$*temperature*$] = t_0[$*voltage*$]$, *where* $A_2 =$ *voltage*, $A_3 =$ *temperature*. *The corresponding repaired tuple is thus*

$$t_0' = \mathbf{x} t_0 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 15{:}40 \\ 13.7 \\ 33.3 \\ 330 \end{pmatrix} = \begin{pmatrix} 15{:}40 \\ 33.3 \\ 13.7 \\ 330 \end{pmatrix}.$$

B. Repairing Problem

For any swapping repair $\mathbf{x}$, the corresponding repaired tuple $t_0' = \mathbf{x} t_0$ has likelihood $\hat{\ell}(t_0')$. It is desired to find the optimal swapping repair $\mathbf{x}$ with the maximum likelihood $\hat{\ell}(t_0')$.

Referring to Formula 3, to maximize the likelihood $\hat{\ell}(t_0')$, it is equivalent to minimize the distances between t_0' and its κ-NN in r, i.e.,

$$\begin{aligned} \sum_{t_l \in N_r^{\kappa}(t_0')} \Delta(t_0', t_l) &= \sum_{t_l \in N_r^{\kappa}(t_0')} \sum_{A_i \in R} \Delta(t_0'[A_i], t_l[A_i]) \\ &= \sum_{t_l \in N_r^{\kappa}(t_0')} \sum_{A_i \in R} \sum_{A_j \in R} x_{ij} \cdot \Delta(t_0[A_j], t_l[A_i]). \end{aligned} \tag{4}$$

Thereby, we define the distance cost of swapping repair $\mathbf{x}$ over the κ-NN of the repaired tuple $t_0' = \mathbf{x} t_0$.

Definition 2. *The* κ-NN *distance cost of a swapping repair* $\mathbf{x}$ *is*

$$\Theta_r^{\kappa}(\mathbf{x}) = \Theta(\mathbf{x}) = \sum_{t_l \in N_r^{\kappa}(\mathbf{x} t_0)} \sum_{A_i \in R} \sum_{A_j \in R} x_{ij} \cdot \Delta(t_0[A_j], t_l[A_i]). \tag{5}$$

To find the repair $\mathbf{x}$ with the maximum likelihood $\hat{\ell}(\mathbf{x} t_0)$, it is equivalent to minimize the κ-NN distance cost $\Theta(\mathbf{x})$.

Problem 1. *For a tuple* t_0 *in relation* r *over schema* R, *the* OPTIMUM REPAIR *problem is to find a* swapping repair $\mathbf{x}$ *such that the* κ-NN *distance cost* $\Theta(\mathbf{x})$ *is minimized.*

The corresponding decision problem is thus to determine whether there is swapping repaired tuple with likelihood higher than the input tuple t_0.

Problem 2. *For a tuple* t_0 *in relation* r *over schema* R, *the* REPAIR CHECKING *problem is to determine whether exists a* swapping repair $\mathbf{x}$ *with* κ-NN *distance cost* $\Theta(\mathbf{x}) \le \Theta(\mathrm{I})$, *where* I *is an identity matrix (unit matrix) of size* m.

Here, $\Theta(\mathrm{I})$ denotes the κ-NN distance cost of the input tuple $t_0 = \mathrm{I} t_0$ without swapping.

Example 4 (Example 3 continued)**.** *Given* $\kappa = 3$, *the* κ-NN *of the swapping repaired tuple* t_0' *in Example 3 is* $N(t_0') = \{t_0, t_1, t_7\}$. *According to Formula 5, the* κ-NN *distance cost is* $\Theta(\mathbf{x}) = (1*0+1*1+1*1+1*0) + (1*0.309+1*0.503+1*0.918+1*1) + (1*0.067+1*1+1*0.259+1*0.993) = 7.049$.

Consider another swapping repair for t_0 *as follows,*

$$\mathbf{x}'' = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}.$$

That is, swapping the *voltage* *and* *direction* *attribute values. It has* $N(t_0'') = \{t_0, t_4, t_7\}$ *with* κ-NN *distance cost* $\Theta(\mathbf{x}'') = (1*0+1*1+1*0+1*1+1*0) + (1*0.197+1*1+1*0.798+1*1) + (1*0.067+1*1+1*1+1*1) = 8.062$.

As the first swapping repair $\mathbf{x}$ *in Example 3 has lower distance cost, it leads to the repaired tuple* $t_0' = (15{:}40, 33.3, 13.7, 330)$.

C. Joint Repair Pipeline

In addition to the studied misplacement errors, there are a wide range of errors that happen in real datasets, as discussed in [7]. For instance, let t_8 = (15:45, 28.5, 0, 10.1) be another tuple appearing after t_0 in Figure 1. The value 0 is not only dirty (a default direction value returned owing to device disturbance) but also misplaced in the temperature attribute. While this study focuses on repairing misplacement errors, we further consider joint repair to address other co-existing errors.

When addressing various kinds of errors, as studied in [12], [15], the order of repair steps affects the final results. It is worth noting that applying first the in-attribute repair would destroy the original ground truth of misplaced values, e.g., modifying the true temperature value 10.1 misplaced in direction. In this sense, the swapping repair is suggested to be performed before handling other types of errors. Moreover, as discussed in [13], since the in-attribute error repair can help resolving conflicts and duplications, the imputation for incomplete values can be enriched and improved. Therefore, the missing data imputation could be finally performed after swapping and in-attribute repair. (See experiments on varying the order of repair steps in Figure 11 in Section VII-C.)

The dirty values may lead to the corresponding distances corrupted. While the dirty values have negative effects, our swapping repair model is compatible with existing data cleaning techniques to handle such a case. The rationale is that humans or machines always try to avoid mistakes in practice [24]. In this sense, we argue that dirty values are often not significantly far from the corresponding truths as well as some other values in the same attribute. For instance, a default direction value 0 is returned rather than a random irrational value 65536, when the device is disturbed. Although the computed distances will also be dirty, they could be close, e.g., the default value 0 is close to other direction values. A correct swapping repair would lead to closer distances, while an incorrect swapping leaves the distances large and thus will not be performed. For instance, a correct swapping repair t'_8 =(15:45, 28.5, 10.1, 0) of t_8 has a closer neighbor distance $\sum_{t_l \in N(t'_8)} \Delta(t'_8, t_l) = 6.926$, where $N(t'_8) = \{t_2, t_3, t_8\}$, compared to an incorrect swapping repair t''_8 =(15:45, 10.1, 28.5, 0) with distance $\sum_{t_l \in N(t''_8)} \Delta(t''_8, t_l)$ =8.255, where $N(t''_8) = \{t'_0, t_3, t_8\}$.

Given a dataset, one may first check each tuple t_0 in r whether swapping repair is applicable. If a swapping repaired tuple t'_0 is returned with significantly higher likelihood (lower distance cost) w.r.t. r, the tuple t_0 contains misplaced attribute values and should be repaired to t'_0. Otherwise, tuple t_0 is not changed in swapping, i.e., no misplaced attribute values found.

Example 5. *Let t_8 =(15:45, 28.5, 0, 10.1) be another tuple appearing after t_0 in Figure 1. The value 0 is not only dirty (a default direction value returned owing to device disturbance) but also misplaced in the temperature attribute. After swapping repair by our proposal, we have t'_8 =(15:45, 28.5, 10.1, 0). Existing in-attribute repair technique, e.g., [38], is then applied to repair the dirty value in t'_8[direction]. It returns a jointly repaired tuple t''_8 =(15:45, 28.5, 10.1, 330).*

IV. All Neighbors

We show (Proposition 1) that if all tuples in r are considered as neighbors when evaluating a repaired tuple, i.e., $\kappa = |r|$, the optimum repair problem is polynomial time solvable.

It is worth noting that the result is not only interesting for the special case of $\kappa = |r|$, but also enables the efficient computation for repairing with general κ (Algorithm 1 in Section V) and the approximation (Algorithm 2 in Section VI).

In the following, we first formalize the repairing problem as integer linear programming, which can thus be efficiently solved as an assignment problem in Section IV-B.

A. ILP Formulation

For $\kappa = |r|$, the κ-NN distance cost in Formula 5 is rewritten

$$\Theta_r^{|r|}(\mathbf{x}) = \sum_{t_l \in r} \sum_{A_i \in R} \sum_{A_j \in R} x_{ij} \cdot \Delta(t_0[A_j], t_l[A_i]). \quad (6)$$

The optimum repair problem can be formulated as Integer Linear Programming (ILP) [36].

$$\begin{aligned} \min & \sum_{l=1}^{n} \sum_{i=1}^{m} \sum_{j=1}^{m} x_{ij} \mathrm{d}_{lij} \\ & \sum_{i=1}^{m} x_{ij} = 1, && 1 \le j \le m \\ & \sum_{j=1}^{m} x_{ij} = 1, && 1 \le i \le m \\ & x_{ij} \in \{0, 1\}, && 1 \le i \le m, 1 \le j \le m \end{aligned}$$

where $x_{ij} = 1$ denotes $t'_0[A_i] = t_0[A_j]$, otherwise 0; and

$$\mathrm{d}_{lij} = \Delta(t_0[A_j], t_l[A_i]), 1 \le l \le n, 1 \le i \le m, 1 \le j \le m,$$

is the distance between $t_l[A_i]$ and $t'_0[A_i]$, where $t'_0[A_i]$ is replaced with $t_0[A_j]$.

Example 6. *Consider tuple t_0 for repairing and tuples $r = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_0\}$ in Figure 1. Let $\kappa = |r| = 8$. For each neighbor $t_l \in r$, we obtain the distance $\mathrm{d}_{lij} = \Delta(t_0[A_j], t_l[A_i])$ on attributes $A_j, A_i \in R$, for instance, $\mathrm{d}_{123} = \Delta(t_0[\textit{temperature}], t_1[\textit{voltage}]) = 0.503$.*

To minimize $\sum_{l=1}^{n} \sum_{i=1}^{m} \sum_{j=1}^{m} x_{ij} \mathrm{d}_{lij}$, the ILP solution leads to a repair $\mathbf{x}$, which is same as that in Example 3, i.e., $t'_0 = \mathbf{x} t_0 = (15{:}40, 33.3, 13.7, 330)$.

B. Solving as Assignment Problem

While existing ILP solvers [29] can be employed, we show below that the optimum repair problem with $\kappa = |r|$ can be solved efficiently as an assignment problem, i.e., the minimum weight perfect matching (MWPM) problem [21].

Consider a complete bipartite graph $G = (U, V; E)$ with m worker vertices (U) and m job vertices (V), and each edge $(u_i, v_j) \in E$ has a nonnegative cost w_{ij}. A matching is a subset $M \subseteq E$ such that at most one edge in M is incident

upon v, for each $v \in U \cup V$. A matching M is said perfect if every vertex in $U \cup V$ is adjacent to some edge in M. The cost of a matching M is defined as $\sum_{(u_i,v_j)\in M} w_{ij}$. The MWPM problem is to find a perfect matching M^* with minimum cost.

We now present the algorithm SRAN(t_0, r), Swapping Repair with All Neighbors, which considers all the tuples in r as neighbors. Specifically, we interpret the attributes in R as the agents and tasks in the MWPM problem, i.e., $U = V = R$. Given t_0 and r, we define the weight

$$w_{ij} = \sum_{t_l \in r} \Delta(t_0[A_j], t_l[A_i]) = \sum_{t_l \in r} \mathrm{d}_{lij}, \tag{7}$$

for $1 \le i \le m, 1 \le j \le m$. By using the Hungarian method [21], it returns an optimal matching M^*. For each $(u_i, v_j) \in M^*$, we assign $x_{ij} = 1$, i.e., $t'_0[A_i] = t_0[A_j]$. It forms a swapping repair $\mathbf{x}$ with cost

$$\begin{aligned}\Theta_r^{|r|}(\mathbf{x}) &= \sum_{t_l \in r} \sum_{A_i \in R} \sum_{A_j \in R} x_{ij} \cdot \Delta(t_0[A_j], t_l[A_i]) \\ &= \sum_{i=1}^{m} \sum_{j=1}^{m} x_{ij} w_{ij} = \sum_{(u_i,v_j)\in M^*} w_{ij}.\end{aligned} \tag{8}$$

Proposition 1. *The* SRAN *algorithm returns an optimal repair in $O(nm^2 + m^3)$ time, where $n = |r|, m = |R|$.*

Example 7. *Consider tuple t_0 in Figure 1 for repairing. The* SRAN *algorithm employs all the tuples in r, i.e., $r = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_0\}$, as neighbors when evaluating the swapping repaired tuple.*

Referring to Formula 7, we initialize the weight w_{ij} for each variable x_{ij}, e.g., $w_{23} = 0.503 + 0.983 + 0.999 + 0.593 + 0.999 + 1 + 1 + 1 = 7.077$. The swapping repair problem is then solved as an assignment problem, having $U = V =$ {time, voltage, temperature, direction} together with the corresponding weight w_{ij} for each edge $(u_i, v_j) \in E$.

By calling the Hungarian method for solving the MWPM problem, a perfect matching M^* is computed, having $M^* = \{(u_1, v_1), (u_2, v_3), (u_3, v_2), (u_4, v_4)\}$ with total weight 22.467. The corresponding optimal swapping repair for t_0 can thus be generated accordingly. For example, $(u_2, v_3) \in E$ denotes that t'_0[voltage] $= t_0$[temperature]. Consequently, the repaired tuple is $t'_0 = (15{:}40, 33.3, 13.7, 330)$ with cost 22.467.

V. Fixed Number of Neighbors

As discussed at the beginning of Section IV, the hardness originates from the number κ of nearest neighbors. We show (in Proposition 2) that for a fixed number κ of neighbors from r in evaluating a repaired tuple, the optimum repair problem can be solved in polynomial time.

Let us first present the quadratic programming formulation of the repairing problem, which sparks the solution in Section V-B by calling the aforesaid SRAN algorithm on all neighbors in Section IV-B. Pruning technique is devised for efficient repairing.

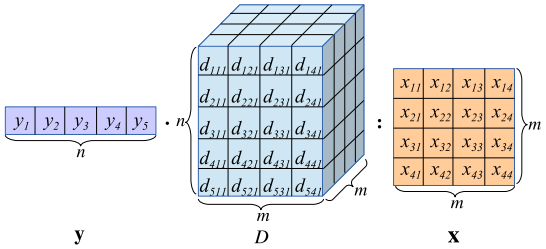


Fig. 3. The objective of quadratic programming

A. QP Formulation

The κ-NN distance cost in Formula 5 is rewritten as

$$\Theta(\mathbf{x}) = \sum_{t_l \in r} y_l \sum_{A_i \in R} \sum_{A_j \in R} x_{ij} \cdot \Delta(t_0[A_j], t_l[A_i]), \tag{9}$$

where $y_l = 1$ denotes that t_l is the κ-NN of the repaired tuple $t'_0 = \mathbf{x}t_0$, i.e., $t_l \in N(\mathbf{x}t_0)$; otherwise $y_l = 0$.

The optimum repair problem can be formulated as Quadratic Programming (QP) [27].

$$\begin{aligned}
\min \quad & \sum_{l=1}^{n} y_l \sum_{i=1}^{m} \sum_{j=1}^{m} x_{ij} \mathrm{d}_{lij} \\
\text{s.t.} \quad & \sum_{l=1}^{n} y_l = \kappa, \\
& \sum_{i=1}^{m} x_{ij} = 1, && 1 \le j \le m \\
& \sum_{j=1}^{m} x_{ij} = 1, && 1 \le i \le m \\
& x_{ij} \in \{0, 1\}, && 1 \le i \le m, 1 \le j \le m \\
& y_l \in \{0, 1\}, && 1 \le l \le n
\end{aligned}$$

where both x_{ij} and y_l are variables to solve.

Figure 3 illustrates the relationships among variables y_l, variables x_{ij} and constants d_{lij}. Let $\mathbf{y}$ denote the vector of variables y_l, $\mathbf{x}$ be the matrix of variables x_{ij} as in Definition 1, and D be the tensor of constants d_{lij}. The objective $\sum_{l=1}^{n} y_l \sum_{i=1}^{m} \sum_{j=1}^{m} x_{ij} \mathrm{d}_{lij}$ is rewritten as $\mathbf{y} \cdot D : \mathbf{x}$. As the tensor multiplication has law of commutation, we first consider the double inner product between tensor D and $\mathbf{x}$, having

$$D : \mathbf{x} = \sum_{i=1}^{m} \sum_{j=1}^{m} \sum_{p=1}^{m} \sum_{q=1}^{m} d_{lij} x_{pq} \delta_{ip} \delta_{jq},$$

$$\delta_{ip} = \begin{cases} 1, i = p \\ 0, i \ne p \end{cases}, \quad \delta_{jq} = \begin{cases} 1, j = q \\ 0, j \ne q \end{cases}.$$

Then the dot product between $\mathbf{y}$ and the aforesaid double inner product result $D : \mathbf{x}$ is exactly the distance cost $\Theta(\mathbf{x})$.

Example 8 (Example 6 continued)**.** *By transforming the problem to quadratic programming, each tuple $t_l \in \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_0\}$ is associated with a variable y_l, which denotes whether it is considered as one of the κ-*NN

for the repaired $t_0' = \mathbf{x}t_0$. For instance, given $\kappa = 3$, there are only three tuples in r having $y_l = 1$. Variables x_{ij} and constants d_{lij} have the same meaning as in Example 6.

By minimizing the objective $\sum_{l=1}^{n} y_l \sum_{i=1}^{m}\sum_{j=1}^{m} x_{ij}\mathrm{d}_{lij}$, it leads to a QP solution where the result of variables x_{ij} is the same as that in ILP in Example 6, and $y_1 = y_7 = y_0 = 1$, $y_2 = y_3 = y_4 = y_5 = y_6 = 0$. The repaired tuple is $t_0' = \mathbf{x}t_0 = (15{:}40, 33.3, 13.7, 330)$.

B. Solving as All Neighbors

Referring to the aforesaid QP formulation, we show that the repairing problem with κ neighbors can indeed be solved by calling SRAN algorithm for all neighbors. The idea is to consider (all) the sets $T \subseteq r$ as potential κ-NN, compute the optimal repair over each T by calling $\textsc{sran}(t_0, T)$, and among them return the one with the minimum distance cost.

Algorithm 1, $\textsc{srkn}(t_0, r, T, S)$, Swapping Repair with κ Neighbors, presents the recursive computing of considering all the subsets of r with size κ, where T is the current set of potential κ-NN, and S denotes the remaining tuples that may form potential κ-NN. To initialize, we have $T = \{t_1, t_2, \ldots, t_\kappa\}$, $S = r \setminus T = \{t_{\kappa+1}, t_{\kappa+2}, \ldots, t_n\}$,

Let $\mathbf{x}^*$ be the currently known best solution with distance cost $\theta^* = \Theta_r^\kappa(\mathbf{x}^*)$ as defined in Formula 5. Initially, we have $\mathbf{x}^* = \textsc{sran}(t_0, T)$ and $\theta^* = \Theta_T^{|T|}(\mathbf{x}^*)$ referring to Formula 6.

Lines 1 to 4 generate the next potential T' by removing one tuple t_p from T and adding another t_q from S. (Lines 5, 6, 13 and 14 are used for pruning, which will be introduced below.) For this T', Line 7 calls $\textsc{sran}(t_0, T')$ to find the optimal repair $\mathbf{x}$ over T'. If its cost θ is lower than the currently known best θ^*, $\mathbf{x}$ will become $\mathbf{x}^*$. Line 12 recursively calls the algorithm for the current T' and S'.

Algorithm 1: $\textsc{srkn}(t_0, r, T, S)$

Input: t_0 a tuple considered for repairing, r a set of tuples,
T current set of potential κ-NN,
S remaining tuples that may form potential κ-NN
Output: an optimal swapping repair $\mathbf{x}^*$ of t_0

1 **for** *each* $t_p \in T$ **do**
2 **for** *each* $t_q \in S$ **do**
3 $T' \leftarrow T \setminus \{t_p\} \cup \{t_q\}$;
4 $S' \leftarrow S \setminus \{t_q\}$;
5 $\theta_{T'}^{\min} \leftarrow$ lower bound computed by Formula 11;
6 **if** $\theta^* > \theta_{T'}^{\min}$ **then**
7 $\mathbf{x} \leftarrow \textsc{sran}(t_0, T')$;
8 $\theta \leftarrow \Theta_{T'}^{|T'|}(\mathbf{x})$ by Formula 8;
9 **if** $\theta < \theta^*$ **then**
10 $\theta^* \leftarrow \theta$;
11 $\mathbf{x}^* \leftarrow \mathbf{x}$;
12 $\textsc{srkn}(t_0, r, T', S')$;
13 **else**
14 break;
15 **return** $\mathbf{x}^*$

Example 9. *Consider the tuples in r in Figure 1. Given $\kappa = 3$, we initialize $T = \{t_1, t_2, t_3\}$, $S = r \setminus T = \{t_4, t_5, t_6, t_7, t_0\}$. The currently known best solution $\mathbf{x}^*$ with distance cost $\theta^* = 9.223$ for t_0 is calculated by calling the algorithm $\textsc{sran}(t_0, T)$. Lines 1 to 3 remove one tuple t_1 from T and add another tuple t_4 from S to generate the next potential subset $T' = \{t_2, t_3, t_4\}$. (By ignoring Lines 5 and 6 which will be introduced below for pruning) Line 7 calls the algorithm $\textsc{sran}(t_0, T')$ to generate the swapping repair $\mathbf{x}$, and the corresponding distance cost θ is calculated according to Formula 8 in Line 8, having $\theta = 9.283 > \theta^*$. It thus considers the next tuples in S. For $t_7 \in S$, we have $T' = \{t_2, t_3, t_7\}$, with the distance cost $\theta = 8.812 < \theta^*$. It updates the currently known best swapping repair $\mathbf{x}^*$ with distance cost θ^* accordingly, and recursively calls the algorithm $\textsc{srkn}(t_0, r, T', S')$ using the current T' and S' in Line 12.*

Proposition 2. *Algorithm 1* SRKN *returns an optimal repair in $O(n^\kappa(\kappa m^2 + m^3))$ time, where $n = |r|, m = |R|$.*

C. Pruning by Bounds on Neighbor Distances

Given any set $T' \subseteq r$, we propose to compute a lower bound of distance cost $\theta_{T'}^{\min}$ for the optimal repair over T'. As illustrated in Line 6 in Algorithm 1, if the lower bound $\theta_{T'}^{\min}$ is no better than θ^*, this T' can be pruned (as presented below, the recursion on T' stops as well).

1) Bounds on Neighbor Distances: We study the distance cost on each tuple $t_l \in r$ for potential repairs. It is utilized to compute the bound of distance costs on any set $T \subseteq r$.

For each tuple $t_l \in r$, by calling $\mathbf{x}^l = \textsc{sran}(t_0, \{t_l\})$, we compute the optimal repair $\mathbf{x}^l$ over one (all) neighbor t_l. Referring to Formula 6, the corresponding distance cost $\theta_l^{\min}$ is

$$\theta_l^{\min} = \Theta_{\{t_l\}}^1(\mathbf{x}^l) = \sum_{i=1}^{m}\sum_{j=1}^{m} x_{ij}^l \mathrm{d}_{lij}. \tag{10}$$

Given any set $T \subseteq r$ of potential κ-NN, we define $\theta_T^{\min} = \sum_{t_l \in T} \theta_l^{\min}$. The proposition below shows that $\theta_T^{\min}$ is a lower bound of the distance cost for the optimal repair $\mathbf{x}^*$ over T.

Proposition 3. *For any tuple set $T \subseteq r$, we have*

$$\theta_T^{\min} = \sum_{t_l \in T} \theta_l^{\min} \leq \Theta_T^{|T|}(\mathbf{x}^*), \tag{11}$$

where $\mathbf{x}^*$ is an optimal swapping repair of t_0 over T.

Example 10 (Example 9 continued)**.** *Consider a potential subset $T' = \{t_2, t_3, t_4\}$ in Example 9. We compute the distance cost $\theta_l^{\min}$ for each $t_l \in T'$ w.r.t. tuple t_0, using Formula 10. For example, we have $\theta_2^{\min} = 1{*}0.274{+}1{*}0.983{+}1{*}0.999{+}1{*}1 = 3.256$. Similarly, we get $\theta_3^{\min} = 3.236$ and $\theta_4^{\min} = 2.791$. The lower bound $\theta_{T'}^{\min}$ of T' is computed referring to Formula 11, having $\theta_{T'}^{\min} = \theta_2^{\min} + \theta_3^{\min} + \theta_4^{\min} = 9.283$. For any possible swapping repair of t_0, e.g., $t_0'' = \mathbf{x}''t_0$ =(15:40, 330, 33.3, 13.7) with distance cost $\Theta_{T'}^{|T'|}(\mathbf{x}'') = 9.505$ in Example 4, we always have $\theta_{T'}^{\min} \leq \Theta_{T'}^{|T'|}(\mathbf{x}^*)$.*

2) Pruning by Bounds: We show that the lower bound in Proposition 3 enables not only the pruning of the current T' in Line 6 in Algorithm 1, but also the subsequent recursion on T' in Line 12.

To ensure correctness of pruning, tuples $t_l \in r$ are sorted in ascending order according to $\theta_l^{\min}$. It guarantees that the lower bounds of considered sets are non-decreasing in recursion.

Proposition 4. *For the set T' considered in Line 3 in Algorithm 1, it always has $\theta_T^{\min} \leq \theta_{T'}^{\min}$.*

Once we have $\theta_{T'}^{\min} \geq \theta^*$ in Line 6 in Algorithm 1, all the subsequent sets (say T'') in the recursion on T' always have $\theta_{T''}^{\min} \geq \theta^*$ and thus can be directly pruned as well.

Example 11 (Example 9 continued)**.** *In order to guarantee the correctness of pruning in Line 6 in Algorithm 1, we first sort the tuples $t_l \in r$ in an ascending order w.r.t. $\theta_l^{\min}$, having $r = \{t_0, t_7, t_1, t_4, t_6, t_5, t_3, t_2\}$. Suppose that $T' = \{t_7, t_1, t_4\}$ and $S = \{t_6, t_5, t_3, t_2\}$, with the currently known minimum distance cost $\theta^* = 7.049$. The next potential subset $T' = \{t_7, t_1, t_4\}$ has lower bound $\theta_{T'}^{\min} = 2.730 + 2.791 + 2.978 = 8.499$. Since we have $\theta^* < \theta_{T'}^{\min}$, i.e., the condition in Line 6 is not satisfied, the enumeration of S can be directly stopped in Line 2. The reason is that the lower bounds of the next potential T' must be in an ascending order (Proposition 4), with distance cost greater than θ^*.*

VI. Fixed Sets of Neighbors

Instead of considering (all) the potential sets $T \subseteq r$ of κ-NN to compute an exact solution in Algorithm 1, we propose to heuristically consider a number of fixed neighbor sets. Intuitively, the κ-NN of a tuple should be close with each other as well. Rather than enumerating all the subsets of r (with size κ), we may consider only the κ-NN of each tuple $t_l \in r$ as potential set T.

Algorithm 2, SRFN(t_0,r), Swapping Repair with Fixed Neighbors, presents the approximation computation. Let $\mathbf{T}$ be the κ-NN of each tuple t_l in r

$$\mathbf{T} = \{N_r^\kappa(t_l) \mid t_l \in r\}. \tag{12}$$

Line 2 considers only the set $T \in \mathbf{T}$ as potential κ-NN for evaluating a repair. For each $T \in \mathbf{T}$, the computation of a repair is generally similar to Lines 5-11 in Algorithm 1.

Algorithm 2: SRFN(t_0,r)

Input: t_0 a tuple considered for repairing, r a set of tuples
Output: a near optimal swapping repair $\mathbf{x}^*$ of t_0

```
1  T ← all sets T of potential κ-NN defined in Formula 12;
2  for each T ∈ T do
3      θ_T^min ← lower bound computed by Formula 11;
4      if θ* > θ_T^min then
5          x ← SRAN(t_0, T);
6          θ ← Θ_T^{|T|}(x) by Formula 8;
7          if θ < θ* then
8              θ* ← θ;
9              x* ← x;
10 return x*
```

Example 12. *Consider tuple t_0 in r in Figure 1 with $\kappa = 3$. Rather than enumerating all the possible subsets $T \subseteq r$ with size κ to compute an exact solution in Algorithm 1, we consider only the κ-NN of each tuple $t_l \in r$ as potential set T. For instance, given $T = N_r^\kappa(t_6) = \{t_1, t_6, t_7\}$ in Line 2 in Algorithm 2, we first compute the lower bound of distance cost by Formula 11, having $\theta_T^{\min} = \theta_1^{\min} + \theta_6^{\min} + \theta_7^{\min} = 2.730 + 2.978 + 2.319 = 8.027$. As the currently known minimum distance cost is greater than $\theta_T^{\min}$, we call the* SRAN *algorithm to compute the swapping repair $\mathbf{x}$ for t_0, which leads to the repaired tuple $\mathbf{x}t_0 = (15{:}40, 33.3, 13.7, 330)$.*

TABLE II
Dataset summary

Dataset	$\lvert r\rvert$	# clean tuples	# error tuples	$\lvert R\rvert$	# DCs
Magic	19k	17k	2k (synthetic)	10	4
Restaurant	864	764	100 (synthetic)	4	7
Chess	28k	26k	2k (synthetic)	6	12
Skin	245k	242k	3k (synthetic)	4	3
Turbine	5.2k	5k	200 (real)	9	9
FEC	50k	46.7k	3.3k (real)	15	15

VII. Experiment

In this section, we evaluate our proposal over real-world misplaced attribute values. Moreover, we show how the swapping repair complements existing methods to remedy both misplaced-attribute and in-attribute errors in a dataset.

A. Experimental Settings

All programs are implemented in Java. Experiments are performed on a machine with 3.1GHz CPU and 16GB memory.

1) Datasets: We employ 6 real datasets in evaluation. Table II lists the major statistics of the datasets.

Turbine is a dataset with real misplaced attribute values and manually labeled truth, collected by a world leading wind turbine manufacturer. Each tuple records 9 data fields of a wind turbine, including timestamp, site id, turbine id, three phase voltage values and three temperature values of pitch inverters. As introduced in Example 1, owing to mis-plugging by workers, three phase voltage values and the temperature values of three pitch inverters are misplaced. There are total 200 tuples with misplacement and manually labeled truth.

FEC [2] is another dataset with real-world misplaced errors and manually labeled truth. Each tuple records a committee registered with the Federal Election Commission, including federal political action committees, party committees, etc. As identified in [20], the FEC dataset has some rows with the wrong number of columns, leading to data type errors, i.e., misplacement. When parsing the raw data, since commas in values are mistakenly interpreted as separators, a number of 3.3k tuples are observed with misplacement out of 50k.

Magic [4] consists of generated data to simulate the registration of high energy gamma particles in a ground-based atmospheric Cherenkov gamma telescope using the imaging technique. The classification task is to discriminate the images generated by primary gammas (signal, class label g) from the

images of hadronic showers initiated by cosmic rays in the upper atmosphere (background, class label h).

Chess [1] represents the positions on board of the white king, the white rook, and the black king. The classification (with 18 classes) is to determine the optimum number of turns required for white to win the game, which can be a draw if it takes more than sixteen turns.

Restaurant [5] is a collection of restaurant records containing 112 duplicates. Each record includes four attributes, name, address, city and type. It has been used as a benchmark dataset in record matching [32] and data cleaning [39].

Skin [6] is collected by randomly sampling B,G,R values from face images of various age groups (young, middle, and old), race groups (white, black, and asian), and genders. It is utilized to evaluate the scalability for our proposed approaches.

2) Error Generation: Since the Magic, Restaurant, Chess and Skin datasets are originally clean, following the same line of evaluating data cleaning algorithms [8], we introduce synthetic errors in the data.

To inject misplaced-attribute errors, we consider a set of attributes $W \subseteq R$ for replacement, e.g., $W = \{A_1, A_2, A_3\}$. For any tuple $t_0^{\#}$ that is originally clean (ground truth), the corresponding dirty tuple t_0 is generated by randomly re-assigning the values on attributes W, such as $t_0[A_1] = t_0^{\#}[A_2], t_0[A_2] = t_0^{\#}[A_3], t_0[A_3] = t_0^{\#}[A_1]$. Each value $t_0^{\#}[A_i]$ is only re-assigned once.

To inject in-attribute errors (for evaluating the joint repair of swapping and existing methods), we consider various types of errors, including errors detectable by constraints, outliers and missing values. Following [8], FDs are discovered by TANE [17] and detectable errors are generated based on the mined FDs. Moreover, we randomly remove/modify cells from various attributes as missing/outlier values.

3) Evaluation Criteria: For numerical values, to measure the repair accuracy, we employ the repair error measure, root-mean-square error (RMS) [18]. For each tuple t_0, it evaluates how close the repaired result t_0' to the corresponding truth $t_0^{\#}$, $\mathsf{RMS}(t_0^{\#}, t_0') = \sqrt{\sum_{A \in R} \frac{(t_0'[A] - t_0^{\#}[A])^2}{|R|}}$. The lower the RMS error is, the more accurate the repair is (closer to the truth).

For categorical values, e.g., in FEC, Restaurant and Chess, we measure the accuracy on whether a repair equals exactly the ground truth. Let truth be the set of truths for dirty values and found be the set of repair results returned by cleaning algorithms. The accuracy is given by $\mathsf{accuracy} = \frac{|\mathsf{truth} \cap \mathsf{found}|}{|\mathsf{truth}|}$, i.e., the proportion of dirty values that are accurately repaired. We also utilize the accuracy measurement to evaluate the error detection accuracy for both numerical and categorical data.

4) Baselines and Implementation Details: For reference, we compare our proposed approach against several competing in-attribute repair methods, including the constraint based data repair algorithms HoloClean [33], ER [15], DD [38], and the non-constraint based data cleaning approaches SCARE [41], ERACER [25] and DORC [39].

SCARE [41] performs data repairing based on the probabilistic correlations between reliable attributes with correct values and flexible attributes with possible dirty values. Trusted data are used in SCARE, i.e., the tuples without injected errors in the Magic, Restaurant and Chess datasets of synthetic errors, or the tuples without errors identified in the Turbine and FEC datasets with real-world errors. The sizes of trusted data are reported in Table II. We utilize all the trusted data to train the prediction model. The left-hand-side (LHS) attributes of the aforesaid discovered FDs are considered as the reliable attributes for SCARE.

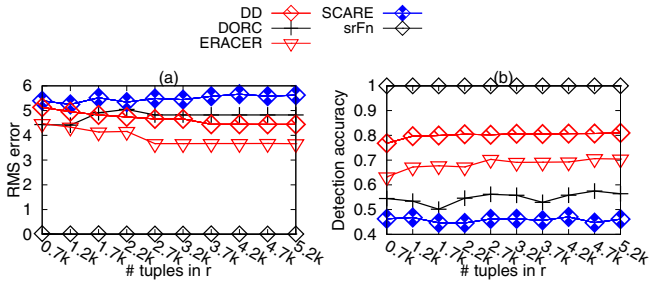


Fig. 4. Swapping repair (SRFN) over Turbine data with 200 tuples containing 6 real-world misplaced attributes

ER [15] determines certain fixes with editing rules defined on value equality. Similar to SCARE, the reference data of ER are the tuples without synthetic errors injected or real-world errors identified. The sizes of reference data used in ER are the same as the trusted data for SCARE, reported in Table II. We discover CFDs from the clean data by [14] as editing rules to find certain fixes.

HoloClean [33] employs not only the constraints but also the statistical learning to repair dirty values. Again, general DCs are discovered from the clean data by [11] for HoloClean. The number of discovered DCs is presented in Table II. These rules cover all the attributes with synthetic or real-world errors. We adapt its open source implementation [3] for misplacement repair by using other values in the same tuple as candidates for the erroneous cell.

DD [38] introduces extensions with distance functions. Since DCs with distance functions are not directly supported in the current HoloClean implementation [3], we consider the repair under the DD (neighborhood) constraints on distances [38]. Again, DDs are discovered from the clean data by [37].

ERACER [25] iteratively learns a global relational dependency model, i.e., linear regression model for numerical data and relational dependency network for categorical data in our experiments, to infer the probabilistic relationships among attributes. Again, the same LHS attributes of FDs are utilized as the determinant attributes for ERACER.

DORC [39] minimally repairs the outliers (e.g., the tuples with misplaced attribute values) to existing points (tuples). Since the exact algorithm needs to call the costly ILP solver, we use the quadratic time approximation version QDORC.

B. Swapping Repair of Misplaced-Attribute Errors

Figures 4 and 5 report the repair and error detection results over the real misplaced errors in Turbine and FEC datasets

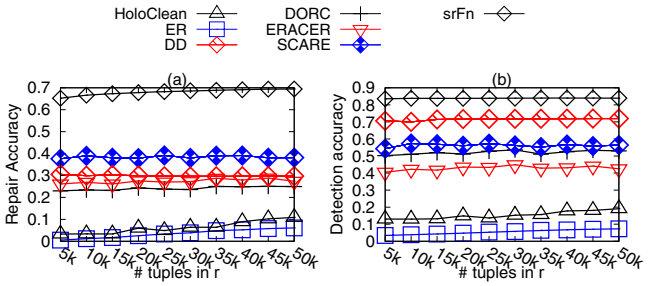


Fig. 5. Swapping repair (SRFN) over FEC data with 3.3k tuples containing 8 real-world misplaced attributes

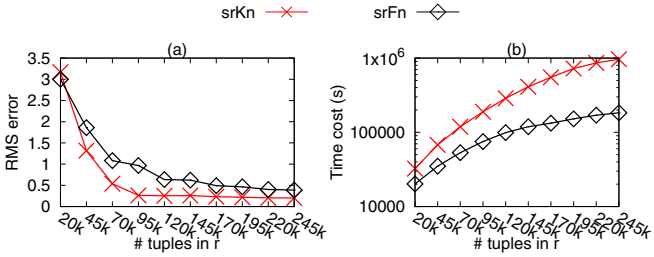


Fig. 6. Comparing the approximate swapping repair SRFN to the exact SRKN over Skin with 3k tuples containing 2 misplaced attributes

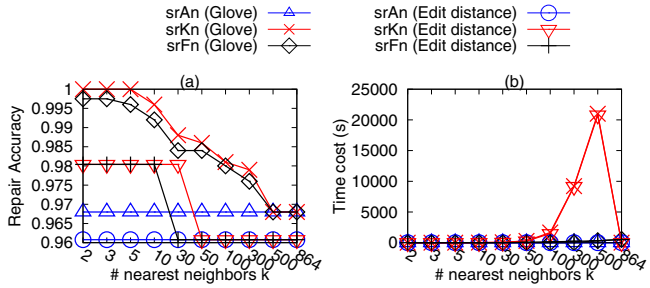


Fig. 7. Varying the number of nearest neighbors κ in swapping repair over Restaurant data with 50 tuples containing 2 misplaced attributes

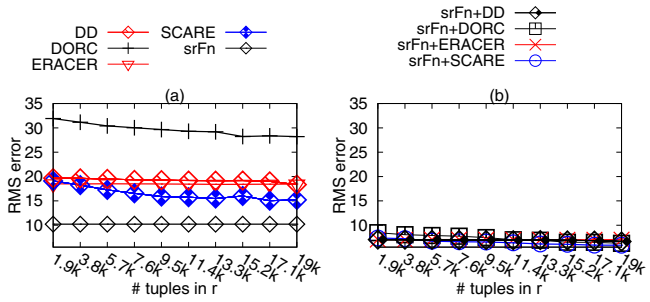


Fig. 8. Joint repair over Magic data with 1k tuples containing 2 misplaced attributes and 1k tuples having in-attribute errors, including 1/3 constraint detectable errors, 1/3 outliers and 1/3 missing values

under various numbers of tuples in r. It is worth noting that there are no existing approaches dedicated to repair misplaced-attribute errors. For reference, we report the results of directly applying in-attribute repair methods, including the constraint based data repair algorithms HoloClean [33], ER [15], DD [38], and the non-constraint based data cleaning approaches SCARE [41], ERACER [25] and DORC [39]. In particular, the SCARE approach considers the distribution of values in the attribute domain and across attributes. As shown in Figures 4 and 5, approaches with higher error detection accuracy lead to more accurate repair accuracy as well.[2] Indeed, it is not a fair comparison, and a more reasonable yet practical evaluation is to perform joint repair, as presented in Section VII-C below. Nevertheless, as shown in Figures 4(b) and 5(b), comparing to the multivariate outlier detection approaches handling outliers, e.g., DORC, ERACER and SCARE, the swapping repair SRFN is very effective in repairing misplacement, with a higher misplaced error detection accuracy. The reason is that multivariate outlier detection may return false-positives of misplaced errors, i.e., true outliers without misplaced values.

To show the scalability of our proposal, Figure 6 reports the results over a large Skin dataset with up to 245k tuples. While the complexity of the exact Algorithm 1 SRKN is high, the approximate Algorithm 2 SRFN reduces the complexity and shows significantly lower time costs in Figure 6. Nevertheless, the RMS error of SRFN is still very close to that of SRKN.

Figure 7 presents the results with different distance functions, i.e., using pre-trained embedding Glove [30] and edit distance [26]. As shown, the repair accuracy is improved with the help of pre-trained embedding techniques, comparing to traditional measurement function edit distance. In addition to the exact SRKN and the approximate SRFN, we also report SRAN as baseline, where all the tuples are considered as neighbors (and thus the result of SRAN does not change with κ). As shown in Figure 7(a), the approximate SRFN shows almost the same results as the exact SRKN, when κ is small. The repair accuracy is lower if κ is too large, since irrelevant tuples may be considered as neighbors and obstruct repairing. When $\kappa = n$, it is not surprising that SRKN shows the same results as SRAN. To determine a proper κ, one can sample some data from r, manually injecting misplaced errors, and see which κ can best repair these errors (like Figure 7). The remaining data are then evaluated using the selected κ.

C. Joint Repair of Misplaced-Attribute and In-Attribute Errors

Figures 8, 9 and 10 report the results with various error types, including misplacement, constraint detectable errors, outliers and missing values as injected in Section VII-A2. As shown, ERACER and SCARE, which can handle various types of errors, achieve a better performance than the other baselines alone.[3] Figures 8(b), 9(b), and 10(b) present the joint repair where our proposal SRFN is paired with the existing in-attribute repair approaches. As shown, SRFN+ERACER and SRFN+SCARE show higher accuracy. The result is not surprising referring to the better performance of ERACER and SCARE, compared with DORC and so on in Figures 8(a), 9(a), 10(a). The joint repair such as SRFN+SCARE shows better performance than any individual ones. These promising results

[2]ER and HoloClean with clearly worse results are omitted in Figure 4.

[3]ER and HoloClean with clearly higher RMS error are omitted in Figure 8.

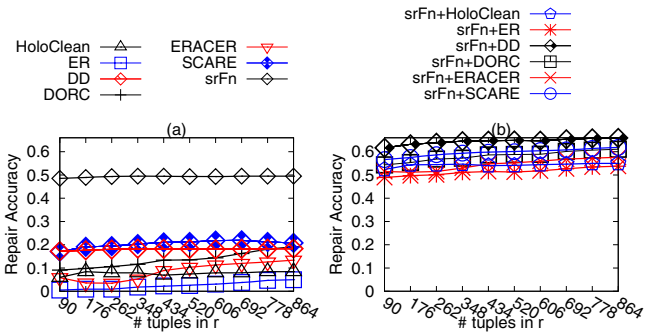


Fig. 9. Joint repair over Restaurant data with 50 tuples containing 2 misplaced attributes and 50 tuples having in-attribute errors, including 1/3 constraint detectable errors, 1/3 outliers and 1/3 missing values

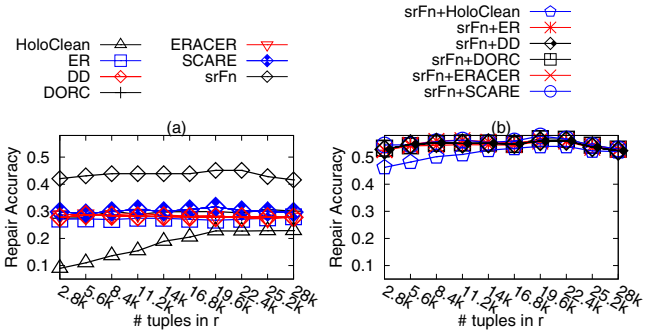


Fig. 10. Joint repair over Chess data with 1k tuples containing 2 misplaced attributes and 1k tuples having in-attribute errors, including 1/3 constraint detectable errors, 1/3 outliers and 1/3 missing values

demonstrate not only the necessity of studying swapping repairs for misplacement, but also the practical solution for jointly remedying both error types.

To illustrate that the order of repair steps affects the final results of the joint repair in Section III-C, Figure 11 considers various combinations of swap, repair and impute steps. SCARE [41] is considered in repair and imputation, and SRFN is used for swapping. The results verify that dirty values in attributes have little effect on the swapping repair for the misplaced errors. The pipeline Swap-Repair-Impute achieves the best performance. In contrast, other pipelines such as Repair-Impute-Swap applying in-attribute error repair first have low accuracy.

VIII. Related Work

While distance has been recognized as an important signal of data cleaning in [39], this paper is different from other studies such as [34] and [35] in both the conceptual and technical aspects. (1) The concepts on distances are different. While this paper studies the likelihood of distances between tuples, [34] considers the constraints on distances and [35] learns regression models to predict the distances among attributes. (2) The problems are different. This paper proposes to maximize the distance likelihood of a tuple by swapping its values to address misplacement errors. Instead, [34] is to minimize the changes towards the satisfaction of distance constraints

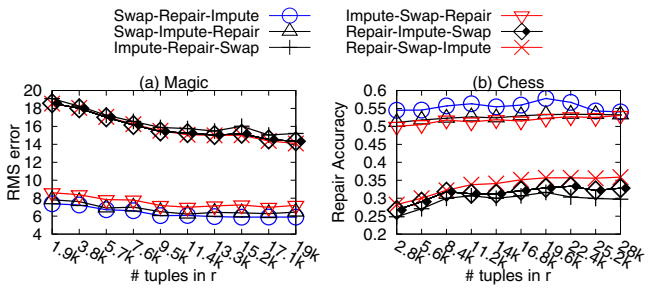


Fig. 11. Joint repair (SRFN+SCARE) using various pipelines over (a) Magic and (b) Chess data with 1k tuples containing 2 misplaced attributes and 1k tuples having in-attribute errors, including 1/3 constraint detectable errors, 1/3 outliers and 1/3 missing values

to eliminate in-attribute errors, and [35] imputes the missing values w.r.t. the predicted distances on an attribute. (3) The devised techniques are also very different given the aforesaid distinct problems. In order to avoid enumerating the κ-NN combinations for all the possible swapping repaired tuples, this paper considers approximately the fixed sets of neighbors. On the contrary, [34] proposes to utilize the bounds of repair costs for pruning and approximation. Moreover, [35] imputes each incomplete attribute individually in approximation, which is unlikely in the scenario of this study (swapping occurs between at least two attributes).

A. Data Repairing

While no studies have been found to address misplacement, as illustrated in Section III-C, our proposal could complement the existing approaches to repair both misplaced-attribute and in-attribute errors. We briefly summarize below the typical data repairing methods, for in-attribute errors. Editing rules (ER) rely on certain regions [15] to determine certain fixes, where constraints are built upon equality value relationships between the dirty tuples and master data. Owing to the strict value equality relationships, the numerical or heterogeneous values with various information formats often prevent finding sufficient neighbors from master data. It makes the dirty values barely fixed. Statistical-based approaches employ statistic models for data repairing. In SCARE [41], the attributes in a relation for repairing are divided into two parts, i.e., reliable attributes with correct values and flexible attributes with dirty values. Probabilistic correlations between reliable attributes and flexible attributes are then modeled, referring to the value distribution. The repairing objective is thus to modify the data to maximize the likelihood. ERACER [25] constructs a relational dependency network to model the probabilistic relationships among attributes, where the cleaning process performs iteratively and terminates when the divergence of distributions is sufficiently small.

B. Outlier Detection and Cleaning

Distance-based outlier detection [19] determines a fraction p and distance threshold ϵ according to data distributions, and considers an object as an outlier if at least p of objects have distances greater than ϵ to it. Our proposed methods share the

similar idea that a tuple with occasionally misplaced attribute values is outlying. In this sense, it extends the existing outlier detection technique, i.e., (1) detecting outliers as suspected tuples with potentially misplaced values, and (2) swapping attribute values in an outlier to see whether it possibly becomes an inlier. Of course, an outlier may not be changed after repair checking (i.e., no swapped tuple shows higher likelihood than the original outlier tuple). It denotes that no misplaced values are detected in this outlier tuple. In contrast, the existing DORC [39] repairs all the outlier tuples by the values of other tuples, to make each outlier an inlier. It may excessively over-repair the outliers where no errors indeed occur.

IX. Conclusion

In this paper, we first summarize the sources of misplaced attribute values, ranging from Web forms to IoT scenarios, covering all the ETL phases. Unlike the widely considered in-attribute errors, the true value of misplaced-attribute error is indeed in some other attribute of the same tuple. While swapping repair is intuitional, it is non-trivial to evaluate the likelihood of a tuple on whether its values belong to the corresponding attributes. As illustrated in Section II-A, owing to the sparsity and heterogeneity issues, studying the distribution directly on values may not work. Instead, we argue to evaluate the likelihood by how the values are similar/distant to others. The rationale of distance likelihood lies in the Poisson process of nearest neighbor appearance. To find the optimum swapping repair with the maximum distance likelihood, we show that the optimum repair problem is polynomial time solvable, in Proposition 1, when considering all the tuples as neighbors; devise an exact algorithm for a fixed number of neighbors, together with bounds of distances in Proposition 3 for pruning; and propose an approximation algorithm by considering fixed sets of neighbors. Extensive experiments on datasets with real-world misplaced attribute values demonstrate the effectiveness of our proposal in repairing misplacement.

Acknowledgement

This work is supported in part by the National Natural Science Foundation of China (61572272, 71690231).

References

[1] Chess dataset. *https://sci2s.ugr.es/keel/dataset.php?cod=197*.
[2] FEC dataset. *https://www.fec.gov/data/browse-data/?tab=bulk-data*.
[3] HoloClean. *https://github.com/HoloClean/HoloClean*.
[4] Magic dataset. *https://sci2s.ugr.es/keel/dataset.php?cod=102*.
[5] Restaurant dataset. *http://www.cs.utexas.edu/users/ml/riddle/data.html*.
[6] Skin dataset. *http://archive.ics.uci.edu/ml/datasets/Skin+Segmentation*.
[7] Z. Abedjan, X. Chu, D. Deng, R. C. Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, and N. Tang. Detecting data errors: Where are we and what needs to be done? *PVLDB*, 9(12):993–1004, 2016.
[8] 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.
[9] P. Bohannon, M. Flaster, W. Fan, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*, pages 143–154, 2005.
[10] S. Chen, B. Ma, and K. Zhang. On the similarity metric and the distance metric. *Theor. Comput. Sci.*, 410(24-25):2365–2376, 2009.
[11] X. Chu, I. F. Ilyas, and P. Papotti. Discovering denial constraints. *PVLDB*, 6(13):1498–1509, 2013.
[12] X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. In *ICDE*, pages 458–469, 2013.
[13] W. Fan and F. Geerts. *Foundations of Data Quality Management*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.
[14] W. Fan, F. Geerts, L. V. S. Lakshmanan, and M. Xiong. Discovering conditional functional dependencies. In *ICDE*, pages 1231–1234, 2009.
[15] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Towards certain fixes with editing rules and master data. *PVLDB*, 3(1):173–184, 2010.
[16] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques, 3rd edition*. Morgan Kaufmann, 2011.
[17] 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.
[18] S. R. Jeffery, M. N. Garofalakis, and M. J. Franklin. Adaptive cleaning for RFID data streams. In *VLDB*, pages 163–174, 2006.
[19] E. M. Knorr, R. T. Ng, and V. Tucakov. Distance-based outliers: Algorithms and applications. *VLDB J.*, 8(3-4):237–253, 2000.
[20] S. Krishnan, M. J. Franklin, K. Goldberg, and E. Wu. Boostclean: Automated error detection and repair for machine learning. *CoRR*, abs/1711.01299, 2017.
[21] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics (NRL)*, 2(1-2):83–97, 1955.
[22] Y. Li and B. Liu. A normalized levenshtein distance metric. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(6):1091–1095, 2007.
[23] G. Little and Y.-A. Sun. Human ocr: Insights from a complex human computation process. In *Workshop on Crowdsourcing and Human Computation, Services, Studies and Platforms, ACM CHI*, 2011.
[24] E. Livshits, B. Kimelfeld, and S. Roy. Computing optimal repairs for functional dependencies. In *PODS*, pages 225–237, 2018.
[25] C. Mayfield, J. Neville, and S. Prabhakar. ERACER: a database approach for statistical inference and data cleaning. In *SIGMOD*, pages 75–86, 2010.
[26] G. Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, 2001.
[27] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer New York, 2006.
[28] Y. Noh, F. C. Park, and D. D. Lee. Diffusion decision making for adaptive k-nearest neighbor classification. In *NIPS*, pages 1934–1942, 2012.
[29] G. Optimization. Inc.,"gurobi optimizer reference manual," 2015. *URL: http://www. gurobi. com*, 2014.
[30] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543, 2014.
[31] A. Petrie and C. Sabin. *Medical statistics at a glance*. John Wiley & Sons, 2019.
[32] P. Ravikumar and W. W. Cohen. A hierarchical graphical model for record linkage. In *UAI*, pages 454–461, 2004.
[33] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré. Holoclean: Holistic data repairs with probabilistic inference. *PVLDB*, 10(11):1190–1201, 2017.
[34] S. Song et al. Data cleaning under distance constraints. *Unpublished work*.
[35] S. Song et al. Imputing various incomplete attributes via distance likelihood maximization. *Unpublished work*.
[36] A. Schrijver. *Theory of linear and integer programming*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999.
[37] S. Song and L. Chen. Differential dependencies: Reasoning and discovery. *ACM Trans. Database Syst.*, 36(3):16:1–16:41, 2011.
[38] S. Song, H. Cheng, J. X. Yu, and L. Chen. Repairing vertex labels under neighborhood constraints. *PVLDB*, 7(11):987–998, 2014.
[39] 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.
[40] G. J. J. van den Burg, A. Nazábal, and C. Sutton. Wrangling messy CSV files by detecting row and type patterns. *CoRR*, abs/1811.11242, 2018.
[41] M. Yakout, L. Berti-Équille, and A. K. Elmagarmid. Don't be scared: use scalable automatic repairing with maximal likelihood and bounded changes. In *SIGMOD*, pages 553–564, 2013.