Don't be SCAREd: Use SCalable Automatic REpairing with Maximal Likelihood and Bounded Changes

Mohamed Yakout
Microsoft Corp.
myakout@microsoft.com

Laure Berti-Équille
Institut de Recherche pour le Développement
laure.berti@ird.fr

Ahmed K. Elmagarmid
Qatar Computing Research Institute
aelmagarmid@qf.org.qa

ABSTRACT

Various computational procedures or constraint-based methods for data repairing have been proposed over the last decades to identify errors and, when possible, correct them. However, these approaches have several limitations including the scalability and quality of the values to be used in replacement of the errors. In this paper, we propose a new data repairing approach that is based on maximizing the likelihood of replacement data given the data distribution, which can be modeled using statistical machine learning techniques. This is a novel approach combining machine learning and likelihood methods for cleaning dirty databases by value modification. We develop a quality measure of the repairing updates based on the likelihood benefit and the amount of changes applied to the database. We propose SCARE (SCalable Automatic REpairing), a systematic scalable framework that follows our approach. SCARE relies on a robust mechanism for horizontal data partitioning and a combination of machine learning techniques to predict the set of possible updates. Due to data partitioning, several updates can be predicted for a single record based on local views on each data partition. Therefore, we propose a mechanism to combine the local predictions and obtain accurate final predictions. Finally, we experimentally demonstrate the effectiveness, efficiency, and scalability of our approach on real-world datasets in comparison to recent data cleaning approaches.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Data Mining; H.4 [**Information Systems Applications**]: Miscellaneous

Keywords

data cleaning, inconsistent data

1. INTRODUCTION

Data quality experts estimate that erroneous data can cost a business as much as 10 to 20% of its total system implementation budget [7]. They agree that as much as 40 to 50% of a project budget might be spent correcting data errors in time-consuming, labor-intensive and tedious processes. The proliferation of data also heightens the relevance of data cleaning and makes the problem more challenging: more sources and larger amounts of data imply larger variety and intrication of the data quality problems and higher complexity for maintaining the quality of the data in a cost-effective way. As a result, various computational procedures for data cleaning have been proposed by the database community to (semi-)automatically identify errors and, when possible, correct them.

Most existing solutions to repair dirty databases by value modification follow constraint-based repairing approaches [3, 19, 18], which search for minimal change of the database to satisfy a predefined set of constraints. While a variety of constraints (e.g., integrity constraints, conditional functional and inclusion dependencies) can detect the presence of errors, they are recognized to fall short of guiding to correct the errors; and worse, may introduce new errors when repairing the data [10]. Moreover, despite the research conducted on integrity constraints to ensure the quality of the data, in practice, databases often contain a significant amount of non-trivial errors. These errors, both syntactic and semantic, are generally subtle mistakes which are difficult or even impossible to detect and express using the general types of constraints available in modern DBMSs [20]. This highlights the need for different techniques to clean dirty databases.

In this paper, we address the issues on scalability and accuracy of replacement values by leveraging Machine Learning (ML) techniques for predicting better quality updates to repair dirty databases. Our proposed approach is complementary to existing efforts that leverage reference data [10] and user's interaction [22].

Statistical ML techniques (e.g., decision tree, Bayesian networks) can capture dependencies, correlations, and outliers from datasets based on various analytic, predictive or computational models [23]. Existing efforts in data cleaning using ML techniques mainly focused on data imputation (e.g., [20]) and deduplication (e.g., [6]). To the best of our knowledge, our work is the first approach to consider ML techniques for repairing databases by value modification.

Involving ML techniques for repairing erroneous data is not straightforward and it raises four major challenges: (1) Several attribute values (of the same record) may be dirty. Therefore, the process is not as simple as predicting values for a single erroneous attribute. This requires accurate modeling of correlations between the database attributes, which assumes that a subset is dirty and its complement is reliable. (2) A ML technique can predict an update for each tuple in the database; and the question is how to distinguish the predictions that should be applied. Therefore, a measure to quantify the quality of the predicted updates is required. (3) An over-fitting problem may occur when modeling a database with a large variety of dependencies that may hold locally for data subsets but do not hold globally. (4) Finally, the process of learning a model from a very large database is expensive, and the prediction model itself may not fit in the main memory. Despite the existence of scalable ML techniques for large datasets, they are either model

SIGMOD'13, June 22–27, 2013, New York, New York, USA.

dependent (*i.e.*, limited to specific models, for example SVM [21]) or data dependent (e.g., limited to specific types of datasets such as scientific data and documents repository). What is more, scalability is also an issue for constraint-based repairing approaches [8].

Such limitations motivate the need for effective and scalable methods to accurately predict cleaning updates with statistical guarantees. Precisely in this paper, our contributions can be summarized as follows:

- We formalize a novel data repairing approach that maximizes the likelihood of the data given the underline data distribution, which can be modeled using statistical ML techniques. The objective is to apply selected database updates that *(i)* will best preserve the relationships among the data values; and *(ii)* will introduce a small amount of changes. This approach enables a variety of ML techniques to be involved for the purpose of accurately repairing dirty databases by value modification. This way we eliminate the necessity to predefine database constraints, which requires the added expense of experts involvement. In contrast to the constraint-based data repair approaches, which find the minimum number of changes to satisfy a set of constraints, our likelihood-based repair approach finds the bounded amount of changes to maximize the data likelihood.

- One of the challenges is that *multiple* attributes values may be considered dirty. Therefore, we introduce a technique to provide predictions for multiple attributes at a time, while taking into account two types of dependencies: *(i)* the dependency between the identified clean attributes and dirty attributes; as well as, *(ii)* the dependency among the dirty attributes themselves. We present our technique by introducing the probabilistic principles which it relies upon.

- We propose SCARE (SCalable Automatic REpairing), a systematic scalable framework for repairing erroneous values that follows our approach and, more importantly, it is scalable for very large datasets. SCARE has a robust mechanism for horizontal data partitioning to ensure the scalability and enable parallel processing of data blocks; various ML methods are applied to each data block to model attributes values correlations and provide "local" predictions. We then provide a novel mechanism to combine the local predictions from several data partitions. The mechanism computes the validity of the predictions for the individual ML models and takes into account the models' reliability in terms of minimizing the risk of wrong predictions, as well as, the significance of partitions' sizes used in the learning stage. Finally, given several local predictions for repairing a tuple, we incorporate these predictions into a graph optimization problem, which captures the associations between the predicted values across the partitions and obtain more accurate, final tuple repair predictions.

- We present an extensive experimental evaluation to demonstrate the effectiveness, efficiency, and scalability of our approach on very large real-world datasets.

The rest of the paper is organized as follows: Section 2 defines the problem and introduces the notion of maximal likelihood repair. Section 3 presents our solutions for modeling dependencies and predicting accurate replacement values. Section 4 presents SCARE, our scalable solution to repair the data. We demonstrate the validity of our approach and experimental results in terms of efficiency and scalability in Section 5. We discuss related work in Section 6 and conclude the paper in Section 7.

2. PROBLEM DEFINITION AND SOLUTION APPROACH

In this section, we formalize our maximal likelihood repair problem and introduce our solution approach.

2.1 Problem Definition

We consider a database instance D over a relation schema $\mathcal{R}$ with $\mathcal{A}$ denoting its set of attributes. The domain of an attribute $A \in \mathcal{A}$ is denoted by $\texttt{dom}(A)$.

In the relation $\mathcal{R}$, a set $F = \{E_1, \ldots, E_K\} \in \mathcal{A}$ represents the *flexible* attributes, which are allowed to be modified (in order to substitute the possibly erroneous values), and the other attributes $R = \mathcal{A} - F = \{C_1, \ldots C_L\}$ are called *reliable* with correct values. Hence, a database tuple t has two parts: the reliable part ($t[R] = t[C_1, \ldots C_L]$), and the flexible part ($t[F] = t[E_1, \ldots E_K]$). For short we refer to $t[R]$ and $t[F]$ as r and f, respectively (*i.e.,* $t = rf$). Note that the detection of the erroneous records is not the scope of our paper. We assume that it is possible to identify a subset $D_c \subset D$ of clean (or correct) tuples and $D_e = D - D_c$ represents the remaining *possibly* dirty tuples. This distinction does not have to be accurate in specifying the dirty records, but it should be accurate in specifying the clean records. Our objective is to learn from the correct tuples in D_c to predict accurate replacement values for the possibly dirty tuples in D_e.

There are various techniques to distinguish D_c as it is always possible to use reference data and existing statistical techniques (e.g., [17, 23]), as well as, database constraints (if available) to provide a score $P_e(t) \in [0..1]$ for each database tuple t for being erroneous. Applying a conservative threshold on the scores of each tuple $P_e(t)$, we can select high quality records to be used for training.

Example 1: Consider the example relation in Figure 1 with a sample of 8 tuples about some personal information: `Name`, `Institution`, area code `AC`, telephone number `Tel`, in addition to address information: `City`, `State` and `Zip`.

	Name	Institution	AC	Tel	City	State	Zip
t_1	H. Garcia-Molina	Stanford Univ.	650	723-0685	Santa Clara	CA	94305
t_2	S. Madden	MIT	617	258-6643	Cimradge	MA	02139
t_3	J. Han	UIUC	217	333-6903	Chicago	IL	61801
t_4	C. Clifton	Purdue Univ.	765	494-6005	Lafayette	IN	47906
t_5	W. Aref	Purdue Univ.	765	494-1997	WLafayette	IN	47907
t_6	E. Bertino	Purdue Univ.	765	496-2399	WLafayette	IN	47907
t_7	J. Widom	Stanford Univ.	650	723-7690	Stanford	CA	94305
t_8	M. Stonebraker	MIT	617	253-3538	Cambridge	MA	02139

Figure 1: Illustrative example

This data is a result of integrating professional contact information and lookup address database. Due to the integration process, we know that some of the address attributes (`City`, `State` and `Zip`) may contain errors. Therefore, we call the address attributes *flexible* attributes. After the integration process, we could separate high quality records by consulting other reference data or verifying some widely known relationships among the attributes. For this example, tuples $t_5, \ldots, t_8 \in D_c$ are identified as correct ones, while we are not sure about tuples $t_1, \ldots, t_4 \in D_e$. □

We introduce the data repair likelihood given the data distribution as a technique to guide the selection of the updates to repair the dirty tuples. Our approach is different from the maximum likelihood approach to learn a ML model parameter. In that case, the model's parameters and the captured distribution changes according to the observed data. In our case instead, we learn initially the data distribution from the set of clean tuples; and then, we change the dirty tuples *carefully* to maximize the data agreement with the learnt distribution. Our hypothesis is that the more the update will

make the data follows the underline data distribution with least cost, the more likely the update to be correct.

Note that the tuples usually come to exist in the database independently (e.g., new customer records are inserted in the database independent from other customers information); errors in the tuples can be detected because of the existing values correlations among the database attributes that hold across the tuples. Such values correlations can be learnt using a ML model; and hence, the correctness probability of the tuples is conditionally independent given the learnt model. Consequently, the likelihood of the database D is the product of the tuples' probabilities given a probability distribution for the tuples in the database. Given the identified clean subset of the database D_c, we can model the probability distribution $P(R, F)$. Then, the likelihood of the possibly erroneous subset D_e can be written (as log likelihood):

$$L(D_e|D_c) = \sum_{t \in D_e} \log P(t \mid D_c) = \sum_{t=rf \in D_e} \log P(f \mid r) \quad (1)$$

where we use $P(t \mid D_c) = P(f \mid r)$, which we discuss in Section 3.

Assuming for a given tuple $t = rf$ a ML technique predicted f' instead of f. We say that the update u is predicted to replace f by f'. Applying u to the database will change the likelihood of the data; we call the amount of increase in the data likelihood given the data distribution as the likelihood benefit of u.

DEFINITION 1. ***Likelihood benefit of an update*** u ***($l(u)$):*** *Given a database $D = D_c \cup D_e$, $t = rf \in D_e$ and an update u to replace f by f', the likelihood benefit of u is the increase in the database likelihood given the data distribution learnt from D_c, or $(L(D_e^u|D_c) - L(D_e|D_c))$, where D_e^u refers to D_e when the update u is applied. Using Eq. 1 we obtain:*

$$l(u) = \log P(f' \mid r) - \log P(f \mid r). \quad (2)$$

We also define the cost of an update as follows:

DEFINITION 2. ***Cost of an update*** u ***($c(u)$):*** *For a given database tuple $t = rf$ and an update u to replace f by f', the cost of u is the distance between f and f',*

$$c(u) = \sum_{E \in F} d_E(f[E], f'[E]) \quad (3)$$

where $d_E(f[E], f'[E])$ is a distance function for the value domain of attribute E that returns a score between 0 and 1. Examples of distance functions for string attributes include the normalized Edit distance or Jaro coefficient; for numerical attributes, the normalized distance can be used, e.g., $d_E(a, b) = \frac{|a-b|}{max_E - min_E}$, where a and b are two numbers in $\mathtt{dom}(E)$, and max_E, min_E are the maximum and minimum values in $\mathtt{dom}(E)$ respectively.

Our objective is to modify the data to maximize its likelihood; however, and similar to existing repairing approaches, we need to be conservative in modifying the data. Therefore, we bound the amount of changes introduced to the database by a parameter δ. Hence, the problem becomes: given an allowed amount of changes δ, how do we best select the cleaning updates from all the predicted updates? This is a constrained maximization problem where the objective is to find the updates that maximizes the likelihood value under the constraint of a bounded amount of database changes, δ. We call this problem the "Maximal Likelihood Repair".

DEFINITION 3. ***Maximal Likelihood Repair:*** *Given a scalar δ and a database $D = D_e \cup D_c$. The Maximal Likelihood Repair problem is to find another database instance $D' = D'_e \cup D_c$, such that $L(D'_e \mid D_c)$ is maximum subject to the constraint $Dist(D, D') \leq \delta$.*

where $Dist$ is a distance function between the two database instances D and D' before and after the repairing and it can be defined as $Dist(D, D') = \sum_{\forall t \in D, A \in \mathcal{A}} d_A(t[A], t'[A])$, where $t' \in D'$ is the repaired tuple corresponding to tuple $t \in D$.

Note that δ can take values from 0 (*i.e.,* no changes to the data is allowed) to $|D_e| \times |F|$ (*i.e.,* change all flexible attributes of the dirty tuples). Regarding δ estimation, it is possible to use the score $P_e(t)$, which estimates the erroneousness of tuple t, to estimate $\delta = \epsilon \sum_{t \in D_e} P_e(t)$, where $\epsilon \in [0..1]$. The idea is that a possibly erroneous tuple is expected to be modified according to its score of being erroneous. ϵ can be chosen close to zero to be more conservative to the amount of introduced changes.

2.2 Solution Approach

For each tuple $t = rf$, we obtain the prediction f' that represents an update u to t. We compute the likelihood benefit and cost of u. Finally, we need to find the subset of updates that maximizes the overall likelihood subject to the constraint that the total cost is not more than δ, *i.e.*, $Dist(D, D') \leq \delta$.

Formally, given a set $\mathcal{U}$ of updates and, for each update u, we compute $l(u)$ and $c(u)$ using Eq. 2 and 3, respectively. Our goal is to find the set of updates $\mathcal{U}' \subseteq \mathcal{U}$, such that:

$$\sum_{\forall u \in \mathcal{U}'} l(u) \text{ is maximum} \quad \text{subject to:} \sum_{\forall u \in \mathcal{U}'} c(u) \leq \delta. \quad (4)$$

This is typically a 0/1 knapsack problem setting, which implies that the maximal likelihood repair problem is NP-complete.

Heuristic and quality measure: To solve the above problem, we use the famous heuristic to solve the 0/1 knapsack problem by processing the updates in decreasing order of the ratio $\frac{l(u)}{c(u)}$. This heuristic suggests that the "correctness measure" of an update u is *the ratio of the update's likelihood benefit to the cost of applying the update to the database* (*i.e.*, the higher the likelihood benefit with small cost, the more likely the update to be correct). Empirically, this gives good predictions for the updates as we will illustrate in our experiments.

Example 2: In Figure 1, assume that two updates were predicted to the database. u_1 updates t_3 such that f'_3={"Chicago", "IL", "60614"} and u_2 updates t_4 such that f'_4={"WLafayette", "IN", "47907"}. Assume also that both of $l(u_1), l(u_2)$ have the same likelihood benefit. In this case, u_2 will encounter lower cost in updating one character in both the `Zip` and the `City` attributes (update the `Zip` from"47906" to "47907" and the `City` from "Lafayette" to "WLafayette"), while u_1 will cost updating 4 characters in the `Zip` from "61801" to "60614". Hence, for $\delta \leq 2$ characters, only u_2 will be applied to the database. □

3. MODELING DEPENDENCIES AND PREDICTING UPDATES

The key challenge when considering data repair using the data distribution is that multiple attributes values may be dirty. In the case when a single attribute is erroneous, the problem is to model the conditional probability distribution of the erroneous attribute given the other attributes; and hence, a single classification model can be used to obtain the predicted values for the erroneous attribute. However, it is mostly the case that a set of attributes have low quality values and not a single attribute. Therefore, we need to model the probability distribution of the subset of dirty attributes given the other attributes that have reliable values (*i.e.*, most likely to be correct) to achieve a better prediction of the replacement values.

Example 3: In Figure 1 assuming we know that only the `City` attribute contains some errors. This is the simple case because a ML model can be trained by the database tuples considering the `City` as the label to be predicted. However in practice, more than

one attribute can be dirty at the same time, for example, all the address attributes. In this case, we need a ML technique to model the distribution of the combination (City, State, Zip)—taking into account their possible inter-dependencies—given existing reliable values of attributes, e.g., (Name, Institution, AC, Tel). □

3.1 Modeling Dependencies

Let $\mathcal{S}_R = \texttt{dom}(C_1) \times \texttt{dom}(C_2) \cdots \times \texttt{dom}(C_L)$ denotes the space of possible reliable parts of tuples $t[C_1 \dots C_L]$ (with clean attribute values), and $\mathcal{S}_F = \texttt{dom}(E_1) \times \texttt{dom}(E_2) \cdots \times \texttt{dom}(E_K)$ denotes the space of possible flexible parts of tuples $t[E_1 \dots E_K]$ (with possibly erroneous values). Assuming that the tuples of D are generated randomly according to a probability distribution $P(R, F)$ on $\mathcal{S}_R \times \mathcal{S}_F$, $P(F|r)$ is the conditional distribution of F given $R = r$ and $P_{E_i}(E_i|r)$ is the corresponding marginal distribution of the values of attribute E_i,

$$P_{E_i}(e_i|r) = \sum_{f \in \mathcal{S}_F | f[E_i] = e_i} P(f \mid r)$$

Note that the posterior probability distribution $P(F \mid r)$ provides the means to analyze the dependencies among the flexible attributes. The distribution informs about the probability of each *combination* of values for the flexible attributes $\langle e_1, \dots, e_K \rangle$, where $e_1 \in \texttt{dom}(E_1), \dots, e_K \in \texttt{dom}(E_K)$.

Given a database tuple $t = rf$, the conditional probability of each combination of the flexible attribute values f can be computed using the product rule:

$$P(f \mid r) = P(f[E_1] \mid r) \prod_{i=2}^{K} P(f[E_i] \mid r, f[E_1 \dots E_{i-1}]). \quad (5)$$

Note that we assume a particular order in the dependencies among the flexible attributes $\{E_1, \dots, E_K\}$. To obtain this order, we leverage an existing technique [14] to construct a dependency network for the database attributes. The dependency network is a graph with the database attributes as the vertices; and there is a directed edge from A_i to A_j if the analysis determined that A_j depends on A_i. In our case, there will be two sets of vertices; the reliable set R and flexible set F. The first flexible attribute E_1 in the order is the one that has the maximum number of reliable attributes as its parents in the graph. Then subsequently, the next attribute in order i is the one with maximum number of parents that are either reliable attributes or flexible attributes with an assigned order. In our experiments, we followed this procedure by analyzing a sample of the database to determine the dependency order of the flexible attributes. Another alternative method to compute the conditional probability $P(f \mid r)$ without considering any particular order of the flexible attributes is to use Gibbs sampling [13]; however, it is very expensive to be applied to even moderate size databases. Please refer to [14] for further details.

In Section 3.2, we introduce an efficient way to obtain the predictions f' that is desirable for our cleaning approach and compute the conditional probabilities $P(f \mid r)$.

3.2 Predicting Updates

We use a ML model $\mathcal{M}$ (as predictor) to model the above joint distribution in Eq. 5. The model $\mathcal{M}$ is a mapping $\mathcal{S}_R \rightarrow \mathcal{S}_F$ that assigns (or predicts) the flexible attributes values f' for a database tuple $t = rf$ given the values r of the reliable attributes R. The prediction takes the form:

$$\mathcal{M}(r) = \langle M_1(r), \dots, M_K(r) \rangle = f'.$$

To estimate the joint distribution of the flexible attribute values, $P(f \mid r)$ in Eq. 5, we learn K classification models $M_i(\cdot)$ on the input space $\mathcal{S}_R \times \texttt{dom}(E_1) \times \cdots \times \texttt{dom}(E_{i-1})$, (*i.e.*, using all the reliable attributes and the flexible attributes up to attribute E_i).

$$M_i : \mathcal{S}_R \times \texttt{dom}(E_1) \times \cdots \times \texttt{dom}(E_{i-1}) \rightarrow \texttt{dom}(E_i)$$

We assume that M_i is a probabilistic classifier (e.g., Naïve Bayesian) that will be trained using D_c and produce a probability distribution over the values of the flexible attribute E_i given $\langle r, f[E_1], \dots, f[E_{i-1}] \rangle$.

One efficient *greedy* way to approximate the optimal prediction f' is to proceed as follows: given a tuple $t = rf$, the classifier M_1 is used to predict the value of attribute E_1 (*i.e.*, $f'[E_1]$) given r. Then, M_2 predicts the value for attribute E_2 given r and $f'[E_1]$ as input. Proceeding in this way, M_i predicts the value of attribute E_i given r and $f'[E_1] \dots f'[E_{i-1}]$. This approach can be considered as searching greedily for a path in a tree that has the possible values of $f' \in \mathcal{S}_F$ at the leaves. We call this tree as the *flexible attributes values search tree*. Needless to say this approach does not guarantee finding the prediction f' with the highest probability.

For a tuple t, to find a better prediction that is desired for our cleaning approach, we follow the conservative assumption in updating the database by considering and preferring the original attributes values in the tuple. Based on this assumption the best prediction will be among these explored tree paths, which involve the original values of the tuple t. Hence, we can compute, in addition to the greedy path, additional paths that assume that the original values in the tuple are the supposed predictions. The algorithm to compute a set of predictions for the flexible attributes F for a given tuple t is described in Algorithm 1, *GetPredictions*. Basically, *GetPredictions* proceeds recursively in the flexible attributes values search tree. At each node tree level i, two branches are considered when the prediction of attribute E_i is different from its original value in the tuple, otherwise, a single branch is considered. The initial call to Algorithm 1 to get predictions for tuple $t = rf$ is *GetPredictions(M_0, r, 1.0, $t = rf$)*.

Algorithm 1 GetPredictions(Classification Model M_i, $\langle r, f[E_1], \dots f[E_{i-1}] \rangle$ input tuple r_i, Probability P, Database Tuple $t = rf$)

```
1:  if (i > K) then
2:      f' = r_i − r
3:      AllPredictions = AllPredictions ∪{(f', P)}
4:      return
5:  end if
6:  f_{E_i} = M_i(r_i)
7:  r_s = ⟨r_i, f_{E_i}⟩
8:  P_s = P × P(f_{E_i} | r_i)
9:  GetPredictions(M_{i+1}, r_s, P_s, t)
10: if f_{E_i} ≠ t[E_i] then
11:     r'_s = ⟨r_i, t[E_i]⟩ {Adding the original E_i's value to the next input}
12:     P'_s = P × P(t[E_i] | r_i)
13:     GetPredictions(M_{i+1}, r'_s, P'_s, t) {Predicting attribute E_i's value}
14: end if
```

In Algorithm 1, Line 1 checks if we reached the prediction of the last flexible attribute E_K; and in this case, we add the flexible part f' of the obtained final tuple s to AllPredictions list. In Line 6, we predict the value f_{E_i} of attribute E_i. Lines 7 and 8 compose the new input r_s by adding f_{E_i} to r_i and compute the prediction probability so far, P_s. We then proceed recursively to get the prediction for the next flexible attribute $E_{(i+1)}$. The lines 11-13 are executed if the predicted value f_{E_i} is different from the original value $t[E_i]$. In this case, we compose another input r'_s using the original value $t[E_i]$ and compute the prediction probability so far using $P(t[E_i] \mid r_i)$ from the model M_i, then finally, proceed recursively to get a prediction for $E_{(i+1)}$.

Example 4: Consider the example relation in Figure 1. Assume

that tuple t_4 was marked as erroneous and we want to obtain predictions for its flexible attributes. In *GetPredictions* initially r_i is the reliable attributes values {"C. Clifton", "Purdue Univ.", "765", "494-6005"}. In Line 6 the classifier M_0, which was trained using only the set of reliable attributes to predict the first flexible attribute `City`, provides the prediction to be "WLafayette". Then r_s is composed to be the input to the next classifier M_1, which was trained by the reliable attributes and the first flexible attribute `City` to predict the second flexible attribute `State`, r_s = {"C. Clifton", "Purdue Univ.", "765", "494-6005", "WLafayette"}. Since the predicted `City` is different from the one in the table, we compose another input to the classifier M_1 with the original `City` value, r'_s = {"C. Clifton", "Purdue Univ.", "765", "494-6005", "Lafayette"}. We obtain the prediction for the `State` given the two inputs r_s and r'_s and proceed recursively until we used M_3 to predict the `Zip`, and we finally extract f' from each r_i in Line 2 to end up with the list of AllPredictions. □

GetPredictions produces, for a given tuple t, at most 2^K predictions with their probabilities; however, in practice, the number of predictions are far less than 2^K. We select the prediction f' with the best benefit-cost ratio, *i.e.*, the update u that replaces f with f' and results in the highest $\frac{l(u)}{c(u)}$. Note that we need to compute $l(u)$ for only f' with $P(f' \mid r)$ being greater than $P(f \mid r)$, the probability of the original values, otherwise the likelihood benefit of the predicted update will be negative. Note also that $P(f \mid r)$ is included as well in the output of *GetPredictions*.

In Section 4, we present the method to scale up the maximal likelihood repair problem and get the predicted updates u along with their likelihood benefit $l(u)$. The cost $c(u)$ is straight forward to compute.

4. SCALING UP THE MAXIMAL LIKELIHOOD REPAIRING APPROACH

One of the key challenges in repairing dirty databases is the scalability [8]. In our case, the scalability issue is mainly due to learning a set of classification models to predict the flexible attributes values. The learning process in most ML techniques is known to be at least quadratic in the database size and the model itself may not fit in main memory. Indeed, there are efforts on learning from large scale datasets (e.g., scalable learning using SVM [21]). However, there is no much efforts in learning from large databases with most of attributes are string attributes and the number of correlations is large because of the large domain size of each database attribute.

In this section, we present a *model-independent* method to learn and predict updates to the database that is based on horizontally partitioning the database. Each database tuple will be a member of several partitions (or blocks). Each partition b is processed to provide predictions to the erroneous tuples $t \in b$ depending on the database distribution learnt from block b (*i.e., local* predictions). Finally, we present a novel mechanism to combine the local predictions from the different partitions and determine more accurate final predictions.

This method is in the flavor of learning *ensemble* models [5] or committee-based approaches, where the task is to predict a single class attribute by partitioning the dataset into several smaller partitions; then a model is trained by each data partition. For a given tuple, each model provide a prediction on the class attribute, and the final prediction is the one with the highest aggregated prediction probability. But, in our case, we want to predict the values of multiple flexible attributes together; and we are not limited to predict a single attribute value. Hence for a given tuple, we obtain a prediction (a combination f' of the flexible attribute values) from each data partition. We then propose a technique to combine the models' predictions into a graph optimization problem to find the final prediction for the flexible attributes. Our main insight is that the final (combinations of) predicted values are those which would maximize the associations among the predicted values across the partitions. Our mechanism to collect and incorporate the predicted updates takes into account the reliability of the learnt classification models themselves to minimize the risk of the predicted updates.

After obtaining the final predicted values with their likelihood benefit $l(u)$, we use them into the maximal likelihood repair problem (Eq. 4).

Algorithm 2 SCARE(D dirty database, $H = \{h_1, \ldots, h_J\}$ DB partitioning functions)

```
1: Given H, partition D into blocks b_ij.
2: for all block b_ij do
3:    Learn the models M_ij.
4:    for all tuple t = rf ∈ b_ij ∧ t ∈ D_e do
5:       Use M_ij to predict f'_j and get P_ij(f'_j | r) and P_ij(f | r)
6:       Store f'_j, P_ij(f'_j | r) and P_ij(f | r) in RS. {store in the
         Repair Storage.}
7:    end for
8: end for
9: for all tuple t = xy ∈ D do
10:   RS(t) = the candidate tuple repairs for t in RS.
11:   f' =SelectFinalPrediction(RS(t))
12:   For the update u to change f to f', compute the likelihood
      measure l(u) if f ≠ f'.
13: end for
```

4.1 Process Overview

Algorithm 2 illustrates the main steps of the SCARE process to get the predicted updates along with their likelihood benefit. The primary input to the framework is a database instance D. The second input is a set of database partitioning functions (or criteria) $H = \{h_1, \ldots, h_J\}$.

There are two main phases for SCARE: (1) Updates generation phase (lines 1-8), and (2) Tuple repair selection phase (lines 9-13).

In Phase 1 (Line 1), each function $h_j \in H$ will partition D into blocks $\{b_{1j}, b_{2j}, \ldots\}$. Then, the loop in lines 2-8 processes each block b_{ij} as follows: *(i)* Learn the set of classifiers $\mathcal{M}_{ij}$ from the identified clean tuples in b_{ij} (lines 3); *(ii)* Use $\mathcal{M}_{ij}$ to predict the flexible attributes values for the possibly erroneous tuples in b_{ij} using Algorithm 1 (lines 4-7). For each tuple, the prediction is considered a *candidate tuple repair* and it is stored in a temporary repair storage, denoted as RS. Since each tuple will be a member of several data partitions, we will end up with a set of candidate tuple repairs for each possibly erroneous tuple. The details of the repair generation is provided in Section 4.2.

Phase 2 (lines 9-13) loops on each tuple $t \in D_e$ and retrieves all its candidate tuple repairs from the repair storage RS, then uses Algorithm 3 *SelectTupleRepair* to get the final tuple repair (update) with its estimated likelihood benefit. The details of the repair selection algorithm is provided in Section 4.3. Note that each iteration in Phase 1 does not depend on other iterations (similarly for the iterations of Phase 2). Hence, SCARE can be efficiently parallelized.

4.2 Repair Generation Phase

In this phase, the data is partitioned, as we will explain shortly, for two main benefits: *(i)* scale for large data by enabling independent processing for each partition; and *(ii)* more accurate and efficient learning of the classification models for the prediction task. The first benefit is obvious and the second benefit is obtained because of the following: When we train a classification model for prediction, ideally, we need the model to provide high prediction accuracy capturing all the possible dependencies from the data (we call it a model with *global view*). All the statistically significant dependencies are considered as the model's search space. How-

ever, if the space contains a lot of *weak* dependencies, most likely, the model will not be able to capture them. But if it does, the global view will not be accurate enough for prediction because of the model overfitting. Partitioning the database helps to capture local correlations that are significant within subsets of the database and that require a different degree of "zooming" to be recognized. Each of the partition functions $h \in H$ provides the search space partitioned according to the criteria shared by the tuples within the same block. If we train models on multiple blocks, we will have models with several *local views* (or specialized models, sometimes called *experts* [16]) for portions of the search space. Combining these local views, will result in a better prediction accuracy.

Partitioning the database: Each partition function or criterion $h(\cdot)$ maps each tuple to one of a set of partitions. Multiple criteria $H = \{h_1, \ldots, h_J\}$ is used to partition the database in different ways. Each tuple t is mapped to a set of partitions, *i.e.*, $H(t) = \bigcup_{\forall\ j}\ h_j(t)$.

A simple way to choose the partition criteria is **Random** (*i.e.,* randomly partition the data many times). Another way to choose the criteria is **Blocking**, where partitions are constructed under the assumption that similar tuples will fit in the same block or inversely, tuples across different blocks are less likely to be similar. Many techniques for blocking have been introduced for the efficient detection of the duplicate records (refer to [6] for a survey).

It is worth mentioning that increasing the number of partition functions will result in a more accurate final prediction, because the variance in the predictions decreases as we increase the number of ways (partition functions) to partition the data. We found that partitioning the data using different blocking techniques provided more accurate predictions with less number of partition functions in comparison with the random partitioning.

Example 5: Consider again the relation in Figure 1. In this example, one may partition the database based on the `Institution` attribute (as a partition function) to get the tuples partitioned as follows: $\{t_1, t_7\}, \{t_2, t_8\}, \{t_3\}, \{t_4, t_5, t_6\}$. The result of the learning process from these data partitions will be expert models based on the person's institution. Another function may use the `AC` or a combination of attributes. Partition functions can be designed based on a signature based scheme or clustering as we elaborate in the experimental section. □

Reliability measure and risk minimization: In order to be conservative in considering the predictions from each block b_{ij} and its model $\mathcal{M}_{ij}$, we propose a mechanism to measure the reliability of a model and adapt the obtained prediction probability accordingly to support or detract the model's predictions.

Two major components help us judging the reliability of a model $\mathcal{M}_{ij}$: (*i*) the model quality, which is classically quantified by its *loss* $\mathcal{L}(\mathcal{M}_{ij}) = \frac{1}{|b_{ij}|}\sum_{t\in b_{ij}, t\in D_c, E\in F} d_E(f[E], f'_{ij}[E])$, where $|b_{ij}|$ is the number of tuples in partition b_{ij}, E is one of the flexible attributes F, d_E is a distance function for the domain of attribute E and f'_{ij} is the prediction of Model $\mathcal{M}_{ij}$ on the flexible attributes F for the tuple $t \in b_{ij}$ and $t \in D_c$; (*ii*) the second component is the size of the block: the smaller the block is, the less reliable the predictions will be. Hence, the reliability of model $\mathcal{M}_{ij}$ can be written as:

$$Re(\mathcal{M}_{ij}) = \frac{|b_{ij}|}{|D|}\left(1 - \mathcal{L}(\mathcal{M}_{ij})\right). \quad (6)$$

Finally, the prediction probabilities obtained from model $\mathcal{M}_{ij}$ are scaled to be: $\tilde{P}_{ij}(f' \mid r) = P_{ij}(f' \mid r) \times Re(\mathcal{M}_{ij})$.

Aggregating Suggestions: As mentioned earlier, a tuple $t = rf$ will be a member of $|H(t)|$ data partitions. From each partition, we get a candidate tuple repair for t, which are then stored in the storage RS with the following schema: $\{t\_id, partition, E_1, ..., E_K, \tilde{P}(f'|r), \tilde{P}(f|r)\}$, where t_id is the original tuple identifier, $partition$ is the partition name, $E_k \in F$,

RS(t_4)	City	State	Zip	P($f'|r$)	P($f|r$)
f'_1	WLafayette	IN	47907	0.7	0.6
f'_2	lafytte	IN	47906	0.6	0.5
f'_3	WLafayette	IN	47906	0.4	0.3
f'_4	Lafayette	IN	47907	0.8	0.6
f'_5	Lafayette	IL	47906	0.5	0.5

Figure 2: Generated predictions for tuple repairs with their corresponding prediction probabilities for tuple t_4 in Figure 1.

$\tilde{P}(f' \mid r)$ is the prediction probability of the repairing update f', and $\tilde{P}(f \mid r)$ is the probability of the original values in t. The space required for RS is of $O(|D| \times |H|)$.

Example 6: Consider tuple t_4 in Figure 1 and the flexible attributes are `City` and `State`, `Zip`. Assume that we used 5 partition functions and hence t_4 was a member of 5 partitions, consequently, we obtain 5 possible candidate tuple repairs. The table in Figure 2 illustrates the candidate tuple repairs of t_4, RS(t_4), with the corresponding prediction probabilities obtained from each partition. □

4.3 Tuple Repair Selection Phase

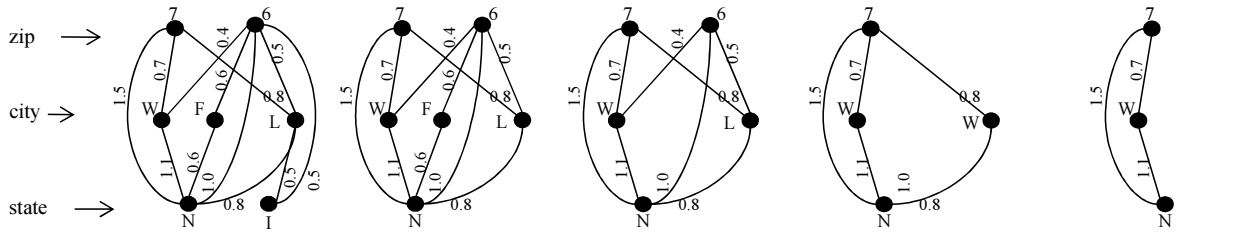
Once the candidate tuple repairs are generated, we need a repair selection strategy to pick the best one among the candidate set. One suggestion for a selection strategy can be the *majority voting*. For a tuple t, the majority voting can be done by selecting the most voted value from the partitions on each attribute E_i individually.

Majority Voting (MV): The majority voting strategy implies the assumption that each attribute was predicted independently from the others. For a tuple $t = rf$, we predict the combination of the flexible attributes f' together. Thus, the independence assumption of the attributes is not valid. Therefore, we propose a mechanism to vote for a final combination of the flexible attributes that takes into account the dependencies between the predicted values obtained from each partition.

Example 7: Consider the candidate tuple repairs of t_4 in Figure 2. Note that if we use the majority voting while using the prediction probability as the voter's certainty, the final prediction would be {"Lafayette", "IN", "47906"}. This solution does not take into account the dependencies between the predicted values within the same tuple repair. For example, there is a stronger association between "47907" and "IN" than between "47906" and "IN". This relationship is reflected on their corresponding prediction probabilities. The values "47907" and "IN" were predicted in f'_1, f'_4 with probabilities 0.7 and 0.8, while "47906" and "IN" were predicted in f'_2, f'_3 and their probabilities are smaller, 0.4 and 0.6. The same applies for the relationship between "WLafayette" and "IN", which have a stronger relationship than "Lafayette" and "IN". A more desired prediction will be {"WLafayette", "IN", "47907"}. □

For a given database tuple $t = rf$, our goal is to find the final combination $f'^* = \langle e^*_1, \ldots, e^*_K\rangle$ such that $\sum_{b_{ij}, t\in b_{ij}} P(f'^* \mid r)$ is maximum. This requires the computation of the probability of each possible combination of the flexible attribute in each data block. Instead, we can search for the values that can maximize all the pairwise joint probabilities. In principle, if we maximize the pairwise association between the predicted values, then this implies maximizing the full association between the predicted values. Hence, the final update is the one that would maximize the prediction probabilities for each pair of attribute values. We formalize this problem as follows:

DEFINITION 4. *The Tuple Repair Selection Problem: Given a set of predicted combination for the flexible attributes RS(t)= $\{f'_1, \ldots, f'_{|H|}\}$ for database tuple $t = rf$ along with the prediction probabilities of each combination, (i.e., for $f'_j \in$RS(t), we have the*



(a) Constructed Graph (b) After removing I (c) After removing F (d) After removing 6 (e) After removing W

Figure 3: Step by step demonstration for the SelectTupleRepair algorithm. At each iteration, the vertex with minimum weighted degree is removed as long as it is not the only vertex in its corresponding vertex set.

corresponding prediction probabilities $P(f'_j \mid r)$), the tuple repair selection problem for tuple t is to find $f'^* = \langle e^*_1, \ldots, e^*_K \rangle$ such that the following sum is maximum

$$\sum_{\forall\ e^*_i,\ e^*_k,\ i \neq k} \quad \sum_{\forall\ f' \in RS(t), e^*_i = f'[E_i],\ e^*_k = f'[E_k]} p(f' \mid r).$$

Solving the Tuple Repair Selection Problem: To find a solution, we map this problem to a graph optimization problem for finding the K-heaviest subgraph (KHS) [2] in a K-partite graph (KPG). The key idea is to process each database tuple t individually and use its set of candidate tuple repairs, RS(t), to construct a graph, where each vertex is an attribute value, and an edge is added between a pair of vertices *iff* the corresponding values co-occur in a prediction $f' \in$RS(t). The edges will have a weight derived from the obtained prediction probabilities. It is worth noting that this strategy is applied for each tuple on separate, therefore, this phase can be efficiently parallelized.

Finding KHS in KPG: The K-heaviest subgraph (KHS) problem is an NP optimization problem [2]. In an instance of the KHS problem, we are given a graph $G = (V_G, E_G)$, where V_G is the set of vertices of size n, E_G is the set of edges with non-negative weights (W_{wv} denotes the weight on the edge between vertices w, v), and a positive integer $K < n$.

The goal is to find $V' \subset V_G$, $|V'| = K$, where $\sum_{(w,v) \in E_G \cap (V' \times V')} W_{wv}$ is maximum. In other words, the goal is to find a K-vertex subgraph with the maximum weight.

A graph $G = (V_G, E_G)$ is said to be K-partite if we can divide V_G into K subsets $\{V_1, \ldots, V_K\}$, such that two vertices in the same subset can *not* be adjacent. We call *KHS in KPG problem*, the problem of finding KHS in a K-partite graph such that the subgraph contains a vertex from each partite.

DEFINITION 5. ***The KHS in KPG problem.*** *Given a K-partite graph $G = (V_1, \ldots, V_K, E_G)$, find $V' = \{v_1, \ldots, v_K\}$ such that $v_k \in V_k$ and $\sum_{(v_i,v_j) \in E_G \cap (V' \times V')} W_{v_i v_j}$ is maximum.*

LEMMA 1. *The KHS in KPG is NP-Complete.*

PROOF. This is a proof sketch. It is straight forward to see that we can reduce the problem of finding K-Clique (Clique of size K) in K-partite graph to the KHS in KPG problem. The problem of K-Clique in K-partite graph G is NP-complete by reduction from the problem of $(n-K)$-vertex cover in the complement K-partite graph G', which is NP-Complete (see [15] for details). □

Mapping the Tuple Repair Selection Problem to the KHS in KPG problem (KHSKPG): Given a set of predictions RS(t)= $\{f'_1, \ldots, f'_{|H|}\}$ for the flexible attributes of a tuple $t = rf$, where $f'_j = \langle e^{(j)}_1, \ldots, e^{(j)}_K \rangle$ and $K = |F|$ is the number of flexible attributes.

The repair selection problem can be mapped to the KHS in KPG problem using the following steps:

1. *Building vertex sets for each attribute E_k:* For each attribute E_k, create a vertex v for each distinct value in $\{e^{(1)}_k, \ldots, e^{(|H|)}_k\}$. Note that we have a set of vertices for each attribute E_k (*i.e.*, partite).

2. *Adding edges:* Add an edge between vertices v, w when their corresponding values co-occur in a candidate tuple repair. Note that v, w can not belong to the same vertex set.

3. *Assign edge weights:* For an edge between v, w, the weight is computed as follows: Let $f_{(v,w)} = \{f'_j | f'_j$ contains both $v, w\}$, *i.e.*, the set of predictions that contain both the values v, w.

$$W_{vw} = \sum_{f'_j \in f_{(v,w)}} P_{ij}(f'_j \mid r)$$

where $P_{ij}(f'_j \mid r)$ is the prediction probability of f'_j obtained from partition b_{ij}.

The graph construction requires a single scan over the predictions RS(t)= $\{f'_1, \ldots, f'_{|H(t)|}\}$, hence, it is of $O(K\ |H|)$. The number of vertices is the number of distinct values in the candidate tuple repairs.

Example 8: Figure 3(a) shows the constructed 3-partite graph from the predictions in Figure 2 for tuple t_4 in the original relation of Figure 1. For each attribute, there is a vertex set (or partite), e.g., the corresponding set of the `Zip` attribute contains {"47906", "47907"}. In the graph, we replaced the actual attributes values by a character abbreviation to have a more compact graph as follows: {"6" → "47906", "7" → "47907", "L" → "Lafayette", "W" → "Wlafayette", "F" → "lafytte", "N" → "IN", "I" → "IL"}.

Note that there is an edge between "W" and "N" with edge weight of 1.1 (= 0.4 + 0.7). This is because "WLafayette" and "IN" co-occur twice in f'_1 and f'_3 and their probabilities are 0.7 and 0.4 respectively. Also, there is an edge between "I" and "6" with weight of 0.5, because "IL" and "47906" co-occur once in f'_5 with probability 0.5. Similarly, the rest of the graph is constructed. □

Finally, finding the KHS in the constructed KPG is a solution to the tuple repair selection problem. The underlying idea is that the resulting K-subgraph $G'(V', E')$ will contain exactly a single vertex from each vertex set. This corresponds to selecting a value for each flexible attribute. Moreover, the weight of the selected subgraph corresponds to the result of maximizing the summation in Definition 4.

Computing the likelihood benefit: For a tuple $t = rf$, the solution of the KHS in KPG problem is the final prediction f' for the flexible attributes. The final prediction probability of f' is computed from the solution graph $G'(V', E')$ by

$$P(f' \mid r) = \frac{1}{|E'|} \sum_{e_{vw} \in E'} \frac{1}{|f_{(v,w)}|} \sum_{f'_j \in f_{(v,w)}} P_{ij}(f'_j \mid r).$$

The inner summation averages the probability of each pair of attribute values (*i.e.*, each edge in G') in the final prediction f'. The

outer summation averages the probability over all the edges in the final graph G'.

The prediction probability of the original values in the flexible attribute f is computed following the ensemble method by averaging the obtained probability from each partition, *i.e.*, $P(f \mid r) = \frac{1}{|H|}\sum_{b_{ij}, t \in b_{ij}} P_{ij}(f \mid r)$.

Finally, for the update u changing f into f', we can compute the likelihood benefit $l(u)$ using Equation 2.

Example 9: Consider the constructed initial graph in Figure 3(a). Assuming that the solution for KHS in KPG is the subgraph {"W", "N", "7"} shown in Figure 3(e). Now, we have an update u to change the original values in tuple t_4 in Figure 1 from f ={"Lafayette", "IN", "47906"} to f' ={"WLafayette", "IN", "47907"}. From the RS(t_4) in Figure 2, we get $P(f \mid r) = avg\{0.6, 0.5, 0.3, 0.6, 0.5\} = 0.5$. For $P(f' \mid r) = \frac{1}{3}\left[\frac{1}{2}(0.7 + 0.4) + \frac{1}{2}(0.7 + 0.8) + 0.7\right] = 0.66$. Finally, we can use Eq. 2 to compute $l(u) = 0.12$. □

4.4 Approximate Solution for Tuple Repair Selection

For the general problem of finding the KHS, many approximate algorithms were introduced (e.g., [1, 2, 12]). For instance, in [1] the authors model the problem as a quadratic 0/1 program and apply random sampling and randomized rounding techniques resulting in a polynomial-time approximation scheme, and in [12], the algorithm is based on semi-definite programming relaxation.

If K is very small, then the optimal solution can be found by enumeration. For the case where K is not very small, we provide here an approximate solution that is inspired by the greedy heuristic discussed in [2]. For the general case graph problem, the heuristic repeatedly deletes a vertex with the least weighted degree from the current graph until K vertices are left. The vertex weighted degree is the sum of weights on the edges attached to it.

In the following, we follow the same heuristic for the case of K-partite graph. However, we iteratively remove the vertex with least weighted degree as long as it is *not* the only vertex left in the partite, otherwise, we find the next least weighted degree vertex. The algorithm is a greedy 2-approximation following the analysis discussed in [2].

Algorithm 3 SelectTupleRepair($G(V, E)$ graph, $\mathcal{S} = \{S_1, \ldots, S_K\}$)

```
1: while ∃S ∈ S s.t. |S| > 1 do
2:    v =GetMinWeightedDegreeVertex(G, S)
3:    If v = null Then break;
4:    for all vertex w ∈ V s.t. e_wv ∈ E do
5:       Remove e_wv from G.
6:       Weighted_Degree(w) - = W_wv
7:    end for
8:    Remove v from its corresponding set S.
9: end while
```

Algorithm 3 shows the main steps to find the final tuple repair. There are two inputs to the algorithm: *(i)* the constructed graph $G(V_G, E_G)$ from the predictions; and *(ii)* the sets of vertices $\mathcal{S} = \{S_1, \ldots, S_K\}$, where each S_k represents the predicted values for attribute E_k. We store for each vertex v its current weighted degree in Weighted_Degree(v)=$\sum_{\forall e_{vw} \in E_G} W_{vw}$, which is the sum of the edges weights that are incident to v.

The algorithm proceeds iteratively in the loop illustrated in lines 1-9. The loop stops when a solution is found, where there is only one vertex in each vertex set, *i.e.*, $|S| = 1\ \forall S \in \mathcal{S}$. In each loop iteration, we start (Line 2) by finding the vertex v that has the minimum weighted degree using the Algorithm GetMinWeightedDegreeVertex($G, \mathcal{S}$). Then, we remove all the edges incident to v and update the WeightedDegree(w) by and subtracting W_{wv}, where w was connected to v by the removed edge e_{wv} (Lines 4-7). Finally, vertex v is removed from G and from its corresponding vertex set in Line 8.

GetMinWeightedDegreeVertex goes through the vertex sets that has more than one vertex and returns the vertex that has the minimum weighted degree.

Analysis: Algorithm 3 requires: First, visiting all n vertices to remove them except for K ones. For each vertex, each set $S \in \mathcal{S}$ of the K sets is visited to get its minimum vertex according to the weighted degree. This requires $O(nK \log |S|)$, where $n \approx O(K|H|)$ and $|S|$'s worst case is $O(|H|)$. Hence, visiting the vertices is of $O(K^2|H| \log |H|)$. Second, removing the vertices requires visiting their edges, $O(|E_G|)$, which has a worst case of $O(K^2|H|)$. Then, the overall complexity of Algorithm 3 is $O(K^2|H| \log |H|)$.

Example 10: The SelectTupleRepair algorithm is illustrated step-by-step in Figure 3. The algorithm looks for the vertex with the least weighted degree to be removed. The first vertex is "I", which has a weighted degree equal $1.0 = 0.5 + 0.5$, corresponding to the two incident edges in "I". This leaves the vertex set of the `State` attribute with only one vertex, "N". Therefore, we do not consider removing the vertex "N" in further iterations of the algorithm. The next vertex to remove is "F" to get Figure 3(c), and so on.

Finally, we got the final solution in Figure 3(e), which corresponds to a subgraph with 3 vertices —there is a vertex from each initial partite. This graph is the heaviest subgraph of size 3 (*i.e.*, the sum of the edges weight is the maximum), where each vertex belongs to a different partite. It is worth mentioning that the final graph does not have to be fully connected. Thus, the final prediction is {"WLafayette", "IN", "47907"}. □

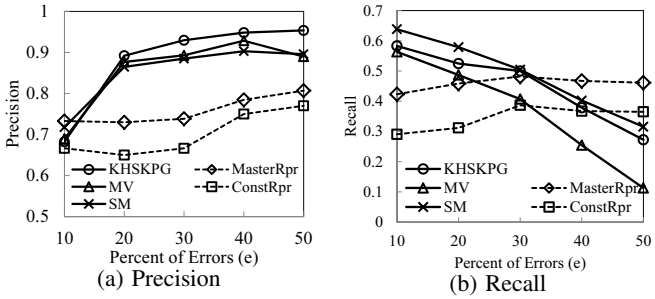
5. EXPERIMENTS

In this section, we evaluate our data repair approach; specifically, the objectives of the experiments are as follows: (1) Quality evaluation of SCARE and the notion of maximal likelihood repair in comparison with the constraint-based repairing approaches of [3] and [11], the single model approach, and majority voting; (2) Comparison between SCARE and ERACER [20] for missing values prediction; (3) Study of SCARE parameters varying the number of changes, SCARE iterations and partition functions; (4) Assessment of the scalability of SCARE.

Datasets: In our evaluations, we use three datasets: (*i*) Dataset 1 is a real-world dataset obtained by integrating (anonymized) emergency room visits from 74 hospitals. This is the dataset we use for most of the quality experiments. Since such data is coming from several sources, a myriad of data quality issues arose due to the different health care information systems used by these hospitals and the different operators responsible for entering the data. We selected a subset of the available patient attributes, namely `Patient ID`, `Age`, `Sex`, `Classification`, `Complaint`, `HospitalName`, `StreetAddress`, `City`, `Zip`, `State` and `VisitDate`. This is in addition to the `Longitude` and `Latitude` of the address information. (*ii*) Dataset 2 is the US Census Data (1990) Dataset[1] containing about 2 M tuples. It has been used only in the scalability experiments. (*iii*) Dataset 3 is the Intel Lab Data (ILD[2]) used to evaluate SCARE for predicting missing values and compare it to ERACER [20], as a recent system that relies on relational learning for predicting missing data in relational databases.

Setup: To evaluate the quality of the repairing techniques in the first comparative study, we manually cleaned the dirty Dataset 1: we used addresses web sites and repositories, external reference

[1]http://archive.ics.uci.edu/ml/datasets/US+Census+Data+ (1990)
[2]http://db.csail.mit.edu/labdata/labdata.html.



(a) Precision (b) Recall

Figure 4: Quality vs. the percentage of errors: SCARE maintains high precision by making the best use of δ, the allowed amount of changes in Dataset 1.

data sources, and visual inspection to obtain the clean dataset considered as the Golden Standard. We selected a set of attributes to be the flexible ones where we injected an increasing percentage of errors introduced in multiple attributes of the records. The flexible attributes of Dataset 1 that we considered for error injection and repairing were (`City`, `Zip`, `HospitalName`, `Longitude` and `Latitude`). Then, we compared the clean version of Dataset 1 (Golden Standard) with the repaired dataset outputs of each method (including ours).

Parameters: In our evaluations, we study several parameters that we list here with their assigned default values: (1) e: the percentage of the erroneous tuples in the dataset (default 30%), (2) d: the dataset size (default 10,000 tuples), (3) δ: the maximum amount of changes, as a fraction of d, the dataset size that SCARE is allowed to update (default 0.1 or 10% of d). (4) I: the number of iterations to run SCARE (default 1). (5) $|H|$: the number of partition functions (default 5).

Regarding the partition functions using blocking, we repeat the following process $|H|$ times: we randomly sample from the dataset a small number of tuples to be clustered in $\frac{|D|}{n_b}$ clusters, where n_b is the average number of tuples per partition. Then, each tuple is assigned to the closest cluster as its corresponding partition name. This process allows for having different blocking functions due to the random sample of tuples used in the clustering step of each iteration. The tuples that have been assigned to the same partition have common or similar features due to their assignment to the closest cluster. In all our quality experiments, we use blocking as the technique to partition the dataset. Another simple way to partition the dataset is to use random partitioning functions. In this case, given n_b, we assign each tuple to a partition name b_{ij}, where i is a random number from $\{1, \ldots, \frac{|D|}{n_b}\}$, and $j = 0$ initially. This process is repeated $|H|$ times while incrementing j each time.

All the experiments were conducted on a server running Linux with 32 GB RAM and 2 processors each with 3 GHz speed. We use MySQL to store and query the tuples. For the probabilistic classifiers, we use the Naïve Bayesian, specifically, we use the NBC WEKA implementation[3] with the default parameters settings. Java is used to implement our approach and we use Java Threads to benefit from a multiprocessor environment.

5.1 Quality Evaluation: Comparison with Constraint Based Approaches

In the following experiments, we use the standard precision and recall to measure the quality of the applied updates for string attributes.

The precision is defined as the ratio of the number of values that have been correctly updated to the total number of values that were updated, while the recall is defined as the ratio of the number of values that have been correctly updated to the number of incorrect values in the entire database. For numerical attributes, we use the mean absolute error (MAE): $\frac{1}{N}\sum_{i=1}^{d}|v_i - a_i|$ where v_i is the suggested value by SCARE, a_i is the actual value of the original data. We can compute these values since we know the ground truth for Dataset 1.

We report the quality results for four approaches:

- *KHSinKPG*: This is SCARE with the described tuple repair selection strategy as described in Section 4.3.
- *MV*: In this approach, SCARE uses directly the majority voting to select an attribute value from the candidate tuple repairs. We include this approach in the evaluation to compare our tuple repair selection strategy to the straight forward majority voting strategy.
- *SM*: The single model approach, where the whole dataset is considered as a single partition. Afterward, the likelihood benefit and cost are computed to select the best updates. This approach is included to show the advantages of combining several models with local views rather than using a single model with global view on the data.
- *ConstRepair*: One of the recent constraint-based repair approaches. We implemented (to the best of our understanding) the technique described in [3], which uses CFDs as constraints to find dirty tuples and derive repairs. We manually compiled a list of CFDs during the process of cleaning Dataset 1. Moreover, we implemented a CFD discovery algorithm [9] to be used as input to the repairing algorithm. To have a high quality rules and be fair to this approach, we discovered CFDs from the original "correct" data, which can not be the case in a realistic situation as we usually start with dirty data. We specified the rules support threshold to be 1%. Moreover, we assigned the attributes values correctness score following the technique described in [3]. The implementation to discover the CFDs is very time-consuming, therefore, this approach does not show up in all of our plots.
- *MasterRpr*: This is a recent data repair approach described in [11] that rely on a master clean data in addition to Matching Dependency (MD) rules as well as CFD rules. Since we split the database into D_c and D_e, we can use D_c as the master data. Also, we derived some MDs from the CFDs. More information on the MDs and how it can be used to clean the data is found in [11].

We use Dataset 1 and report in Figure 4 the precision and recall results for the applied updates while changing e, the percentage of tuples with erroneous values, from 10 to 50%. Generally, the approaches that maximize the data likelihood substantially outperform the constraint-based approach for the precision. Moreover, for the recall the likelihood approaches outperform the constraint-based approach when the errors is up to 30%, afterwards, the recall is comparable for all the approaches when errors is more than 30%.

SCARE with $KHSinKPG$ shows the highest precision. The precision increases using the three likelihood-based approaches with the increase in the amount of errors, but the recall is decreasing. For the `Longitude` and `Latitude` attributes, SCARE-based approaches ($KHSinKPG$ and MV) corrected these attributes with error rate between 1% and 5%.

For the SCARE-based approaches, the precision increases because we use a fixed δ. When the amount of errors in the data (noted e) is small (10 to 20%), SCARE-based approaches modify more data that allow for less accurate updates resulting in less precision and relatively high recall. As e increases, the data needs more updates to correct it, however, SCARE applies fewer, yet more accurate updates, and hence the precision increases, but the recall

[3]NBC WEKA available at http://www.cs.waikato.ac.nz/ml/weka

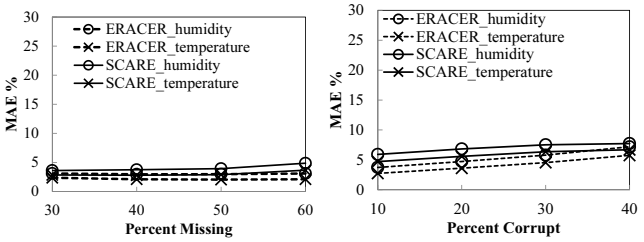


Figure 5: Comparison between SCARE and ERACER to predict missing values. Generally, both SCARE and ERACER show high accuracy in predicting the missing values. SCARE uses in this experiment Naïve Bayesian model, while ERACER leverages domain knowledge interpreted in carefully designed Bayesian Network.

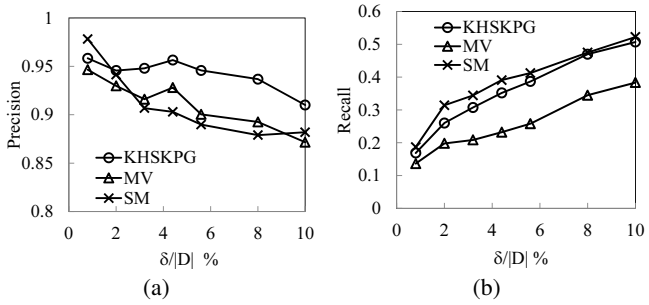


Figure 6: δ controls the amount of changes to apply to the database: small δ guarantees high precision at the cost of the recall and vice versa.

decreases (Figure 4(b)). The $KHSinKPG$ approach outperforms MV approach, because $KHSinKPG$ takes into account further associations between the predicted values across the data partitions. These associations are ignored in the MV approach. Both SCARE-based approaches that rely on data partitioning ($KHSinKPG$ and MV) show a comparable, and sometimes even better, accuracy than that of the SM approach.

The $ConstRepair$ and $MasterRpr$ have been outperformed by all the likelihood-based approaches. The $ConstRepair$ relies on the heuristics of finding values replacement that are close to the original data without considering any information on the data distributions and relationships. Moreover, the discovered CFDs that were used in $ConstRepair$ were not exhaustive enough to cover all the problems in the data and this explains the low recall in addition to the low precision. The $MasterRpr$ shows an improved performance over the $ConstRepair$ because, in addition to the CFDs, the $MasterRpr$ relies on a master clean data which helps in selecting the right replacement values through the MD rules. Therefore, both the precision and recall of $MasterRpr$ are better than those for the $ConstRepair$. But, the precision and recall are still low because, similarly with $ConstRepair$, the discovered CFDs and MDs are not exhaustive enough to cover the data problems. In contrast, SCARE relies on statistical ML techniques, which are able to cope with variable and multiple problems in the data and still rely on the data distributions to accurately predict the replacement values.

The recall of all the repairing approaches is in the range of 30 to 65%. However, for the likelihood-based approaches, the recall can be improved by running the approach again over the resulting database instance. We illustrate this improvement later in the experiment of Figure 7. But, the $ConstRepair$ approach achieves about 35% recall that can not be further improved given a fixed set of constraints.

To conclude, in comparison to the constraint-based repairing approaches, which demonstrated both low precision and recall, our likelihood based approach demonstrated accurate updates with high precision. Moreover, partitioning the data and combining the different predictions across the partitions provide more accurate predictions, because partitioning allows to learn data relationships at different granularity levels (local and global).

5.2 SCARE vs ERACER to predict missing values

In this experiment, we use Dataset 3, which includes a number of measurements taken from 54 sensors once every 31 seconds. It contains only numerical attributes. We include this dataset to evaluate SCARE for predicting missing values and compare it to ERACER. We used the same dataset with the same introduced errors as reported in ERACER evaluation of [20]. Thus, we were able to compare the effectiveness results of ERACER based on the same dataset. The efficiency of ERACER was not demonstrated in [20] and it requires domain expertise unlike our approach.

ERACER is a recent machine learning technique for data cleaning; it is specifically dedicated to replace missing values but it can not be used for data repair by value modification. ERACER leverages domain expert knowledge about the dataset to design a Bayesian network. Then, ERACER uses an underlying relational database design to store all constructed model's parameters.

In Figure 5, we evaluate SCARE for predicting the missing values in comparison with ERACER. Here, we do not use δ as SCARE is not updating an existing database value. Instead, we consider all the predictions obtained from SCARE that fill a missing value.

Figure 5(a) reports the mean absolute error (MAE) while errors in the dataset are in the form of missing values. Figure 5(b) reports also the MAE, while errors are in the form of corrupting values, e.g., adding random values. More details on how this data was corrupted is provided in [20].

There are two major numerical attributes in the Dataset 3: humidity and temperature. In Figure 5(a), both SCARE and ERACER predict the missing values for the humidity and temperature with almost the same low error percentage (2 to 4 %); and also in Figure 5(a), they both behave similarly when increasing the percentage of corrupted data, and in this case, the error percentage in prediction is between 3 and 7 %.

These numbers show that both techniques provide high accuracy predictions. However, SCARE does not require the expensive domain expert to design a Bayesian network as for ERACER. For this experiment, SCARE used the Naïve Bayesian for the statistical models. The Naïve Bayesian did well when plugged into SCARE in comparison to the Bayesian network which has to be carefully designed for ERACER. This is thanks to the partitioning technique used in SCARE, which enables several local views of the data that are then combined at the end to obtain the most reliable global view for accurate predictions. Moreover, SCARE can benefit from carefully designed learning techniques, like ERACER, and plug it in the learning step to get more accurate predictions.

5.3 Parameters Study

Quality vs. the amount of changes δ: In this experiment, we study the effect of bounding δ, the number of changes fixed by the user on the repair quality. We report in Figure 6 the resulting precision and recall when $\frac{\delta}{|D|}$ changes from 1% to 10% (e.g., if $\frac{\delta}{|D|} = 5\%$, then SCARE can change up to 10% of the tuples by replacing a selected attribute value v by v' with distance $d(v, v') = 0.5$). Generally, low values of δ guarantee high precision and inversely. Increasing δ gives more chance to SCARE to modify the data. Hence, the recall increases as we increase δ, but updates with a lower confidence could be made. This justifies the decrease in the precision when we increase δ.

The conclusion from this experiment is that small δ guarantees high precision at the cost of the recall. However, the recall can be improved by further SCARE iterations over the data as we will see in the next experiment.

Quality vs. the number of SCARE iterations: This experiment shows the effect on the quality if we repeatedly execute SCARE over the dataset I times, $I = \{1, \ldots, 5\}$. After each iteration, the repaired tuples are considered members of the clean subset of the database, which is used in training the ML models. We report the obtained precision and recall in Figure 7. For all the SCARE-based approaches, the recall substantially improves from about 35% to close to 70% as we increase the number of iterations, while, the precision slightly decreases from the 90's to the 80's %. This indicates that the overall quality improves as we run more SCARE iterations. The $KHSinKPG$ outperforms the other approaches in terms of both precision and recall.

In each iteration, SCARE tries to repair the data to maximize the data likelihood given the learnt classification models subject to a constant amount of changes, δ. In the first few iterations, the discovered updates have higher correctness measure (*i.e.*, ratio of the likelihood benefit to the cost of the update, $\frac{l(u)}{c(u)}$) than those that are discovered later. Therefore, the applied updates in the first iterations have higher confidence, and hence, the precision starts high and decreases in later iterations. The recall increases faster than the decrease in precision, and therefore, the overall quality is improving. The main reason is that most of the applied updates are correct in the first few iterations, so the obtained database instances after each iteration are of higher quality to be learnt and modeled for predictions in the later iterations of SCARE. A stopping criteria for the iterations can be computed from the obtained overall likelihood benefit of the updates. If the benefit is not significant, then it is better to stop SCARE. In another setting, the user can be involved interactively (*e.g.*, [22]) to inspect very small number of the least beneficial updates after each iteration. If the least beneficial updates are correct, then most of the updates are correct according to the data distribution.

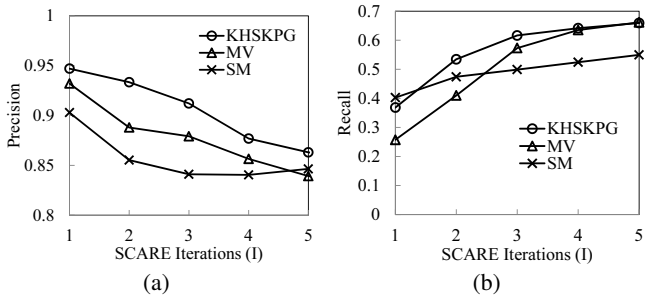
Quality vs. the partition functions: Here, we study the use of different partition functions (blocking techniques), as well as, the effect of the number of partition functions $|H|$ on the quality of SCARE on all the datasets. In Figure 8(a), we report the repair precision for Dataset 1 when we use four different blocking techniques as we increase the number of partition functions. One of the blocking techniques follows the clustering approach as described earlier in this section. The remaining three blocking techniques are signature based blocking, where we compute local sensitive hashing (LSH) for different attributes (`Zip`, `City`, `StreetAddress`) with different random seeds.

For each dataset, we note that as we increase the number of functions, the performances of all the blocking techniques improve. They all converge to obtain similar accuracy. Increasing $|H|$ will increase the chance that a tuple belongs to a larger number of partitions. SCARE learns a model from a local view (*i.e.*, from one partition) and predicts the values for the attributes of the considered tuple. Consequently, a larger number of candidate tuple repairs is proposed when increasing the number of partitions and the variance of the predictions decreases. The predictions become more independent from the blocking technique being used. The repair selection strategy combines the predictions of the local view models, and this increases the chance to obtain more accurate predictions.

5.4 SCARE Scalability

The scalability is one of the main advantages provided by SCARE, this is in addition to the quality of the updates demonstrated in the previous experiments. In the following, we assess the scalability of SCARE to large datasets.

In Figure 8(b), we report the scalability of SCARE on Dataset 2.



(a) (b)

Figure 7: Using SCARE in an iterative way helps improving the recall and the overall quality of the updates. The decrease in the precision is small compared to the increase in the recall, achieving an overall high quality improvement demonstrated by the f-measure.

We report as well the fraction of time for each of the two phases of SCARE. The reported time includes the time for learning the classification models. SCARE scales linearly, because of its systematic process of handling each data partition. Note also that SCARE finished the processing of a 1 M tuples in less than 6 minutes. Phase 1, the updates generation, takes 80-85 % of the time because of the process of learning models and obtaining predictions.

In Figure 8(c), we report the overall time taken by SCARE to handle datasets from Dataset 1 with different size from 5,000 to 50,000 tuples. We use two different partition techniques: Random and Blocking. In general, SCARE still scales linearly with the dataset size. Moreover, it is noted that partitioning the data by blocking makes SCARE more efficient. The reason is that blocking makes a data partition containing less diversity of the domain values. This results in a faster processing to train the classification models for prediction.

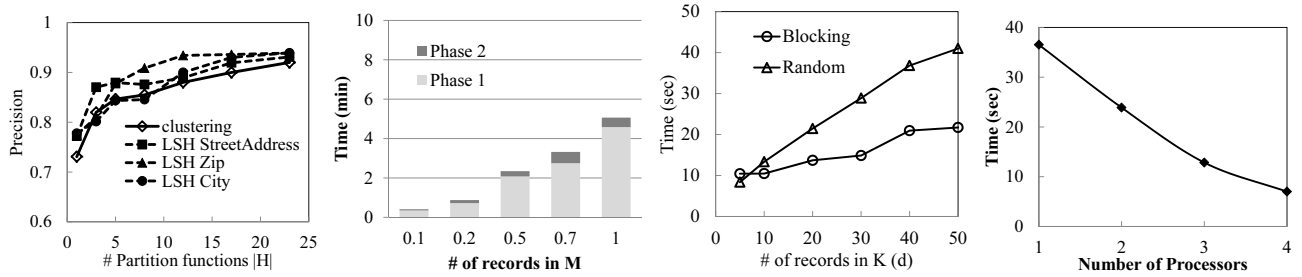
Usually, the process of statistical modeling and prediction tasks is quadratic, but this is not the case with SCARE because of the robust mechanism of partitioning the data and the strategy used to combine multiple predictions.

Finally, we mentioned that SCARE can be efficiently parallelized. In our implementation we used Java Threads and in Figure 8(d), we demonstrate the speedup we obtain as we increase the number of processors. Note that SCARE achieves a linear speedup, which supports our claim.

6. RELATED WORK

Improving data quality has been the focus of a large body of research for decades. Our work is closely related to two research areas: *i)* constraint-based data repair; and *ii)* statistical machine learning techniques for data cleaning.

Constraint-based data repair: In this approach, usually we have a database instance that is inconsistent with a predefined set of constraints. The objective is to find another database instance that minimally differs from the original one and that is consistent with the constraints. For example in [3], conditional FDs (CFDs) are used as database constraints. The repairing technique is based on a cost-based greedy heuristic to decide upon a repair of errors. The authors use equivalent classes to group the attributes values that are equivalent in obtaining a final consistent database instance. The work in [18] uses FDs to map the repairing problem to hyper-graph optimization problem, where a heuristic vertex cover algorithm can help finding the minimal number of attributes values to modify in order to find a database consistent with the FDs. The main drawback of these approaches is that the data should be covered by a set of constraints that have been specified or validated by domain experts, which may be an expensive manual process and may not be affordable for all data domains. Moreover, the constraints usually fall short to correctly identify the right fixes [10].



(a) Blocking Tech. (b) Dataset 2 (c) Dataset 1 (d) Speedup

Figure 8: The effect of the blocking techniques and SCARE scalability experiments

The work in [11] presents an entropy based technique to provide a guarantee on the applied cleaning updates. For SCARE, since the data is partitioned and the predictions are aggregated, it is not clear how the entropy measures can be aggregated. Instead, we rely on the updates likelihood benefit-to-cost ratio as a measure for the updates quality. Our experiments demonstrate overall high precision in updating the database.

Statistical machine learning techniques for data cleaning: Data cleaning using ML techniques mainly focused on deduplication (refer to [6] for survey), data imputation (e.g., [20]) and errors detecting (e.g., [23, 17]). For instance, in [20], data imputation is based on relational learning to learn the characteristics of the attributes relationships in a relational database. Then, the learnt model is used to infer the missing values. This technique requires *a priori* knowledge about the relationships between the attributes to construct the appropriate Bayesian network. Most of similar techniques for data imputation are limited to numerical or categorical attributes. SCARE does not have such limitation. Our probabilistic setting is related to the efforts on predicting multiple class labels [4] (*i.e.*, multiple values for a single attribute). However, our task is to predict a single value from multiple attributes predictions.

The main challenges for ML techniques are mainly *(i)* the scalability for large databases to be modeled considering all existing significant data correlations; and *(ii)* the accuracy of the replacement values prediction due to the fact that existing methods usually capture either local or global data relationships and do not combine both views whereas SCARE approach does. Nevertheless, the usefulness of ML techniques can be leveraged inside the learning step of SCARE and help further improving the accuracy of the predicted updates to the data. Note that our approach is not limited to a specific (probabilistic) classifier. Moreover, it can also take advantage of DB constraints and FDs when they are available.

7. CONCLUSION

In this paper, we propose SCARE, a robust and scalable approach for accurately repairing erroneous data values. Our solution offers several advantages over previous methods for data repairing using database constraints: the accuracy of the replacement values and the scalability of the method are guaranteed; the cost of the repair is bounded by the amount of changes that the user is willing to tolerate; no constraint or editing rule is needed since SCARE analyzes the data, learns the correlations from the correct data and take advantages of them for predicting accurate replacement values. Finally, as shown in our experiments, SCARE outperforms existing methods on large databases with no limitation on the type of data (*i.e.*, string, numeric, ordinal, categorical) or on the type of errors (*i.e.*, missing, incomplete, outlying, or inconsistent values). Our current work is to extend SCARE to enable duplicate elimination while repairing errors both at the value and tuple levels.

Future work is to efficiently and interactively involve the user in a visual repairing scenario so that the selection of the candidate replacement values as well as the entire database repairing process can be user-approved in real-time.

8. ACKNOWLEDGMENT

This research is based in part upon work supported by QCRI and by NSF under Grant Number IIS 0916614.

9. REFERENCES

[1] S. Arora, D. Karger, and M. Karpinski. Polynomial time approximation schemes for dense instances of np-hard problems. In *STOC*, 1995.
[2] Y. Asahiro, K. Iwama, H. Tamaki, and T. Tokuyama. Greedily finding a dense subgraph. In *Journal of Algorithms*, 2000.
[3] G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. Improving data quality: consistency and accuracy. In *VLDB*, 2007.
[4] K. Dembczynski, W. Cheng, and E. Hullermeier. Bayes optimal multilabel classification via probabilistic classifier chains. In *ICML*, 2010.
[5] T. G. Dietterich. Ensemble methods in machine learning. In *MCS workshop*, 2000.
[6] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *TKDE*, 2007.
[7] L. P. English. *Information quality applied: best practices for improving business information, processes and systems*. Wiley, 2009.
[8] W. Fan. Dependencies revisited for improving data quality. In *PODS*, 2008.
[9] W. Fan, F. Geerts, L. V. Lakshmanan, and M. Xiong. Discovering conditional functional dependencies. In *ICDE*, 2009.
[10] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Towards certain fixes with editing rules and master data. *PVLDB*, 2010.
[11] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Interaction between record matching and data repairing. In *SIGMOD*, 2011.
[12] U. Feige and M. Seltser. On the densest k-subgraph problem. *Algorithmica*, 1997.
[13] S. German and D. German. Neurocomputing: foundations of research. chapter Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. 1988.
[14] D. Heckerman, D. M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie. Dependency networks for inference, collaborative filtering, and data visualization. *J. Mach. Learn. Res.*, 2001.
[15] D. S. Hochbaum. Efficient bounds for the stable set, vertex cover and set packing problems. In *Discrete Applied Mathematics*, 1983.
[16] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Comput.*
[17] J. L. Y. Koh, M. L. Lee, W. Hsu, and K. T. Lam. Correlation-based detection of attribute outliers. In *DASFAA*, 2007.
[18] S. Kolahi and L. V. S. Lakshmanan. On approximating optimum repairs for functional dependency violations. In *ICDT*, 2009.
[19] A. Lopatenko and L. Bravo. Efficient approximation algorithms for repairing inconsistent databases. In *ICDE*, 2007.
[20] C. Mayfield, J. Neville, and S. Prabhakar. Eracer: a database approach for statistical inference and data cleaning. In *SIGMOD*, 2010.
[21] S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf. Large scale multiple kernel learning. *J. Mach. Learn. Res.*, 2006.
[22] M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, and I. F. Ilyas. Guided data repair. *PVLDB*, 2011.
[23] X. Zhu and X. Wu. Class noise vs. attribute noise: a quantitative study of their impacts. *Artif. Intell. Rev.*, 22:177–210, November 2004.